

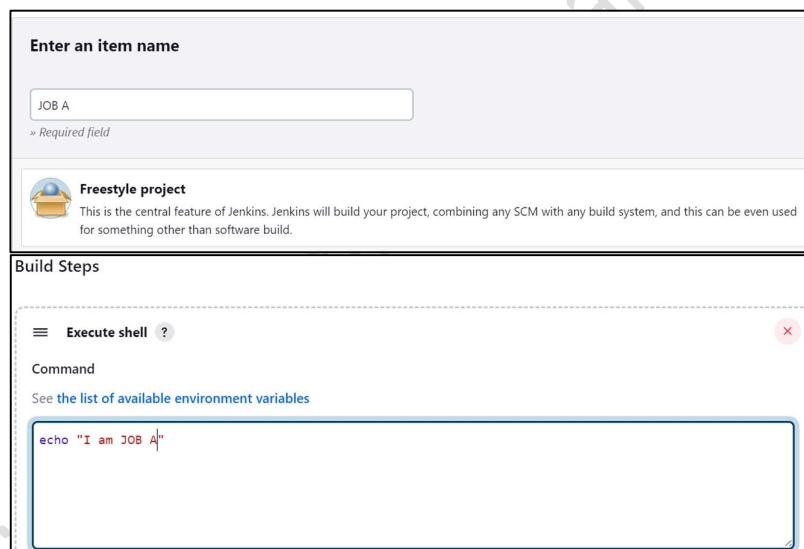
## What is a Pipeline?

Well, think of it as a task assembly line. Similar to how a factory operates, where various workers contribute their skills to craft a finished product, a pipeline in different domains like software development, manufacturing, or data processing involves a series of steps or stages. These stages are carried out in sequence to accomplish a specific objective. Each step in the pipeline focuses on a particular part of the process, and the outcome of one step becomes the input for the next step until the desired outcome is achieved. This systematic approach enhances efficiency and organization, much like how an assembly line streamlines production.

To illustrate, picture a company where different teams collaborate: the Developer Team, the Testing Team, and the Production Team. The Developer Team's task is to craft the code for constructing the application as per the requirements. The Testing Team takes on the responsibility of designing test cases that ensure the application is free of any glitches. Once testing is finished, the Production Team manages the deployment of the application in the production environment.

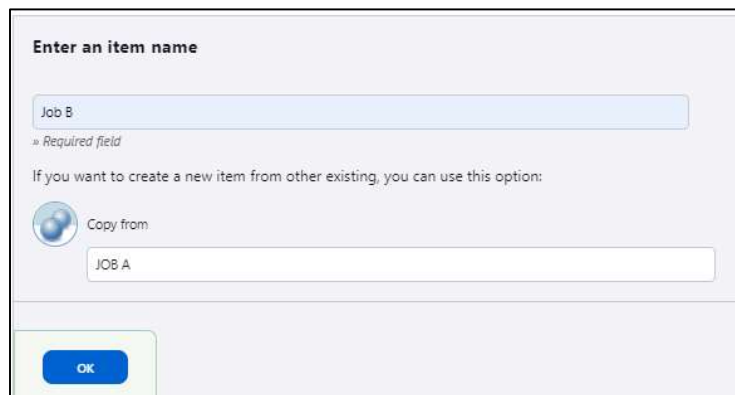
In this setup, all these teams interdepend on one another. The Developer Team shares its work with the Testing Team, which then hands over the tested code to the Production Team. This mutual reliance characterizes the situation. When things run smoothly, the final product is ready. By merging these collaborative efforts, we can visualize this process as a pipeline where each stage links to the next in succession.

Now, I am going to illustrate how we can create a pipeline using GUI.



The screenshot shows the Jenkins configuration interface for a Freestyle project. At the top, there is a text input field labeled 'Enter an item name' containing 'JOB A'. Below this is a section titled 'Freestyle project' with a description: '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.' The main section is 'Build Steps', which contains a single step named 'Execute shell'. The command field for this step contains the text 'echo "I am JOB A"'. There is a link 'See the list of available environment variables' and a red 'X' icon in the top right corner of the build steps section.

In this Job I just print the message I am JOB A. Like this I am going to create 2 More Jobs.

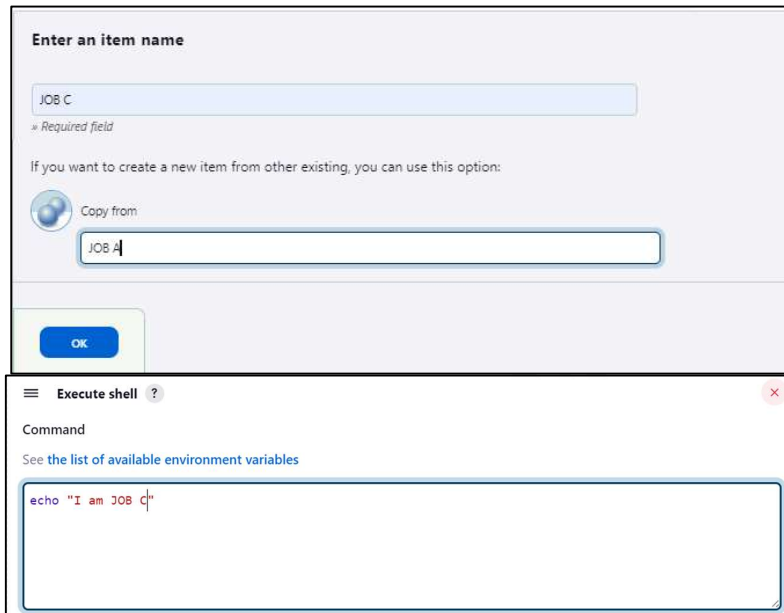


The screenshot shows the 'Copy from' dialog box in Jenkins. It has a text input field labeled 'Enter an item name' containing 'Job B'. Below this is a section titled 'If you want to create a new item from other existing, you can use this option:'. There is a 'Copy from' button with a globe icon, and a text input field containing 'JOB A'. At the bottom, there is a blue 'OK' button.

Create New Job and I am going to copy the existing JOB A, For this I have selected this Option to create a Job using existing Job and then entered JOB A in the Input box.

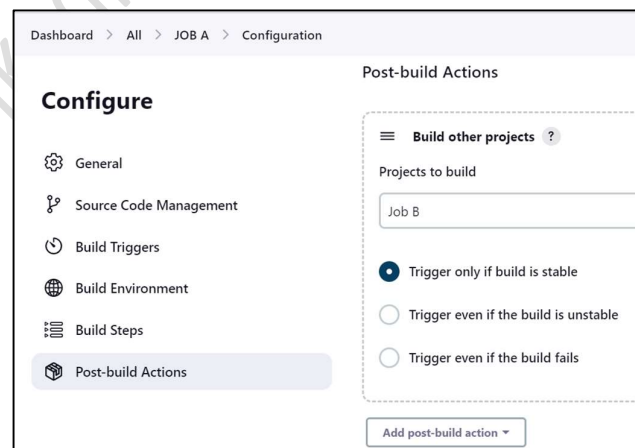


And then print one message I am JOB B. Now I am going to create JOB C using same way.



Now my requirement is, as soon as JOB A runs then after its execution JOB B start and then JOB C.

For this, Go into JOB A → Go into the Configuration → Go into Build Triggers section → Select Build after other Projects are built.



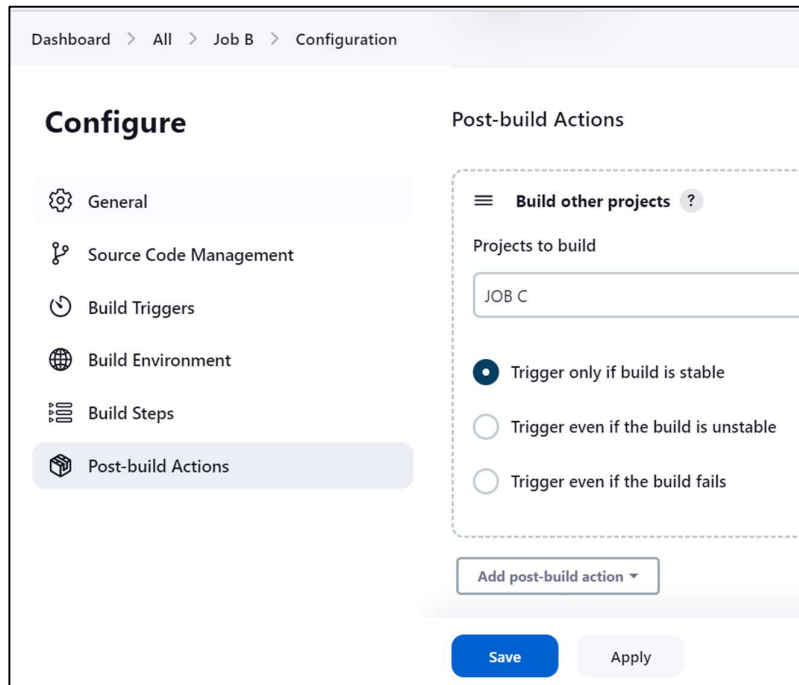
Here we have 3 Option:

- Trigger only if build is stable: It means that if the Job A Is executed successfully then run Job B
- Trigger even if the build is unstable: It means that regardless of whether the ongoing build process encounters errors, failures, or instability, a specific event or action will still be initiated.

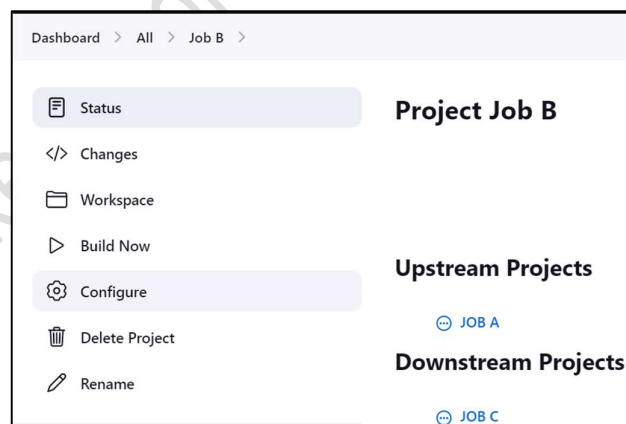
- Trigger even if the build fails: it means that a specific action will be performed even if the build is failed. This action could involve sending notifications to team members, generating reports, initiating debugging procedures, or performing other tasks that help in diagnosing and resolving issues that caused the build to fail.

Here I have selected Trigger only if build is stable.

Same thing I have done in JOB B. I would like to run the JOB C after the execution of JOB B.



And when I come in the Main page of JOB B and then we can see that JOB A is the Upstream and JOB B is the Downstream.



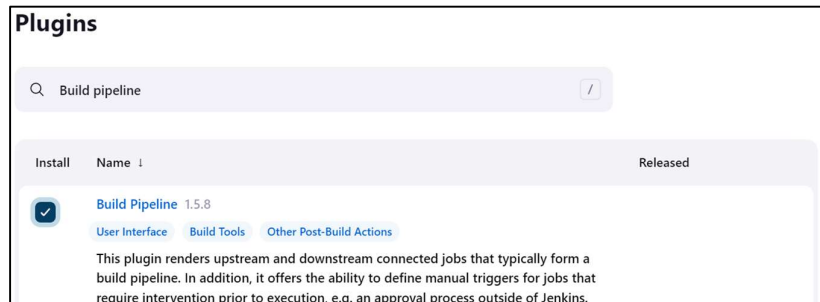
Now I am going to run the JOB A, and then it will automatically start other JOB B and after that JOB C.

...	☀	JOB A	N/A	N/A	N/A	▶
...	☀	Job B	N/A	N/A	N/A	▶
...	☀	JOB C	N/A	N/A	N/A	▶

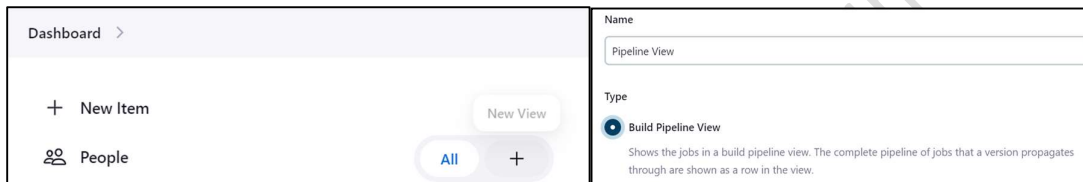
Here we can see the status as all the Jobs are executed successfully.

✓	☀	JOB A	1 min 1 sec	#1	N/A	0.1 sec	▶
✓	☀	Job B	51 sec	#1	N/A	19 ms	▶
✓	☀	JOB C	41 sec	#1	N/A	23 ms	▶

For having better visualisation, I am going to install one plugin **Build Pipeline**.

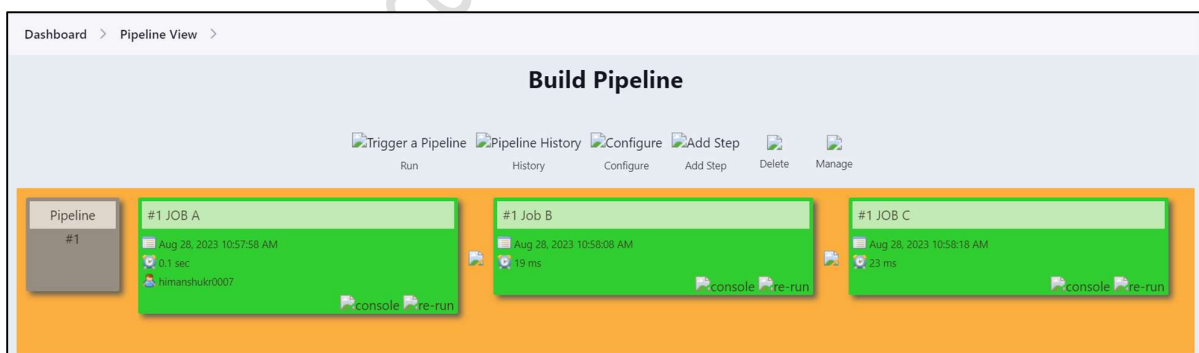


Install this plugin and after that restart the Jenkins and then login into Jenkins using your credentials.



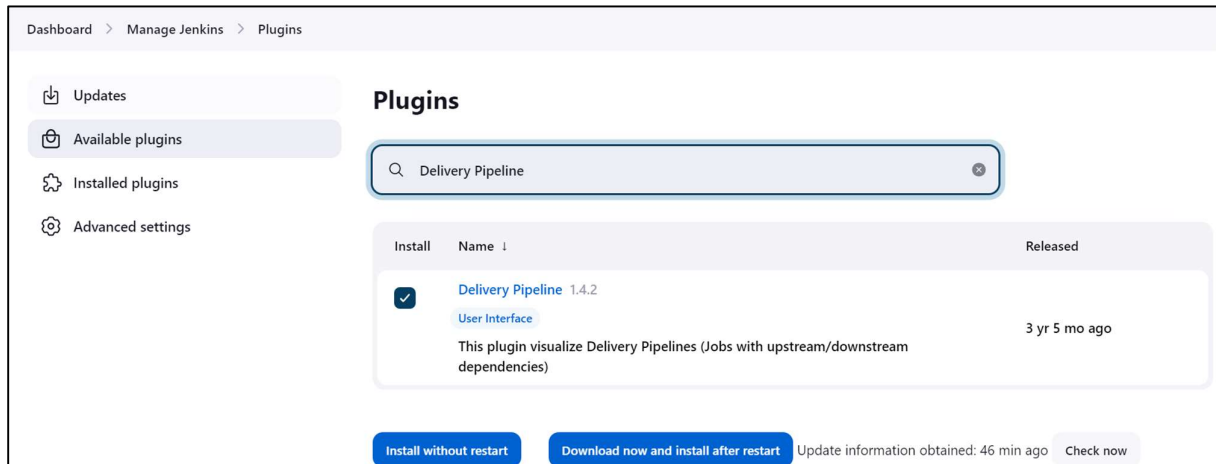
Click on + Sign. Then enter the View Name and select Build Pipeline View then Click on Create Button.  
From Pipeline Flow → From Layout Section → Select Initial Job as **JOB A**. then click on ok.

Now From Dashboard Click on **Pipeline View**.



Here I have only set the Initial JOB as JOB A and If I try to run JOB A then other JOB will start executing one after another.

We have another Plugin available as **Delivery Pipeline** download this Plugin and restart the Jenkins.

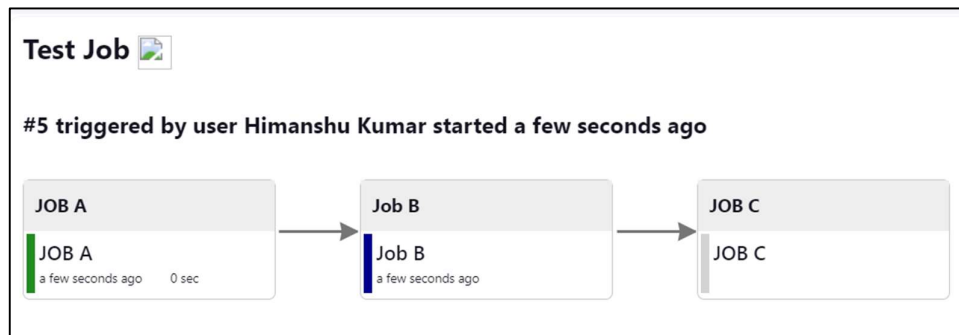


From Jenkins Dashboard Click ON + Sign to add the New View.

This time I have selected Delivery Pipeline View and entered the same name as Delivery Pipeline View.

Now From the Configuration Page, select those Check Box that You want to view on the Page. Then From the Components click on Add Component and Enter the Component Name and Select the Initial JOB.

Then Click on OK to save the VIEW and click on Build NOW button to run the JOB.

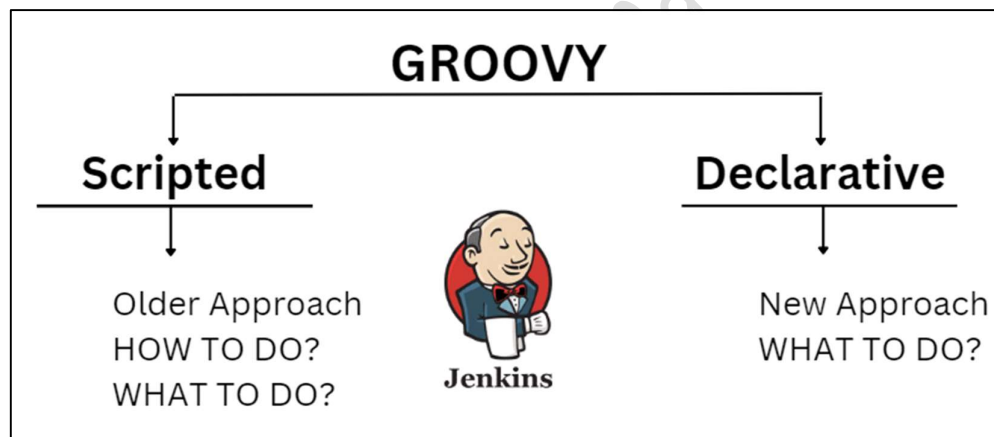


Here As soon as I clicked on Build Now button then All the JOB are started and from here, we can see that As soon as JOB A complete then JOB B starts and when JOB B Complete then JOB C start executing.

In real world pipeline, we don't create Pipeline in this way. Here we need to manually run the JOB this is not the fully automated.

Now Let's try to understand how we can do this thing with the help of Pipeline as CODE.

Now, let's delve into comprehending how we can accomplish this using Pipeline as Code. While creating jobs manually is one approach, our aim is to attain automation. To achieve this, we're employing a pipeline as code methodology. In this context, we're utilizing the GROOVY language to script our pipeline.



While Creating Pipeline I am following New Approach where I am going to write my requirement and Jenkins will do all the things for me.

For performing this setup, I am going to create one new Repository in my GitHub account.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

Owner \*  / Repository name \*

✓ Jenkins-Pipeline-Demo is available.

From my Local OS I am going to create one file with the Name of **Jenkinsfile** where I am going to write the code for my Pipeline. We can give any other file name also, but Jenkinsfile is a standard name in Market.

```

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins
$ git init jenkins-Pipeline-Demo
Initialized empty Git repository in C:/Users/Himanshu Kumar/Desktop/Jenkins/jenkins-Pipeline-Demo/.git/

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins
$ cd jenkins-Pipeline-Demo/

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ ls

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ |

```

Initialized new repository where I am going to write the code for my Jenkins.

In Pipeline as a Code, Job is known as Stage and stage contains steps. As we gathered enough Knowledge about Content, now we can easily understand the code with the help of syntax.

```

pipeline {
    agent { ... } // Defines where the pipeline will run (e.g., on a specific node or agent)
    parameters {
        string(name: 'ENVIRONMENT', defaultValue: 'dev', description: 'Environment for deployment')
        choice(name: 'BUILD_TYPE', choices: ['debug', 'release'], description: 'Select build type')
        booleanParam(name: 'RUN_TESTS', defaultValue: true, description: 'Run tests after build')
    }
    stages {
        stage('Stage 1') {
            steps {
                // Define steps to be executed in this stage
            }
        }
        stage('Stage 2') {
            steps {
                // Define steps for another stage
            }
        }
        // More stages can be added
    }
    post {
        always {
            // Define actions that will always run
        }
        success {
            // Define actions that will run if the pipeline succeeds
        }
        failure {
            // Define actions that will run if the pipeline fails
        }
        // Additional post-build actions can be defined
    }
}

```

This is just a simplified overview of the Declarative Pipeline syntax. Jenkins also supports Scripted Pipelines, which provide more flexibility but require more scripting.

In the code, I've used "agent any," which implies that the job will commence execution on any available agent. Alternatively, if we have a specific agent's name, we can specify it here.

In my Code, I have written very simple steps for printing statements.

```
pipeline {  
  
    agent any  
  
    stages {  
  
        stage('DEVELOPER') {  
            steps {  
                echo 'INITIAL PHASE STARTS'  
                echo 'This is Development PHASE'  
            }  
        }  
  
        stage('TEST') {  
            steps {  
                echo 'This is TESTING PHASE'  
            }  
        }  
  
        stage('QA') {  
            steps {  
                echo 'This is QUALITY ASSESSMENT PHASE'  
            }  
        }  
  
        stage('PRODUCTION'){  
            steps{  
                echo 'This is PRODUCTION PHASE'  
                echo 'ALL PHASES COMPLETED'  
            }  
        }  
    }  
}
```

Now I am going to Push this code to my GitHub Account by following commands.

- **git add README.md**
- **git commit -m "first commit"**
- **git branch -M main**
- **git remote add origin <https://github.com/Himanshu-kr-007/Jenkins-Pipeline-Demo.git>**
- **git push -u origin main**



Now the Code is stored in my Git Hub Account.

```
MINGW64:/c:/Users/Himanshu Kumar/Desktop/Jenkins/jenkins-Pipeline-Demo
Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ notepad "Jenkinsfile"

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ ls
Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ git add .

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ git commit -m "Jenkinsfile Created and Content Added"
[master (root-commit) 4dcca57] Jenkinsfile Created and Content Added
1 file changed, 27 insertions(+)
create mode 100644 Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (master)
$ git branch -M main

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git remote add origin https://github.com/Himanshu-kr-007/Jenkins-Pipeline-Demo.git

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 435 bytes | 108.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Himanshu-kr-007/Jenkins-Pipeline-Demo.git
* [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Just Make sure that the File name does not contain any extensions it should be only Jenkinsfile. I have just changed the file name as Jenkinsfile and removed the file with .txt extension.

```
Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ ls
Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ notepad Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ ls
Jenkinsfile Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ rm Jenkinsfile
Jenkinsfile      Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ rm Jenkinsfile.txt

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ ls
Jenkinsfile
```

Same I am going to upload in My GitHub.

```
Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git add .

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git commit . -m "File Name Modified"
[main 229e1af] File Name Modified
1 file changed, 0 insertions(+), 0 deletions(-)
rename Jenkinsfile.txt => Jenkinsfile (100%)



Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (2/2), 250 bytes | 83.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Himanshu-kr-007/Jenkins-Pipeline-Demo.git
4dcca57..229e1af main -> main
branch 'main' set up to track 'origin/main'.
```

Now, I am going to create one Job where I want my Pipeline as code will run. For this we require 1 plugin **Pipeline**.

**Pipeline** 596.v8c21c963d92d

A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code with Jenkins](#) for more details.

[Report an issue with this plugin](#)






Install this plugin if not Installed.


Create a New Job using Pipeline and pipeline is also known as workflows.

**Enter an item name**

» 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.


**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.

Click on OK Button. → From Pipeline → From Definition Select **Pipeline script from SCM**.

**Configure**

Pipeline

Definition

SCM ?

Repositories ?

Repository URL ?

Select Git From SCM → Enter the Repository URL → I have this Script in the **main** Branch with the name of Jenkinsfile.

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Script Path ?

Jenkinsfile

main 1 branch 0 tags

Go to file Add file <> Code

HimanshuKumar-Adrosonic File Name Modified 229e1af 12 minutes ago 2 commits

Jenkinsfile File Name Modified 12 minutes ago

Now from the General Setting → Enable the POLL SCM → and run this Job for every Minute then Save the Job.

Configure

Poll SCM ?

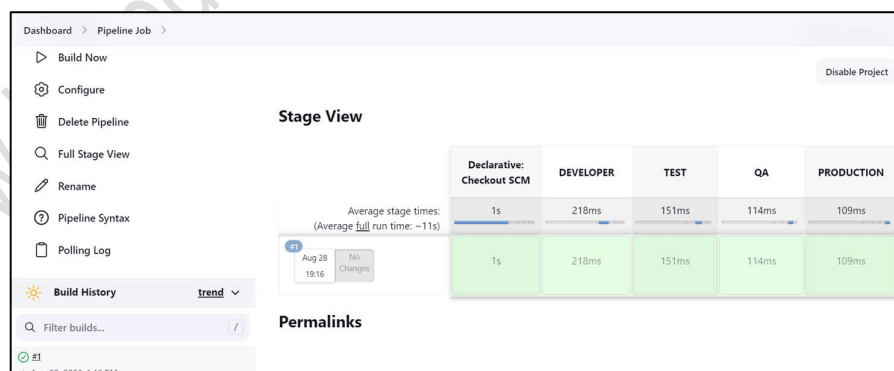
Schedule ?

\*\*\*\*\*

General

From Now On wards, you don't need to come to Jenkins also. If you made any changes in the Jenkinsfile and upload the code in the GitHub that code will apply.

Here In this Setup, The SCM will check if any code comes then it will go to GitHub and download the code. Then other Stages will start executing and if No error comes, then the Code will deploy in the Production.



In the Code, I have added one Stage with the Name of Pre-Prod and now I am going to deploy this code in the GitHub.

```

pipeline {
  agent any
  stages {
    stage('DEVELOPER') {
      steps {
        echo 'INITIAL PHASE STARTS'
        echo 'This is Development PHASE'
      }
    }
    stage('TEST') {
      steps {
        echo 'This is TESTING PHASE'
      }
    }
    stage('QA') {
      steps {
        echo 'This is QUALITY ASSESSMENT PHASE'
      }
    }
    stage('PRE-PRODUCTION'){
      steps{
        echo 'This is PRE-PRODUCTION PHASE'
      }
    }
    stage('PRODUCTION'){
      steps{
        echo 'This is PRODUCTION PHASE'
        echo 'ALL PHASES COMPLETED'
      }
    }
  }
}

```

Uploaded the Code in the GitHub

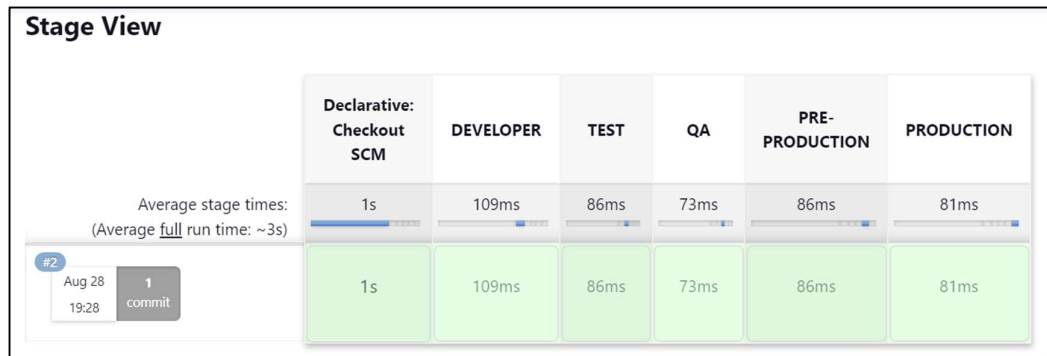
```

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git add .

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git commit -m "Pre-Production Added"
[main a7ac12c] Pre-Production Added
1 file changed, 5 insertions(+)

Himanshu Kumar@AICPL-L128 MINGW64 ~/Desktop/Jenkins/jenkins-Pipeline-Demo (main)
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes | 101.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Himanshu-kr-007/Jenkins-Pipeline-Demo.git
229e1af..a7ac12c main -> main
branch 'main' set up to track 'origin/main'.

```



Here you can see One Pre-Production Environment is also Added and this thing is done automatically.

Pipeline as Code is important because it automates and standardizes how software is developed. By using code to define steps like building, testing, and deploying, you get benefits like:

- **Consistency:** Code ensures each step happens the same way, preventing mistakes.
- **Version Control:** Storing pipeline code lets you track changes and go back to earlier versions if needed.
- **Reuse:** Code-based pipelines can be shared, saving time across projects.
- **Automation:** Code automates repetitive tasks, making development faster.
- **Clear Steps:** Each pipeline part is documented in code, helping audits and understanding.
- **Scaling:** Code-based pipelines handle complex work and different setups.
- **Living Docs:** Pipeline code shows exactly how apps are built and deployed.
- **Teamwork:** Teams work together on pipeline code, improving how things are done.
- **Reproducibility:** Code pipelines ensure consistent results, no matter where they run.
- **Quick Changes:** You can update the pipeline quickly and safely with code.

Pipeline as Code improves software development by automating, being consistent, and boosting teamwork.

**THANK YOU**