**What is a Jenkins pipeline?**

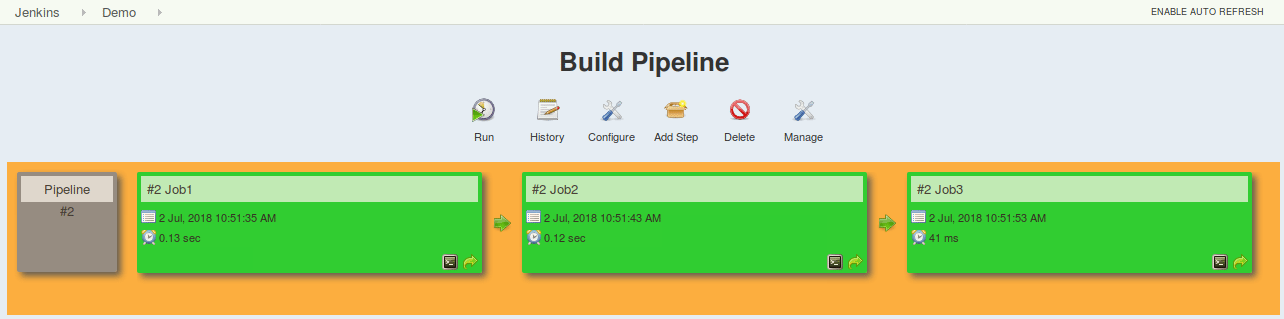
A pipeline is a collection of jobs that brings the software from version control into the hands of the end users by using automation tools. It is a feature used to **incorporate continuous delivery** in our software development workflow.

Over the years, there have been multiple Jenkins pipeline releases including, Jenkins Build flow, Jenkins Build Pipeline plugin, Jenkins Workflow, etc. What are the key features of these plugins?

* They represent multiple Jenkins jobs as one whole workflow in the form of a pipeline.
* What do these pipelines do? These pipelines are a **collection of Jenkins jobs** which trigger each other in a specified sequence.

Let me explain this with an example. Suppose I’m developing a small application on Jenkins and I want to build, test and deploy it. To do this, I will allot 3 jobs to perform each process. So, job1 would be for build, job2 would perform tests and job3 for deployment. I can use the Jenkins build pipeline plugin to perform this task. After creating three jobs and chaining them in a sequence, the build plugin will run these jobs as a pipeline.

This image shows a view of all the 3 jobs that run concurrently in the pipeline.



This approach is effective for deploying small applications. But what happens when there are complex pipelines with several processes (build, test, unit test, integration test, pre-deploy, deploy, monitor) running 100’s of jobs?

The maintenance cost for such a complex pipeline is huge and increases with the number of processes. It also becomes tedious to build and manage such a vast number of jobs. To overcome this issue, a new feature called **Jenkins Pipeline Project** was introduced.

The key feature of this pipeline is to define the entire deployment flow through code. What does this mean? It means that all the standard jobs defined by Jenkins are manually written as one whole script and they can be stored in a version control system. It basically follows the ‘**pipeline as code**’ discipline. Instead of building several jobs for each phase, you can now code the entire workflow and put it in a **Jenkinsfile**. Below is a list of reasons why you should use the Jenkins Pipeline.

**Jenkins Pipeline Advantages**

* It models simple to complex pipelines as code by using **Groovy DSL** (Domain Specific Language)
* The code is stored in a text file called the Jenkinsfile which can be **checked into a SCM** (Source Code Management)
* Improves user interface by incorporating **user****input** within the pipeline
* It is durable in terms of unplanned restart of the Jenkins master
* It can restart from saved **checkpoints**
* It supports complex pipelines by incorporating conditional loops, fork or join operations and allowing tasks to be performed in parallel
* It can integrate with several other plugins

**What is a Jenkinsfile?**

A Jenkinsfile is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system. How is this advantageous? This enables the developers to **access, edit and check the code at all times**.

The Jenkinsfile is written using the Groovy DSL and it can be created through a text/groovy editor or through the configuration page on the Jenkins instance. It is written based on two syntaxes, namely:

1. **Declarative pipeline syntax**
2. **Scripted pipeline syntax**

Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.

Whereas, the scripted pipeline is a traditional way of writing the code. In this pipeline, the Jenkinsfile is **written on the Jenkins UI instance**. Though both these pipelines are based on the groovy DSL, the scripted pipeline uses stricter groovy based syntaxes because it was the first pipeline to be built on the groovy foundation. Since this Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax.

The declarative pipeline is defined within a block labelled ‘pipeline’ whereas the scripted pipeline is defined within a ‘**node**’. This will be explained below with an example.

## ****Pipeline concepts****

### ****Pipeline****

This is a user defined block which contains all the processes such as build, test, deploy, etc. It is a collection of all the stages in a Jenkinsfile. All the stages and steps are defined within this block. It is the key block for a declarative pipeline syntax.



### ****Node****

A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.



There are various mandatory sections which are common to both the declarative and scripted pipelines, such as stages, agent and steps that must be defined within the pipeline.

These are explained below:

### ****Agent****

An agent is a directive that can run multiple builds with only one instance of Jenkins. This feature helps to distribute the workload to different agents and execute several projects within a single Jenkins instance. It instructs Jenkins to **allocate an executor** for the builds.

A single agent can be specified for an entire pipeline or specific agents can be allotted to execute each stage within a pipeline. Few of the parameters used with agents are:

##### **Any**

Runs the pipeline/ stage on any available agent.

##### **None**

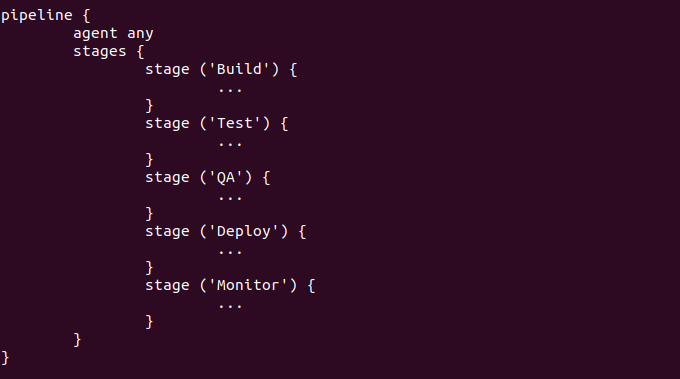
This parameter is applied at the root of the pipeline and it indicates that there is no global agent for the entire pipeline and each stage must specify its own agent.

##### **Label**

Executes the pipeline/stage on the labelled agent.

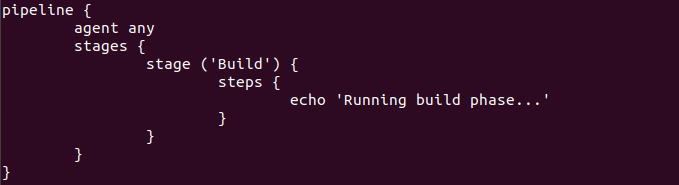
### ****Stages****

This block contains all the work that needs to be carried out. The work is specified in the form of stages. There can be more than one stage within this directive. Each stage performs a specific task. In the following example, I’ve created multiple stages, each performing a specific task.



### ****Steps****

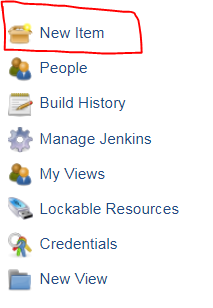
A series of steps can be defined within a stage block. These steps are carried out in sequence to execute a stage. There must be at least one step within a steps directive. In the following example I’ve implemented an echo command within the build stage. This command is executed as a part of the ‘Build’ stage.



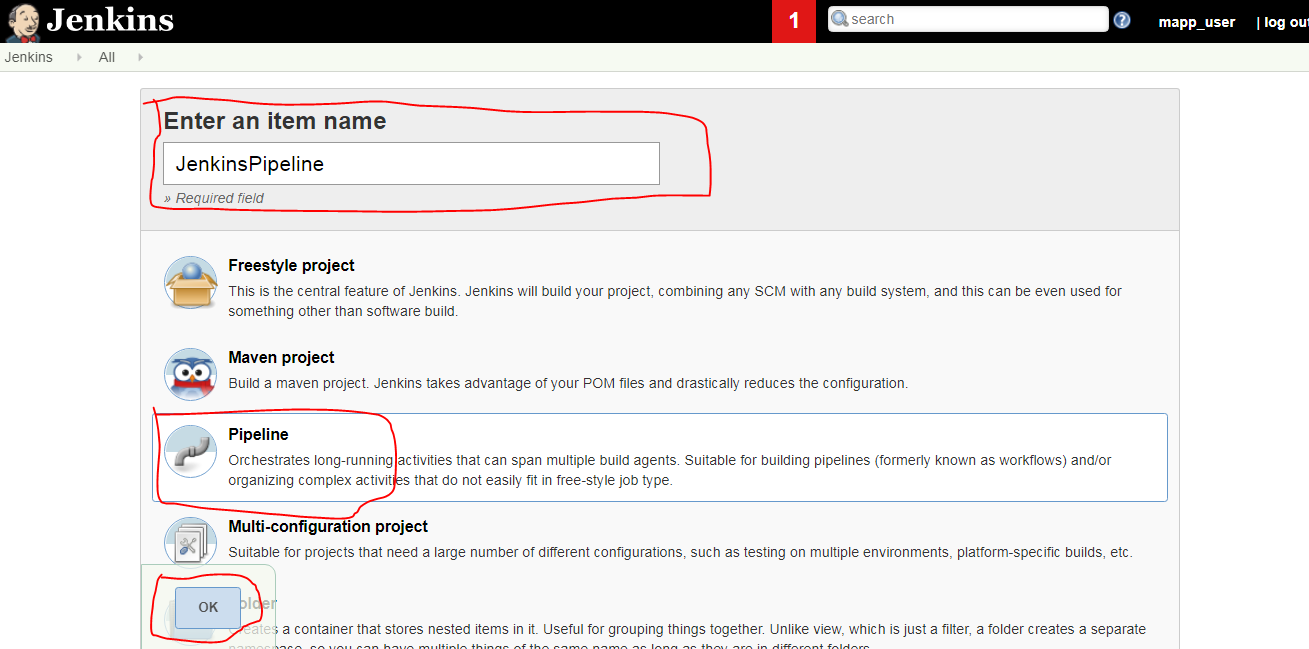
Now that you are familiar with the basic pipeline concepts let’s start of with the Jenkins pipeline tutorial. Firstly, let’s learn how to create a Jenkins pipeline.

### ****Creating your first Jenkins pipeline.****

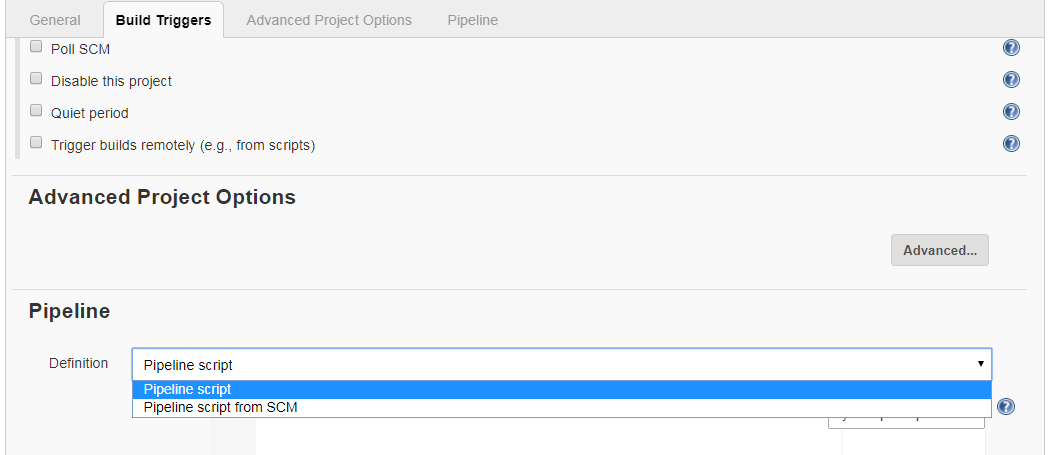
**Step 1**: Log into Jenkins and select ‘New item’ from the dashboard.



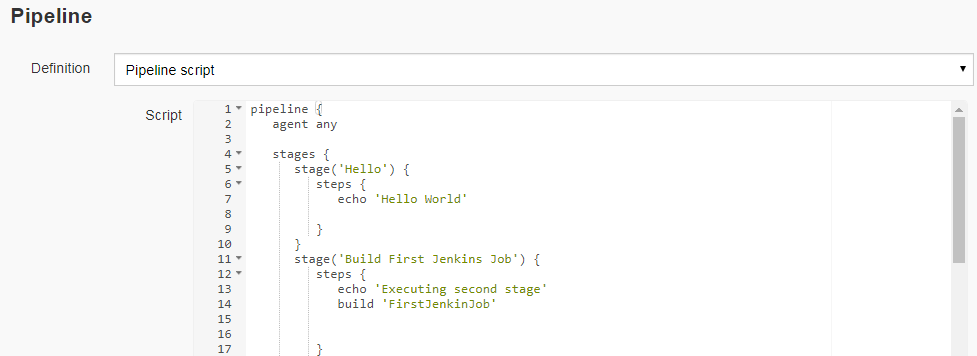
**Step 2**: Next, enter a name for your pipeline and select ‘pipeline’ project. Click on ‘ok’ to proceed.



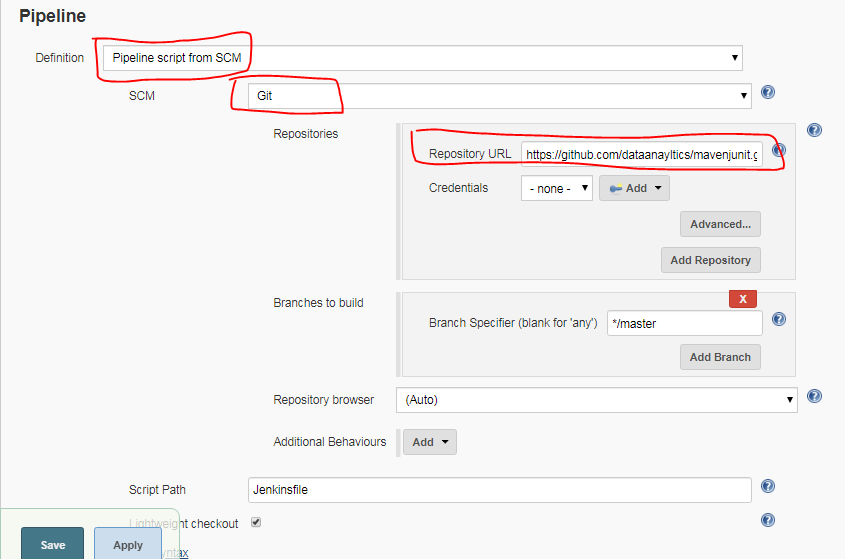
**Step 3**: Scroll down to the pipeline and choose if you want a declarative pipeline or a scripted one.



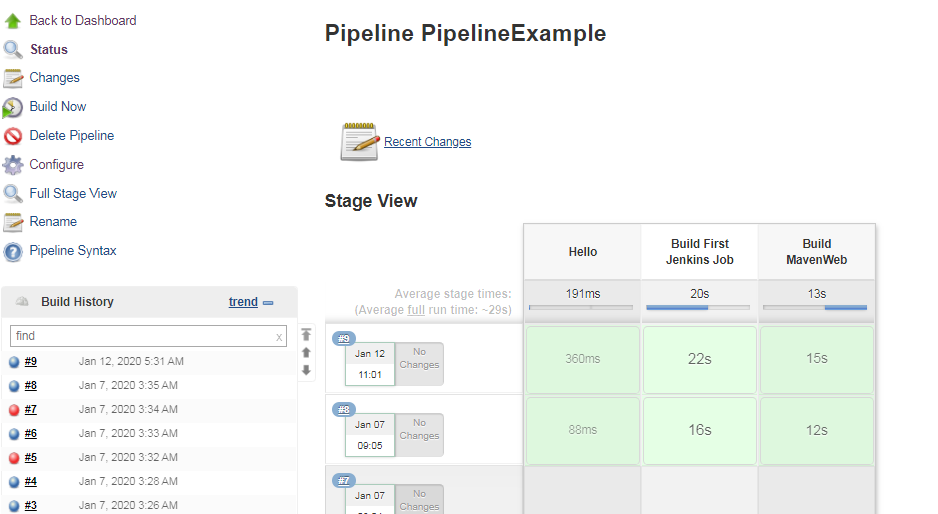
**Step 4a**: If you want a scripted pipeline then choose ‘pipeline script’ and start typing your code.



**Step 4b**: If you want a declarative pipeline then select ‘pipeline script from SCM’ and choose your SCM. In my case I’m going to use Git throughout this demo. Enter your repository URL.

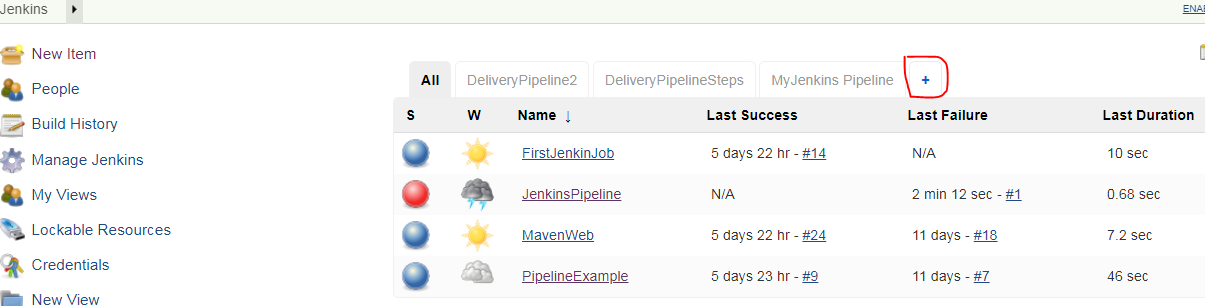


**Stage View**

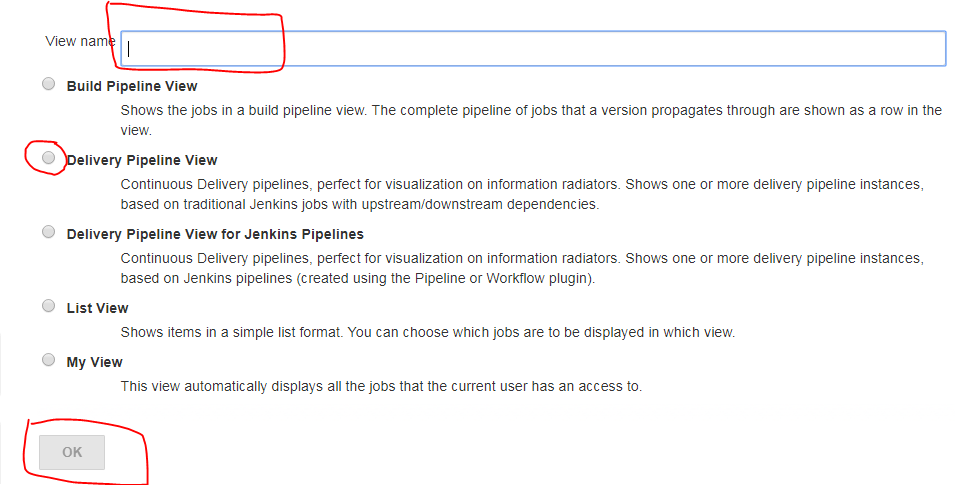


**Create Delivery Pipeline View**

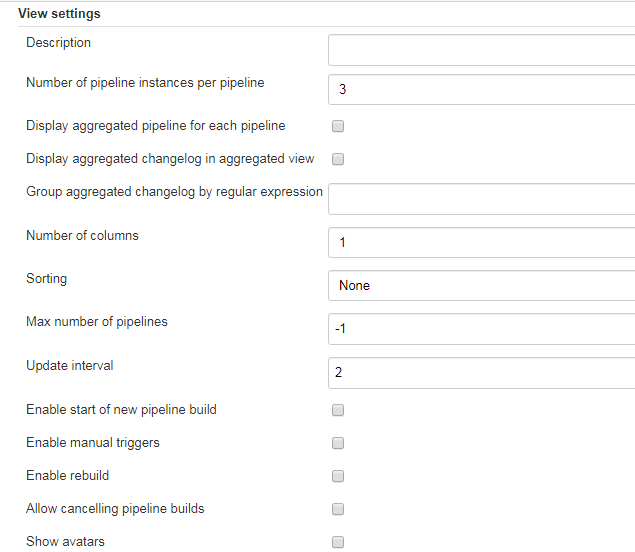
**1)** Click on + button

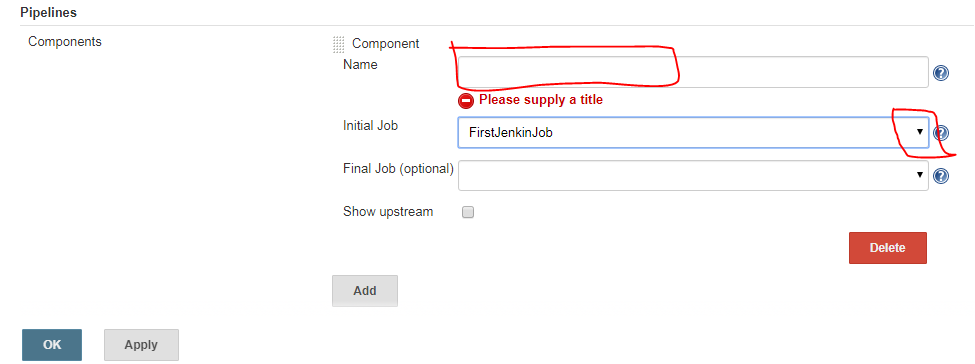


2) Provide View name and Select Delivery Pipeline

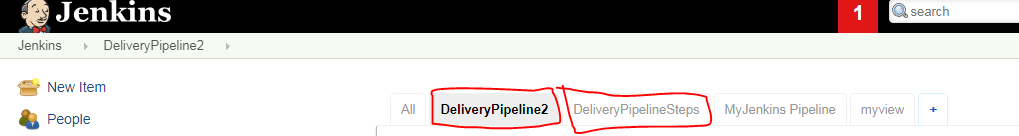


**3)** Keep Default View settings or you can change according to your requirement.





**5)** You will get newly created view as Tab



**6)** you can manually Trigger pipeline

