Why Pipeline?

Jenkins pipeline is a single platform that runs the entire pipeline as code.

Instead of building several jobs for each phase, you can now code the workflow and put it in a Jenkinsfiles.

While standard Jenkins “freestyle” jobs support simple continuous integration by allowing you to define sequential tasks in an application lifecycle, they do not create a persistent record of execution, enable one script to address all the steps in a complex workflow, or confer the other advantages of pipelines.

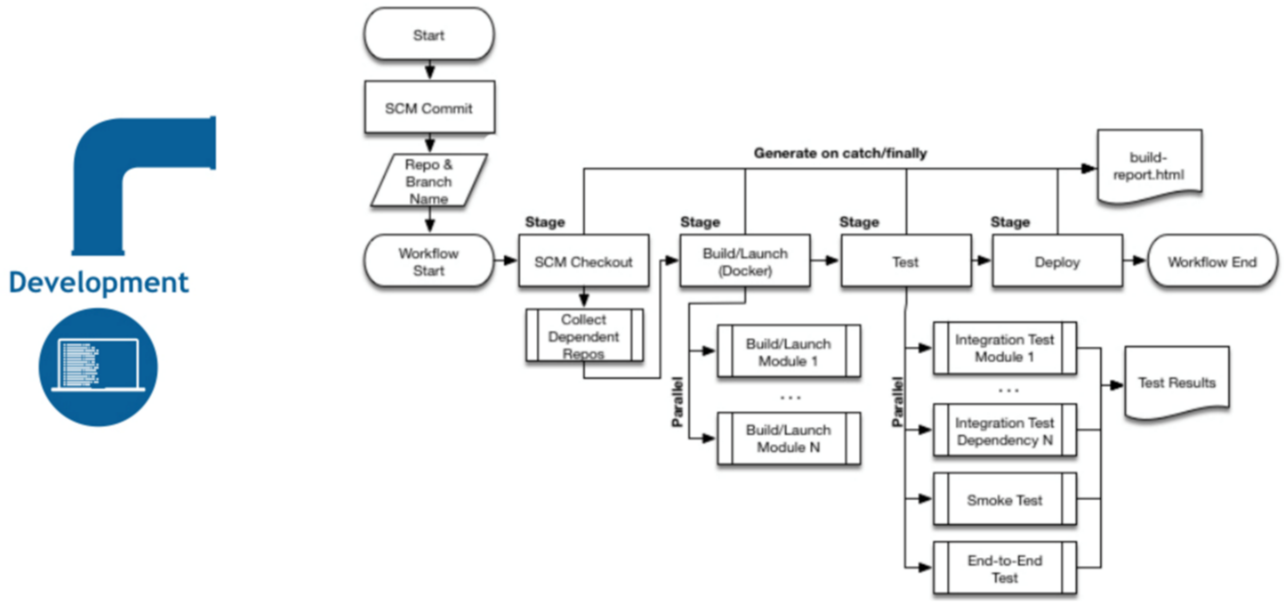
In contrast to freestyle jobs, pipelines enable you to define the whole application lifecycle. Pipeline functionality helps Jenkins to support continuous delivery (CD). The Pipeline plugin was built with requirements for a flexible, extensible, and script-based CD workflow capability in mind.

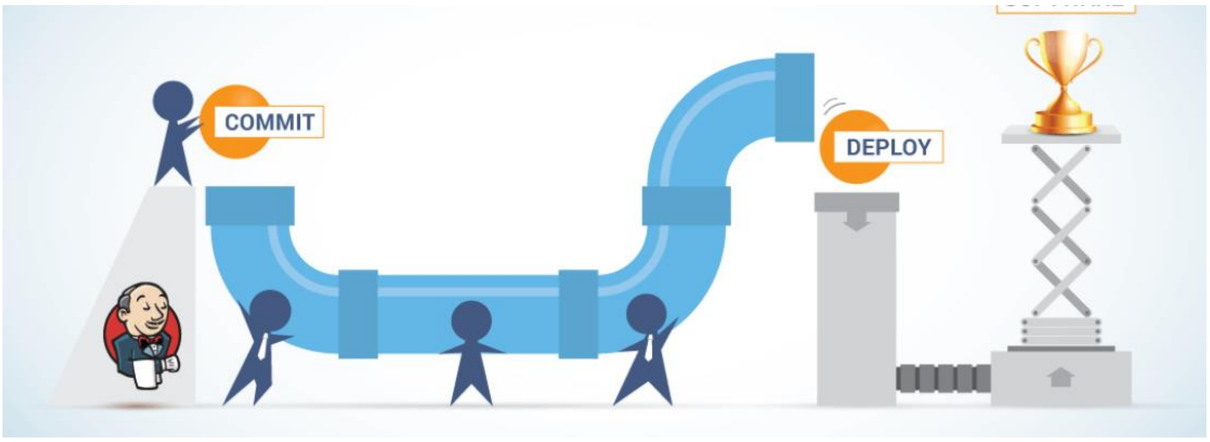
Accordingly, pipeline functionality is:

- Durable: Pipelines can survive both planned and unplanned restarts of your Jenkins master.
- Pausable: Pipelines can optionally stop and wait for human input or approval before completing the jobs for which they were built.
- Versatile: Pipelines support complex real-world CD requirements, including the ability to fork or join, loop, and work in parallel with each other.
- Efficient: Pipelines can restart from any of several saved checkpoints.
- Extensible: The Pipeline plugin supports custom extensions to its DSL (domain scripting language) and multiple options for integration with other plugins.

The flowchart below is an example of one continuous delivery scenario enabled by the Pipeline plugin:

Jenkins Workflow



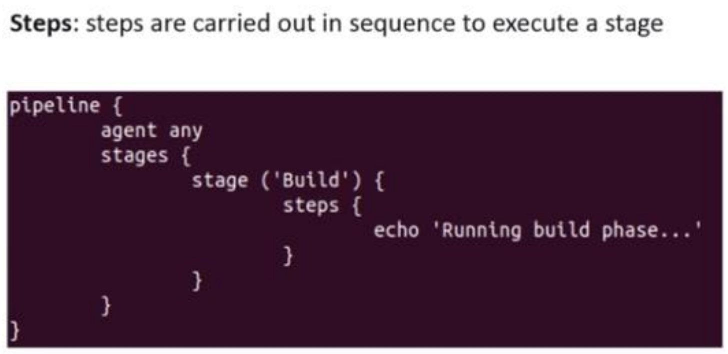


Pipeline Vocabulary

Pipeline terms such as “step,” “node,” and “stage” are a subset of the vocabulary used for Jenkins in general.

Step

A “step” (often called a “build step”) is a single task that is part of sequence. Steps tell Jenkins what to do.



Steps: steps are carried out in sequence to execute a stage

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        steps {
                                echo 'Running build phase...'
                        }
                }
        }
}
```

Node

In pipeline coding contexts, as opposed to Jenkins generally, a “node” is a step that does two things, typically by enlisting help from available executors on agents:

- Schedules the steps contained within it to run by adding them to the Jenkins build queue (so that as soon as an executor slot is free on a node, the appropriate steps run).
- Creates a workspace, meaning a file directory specific to a particular job, where resource-intensive processing can occur without negatively impacting your pipeline performance. Workspaces last for the duration of the tasks assigned to them.

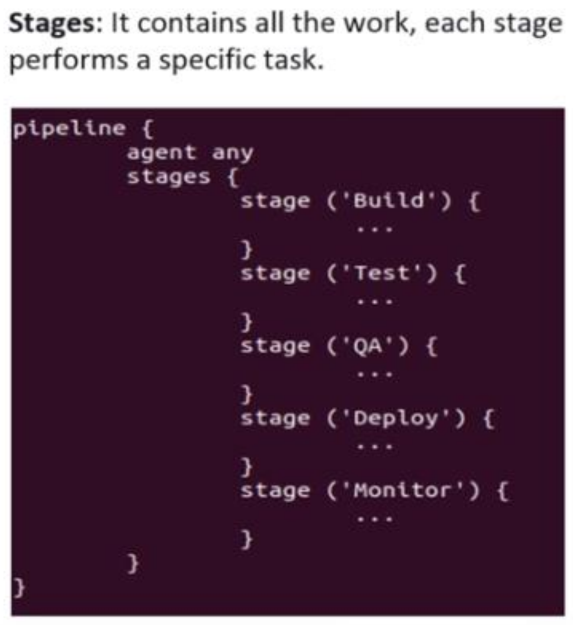
(In Jenkins generally, a “node” means any computer that is part of your Jenkins installation, whether that computer is used as a master or as an agent).

Stage

A “stage” is a step that calls supported APIs. Pipeline syntax is comprised of stages. Each stage can have one or more build steps within it.

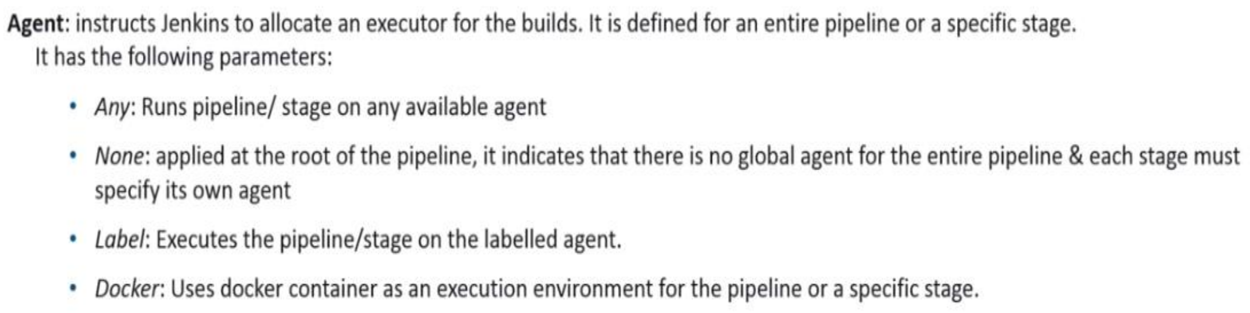
Familiarity with Jenkins terms such as “master,” “agent,” and “executor” also helps with understanding how pipelines work. These terms are not specific to pipelines:

- master - A “master” is the basic installation of Jenkins on a computer; it handles tasks for your build system. Pipeline scripts are parsed on masters, and steps wrapped in node blocks are performed on available executors.
- agent - An “agent” (formerly "slave") is a computer set up to offload particular projects from the master. Your configuration determines the number and scope of operations that an agent can perform. Operations are performed by executors.
- executor - An “executor” is a computational resource for compiling code. It can run on master or agent machines, either by itself or in parallel with other executors. Jenkins assigns a *java.lang.Thread* to each executor.



Stages: It contains all the work, each stage performs a specific task.

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        ...
                }
                stage ('Test') {
                        ...
                }
                stage ('QA') {
                        ...
                }
                stage ('Deploy') {
                        ...
                }
                stage ('Monitor') {
                        ...
                }
        }
}
```



Agent: instructs Jenkins to allocate an executor for the builds. It is defined for an entire pipeline or a specific stage.
It has the following parameters:

- *Any*: Runs pipeline/ stage on any available agent
- *None*: applied at the root of the pipeline, it indicates that there is no global agent for the entire pipeline & each stage must specify its own agent
- *Label*: Executes the pipeline/stage on the labelled agent.
- *Docker*: Uses docker container as an execution environment for the pipeline or a specific stage.

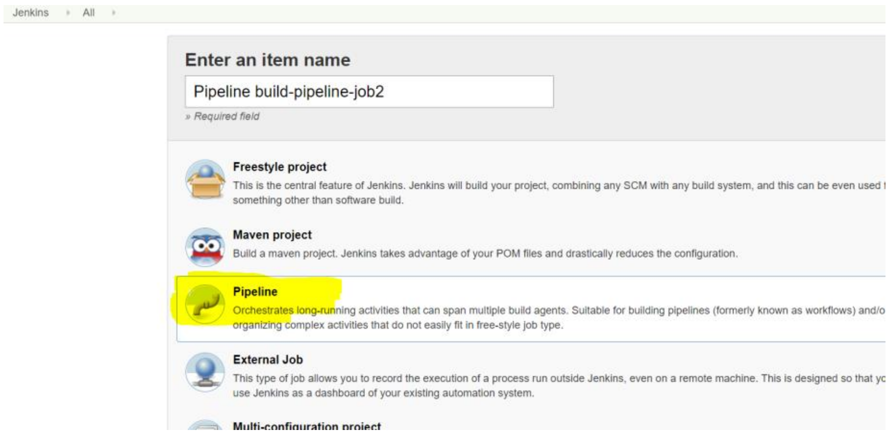
Lab 1:

To create a simple pipeline from the Jenkins interface, perform the following steps:

1. Click **New Item** on your Jenkins home page, enter a name for your (pipeline) job, select **Pipeline**, and click **OK**.
2. In the Script text area of the configuration screen, enter your pipeline syntax. If you are new to pipeline creation, you might want to start by opening Snippet Generator and selecting the “Hello Word” snippet. **Note:** Pipelines are written as Groovy scripts that tell Jenkins what to do when they are run, but because relevant bits of syntax are introduced as needed, you do not need deep expertise in Groovy to create them, although basic understanding of Groovy is helpful.

3. Check the Use Groovy Sandbox option below the Script text area. **Note:** If you are a Jenkins administrator (in other words, authorized to approve your own scripts), sandboxing is optional but efficient, because it lets scripts run without approval as long as they limit themselves to operations that Jenkins considers inherently safe.
4. Click **Save**.
5. Click **Build Now** to create the pipeline.
6. Click ▾ and select **Console Output** to see the output.

Lab 2:



Enter an item name

Pipeline build-pipeline-job2

» 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 t
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/o
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 yo
use Jenkins as a dashboard of your existing automation system.

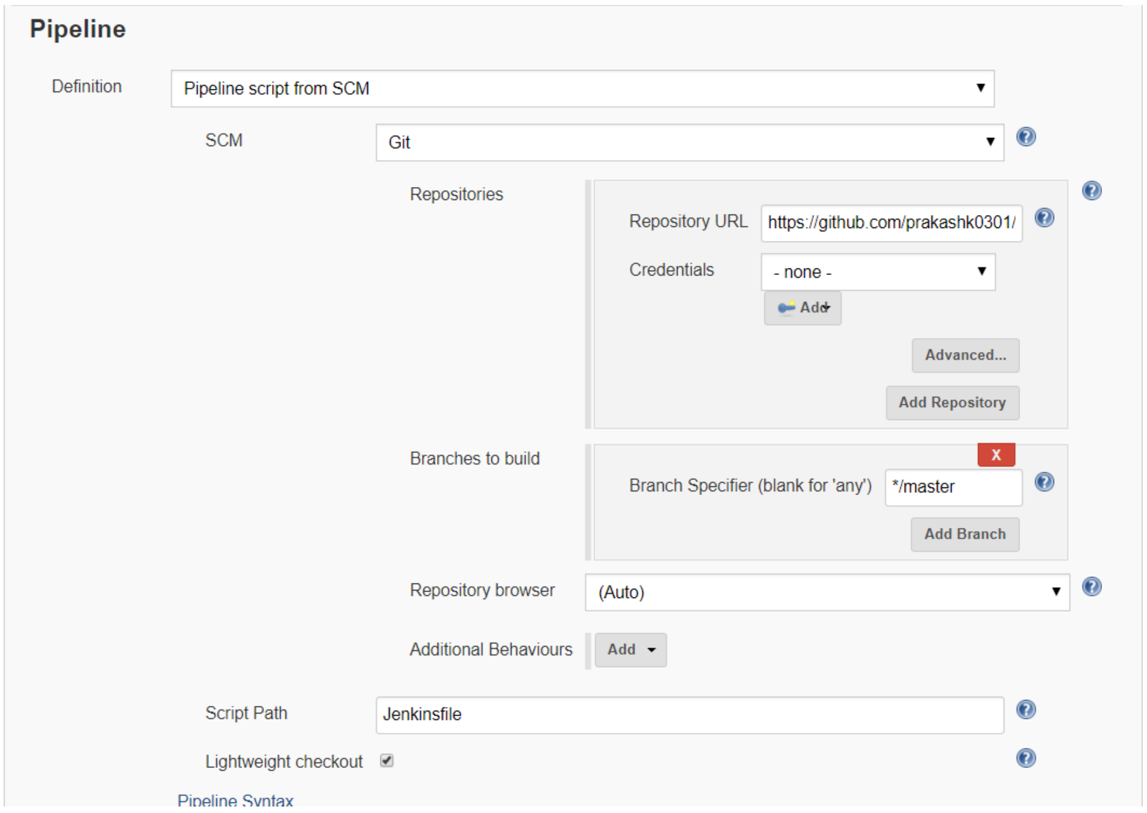
Multi-configuration project

Pipeline section

(https://github.com/prakashk0301/jenkins_pipeline_hello.git)

and

https://github.com/prakashk0301/jenkins_pipeline



Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/prakashk0301/

Credentials: - none -

Branches to build

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

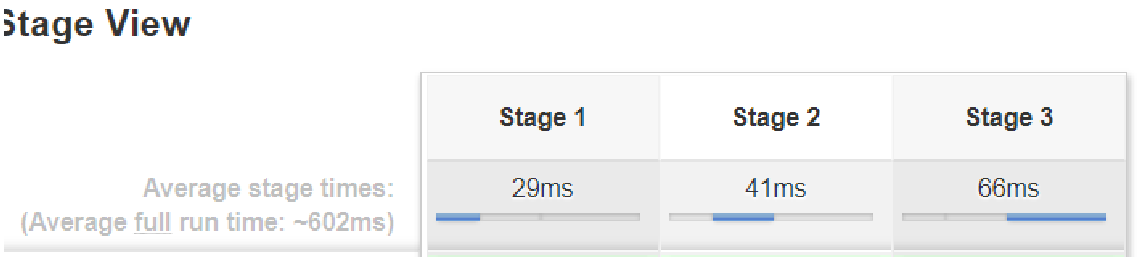
Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Lightweight checkout ✓

Console output:



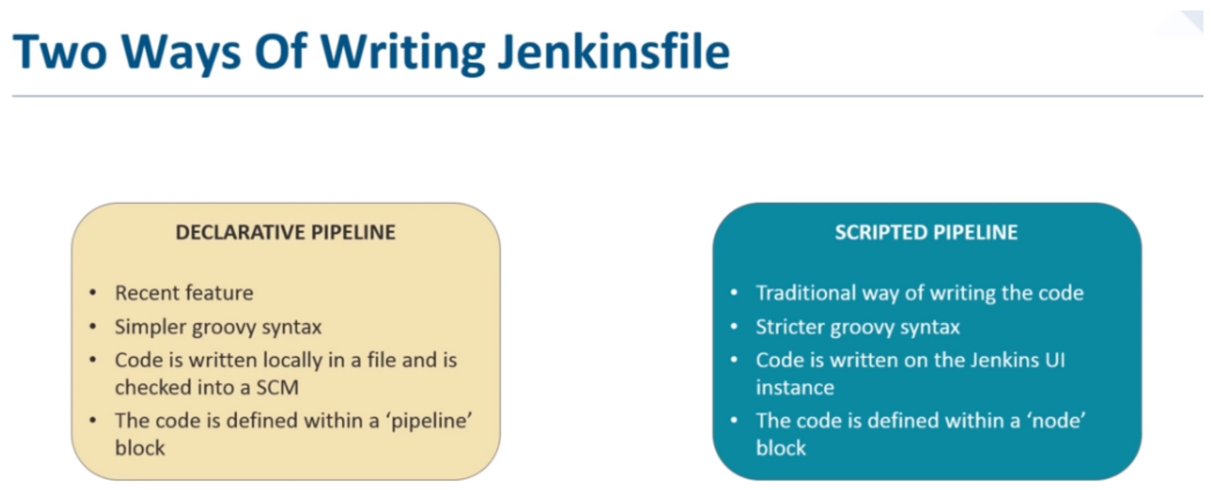
Stage View

	Stage 1	Stage 2	Stage 3
Average stage times: (Average full run time: ~602ms)	29ms	41ms	66ms

---

https://github.com/prakashk0301/jenkins_pipeline

(install pipeline maven integration plugin)



Two Ways Of Writing Jenkinsfile

DECLARATIVE PIPELINE

- Recent feature
- Simpler groovy syntax
- Code is written locally in a file and is checked into a SCM
- The code is defined within a 'pipeline' block

SCRIPTED PIPELINE

- Traditional way of writing the code
- Stricter groovy syntax
- Code is written on the Jenkins UI instance
- The code is defined within a 'node' block

Declarative Pipeline:

https://github.com/prakashk0301/git-test