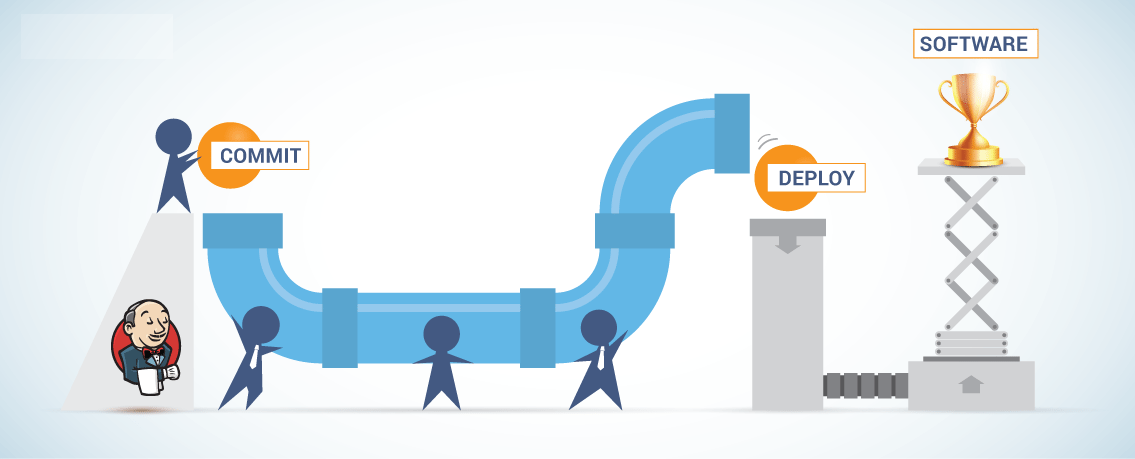
**Jenkins Pipeline**

* Jenkins has proven to be an expert in implementing continuous integration, continuous testing and continuous deployment.
* When it comes to **continuous delivery**, Jenkins uses a feature called **Jenkins pipeline**.
* In order to understand why Jenkins pipeline was introduced, we have to understand what continuous delivery is and why it is important.



**Continuous Delivery:**

* Continuous delivery is the capability to release a software at all times. It is a practice which ensures that the software is always in a **production-ready state.**
* It means that every time a change is made to the code or the infrastructure, the software team must work in such a way that these changes are built quickly and tested using various **automation tools** after which the build is subjected to production.
* Continuous delivery ensures that the software is built, tested and released more frequently. It reduces the cost, time and risk of the incremental software releases.

## ****What is a Jenkins pipeline?****

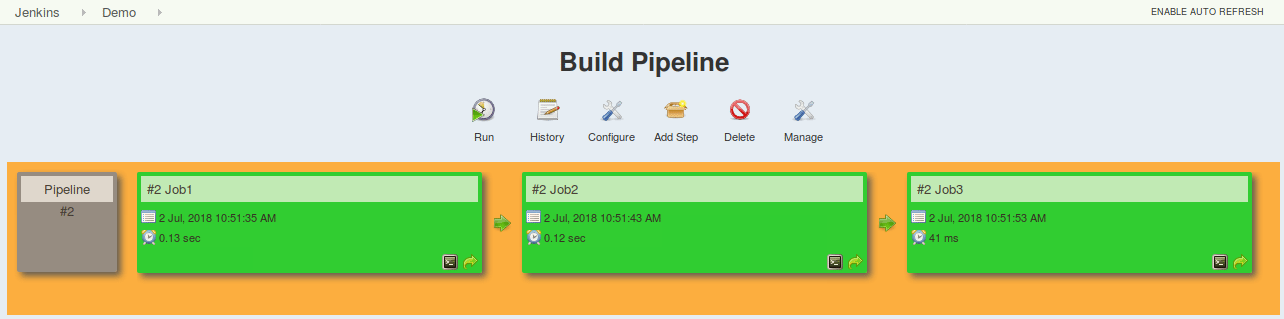
A pipeline is a collection of jobs that brings the software from version control into the hands of the end users by using automation tools. It is a feature used to **incorporate continuous delivery** in our software development workflow.

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.



Build Pipeline Plugin output – Jenkins Pipeline Tutorial

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:

### ****Declarative pipeline syntax****

### ****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 {

agent any

stages {

stage ('Checkout') {

echo "Do the SVN/GIT Checkout"

}

stage ('Build') {

echo "Build by using ANT/MAVEN"

}

stage ('Test') {

echo "Do the test by using Junit/Selenium"

}

stage {'QA'} {

echo "Do the Qa testing"

}

stage ('deploy') {

echo "Deploy the artifacts respectuve servers by using Ansible/chef/docker...etc"

}

stage ('Monitor') {

echo "Monitot the application by using sonarcube/Javamelody"

}

}

}

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

###---Declarative Pipeline---###

pipeline {

agent any

stages {

stage('One') {

steps {

echo 'Hi, this is amar from Joctopus'

}

}

stage('Two') {

steps {

input('Do you want to proceed?')

}

}

stage('Three') {

when {

not {

branch "master"

}

}

steps {

echo "Hello"

}

}

stage('Four') {

parallel {

stage('Unit Test') {

steps {

echo "Running the unit test..."

}

}

stage('Integration test') {

agent {

docker {

reuseNode true

image 'ubuntu'

}

}

steps {

echo "Running the integration test..."

}

}

}

}

}

}

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

###---Scripted Pipeline Demo---###

node {

for (i=0; i<2; i++) {

stage "Stage #"+i

print 'Hello, world !'

if (i==0)

{

git "https://github.com/amara401/amar\_demo"

echo 'Running on Stage #0'

}

else {

build 'Declarative pipeline'

echo 'Running on Stage #1'

}

}

}