Casestudy: Suppose, I have created Jenkins job in 1 server, and I want to create similar job in another server, how I can do that? Answer: AMI, terraform, or export/import plugin.

**Best answer:** write WORA Jenkin file.

**What is Jenkins Pipeline?**

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the [Pipeline domain-specific language (DSL) syntax](https://jenkins.io/doc/book/pipeline/syntax).

The definition of a Jenkins Pipeline is written into a text file (called a [Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile)) which in turn can be committed to a project’s source control repository. [[2](https://jenkins.io/doc/book/pipeline/#_footnotedef_2)] This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

* Automatically creates a Pipeline build process for all branches and pull requests.
* Code review/iteration on the Pipeline (along with the remaining source code).
* Audit trail for the Pipeline.
* Single source of truth [[3](https://jenkins.io/doc/book/pipeline/#_footnotedef_3)] for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

**Declarative versus Scripted Pipeline syntax**

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

* provides richer syntactical features over Scripted Pipeline syntax, and
* is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in [Pipeline concepts](https://jenkins.io/doc/book/pipeline/#pipeline-concepts) and [Pipeline syntax overview](https://jenkins.io/doc/book/pipeline/#pipeline-syntax-overview) below.

**Why Pipeline?**

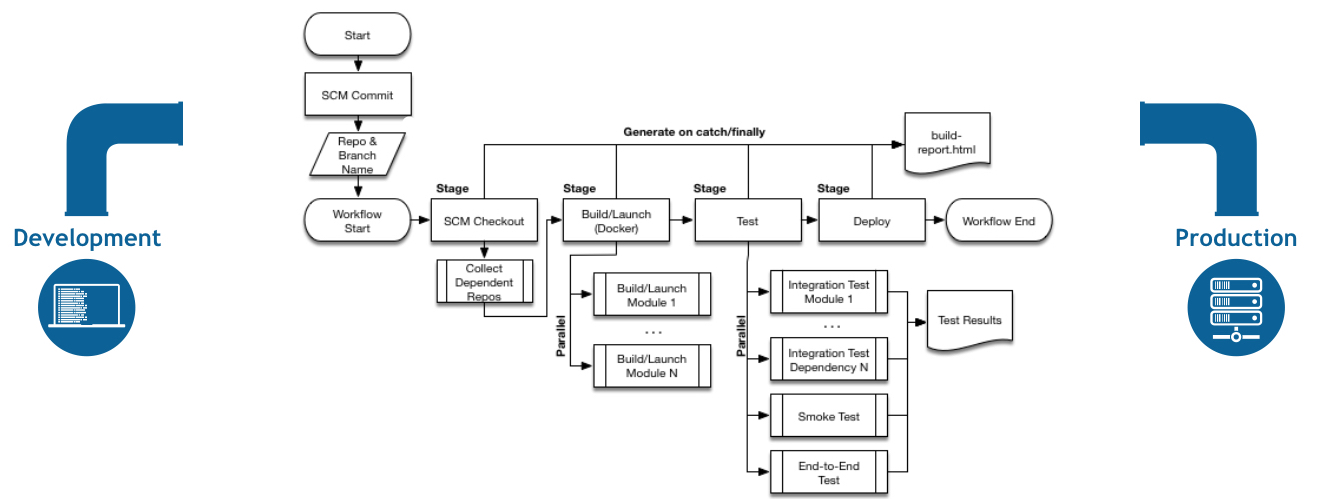
Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

* **Code**: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
* **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins master.
* **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
* **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
* **Extensible**: The Pipeline plugin supports custom extensions to its DSL [[1](https://jenkins.io/doc/book/pipeline/#_footnotedef_1)] and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, [[4](https://jenkins.io/doc/book/pipeline/#_footnotedef_4)] Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with [Pipeline Shared Libraries](https://jenkins.io/doc/book/pipeline/shared-libraries) and by plugin developers. [[5](https://jenkins.io/doc/book/pipeline/#_footnotedef_5)]

The flowchart below is an example of one CD scenario easily modeled in Jenkins Pipeline:



**Pipeline concepts**

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the [overview](https://jenkins.io/doc/book/pipeline/#pipeline-syntax-overview) below).

**Pipeline**

A Pipeline is a user-defined model of a CD pipeline. A Pipeline’s code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Also, a pipeline block is a [key part of Declarative Pipeline syntax](https://jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals).

**Node** A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.Also, a node block is a [key part of Scripted Pipeline syntax](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals).

**Stage** A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. [[6](https://jenkins.io/doc/book/pipeline/#_footnotedef_6)]

**Step** A single task. Fundamentally, a step tells Jenkins *what* to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, [[1](https://jenkins.io/doc/book/pipeline/#_footnotedef_1)] that typically means the plugin has implemented a new *step*.

**Pipeline syntax overview** The following Pipeline code skeletons illustrate the fundamental differences between [Declarative Pipeline syntax](https://jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals) and [Scripted Pipeline syntax](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals).

Be aware that both [stages](https://jenkins.io/doc/book/pipeline/#stage) and [steps](https://jenkins.io/doc/book/pipeline/#step) (above) are common elements of both Declarative and Scripted Pipeline syntax.

**Declarative Pipeline fundamentals**

In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline)

pipeline {

agent any

stages {

stage('Build') {

steps {

//

}

}

stage('Test') {

steps {

//

}

}

stage('Deploy') {

steps {

//

}

}

}

}

|  |  |
| --- | --- |
|  | Execute this Pipeline or any of its stages, on any available agent. |
|  | Defines the "Build" stage. |
|  | Perform some steps related to the "Build" stage. |
|  | Defines the "Test" stage. |
|  | Perform some steps related to the "Test" stage. |
|  | Defines the "Deploy" stage. |
|  | Perform some steps related to the "Deploy" stage. |

**Scripted Pipeline fundamentals**

In Scripted Pipeline syntax, one or more node blocks do the core work throughout the entire Pipeline. Although this is not a mandatory requirement of Scripted Pipeline syntax, confining your Pipeline’s work inside of a node block does two things:

1. Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an executor is free on a node, the steps will run.
2. Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control.  
   **Caution:** Depending on your Jenkins configuration, some workspaces may not get automatically cleaned up after a period of inactivity. See tickets and discussion linked from [JENKINS-2111](https://issues.jenkins-ci.org/browse/JENKINS-2111) for more information.

Jenkinsfile (Scripted Pipeline)

node {

stage('Build') {

//

}

stage('Test') {

//

}

stage('Deploy') {

//

}

}

|  |  |
| --- | --- |
|  | Execute this Pipeline or any of its stages, on any available agent. |
|  | Defines the "Build" stage. stage blocks are optional in Scripted Pipeline syntax. However, implementing stage blocks in a Scripted Pipeline provides clearer visualization of each `stage’s subset of tasks/steps in the Jenkins UI. |
|  | Perform some steps related to the "Build" stage. |
|  | Defines the "Test" stage. |
|  | Perform some steps related to the "Test" stage. |
|  | Defines the "Deploy" stage. |
|  | Perform some steps related to the "Deploy" stage. |

**Pipeline example**

Here is an example of a Jenkinsfile using Declarative Pipeline syntax - its Scripted syntax equivalent can be accessed by clicking the **Toggle Scripted Pipeline** link below:

Jenkinsfile (Declarative Pipeline)

pipeline {

agent any

options {

skipStagesAfterUnstable()

}

stages {

stage('Build') {

steps {

sh 'make'

}

}

stage('Test'){

steps {

sh 'make check'

junit 'reports/\*\*/\*.xml'

}

}

stage('Deploy') {

steps {

sh 'make publish'

}

}

}

}