**JenkinFile**

There are two types of Jenkinfile **Scripted** and **Declarative**.

**Declarative Pipelines :**

A valid Declarative pipeline must be defined with the "pipeline" sentence.

**Required sections.**

**Agent** : Used to specify node where pipeline has to be run. **Parameters** :   
 **Any :** execute the pipeline on any available agent.  
 **None :** when none is applied on to level agent then each stage needs to have its own agent section.  
 **label :** execute pipeline on agent with provided label.

**Stages :** his section allows generation of different stages on your pipeline. Stages must be named accordingly since Jenkins will display on interface.

**Stage :** at least one stage must be defined under stages.

**Steps :** which is defined in a "stage." At least one step must be defined in the "steps" section.

**Optional Required sections.**

**Environment :** This can be defied at pipeline(applied to all stages) or stage(applied to particular stage) level.

environment {

OUTPUT\_PATH = './outputs/'

}

**Input :** This is defined at stage , which will provide prompt for user input.

The following configuration options can be used for this directive:

**message**: This is a required option where the message to be displayed to the user is specified.

**id**: Optional identifier for the input. By default, the "stage" name is used.

**ok**: Optional text for the Ok button.

**submitter**: Optional list of users or external group names who are allowed to submit the input. By default, any user is allowed.

**submitterParameter**: Optional name of an environment variable to set with the submitter name, if present.

**parameters**: Optional list of parameters to be provided by the submitter.

input{

message "Press Ok to continue"

submitter "user1,user2"

parameters {

string(name:'username', defaultValue: 'user', description: 'Username of the user pressing Ok')

}

}

**Options :** This applied at pipeline level, this directive will group specific option.

**buildDiscarder :** Persist artifacts and console output for the specific number of recent Pipeline runs. example: options { buildDiscarder(logRotator(numToKeepStr: '1')) }

**checkoutToSubdirectory :** Perform the automatic source control checkout in a subdirectory of the workspace.

example: options { checkoutToSubdirectory('foo') }

**disableConcurrentBuilds :** Disallow concurrent executions of the Pipeline. Can be useful for preventing simultaneous accesses to shared resources, etc.   
example: options { disableConcurrentBuilds() }

**newContainerPerStage :** Used with docker or dockerfile top-level agent. When specified, each stage will run in a new container instance on the same node, rather than all stages running in the same container instance.

**preserveStashes :** Preserve stashes from completed builds, for use with stage restarting.

example: options { preserveStashes() } preserve the stashes from the most recent completed build, options { preserveStashes(5) } to preserve the stashes from the five most recent completed builds.

**Retry :** On failure, retry the entire Pipeline the specified number of times.

example: options { retry(3) }

**skipDefaultCheckout :** Skip checking out code from source control by default in the agent directive. example: options { skipDefaultCheckout() }

**skipStagesAfterUnstable** : Skip stages once the build status has gone to UNSTABLE.

example: options { skipStagesAfterUnstable() }

**timeout :** Set a timeout period for the Pipeline run, after which Jenkins should abort the Pipeline. example: options { timeout(time: 1, unit: 'HOURS') }

**timestamps :** Prepend all console output generated by the Pipeline run with the time at which the line was emitted.   
example: options { timestamps() }

**Parallel :** This is applied on stage, Stages can have other stages nested inside that will be executed in parallel.

stages {

stage('Run Tests') {

parallel {

stage('Test On Windows') {

agent {

label "windows"

}

steps {

bat "run-tests.bat"

}

}

stage('Test On Linux') {

agent {

label "linux"

}

steps {

sh "run-tests.sh"

}

}

}

}

**Parameters** : This directive allows you to define a list of parameters to be used in the script. It should be defined at a top level and only one directive is allowed for the whole pipeline.

parameters {

string(name: 'user', defaultValue: 'John', description: 'A user that triggers the pipeline')

}

**Post :** Post sections can be added at a pipeline level or on each stage block and sentences included in it are executed once the stage or pipeline completes. Several post-conditions can be used to control whether the post executes or not:

**always**: Steps are executed regardless of the completion status.

**changed**: Executes only if the completion results in a different status than the previous run.

**fixed**: Executes only if the completion is successful and the previous run failed

**ecutes** only if current execution fails, aborts or is unstable and the previous run was successful.

**aborted**: Steps are executed only if the pipeline or stage is aborted.

**failure**: Steps are executed only if the pipeline or stage fails.

**success**: Steps are executed only if the pipeline or stage succeeds.

**unstable**: Steps are executed only if the pipeline or stage is unstable.

post {

always {

echo“ Pipeline finished”

bat. / performCleanUp.bat

}

**Script:** This step is used to add Scripted Pipeline sentences into a Declarative one, thus providing even more functionality. This step must be included at "stage" level.

stage('Sample') {

steps {

echo "Scripted block"

script {

}

}

}

**Tools :** The "tools" directive can be added either at a pipeline level or at the stage level. It allows you to specify which maven, jdk, or gradle version to use on your script.

tools {

maven 'apache-maven-3.0.1'

}

**Triggers :** Triggers allow Jenkins to automatically trigger pipelines by using any of the available ones:

**cron**: By using cron syntax.

triggers {

//Execute weekdays every four hours starting at minute 0

cron('0 \*/4 \* \* 1-5')

}

**pollSCM**: By using cron syntax, it allows you to define when Jenkins will check for new source repository updates.

triggers {

//Query repository weekdays every four hours starting at minute 0

pollSCM('0 \*/4 \* \* 1-5')

}

**upstream**: Takes as input a list of Jenkins jobs and a threshold. The pipeline will be triggered when any of the jobs on the list finish with the threshold condition.

triggers {

//Execute when either job1 or job2 are successful

upstream(upstreamProjects: 'job1, job2', threshold: hudson.model.Result.SUCCESS)

}

**When :** Pipeline steps could be executed depending on the conditions defined in a "when" directive. If conditions match, the steps defined in the corresponding stage will be run. It should be defined at a stage level.

tage('Deploy stage') {

when {

branch 'master'

}

steps {

echo 'Deploy master to stage'

...

}

<https://jenkins.io/doc/book/pipeline/syntax/#options>

<https://jenkins.io/doc/book/pipeline/jenkinsfile/>

<https://jenkins.io/doc/pipeline/tour/hello-world/>

<https://dzone.com/refcardz/declarative-pipeline-with-jenkins?chapter=7>