



ETECHCONSULTINGLLC

Jenkins-series
Pipeline as a code
Etech Consulting Devops Master class
+13478735512/+16677868741

Declarative Pipeline is a relatively recent addition to Jenkins Pipeline ^[1] which presents a more simplified and opinionated syntax on top of the Pipeline sub-systems.

All valid Declarative Pipelines must be enclosed within a `pipeline` block, for example:

```
pipeline {  
    /* insert Declarative Pipeline here */  
}
```

The basic statements and expressions which are valid in Declarative Pipeline follow the same rules as [Groovy's syntax](#) with the following exceptions:

- The top-level of the Pipeline must be a *block*, specifically: `pipeline { }`
- No semicolons as statement separators. Each statement has to be on its own line
- Blocks must only consist of declarative sections, declarative directives, declarative steps, or assignment statements.
- A property reference statement is treated as no-argument method invocation. So for example, `input` is treated as `input()`

Sections

Sections in Declarative Pipeline typically contain one or more declarative directives or declarative steps.

post

The `post` section defines actions which will be run at the end of the Pipeline run. A number of additional post conditions blocks are supported within the `post` section: `always`, `changed`, `failure`, `success`, and `unstable`. These blocks allow for the execution of steps at the tail-end of the Pipeline run, depending on the status of the Pipeline.

Required	No
Parameters	None
Allowed	In the top-level <code>pipeline</code> block and each <code>stage</code> block.

Conditions

`always`

Run regardless of the completion status of the Pipeline run.

changed

Only run if the current Pipeline run has a different status from the previously completed Pipeline.

failure

Only run if the current Pipeline has a "failed" status, typically denoted in the web UI with a red indication.

success

Only run if the current Pipeline has a "success" status, typically denoted in the web UI with a blue or green indication.

unstable

Only run if the current Pipeline has an "unstable" status, usually caused by test failures, code violations, etc. Typically denoted in the web UI with a yellow indication.

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
  post {
    always {
      echo 'I will always say Hello again!'
    }
  }
}
// Script //
```

Conventionally, the **post** section should be placed at the end of the Pipeline.

The post conditions blocks can use steps.

stages

A sequence of one or more stage directives, the **stages** section is where the bulk of the "work" described by a Pipeline will be located. At a minimum it is recommended that **stages** contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy.

Required	Yes
Parameters	None

Allowed	Only once, inside the pipeline block.
----------------	--

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
}
```

```
// Script //
```

The **stages** section will typically follow the directives such as **agent**, **options**, etc.
steps

Defines a series of steps to be executed in a given **stage** directive.

Required	Yes
Parameters	None
Allowed	Inside each stage block.

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
}
```

```
// Script //
```

The **steps** section must contain one or more steps.
Directives

agent

The `agent` directive specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment depending on where the `agent` directive is placed. The directive must be defined at the top-level inside the `pipeline` block, but stage-level usage is optional.

Required	Yes
Parameters	Described below
Allowed	In the top-level <code>pipeline</code> block and each <code>stage</code> block.

Parameters

In order to support the wide variety of use-cases Pipeline authors may have, the `agent` directive supports a few different types of parameters. These parameters can be applied at the top-level of the `pipeline` block, or within each `stage` directive.

any

Execute the Pipeline, or stage, on any available agent. For example: `agent any`

none

When applied at the top-level of the `pipeline` block no global agent will be allocated for the entire Pipeline run and each `stage` directive will need to contain its own `agent` directive. For example: `agent none`

label

Execute the Pipeline, or stage, on an agent available in the Jenkins environment with the provided label. For example: `agent { label 'my-defined-label' }`

node

`agent { node { label 'labelName' } }` behaves the same as `agent { label 'labelName' }`, but `node` allows for additional options (such as `customWorkspace`).

docker

Execute the Pipeline, or stage, with the given container which will be dynamically provisioned on a node pre-configured to accept Docker-based Pipelines, or on a node matching the optionally defined `label` parameter. `docker` also optionally accepts an `args` parameter which may contain arguments to pass directly to a `docker run` invocation. For example: `agent { docker 'maven:3-alpine' }` or

```
agent {  
  docker {  
    image 'maven:3-alpine'  
    label 'my-defined-label'  
    args '-v /tmp:/tmp'  
  }  
}
```

dockerfile

Execute the Pipeline, or stage, with a container built from a `Dockerfile` contained in the source repository. Conventionally this is the `Dockerfile` in the root of the source repository: `agent { dockerfile true }`. If building a `Dockerfile` in another directory, use the `dir` option: `agent { dockerfile { dir 'someSubDir' } }`.

Common Options

These are a few options for two or more **agent** implementations. They are not required unless explicitly stated.

label

A string. The label on which to run the Pipeline or individual **stage**.

This option is valid for **node**, **docker** and **dockerfile**, and is required for **node**.

customWorkspace

A string. Run the Pipeline or individual **stage** this **agent** is applied to within this custom workspace, rather than the default. It can be either a relative path, in which case the custom workspace will be under the workspace root on the node, or an absolute path. For example:

```
agent {  
  node {  
    label 'my-defined-label'  
    customWorkspace '/some/other/path'  
  }  
}
```

This option is valid for **node**, **docker** and **dockerfile**.

reuseNode

A boolean, false by default. If true, run the container in the node specified at the top-level of the Pipeline, in the same workspace, rather than on a new node entirely.

This option is valid for **docker** and **dockerfile**, and only has an effect when used on an **agent** for an individual **stage**.

Example

```
// Declarative //  
pipeline {  
  agent { docker 'maven:3-alpine' }  
  stages {  
    stage('Example Build') {  
      steps {  
        sh 'mvn -B clean verify'  
      }  
    }  
  }  
}
```

```
// Script //
```

Execute all the steps defined in this Pipeline within a newly created container of the given name and tag (**maven:3-alpine**).

Stage-level **agent** directive

```
// Declarative //
pipeline {
  agent none
  stages {
    stage('Example Build') {
      agent { docker 'maven:3-alpine' }
      steps {
        echo 'Hello, Maven'
        sh 'mvn --version'
      }
    }
    stage('Example Test') {
      agent { docker 'openjdk:8-jre' }
      steps {
        echo 'Hello, JDK'
        sh 'java -version'
      }
    }
  }
}
// Script //
```

Defining **agent none** at the top-level of the Pipeline ensures that executors will not be created unnecessarily. Using **agent none** requires that each **stage** directive contain an **agent** directive. Execute the steps contained within this stage using the given container.

Execute the steps contained within this steps using a different image from the previous stage. environment

The **environment** directive specifies a sequence of key-value pairs which will be defined as environment variables for the all steps, or stage-specific steps, depending on where the **environment** directive is located within the Pipeline.

This directive supports a special helper method **credentials()** which can be used to access pre-defined Credentials by their identifier in the Jenkins environment. For Credentials which are of type "Secret Text", the **credentials()** method will ensure that the environment variable specified contains the Secret Text contents. For Credentials which are of type "Standard username and password", the environment variable specified will be set to **username:password** and two additional environment variables will be automatically be defined: **MYVARNAME_USR** and **MYVARNAME_PSW** respective.

Required	No
Parameters	None
Allowed	Inside the pipeline block, or within stage directives.

Example

```
// Declarative //
pipeline {
  agent any
  environment {
    CC = 'clang'
  }
  stages {
    stage('Example') {
      environment {
        AN_ACCESS_KEY = credentials('my-prefined-secret-text')
      }
      steps {
        sh 'printenv'
      }
    }
  }
}
// Script //
```

An **environment** directive used in the top-level **pipeline** block will apply to all steps within the Pipeline.

An **environment** directive defined within a **stage** will only apply the given environment variables to steps within the **stage**.

The **environment** block has a helper method **credentials()** defined which can be used to access pre-defined Credentials by their identifier in the Jenkins environment.

options

The **options** directive allows configuring Pipeline-specific options from within the Pipeline itself. Pipeline provides a number of these options, such as **buildDiscarder**, but they may also be provided by plugins, such as **timestamps**.

Required	No
Parameters	None
Allowed	Only once, inside the pipeline block.

Available Options

buildDiscarder

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

disableConcurrentBuilds

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

skipDefaultCheckout

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

timeout

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

retry

On failure, retry the entire Pipeline the specified number of times. For example: `options { retry(3) }`


timestamps

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

Example

```
// Declarative //
pipeline {
  agent any
  options {
    timeout(time: 1, unit: 'HOURS')
  }
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
}
// Script //
```

Specifying a global execution timeout of one hour, after which Jenkins will abort the Pipeline run.

 A comprehensive list of available options is pending the completion of [INFRA-1503](#).
parameters

The `parameters` directive provides a list of parameters which a user should provide when triggering the Pipeline. The values for these user-specified parameters are made available to Pipeline steps via the `params` object, see the [Example](#) for its specific usage.

Required	No
Parameters	None
Allowed	Only once, inside the <code>pipeline</code> block.

Available Parameters

string

A parameter of a string type, for example: `parameters { string(name: 'DEPLOY_ENV', defaultValue: 'staging', description: ") }`

booleanParam

A boolean parameter, for example: `parameters { booleanParam(name: 'DEBUG_BUILD', defaultValue: true, description: ") }`

Example

```
// Declarative //
pipeline {
  agent any
  parameters {
    string(name: 'PERSON', defaultValue: 'Mr Jenkins', description: 'Who should I say hello
to?')
  }
  stages {
    stage('Example') {
      steps {
        echo "Hello ${params.PERSON}"
      }
    }
  }
}
// Script //
```

A comprehensive list of available parameters is pending the completion of [INFRA-1503](#).
triggers

The `triggers` directive defines the automated ways in which the Pipeline should be re-triggered. For Pipelines which are integrated with a source such as GitHub or Bitbucket, `triggers` may not be necessary as webhooks-based integration will likely already be present. Currently the only two available triggers are `cron` and `pollSCM`.

Required	No
Parameters	None
Allowed	Only once, inside the <code>pipeline</code> block.

cron

Accepts a cron-style string to define a regular interval at which the Pipeline should be re-triggered, for example: `triggers { cron('H 4/* 0 0 1-5') }`

pollSCM

Accepts a cron-style string to define a regular interval at which Jenkins should check for new source changes. If new changes exist, the Pipeline will be re-triggered. For example: `triggers { pollSCM('H 4/* 0 0 1-5') }`

The `pollSCM` trigger is only available in Jenkins 2.22 or later.

Example

```
// Declarative //
pipeline {
  agent any
  triggers {
    cron('H 4/* 0 0 1-5')
  }
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
}
```

// Script //

stage

The `stage` directive goes in the `stages` section and should contain a `steps` directive, an optional `agent` directive, or other stage-specific directives. Practically speaking, all of the real work done by a Pipeline will be wrapped in one or more `stage` directives.

Required	At least one
Parameters	One mandatory parameter, a string for the name of the stage.
Allowed	Inside the <code>stages</code> section.

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
}
```

```

    }
  }
}
// Script //

```

tools

A section defining tools to auto-install and put on the **PATH**. This is ignored if **agent none** is specified.

Required	No
Parameters	<i>None</i>
Allowed	Inside the pipeline block or a stage block.

Supported Tools

maven
jdk
gradle

Example

```

// Declarative //
pipeline {
  agent any
  tools {
    maven 'apache-maven-3.0.1'
  }
  stages {
    stage('Example') {
      steps {
        sh 'mvn --version'
      }
    }
  }
}
// Script //

```

The tool name must be pre-configured in Jenkins under **Manage Jenkins → Global Tool Configuration**.

when

The **when** directive allows the Pipeline to determine whether the stage should be executed depending on the given condition.

Required	No
Parameters	<i>None</i>
Allowed	Inside a <code>stage</code> directive

Built-in Conditions

branch

Execute the stage when the branch being built matches the branch pattern given, for example: `when { branch 'default' }`

environment

Execute the stage when the specified environment variable is set to the given value, for example: `when { environment name: 'DEPLOY_TO', value: 'production' }`

expression

Execute the stage when the specified Groovy expression evaluates to true, for example: `when { expression { return params.DEBUG_BUILD } }`

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example Build') {
      steps {
        echo 'Hello World'
      }
    }
    stage('Example Deploy') {
      when {
        branch 'production'
      }
      echo 'Deploying'
    }
  }
}
// Script //
```

matrix

This is a directive similar to `stages` and `parallel` and exclusive of those.
axes

Required, contains 1 or more axis directives. All values of n-axis directives are crossed together to form an n-dimensional matrix.

axis

Directive specifies the name of an axis and the list of values for that axis.
name

Create an axis with this name. Axis names must be unique with a matrix.
values

List of one or more values for this axis.
excludes

Optional, contains 1 or more exclude directives. Combines exclude directives (logical-or) to select cells to exclude.

exclude

Contains 1 or more axis directives. Combines axis directives as (logical-and) to select cells to exclude.
axis

An axis to filter.

name

Select an axis by name.

values/notValues

Match axis values to either a specific list of values or to all values not in a list of values (notValues).
agent

The same as the stage **agent** directive but applied to each cell. Axis values may be used by this directive.
environment

The same as the stage **environment** directive but applied to each cell. Axis values may be used by this directive.
input

The same as the stage **input** directive but applied to each cell. Axis values may be used by this directive.

options

The same as the stage `options` directive but applied to each cell. Axis values may be used by this directive.

post

The same as the stage `post` directive but applied to each cell. Axis values may be used by this directive.

tools

The same as the stage `tools` directive but applied to each cell. Axis values may be used by this directive.

when

The same as the stage `when` directive but applied to each cell. Axis values may be used by this directive.

stages

Required, same behavior as stage `stages` section.

Steps

Declarative Pipelines may use all the available steps documented in the [Pipeline Steps reference](#), which contains a comprehensive list of steps, with the addition of the steps listed below which are **only supported** in Declarative Pipeline.

script

The `script` step takes a block of scripted-pipeline and executes that in the Declarative Pipeline. For most use-cases, the `script` step should be unnecessary in Declarative Pipelines, but it can provide a useful "escape hatch." `script` blocks of non-trivial size and/or complexity should be moved into [Shared Libraries](#) instead.

Example

```
// Declarative //
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'

        script {
          def browsers = ['chrome', 'firefox']
          for (int i = 0; i < browsers.size(); ++i) {
            echo "Testing the ${browsers[i]} browser"
```

```
}  
}  
}  
}  
}  
}  
}  
// Script //
```



ETECHCONSULTINGLLC