Declarative Pipeline with Jenkins

BY PATRICK WOLF

CONTENTS

DECLARATIVE PIPELINE

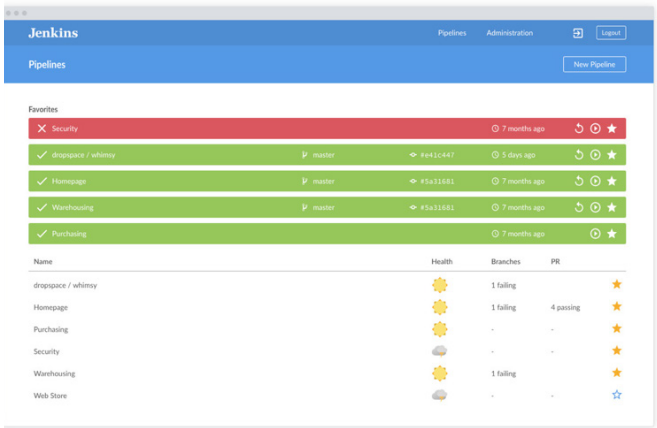
Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive continuous delivery pipelines. The steps to build, test, and deliver each application become part of the application itself, stored in a Jenkinsfile.

Jenkins Pipeline execution engine supports two DSL syntaxes: Scripted Pipeline and Declarative Pipeline. Scripted Pipeline allows users to code their Pipelines using a Groovy DSL. Declarative Pipeline replaces Groovy variable assignments, control structures, loops, and exception handling with a predefined structure and model to allow users of all experience levels to quickly create consistent, concise Pipelines without learning Groovy.

JENKINS BLUE OCEAN

Blue Ocean is a rethinking of the Jenkins user experience, focused on Continuous Delivery pipelines and keeping pace with the expectations for modern development tools. With a simple click, users can switch between traditional Jenkins pages for administrative tasks and Blue Ocean to monitor the progress of their Pipelines.

With the focus on new users and CD, Blue Ocean and Declarative Pipeline were designed to work together; making the creation, review, and visualization of Pipelines accessible to all members of the DevOps team. Users can quickly see their pipeline's status, visually create and edit new Pipelines, personalize their view of important Pipelines, and easily collaborate on code with native integrations for branch and pull requests.



ABOUT THIS REFCARD

This Refcard will focus on Declarative Pipeline and Blue Ocean as the preferred method for all users, especially new and intermediate users, to create, view and edit continuous delivery pipelines using Jenkinsfiles. The Refcard will provide an overview of the essential pieces of Declarative Pipeline and Blue Ocean including a full syntax reference. In addition, a complete real-world Continuous Delivery example is included, combining and extending the previous example snippets shown.

SYSTEM REQUIREMENTS

In order to use Declarative Pipeline and Blue Ocean the following must be installed:

- Jenkins 2.7.4 or Higher
- Pipeline Plugin 2.5 or Higher
- Blue Ocean Plugin 1.0 or Higher

To start a new Jenkins with Pipeline and Blue Ocean pre-installed:

- Ensure Docker is installed.
- Run "docker run -p 8888:8080 jenkinsci/blueocean:latest"
- Browse to localhost:8888/blue

THE HUB OF ENTERPRISE JENKINS AND DEVOPS

Software at the speed of ideas.

Continually build, deliver and improve the software that fuels your business with CloudBees Jenkins Solutions.

Your business has amazing, world-changing ideas - the only thing standing in your way is the time, energy and process it takes to turn code into finished product, to transform ideas into impact. CloudBees - the only secure, scalable and supported Jenkins-based DevOps platform - lets you focus on the ideas you want to bring to life, not the challenge of building, testing and deploying them, so you can start making an impact sooner.

With CloudBees, we'll manage the enterprise DevOps automation letting your team focus on building great software!

DevOps at scale:
Entrust your enterprise DevOps initiative to the only cloud-native Distributed Pipeline Architecture.

Easy to manage:
Enterprise features enable teams to securely share best practices.

Highly available and resilient:
Your business is critical. CloudBees Jenkins Solutions ensure availability with a self-healing Jenkins infrastructure.

Enterprise grade security:
You define fine-grained access, by role. Compliance just got a whole lot easier!

Try CloudBees Jenkins Solutions for free: www.cloudbees.com/get-started

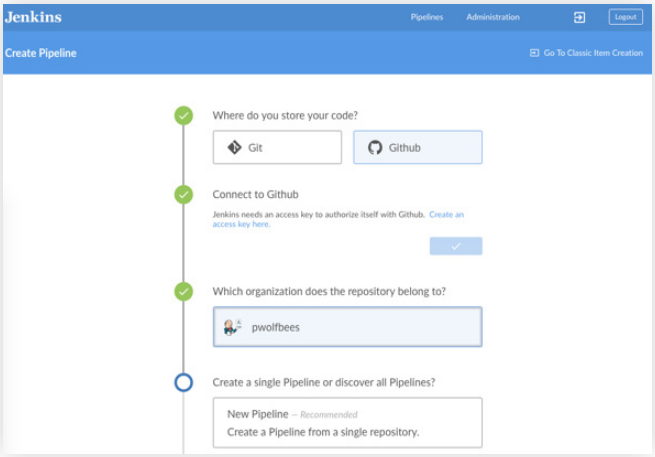
CloudBees, Inc.
2001 Gateway Place, Suite 670W | San Jose, CA 95110 | United States
www.cloudbees.com | info@cloudbees.com

CloudBees Jenkins Solutions are built on top of open source Jenkins, an independent community project. Read more at: www.cloudbees.com/jenkins/about

0517v00

CREATING A JENKINSFILE

The Blue Ocean Pipeline Creation workflow guides users through creating a new Pipeline in clear, easy-to-understand steps.



PIPELINE EDITOR

This will open the Blue Ocean Pipeline Editor, which allows users to quickly design and save a new Jenkinsfile in the selected repository. If a Jenkinsfile is already present in this repository, then Blue Ocean will not open editor and create a new Pipeline based on the existing Jenkinsfile instead. You can always edit this Jenkinsfile in the editor by selecting a branch and clicking on the pencil icon.

BRANCHES AND PULL REQUESTS

When a new Pipeline in Blue Ocean is created, it also creates a new Multibranch Pipeline project for the repository specified. Jenkins will monitor this repository and automatically create a new Pipeline for each Branch and Pull Request in the repository that contains a Jenkinsfile. Removing the Branch, Pull Request, or the Jenkinsfile from the repository will remove the Pipeline from Jenkins as well.

Users can modify the Jenkinsfile in each Branch or Pull Request to perform different steps, but it is recommended that all of the different Pipeline flows are defined in the primary Jenkinsfile with choices made based on the BRANCH_NAME environment variable or other conditions defined in the stages.

PIPELINE FUNDAMENTALS

In its simplest form, a Pipeline will automatically check out the code in the same repository that contains the Jenkinsfile, run on an agent and be grouped into stages that contain steps defining specific actions. All Declarative Pipelines must start with `pipeline` and include these 4 directives to be syntactically correct: `agent`, `stages`, `stage`, and `steps`. These will be described in more detail in following sections. The following is an example of a bare minimum Pipeline:

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'npm --version'
      }
    }
  }
}
```

STEPS AND STAGES

STEPS

At the most basic level, Pipelines are made up of multiple steps that allow you to build, test, and deploy applications. Think of a “step” like a single command which performs a single action. When a step succeeds it moves onto the next step. steps are declared in their own section to differentiate them from configuration settings. A steps section may only be contained within a stage.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'echo “Hello World”'
        sh '''
          echo “Multiline shell steps”
          ls -lah
        '''
      }
    }
  }
}
```

SHELL STEPS

Jenkins is a distributed system meant to work across multiple nodes and executors allowing Jenkins to scale the number of Pipelines being run simultaneously and to orchestrate tasks on nodes with different operating systems, tools, environments, etc. Most Pipelines work best by running native CLI commands on different executors. This allows users to automate tasks run on their local machines by copying the commands or scripts directly into Pipeline.

Pipeline supports sh for Linux and macOS, and either bat or powershell on Windows.

TIMEOUTS, RETRIES AND OTHER WRAPPERS

There are some powerful steps that “wrap” other steps which can easily solve problems like retrying (retry) steps until successful or exiting if a step takes too long (timeout).

Wrappers may contain multiple steps or may be recursive and contain other wrappers. We can compose these steps together. For example, if we wanted to retry our deployment five times, but never want to spend more than 3 minutes in total before failing the stage:

```
pipeline {
  agent any
  stages {
    stage('Deploy') {
     steps {
       timeout(time: 3, unit: 'MINUTES') {
         retry(5) {
           powershell '.\flakey-deploy.ps1'
         }
       }
     }
    }
  }
}
```

STAGES

A stage in a Pipeline is a collection of related steps that share a common execution environment. Each stage must be named and must contain a steps section. All stages are declared in the stages section of your Pipeline.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying'
      }
    }
  }
}
```

WHEN

A stage may be optionally skipped in a Pipeline based on criteria defined in the when section of the stage. branch and environment are supported keywords while expression can be utilized to evaluate any Groovy expression with a Boolean return.

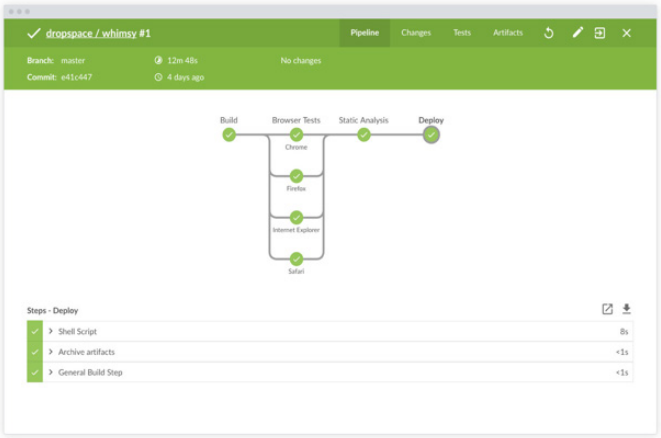
```
pipeline {
  agent any
  stages {
    stage('Build') {
      when {
        expression {
          "foo" == "bar"
        }
      }
      steps {
        echo 'Building'
      }
    }
    stage('Test') {
      when {
        environment name: 'JOB_NAME', value: 'foo'
      }
      steps {
        echo 'Testing'
      }
    }
    stage('Deploy') {
      when {
        branch 'master'
      }
```

Multiple conditions may be combined in when by using anyOf, allOf or not to create complex skip conditions.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      when {
        allOf {
          not { branch 'master' }
          environment name: 'JOB_NAME', value: 'Foo'
        }
      }
      steps {
        echo 'Building'
      }
    }
  }
}
```

PARALLEL STAGES

To maximize efficiency of your Pipeline some stages can be run in parallel if they do not depend on each other. Tests are a good example of stages that can run in parallel.




```
pipeline {
  agent any
  stages {
    stage('Browser Tests') {
      parallel {
        stage('Chrome') {
          steps {
            echo "Chrome Tests"
          }
        }
        stage('Firefox') {
          steps {
            echo "Firefox Tests"
          }
        }
      }
    }
  }
}
```

STAGE SETTINGS

environment, ***post***, ***agent***, and ***tools*** may be optionally defined in a stage. These settings are defined in detail below:

AGENTS

The agent directive tells Jenkins where and how to execute the Pipeline, or subset thereof. As you might expect, the agent is required for all Pipelines.

So far, all examples have used agent any, meaning that the Pipeline can execute on any available agent. Underneath the hood, there are a few things agent causes to happen:

- All the steps contained within the block are queued for execution by Jenkins. As soon as an executor is available, the steps will begin to execute.
- A workspace is allocated which will contain files checked out from source control, as well as any additional working files for the Pipeline.

One or more agents in Jenkins are grouped by a label. Specifying a label for your agent will restrict the potential agents that can be used in your Pipeline.

```
pipeline {
  agent {
    node { label 'my-agent' }
  }
}
```

It is also possible to use a custom workspace directory on each agent using a relative or absolute path reference to maintain a consistent file location.

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

DOCKER

Pipeline is designed to easily use Docker images and containers. This allows the Pipeline to define the environment and tools required without having to configure various system tools and dependencies on agents manually. This approach allows you to use practically any tool which can be packaged in a Docker container.

```
pipeline {
  agent {
    docker {
      label 'docker'
      image 'maven:3.5.0-jdk-8'
    }
  }
}
```

One of the advantages of using containers is creating an immutable environment that defines only the tools required in a consistent manner. Rather than creating one large image with every tool needed by the Pipeline, it is possible to use different containers in each stage and reuse the workspace, keeping all of your files in one place.

```
pipeline {
  agent {
    node { label ‘my-docker’ }
  }
  stages {
    stage(“Build”) {
      agent {
        docker {
        reuseNode true
        image ‘maven:3.5.0-jdk-8’
        }
      }
      steps {
        sh ‘mvn install’
      }
    }
  }
}
```

This will check out the source code onto an agent with the label my-docker and re-use the workspace while running the Build steps inside the selected Docker container.

ENVIRONMENT VARIABLES AND CREDENTIALS

Environment variables can be set globally, like the example below, or per stage. As you might expect, setting environment variables per stage means they will only apply to the stage in which they’re defined.

```
pipeline {
  agent any
  environment {
    DISABLE_AUTH = 'true'
    DB_ENGINE    = 'sqlite'
  }
  stages {
    stage('Build') {
      steps {
        sh 'printenv'
      }
    }
  }
}
```

This approach to defining environment variables from within the Jenkinsfile can be very useful for instructing scripts, such as a Makefile, to configure the build or tests differently to run them inside of Jenkins.

Another common use for environment variables is to set or override “dummy” credentials in build or test scripts. Because it’s (obviously) a bad idea to put credentials directly into a Jenkinsfile, Jenkins Pipeline allows users to quickly and safely access pre-defined credentials in the Jenkinsfile without ever needing to know their values.

CREDENTIALS IN THE ENVIRONMENT

If your Jenkins environment has credentials configured, such

as build secrets or API tokens, those can be easily inserted into environment variables for use in the Pipeline. The snippet below is for "Secret Text" type Credentials, for example.

```
environment {
  AWS_ACCESS = credentials('AWS_ KEY')
  AWS_SECRET = credentials('AWS_SECRET')
}
```

Just as the first example, these variables will be available either globally or per-stage depending on where the environment directive is located in the Jenkinsfile.

The second most common type of Credentials is "Username and Password" which can still be used in the environment directive, but results in slightly different variables being set.

```
environment {
  SAUCE_ACCESS = credentials('sauce-dev')
}
```

This will actually set 3 environment variables:

```
SAUCE_ACCESS containing <username>:<password>
SAUCE_ACCESS_USR containing the username
SAUCE_ACCESS_PSW containing the password
```

POST ACTIONS

The entire Pipeline and each Stage may optionally define a Post section. This Post section of the is guaranteed to run at the end of the enclosing Pipeline or Stage so it is useful for cleaning up files, archiving results, or sending notifications. A number of additional conditions blocks are supported within the post section: `always`, `changed`, `failure`, `success`, and `unstable`. They are executed at the end of a Stage or Pipeline depending upon the status of the run.

```
pipeline {
  agent any
  stages {
    stage('No-op') {
      steps {
        sh 'ls'
      }
    }
  }
  post {
    always {
      echo 'I have finished'
      deleteDir() // clean up workspace
    }
    success {
      echo 'I succeeded!'
    }
    unstable {
      echo 'I am unstable :/'
    }
    failure {
      echo 'I failed :('
    }
    changed {
      echo 'Things are different...'
    }
  }
}
```

NOTIFICATIONS

EMAIL

```
post {
  failure {
    mail to: 'team@example.com',
      subject: 'Failed Pipeline',
      body: "Something is wrong"
  }
}
```

HIPCHAT

```
post {
  failure {
    hipchatSend color: 'RED',
    message:'@here build failed.
  }
}
```

SLACK

```
post {
  success {
    slackSend channel:'#ops-room',
      color: 'good',
      message: 'Completed successfully.'
  }
}
```

HUMAN INPUT

Often, when passing between stages, especially environment stages, you may want human input before continuing. For example, to judge if the application is in a good enough state to "promote" to the production environment. This can be accomplished with the input step. In the example below, the "Sanity check" stage actually blocks for input and won't proceed without a person confirming the progress.

```
stage('Sanity check') {
  steps {
    input "Does the staging environment look ok?"
  }
}
```

ADVANCED PIPELINE SETTINGS

For more information on advanced topics such as Shared Libraries, Script Blocks, Pipeline Options, or Parameterized Pipelines, please refer to the Guided Tour and Jenkins Handbook at jenkins.io/doc

Examples of Pipelines can be found on Jenkins.io or shared on GitHub (github.com/jenkinsci/pipeline-examples).

EXAMPLE PIPELINE

```
pipeline {
  agent { label “docker” }
  //Run everything on an agent with the docker daemon
  environment {
    IMAGE = readMavenPom().getArtifactId()
    //Use Pipeline Utility Steps
    VERSION = readMavenPom().getVersion()
  }
  stages {
    stage(‘Build’) {
      agent {
        docker {
          reuseNode true
          /* reuse the workspace on the agent
          defined at top-level\ */
          image ‘maven:3.5.0-jdk-8’
        }
      }
      steps {
        sh ‘mvn clean install’
        junit(allowEmptyResults: true,
          testResults: ‘**/target/surefire-reports/TEST-*.xml’)
      }
    }
    stage(‘Quality Analysis’) {
      environment {
        SONAR = credentials(‘sonar’) //use ‘sonar’ credentials
      }
      parallel {
      // run Sonar Scan and Integration tests in parallel
        stage (“Integration Test”} {
          steps {
            echo ‘Run integration tests here...’
          }
        }
        stage(“Sonar Scan”) {
          steps {
            sh “mvn sonar:sonar -Dsonar.login=$SONAR_PSW”
          }
        }
      }
    }
    stage(‘Build and Publish Image’) {
      when {
        branch ‘master’
        //only run these steps on the master branch
      }
      steps {
        sh “””
          docker build -t ${IMAGE} .
          docker tag ${IMAGE} ${IMAGE}:${VERSION}
          docker push ${IMAGE}:${VERSION}
        “””
      }
    }
  }
  post {
    failure {  // notify users when the Pipeline fails
      mail(to: ‘me@example.com’, subject: “Failed Pipeline”,
        body: “Something is wrong.”)
    }
  }
}
```

COMPLETE SYNTAX REFERENCE

pipeline (required) - contains the entire Pipeline definition

agent (required) - defines the agent used for entire pipeline or a stage

- ***any*** - use any available agent
- ***none*** - do not use a node
- ***node*** - allocate a specific executor
  - ***label*** - existing Jenkins node label for agent
  - ***customWorkspace*** - use a custom workspace directory on agent
- ***docker*** - requires docker-enabled node
  - ***image*** - run inside specified docker image
  - ***label*** - existing Jenkins node label for agent
  - ***registryUrl*** - use a private registry hosting image
  - ***registryCredentialsId*** - id of credentials to connect to registry
  - ***reuseNode*** - (Boolean) reuse the workspace and node allocated previously
  - ***args*** - arguments for docker container.
  - ***customWorkspace*** - use a custom workspace directory on agent
- ***dockerfile*** - use a local dockerfile
  - ***filename*** - name of local dockerfile
  - ***dir*** - subdirectory to use
  - ***label*** - existing Jenkins node label
  - ***reuseNode*** - (Boolean) reuse the workspace and node allocated previously
  - ***args*** - arguments for docker container
  - ***customWorkspace*** - use a custom workspace directory on agent

stages (required) - contains all stages and steps within Pipeline

stage (required) - specific named “Stage” of the Pipeline

- ***steps (required)*** - build steps that define the actions in the stage. Contains one or more of following:
  - any build step or build wrapper defined in Pipeline. e.g. ***sh***, ***bat***, ***powershell***, ***timeout***, ***retry***, ***echo***, ***archive***, ***junit***, etc.
  - ***script*** - execute Scripted Pipeline block
- ***when*** - executes stage conditionally
  - ***branch*** - stage runs when branch name matches ant pattern
  - ***expression*** - Boolean Groovy expression
  - ***anyOf*** - any of the enclosed conditions are true
  - ***allOf*** - all of the enclosed conditions are true
  - ***not*** - none of the enclosed conditions are true

- ***parallel*** -
  - ***stage*** - stages are executed in parallel
- ***agent***, ***environment***, ***tools*** and ***post*** may also optionally be defined in ***stage environment*** - a sequence of "key = value" pairs to define environment variables
- ***credentials('<id>')*** (optional) - Bind credentials to variable.

libraries - load shared libraries from an scm

- ***lib*** - the name of the shared library to load

options - options for entire Pipeline.

- ***skipDefaultCheckout*** - disable auto checkout scm
- ***timeout*** - sets timeout for entire Pipeline
- ***buildDiscarder*** - discard old builds
- ***disableConcurrentBuilds*** - disable concurrent Pipeline runs
- ***ansiColor*** - color the log file output

tools - Installs predefined tools to be available on PATH

triggers - triggers to launch Pipeline based on schedule, etc.

parameters - parameters that are prompted for at run time.

post - defines actions to be taken when ***pipeline*** or ***stage*** completes based on outcome. Conditions execute in order:

- ***always*** - run regardless of Pipeline status.
- ***changed*** - run if the result of Pipeline has changed from last run
- ***success*** - run if Pipeline is successful
- ***unstable*** - run if Pipeline result is unstable
- ***failure*** - run if the Pipeline has failed

ADDITIONAL RESOURCES

CloudBees Network
https://go.cloudbees.com

Declarative Pipeline Guided Tour
https://jenkins.io/doc

Jenkins Pipeline Documentation
jenkins.io/doc/book/pipeline

Jenkins Pipeline Syntax Reference
jenkins.io/doc/book/pipeline/syntax

Jenkins Pipeline Steps Reference
jenkins.io/doc/pipeline/steps

Blue Ocean Documentation
jenkins.io/doc/book/blueocean

Try Declarative and Blue Ocean with Docker
jenkins.io/doc/book/blueocean/getting-started/#blueocean-docker

ABOUT THE AUTHOR

PATRICK WOLF is a Director of Product Management at CloudBees, focused on Jenkins and Pipeline. He has been involved in delivering software products for more than 20 years and has experienced all phases of the evolution of software releases, from producing physical CDs and DVDs for shipment, to modern DevOps practices for Continuous Delivery. He is a regular on IRC #Jenkins to discuss all things Pipeline and frequently speaks and blogs about the latest changes coming in Jenkins.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com