

Jenkins

How Does Jenkins Work?

<https://phoenixnap.com/kb/jenkins-tutorial>

Jenkins takes the development steps from integration to deployment, automating every step of the way.

Each time a developer publishes a commit to the source code repository, Jenkins triggers a build. Typically, the commits post to a development branch.

The build steps include testing the code, ensuring the build does not break. If any error occurs, Jenkins notifies the developer to act accordingly. If all tests pass, the Pipeline proceeds to the integration steps.

Integration takes longer and requires testing the code against multiple system configurations. Jenkins performs parallel integration tests on different nodes, reducing the time needed to try and integrate code.

Further down the Pipeline, Jenkins automates user acceptance testing, which is a requirement before deployment. If all tests pass, the code merges into the main branch, making the code available to the users.

Jenkins Features

The main features of Jenkins are:

- **Easy installation.** The java-based program is autonomous and platform agnostic.
- **Straightforward to configure.** The user-friendly web interface makes configuration an easy process. For first-time setup, refer to our [Jenkins tutorial](#).
- **Customizable with massive plugin availability.** Currently, there are over 1700 available plugins to extend Jenkins' functionality. For any missing features, import additional external or custom-made plugins.
- **Open-source.** Jenkins is completely open-source and free to use, making it a cheap technology to implement.

.
.
.
.
.
.

Core Terminology

Jenkins is a platform for creating a [Continuous Integration/Continuous Delivery \(CI/CD\)](#) environment. The system offers many different tools, languages, and automation tasks to aid in pipeline creation when developing and deploying programs.

Although Jenkins requires scripting some automation steps, the program provides a fast and robust way to systematize the [software development lifecycle](#).

Definition

Jenkins is an automation server written in Java that helps build, test, and continually deploy software. The open-source system is currently one of the leading automation servers.

In general, Jenkins provides support for:

- Various version control tools such as [Git](#).
 - Building Ant and [Apache Maven](#)-based projects.
 - [Running Bash scripts](#) and Windows batch files.
-

A Short History of Jenkins

The Jenkins project started in 2004 under the name Hudson. The developer Kohsuke Kawaguchi, who worked at Sun systems, wanted to create a method to perform continuous integration. The idea was to test the code before committing to avoid breaking builds.

The idea proved successful and quickly spread to the rest of his team. As a result, Kohsuke Kawaguchi created the Jenkins project and open-sourced the program. The usage spread across the world with a current estimate of **1.6 million users**.

Jenkins Pipeline

The Jenkins Pipeline is a user-made model for the continuous delivery pipeline. The Pipeline incorporates various plugins that help define procedure steps from version control to the users.

All software changes and commits go through a complex process before the release. The method includes **three steps**:

- Automated **building**.
- Multi-step **testing**.
- **Deploying** procedures.

In Jenkins, there are **two ways to create a pipeline**:

1. Define the Pipeline through the user interface directly.
2. Use the **Pipeline as Code** methodology and create a *Jenkinsfile*. The text file **uses Groovy-compatible syntax** to define the pipeline process.

The *Jenkinsfile* syntax is either **declarative** or **scripted**.

A basic **declarative** *Jenkinsfile* pipeline is simpler to understand. An example looks like the following:

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }
  }
}
```

```

    }
}
stage('Test') {
    steps {
        echo 'Testing...'
    }
}
stage('Deploy') {
    steps {
        echo 'Deploying...'
    }
}
}
}

```

The code has the following elements:

- The mandatory `pipeline { }` block invokes the Jenkins Pipeline plugin.
- The keyword `agent` defines where the Pipeline runs, where `any` indicates the Pipeline runs on any available agent.
- The `stages { }` block represents the sequence in which the Pipeline runs.
- The code contains three stages: `Build`, `Test` and `Deploy`, each with its respective `steps { }`.
- The `steps { }` tell Jenkins what to do at a particular point.

The **scripted** equivalent of the *Jenkinsfile* looks like the following:

```

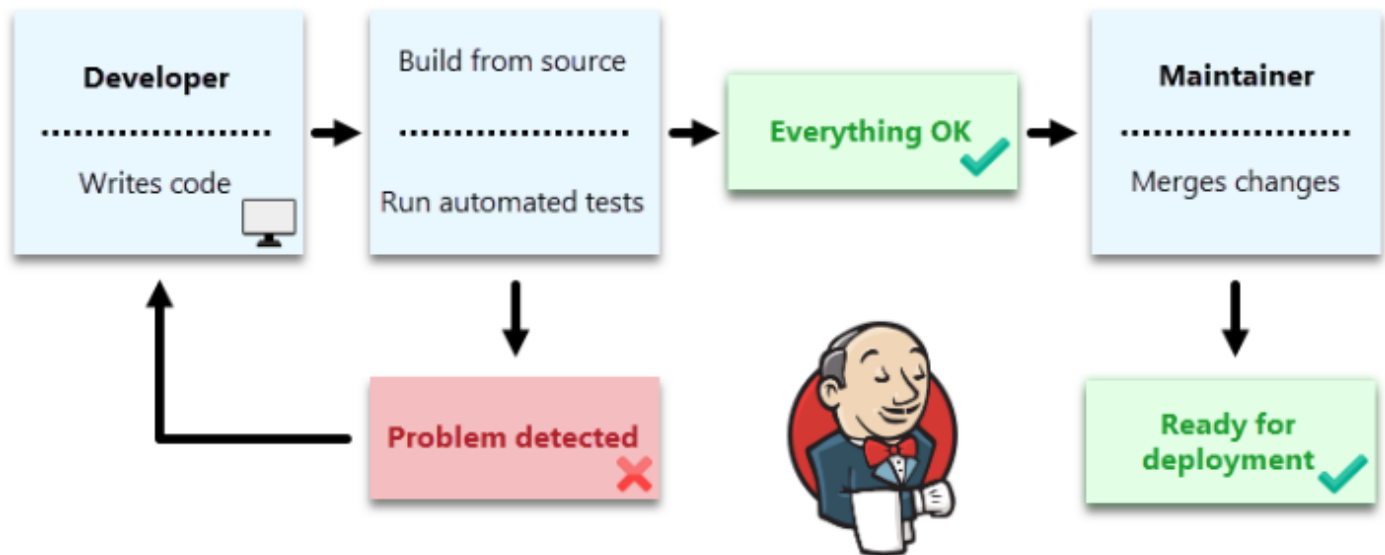
node {
    stage('Build') {
        echo 'Building...'
    }
    stage('Test') {
        echo 'Testing...'
    }
    stage('Deploy') {
        echo 'Deploying...'
    }
}
}

```

Checking a Jenkinsfile into a source control tool allows the whole team to edit, review, and adapt the steps in the delivery pipeline.

Continuous Integration

Continuous integration is a software development procedure where each applied change invokes an automated build test. The process ensures the code integrates into a working executable form without bugs.



Continuous integration is an essential aspect in environments with multiple developers. Each developer makes changes to code, and each change has potential issues.

A continuous integration tool such as Jenkins helps test, identify, and address problems before applying changes to production.

Automated Testing

[Automated testing](#) for Jenkins presets test execution and stores the results. The idea is to ensure code works in various scenarios. Creating automated tests for distinct environments, such as several [Java versions](#) or operating systems, helps foresee and prevent problems in later releases.

The automated testing phases embed into the CI pipeline in Jenkins seamlessly. Various plugins help run unit, integration, functional, and regression tests and store the results for later viewing and analysis.

Controller (Formerly Master)

The Jenkins architecture caters to distributed builds. One node is the central control unit and organizer, known as the controller.

The controller is the central process where the Jenkins configurations reside. Formerly known as the Master, the Jenkins controller manages agents and their connections, helps load plugins, and coordinates the project flow.

Agent (Formerly Slave)

The agents connect to the Jenkins controller to run projects. Agents require a Java installation on a physical or virtual machine, such as [Bare Metal Cloud](#) instances, [Docker images](#), or [Kubernetes clusters](#).

Agents in Jenkins help provide better performance by [load balancing](#) builds and creating a secure environment, separate from the controller.


Node

A node is a general term for agents and controllers, regardless of their actual role. Any machine with the ability to build projects and pipelines is a Jenkins node, and the controller is known as the **built-in node**.


The built-in node monitors the health of all attached nodes and takes them offline if any values go above a threshold.

Project (Formerly Job)


A Jenkins project, or a job, is a user-made automation procedure with a particular goal. Jenkins offers various build jobs by default, and more are available through plugins.

**Freestyle project**


This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Multibranch Pipeline**

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Below is a table with a short description of some project types:

Type	Description
Freestyle	An unrestricted build task with multiple operations.
Pipeline	A multi-step project with several build agents.
Multi-configuration	A project with multiple test environments and different configurations.
Multi-branch Pipeline	A project which creates a set of pipeline projects according to branches in a source control management system.

Build

In Jenkins, builds represent single execution of a job with the current configuration. A build creates software from different sources defined in the project procedure. Depending on the project, the build mechanisms include:

- Gathering dependencies.
- Compiling or transforming code.

- Testing.
- Deploying to different environments.

Therefore, a build is one run of a defined project with various steps.

.

.

.

.

.

Shared Libraries

[Jenkins](#) is an open-source automation server for software developers. It lets users create and manage repeatable elements that make software development and deployment more straightforward and efficient. One of these elements are Jenkins shared libraries.

As Pipeline is adopted for more and more projects in an organization, common patterns are likely to emerge. Oftentimes it is useful to share parts of Pipelines between various projects to reduce redundancies and keep code "DRY".

Pipeline has support for creating "Shared Libraries" which can be defined in external source control repositories and loaded into existing Pipelines.