

# DevOps Certification Training

Study Material - Jenkins



## What is Jenkins?

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

**Jenkins** installers are available for several Linux distributions.

- Debian/Ubuntu
- Fedora
- Red Hat / CentOS

## Prerequisites

Minimum hardware requirements:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team:

- 4 GB+ of RAM
- 50 GB+ of drive space

## Debian/Ubuntu

On Debian and Debian-based distributions like Ubuntu you can install Jenkins through apt.

Long Term Support release

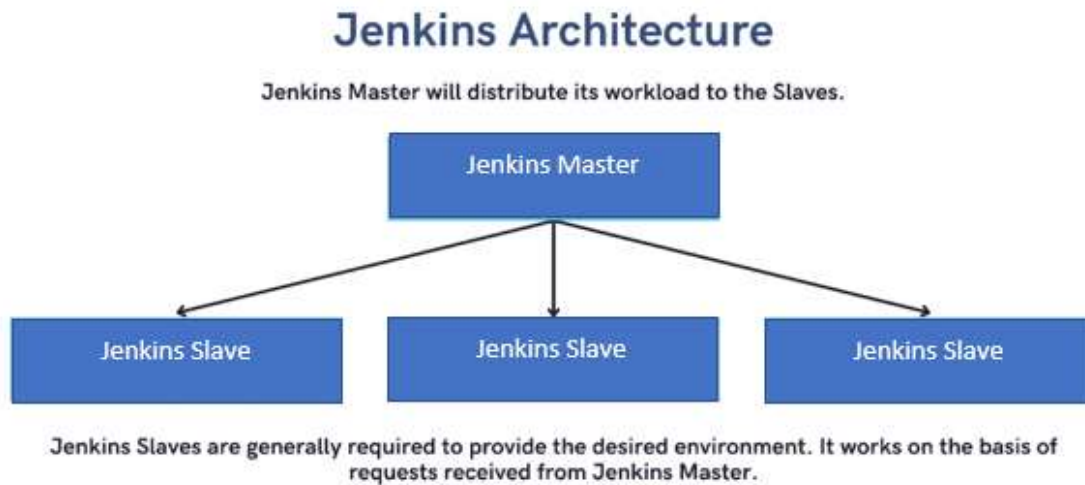
```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins
```

## Jenkins Master-Slave Architecture :



Jenkins master node basically keeps a track of all the build request and delegate the request to the workers for execution, After execution workers update the status back to the Jenkins master.

### Securing Jenkins

Credentials stored in Jenkins can be used:

- anywhere applicable throughout Jenkins (i.e. global credentials),
- by a specific Pipeline project/item (read more about this in the Handling credentials section of Using a Jenkinsfile),
- by a specific Jenkins user (as is the case for Pipeline projects created in Blue Ocean).

Jenkins can store the following types of credentials:

- **Secret text** - a token such as an API token (e.g. a GitHub personal access token),
- **Username and password** - which could be handled as separate components or as a colon separated string in the format username:password (read more about this in Handling credentials),
- **Secret file** - which is essentially secret content in a file,
- **SSH Username with private key** - an SSH public/private key pair,
- **Certificate** - a PKCS#12 certificate file and optional password, or
- **Docker Host Certificate Authentication** credentials.

### Configuring credentials

This section describes procedures for configuring credentials in Jenkins.

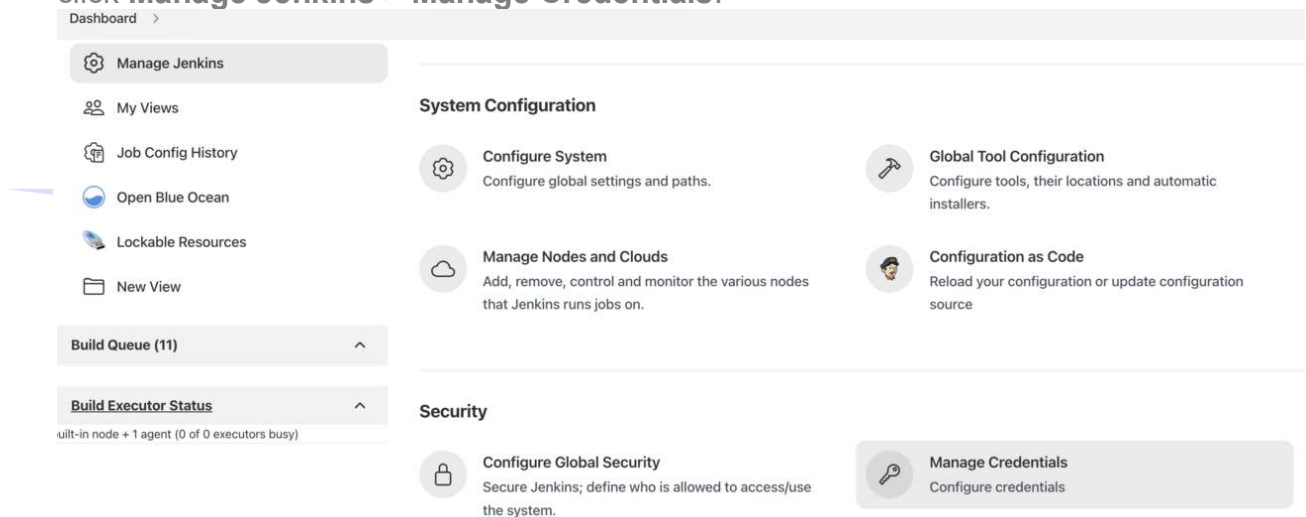
Credentials can be added to Jenkins by any Jenkins user who has the **Credentials > Create** permission (set through **Matrix-based security**). These permissions can be configured by a Jenkins user with the **Administer** permission. Read more about this in the Authorization section of Managing Security.

Otherwise, any Jenkins user can add and configure credentials if the **Authorization** settings of your Jenkins instance's **Configure Global Security** settings page is set to the default **Logged-in users can do anything** setting or **Anyone can do anything** setting.

Adding new global credentials

To add new global credentials to your Jenkins instance:

1. If required, ensure you are logged in to Jenkins (as a user with the **Credentials > Create** permission).
2. From the Jenkins home page (i.e. the Dashboard of the Jenkins classic UI), click **Manage Jenkins > Manage Credentials**.




3. Under **Stores scoped to Jenkins** on the right, click on **Jenkins**.

## Stores scoped to Jenkins



- Under **System**, click the **Global credentials (unrestricted)** link to access this default domain.

### System

Domain ↓	Description
 Global credentials (unrestricted) ▼	Credentials that should be available irrespective of domain specification to requirements matching.

Icon: S M L

- Click **Add Credentials** on the left.  
**Note:** If there are no credentials in this default domain, you could also click the **add some credentials** link (which is the same as clicking the **Add Credentials** link).
- From the **Kind** field, choose the type of credentials to add.
- From the **Scope** field, choose either:
  - Global** - if the credential/s to be added is/are for a Pipeline project/item. Choosing this option applies the scope of the credential/s to the Pipeline project/item "object" and all its descendant objects.
  - System** - if the credential/s to be added is/are for the Jenkins instance itself to interact with system administration functions, such as email authentication, agent connection, etc. Choosing this option applies the scope of the credential/s to a single object only.
- Add the credentials themselves into the appropriate fields for your chosen credential type:
  - Secret text** - copy the secret text and paste it into the **Secret** field.
  - Username and password** - specify the credential's **Username** and **Password** in their respective fields.
  - Secret file** - click the **Choose file** button next to the **File** field to select the secret file to upload to Jenkins.
  - SSH Username with private key** - specify the credentials **Username**, **Private Key** and optional **Passphrase** into their respective fields.  
**Note:** Choosing **Enter directly** allows you to copy the private key's text and paste it into the resulting **Key** text box.
  - Certificate** - specify the **Certificate** and optional **Password**. Choosing **Upload PKCS#12 certificate** allows you to upload the certificate as a file via the resulting **Upload certificate** button.
  - Docker Host Certificate Authentication** - copy and paste the appropriate details into the **Client Key**, **Client Certificate** and **Server CA Certificate** fields.
- In the **ID** field, specify a meaningful credential ID value - for example, jenkins-user-for-xyz-artifact-repository. The inbuilt (default) credentials provider can use uppercase or lowercase letters for the credential ID, as well as any valid separator character, other credential providers may apply further restrictions on

allowed characters or lengths. However, for the benefit of all users on your Jenkins instance, it is best to use a single and consistent convention for specifying credential IDs.

**Note:** This field is optional. If you do not specify its value, Jenkins assigns a globally unique ID (GUID) value for the credential ID. Bear in mind that once a credential ID is set, it can no longer be changed.

10. Specify an optional **Description** for the credential/s.

11. Click **OK** to save the credentials.

## 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. [1]

The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. [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 Jenkins file 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] 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 Jenkins file is the same, it is generally considered best practice to define the Pipeline in a Jenkins file and check that in to source control.

## Declarative versus Scripted Pipeline syntax

A Jenkins file 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.

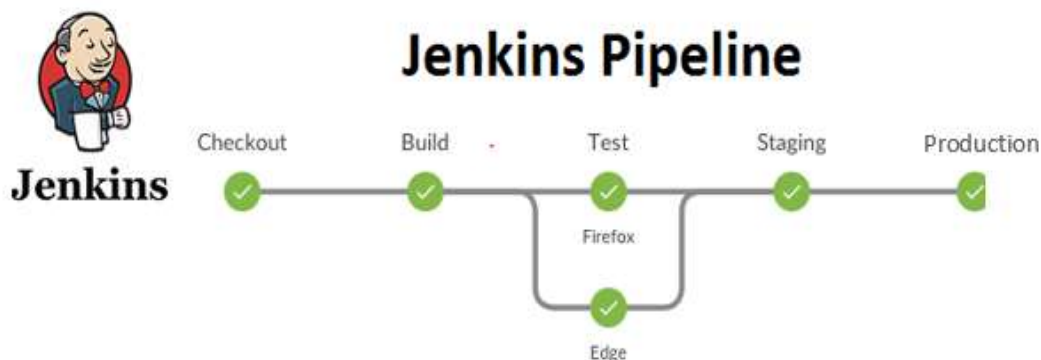
Many of the individual syntactical components (or "steps") written into a Jenkins file, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in Pipeline concepts and Pipeline syntax overview below.

## 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 controller.
- **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] 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] Pipeline makes this concept a first-class citizen in Jenkins.



## What is the difference between Freestyle and Pipeline in Jenkins

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with Pipeline Shared Libraries and by plugin developers. [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 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.

#### **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.

#### **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]

#### **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] 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 and Scripted Pipeline syntax.

Be aware that both stages and steps (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 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'
    }
  }
}
}
}

```

### Toggle Scripted Pipeline (*Advanced*)

- pipeline is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.
- agent is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.
- stage is a syntax block that describes a stage of this Pipeline. Read more about stage blocks in Declarative Pipeline syntax on the Pipeline syntax page. As mentioned above, stage blocks are optional in Scripted Pipeline syntax.
- steps is Declarative Pipeline-specific syntax that describes the steps to be run in this stage.
- sh is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.
- junit is another Pipeline step (provided by the JUnit plugin) for aggregating test reports.
- sh is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.