**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.

**Managing Jenkins**

Most standard administrative tasks can be performed from the screens in the **Manage Jenkins** section of the dashboard. In this chapter, we look at these screens and explain how to use them.

The tiles displayed on the **Manage Jenkins** page are grouped logically. Here we discuss the pages that are part of the standard installation. Plugins may add pages to this screen.

The top of the **Manage Jenkins** screen may contain "Monitors" that alert you when a new version of the Jenkins software or a security update is available. Each monitor includes a description of the issue it is reporting and links to additional information about the issue

Inline help is available on most **Manage Jenkins** pages:

* To access the help, select the ? icon to the right of each field.
* Click the ? icon again to hide the help text.

## System Configuration group

Screens for configuring resources for your Jenkins instance.

[**System**](https://www.jenkins.io/doc/book/managing/system-configuration)

Configure global settings and paths for the Jenkins instance

[**Tools**](https://www.jenkins.io/doc/book/managing/tools)

Configure tools, their locations, and automatic installers

[**Plugins**](https://www.jenkins.io/doc/book/managing/plugins)

Add, update, remove, disable/enable plugins that extend the functionality of Jenkins.

[**Nodes and Clouds**](https://www.jenkins.io/doc/book/managing/nodes)

Add, remove, control, and monitor the nodes used for the agents on which build jobs run.

[**Configuration as Code**](https://www.jenkins.io/doc/book/managing/casc)

Configure your Jenkins instance using a human-readable YAML file rather than the UI. This is an optional feature that appears in this group only when the plugin is installed on your controller.

## Security group

Screens for configuring security features for your Jenkins instance. See [Securing Jenkins](https://www.jenkins.io/doc/book/security/) for general information about managing Jenkins security.

[**Security**](https://www.jenkins.io/doc/book/managing/system-configuration)

Set configuration parameters that secure your Jenkins instance.

[**Manage Credentials**](https://www.jenkins.io/doc/book/using/using-credentials/#adding-new-global-credentials)

Configure the credentials that provide secure access to third-party sites and applications that interact with Jenkins.

**Credential Providers**

Configure credential providers and types

[**Users**](https://www.jenkins.io/doc/book/managing/users)

Manage users defined in the Jenkins user database. This is not used if you use a different security realm such as LDAP or AD.

## Status Information group

[**System Information**](https://www.jenkins.io/doc/book/managing/system-info)

Displays information about the Jenkins environment.

[**System Log**](https://www.jenkins.io/doc/book/system-administration/viewing-logs/)

Jenkins log that contains all java.util.logging output related to Jenkins.

**Load Statistics**

Displays information about resource utilization on you Jenkins instance.

[**About Jenkins**](https://www.jenkins.io/doc/book/managing/about-jenkins)

Provides version and license information for your Jenkins instance.

**File Information in Jenkins home**

+- builds (build records)

+- [BUILD\_ID] (subdirectory for each build)

+- build.xml (build result summary)

+- changelog.xml (change log)

+- config.xml (Jenkins root configuration file)

+- \*.xml (other site-wide configuration files)

+- fingerprints (stores fingerprint records, if any)

+- identity.key.enc (RSA key pair that identifies an instance)

+- jobs (root directory for all Jenkins jobs)

+- [JOBNAME] (sub directory for each job)

+- config.xml (job configuration file)

+- [FOLDERNAME] (sub directory for each folder)

+- config.xml (folder configuration file)

+- jobs (subdirectory for all nested jobs)

+- plugins (root directory for all Jenkins plugins)

+- [PLUGIN] (sub directory for each plugin)

+- [PLUGIN].jpi (.jpi or .hpi file for the plugin)

+- secret.key (deprecated key used for some plugins' secure operations)

+- secret.key.not-so-secret (used for validating \_$JENKINS\_HOME\_ creation date)

+- secrets (root directory for the secret+key for credential decryption)

+- hudson.util.Secret (used for encrypting some Jenkins data)

+- master.key (used for encrypting the hudson.util.Secret key)

+- InstanceIdentity.KEY (used to identity this instance)

+- userContent (files served under your https://server/userContent/)

+- workspace (working directory for the version control system)

# Securing Jenkins

Jenkins is used everywhere from workstations on corporate intranets, to high-powered servers connected to the public internet. To safely support this wide spread of security and threat profiles, Jenkins offers many configuration options for enabling, customizing, or disabling various security features.

Many of the security options are enabled by default when passing the interactive setup wizard to ensure that Jenkins is secure. Others involve environment-specific setup and trade-offs and depend on specific use cases supported in individual Jenkins instances.

This chapter will introduce the various security options available to Jenkins administrators and users, explaining the protections offered, and trade-offs to disabling some of them.

## Basic Setup

[**Controller Isolation**](https://www.jenkins.io/doc/book/security/controller-isolation)

Builds should not be executed on the built-in node, but that is just the beginning: This section discusses what other steps can be taken to protect the controller from being impacted by running builds.  
**This needs to be configured according to the needs of your environment.**

[**Access Control**](https://www.jenkins.io/doc/book/security/access-control)

By default, Jenkins does not allow anonymous access, and a single admin user exists. This chapter discusses which level of access is provided by permissions and how to safely grant access to more users.  
This is set up securely by the setup wizard. If the setup wizard is disabled on first launch, this may not be configured securely by default.

Jenkins Pipeline

Pipeline scripts are another way of job configuration with the help of groovy code.

A Pipeline’s code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

**Advantages:**

* We can divide the jobs into parts like build, deploy and test and we can schedule each stage or part on separate agent
* Parallel execution of stages is easy to configure.
* We can use different versions of tolls in single pipeline like JDK, maven and etc.
* We can check the console and verify the job status.

**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](https://www.jenkins.io/doc/book/pipeline/#_footnotedef_1)] and multiple options for integration with other plugins.

Types of Pipelines:

* **Scripted pipeline**
* **Declarative pipeline**

**Scripted pipeline:**

These are the old and traditional way of job configuration we have limited options here compared to declarative pipeline.The scripted pipeline will start with node. Below is the syntax of sample scripted pipeline.

node{

stage('Preparation') {

echo “hello world”

}

}

**Declarative pipeline:**

It’s recent addition of pipelines here we have many simplified and opinionated syntax when compared to scripted pipeline.

The declarative pipeline will start with pipeline. Below is the syntax of sample declarative pipeline.

pipeline {

agent any

stages {

stage('build'){

steps {

cleanWs()

echo ‘Hello World’

}

}

}

**Pipeline** is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.

[**agent**](https://www.jenkins.io/doc/book/pipeline/syntax#agent) is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.

**Stages** is collection of stage.

**Stage** is a syntax block that describes a [stage of this Pipeline](https://www.jenkins.io/doc/book/pipeline/#stage). stage blocks are optional in Scripted Pipeline syntax.

[**Steps**](https://www.jenkins.io/doc/book/pipeline/syntax#steps) is Declarative Pipeline-specific syntax that describes the steps to be run in this stage.