

DevOps Foundations: Version Control and CI/CD with Jenkins



Jenkins Job and Plugins



Learning Objectives

By the end of this lesson, you will be able to:

- Configure Jenkins jobs effectively to automate the building, testing, and deploying applications at different stages of the development
- Employ Jenkins plugins for automating tasks and integrating with other tools in the software development pipeline
- Outline the key architectural components within Jenkins for robust and scalable CI/CD automation
- Identify and utilize Jenkins freestyle job sections efficiently to implement CI/CD practices and streamline software development processes

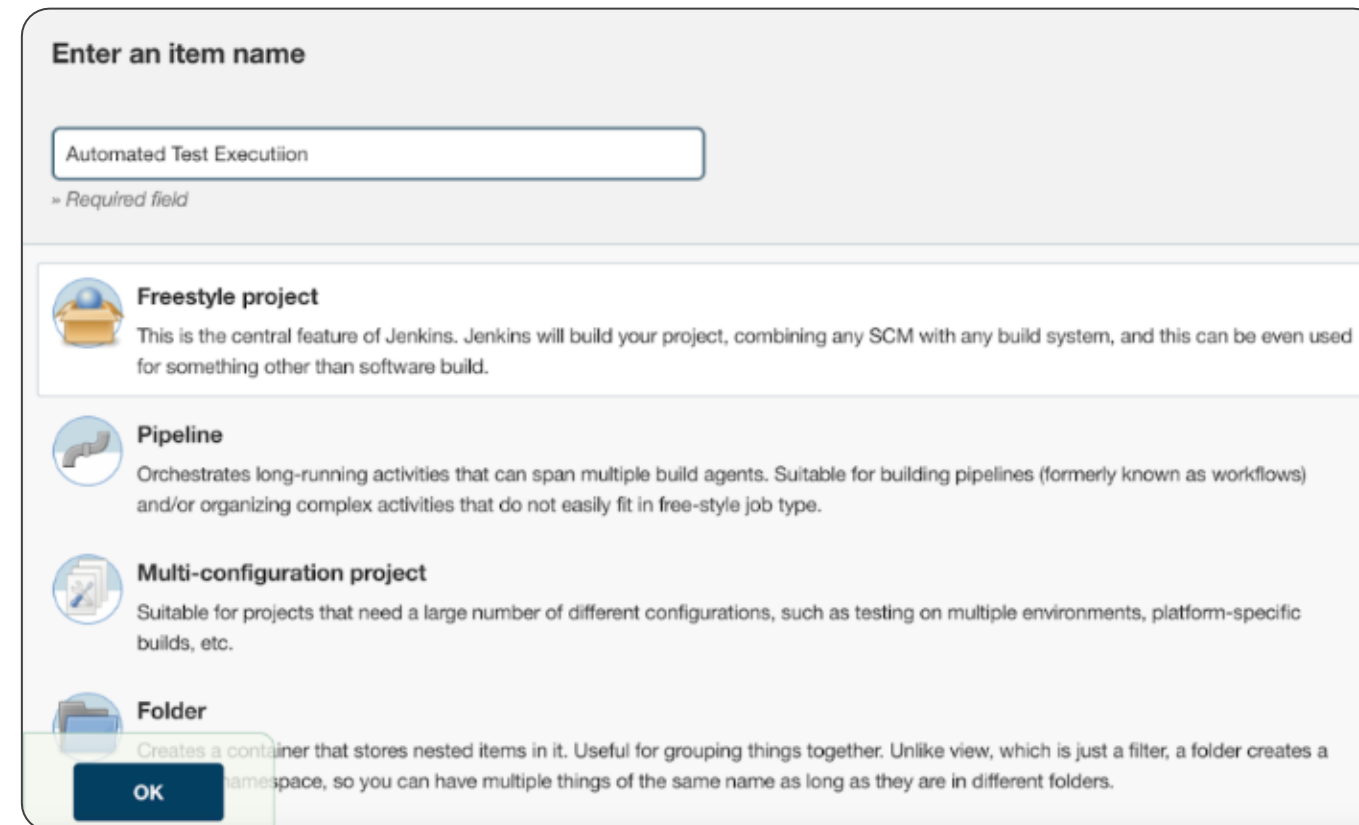




Jenkins Job Configuration

Introduction to Jenkins Job

It is a predefined set of instructions or tasks within Jenkins to automate software tasks like building and deploying applications.



The screenshot shows the 'Enter an item name' dialog in Jenkins. At the top, there is a text input field containing 'Automated Test Executiion' and a small red asterisk icon with the text '» Required field' below it. Below the input field, there are four job type options, each with an icon and a description:

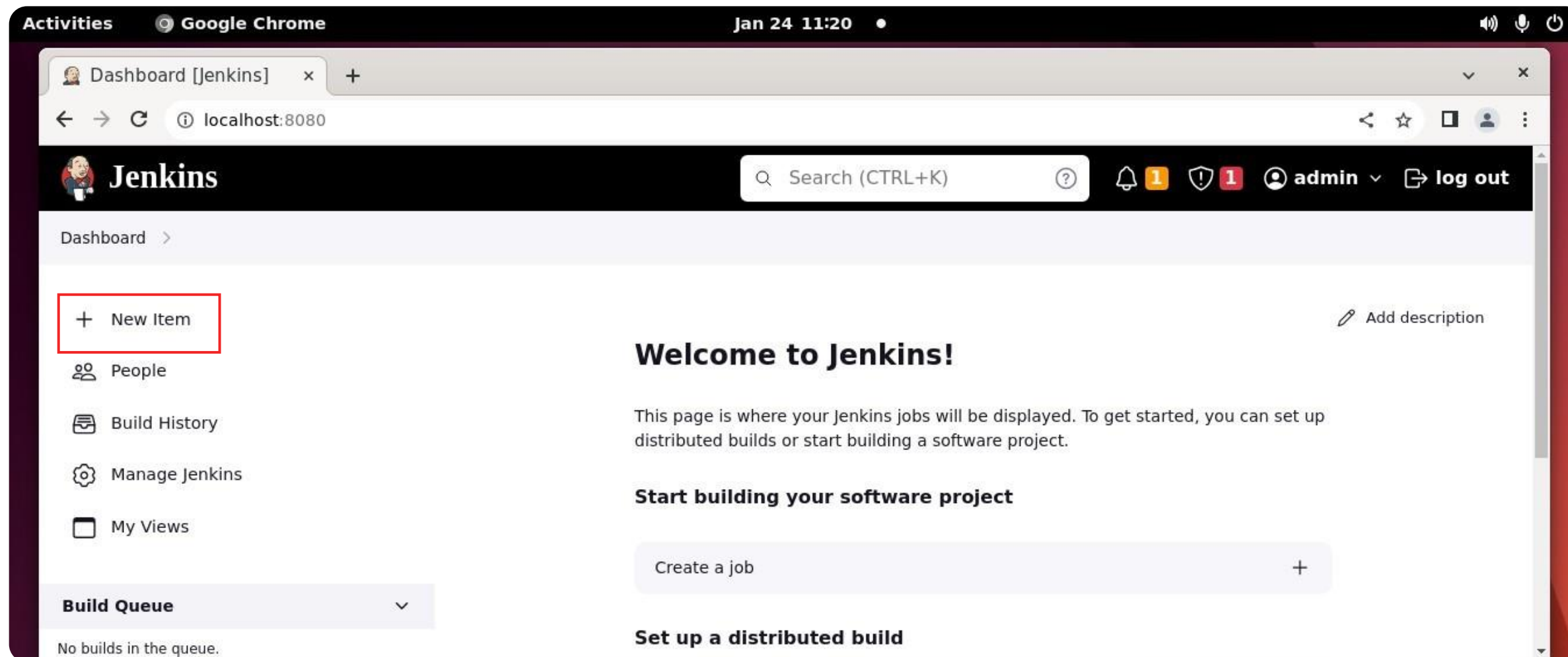
- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable 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.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a namespace, so you can have multiple things of the same name as long as they are in different folders.

At the bottom left of the dialog, there is a blue button labeled 'OK'.

They ensure consistent execution based on configured triggers and parameters.

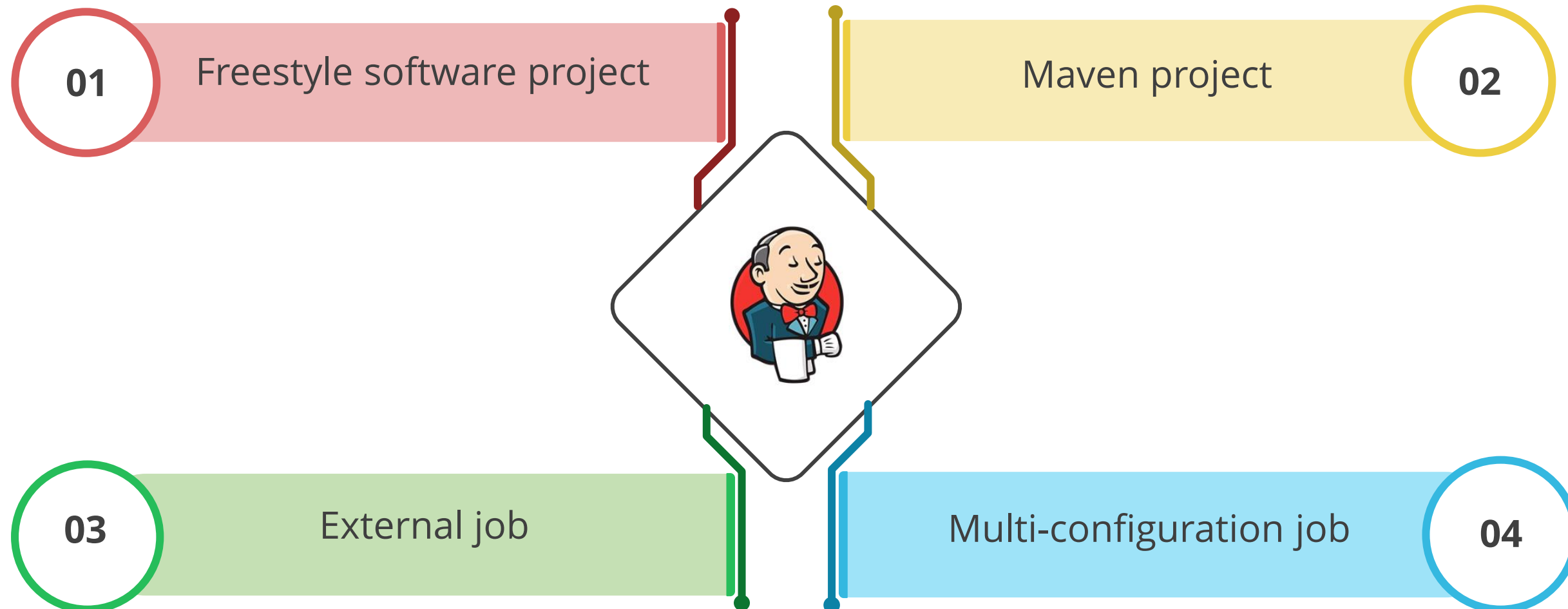
Introduction to Jenkins Job

A new Jenkins build job can be created by clicking on the **New Item** button in the Jenkins dashboard. This action lists the available build jobs in Jenkins to choose from.



Types of Jenkins Job

Following are the four types of build jobs that Jenkins supports:



Types of Jenkins Job

Freestyle software project

- General-purpose build jobs are offered by Freestyle build jobs.
- Maximum flexibility is provided by them.

Maven project

- This job type simplifies builds for projects that utilize the Maven build tool.
- Jenkins interprets the Maven POM files and project structures to create a basic setup for the project.

Types of Jenkins Job

External job

- The external job option helps to monitor non-interactive processes like Cron jobs.
- It allows for integrating Jenkins with external systems and processes seamlessly.

Multi-configuration job

- The multi-configuration job is also known as the matrix job.
- It runs the same build job with different configurations.
- It is used for testing an application in different environments with different databases or on different build machines.

Job Import Plugin

It is a valuable tool in Jenkins that allows you to migrate jobs from one Jenkins server to another.

Purpose:

- Simplifies the process of copying existing Jenkins jobs (configurations and build history) from a source server to a target server
- Eliminates the need to manually recreate complex job configurations on the target server, saving time and effort

Job Import Plugin

Following are the benefits of using job import plugin:

Facilitates efficient migration of Jenkins jobs between servers during upgrades, disaster recovery, or setting up new environments

Eliminates the possibility of errors introduced by manual configuration recreation

Ensures identical job configurations on both servers, promoting consistency in your CI/CD pipeline

Job Import Plugin

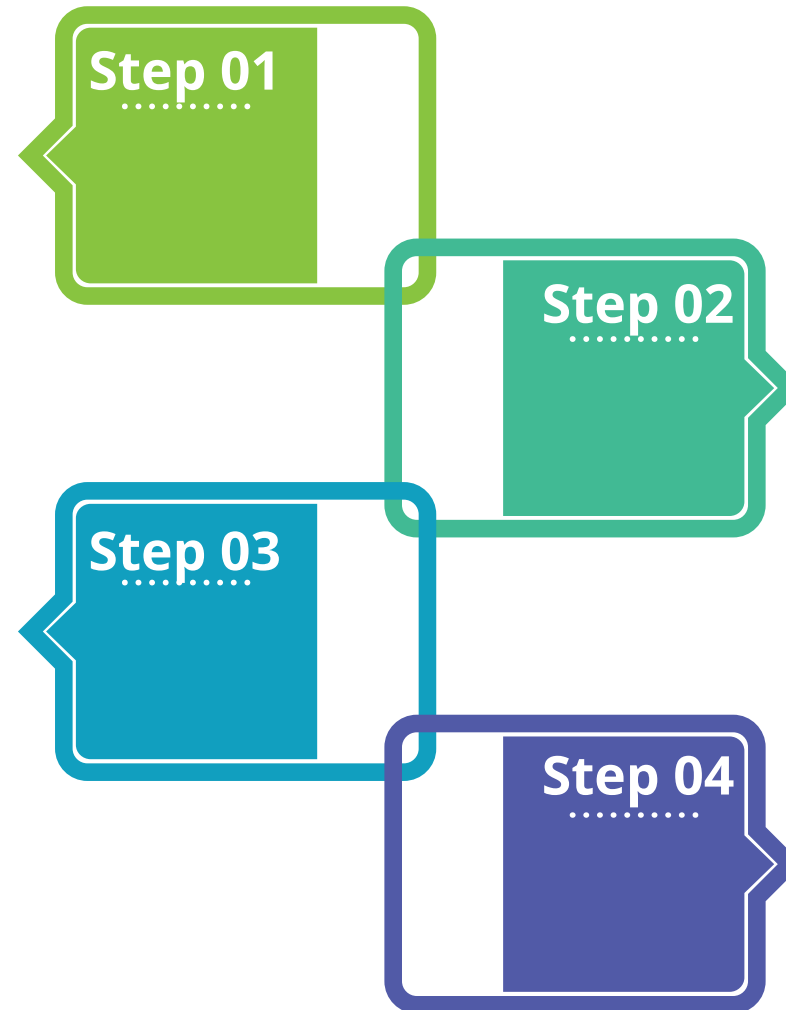
Below is the workflow for installing the job import plugin in Jenkins:

Installation:

Install the job import plugin on both the source and target Jenkins servers

Job selection:

Fetch the list of jobs from the source server and allow users to choose which ones to import



Configuration:

Provide the URL and credentials on the target server to access the source server

Importation:

Import the selected jobs to the target server including their settings and possibly their history

Assisted Practice



Implementing job import plugin

Duration: 10 Min.

Problem statement:

You have been assigned a task to install the job import plugin for importing jobs from an existing Jenkins instance, streamlining job management, and enhancing productivity.

Outcome:

By completing this demo, you will be able to install, configure, and access the job import plugin in Jenkins to streamline job management and enhance productivity.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Install the job import plugin in Jenkins
2. Configure the job import plugin
3. Access the job import plugin



Jenkins Plugin

Introduction to Jenkins Plugin

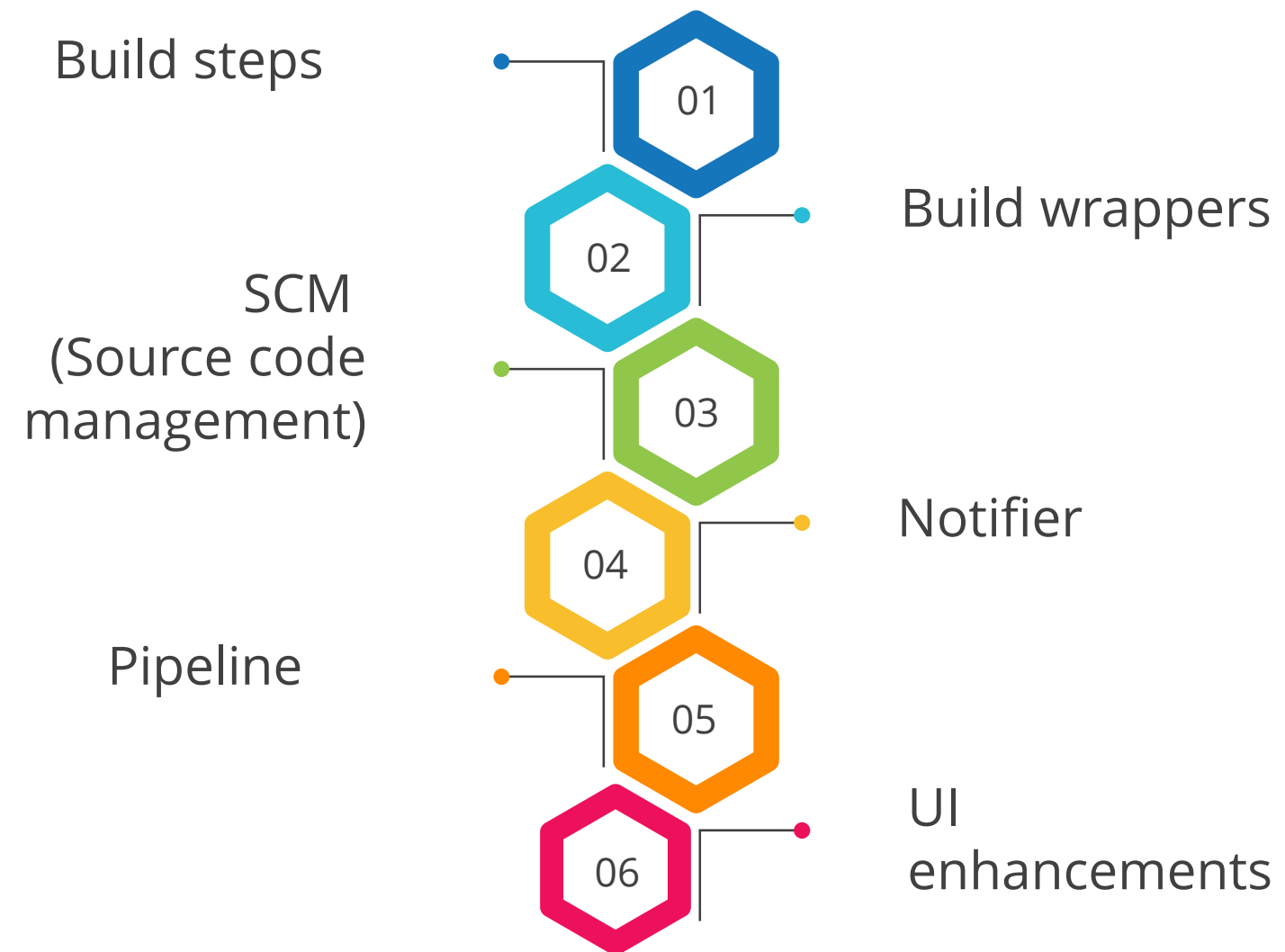
Jenkins plugins expand the core functionality by integrating with tools such as Git, Maven, and Docker, amplifying automation capabilities in software development.



These components enrich Jenkins, enabling seamless integration with various technologies and enhancing efficiency in software delivery workflows.

Types of Jenkins Plugins

Plugins can be categorized into different types based on their functionality:



Types of Jenkins Plugins

Build steps:

Provide specific actions to be executed during a build process

Build wrappers:

Control the environment or settings for a build job

SCM (Source code management):

Integrate with version control systems to manage source code

Notifier:

Send notifications or alerts based on build results

Pipeline:

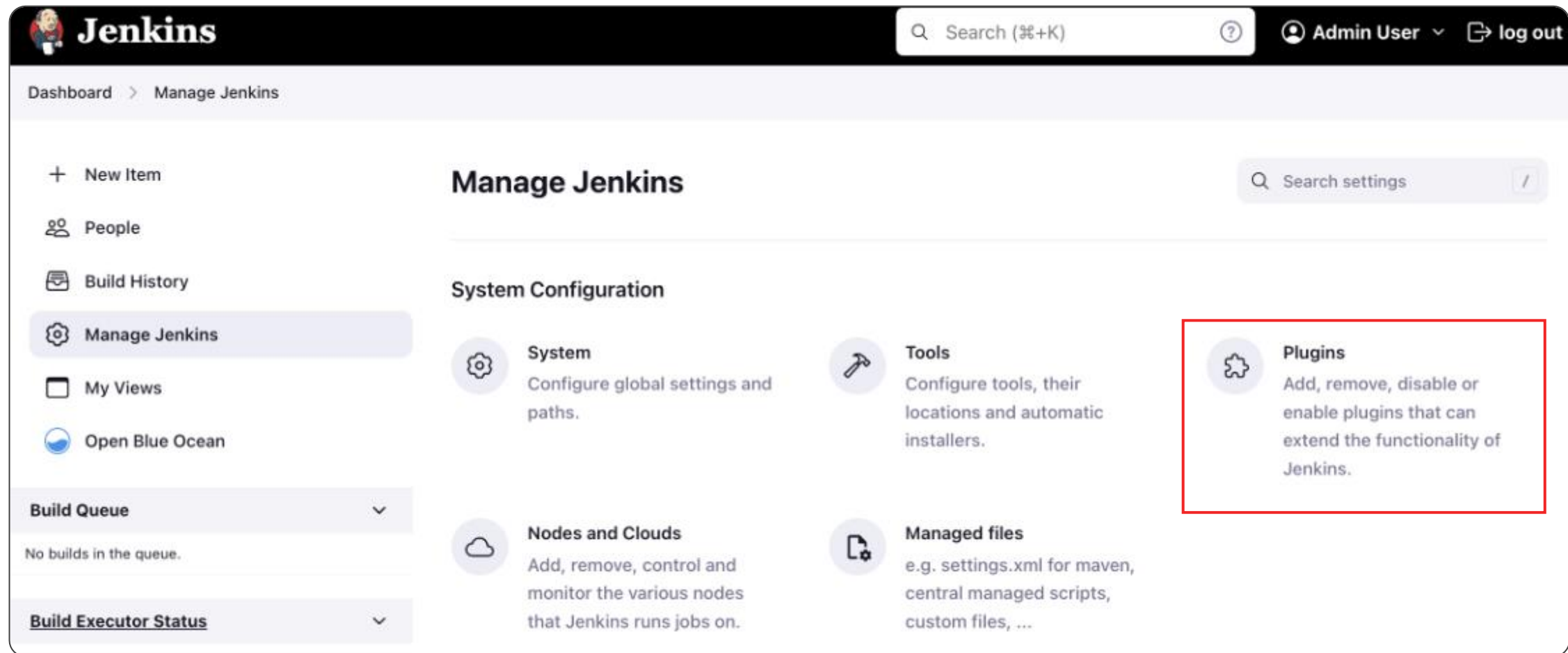
Extend Jenkins pipeline functionality for continuous delivery

UI enhancements:

Improve the Jenkins user interface with additional features

Managing Jenkins Plugins

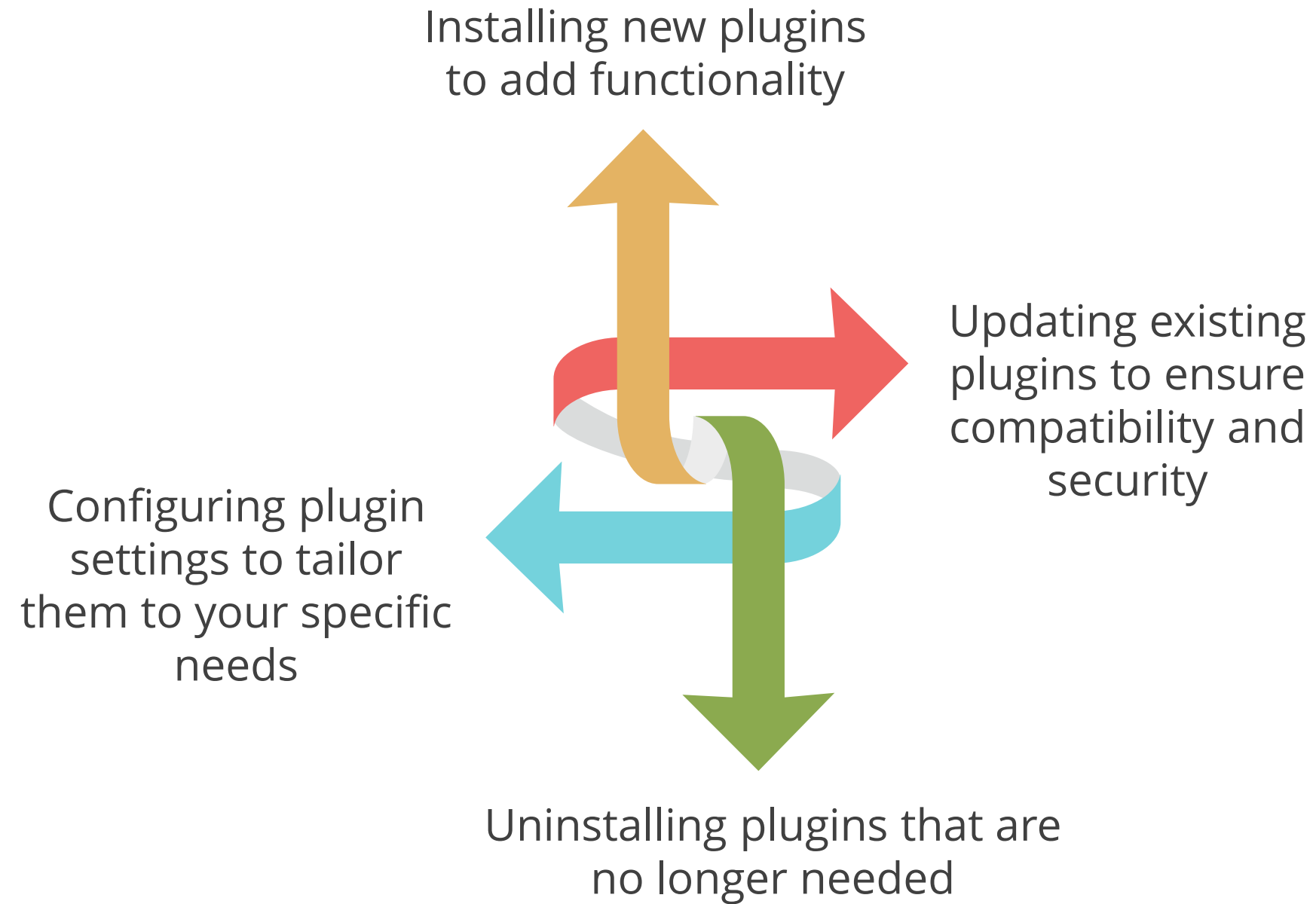
Plugins are like building blocks that allow to tailor Jenkins according to the specific needs within a CI/CD pipeline.



Effective management of Jenkins plugins is essential for maintaining a stable and efficient continuous integration and delivery (CI/CD) pipeline.

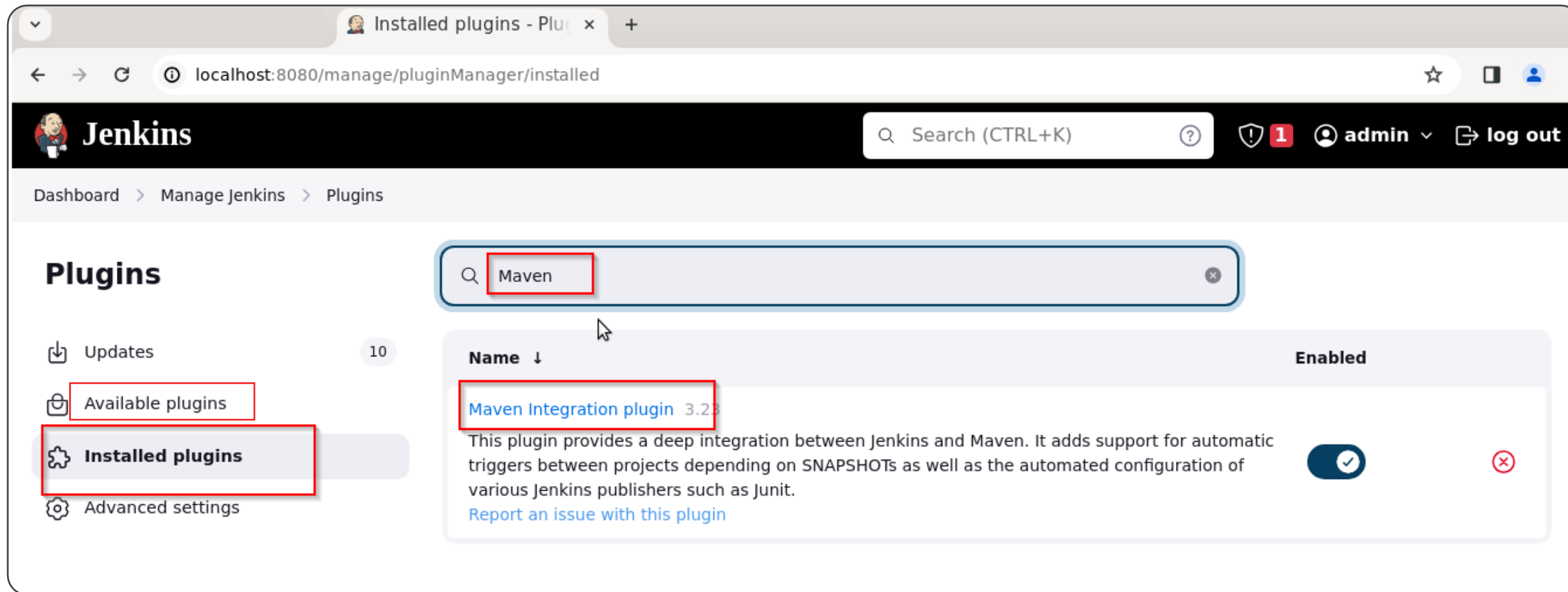
Managing Jenkins Plugins

There are four main tasks for managing Jenkins plugins:



Installing New Jenkins Plugins

It is the process of adding a new functionality to your Jenkins server.



These plugins extend Jenkins capabilities, adapting it for specific requirements in a CI/CD pipeline.

Installing New Jenkins Plugins

There are two main ways to install a Jenkins plugin:

1. Using the Jenkins plugin manager

The Jenkins plugin manager simplifies tasks with its user-friendly interface

- Search and browse
- Review information
- Install with a click

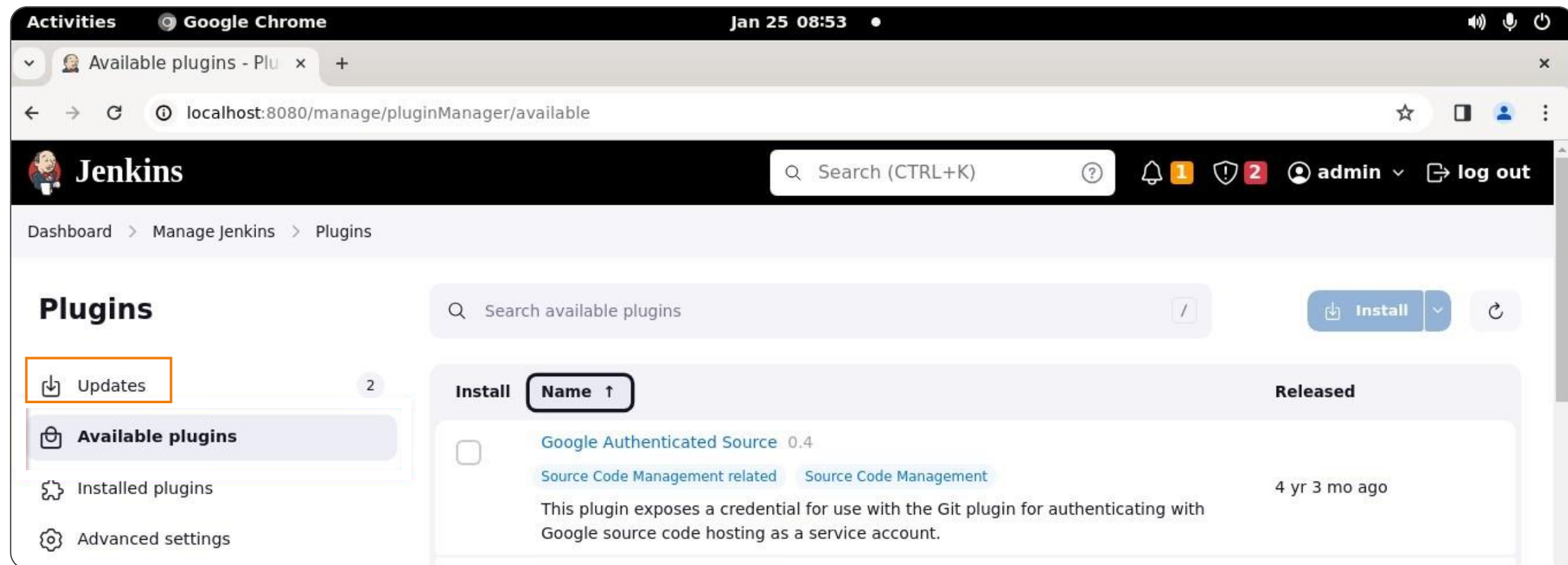
2. Manually downloading and uploading

This approach is for experts or when the plugin is not accessible via the manager

- Finding the plugin file
- Uploading within Jenkins

Updating Jenkins Plugins

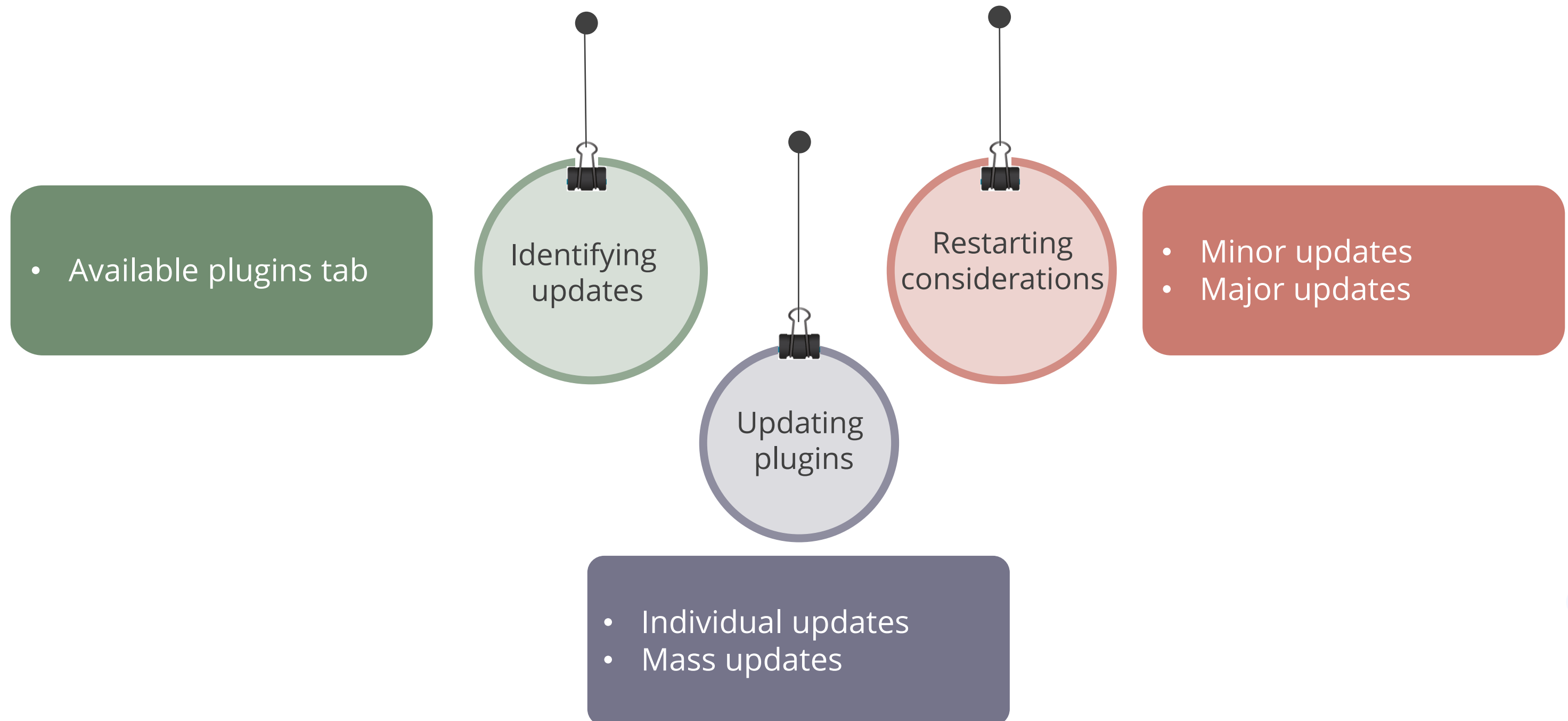
It is crucial to ensure updating Jenkins plugin where CI/CD pipeline remains secure, compatible, and benefits from bug fixes and new features.



The image shows the count of the latest updates available in the **Updates** section.

Updating Jenkins Plugins

Below are ways of updating the Jenkins plugins:



Updating Jenkins Plugins

1. Identifying updates:

Available plugins tab:

- Lists all available plugins on the **Available plugins** tab
- Highlights updates for installed plugins with an exclamation mark (yellow)
- Enables efficient identification and management of plugin updates

Updating Jenkins Plugins

2. Updating plugins:

Individual updates:

- Select the checkbox next to the desired update in the available tab to install specific plugins without having to restart for minor updates

Mass updates:

- Navigate to the updates tab in the plugin manager
- Select the desired updates
- Choose to install all outdated plugins at once without having to restart

Updating Jenkins Plugins

3. Restarting considerations:

Minor updates (No restart needed):

- Click on **Install without restart** for minor updates in Jenkins
- Keep Jenkins running smoothly without interruption

Major updates (Require restart):

- Opt for download and install after restart for major updates
- Ensure changes are fully implemented after system restart

Updating Jenkins Plugins

Benefits of keeping plugins updated:

Security: Address security vulnerabilities discovered in older versions of plugins

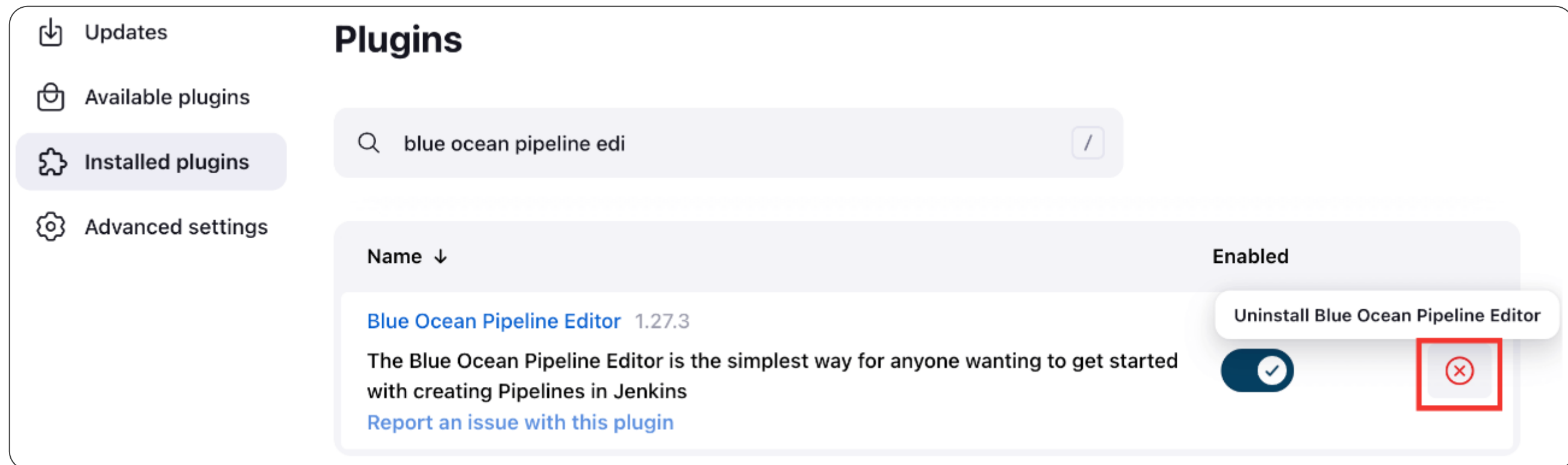
Compatibility: Ensure compatibility with newer versions of Jenkins

Bug fixes: Resolve bugs that might be impacting the functionality of the plugin

New features: Introduce new features and functionalities to the plugin

Uninstalling Jenkins Plugins

It is the process of removing a software extension from the Jenkins server. These plugins add functionalities specific to the CI/CD pipeline needs.



The above screenshot depicts the process of uninstalling the **Blue Ocean Pipeline Editor** plugin from Jenkins.

Uninstalling Jenkins Plugins

Below are the reasons, considerations, and approaches to plugin management:

Reasons to uninstall plugins:

- Remove unsupported or unmaintained plugins
- Eliminate plugins not needed for your CI/CD pipeline
- Resolve conflicts caused by plugins or Jenkins installation

Considerations before uninstalling:

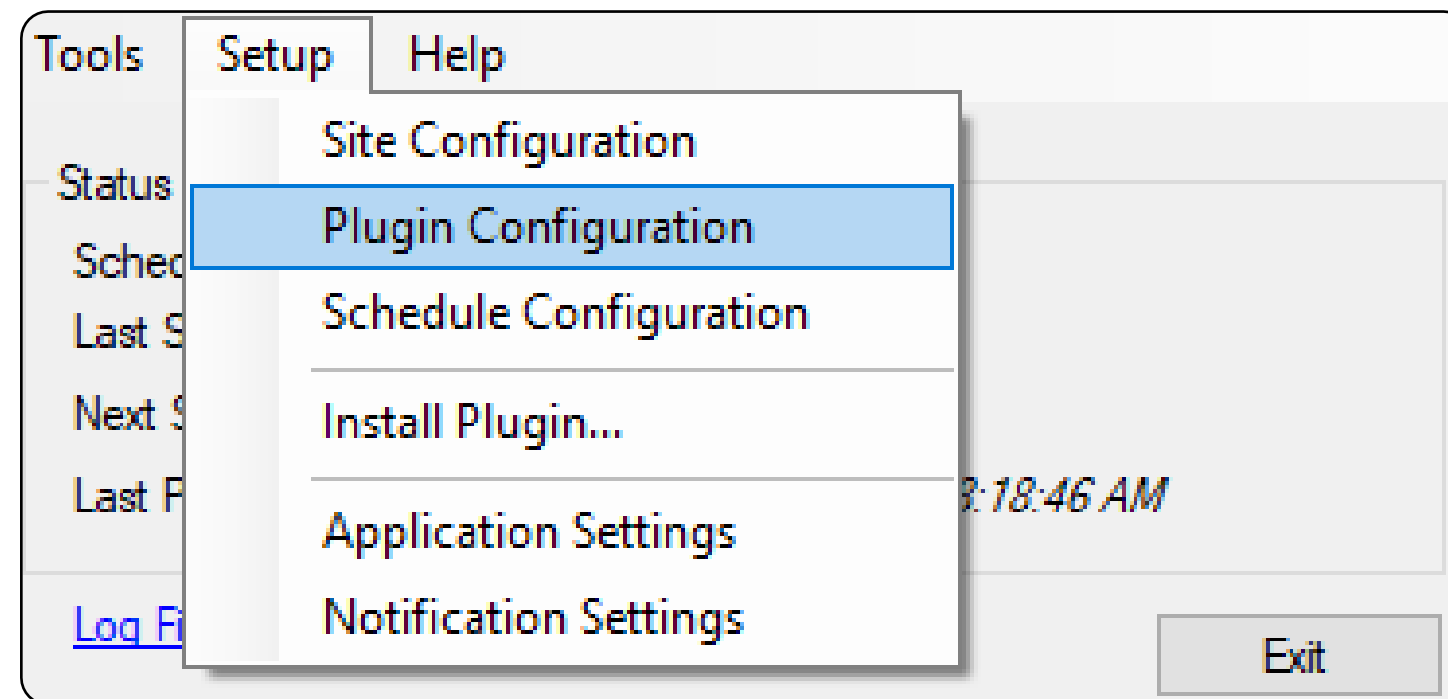
- Assess impacts on jobs or configurations
- Seek alternative plugins
- Backup configurations

Alternative plugin management approaches:

- Disable instead of uninstalling plugins
- Upgrade for issue resolution or new features
- Preserve plugins while preventing execution

Configuring Plugin Settings

It refers to the process of tailoring the behavior of installed plugins to fit specific needs within a CI/CD pipeline.

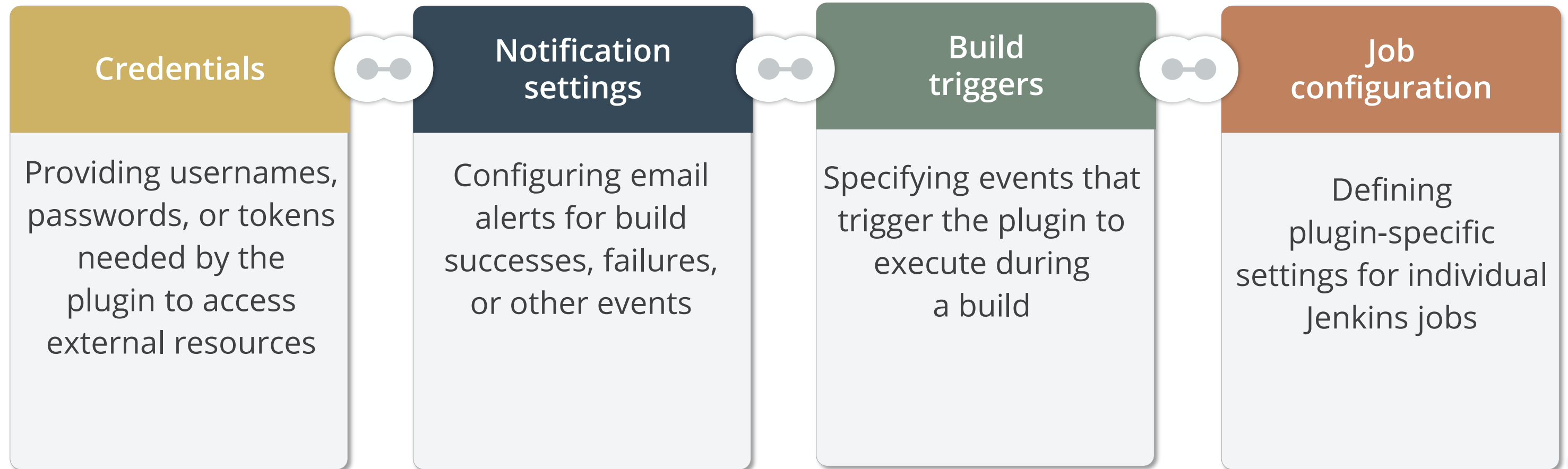


Configure plugin settings by accessing the **Plugin Manager**, selecting the installed plugin, and adjusting parameters or options according to project requirements.

Configuring Plugin Settings

The specific options in configuring plugin will vary depending on the plugin.

Common configurations include:



Assisted Practice



Uninstalling unused Jenkins plugin

Duration: 10 Min.

Problem statement:

You have been assigned a task to uninstall an unused Jenkins plugin for ensuring the smooth functioning of a Jenkins CI/CD pipeline.

Outcome:

By completing this demo, you will be able to uninstall an unused Jenkins plugin to ensure the smooth functioning of a Jenkins CI/CD pipeline.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Log in to Jenkins CI tool
2. Uninstall the Maven Integration plugin

Assisted Practice



Updating Jenkins plugin manager configuration

Duration: 10 Min.

Problem statement:

You have been assigned a task to configure Jenkins plugin manager to fetch updates for enhanced functionality and security.

Outcome:

By completing this demo, you will be able to configure the Jenkins plugin manager to fetch updates, enhancing functionality and security.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Log in to Jenkins CI tool
2. Configure plugin manager update website

Quick Check



Your Jenkins server has several plugins installed, but some are outdated. What are the two primary methods for managing plugin updates?

- A. Manually editing configuration files and restarting Jenkins
- B. Using the built-in Plugin Manager to update or uninstall plugins
- C. Downloading plugin updates from external websites and uploading them
- D. Reinstalling Jenkins with the latest version to get updated plugins



Getting Started with Jenkins Freestyle Job

Jenkins Freestyle Job

It is a versatile job type allowing users to configure build steps, triggers, and post-build actions without using predefined pipeline structures.

Below is the image of **Freestyle project** from Jenkins dashboard:



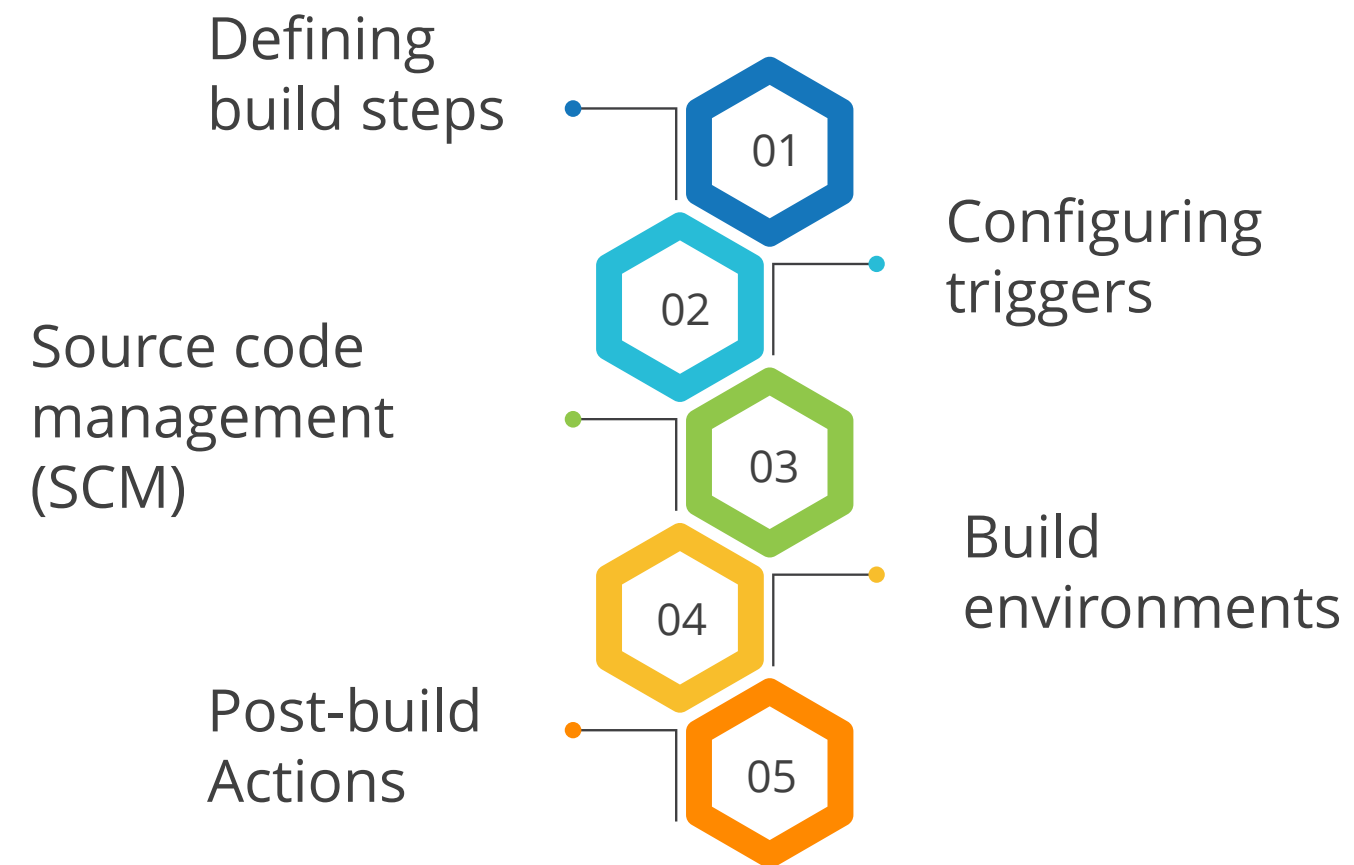
Freestyle project

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.

They offer customizable builds and plugin integrations through a user-friendly graphical interface.

Basic Job Structure

The basic job structure in Jenkins includes:



Basic Job Structure

Defining build steps

Executes instructions during the build process, such as compiling code, running tests, or creating artifact

Configuring triggers

Initiates events or conditions that commence a build, such as code commits, scheduled times, or manual triggers

Source code management (SCM)

Manages and retrieves source code from repositories like Git or SVN for building and testing

Basic Job Structure

Build environments

Defines the execution environment for build steps, including tools, dependencies, and configurations

Post-build Actions

Performs actions after a build, like archiving artifacts, sending notifications, or triggering follow-up jobs

Assisted Practice



Managing Jenkins freestyle jobs

Duration: 20 Min.

Problem statement:

You have been assigned a task to manage Jenkins job by creating, renaming, and deleting Freestyle job, facilitating streamlined job configuration and maintenance within the Jenkins environment.

Outcome:

By completing this demo, you will be able to manage Jenkins jobs by creating, renaming, and deleting a Freestyle job, facilitating streamlined job configuration and maintenance within the Jenkins environment.

Note: Refer to the demo document for detailed steps

An isometric illustration on a blue background depicting digital communication and data analysis. In the foreground, a person in a yellow shirt sits at a desk with a laptop displaying a red speech bubble icon. To their right, a large, stylized white and purple structure represents a digital interface or data dashboard. This structure features various elements: a person in a blue shirt sitting on top with a laptop; a red speech bubble; a yellow speech bubble with a plus icon; a blue speech bubble with a plus icon; a circular profile picture of a person; a bar chart; a line graph with a red trend line; and a document icon. A red headset is also visible near the base of the structure. The overall theme is digital connectivity and data-driven decision-making.

1. Create a new Freestyle Job
2. Rename and delete the Freestyle Job

Quick Check

You are setting up a Jenkins job to automate the build and deployment process for your software project. You need to choose the most appropriate Jenkins job type for executing custom build steps. Which job type should you select?

- A. Freestyle job
- B. Pipeline job
- C. Maven job
- D. Matrix job

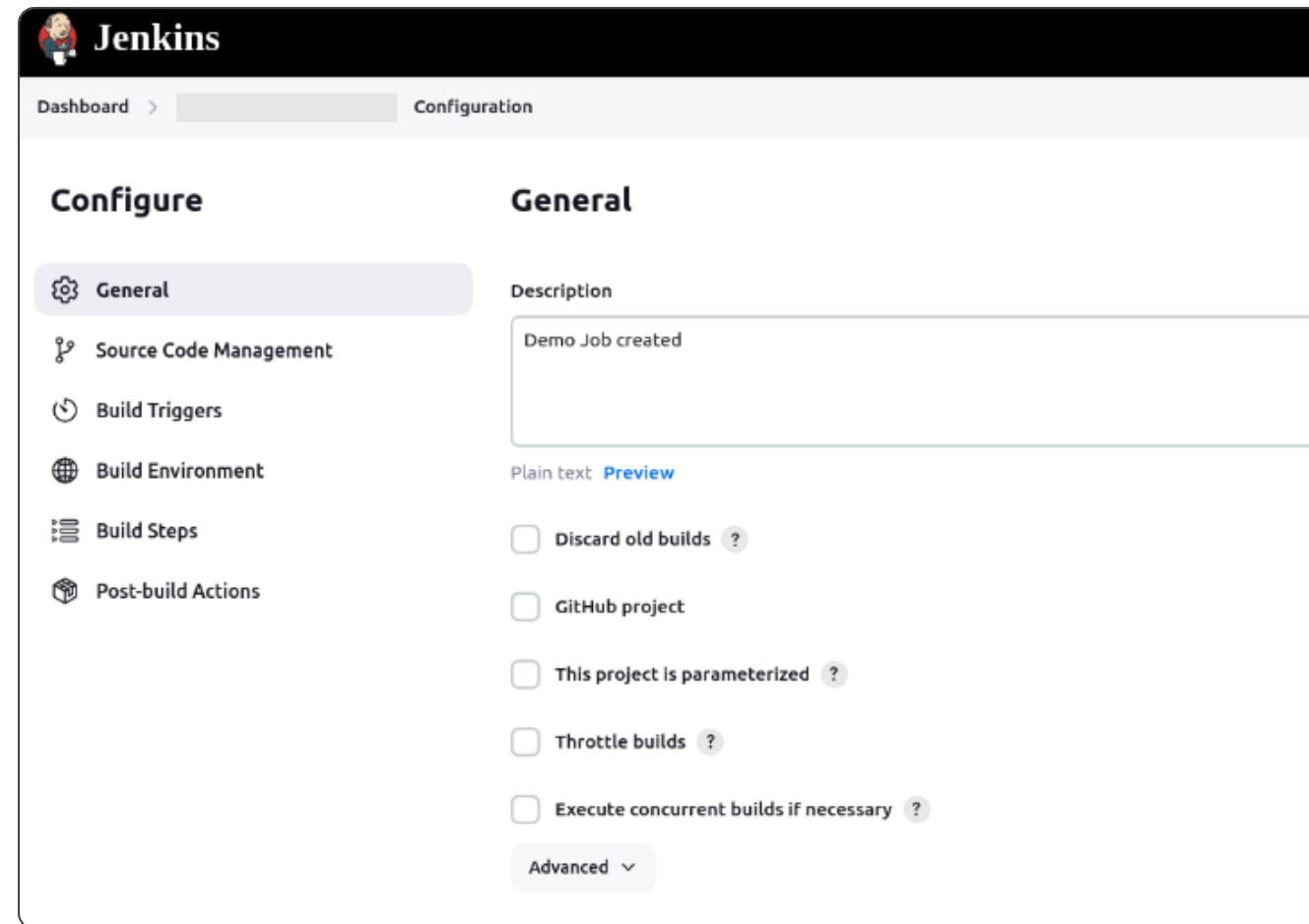




Jenkins Freestyle Job Section

General Section

It refers to the area where you configure basic settings and options for the job.



This section typically includes fields such as **job name**, **description**, **Discard old builds**, **Build Triggers**, **Poll SCM**, and **Concurrent build settings**.

General Section

Below details provide an overview of the configuration and settings related to job execution:

Job name and description:

Specifies the name and provides a brief description of the job or task

Discard old builds settings:

Controls the retention of old build data to manage storage space

Build triggers:

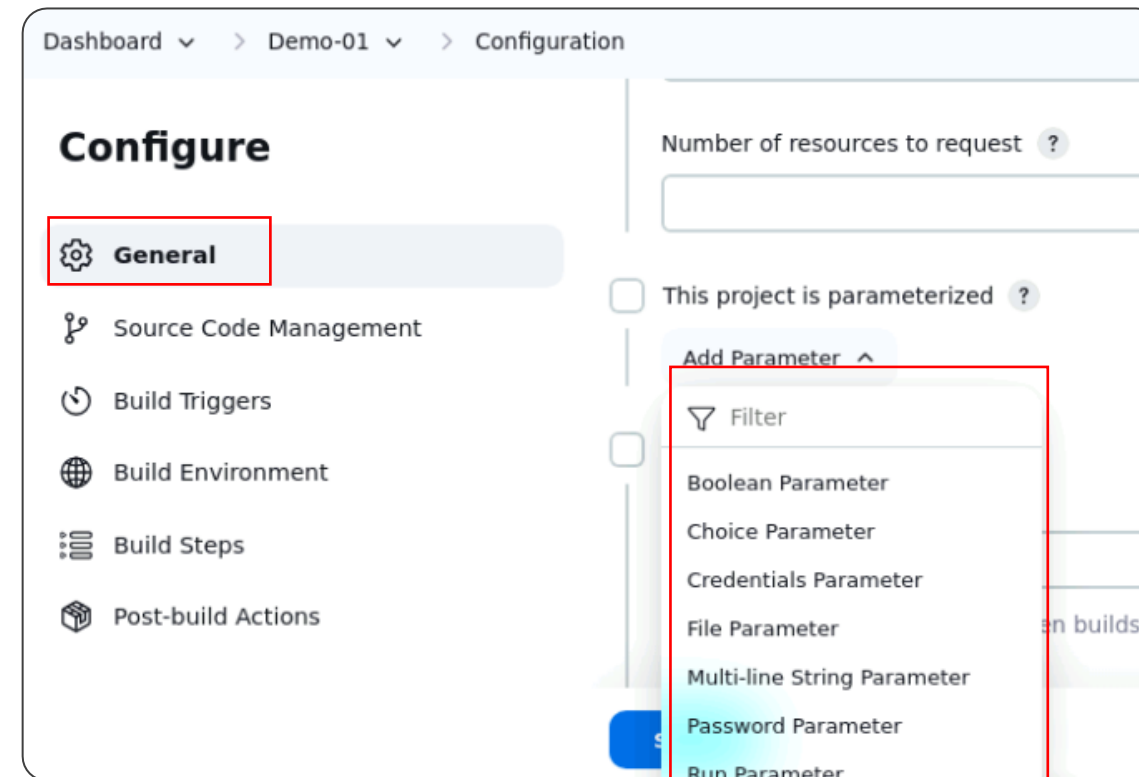
Defines events or conditions, like source code changes or scheduled intervals, that initiate job execution

Concurrent build options:

Manages whether multiple builds of the job can run simultaneously to optimize resource utilization

Build Parameters

They allow customization of job execution by providing dynamic input values during the job run.



Build parameters in Jenkins allow passing various types of data such as Git branch names, secret credentials, hostnames, and ports.

Build Parameters

Some commonly used build parameters are:

Boolean parameter:

Represents a true/false / yes/no option

Example: EnableDebugMode (true/false)

Choice parameter:

Provides a list of predefined choices, and the user selects one option

Example: Environment (Development, Testing, Production)

Credentials parameter:

Allows you to specify credentials (for example, username and password) securely

Example: DatabaseCredentials (username/password)

File parameter:

Enables users to upload a file as part of the build process

Example: ArtifactFile (path to the file)

Build Parameters

Some commonly used build parameters are:

String parameter:

Represents a simple text input

Example: VersionNumber (1.0.0)

Multi-line string parameter:

Accepts multi-line text input

Example: Release Notes (detailed release information)

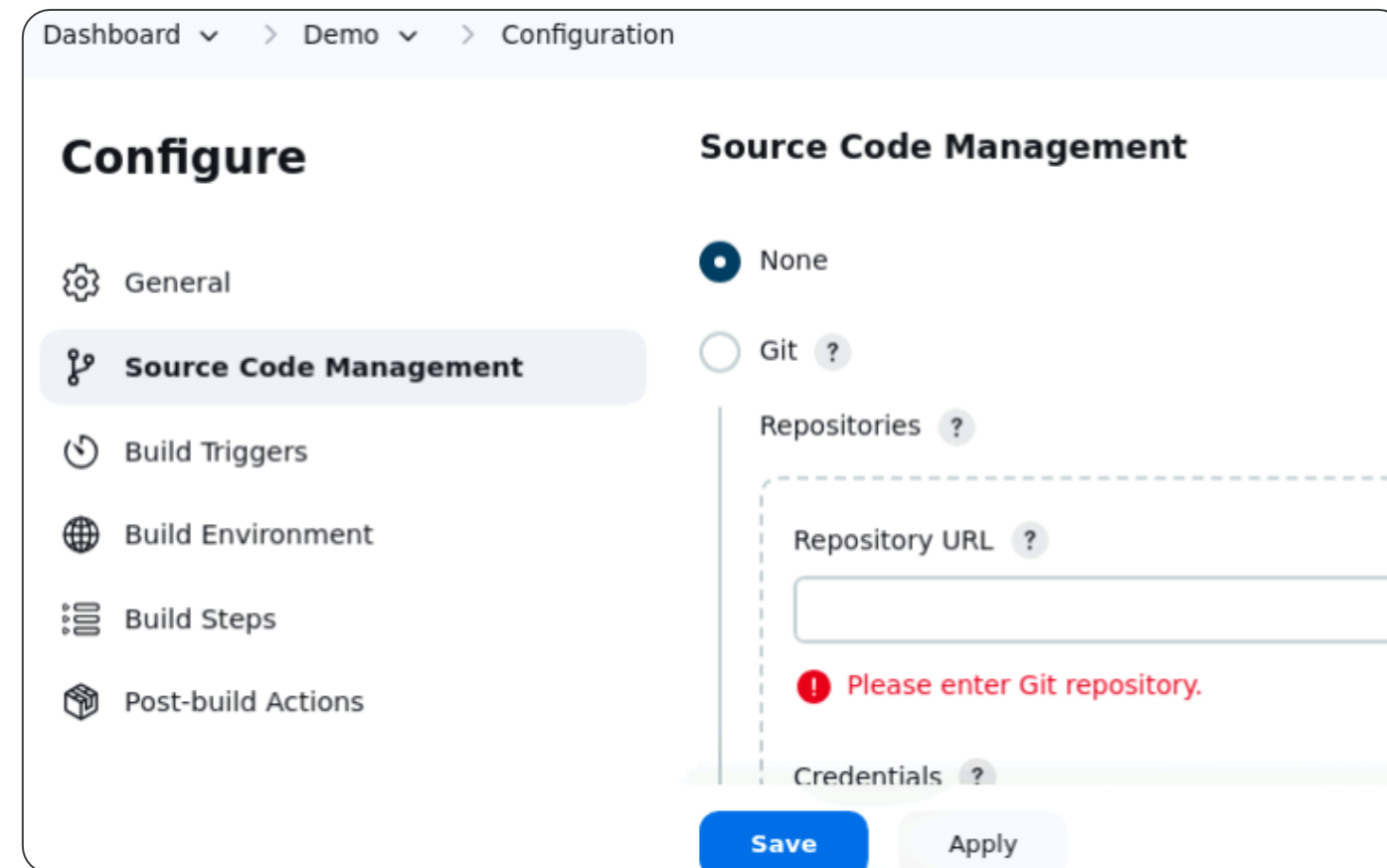
Password parameter:

Hides the input value for security reasons

Example: AdminPassword (*****)

Source Code Management (SCM)

It integrates version control systems by enabling automated builds and workflows by fetching code changes and maintaining version control.



The screenshot shows the Jenkins Configuration page for Source Code Management. The breadcrumb trail at the top is "Dashboard > Demo > Configuration". On the left, the "Configure" sidebar lists: General, Source Code Management (selected), Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main content area is titled "Source Code Management" and features a radio button selection for "None" (selected) and "Git". Below this, a dashed box contains the "Repositories" section with a "Repository URL" input field. A red error message "Please enter Git repository." is displayed below the input field. At the bottom of the dashed box is the "Credentials" section. At the very bottom of the configuration area are "Save" and "Apply" buttons.

SCM integration is essential for triggering builds and ensuring Jenkins retrieves the latest code from repositories for software development projects.

Build Periodically

Key aspects of source code management in Jenkins include:

Version control systems
(VCS)

Support Git, SVN, CVS, Mercurial, and Perforce for managing source code versions

Repository URL

Provide the repository URL where the code is hosted for Jenkins to access

Credentials

Configure credentials (username and password or SSH keys) if authentication is required for repository access

Branches and tags

Specify branches or tags to fetch code from specific versions during builds

Build Periodically

Key aspects of source code management in Jenkins include:

Polling SCM

Enable SCM polling to automatically trigger builds when new commits are detected

Checkout strategy

Choose how Jenkins checks out code, such as to a sub-directory or cleaning workspace before checkout

Additional SCM configurations

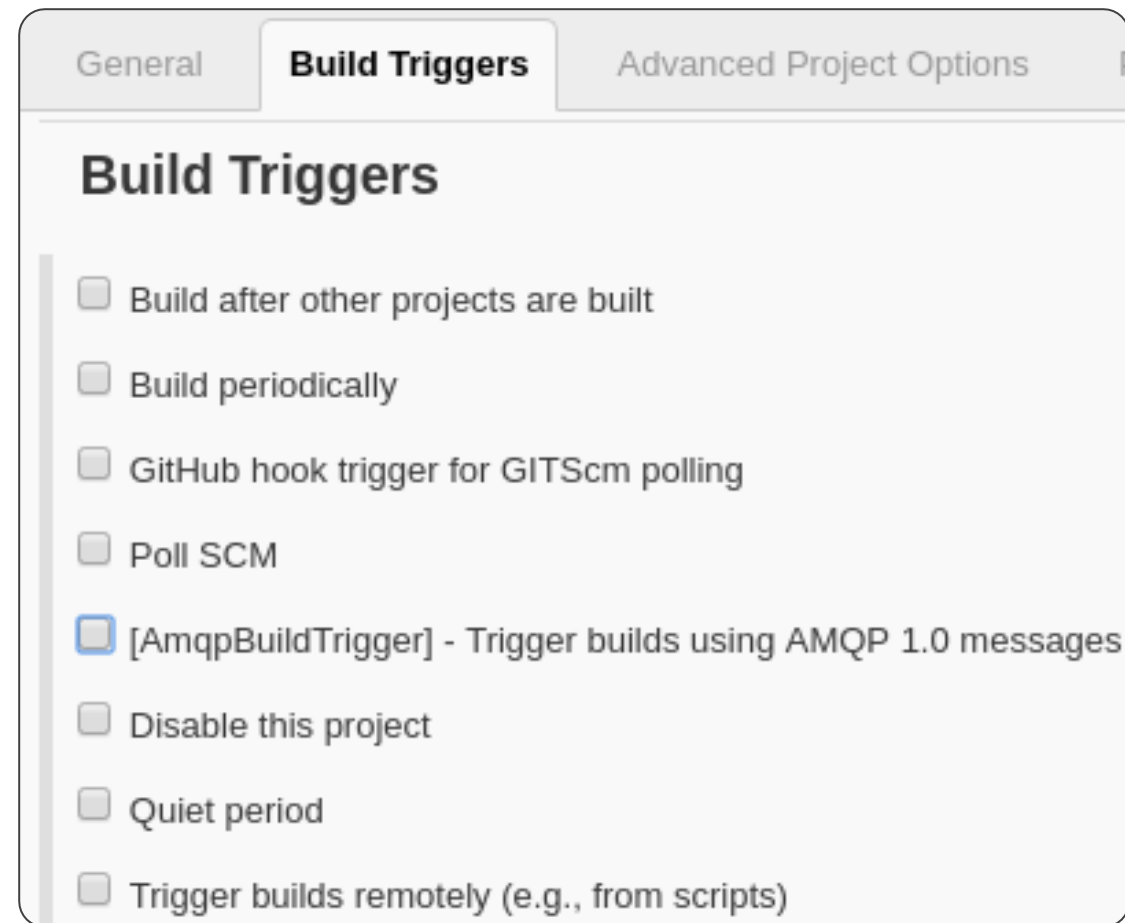
Configure settings like proxy, shallow clones, submodules, or sparse checkouts as needed

Pipeline SCM syntax

Define SCM configurations in Jenkins pipeline using scripted or declarative syntax for complex setups

Build Triggers

These are the configurations that specify when a build should be initiated for a job based on defined events or conditions.



The screenshot shows the Jenkins configuration interface for a job. At the top, there are three tabs: 'General', 'Build Triggers' (which is selected), and 'Advanced Project Options'. Below the tabs, the 'Build Triggers' section is displayed. It contains a list of checkboxes for various build triggers. The checkbox for '[AmqpBuildTrigger] - Trigger builds using AMQP 1.0 messages' is checked, while all other checkboxes are unchecked.

Build Trigger	Selected
<input type="checkbox"/> Build after other projects are built	False
<input type="checkbox"/> Build periodically	False
<input type="checkbox"/> GitHub hook trigger for GITScm polling	False
<input type="checkbox"/> Poll SCM	False
<input checked="" type="checkbox"/> [AmqpBuildTrigger] - Trigger builds using AMQP 1.0 messages	True
<input type="checkbox"/> Disable this project	False
<input type="checkbox"/> Quiet period	False
<input type="checkbox"/> Trigger builds remotely (e.g., from scripts)	False

Build triggers determine the start of build job execution in Jenkins, ensuring automated responses to relevant events or criteria.

Build Triggers

Below are some popular triggers:

Git webhook: Builds the trigger automatically in response to code push or specific events in the Git repository

Poll SCM: Checks the version control system for changes and triggers builds when new commits are detected

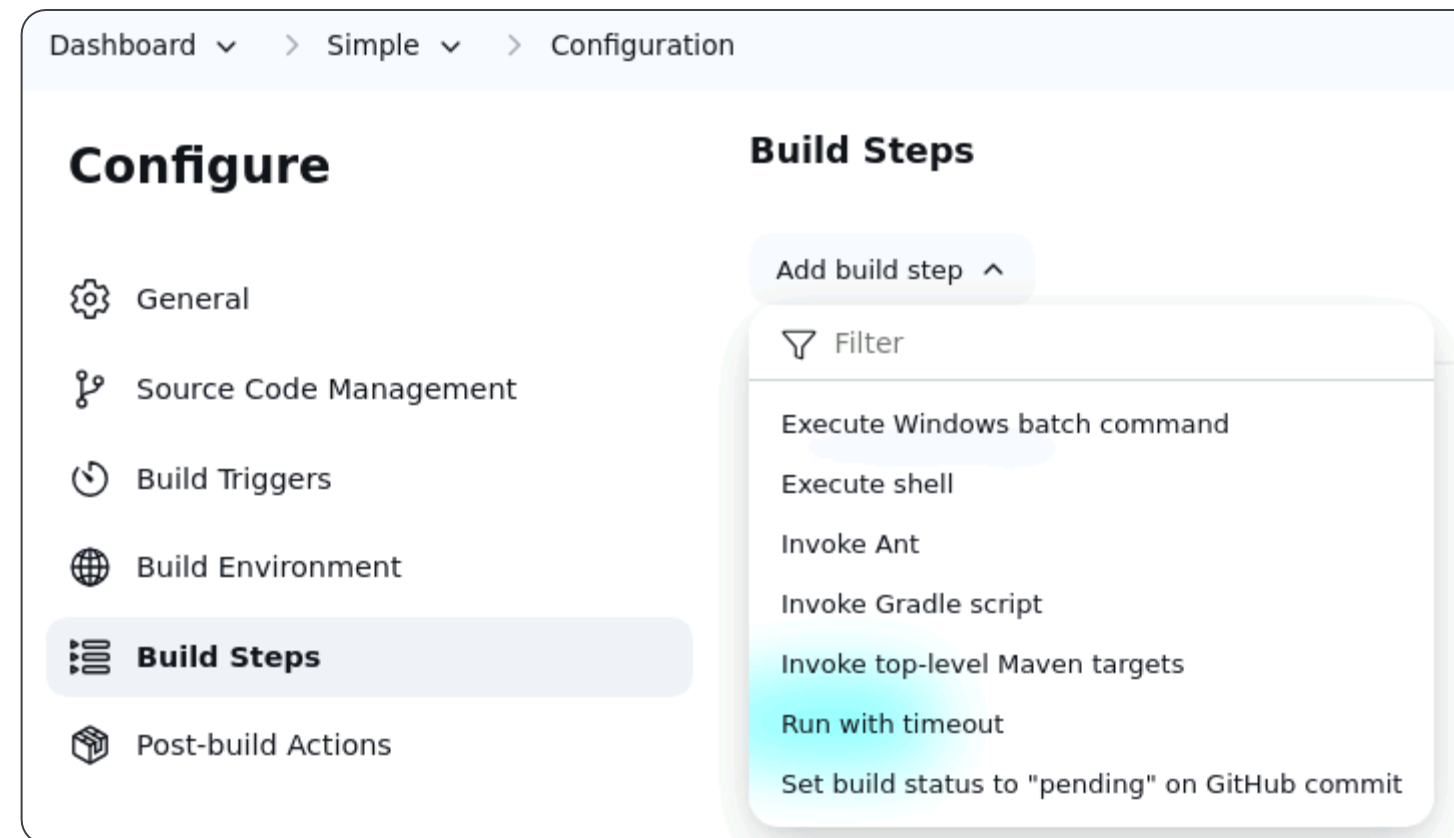
Scheduled job: Initiates builds at predefined times or intervals, enabling regular automated executions

Remote triggers: Starts builds from external sources or scripts, facilitating integration with external systems

Build after other project are built: Initiates downstream projects after completing upstream projects successfully, based on dependencies

Build Steps

It is a defined task or action executed during a job build process, such as compiling code, running tests, or deploying applications.



Build steps are configured within the job settings and executed sequentially, contributing to the automated build process and producing the desired build outcome.

Build Steps

Below are some options for configuring build steps:

Execute windows batch command:

Runs a series of commands specific to windows environments

Execute shell:

Executes a series of commands for Unix-based systems (Linux, macOS)

Invoke Ant:

Calls Apache Ant, a popular build tool for Java projects

Invoke Gradle script:

Executes Gradle, a versatile build tool for various projects

Build Steps

Below are some options for configuring build steps:

Invoke top-level maven targets:

Runs the core functionalities defined in a Maven project pom.xml file

Run with timeout:

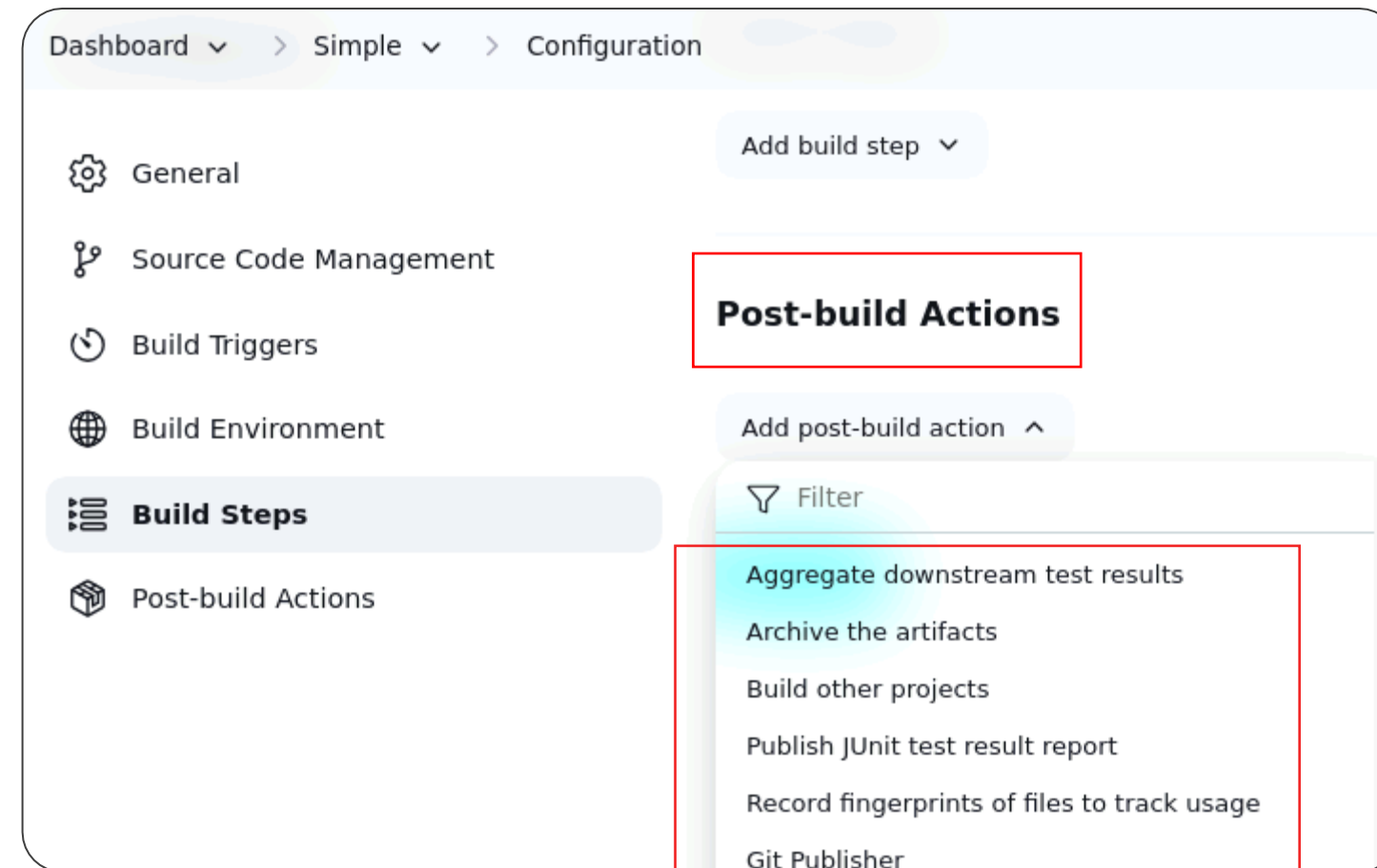
Executes a command or sequence with a time limit, stopping it if unfinished

Set build status to 'pending' on GitHub commit:

Updates the status of a commit on GitHub to indicate an ongoing build process

Post-Build Actions

A Jenkins **Post-build Actions** is a task executed after the build has been completed.



Post-build Actions is typically used to perform additional processing or reporting on the build results or to trigger other actions based on the build outcome.

Post-Build Actions

Below are the list of some post-build actions:

Aggregate downstream test results:

Combines test results from dependent projects

Archive the artifacts:

Saves build outputs for later use (for example, deployment)

Build other projects:

Initiates the build of subsequent projects after the completion of the current project

Publish JUnit test result report:

Generates a report summarizing test execution

Record fingerprints of files to track usage:

Tracks which builds used specific files

Post-Build Actions

Some examples of post-build actions include:

- Sending an email notification with the build results
- Archiving build artifacts for future reference
- Triggering a deployment to a staging or production environment
- Updating a ticketing system with build information

Assisted Practice



Creating a new Jenkins Job to checkout source code

Duration: 10 Min.

Problem statement:

You have been assigned a task to set up a Jenkins job to manage source code, specifically by configuring the Source Code Management section to check out code from a Git repository.

Outcome:

By completing this demo, you will be able to set up a Jenkins job to manage source code by configuring the Source Code Management section to check out code from a Git repository.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Log in and create a Jenkins job
2. Configure source code management



Setting up Jenkins Notifications

Jenkins Notifications

Jenkins supports sending build status notifications.



Several built-in and third-party tools are available for sending notifications.

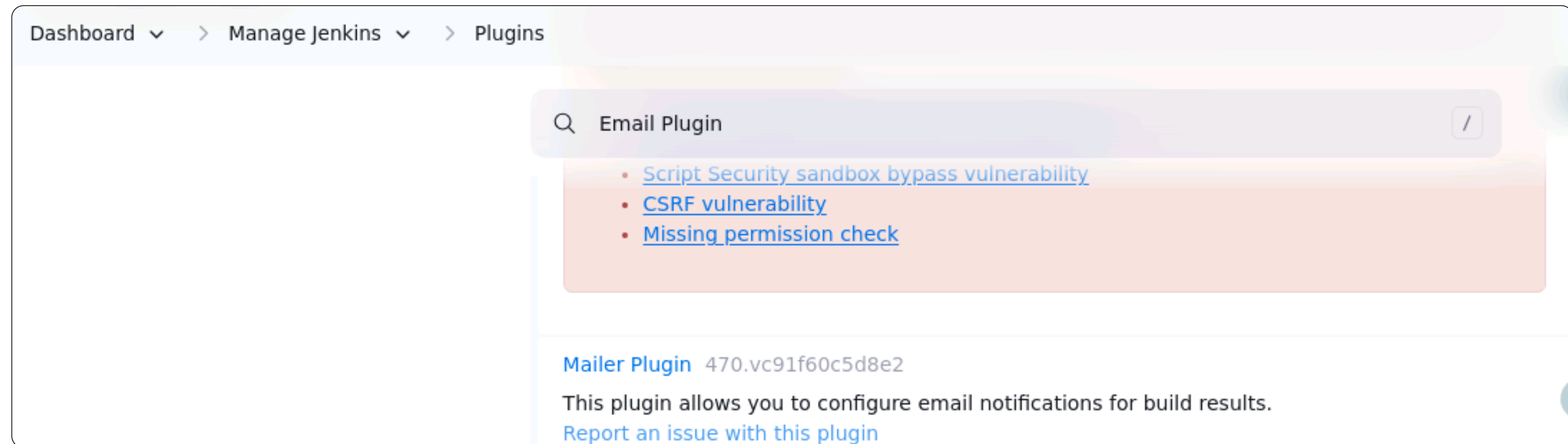


Notifications can also be used to send alerts upon the completion of application deployments.

Jenkins supports email notifications by default. Users can also install various plugins to support different notification channels.

Email Plugin

It facilitates customized email notifications for build events like successes, failures, and other triggers.



By leveraging this email plugin, Jenkins users can stay informed about build statuses and take necessary actions promptly during the CI/CD process.

Email Plugin

Below are some functionalities of the email plugin:

Sends emails based on various build triggers and events, offering more flexibility than the default settings

Supports a wider range of email recipients, allowing to specify project-specific email lists

Enables customization of email content using tokens, providing informative and dynamic notifications

Extended Email Notification Plugin

It is also known as the **Email-ext** plugin, a powerful tool in Jenkins that extends the functionality of the built-in email notification system.

Extended E-mail Notification

SMTP server


smtp.gmail.com

Default user E-mail suffix

☒ Use SMTP Authentication

User Name

Password

 Concealed

Advanced Email Properties

Use SSL

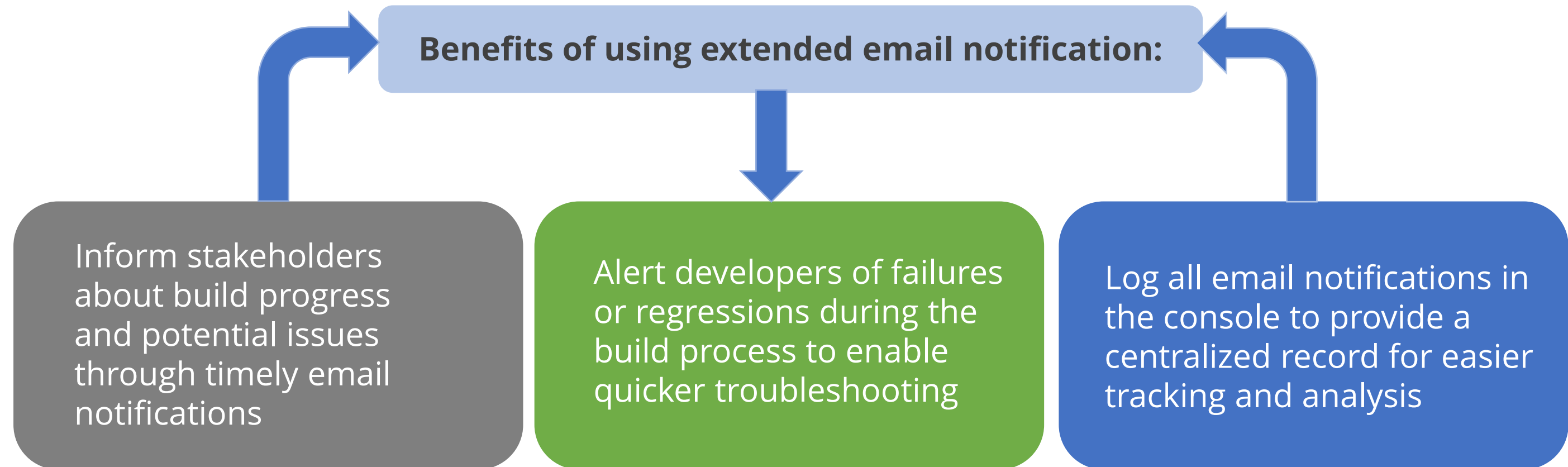
☒

SMTP port

465

Extended Email Notification Plugin

It empowers the creation of a robust and customizable email notification system for the Jenkins builds, promoting better communication and faster issue resolution.



Slack Notifications

It allows integrating Jenkins builds with Slack, a popular communication platform, enabling real-time notifications about builds to be received directly within the Slack workspace.

Slack Notifications

Project Channel	<input type="text" value="#your_chanel_name"/>
Notify Build Start	<input checked="" type="checkbox"/>
Notify Aborted	<input checked="" type="checkbox"/>
Notify Failure	<input checked="" type="checkbox"/>
Notify Not Built	<input checked="" type="checkbox"/>
Notify Success	<input checked="" type="checkbox"/>
Notify Unstable	<input checked="" type="checkbox"/>
Notify Back To Normal	<input checked="" type="checkbox"/>
<input type="checkbox"/> Disable Build (No new builds will be executed until the project is re-enabled.)	
<input type="checkbox"/> Execute concurrent builds if necessary	

Slack Notifications

Below are the components of Slack notifications:

Project channel: Specify the name of the Slack channel where notifications will be sent for the project's builds

Build triggers: Determine when a Slack notification is sent

Notify build start: Send a notification when the build begins

Notify aborted: Send a notification if the build is aborted

Notify failure: Send a notification if the build fails

Slack Notifications

Below are the components of Slack notifications:

Notify not built: Send a notification if the build is not built

Notify success: Send a notification if the build succeeds

Notify unstable: Send a notification if the build is unstable

Notify back to normal: Send a notification if the build recovers from an unstable or failed state

Assisted Practice



Setting up custom SMTP to send email notification

Duration: 15 Min.

Problem statement:

You have been assigned a task to configure SMTP email notification in Jenkins for sending automated notifications and alerts.

Outcome:

By completing this task, you will be able to configure SMTP email notifications in Jenkins for sending automated notifications and alerts by creating an app password and configuring SMTP settings.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Create an app password for SMTP configuration
2. Configure SMTP configurations in Jenkins

Quick Check



During the build process in Jenkins, which section is responsible for specifying where the source code for the project is located and how it should be managed?

- A. Source code management
- B. Build step
- C. Post build section
- D. Demo section

Key Takeaways

- Jenkins freestyle job offers configurable parameterized builds and plugin integrations through a user-friendly graphical interface.
- Plugins are like building blocks that allow to tailor Jenkins according to the specific needs within a CI/CD pipeline.
- General section in Jenkins refers to the area to configure basic settings and options for the job.
- SCM integration is essential for triggering builds and ensuring Jenkins retrieves the latest code from repositories for software development projects.
- The Email extension plugin (**Email-ext**) in Jenkins enhances email notification capabilities beyond the built-in options.



Implementing Job Chaining in Jenkins

Duration: 25 Min.

Project agenda: To build and execute a Jenkins job chaining workflow with the build pipeline plugin and GitHub integration for efficient continuous integration and deployment processes

Description: Imagine a development team at a financial technology company working on a critical Java application that manages customer transactions. The team uses Maven for building their application and Jenkins for continuous integration. The project is hosted on GitHub, and the team wants to build a pipeline that includes tasks such as checking out code, compiling code, and packaging code. In this project, you will create a pipeline in Jenkins where jobs can be set up and chained together by integrating a project hosted on GitHub.



Implementing Job Chaining in Jenkins

Duration: 25 Min.

Perform the following:

1. Create jobs in a Jenkins pipeline
2. Install the build Pipeline plugin in Jenkins
3. Trigger the execution of jobs

Expected deliverables: A Jenkins pipeline demonstrating job chaining for continuous integration





Thank You