Experiment 4:

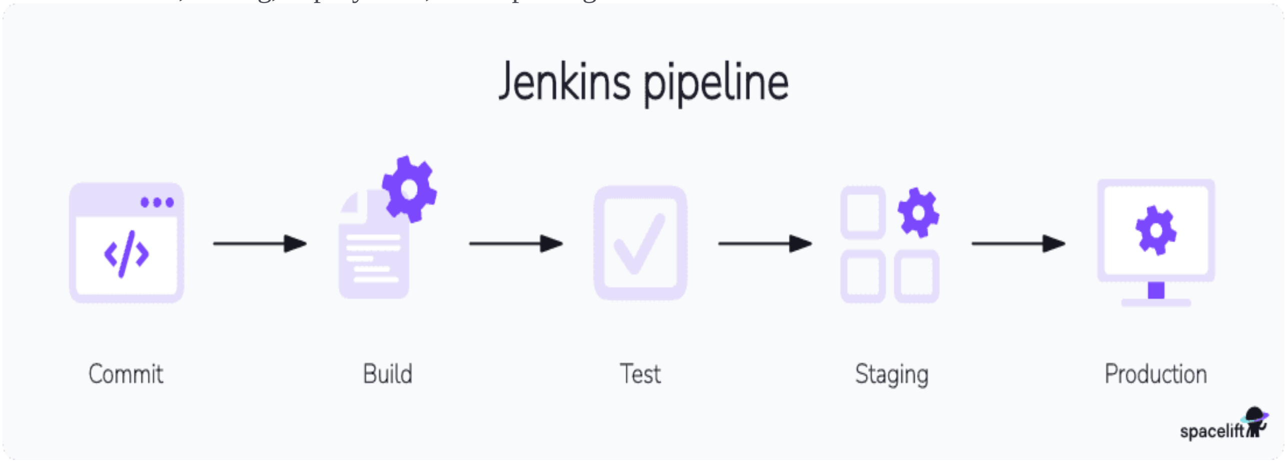
What is Jenkins?

Jenkins is an open-source automation server that helps developers implement continuous integration and continuous deployment (CI/CD) workflows. It simplifies software development by automating build, test, and deployment tasks.

Jenkins reduces manual effort by automating the execution of different development workflows. It operates using highly customizable pipelines that can be configured using Groovy-based scripts.

Developers can commit code changes to Git repositories, and Jenkins then detects these updates to initiate the build process automatically. The build process in Jenkins includes compiling source code and running tests to provide feedback.

Jenkins also supports continuous delivery by managing dependencies and deploying artifacts like JAR files or container images. Its extensive plugin ecosystem allows teams to customize workflows for version control, testing, deployment, and reporting.



At its core, Jenkins uses jobs (or pipelines) to define automation processes like building, testing, and deploying code. These jobs can be configured manually or scripted using Jenkins Pipeline DSL, which allows version-controlled, code-based automation.

A Jenkins environment consists of the master server (controller) and its connected agents, which run the defined jobs. Jenkins also relies heavily on a plugin-based architecture, enabling integration with countless tools and systems in the DevOps ecosystem.

Key features of Jenkins

Jenkins provides a flexible automation platform designed to streamline CI/CD workflows. Its powerful features make it a great choice for software teams

- **Pipeline as code:** Jenkins allows developers to define CI/CD workflows using code for better control and repeatability.
- **Extensive plugin support:** Thousands of plugins help extend functionality for testing, deployments, and integrations.
- **Distributed build system:** Jenkins can run builds across multiple machines to speed up execution and handle large projects.
- **Version control integration:** Jenkins integrates with Git, SVN, and other version control systems to track changes.
- **Customizable build triggers:** Developers can set up automated builds based on code commits, schedules, or external events.
- **Security and access control:** Role-based access ensures secure management of builds and sensitive project configurations.
- **Cloud and container support:** Jenkins works with cloud services and container platforms like AWS, Kubernetes, and Docker
- **Built-in monitoring and reporting:** Jenkins provides real-time logs, build reports, and test results for better project visibility.

Benefits of Jenkins

Jenkins can help automate the entire software build, test, and deployment process, which improves development efficiency and reduces the risk of human error. Here are the main benefits of using Jenkins:

- **Free and open source:** Jenkins offers a cost-effective alternative to proprietary CI/CD tools. It is backed by an active developer community that continually contributes plugins and updates.
- **Accelerated development pipelines:** Jenkins automates tasks like code compilation, testing, and deployment, reducing manual handoffs and significantly shortening release cycles.
- **Proactive issue detection:** Continuous integration enables early identification of regressions or integration conflicts, lowering the likelihood of production defects.
- **Enforced quality gates:** Built-in support for automated testing and static code analysis ensures that only code meeting defined quality thresholds proceeds to production.
- **Horizontal scaling:** Jenkins' master-agent architecture allows teams to manage parallel builds across distributed systems, making it suitable for projects of increasing complexity and scale.
- **Customizable workflows:** Jenkins supports fine-grained control over build pipelines through tools like Jenkins Pipeline and Declarative DSL, enabling teams to tailor automation to project-specific logic.
- **Centralized monitoring and reporting:** Jenkins provides real-time feedback through build dashboards, test summaries, and integration with issue tracking tools, improving visibility and coordination across teams.

How to create a Jenkins pipeline

Below is an example of a simple Jenkins pipeline, which will automate the build, test, and deployment process for a sample project.

The pipeline checks that every change made to the codebase is tested and deployed efficiently. With Jenkins, pipelines provide a way to define these steps as code to maintain and repeat across different projects.

Step 1: Install Jenkins

Jenkins offers a user-friendly Windows installer that simplifies the setup process by installing Jenkins as a Windows service. This ensures that Jenkins runs efficiently in the background with minimal manual configuration. The installer also integrates with system services to automate software builds and deployments easily.

Jenkins also provides platform-specific installation support for users installing Jenkins on other platforms, such as Linux, macOS, Kubernetes, or Docker.

Additionally, Jenkins can be run using the standalone WAR file to achieve greater flexibility in deployment. Check out the official Jenkins documentation to explore the detailed installation guide based on your preferred operating system or environment.

Prerequisites

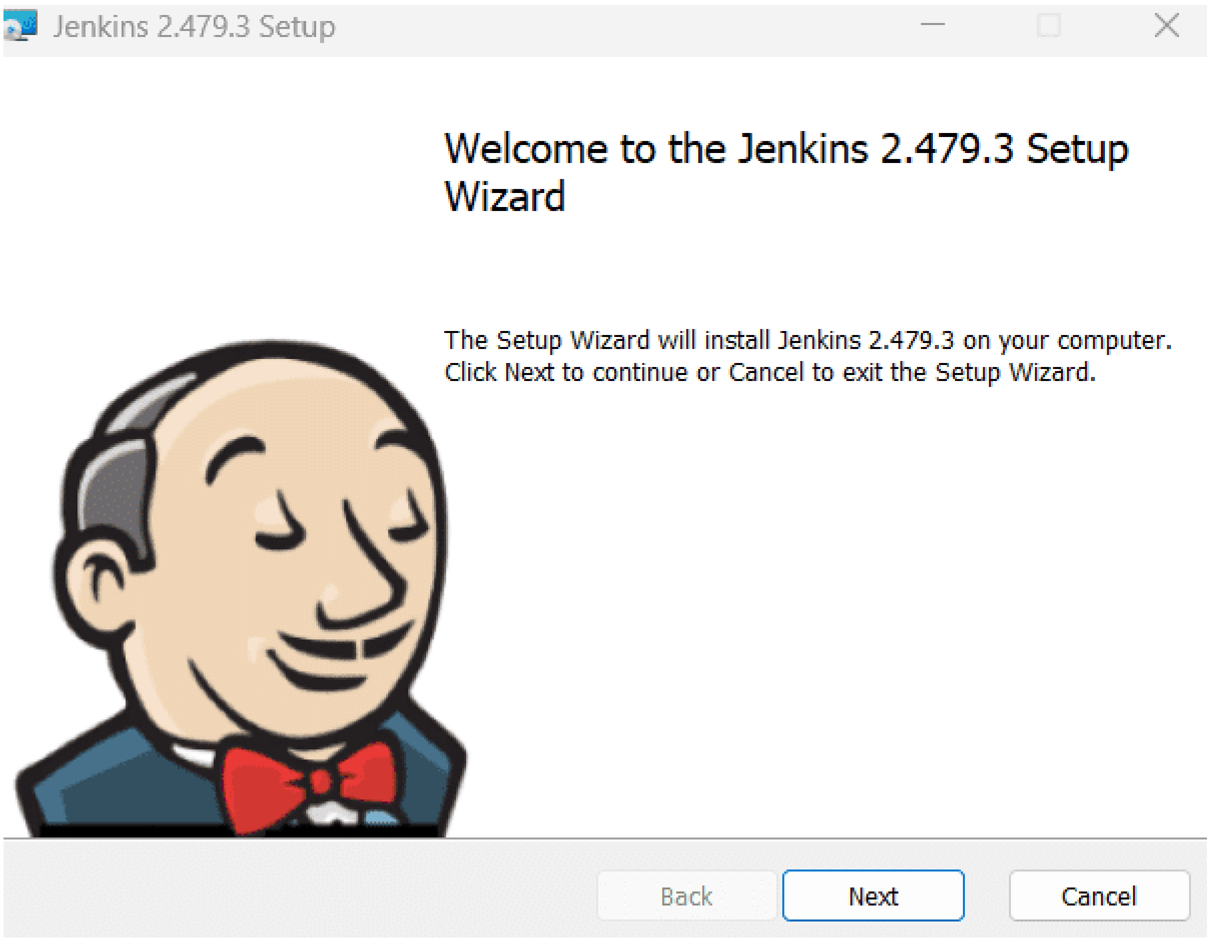
Before installing Jenkins on Windows, your system must have at least **256 MB of RAM** and **1 GB of storage.** If you plan to use Jenkins for Docker setups, you need a minimum of **10 GB** of storage. A setup with 4GB+ of RAM and 50GB+ of storage is ideal for small teams.

Jenkins also requires a compatible **Java version**. For more details, refer to the Java Requirements page. For smooth installation, Jenkins installation requires a **compatible web browser** with Jenkins UI and adherence to the Windows Support Policy.

1. Download the Jenkins installer

To begin the Jenkins installation, visit the Jenkins Download page. There, you can choose between an LTS release and a weekly release for Windows.

After selecting the appropriate version, click the “Windows” option to download the installer package. Once the download is complete, run the installer to begin the installation process.

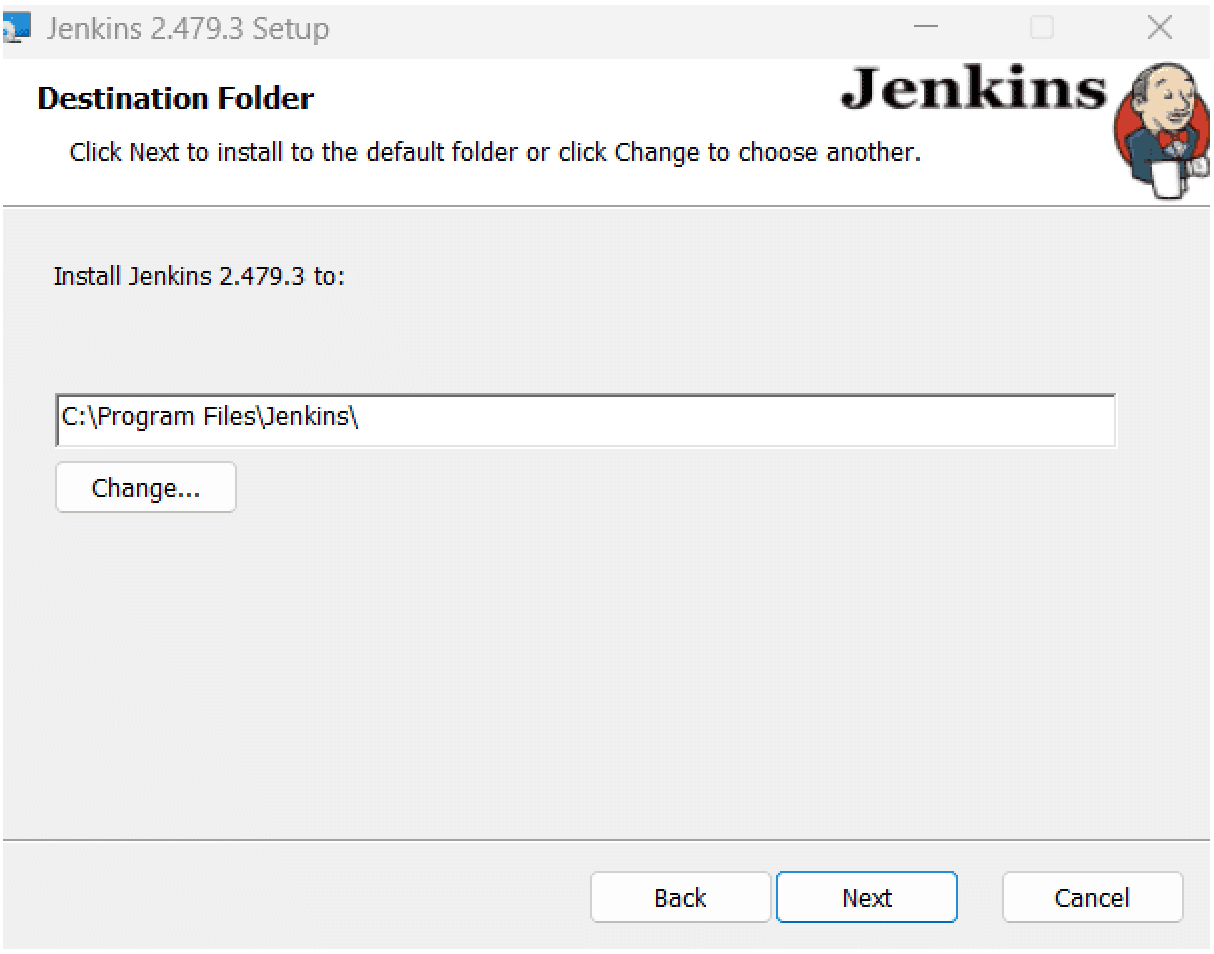

Jenkins 2.479.3 Setup

Welcome to the Jenkins 2.479.3 Setup Wizard

The Setup Wizard will install Jenkins 2.479.3 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

2. Initiate the setup wizard

Once you click *Next*, it directs you to specify the destination folder path.


Jenkins 2.479.3 Setup

Destination Folder

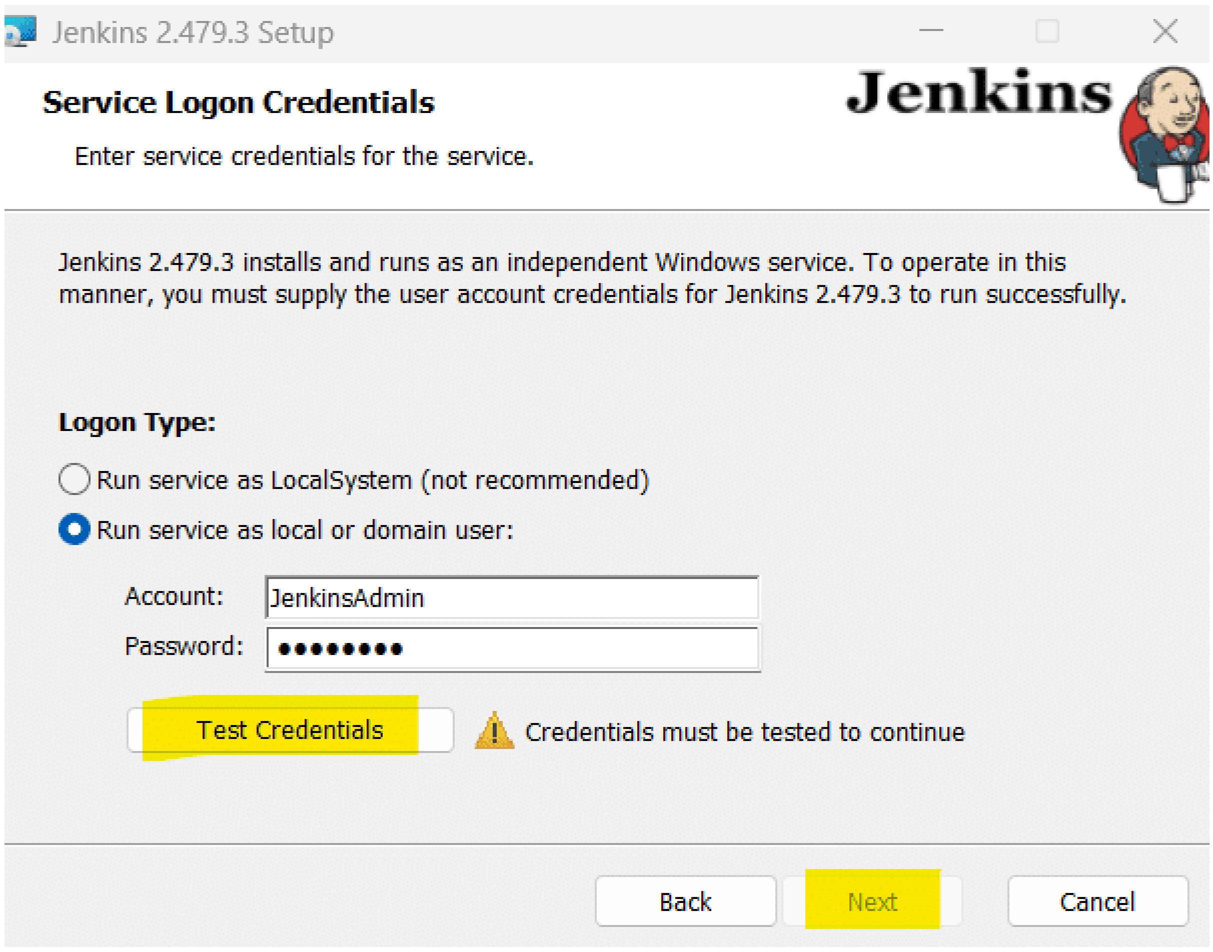
Click Next to install to the default folder or click Change to choose another.

Install Jenkins 2.479.3 to:

C:\Program Files\Jenkins\

3. Configure installation directory and user settings

Choose the directory where Jenkins should be installed and click *Next* to specify a local or domain user for running Jenkins. Click *Test Credentials* to validate permissions, and click *Next* after successful validation.


Jenkins 2.479.3 Setup

Service Logon Credentials

Enter service credentials for the service.

Jenkins 2.479.3 installs and runs as an independent Windows service. To operate in this manner, you must supply the user account credentials for Jenkins 2.479.3 to run successfully.

Logon Type:

Run service as LocalSystem (not recommended)

[✓ Run service as local or domain user:]

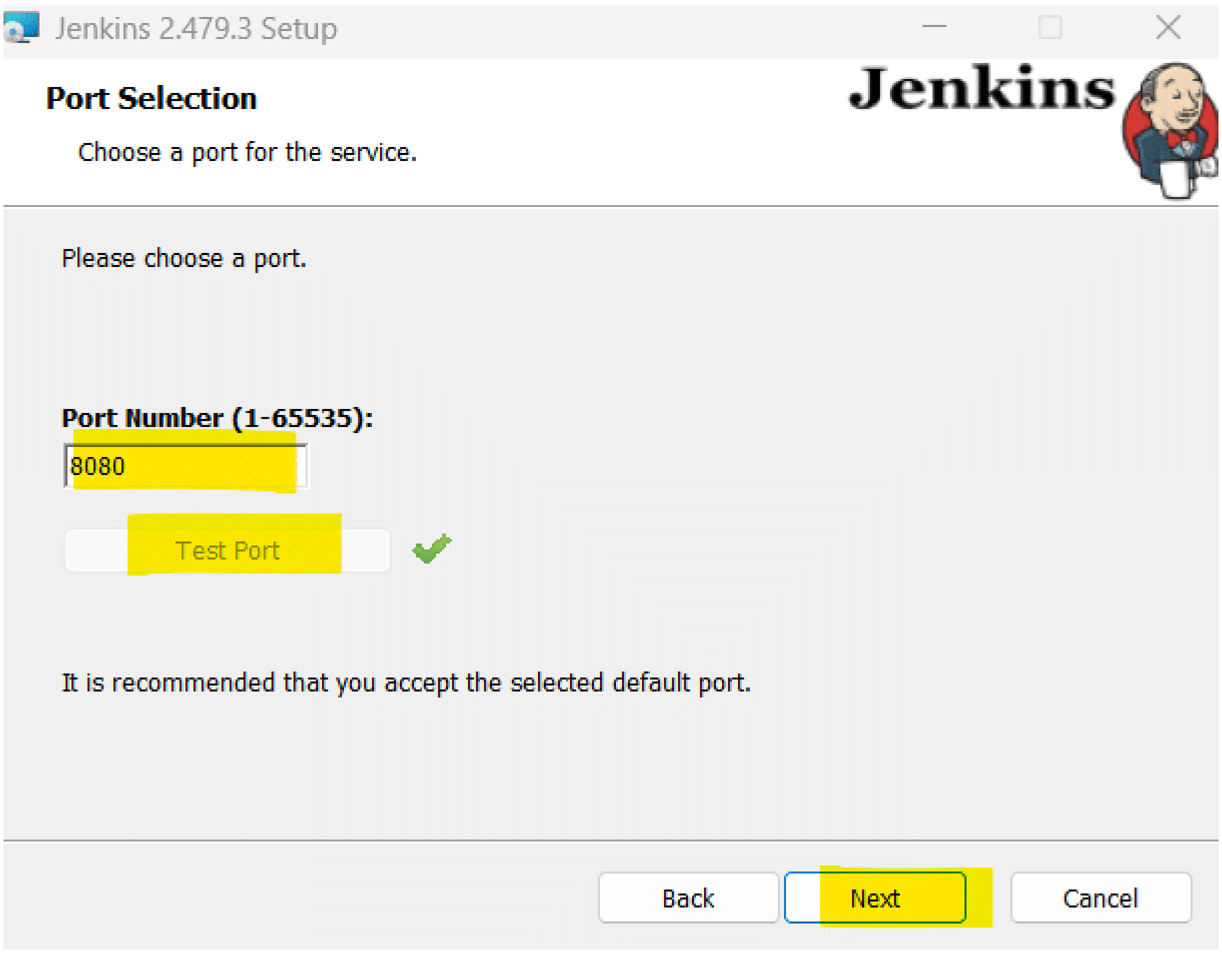
Account: JenkinsAdmin

Password: ●●●●●●●●●

Credentials must be tested to continue

4. Port selection

Choose the port Jenkins should listen on (default: 8080), click the *Test Port* option to ensure it is free, and choose *Next* to proceed.


Jenkins 2.479.3 Setup

Port Selection

Choose a port for the service.

Please choose a port.

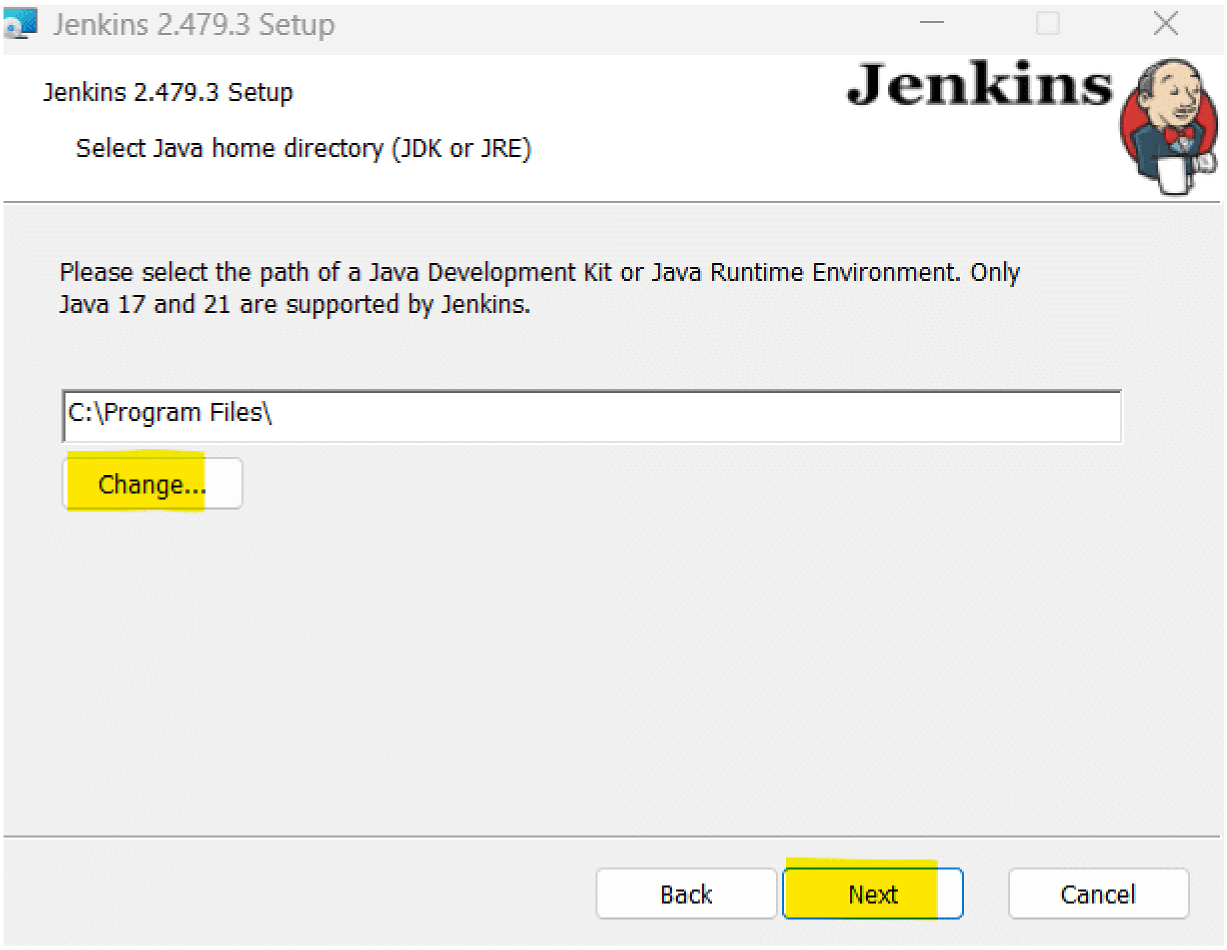
Port Number (1-65535):

8080

It is recommended that you accept the selected default port.

5. Java home directory

The installer detects your Java installation automatically. If Java is missing, install it and specify the Java home directory. Click *Next* to continue.


Jenkins 2.479.3 Setup

Jenkins 2.479.3 Setup

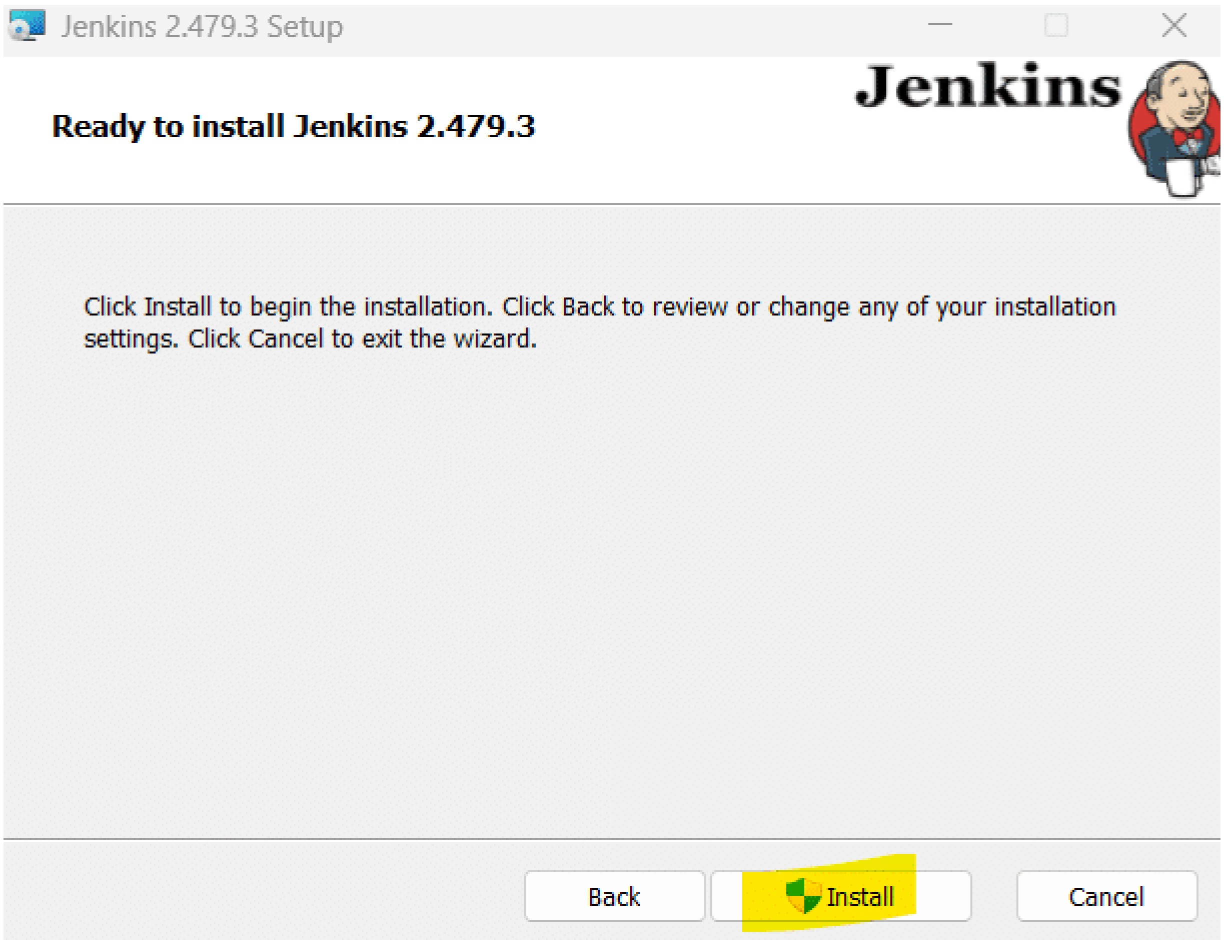
Select Java home directory (JDK or JRE)

Please select the path of a Java Development Kit or Java Runtime Environment. Only Java 17 and 21 are supported by Jenkins.

C:\Program Files\

6. Install Jenkins

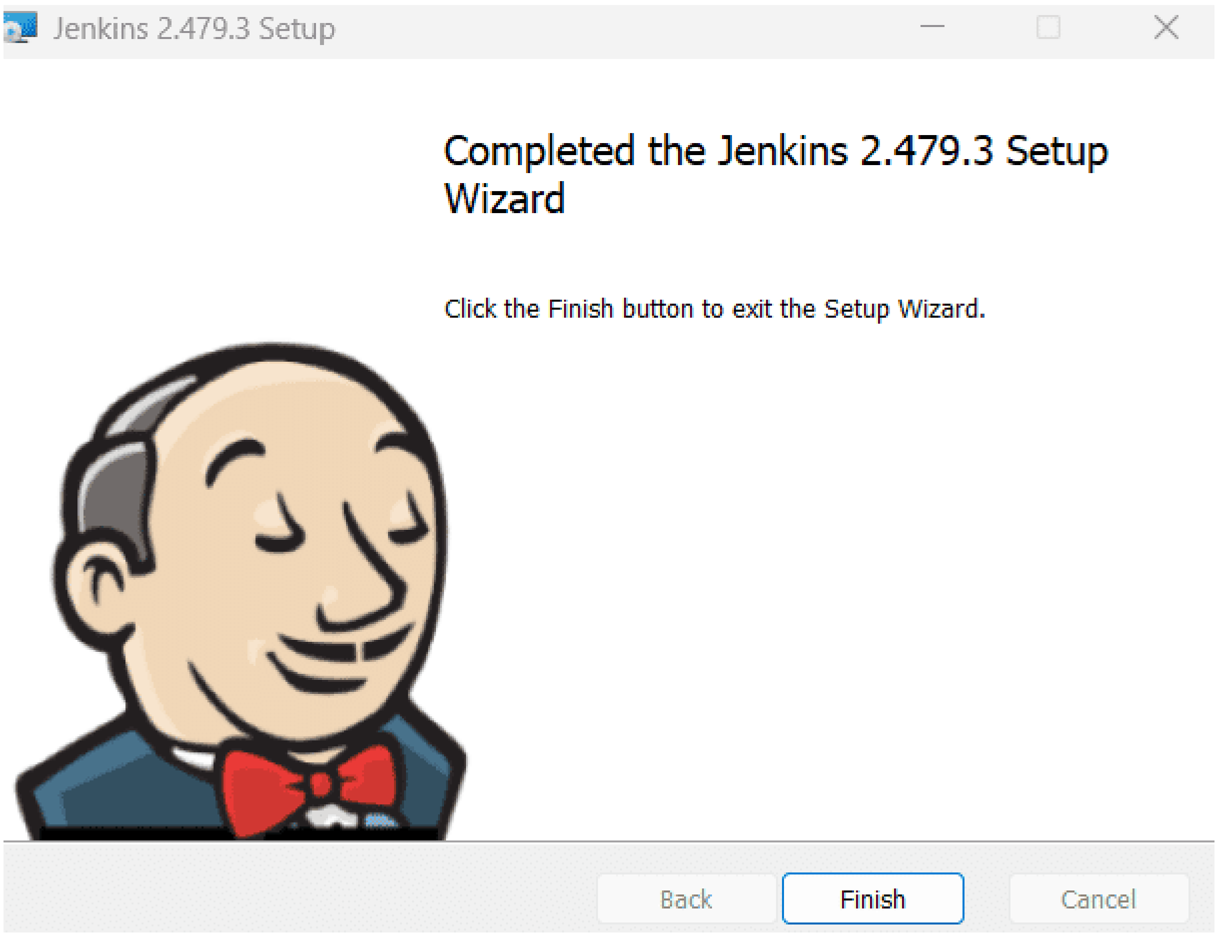
Click *Install* to start the installation and wait for the installation to complete.


Jenkins 2.479.3 Setup

Ready to install Jenkins 2.479.3

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Click *Finish* when complete.

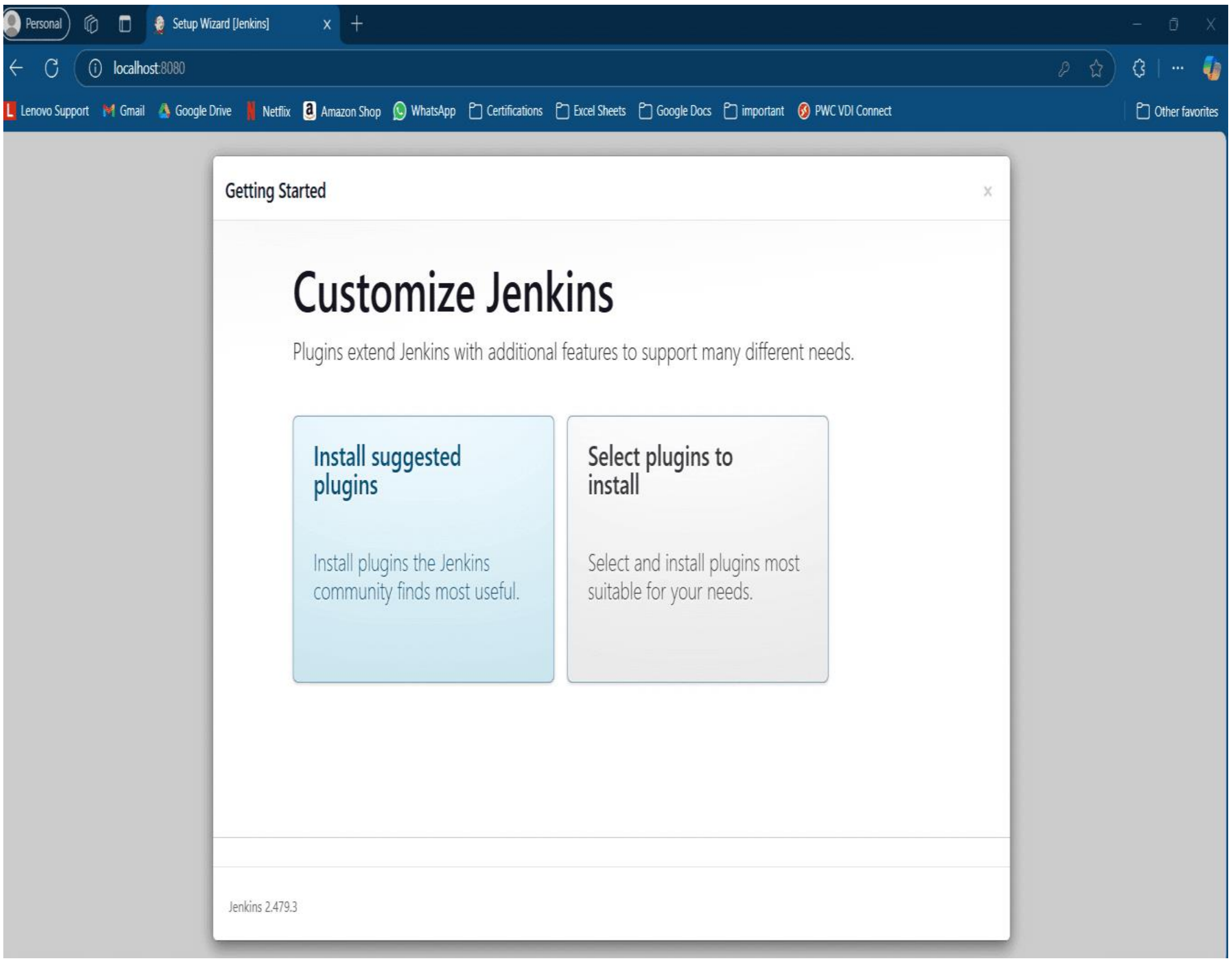

Jenkins 2.479.3 Setup

Completed the Jenkins 2.479.3 Setup Wizard

Click the Finish button to exit the Setup Wizard.

7. Post-installation validation

Open Windows Services and check if Jenkins is running to verify the installation. Otherwise, you can navigate to http://localhost:8080 in your browser for verification.


Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

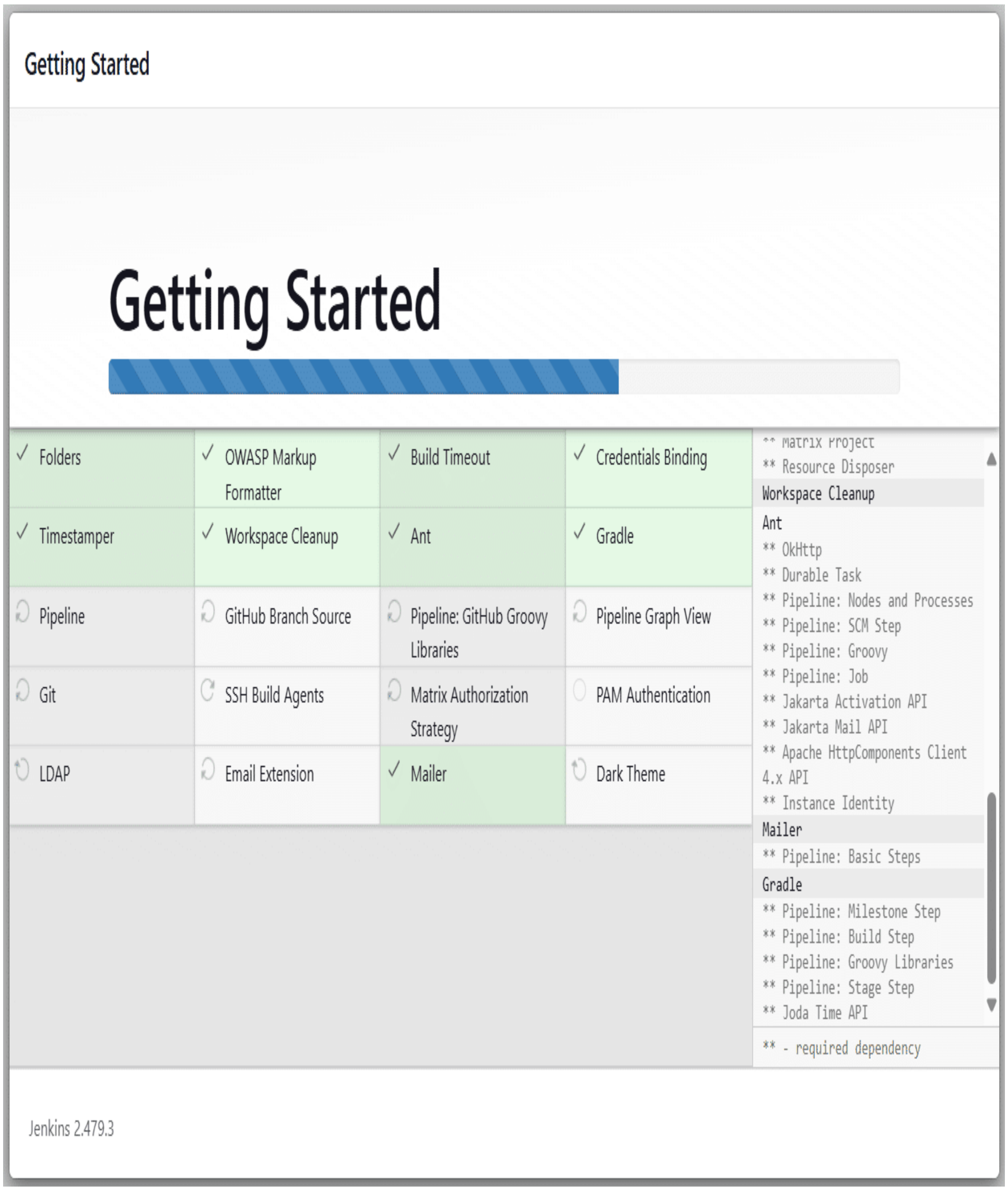
Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.479.3

Once you retrieve the initialAdminPassword from C:/Program Files/Jenkinssecrets and enter it on the Unlock Jenkins page, you can customize Jenkins with Plugins.

You can click *Install Suggested Plugins* for a basic setup.



Getting Started

Getting Started

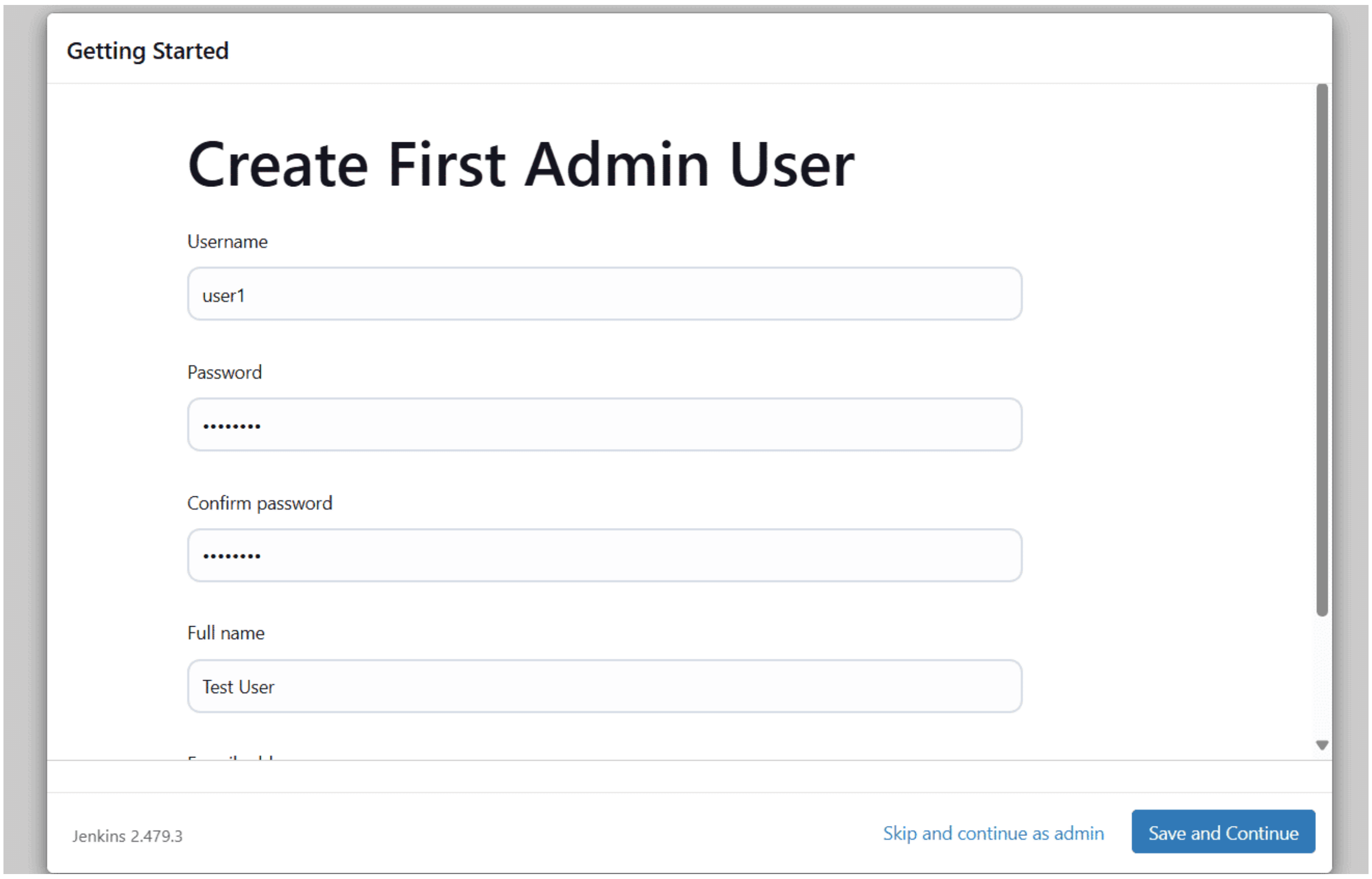
✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline Graph View
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication
LDAP	Email Extension	✓ Mailer	Dark Theme

```
** Matrix Project
** Resource Disposer
Workspace Cleanup
Ant
** OkHttp
** Durable Task
** Pipeline: Nodes and Processes
** Pipeline: SCM Step
** Pipeline: Groovy
** Pipeline: Job
** Jakarta Activation API
** Jakarta Mail API
** Apache HttpComponents Client 4.x API
** Instance Identity
Mailer
** Pipeline: Basic Steps
Gradle
** Pipeline: Milestone Step
** Pipeline: Build Step
** Pipeline: Groovy Libraries
** Pipeline: Stage Step
** Joda Time API
```

** - required dependency

Jenkins 2.479.3

Now, create the First Administrator User by providing a username, password, and a valid email for the admin account. Click *Save and Continue*.


Getting Started

Create First Admin User

Username

user1

Password

••••••••

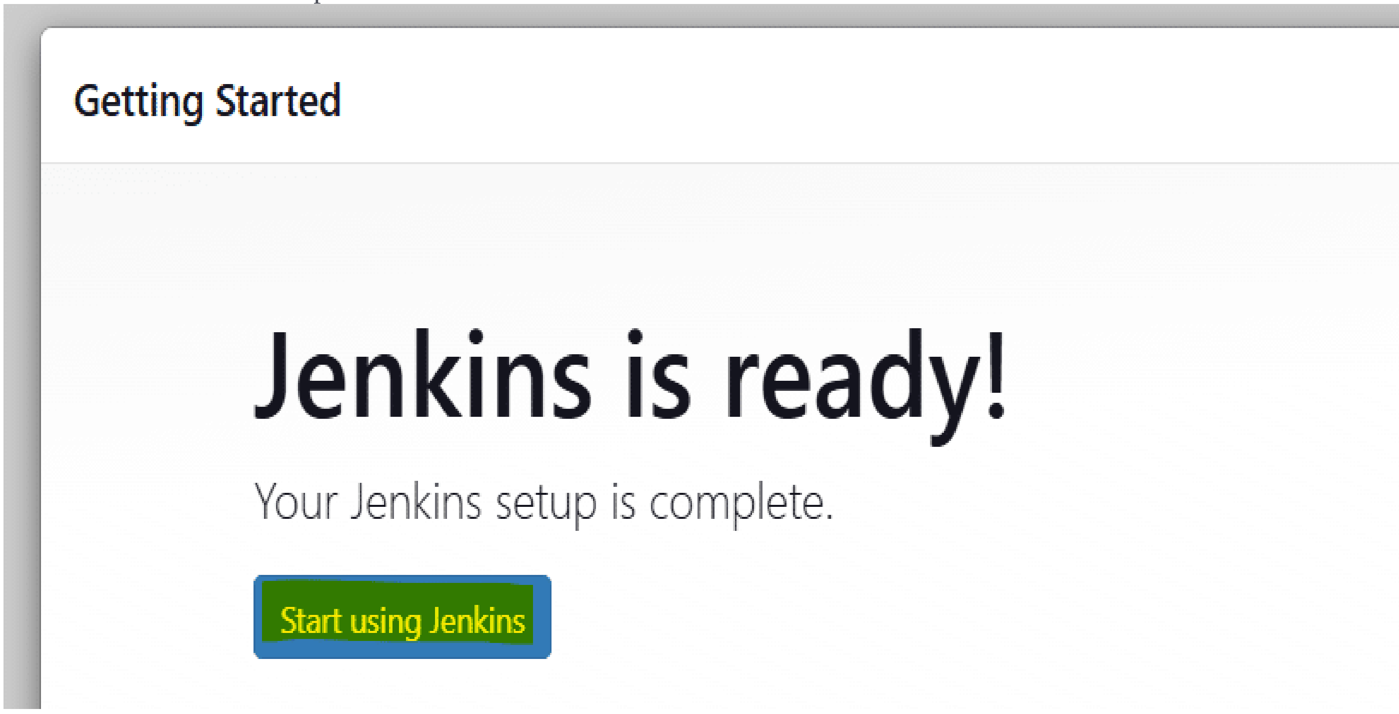
Confirm password

••••••••

Full name

Test User

Jenkins installation is complete.

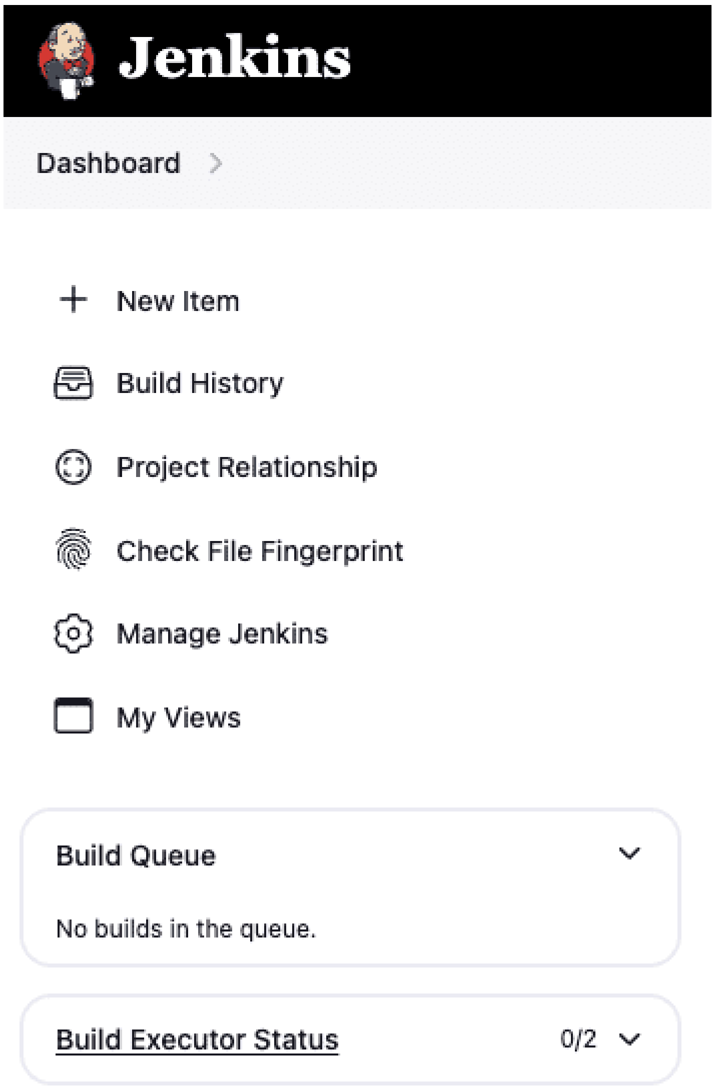

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

Step 2: Log in with admin credentials

After you log in to Jenkins, you can explore the Dashboard options.



No builds in the queue.

The Jenkins dashboard provides essential features like:

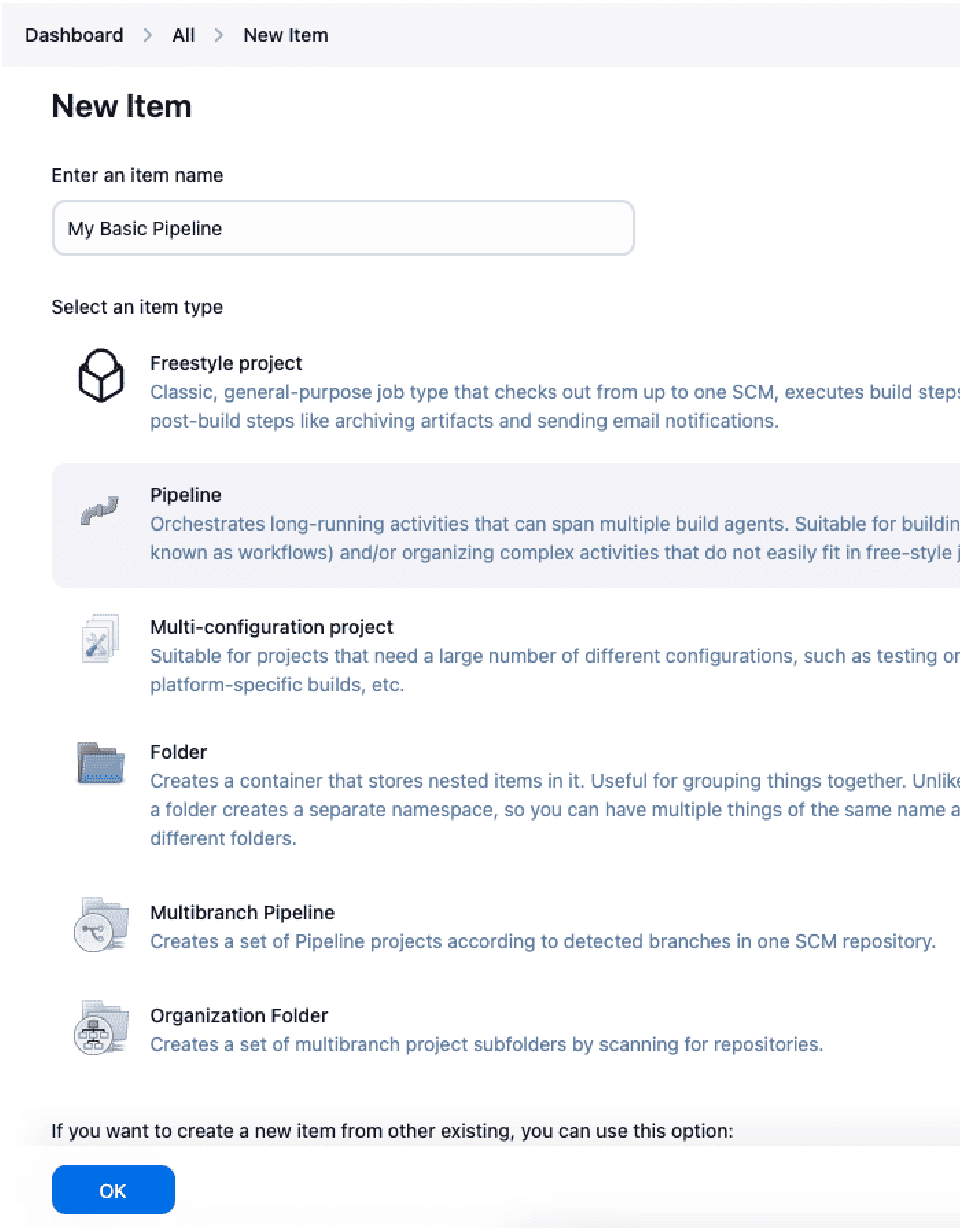
- **Build History** for tracking past builds
- **Manage Jenkins** for configuring system settings
- **Build Queue** to monitor pending builds

You can create and manage Jenkins jobs to automate your software processes and set up distributed builds for scalability. The dashboard has options to configure agents and enhance build performance.

The **Build Executor Status** shows the availability of resources for running builds, allowing you to manage workloads efficiently. With these options, you can streamline and optimize your continuous integration workflow.

Step 3: Create a new item (Pipeline)

From the Jenkins Dashboard, click on the + *New Item* option to open the form below.



Dashboard > All > New Item

New Item

Enter an item name

My Basic Pipeline

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps
post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for buildin
known as workflows) and/or organizing complex activities that do not easily fit in free-style j

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing or
platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike
a folder creates a separate namespace, so you can have multiple things of the same name a
different folders.

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

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.

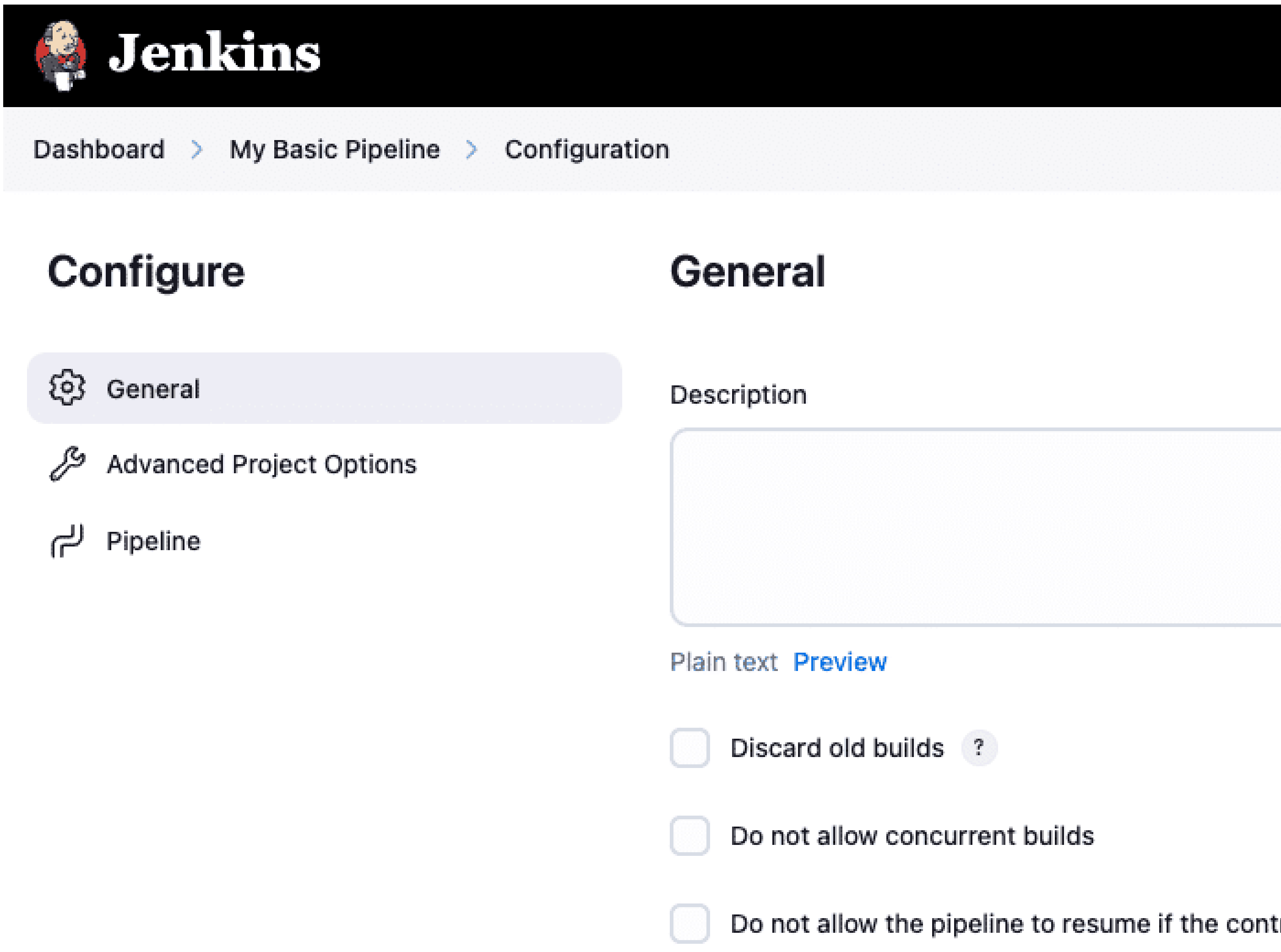
If you want to create a new item from other existing, you can use this option:

Here, you have different options to select (Freestyle project, pipeline, multi-configuration project, and more). Since we will build a pipeline, select "pipeline" as the item type and give it a suitable name, e.g., "My Basic Pipeline."

Click on *OK* to proceed with the pipeline configurations.

Step 4: Set the general pipeline configuration options

The *Configure* page for a Jenkins pipeline provides several configuration options. Under *General*, you can add a description, discard old builds, and control concurrent builds or allow the pipeline to resume after a controller restart.


Jenkins
Dashboard > My Basic Pipeline > Configuration
Configure
General
Advanced Project Options
Pipeline
General
Description
Plain text Preview
Discard old builds ?
Do not allow concurrent builds
Do not allow the pipeline to resume if the cont

You can also link a GitHub project and set pipeline speed/durability or preserve build stashes. The ***Build Triggers*** section allows you to trigger builds based on other project builds, schedule periodic builds, use GitHub hooks, or poll the SCM for changes.

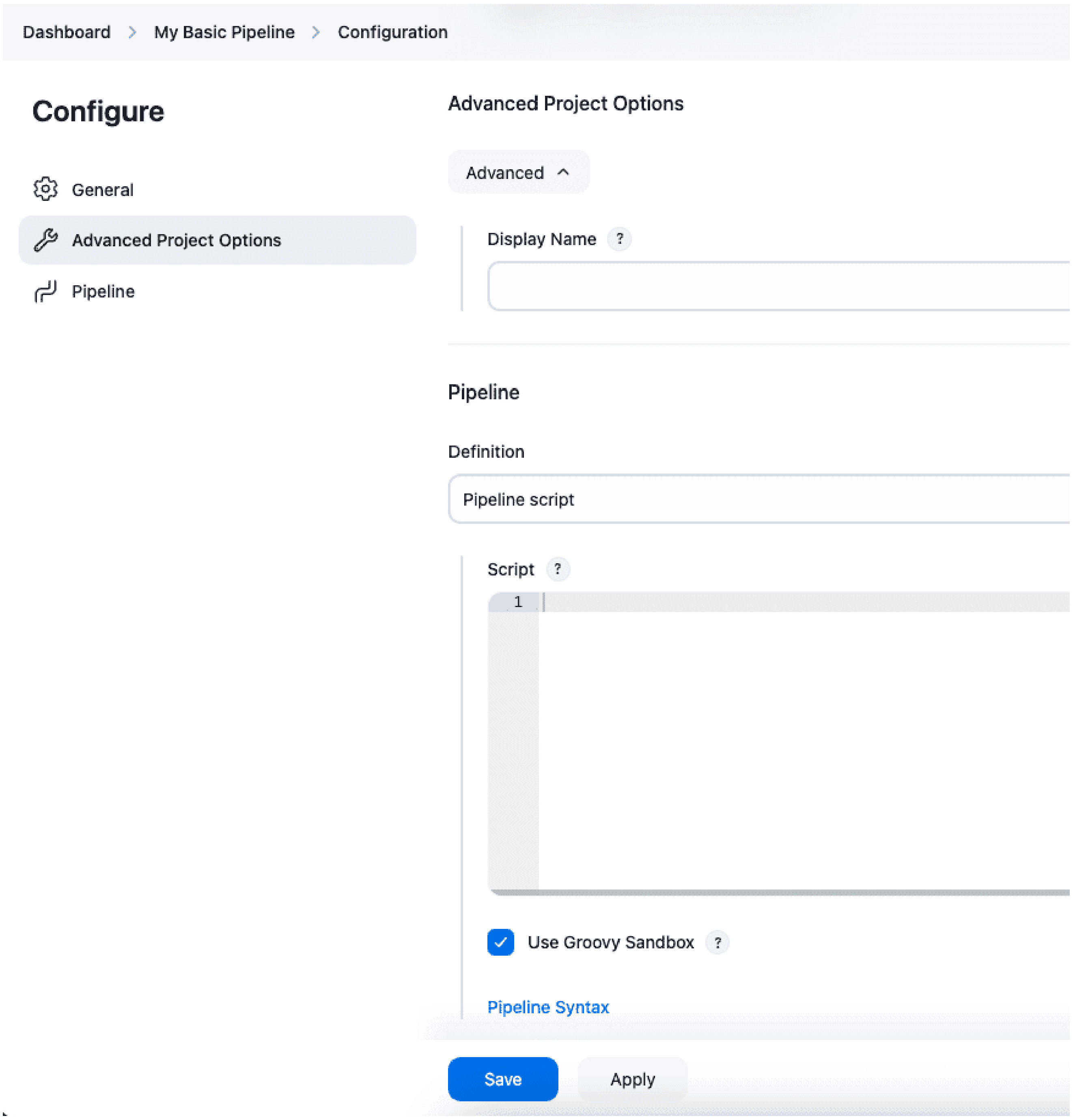
In the ***Pipeline*** section, you can define the pipeline directly using Groovy scripts or fetch it from source control. You can also enable the Groovy sandbox for added security and resiliency.

Step 5: Define the pipeline

At this stage, the advanced project options don't have much to configure, so we can skip them for now and focus on the core pipeline setup.

You will see two options in the Definition field:

1. **Pipeline Script**
2. **Pipeline Script from SCM**


Dashboard > My Basic Pipeline > Configuration

Configure

- General
- Advanced Project Options
- Pipeline

Advanced Project Options

Advanced ^

Display Name ?

Pipeline

Definition

Pipeline script

Script ?

Use Groovy Sandbox ?

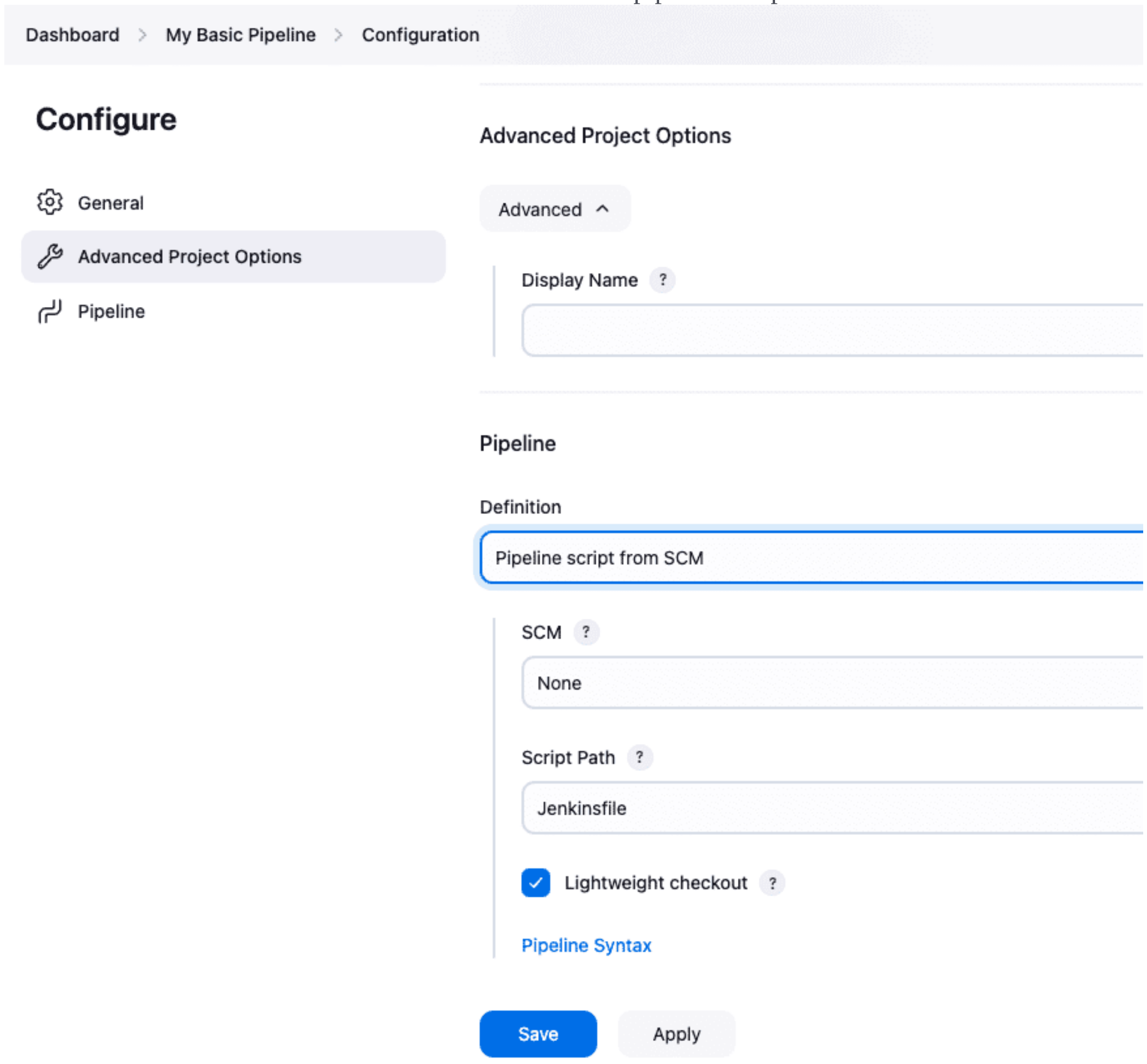
Pipeline Syntax

Save Apply

Selecting the ***Pipeline script"*** allows you to write the declarative pipeline code directly in the provided text box. This script defines the steps Jenkins will execute, such as building, testing, and deploying your software.

For most beginners, using the pipeline block is the simplest and quickest way to start with Jenkins pipelines.

Alternatively, you can choose the ***Pipeline script from SCM*** option to store and manage the pipeline script in a source control management (SCM) system like Git. The scripted pipeline approach benefits teams that need version control and collaboration on the pipeline script.


Dashboard > My Basic Pipeline > Configuration

Configure

General
Advanced Project Options
Pipeline

Advanced Project Options

Advanced ^

Display Name ?

Pipeline

Definition
Pipeline script from SCM

SCM ?
None

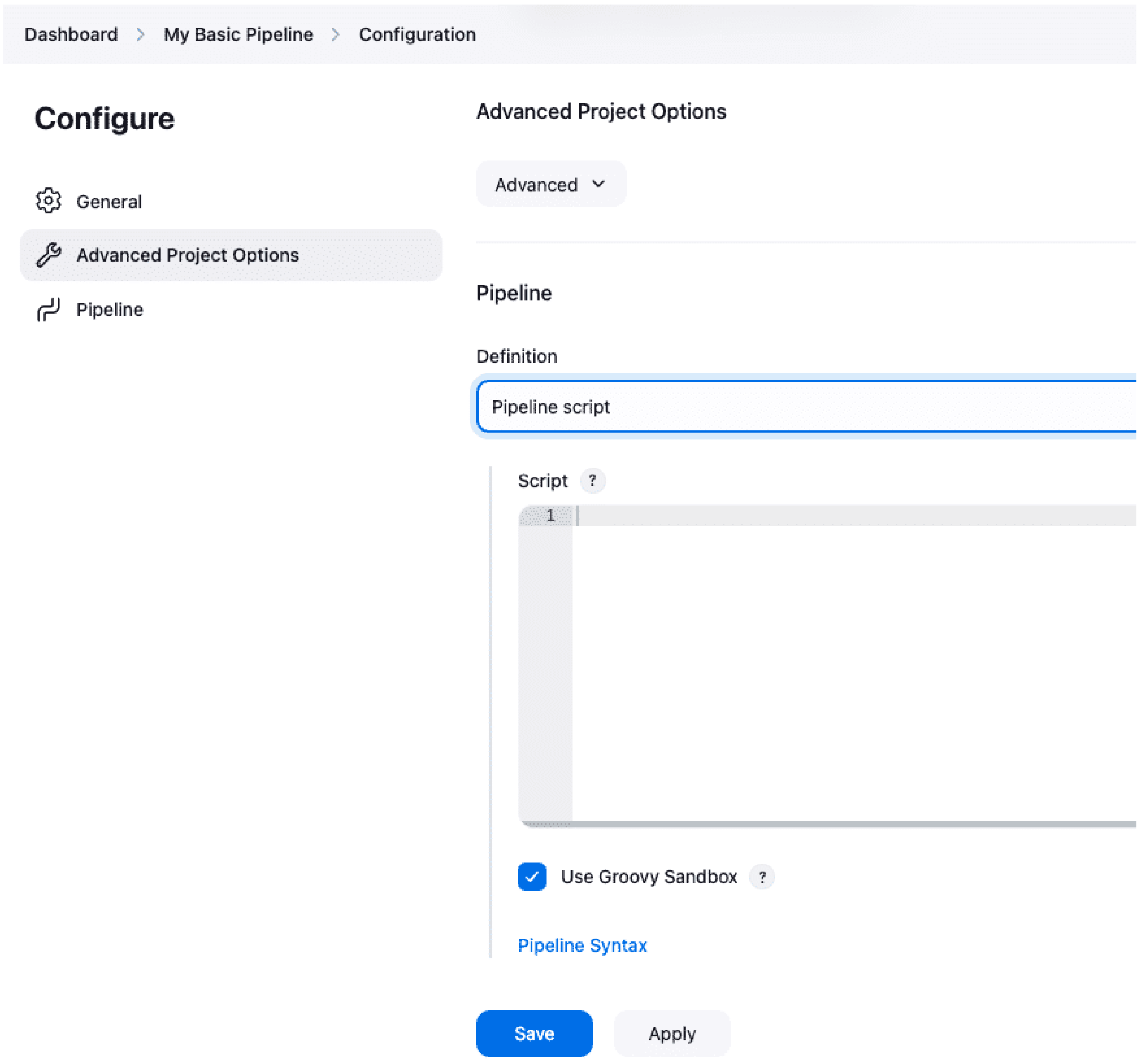
Script Path ?
Jenkinsfile

✓ Lightweight checkout ?

Pipeline Syntax

Save Apply

However, for this example, we'll stick with the *Pipeline script* option to keep things straightforward and focus on directly defining a simple pipeline script in the Jenkins UI.



Step 6: Script the pipeline

Jenkins pipelines are defined using Groovy script, which is a flexible language for defining build steps and stages. The script allows developers to describe build, test, and deployment tasks as code. Jenkins executes such scripts automatically to streamline different CI/CD workflows.

The code below represents a basic pipeline structure that starts with the declaration. The “agent any” line tells Jenkins to execute the pipeline on any available agent or node. This is the most basic setup for picking any available executor for the job.

The agent defines where the pipeline will run, ensuring the execution environment is set.

```
pipeline {
    agent any
}
```

The updated script introduces the stages section to organize the pipeline into different steps. Each stage defines a logical unit of work, such as building, testing, or deploying.

The stages block helps structure the pipeline, making it easy to follow and troubleshoot. You can define multiple stages, each with its own actions to execute.

```
pipeline {
    agent any

    stages {
          // define stages here
    }
}
```

The post section handles actions after the pipeline completes, whether it succeeds or fails. In this section, you can define custom actions based on the pipeline's outcome.

The success block runs after a successful execution, while the failure block runs if the pipeline fails. This section is essential for managing results and taking appropriate actions after the pipeline runs.

```
pipeline {
    agent any

    stages {
                // define stages here
     }

    post {
        success {
            // define next action after success
        }
        failure {
            // define next action after failure
        }
    }
}
```

In the complete example, the stages block defines specific stages like ‘Build,’ ‘Test,’ and ‘Deploy.’ Each stage contains steps that describe actions like printing messages or running shell commands.

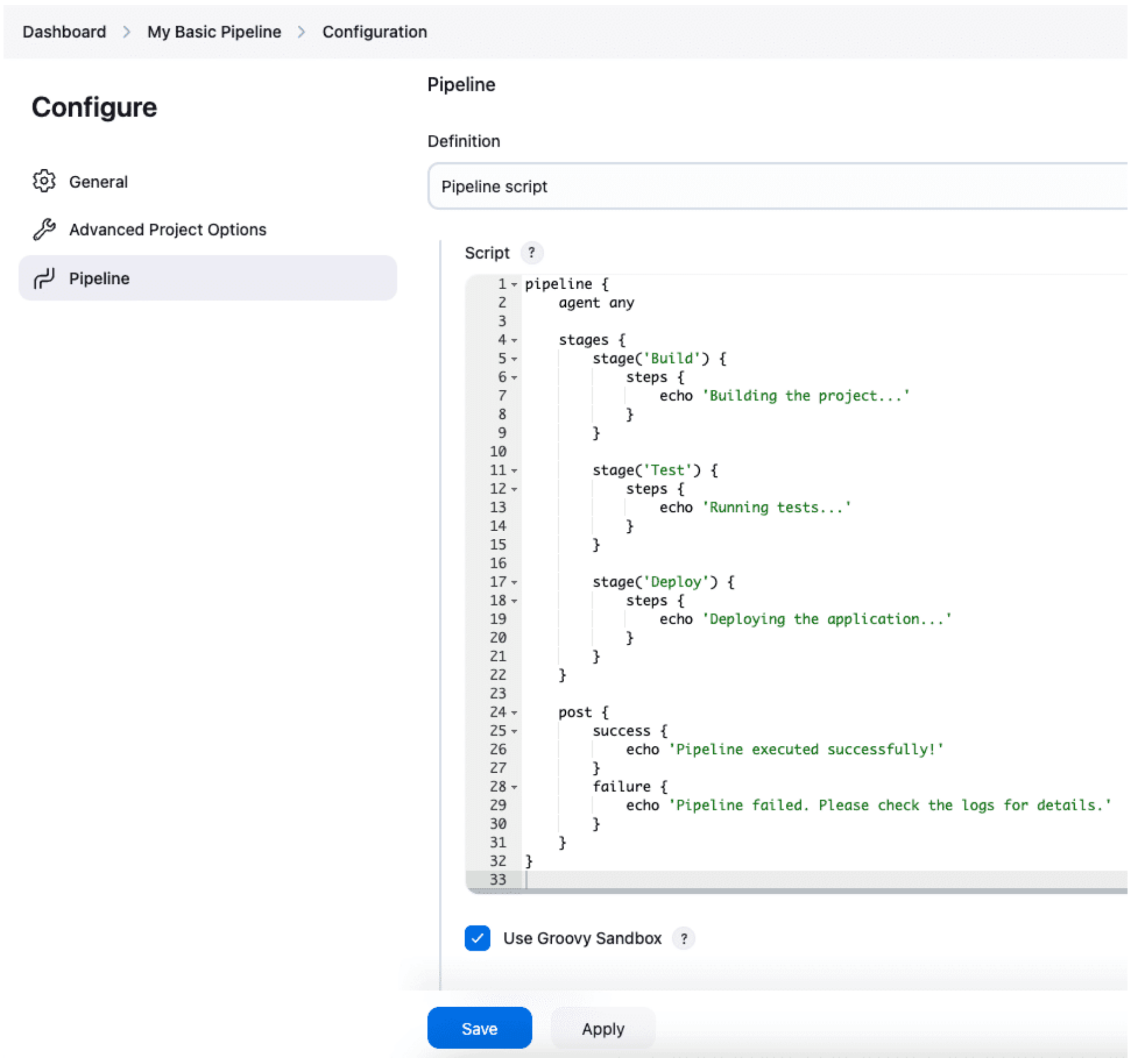
The post section provides feedback on the pipeline’s result, with custom messages for success or failure.

This example demonstrates how Jenkins automates tasks, providing clear feedback on execution outcomes.

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building the project...'
                         // bash/shell commands
            }
        }

        stage('Test') {
            steps {
                echo 'Running tests...'
                         // bash/shell commands
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying the application...'
                         // bash/shell commands
            }
        }
    }
    post {
        success {
            echo 'Pipeline executed successfully!'
        }
        failure {
            echo 'Pipeline failed. Please check the logs for details.'
        }
    }
}
```

To save this script in Jenkins, click the “Save” button at the bottom of the configuration page. Once saved, Jenkins will use the script for the pipeline whenever the job is triggered.

You can always come back to edit the script as needed. Make sure to save your progress to ensure the pipeline runs as expected.



Pipeline

Definition

Pipeline script

Script

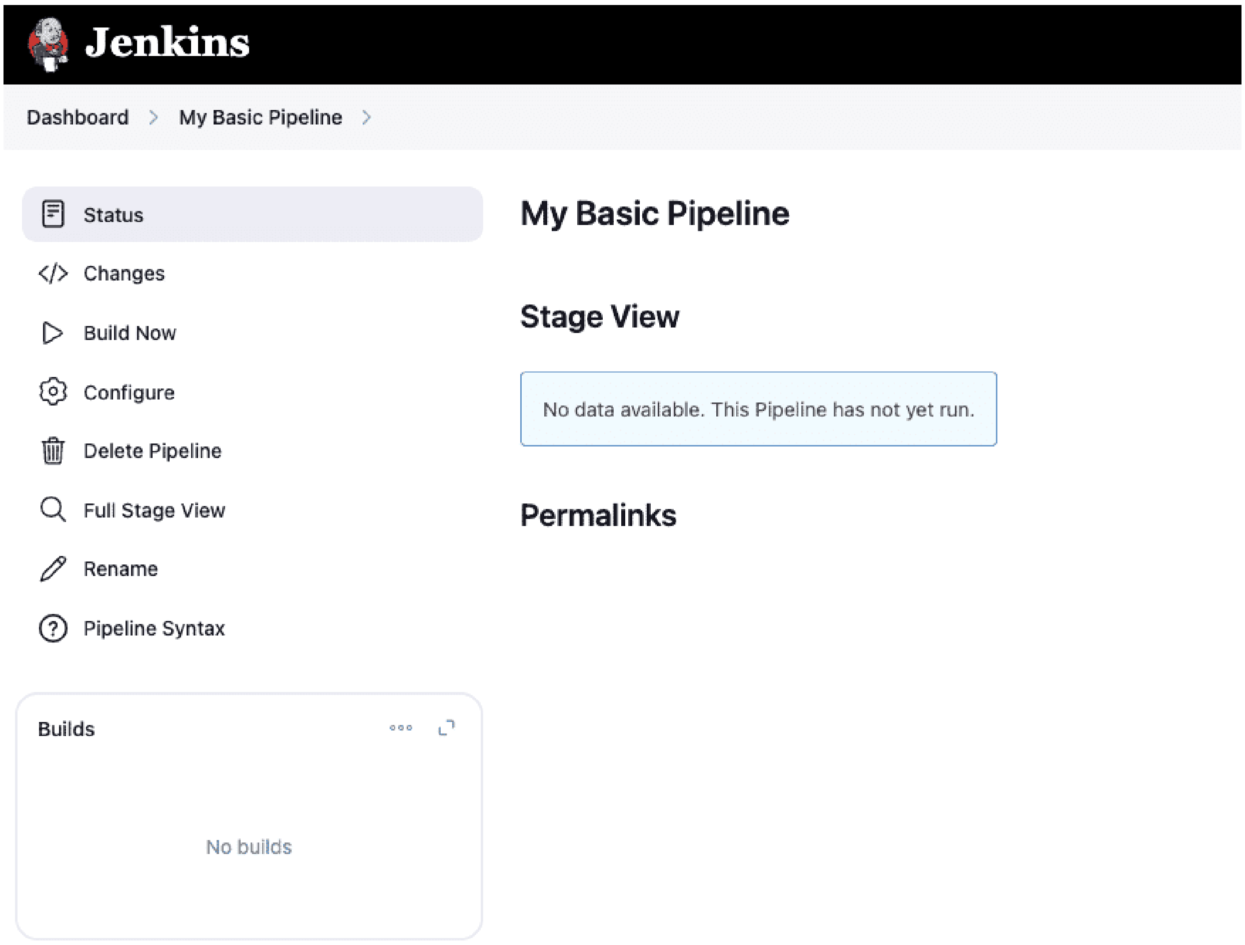
```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building the project...'
            }
        }

        stage('Test') {
            steps {
                echo 'Running tests...'
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying the application...'
            }
        }
    }

    post {
        success {
            echo 'Pipeline executed successfully!'
        }
        failure {
            echo 'Pipeline failed. Please check the logs for details.'
        }
    }
}
```

✓ Use Groovy Sandbox

Step 7: Save the pipeline

After saving the pipeline, visit the main dashboard of the configured pipeline to see its status. You can access this by clicking on the job name from the Jenkins home page.

Here, you’ll find options like ‘Build Now,’ ‘Configure,’ and ‘Delete Project.’ These options allow you to trigger builds, adjust settings, and manage your pipeline.



My Basic Pipeline

Stage View

No data available. This Pipeline has not yet run.

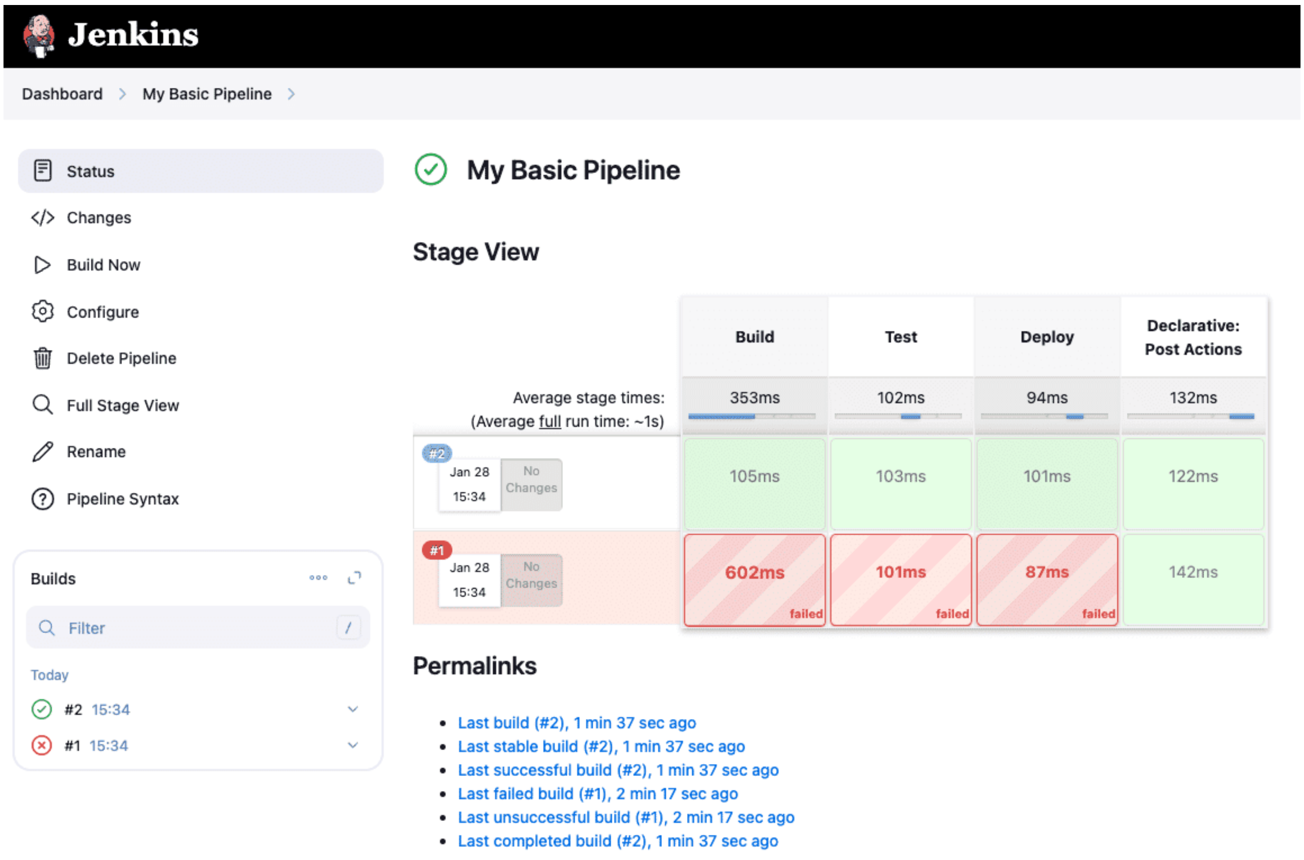
Permalinks

No builds

Step 8: Run the pipeline

To run the pipeline, click the “Build Now” button on the dashboard page. This will trigger the pipeline’s execution according to the defined script. You can monitor the build’s progress in real time and review past executions.

The “Builds” block lists current and past builds, allowing you to revisit previous runs and inspect their outcomes.



My Basic Pipeline

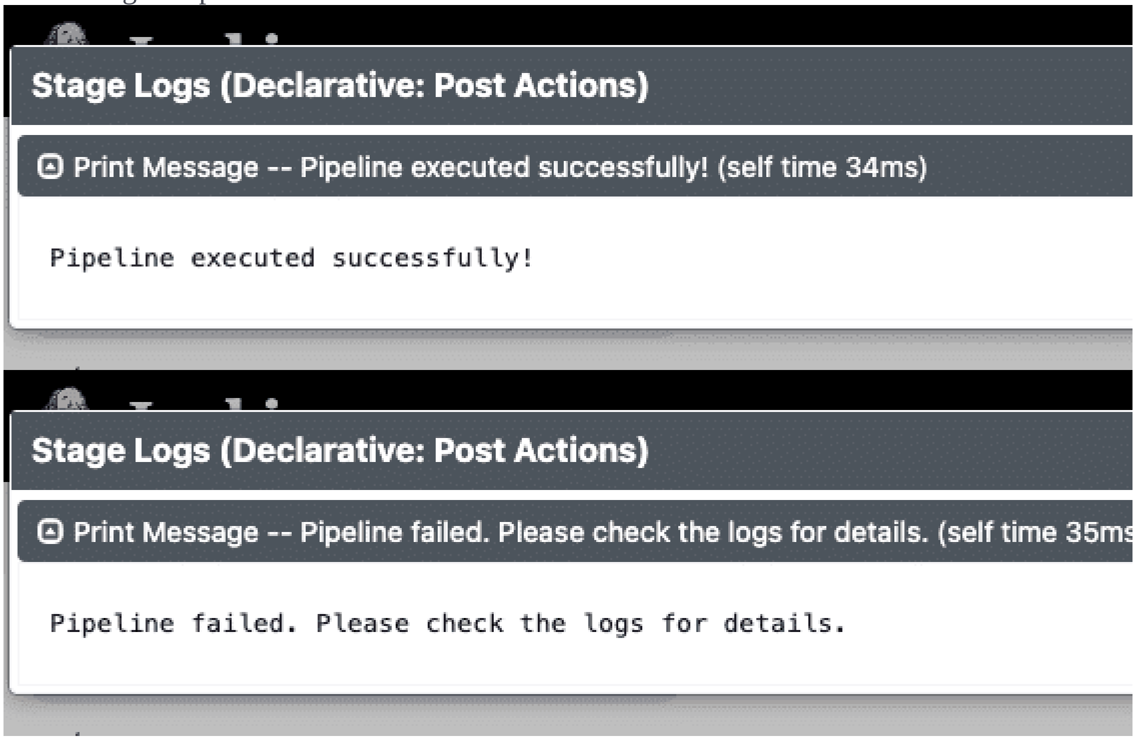
Stage View

	Build	Test	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~1s)	353ms	102ms	94ms	132ms
#2 Jan 28 15:34 No Changes	105ms	103ms	101ms	122ms
#1 Jan 28 15:34 No Changes	602ms failed	101ms failed	87ms failed	142ms

Permalinks

- Last build (#2), 1 min 37 sec ago
- Last stable build (#2), 1 min 37 sec ago
- Last successful build (#2), 1 min 37 sec ago
- Last failed build (#1), 2 min 17 sec ago
- Last unsuccessful build (#1), 2 min 17 sec ago
- Last completed build (#2), 1 min 37 sec ago

As shown above, post actions are executed successfully when the pipeline finishes and both success and failure logs are printed.


Stage Logs (Declarative: Post Actions)

Print Message -- Pipeline executed successfully! (self time 34ms)

```
Pipeline executed successfully!
```

Stage Logs (Declarative: Post Actions)

Print Message -- Pipeline failed. Please check the logs for details. (self time 35ms

```
Pipeline failed. Please check the logs for details.
```

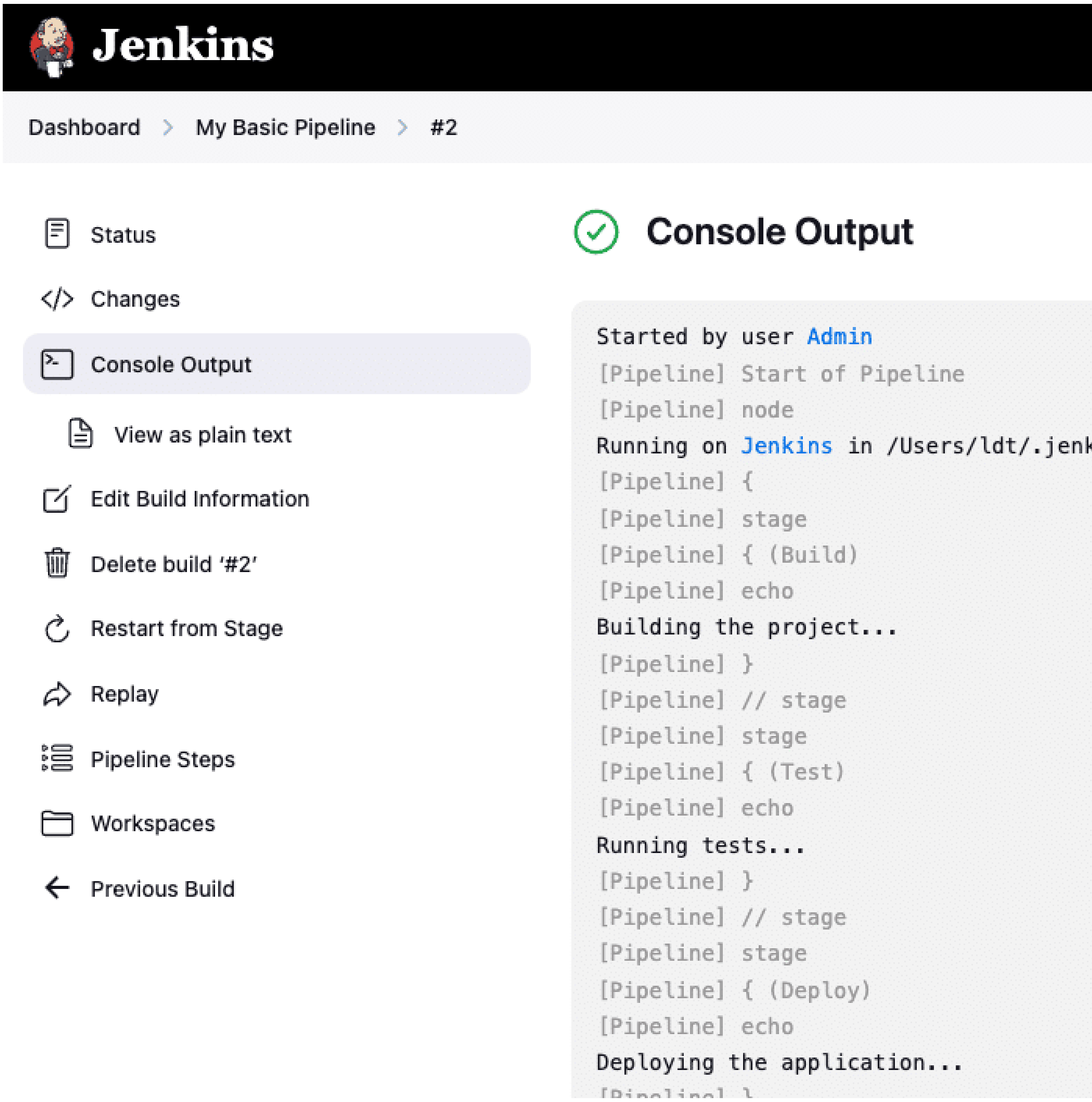
After the pipeline executes, the post actions configured earlier will run based on the outcome. If the pipeline succeeds, the success block prints the success message, and if it fails, the failure block prints the failure message.

These logs give you insight into what happened during the execution. Clicking on the specific build blocks in the ‘Build History’ lets you view the results in detail.

Step 9: Access the logs

To view detailed logs for each pipeline run, click on a specific build number under the “Build History” section. This opens the build’s log, showing every executed action and command. Logs provide crucial feedback for debugging and optimizing the pipeline.

You can see whether each step in the pipeline succeeded or failed, and the exact output for each step.



```
Started by user Admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /Users/ldt/.jenk
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
Building the project...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Running tests...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying the application...
[Pipeline] }
```

Here are the available pipeline management options:

- **Status** – View the current status of the pipeline execution.
- **Changes** – Check recent changes that triggered the pipeline.
- **Console Output** – View detailed logs of the pipeline execution.
- **View as Plain Text** – See the console output in plain text format.
- **Edit Build Information** – Modify details about the current pipeline run.

- **Delete build ‘#2’** – Remove the current build from Jenkins.
- **Restart from Stage** – Re-run the pipeline from a specific stage.
- **Replay** – Re-execute the pipeline without modifying the script.
- **Pipeline Steps** – Inspect and analyze the execution steps of the pipeline.
- **Workspaces** – Access the workspace directory used by the pipeline.
- **Previous Build** – Navigate to the previous build and review the details.

These pipeline management options provide complete control over pipeline execution. You can efficiently manage, track, and troubleshoot your pipeline using these options to ensure smooth and continuous integration.