–

Step-by-step walkthrough for multiconfiguration (Matrix Project) in Jenkins

**Jenkins LAB**

PREPARED BY:

PREPARED FOR:

**Version:** 1.0  
**Department:** Architect Department  
**Delivery:** End of Month

Table of Contents

[**1.** Executive Summary 3](#_Toc209600240)

[2. Create Multiconfiguration Project 3](#_Toc209600241)

[Step 1 – Create New Item 3](#_Toc209600242)

[Step 2 – General Settings 4](#_Toc209600243)

[Step 3 – Define Axes 4](#_Toc209600244)

[Step 4 – Add Build Step 6](#_Toc209600245)

[Step 5 – Save and Build 6](#_Toc209600246)

[Step 6 – View Results 8](#_Toc209600247)

[3. Matrix Project for Selenium Tests 9](#_Toc209600248)

[Step 1: Create Matrix Project 10](#_Toc209600249)

[Step 2: Add Axes 11](#_Toc209600250)

[Step 3: Build Step – Selenium Test Command 12](#_Toc209600251)

[Maven / TestNG Example 15](#_Toc209600252)

# Executive Summary

In Jenkins, when you create a **new item (project/job)**, you see different **project item types**. Each has its own purpose and use case. Here’s a quick breakdown:

**🔹 1. Freestyle Project**

* **Usage**:
  + The most common and flexible project type.
  + Allows you to configure build steps (shell scripts, Windows batch commands, etc.).
  + Can integrate with version control (Git, SVN, etc.), schedule builds, and trigger post-build actions.
* **Best for**: Simple CI/CD pipelines, running scripts, and small automation tasks.

**🔹 2. Pipeline**

* **Usage**:
  + Lets you define your build, test, and deploy process as **code** using a Jenkinsfile (Groovy-based).
  + Supports complex workflows with multiple stages and steps.
  + Easily integrates with Git for versioned pipeline-as-code.
* **Best for**: End-to-end CI/CD automation, multi-stage deployments, and modern DevOps workflows.

**🔹 3. Multibranch Pipeline**

* **Usage**:
  + Automatically creates and manages a pipeline for **each branch** in your repository.
  + Detects branches with a Jenkinsfile and builds them.
  + Great for feature-branch development and pull request builds.
* **Best for**: Git-based projects with multiple branches (GitHub, Bitbucket, GitLab).

**🔹 4. Folder**

* **Usage**:
  + Helps organize jobs into groups (like projects, teams, or environments).
  + Allows permissions and configurations at the folder level.
* **Best for**: Large Jenkins setups with many jobs to organize.

**🔹 5. Multi-Configuration Project (Matrix Project)**

* **Usage**:
  + Used for running the same build in multiple environments or configurations.
  + Example: Testing across different OS versions, JDK versions, or databases.
* **Best for**: Cross-platform builds and parallel testing with different parameters.

**Jenkins Project Types with Usage**

|  |  |
| --- | --- |
| Project Type | Usage / Purpose |
| Freestyle Project | Simple and flexible jobs. Run shell/batch commands, integrate with SCM, schedule builds. |
| Pipeline | Define CI/CD as code in a Jenkinsfile (Groovy). Supports multi-stage workflows. |
| Multibranch Pipeline | Automatically discovers branches with a Jenkinsfile and builds each one separately. |
| Multi-Configuration (Matrix Project) | Run the same job with multiple parameters (e.g., environments, OS, browsers). Useful for parallel testing. |

# Create Multiconfiguration Project

**🎯 Scenario**

You want to test your application with **two different JDK versions** and on **two operating systems**.

* JDK: OpenJDK 8 and OpenJDK 11
* OS: Linux and Windows

That gives you **4 combinations** Jenkins will automatically create and run:

1. Linux + JDK 8
2. Linux + JDK 11
3. Windows + JDK 8
4. Windows + JDK 11

## Step 1 – Create New Item

* Go to Jenkins Dashboard → **New Item**
* Enter a name (e.g., Matrix-Demo)
* Select **Multi-configuration project**
* Click **OK**

A screenshot of a computer

AI-generated content may be incorrect.

## Step 2 – General Settings

* Add a description:

Demo Matrix job that runs echo command for different combinations

**A blue rectangle with white border

AI-generated content may be incorrect.**

## Step 3 – Define Axes

* **Scroll down to Configuration Matrix section.**
* **Click Add Axis → User-defined Axis**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Example setup:**
  + **Name: ENV**
  + **Values: DEV TEST PROD  
    (This will create 3 builds: one for DEV, one for TEST, one for PROD)**

A blue line on a white background

AI-generated content may be incorrect.

## Step 4 – Add Build Step

* In the **Build** section, click **Add build step → Windows batch**
* Add this simple command:

echo "Running build for environment = $ENV"

A screenshot of a computer

AI-generated content may be incorrect.

## Step 5 – Save and Build

* Click **Save**

A screenshot of a computer

AI-generated content may be incorrect.

* Click **Build Now**

A screenshot of a computer

AI-generated content may be incorrect.

## Step 6 – View Results

* Jenkins will automatically create **separate builds** for each axis value:
  + Build for DEV
  + Build for TEST
  + Build for PROD
* In the console log, you’ll see output like:

A screenshot of a computer

AI-generated content may be incorrect.

# Matrix Project for Selenium Tests

A diagram of a process

AI-generated content may be incorrect.

## Step 1: Create Matrix Project

* Jenkins → **New Item** → Name: MatrixSeleniumDemo → **Multi-configuration project** → OK

A screenshot of a computer

AI-generated content may be incorrect.

## Step 2: Add Axes

Testers usually test across **browsers** and **environments**:

* **Axis 1: Browser** → Chrome Firefox
* **Axis 2: Environment** → QA UAT

This automatically creates **4 combinations**:

1. Chrome-QA
2. Chrome-UAT
3. Firefox-QA
4. Firefox-UAT

A white sheet with blue lines

AI-generated content may be incorrect.

## Step 3: Build Step – Selenium Test Command

Assuming you have **Maven-based Selenium tests**, your **Execute Windows batch command** could be:

echo Running Selenium tests on %Browser% - %Environment%

:: Set browser environment variable

set BROWSER=%Browser%

set ENVIRONMENT=%Environment%

:: Run Maven TestNG tests (example)

echo “ mvn clean test -Dbrowser=%BROWSER% -Denvironment=%ENVIRONMENT% “

echo Selenium tests completed on %Browser% - %Environment%

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

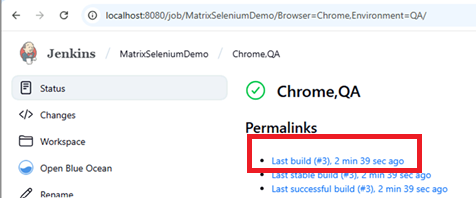
AI-generated content may be incorrect.

Run Build Now

A screenshot of a computer

AI-generated content may be incorrect.

Click on QA for chrome and click Last Build



Check Console Output

A screenshot of a computer

AI-generated content may be incorrect.

* Note In Actual Environment you have below sample

## Maven / TestNG Example

In your **TestNG testng.xml or Selenium code**, read the parameters:

String browser = System.getProperty("browser");

String env = System.getProperty("environment");

// Initialize WebDriver based on browser

if(browser.equals("Chrome")){

WebDriver driver = new ChromeDriver();

}else if(browser.equals("Firefox")){

WebDriver driver = new FirefoxDriver();

}

// Navigate based on environment

if(env.equals("QA")){

driver.get("https://qa.example.com");

}else if(env.equals("UAT")){

driver.get("https://uat.example.com");

}

// Run tests...

driver.quit();