

Lesson-End Project

Implementing Job Chaining in Jenkins

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.

Tools required: Jenkins and GitHub

Prerequisites: You must have Jenkins and GitHub access in the lab to proceed.

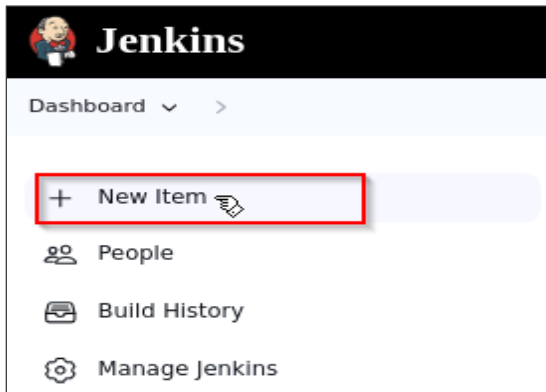
Expected deliverables: A Jenkins pipeline demonstrating job chaining for continuous integration

Steps to be followed:

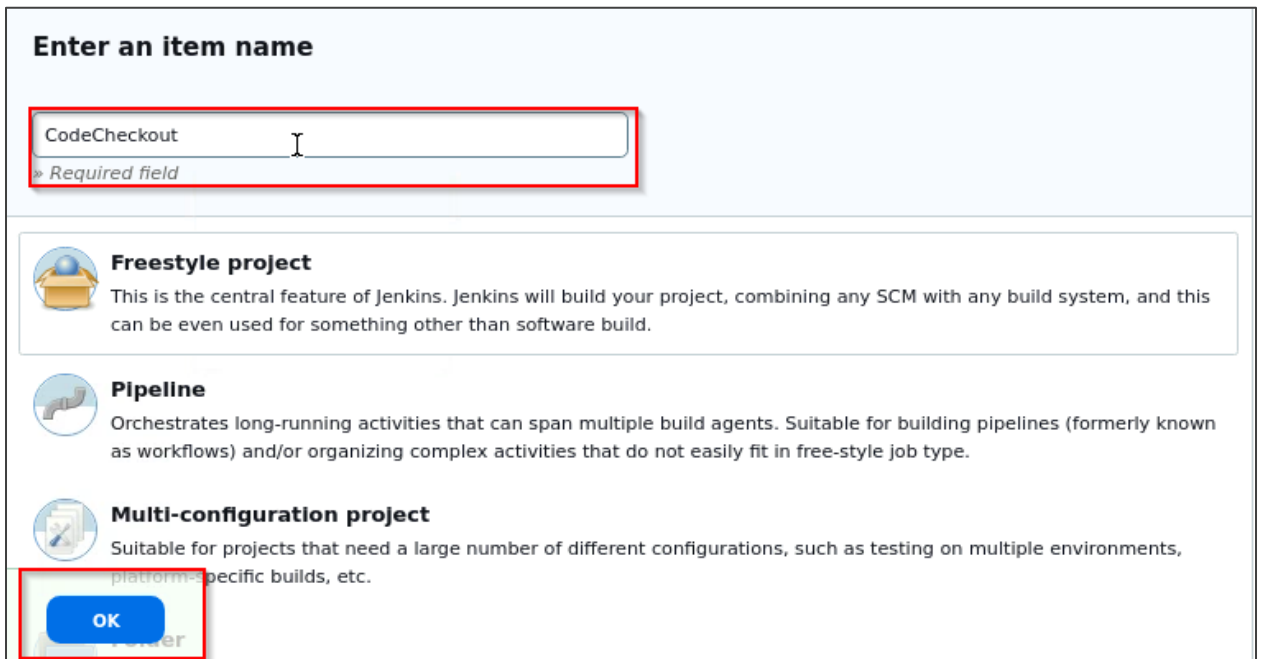
1. Create jobs in a Jenkins pipeline
2. Install the build pipeline plugin in Jenkins
3. Trigger the execution of jobs

Step 1: Create jobs in a Jenkins pipeline

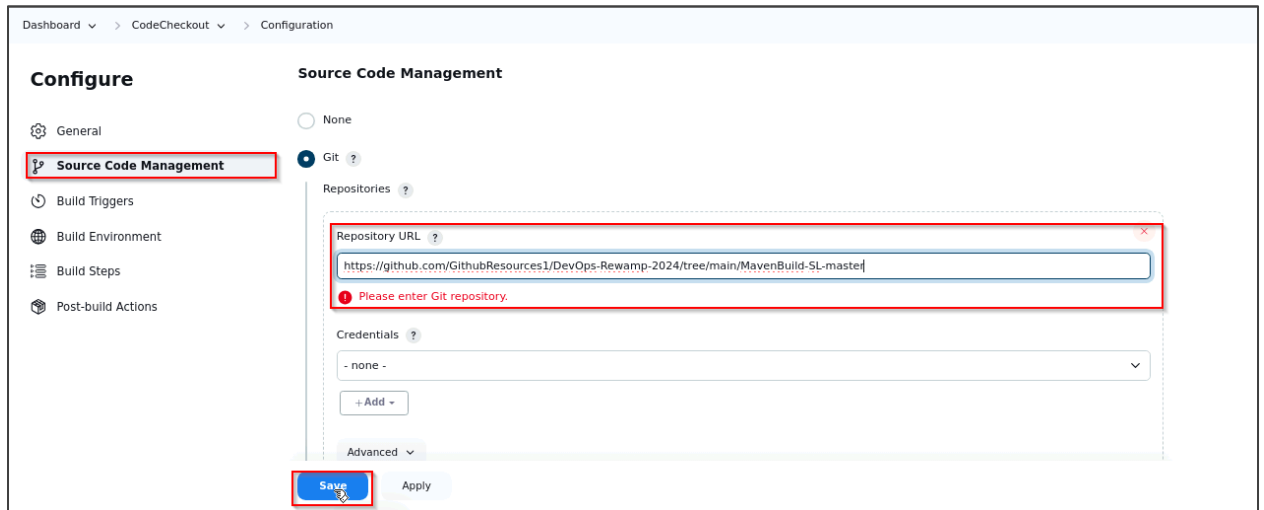
1.1 Click on the **New item** box to create the first job



1.2 Enter the item name as **CodeCheckout**, select the type as **Freestyle project**, and click on **OK**

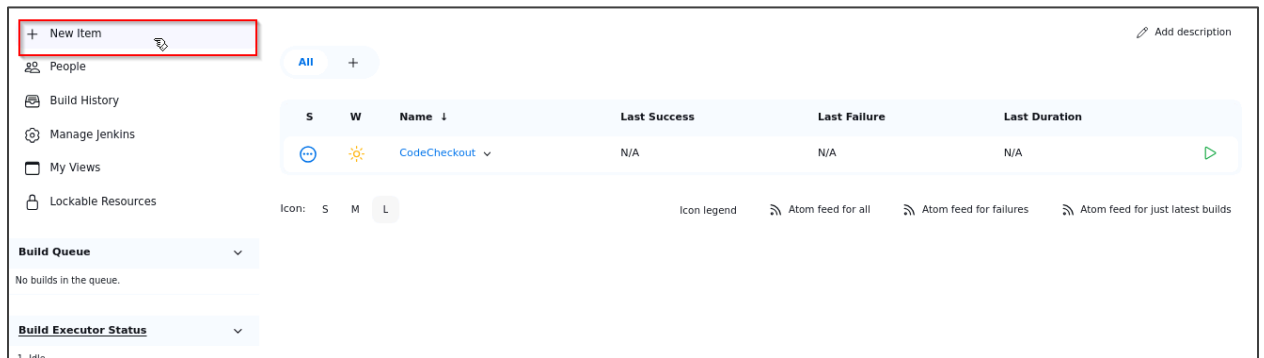
A screenshot of the 'Enter an item name' form in Jenkins. The form has a title 'Enter an item name' at the top. Below the title is a text input field containing the text 'CodeCheckout'. The input field is highlighted with a red rectangular box. Below the input field is a small red text label that says '» Required field'. Below the input field are three radio button options. The first option is 'Freestyle project' with a blue icon of a box and a checkmark. The second option is 'Pipeline' with a blue icon of a flowchart. The third option is 'Multi-configuration project' with a blue icon of a document and a checkmark. Below the radio button options is a blue button labeled 'OK'. The 'OK' button is highlighted with a red rectangular box.

1.3 Click on **Source Code Management** on the job configuration page, select Git, and under Repository URL, give the URL as **https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master**



The screenshot shows the Jenkins 'Configure' page for a job named 'CodeCheckout'. The 'Source Code Management' tab is selected in the left sidebar. Under 'Source Code Management', the 'Git' option is chosen. The 'Repository URL' field is highlighted with a red box and contains the URL 'https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master'. Below it, a red error message states 'Please enter Git repository.' The 'Credentials' dropdown is set to '- none -'. At the bottom, the 'Save' button is highlighted with a red box.

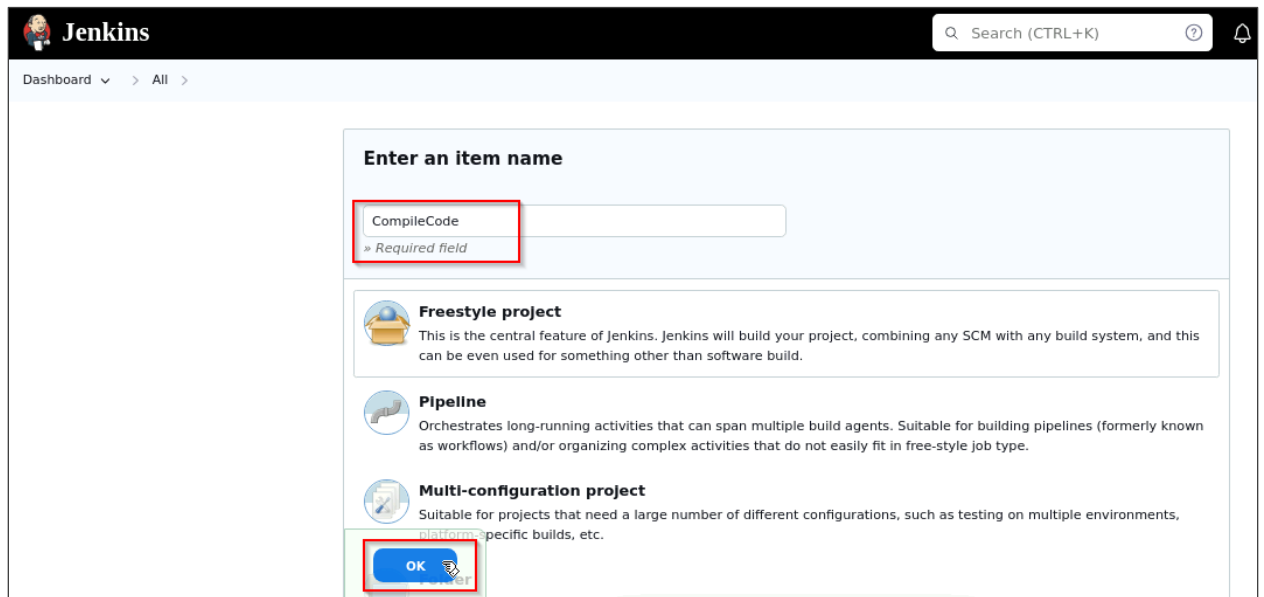
1.4 Go to the Jenkins dashboard and click on the **New item** tab to create the second job



The screenshot shows the Jenkins dashboard. The 'New Item' button is highlighted with a red box in the top left. The main area displays a table of existing jobs. The table has columns for 'S' (Status), 'W' (Icon), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. One job, 'CodeCheckout', is listed with a status of 'S' and a sun icon. Below the table, there are links for 'Icon legend', 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. On the left sidebar, there are links for 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Lockable Resources'. At the bottom, there are sections for 'Build Queue' and 'Build Executor Status'.

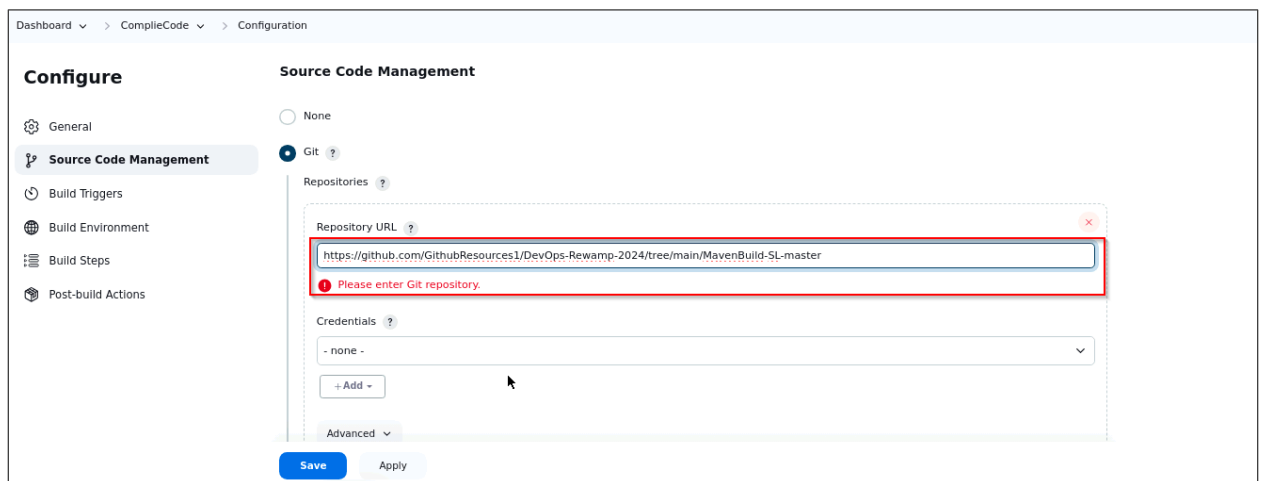
S	W	Name	Last Success	Last Failure	Last Duration
S	☀	CodeCheckout	N/A	N/A	N/A

1.5 Enter the item name as **CompileCode**, select the type as **Freestyle project**, and click on **OK**



The image shows the Jenkins 'Enter an item name' dialog. At the top, there's a search bar and a 'Jenkins' logo. Below the breadcrumb 'Dashboard > All', the main area is titled 'Enter an item name'. It contains a text input field with 'CompileCode' entered, which is highlighted with a red box. Below the input is a red-bordered box containing the text '» Required field'. Underneath, there are three project type options: 'Freestyle project' (with a box icon), 'Pipeline' (with a pipe icon), and 'Multi-configuration project' (with a multi-dot icon). Each option has a brief description. At the bottom left, there is a blue 'OK' button with a white checkmark, also highlighted with a red box.

1.6 Click on **Source Code Management** on the job configuration page, select Git, and under Repository URL give the URL as **<https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master>**



The image shows the Jenkins 'Configure' page for the 'CompileCode' job. The breadcrumb is 'Dashboard > CompileCode > Configuration'. On the left, there's a 'Configure' sidebar with options: 'General', 'Source Code Management' (selected), 'Build Triggers', 'Build Environment', 'Build Steps', and 'Post-build Actions'. The main area is titled 'Source Code Management' and shows 'None' selected with a radio button, and 'Git' selected with a radio button and a help icon. Below this, there's a 'Repositories' section with a red-bordered box around the 'Repository URL' input field. The URL entered is 'https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master'. Below the URL is a red error message: 'Please enter Git repository.' Below the error message is a 'Credentials' dropdown menu showing '- none -' and a '+ Add +' button. At the bottom, there are 'Save' and 'Apply' buttons.

1.7 Click on **Build Triggers**, select the option **Build after other projects are built**, and enter the name as **Codecheckout** in the **Projects to watch** field

Configure

- General
- Source Code Management
- Build Triggers**
- Build Environment
- Build Steps
- Post-build Actions

Build Triggers

☐ Trigger builds remotely (e.g., from scripts) ?

☒ Build after other projects are built ?

Projects to watch

Codecheckout

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☐ Always trigger, even if the build is aborted

☐ Build periodically ?

Save **Apply**

1.8 Click on **Build Steps** and select the option **Invoke top-level Maven targets**

Dashboard > CompileCode > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment**
- Build Steps**
- Post-build Actions

☐ Inspect build log for published build scans

☐ Terminate a build if it's stuck

☐ With Ant ?

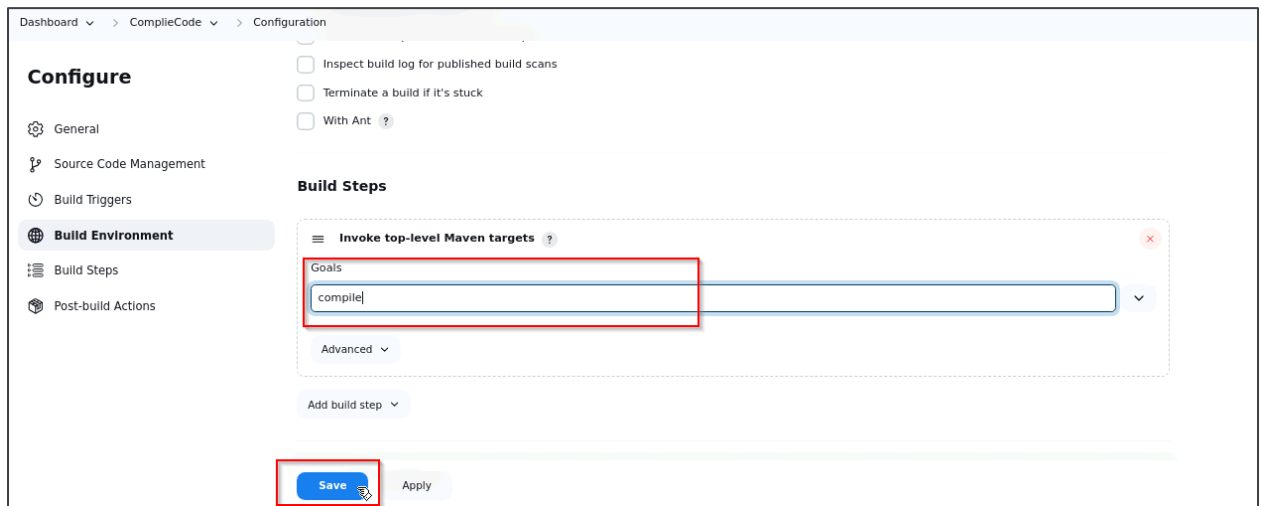
Build Steps

Add build step ^

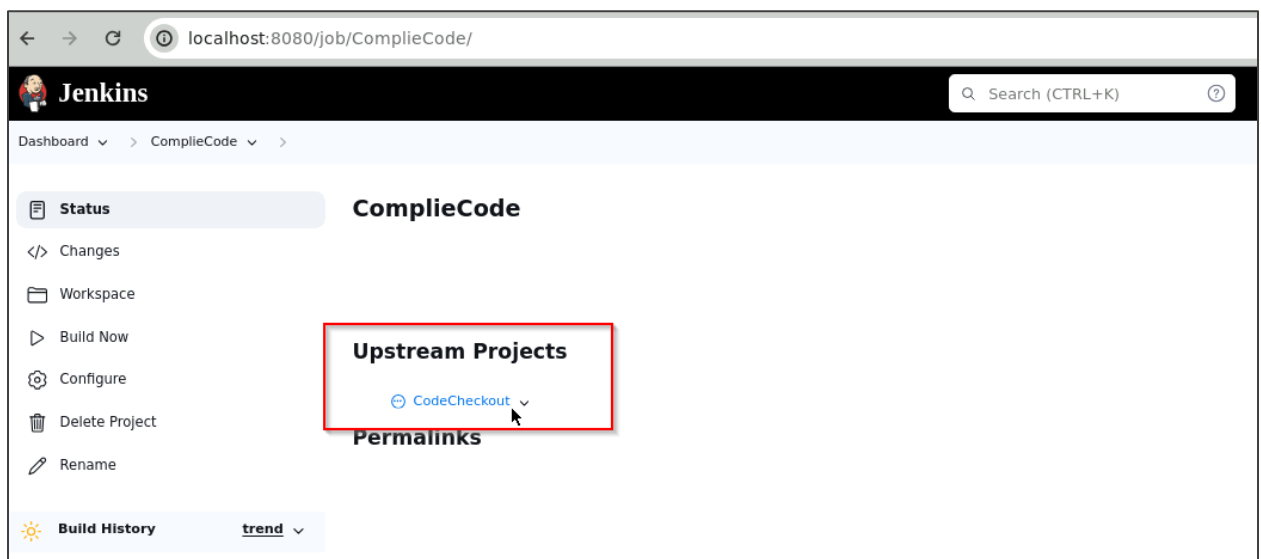
Filter

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets**
- Run with timeout
- Set build status to "pending" on GitHub commit
- Trigger/call builds on other projects

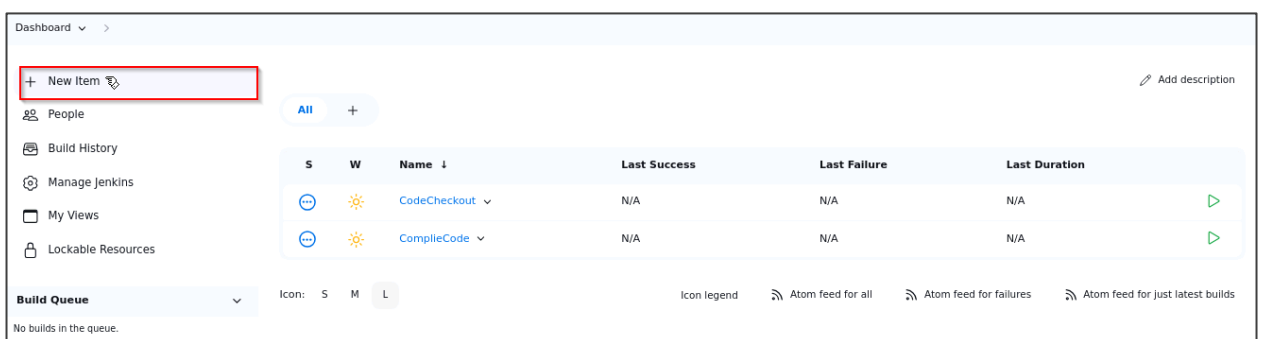
1.9 Enter **compile** as the goal in the **Goals** field and click on **Save**



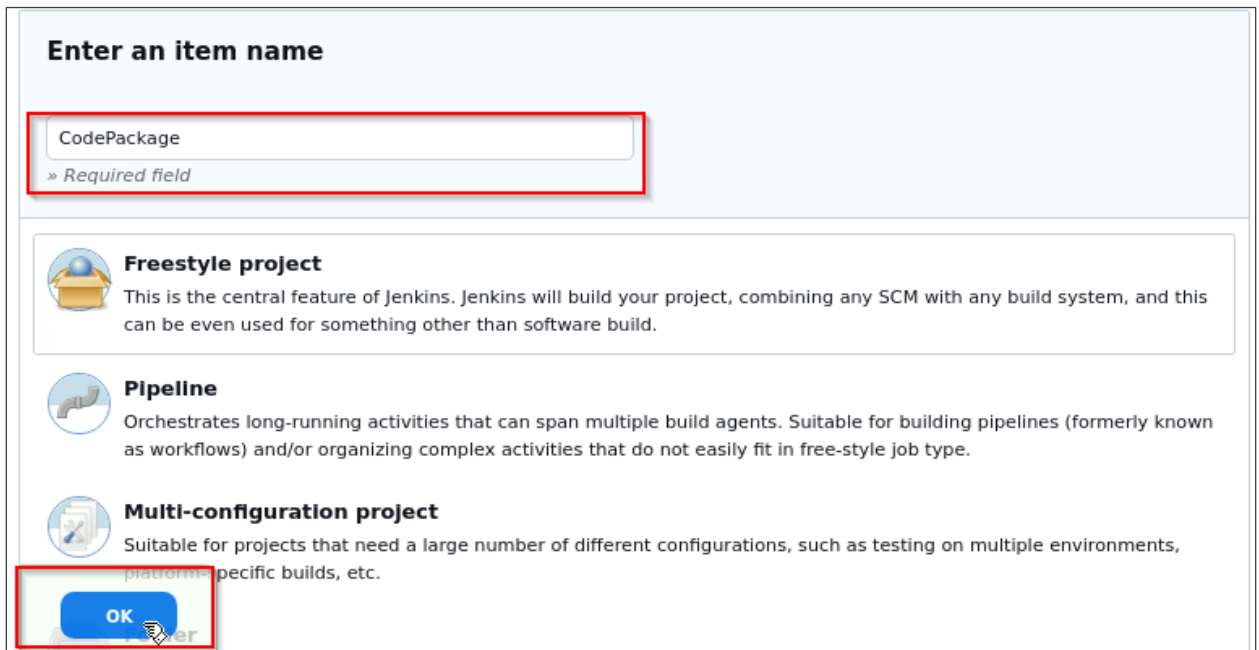
1.10 Refresh the page, and you will see the upstream job added



1.11 Go to the Jenkins dashboard and click on the **New item** box to create the third job



1.12 Enter the item name as **CodePackage**, select the type as **Freestyle project**, and click on **ok**



Enter an item name

CodePackage
» Required field

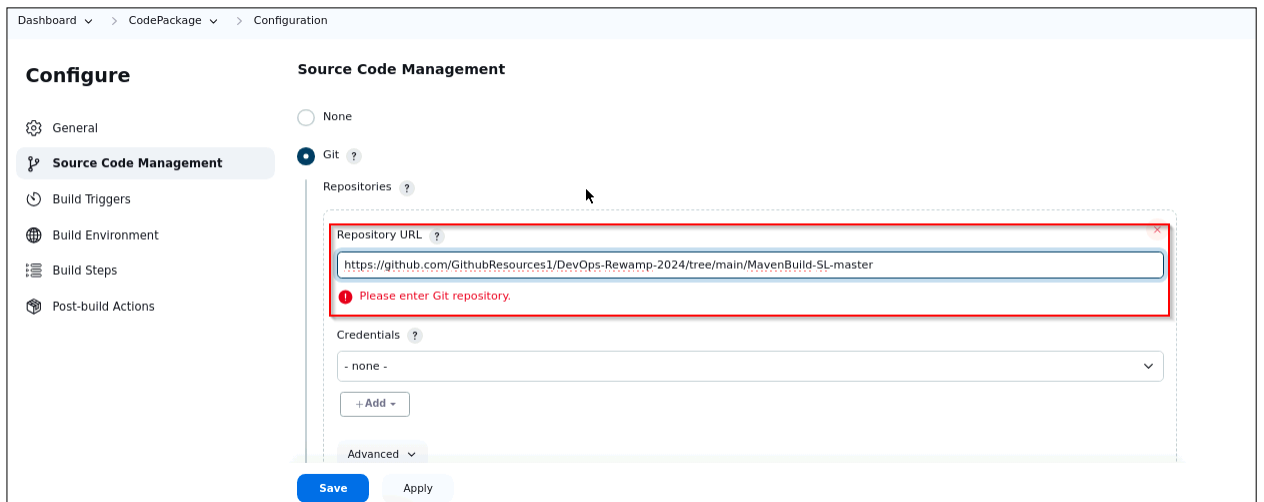
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

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.

OK

1.13 Click on **Source Code Management** on the configuration screen, select Git, and under Repository URL, give the URL as **<https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master>**



Dashboard > CodePackage > Configuration

Configure

- General
- Source Code Management**
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?
<https://github.com/GithubResources1/DevOps-Rewamp-2024/tree/main/MavenBuild-SL-master>

Please enter Git repository.

Credentials ?
- none -

+ Add

Advanced

Save Apply

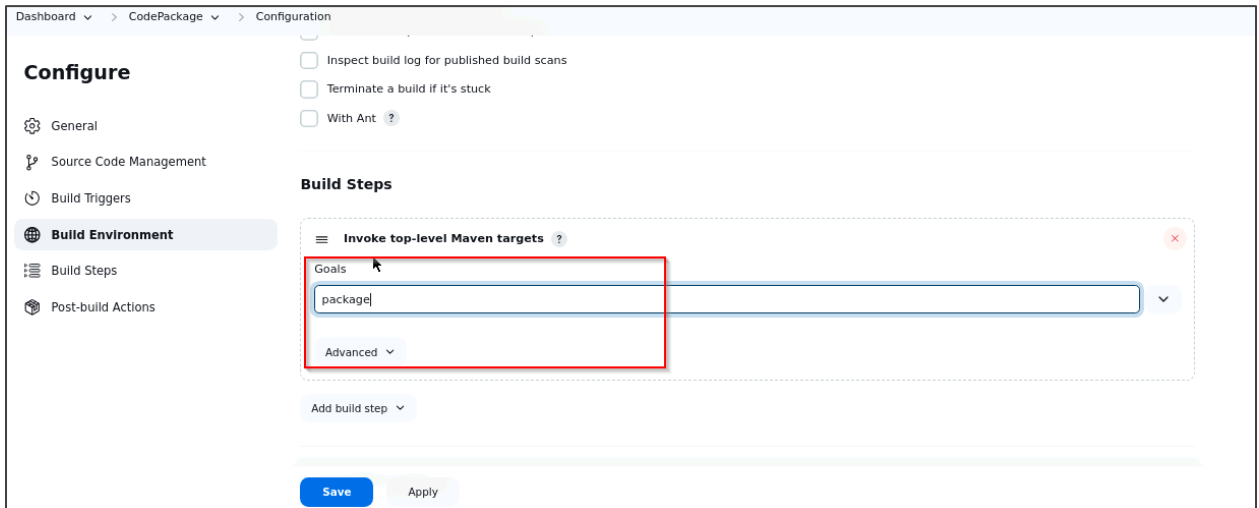
- 1.14 Click on **Build Triggers**, select the option **Build after other projects are built**, and enter the name as **CompileCode** in the **Projects to watch** field

The screenshot shows the 'Configure' page for 'CodePackage' in the 'Configuration' section. The left sidebar lists various configuration categories: General, Source Code Management, Build Triggers (selected), Build Environment, Build Steps, and Post-build Actions. The main content area is titled 'Build Triggers'. It contains several options: 'Trigger builds remotely (e.g., from scripts)' (unchecked), 'Build after other projects are built' (checked), and 'Projects to watch' (a text input field containing 'CompileCode'). Below this, there are radio button options for triggering builds: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', 'Trigger even if the build fails', and 'Always trigger, even if the build is aborted'. A red error message 'No project specified' is visible below the 'Projects to watch' field. At the bottom, there are 'Save' and 'Apply' buttons.

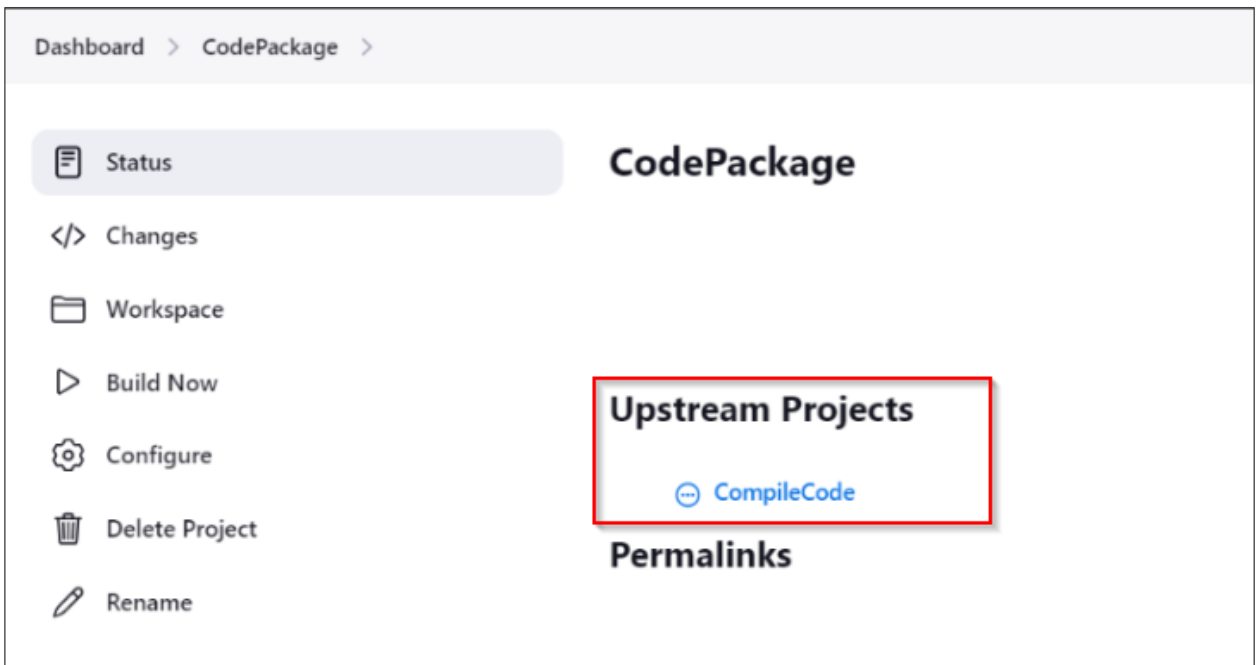
- 1.15 Click on **Build Steps** and select the option **Invoke top-level Maven targets**

The screenshot shows the 'Configure' page for 'CodePackage' in the 'Configuration' section. The left sidebar lists various configuration categories: General, Source Code Management, Build Triggers, Build Environment, Build Steps (selected), and Post-build Actions. The main content area is titled 'Build Steps'. It contains several options: 'Inspect build log for published build scans' (unchecked), 'Terminate a build if it's stuck' (unchecked), 'With Ant' (unchecked), and 'Add build step' (a button). Below this, there is a list of build steps: 'Execute Windows batch command', 'Execute shell', 'Invoke Ant', 'Invoke Gradle script', 'Invoke top-level Maven targets' (selected), 'Run with timeout', 'Set build status to "pending" on GitHub commit', and 'Trigger/call builds on other projects'. The 'Invoke top-level Maven targets' option is highlighted with a red box.

1.16 Enter the goal as **package** in the **Goals** field and click on **Save**

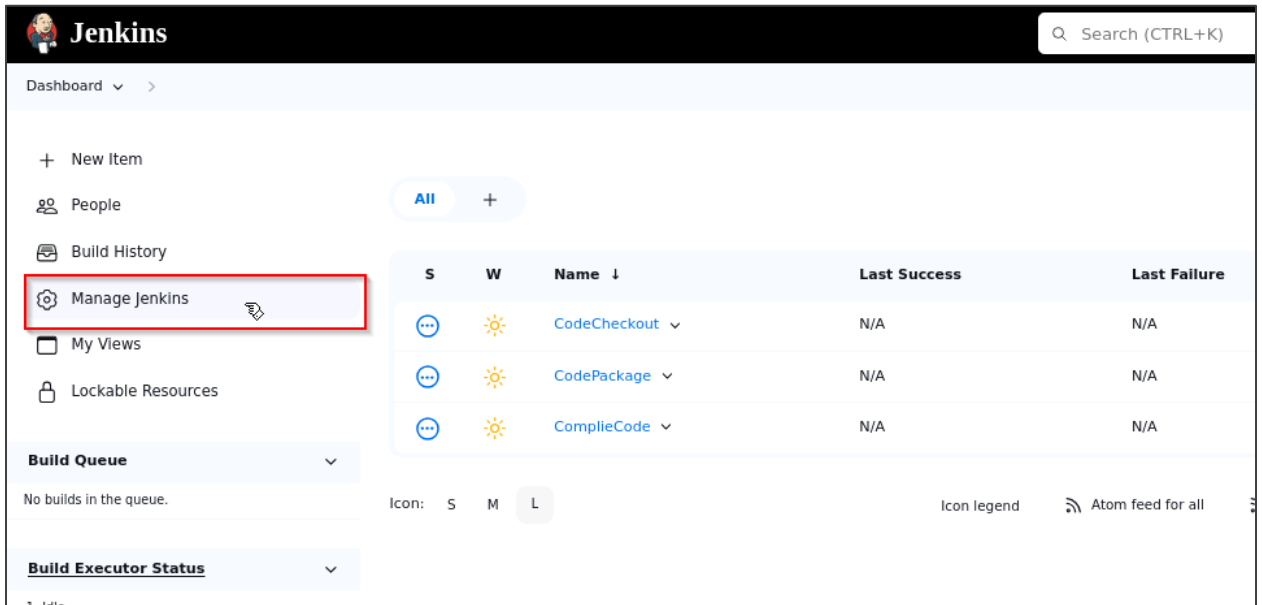


1.17 Refresh the page, and you will see the upstream job added



Step 2: Install the build pipeline plugin in Jenkins

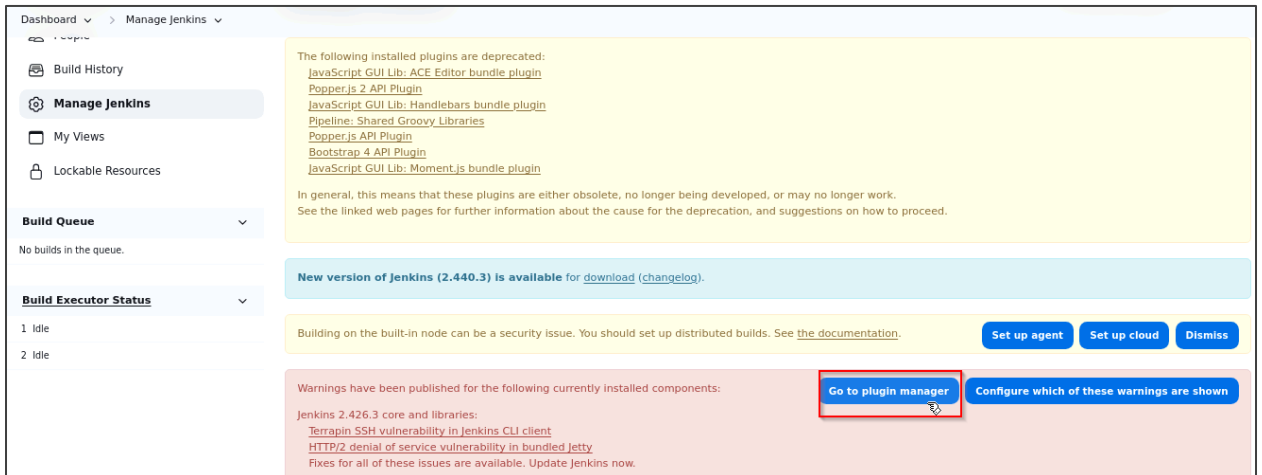
2.1 Go to the **Dashboard** and click on **Manage Jenkins**



The screenshot shows the Jenkins Dashboard. On the left sidebar, the 'Manage Jenkins' option is highlighted with a red rectangle. The main content area displays a table of installed plugins. The table has columns for 'S' (Status), 'W' (Warning), 'Name', 'Last Success', and 'Last Failure'. Three plugins are listed: CodeCheckout, CodePackage, and CompieCode, all with a status of 'N/A' for both success and failure.

S	W	Name ↓	Last Success	Last Failure
...	☀	CodeCheckout ▾	N/A	N/A
...	☀	CodePackage ▾	N/A	N/A
...	☀	CompieCode ▾	N/A	N/A

2.2 Click on **Go to plugin manager**



The screenshot shows the 'Manage Jenkins' page. The 'Manage Jenkins' option in the left sidebar is highlighted. The main content area contains several notifications. A yellow box at the top lists deprecated plugins. A blue box below it announces a new version of Jenkins (2.440.3). A yellow box below that mentions a security issue with the built-in node. At the bottom, a red box contains warnings about vulnerabilities in Jenkins 2.426.3, with the 'Go to plugin manager' button highlighted by a red rectangle.

The following installed plugins are deprecated:

- JavaScript GUI Lib: ACE Editor bundle plugin
- Popper.js 2 API Plugin
- JavaScript GUI Lib: Handlebars bundle plugin
- Pipeline: Shared Groovy Libraries
- Popper.js API Plugin
- Bootstrap 4 API Plugin
- JavaScript GUI Lib: Moment.js bundle plugin

In general, this means that these plugins are either obsolete, no longer being developed, or may no longer work. See the linked web pages for further information about the cause for the deprecation, and suggestions on how to proceed.

New version of Jenkins (2.440.3) is available for [download](#) ([changelog](#)).

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#). [Set up agent](#) [Set up cloud](#) [Dismiss](#)

Warnings have been published for the following currently installed components:

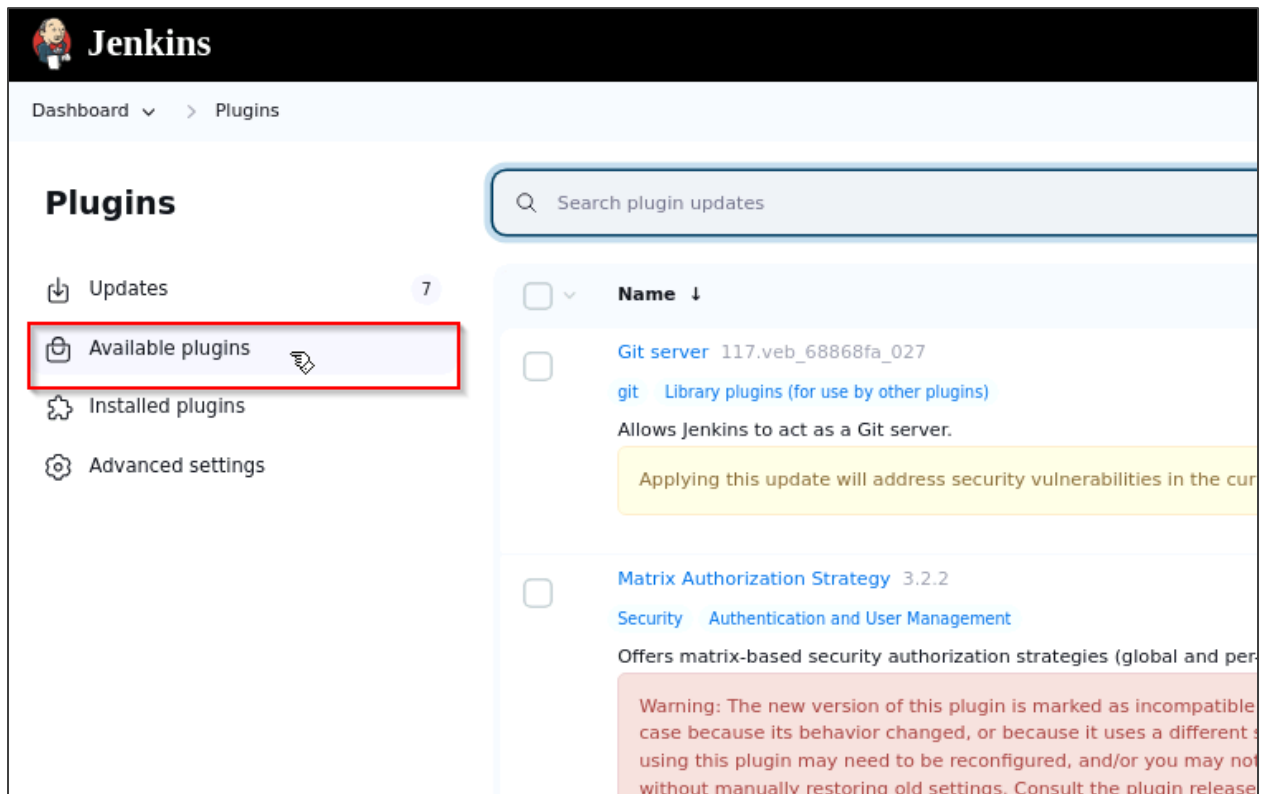
Jenkins 2.426.3 core and libraries:

- [Terrapin SSH vulnerability in Jenkins CLI client](#)
- [HTTP/2 denial of service vulnerability in bundled Jetty](#)

Fixes for all of these issues are available. Update Jenkins now.

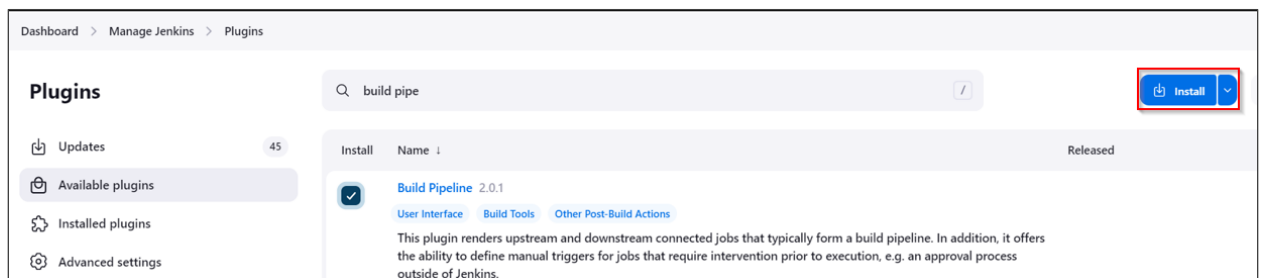
[Go to plugin manager](#) [Configure which of these warnings are shown](#)

2.3 Select the **Available plugins** option



The screenshot shows the Jenkins 'Plugins' page. On the left sidebar, the 'Available plugins' option is highlighted with a red box. The main content area displays a list of plugins. The first plugin, 'Git server', is highlighted. Below it, the 'Matrix Authorization Strategy' plugin is listed with a warning message: 'Warning: The new version of this plugin is marked as incompatible case because its behavior changed, or because it uses a different s using this plugin may need to be reconfigured, and/or you may not without manually restoring old settings. Consult the plugin release'.

2.4 Search for the **Build Pipeline** plugin, select it, and click the **Install** button



The screenshot shows the Jenkins 'Plugins' page with a search bar containing 'build pipe'. The search results show the 'Build Pipeline' plugin (version 2.0.1) under the 'Install' column. The 'Install' button is highlighted with a red box. The plugin description states: 'This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.'

The build pipeline plugin will be installed.

Dashboard > CodeCheckout >

Status

Changes

Workspace

Build Now

Configure

Delete Project

Rename

Build History trend

CodeCheckout

Downstream Projects

CompileCode

Permalinks

Last build (#2), 2 min 50 sec ago

Last stable build (#2), 2 min 50 sec ago

Last successful build (#2), 2 min 50 sec ago

Last completed build (#2), 2 min 50 sec ago

2.5 Go to the Jenkins dashboard and click on the **New View** tab

Dashboard >

New Item

People

Build History

Manage Jenkins

My Views

Lockable Resources

Build Queue

No builds in the queue.

New View

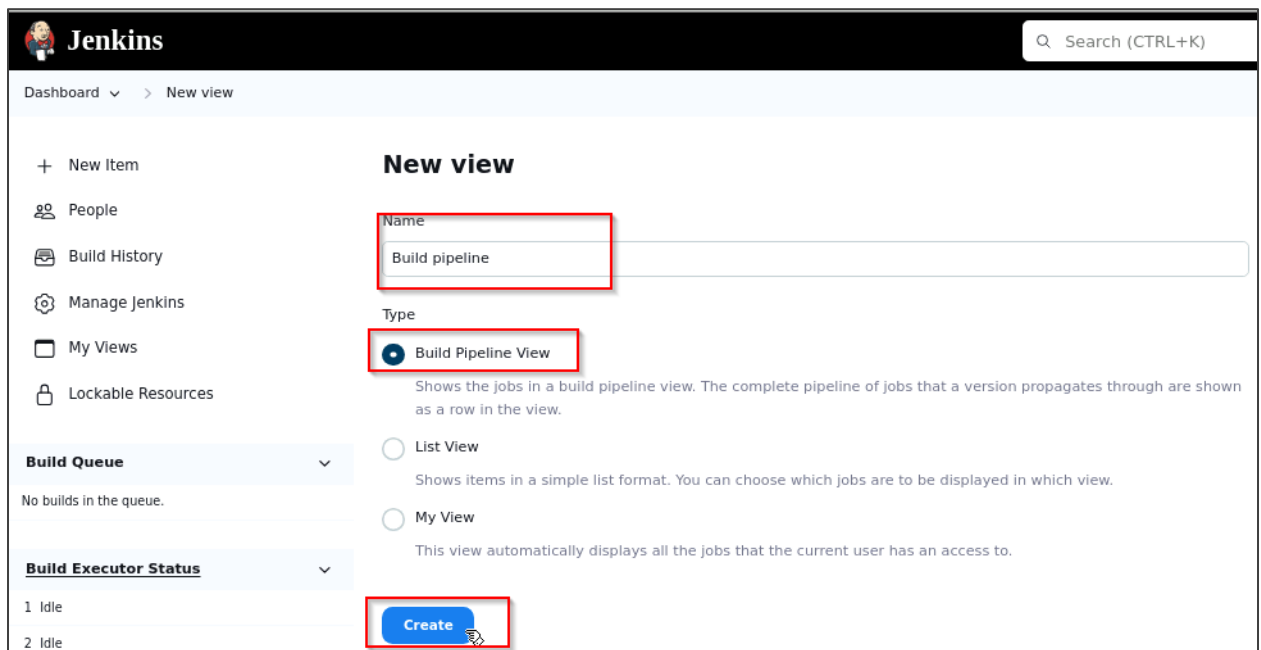
All

S	W	Name ↓	Last Success
...	☀	CodeCheckout	N/A
...	☀	CodePackage	N/A
...	☀	CompileCode	N/A

Icon: S M L

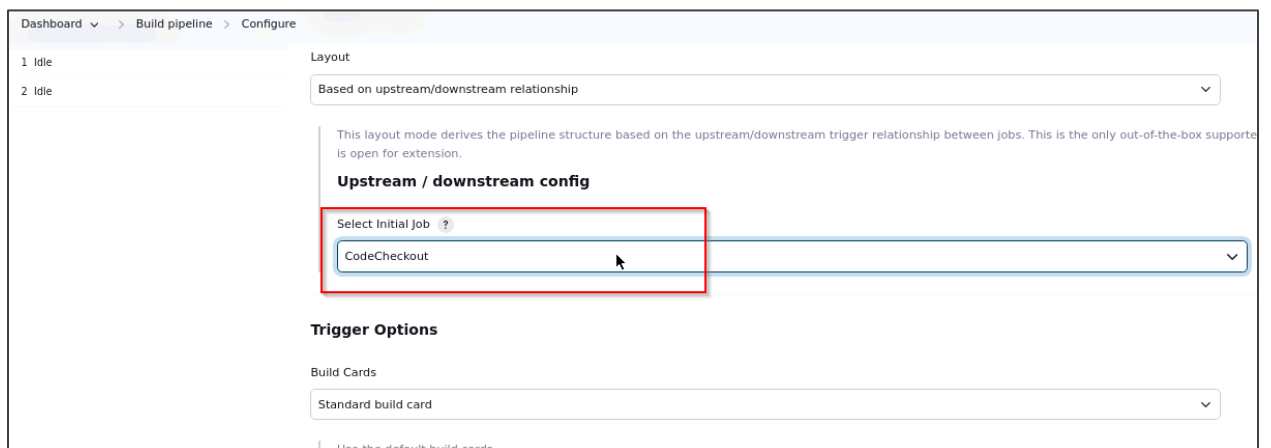
Icon legend

2.6 Enter the name as **Build pipeline** in the **New View** tab, select the type as **Build Pipeline View**, and click on **Create**



The screenshot shows the Jenkins 'New view' configuration page. On the left sidebar, there are links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Lockable Resources'. Below these are two sections: 'Build Queue' showing 'No builds in the queue.' and 'Build Executor Status' showing two 'Idle' executors. The main area is titled 'New view'. It has a 'Name' field containing 'Build pipeline' and a 'Type' section with three radio buttons: 'Build Pipeline View' (selected), 'List View', and 'My View'. Descriptions are provided for each type. At the bottom right, there is a blue 'Create' button.

2.7 Scroll down on the configure page until you see the **Select Initial Job** drop-down. In this, select Initial Job as **CodeCheckout**



The screenshot shows the Jenkins 'Build pipeline' configuration page. The left sidebar shows the 'Build pipeline' structure with two 'Idle' executors. The main area is titled 'Configure'. It has a 'Layout' section with a dropdown menu set to 'Based on upstream/downstream relationship'. Below this is a section titled 'Upstream / downstream config' which contains a 'Select Initial Job' dropdown menu with 'CodeCheckout' selected. Further down is a 'Trigger Options' section with a 'Build Cards' dropdown menu set to 'Standard build card'.

2.8 Click on the **OK** button

Upstream / downstream config

Select Initial Job ?

CodeCheckout

Trigger Options

Build Cards

Standard build card

OK

Apply

Step 3: Trigger the execution of jobs

3.1 Click on the dashboard and trigger the execution of the first job, which is **CodeCheckout**

Dashboard > CodeCheckout >

Status

</> Changes

Workspace

Build Now

Configure

Delete Project

Rename

Build History trend

CodeCheckout

Downstream Projects

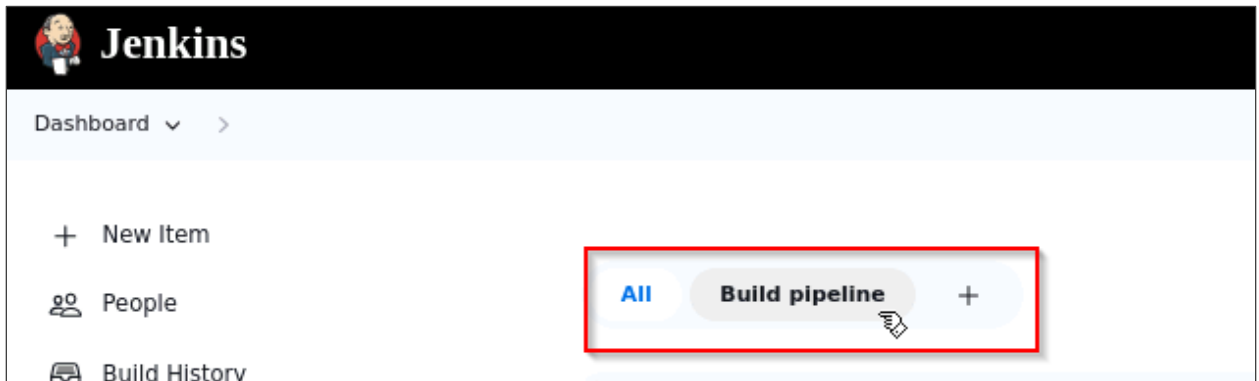
CompileCode

Permalinks

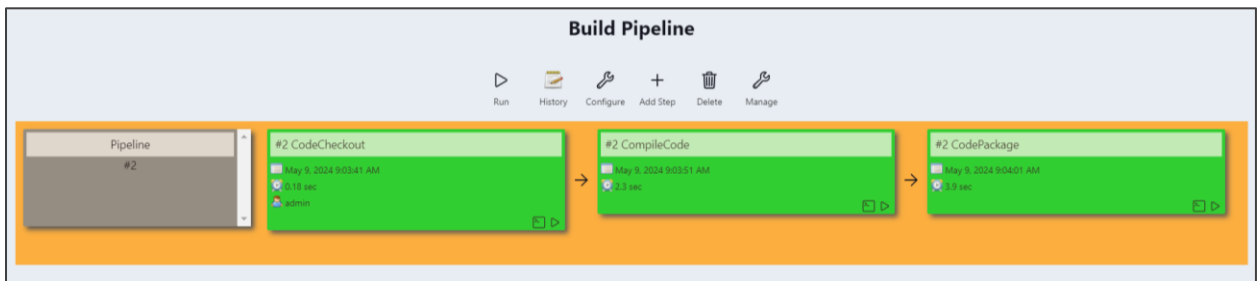
- Last build (#2), 2 min 50 sec ago
- Last stable build (#2), 2 min 50 sec ago
- Last successful build (#2), 2 min 50 sec ago
- Last completed build (#2), 2 min 50 sec ago

The job will run successfully, and it will automatically trigger the execution of the second job which in turn will trigger the execution of the third job sequentially.

3.2 Click on the Jenkins dashboard, select the **build pipeline**, and refresh the Jenkins page



You will be able to see the plugin-based build pipeline



By following these steps, you have successfully built and executed a Jenkins job chaining workflow with the build pipeline plugin and GitHub integration for efficient continuous integration and deployment processes.