

Lab 2: Creating a Manual Pipeline for Java App

Author : Gourav Shah

Publisher: School of Devops

Version : v2024.05.01.02

Following are the learning objectives with this lab

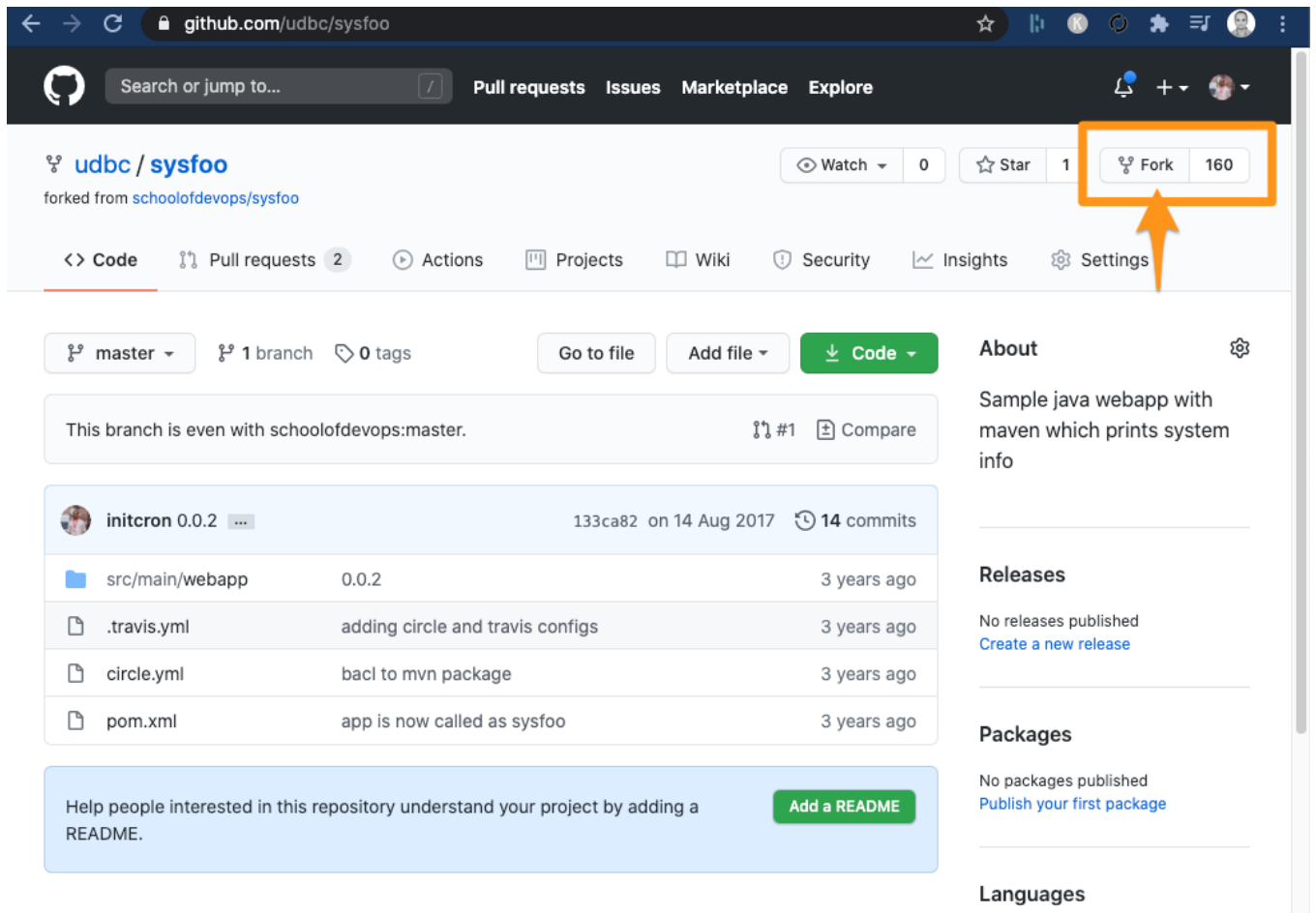
- start creating freestyle and maven jobs
- integrate with tools such as nodeJS and maven
- integrate with GitHub and setup build triggers and.
- setup a pipelines which run automated builds and unit tests

Creating a pipeline for a Maven project

Following is the anatomy of a jenkins job.



You could create jenkins job to build one of the applications with the **sysfoo** sample app. You need to create your version of this app by forking it. Visit [GitHub - udbc/sysfoo: Sample java webapp with maven which prints system info](#) and fork the repository onto your git account.



Now is the time to build the sysfoo project, which is a java application that uses maven as a build tool.

You could follow the following steps to configure maven build job.

Steps:

- Goto `manage jenkins -> global tools configuration` and under **Maven** section, provide name as `Maven 3.9.6` (as a guidance, pick the latest version) and select the maven version `3.9.6`, save those changes.

Maven installations ^

 Edited

Add Maven

≡ **Maven**

Name

Maven 3.9.6

☒ Install automatically ?

≡ **Install from Apache**

Version

3.9.6

Add Installer ▾

Add Maven

Docker installations

Add Docker

Save


Apply

- Install `Maven integration` plugin. Goto `Manage Jenkins => Manage Plugins =>`


Available, search for **Maven Integration** plugin and install it without restart.

Plugins


 Updates




 Available plugins

 Installed plugins


 Advanced settings


 Download progress


 **Jenkins**

  Gourav Shah  log out

Jenkins > Plugin Manager >

 Back to Dashboard

 Manage Jenkins

 Update Center

Updates

Available

Installed

Advanced

Install ↑	Name	Version	Released
	Maven Integration		
	<div>Build Tools</div>		
<input checked="" type="checkbox"/>	<div>This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTS, automated configuration of various Jenkins publishers (Junit, ...).</div>	3.8	9 days 1 hr ago
	Config File Provider		
	<div>External Site/Tool Integrations</div>		
	<div>Groovy-related</div>		
	<div>Maven</div>		
<input type="checkbox"/>	<div>Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files,...) loaded through the UI which will be copied to the job workspace.</div>	3.7.0	1 mo 1 day ago

Static Analysis Utilities

Install without restart

Download now and install after restart

Update information obtained: 16

Once installed, click on **Go back to the top page** to browse to Jenkins main page.

Mailer	● Already Installed
Loading plugin extensions	● Success
Javadoc	● Success
Maven Integration	● Success
Loading plugin extensions	● Success

➡ [Go back to the top page](#) (you can start using the installed plugins right away)

➡ ☐ Restart Jenkins when installation is complete and no jobs are running

- Create `sysfoo` folder for your project which serves as a namespace. Do so by creating a new job with type **folder** with name `sysfoo`

Jenkins

New Item

People

Build History

Manage Jenkins

My Views

add description

Welcome to Jenkins!

Create an agent or configure a cloud to set up distributed builds. [Learn more.](#)

Create a job to start building your software project.

New Item

Enter an item name

sysfoo

Select an item type



Freestyle project

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



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



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 separate namespace, so you can have multiple things of the same name as long as they are in 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:

OK

On the second page, without changing any configurations, just click on save and proceed.

General

Health metrics

Properties

Pipeline Libraries

Description

[Plain text] [Preview](#)

Health metrics

[Health metrics...](#)

Properties

Jira sites

[Add](#)

Docker Label

Docker registry URL

Registry credentials

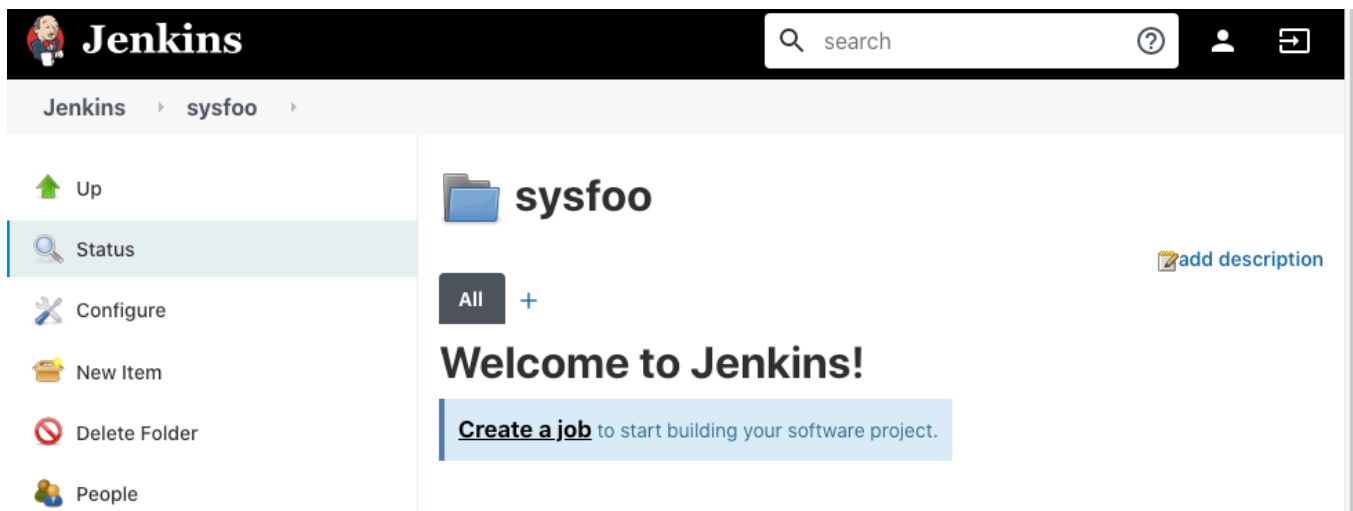
[- none -](#)[Add](#)

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries

[Add](#)[Save](#)[Apply](#)

- That creates a folder by name sysfoo, switch to it from top jenkins page. folder,



Create a new job and select **Maven** as the type of project. Name the job as **build**

New Item

Enter an item name

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to 10 sources, runs arbitrary shell or batch scripts, and supports post-build steps like archiving artifacts and sending email notifications.



Maven project

Build a maven project. Jenkins takes advantage of your POM file to automatically configure the build process.



Pipeline

Orchestrates long-running activities that can span multiple builds (known as workflows) and/or organizing complex activities that can be reused across multiple builds.

- Next, go to source code management, enter the URL of your git repository that you copy from GitHub (use the fork that you created earlier)

master
2 branches
0 tags

This branch is 5 commits ahead, 2 commits behind udbc:master.

initcron Create Dockerfile

src/main/webapp	0.0.2
.travis.yml	adding circle and travis configs
Dockerfile	Create Dockerfile
Jenkinsfile	use docker agent along with an image to provide custom build environ... 7 days ago
circle.yml	bacl to mvn package 3 years ago

Go to file

Add file

Code

Clone

[HTTPS](#)
[SSH](#)
[GitHub CLI](#)

<https://github.com/cicdapp/sysfoo.0>

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

After adding the Repository URL, also change the branch specifier to `*/main`

Git ?

Repositories ?

Repository URL ?

`https://github.com/devops-0005/sysfoo.git`

Credentials ?

- none -

+ Add ▾

Advanced ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

`*/main`

Add Branch

- Go to build step and provide the goal and option as `compile`.

Build

Root POM

?

Goals and options

?

Advanced...

- At the build step, you may want to provide the path to pom.xml. In this example its right inside the root directory of sysfoo, so nothing to change. Save the job and click on **Build Now**.
Observe the job status, console output etc.

Jenkins > sysfoo > build >

Up

Status

Changes

Workspace

Build Now

Configure

Delete Maven project

Modules

Maven project build

Workspace

Recent Changes

Permalinks

Adding unit test and packaging jobs

After creating the build job, its now time to add test and package jobs for the sysfoo application. Start creating jobs inside **sysfoo** folder.

You need to create one more job on your same folder and name it as `test`,

New Item

Enter an item name

test

Select an item type



Freestyle project

Classic, general-purpose job type that post-build steps like archiving artifacts



Maven project

Build a maven project. Jenkins takes a



Pipeline

Orchestrates long-running activities th known as workflows) and/or organizing

From the job creation page while creating `test` job, scroll all the way down to use **Copy from build** job. Follow following steps to complete the configuration.

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



Copy from

OK

- In `test` job, change your description as `test sysfoo java app`. Source code management, repository is same.
- under build step, change the goals and option as `clean test` and remaining will be same, so save the job and build.

Build

Root POM	<input type="text" value="pom.xml"/>	?
Goals and options	<input type="text" value="clean test"/>	?

[Advanced...](#)

[Build](#)

- Next job is `package`, this will compile the application and then generate the war file. Create a job on same folder with the name of `package`, and copy configurations from the `test` job.
- Update the description as `package sysfoo java app, create jar` and only change in the configuration is build step.
- In the build step for the package job, change the goal to `package -DskipTests`, save the changes and build.

Build

Root POM

?

Goals and options

?

Advanced...

Archiving the Artifacts

- After build successful, you could see the jar file created in the workspace. Example is given following , you can verify your workspace by using it.

Status

Changes

Workspace

Build Now

Configure

Delete Maven project

Modules

Move

Favorite

Workspace of package on Built-In Node

package / **target** /

classes

generated-sources/annotations

generated-test-sources/test-annotations

maven-archiver

maven-status/maven-compiler-plugin

test-classes/com/example/sysfoo

sysfoo-1.0.0-SNAPSHOT.jar

sysfoo-1.0.0-SNAPSHOT.jar.original

May 1, 2024, 3:09:50 PM

44.14 MiB

May 1, 2024, 3:09:50 PM

12.14 KiB

[\(all files in zip\)](#)

- Now configure the package job and from the **post build actions**, choose archive artifacts and provide path ****target/*.jar** in the *Files to archive* input field so that the jarfile created at this path (e.g. sysfoo.war) is automatically archived/published.

 Filter

Aggregate downstream test results

Archive the artifacts

Build other projects

Deploy artifacts to Maven repository

Publish HTML reports

Record fingerprints of files to track usage

Git Publisher

Build other projects (manual step)

Editable Email Notification

Set GitHub commit status (universal)

Set build status on GitHub commit [deprecated]

Trigger parameterized build on other projects

Delete workspace when build is done

Add post-build action ^

builds, €

Post-build Actions

≡ Archive the artifacts ?

×

Files to archive ?

**/target/*.jar

Advanced ▾

Add post-build action ▾

- Save the changes and build the job. Once build successful, check the project page to find out your artifacts right out there.

Configuring Artifact Version

Instead of you needing to manually generating the artifact version e.g. sysfoo-1.0.0-SNAPSHOT.jar , you could have it generated using a git commit id which is unique for every change. To achieve that, you could add a script such as follows:

```
# Truncate the GIT_COMMIT to the first 7 characters
GIT_SHORT_COMMIT=$(echo $GIT_COMMIT | cut -c 1-7)

# Set the version using Maven
mvn versions:set -DnewVersion="$GIT_SHORT_COMMIT"
mvn versions:commit
```

Add it using Pre Steps => Execute shell

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Pre Steps**
- Build
- Post Steps
- Build Settings

Pre Steps

Add pre-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

Advanced v

and copy over the script above in the command box

Pre Steps

 **Execute shell** 



Command



See [the list of available environment variables](#)


```
# Truncate the GIT_COMMIT to the first 7 characters
GIT_SHORT_COMMIT=$(echo $GIT_COMMIT | cut -c 1-7)

# Set the version using Maven
mvn versions:set -DnewVersion="$GIT_SHORT_COMMIT"
mvn versions:commit
```

Advanced ▾

Pre Steps

 **Execute shell** 



Command

See [the list of available environment variables](#)

```
# Truncate the GIT_COMMIT to the first 7 characters
GIT_SHORT_COMMIT=$(echo $GIT_COMMIT | cut -c 1-7)

# Set the version using Maven
mvn versions:set -DnewVersion="1.0.0-$GIT_SHORT_COMMIT"
mvn versions:commit
```

Advanced ▾

Add pre-build step ▾

Save and build to see the artifact generated with commit hash now. Validate that the artifact is published on the Package Job status page.

Status

</> Changes

Workspace

Build Now

Configure


Delete Maven project

Modules



Move

Favorite

Maven project package



Last Successful Artifacts

 sysfoo-77a0877.jar 44.14 MiB  view

Permalinks

- [Last build \(#14\), 28 sec ago](#)
- [Last stable build \(#14\), 28 sec ago](#)
- [Last successful build \(#14\), 28 sec ago](#)
- [Last failed build \(#6\), 28 min ago](#)
- [Last unsuccessful build \(#6\), 28 min ago](#)
- [Last completed build \(#14\), 28 sec ago](#)

Connecting Jobs with Upstreams and Downstreams







Here you will learn, how to links the jobs by defining upstreams and downstream configurations. You would also go on to create a pipeline view using a plugin.

Follow the following steps to setup upstream and downstream


- From `build` job configuration page, scroll all the way to **post build actions**, this is where you could define the downstream job. Provide `test` as the job to build and save.


All


+


S	W	Name ↓	Last S
		build	6 min
			N/A
			N/A


Icon: S M L


 Changes


 Workspace


 Build Now


 Configure

 Delete Maven project

 Modules

 Move

 Open Blue Ocean

 Rename

Post-build Actions



Build other projects



Projects to build

test,

- ☒ Trigger only if build is stable
- ☐ Trigger even if the build is unstable
- ☐ Trigger even if the build fails

Add post-build action ▾

- Now you are going to setup upstream for `package . goto package` configuration page. From **build triggers**, check the box for `build after other projects are built` Provide upstream project name as `test job`

Build Triggers

☒ Build whenever a SNAPSHOT dependency is built



☐ Schedule build when some upstream has no successful builds



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



☒ Build after other projects are built



Projects to watch

test,

- ☒ Trigger only if build is stable
- ☐ Trigger even if the build is unstable
- ☐ Trigger even if the build fails

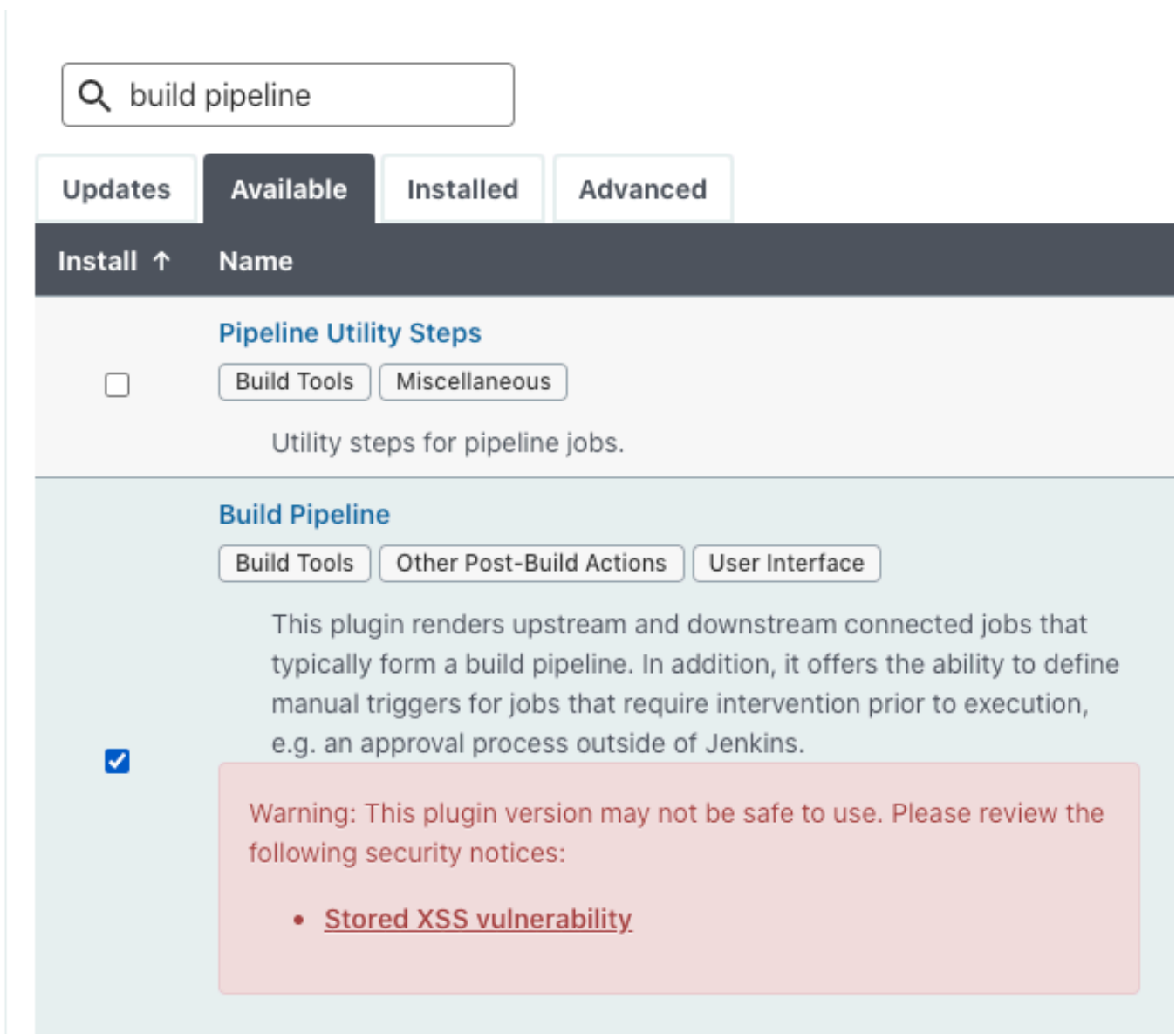
this defines the upstream for package.

Once upstreams and downstream are defined, run `build` and it will automatically run `test` & `package`.

Setup Pipeline View

Now you are going to setup pipeline view for this build jobs,

- Begin by installing `build pipeline` plugin from manage jenkins → Manage Plugins page.



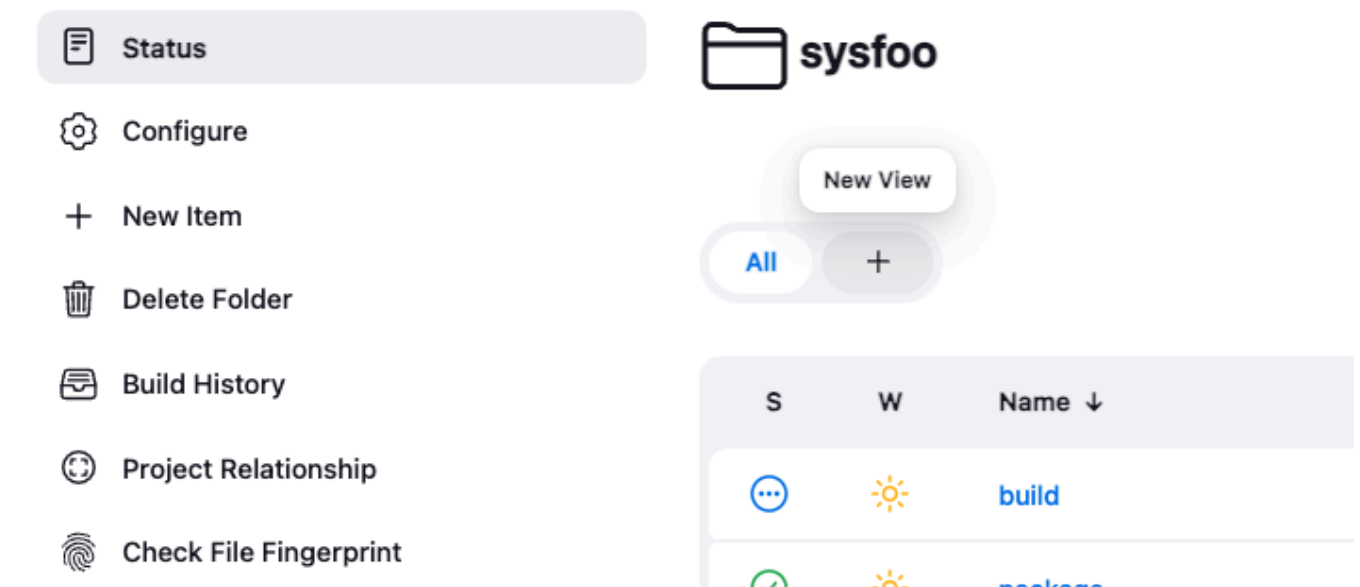
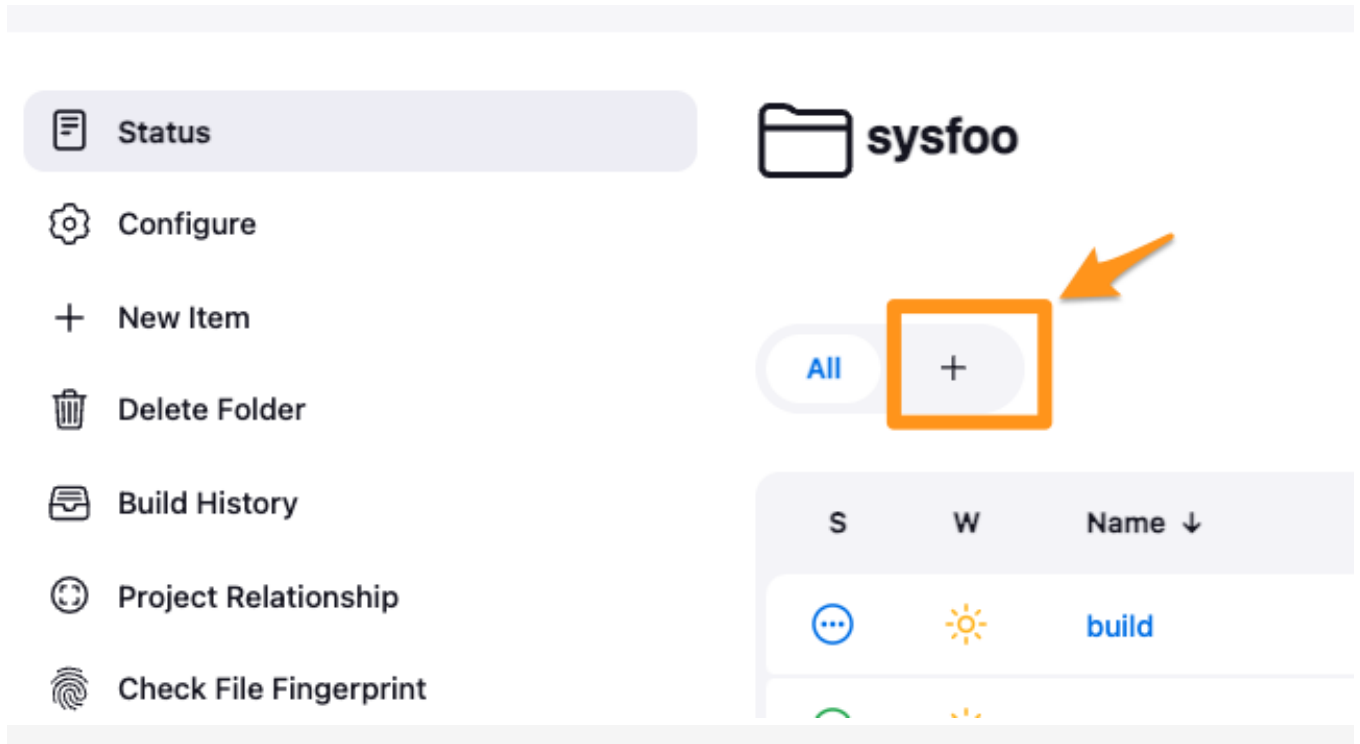
The screenshot shows the Jenkins Manage Plugins interface. At the top, there is a search bar containing 'build pipeline'. Below the search bar are four tabs: 'Updates', 'Available', 'Installed', and 'Advanced'. The 'Installed' tab is selected. Below the tabs is a table with columns 'Install' and 'Name'. The table lists two plugins: 'Pipeline Utility Steps' and 'Build Pipeline'. The 'Build Pipeline' plugin is checked in the 'Install' column. Below the 'Build Pipeline' plugin name, there are three sub-tabs: 'Build Tools', 'Other Post-Build Actions', and 'User Interface'. The 'Build Tools' sub-tab is selected. The description for the 'Build Pipeline' plugin 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.' Below the description, there is a red warning box that reads: 'Warning: This plugin version may not be safe to use. Please review the following security notices:'. Below the warning box, there is a list of security notices: '• [Stored XSS vulnerability](#)'.

You could install this plugin without a restart.

■ You may see a warning about this plugin is being vulnerable and may not be safe to use. Since

you are not going to use this in a production setup and only to understand the pipeline concept visually, its ok to proceed. Do not use this plugin in a live/production environment.

- From Jenkins console, browse to `sysfoo` folder and create a new view by clicking on the `+` tab to create a new view. Select type as pipeline view and provide name for it.



Provide a name and select **Build Pipeline View**

View name

☒ **Build Pipeline View**

Shows the jobs in a pipeline
shown as a row in the table

☐ **Include a global view**

Shows the content of the global view

☐ **List View**

Shows items in a simple list

☐ **My View**

This view automatically updates

- From the configuration page, select the first job in the pipeline and select number of builds as 5, save it.

Pipeline Flow

Layout

Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config

Select Initial Job

sysfoo » build

Trigger Options

Build Cards

Standard build card

Use the default build cards

Restrict triggers to most recent successful builds ☐ Yes ☒ No

Always allow manual trigger on pipeline steps ☐ Yes ☒ No

Display Options

No Of Displayed Builds

Row Headers

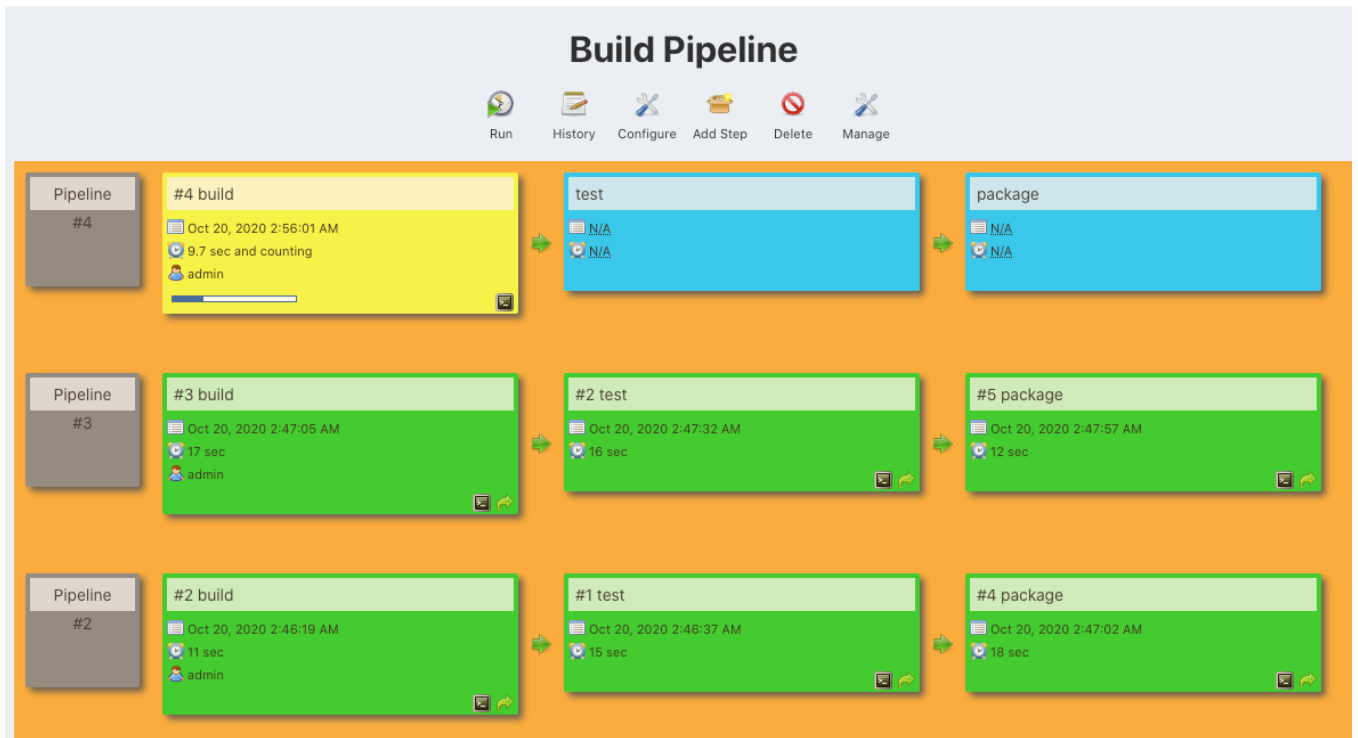
Column Headers

✓ 1
2
3
5
10
20
50
100
200
500

OK

Apply

- Once you complete, you could see build pipeline of your job. Green is successful, red is failed and blue is to do, yellow is in progress.



You have just completed creating a pipeline for a java project.












Set up automated Pipeline Triggers with Polling

You could use anything under build triggers for automatic build, but now you are going to use `poll scm` under build triggers.

- goto your `build` job configuration page and choose `poll SCM` under build triggers. In that poll scm mention schedule for periodically polling the git repository.

H/2 * * * *

- Once you made changes, save the job. goto your job page there you will find `Git polling log`, check your polling logs by using `Git polling log`.

-  Up
-  Status
-  Changes
-  Workspace
-  Build Now
-  Delete Maven project
-  Configure
-  Modules
-  **Git Polling Log**
-  Move
-  Rename

Git Polling Log

Started on Jul 22, 2019 1:54:00 PM
Using strategy: Default
[poll] Last Built Revision: Revision 2a35fe88806956c5ecc407640edced36174a0fd5 (refs/remotes/origin/master)
No credentials specified
> git --version # timeout=10
> git ls-remote -h <https://github.com/mohaninit/example-voting-app.git> # timeout=10
Found 1 remote heads on <https://github.com/mohaninit/example-voting-app.git>
[poll] Latest remote head revision on refs/heads/master is: 2a35fe88806956c5ecc407640edced36174a0fd5 - already built by 2
Done. Took 0.97 sec
No changes

#cicd/labsv3