# Lab 2: Creating a Manual Pipeline for Java App

Author : Gourav Shah

Publisher: School of Devops

Version : v2024.05.01.02

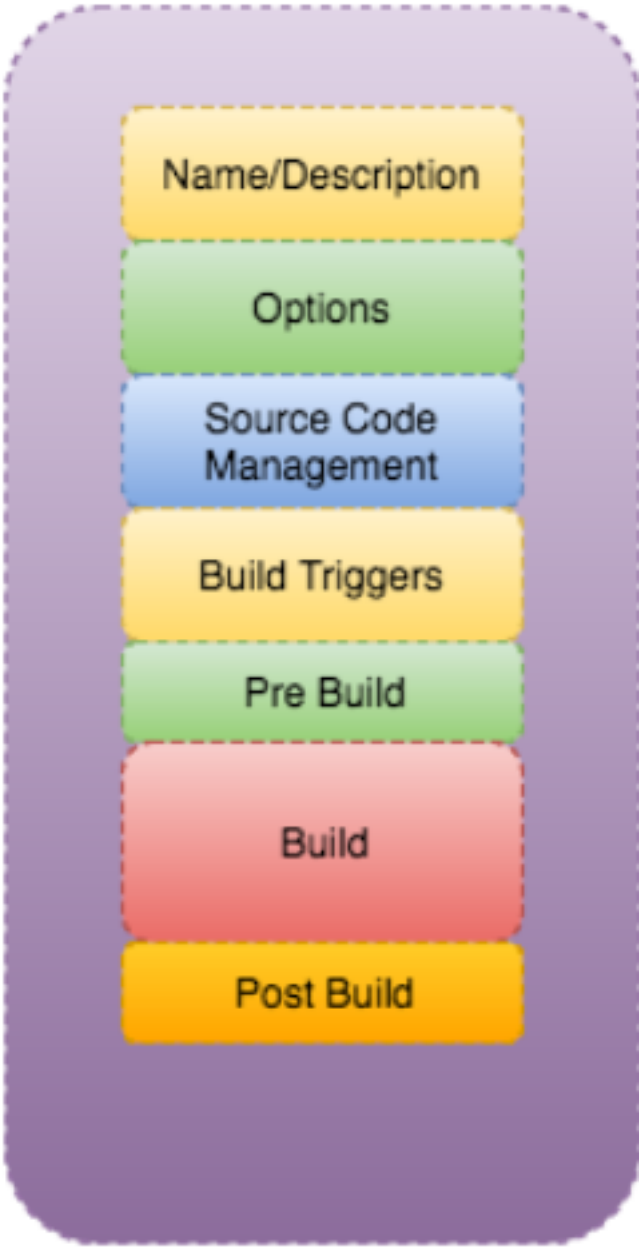
---

Following are the learning objectives with this lab

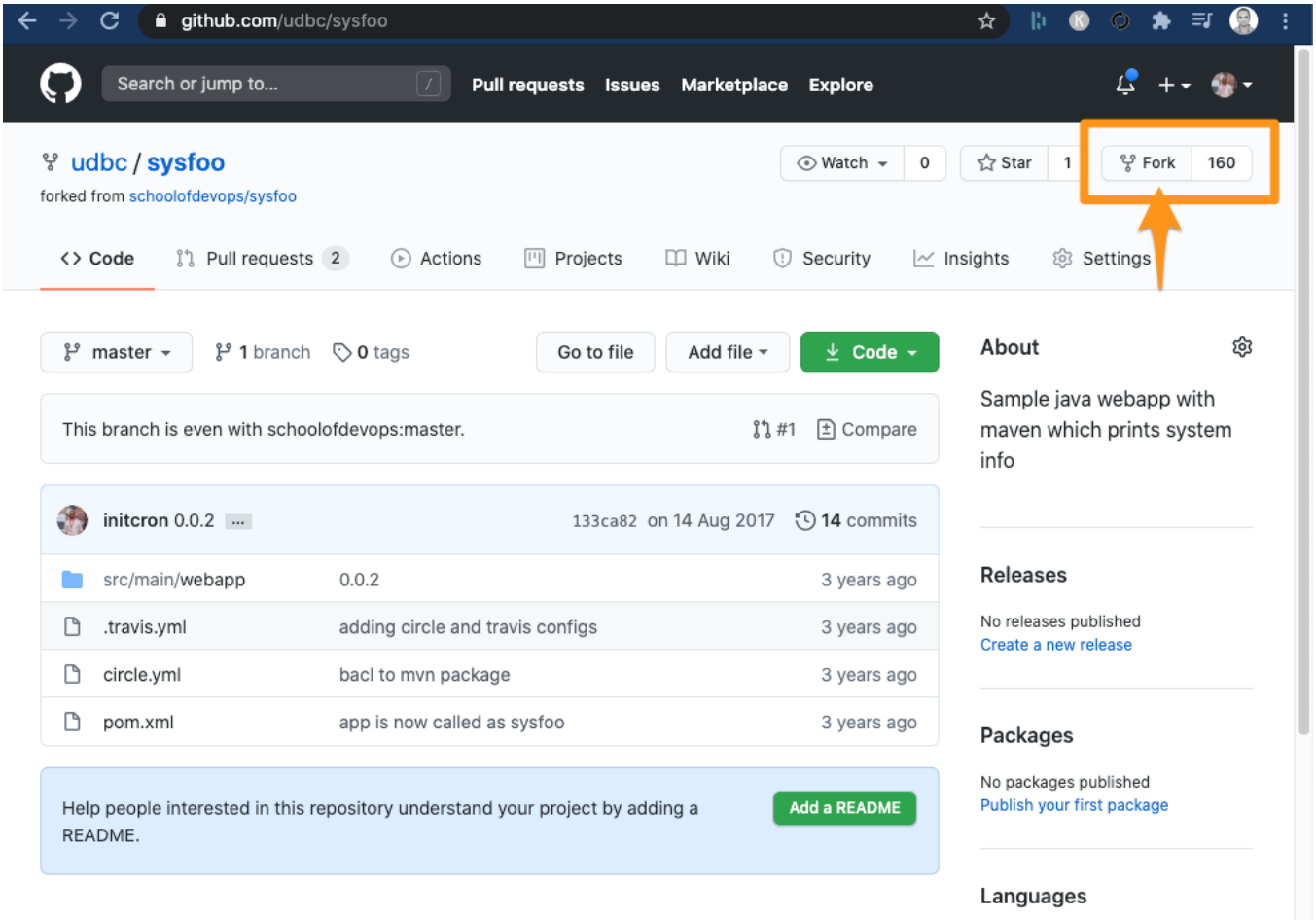
- start creating freestyle and maven jobs
- integrate with tools such as nodeJS  and maven
- integrate with GitHub and setup build triggers  and.
- setup a pipelines which run  automated builds  and unit tests

## 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.

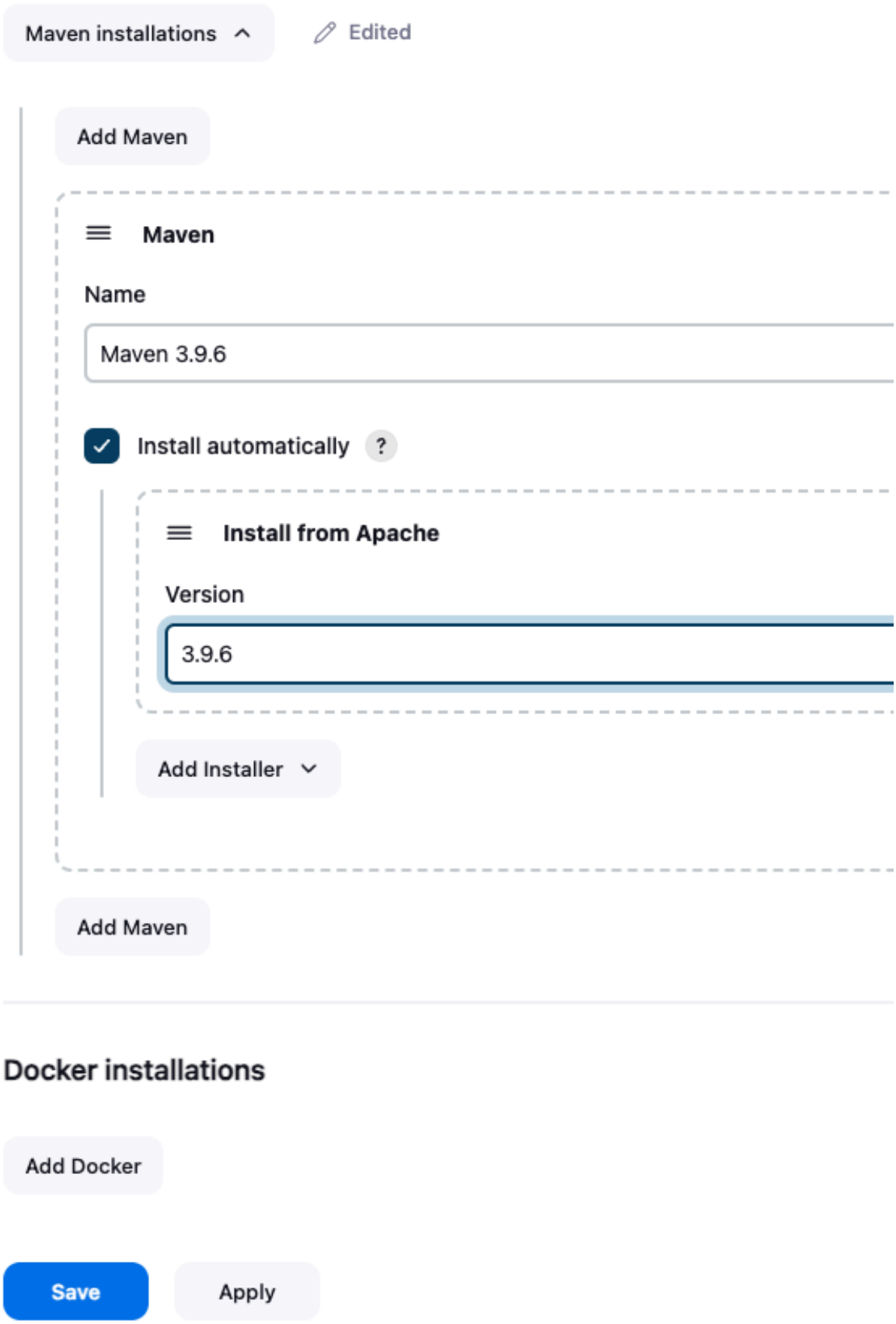



Now is the time to build the sysfoo project, which is a java application that uses maven as a build tool.

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

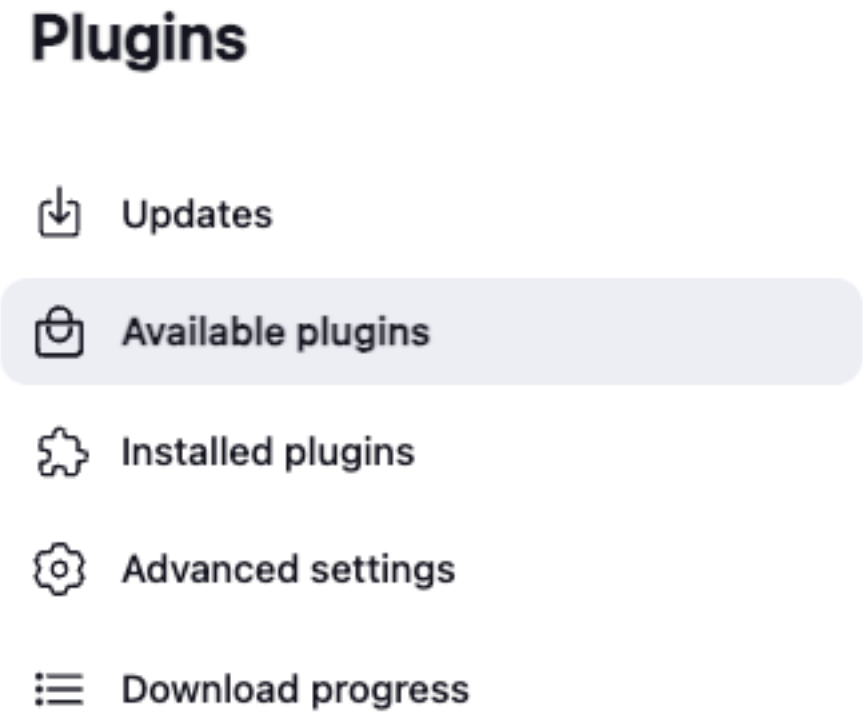
**Steps:**

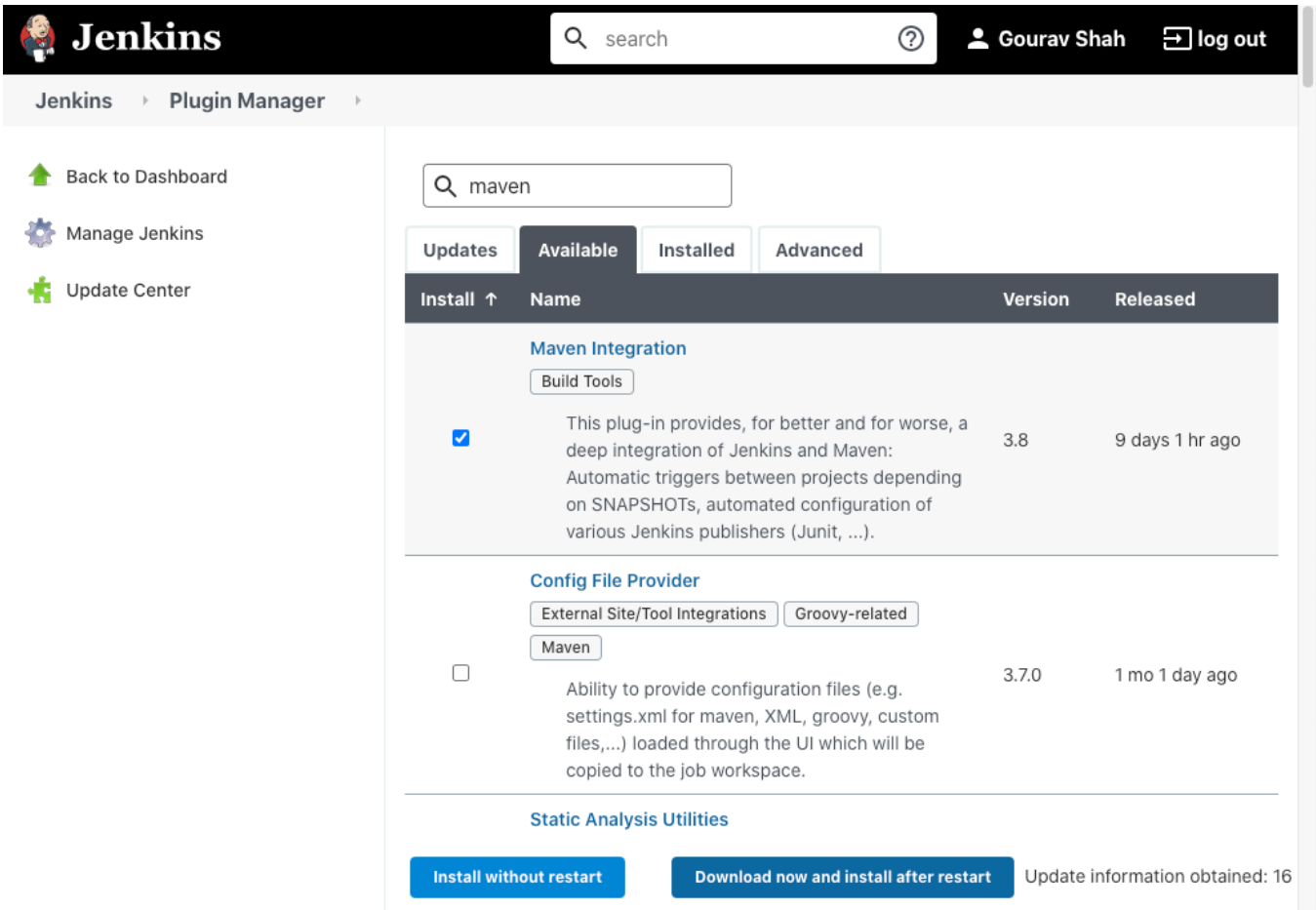
- Goto `manage jenkins -> global tools configuration` and under **Maven** section, provide name as `Maven 3.9.6` (as a guidance, pick the latest version ) and select the maven version `3.9.6`, save those changes.



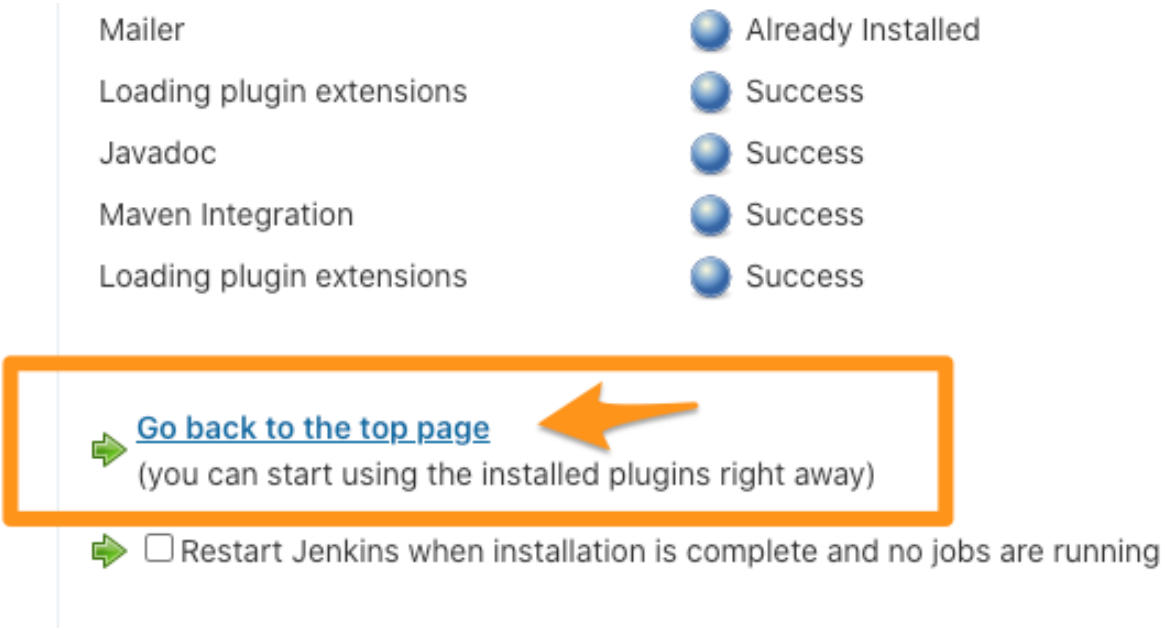

- Install `Maven integration` plugin. Goto `Manage Jenkins => Manage Plugins =>`

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

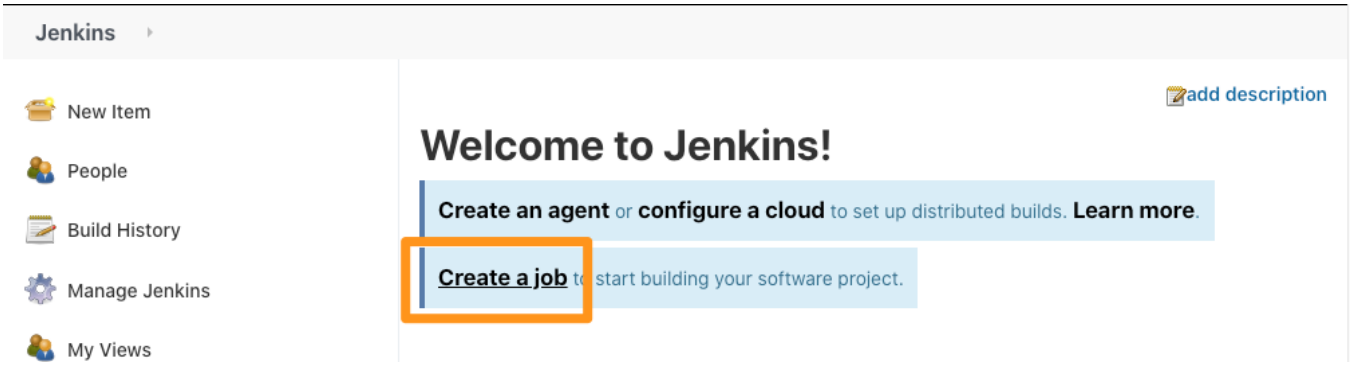


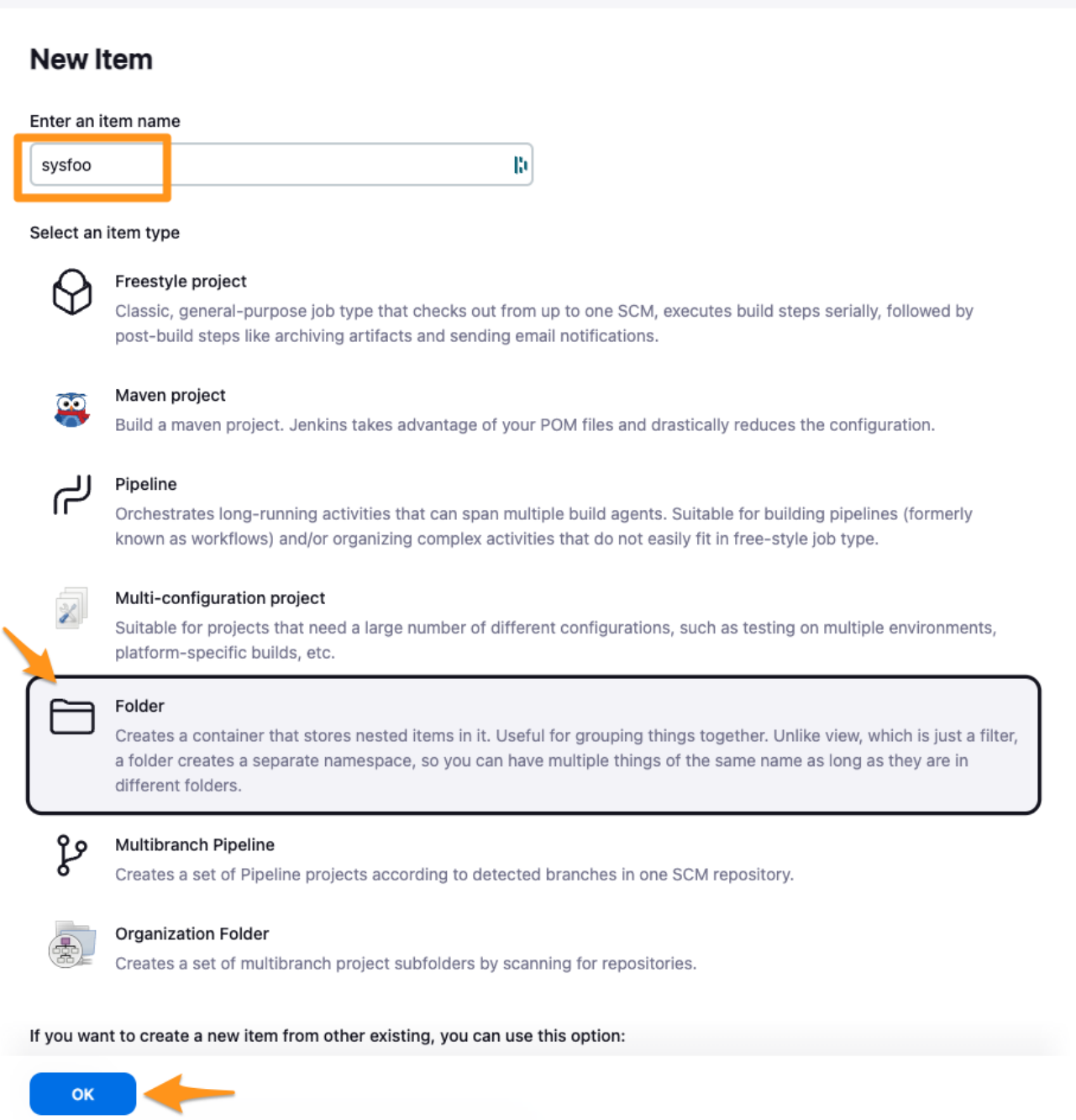



Once installed, clicl on **Go back to the top page** to browse to Jenkins main page.




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






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

sysfoo

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

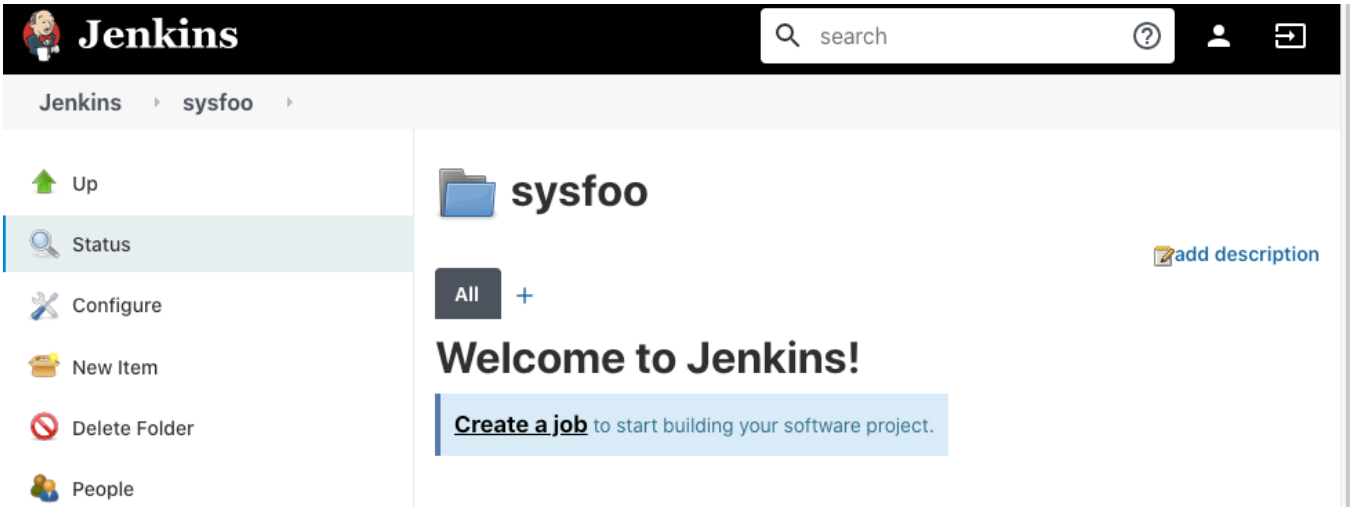
Sharable libraries available to any Pipeline jobs inside this folder. These librarie

Add

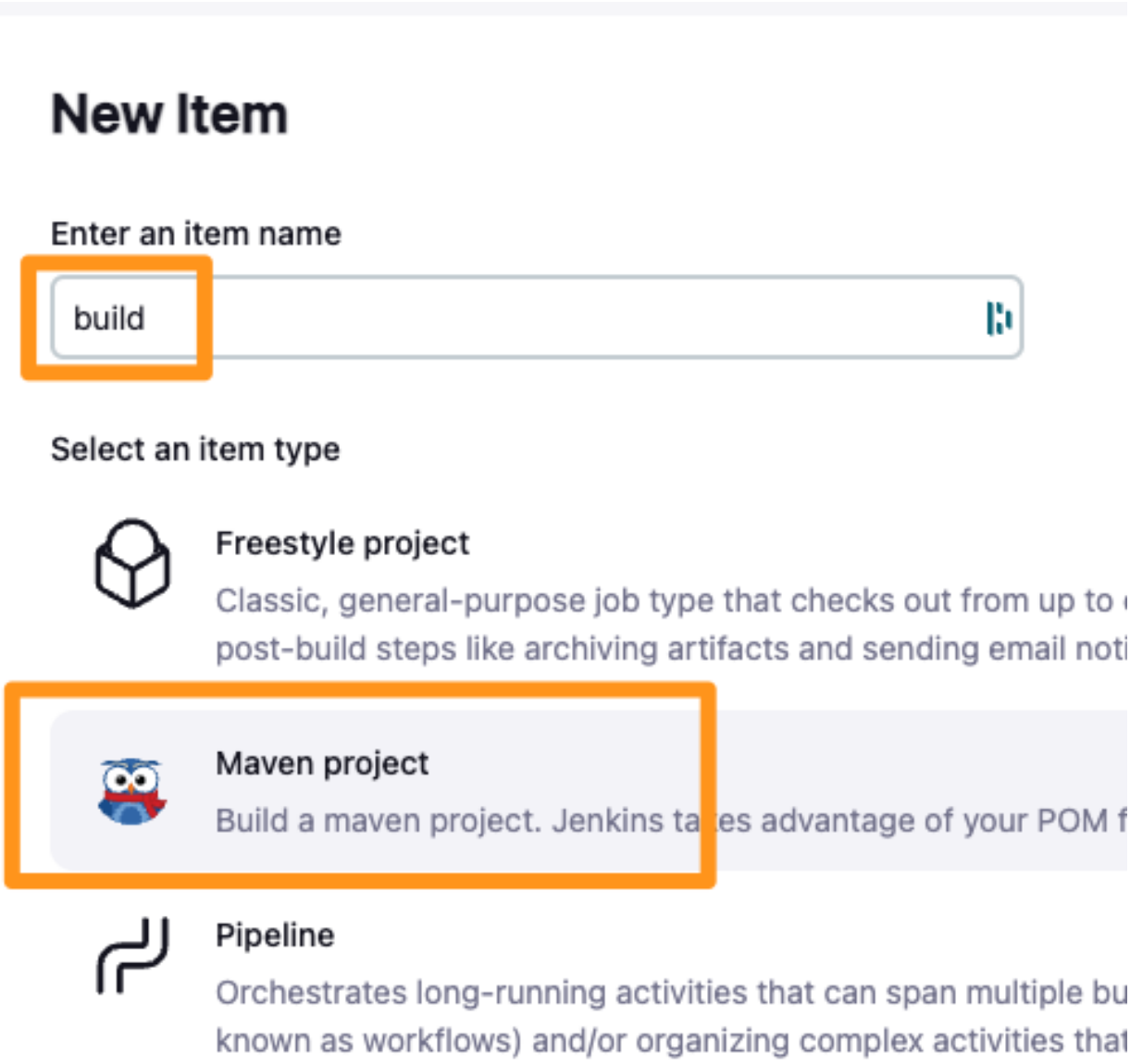
Save

Apply

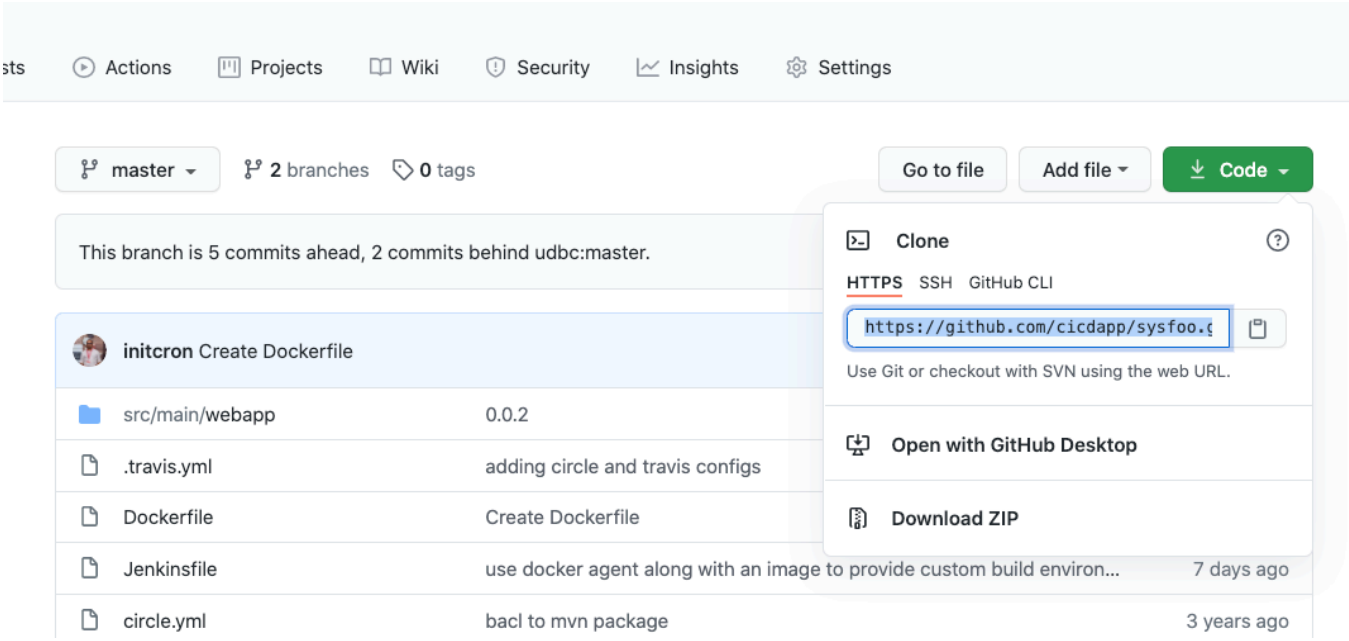
- That creates a folder by name sysfoo, switch to it from top jenkins page.  folder,



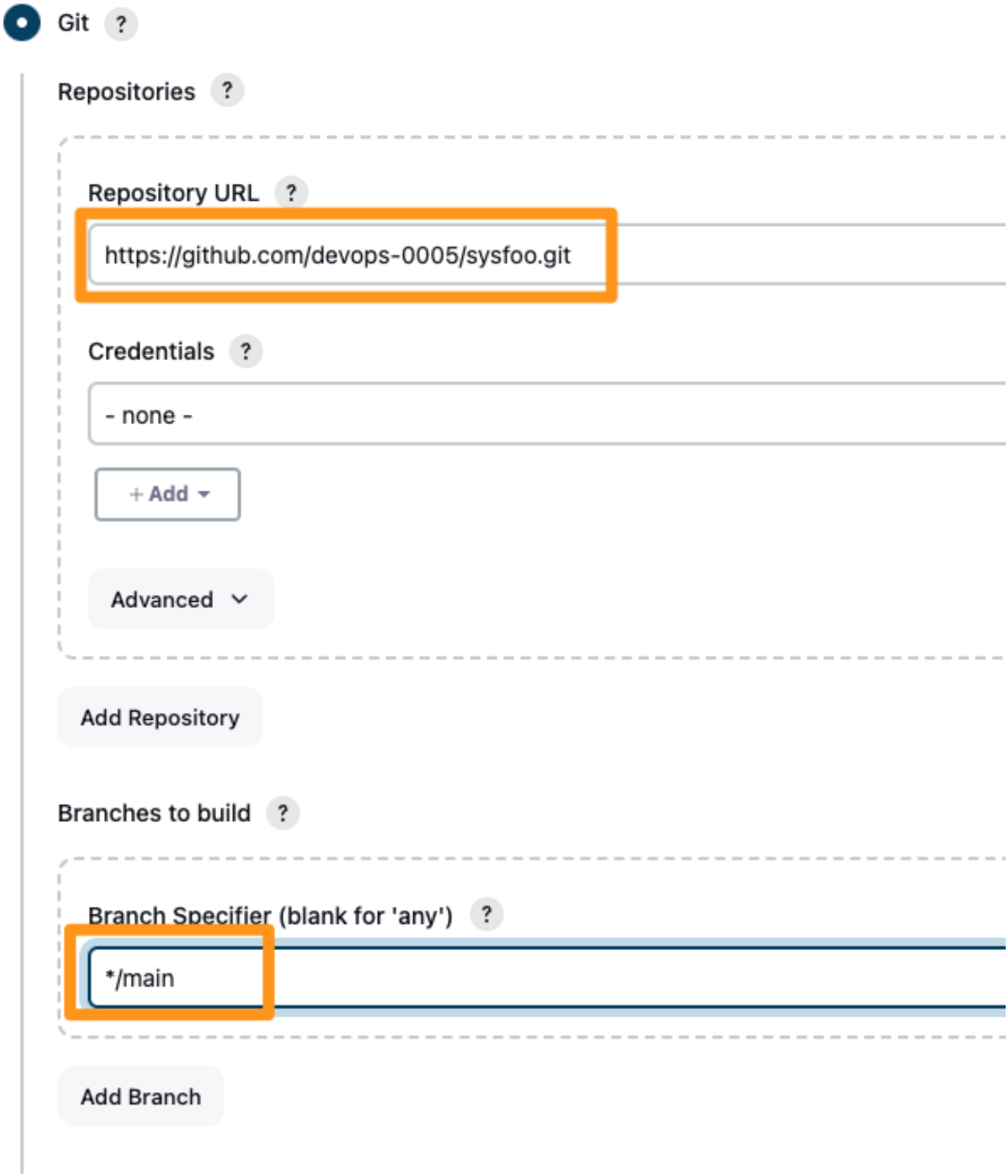

Create a new job and select **Maven** as the type of project. Name the job as **build**




- 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)

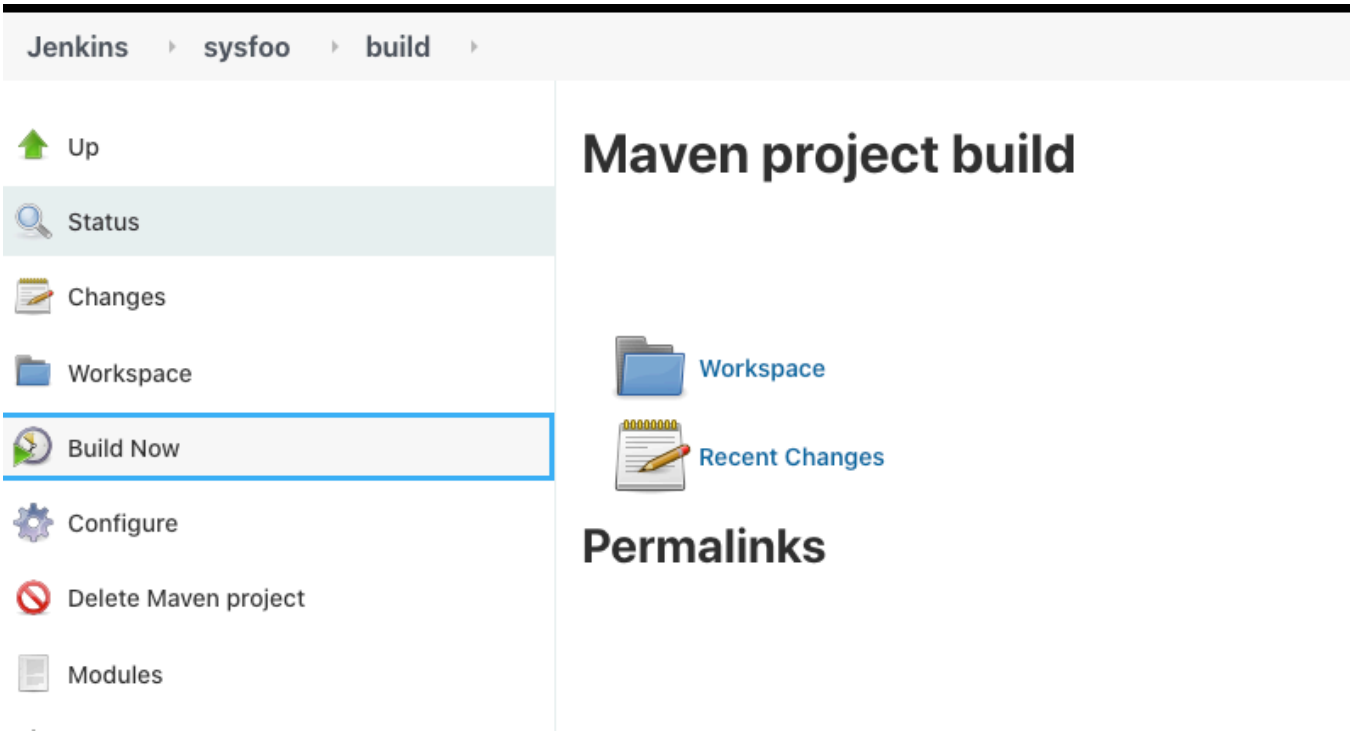



After adding the Repository URL, also change the branch specifier to `*/main`




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

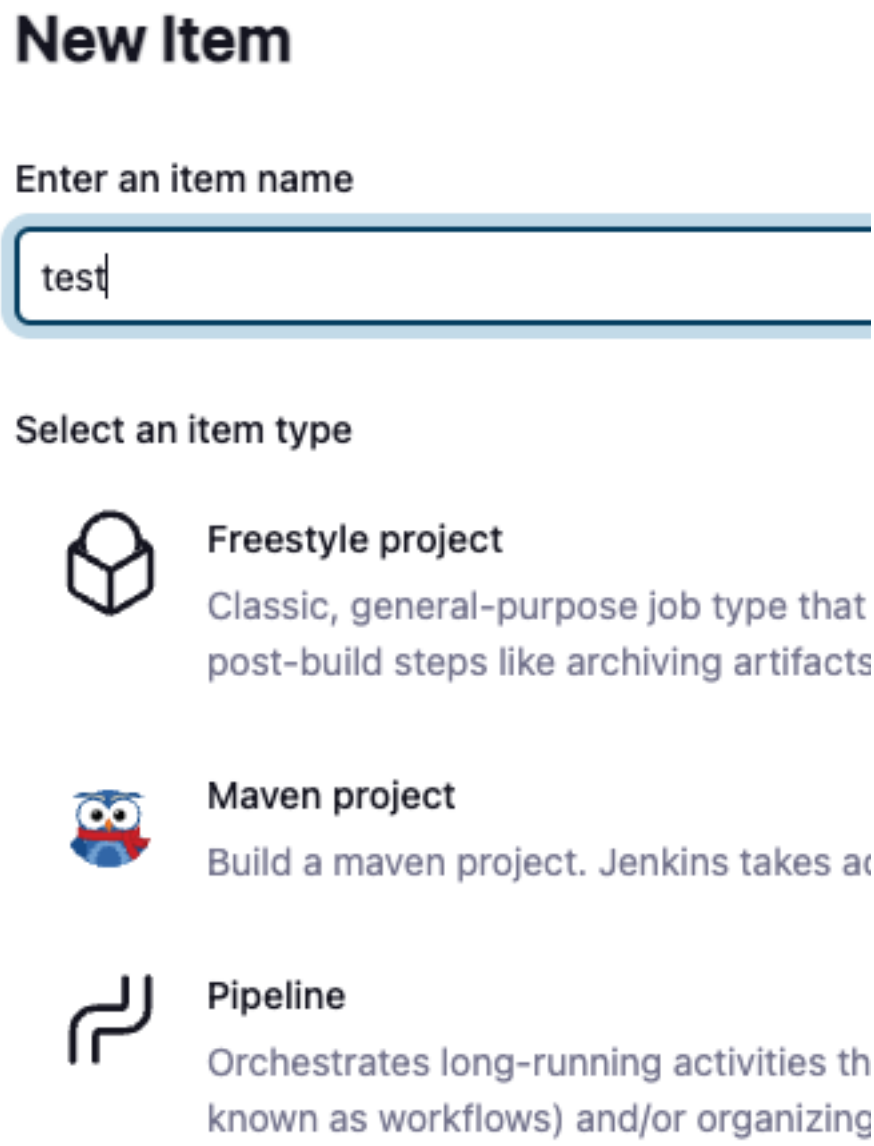

- At the build step, you may want to provide the path to pom.xml. In this example its right inside the root directory of sysfoo, so nothing to change. Save the job and click on **Build Now**. Observe the job status, console output etc.



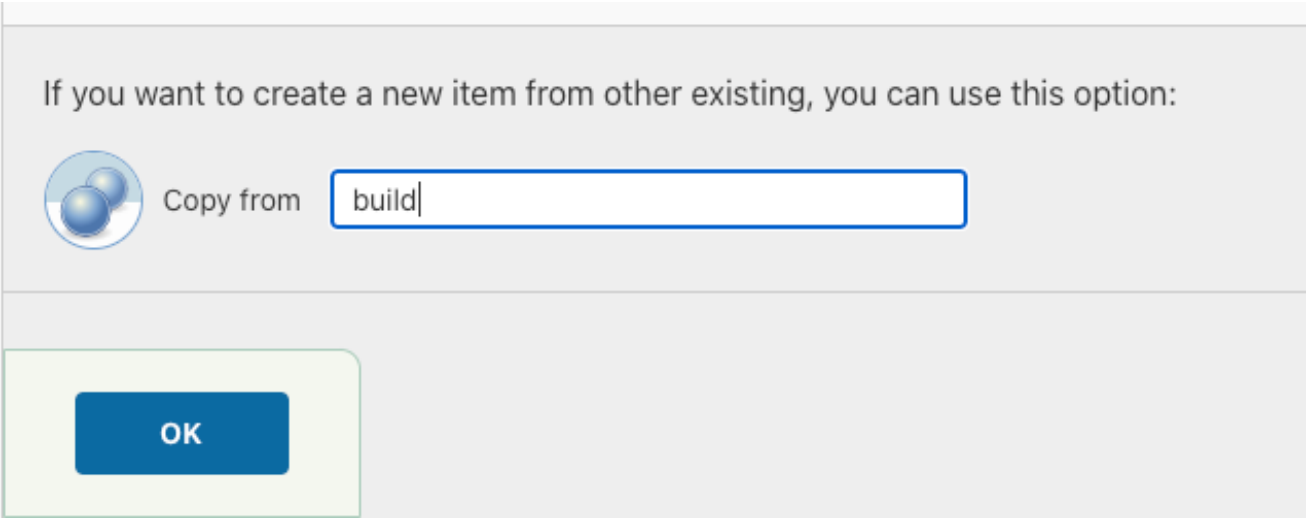

## Adding unit test and packaging jobs

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`,




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.



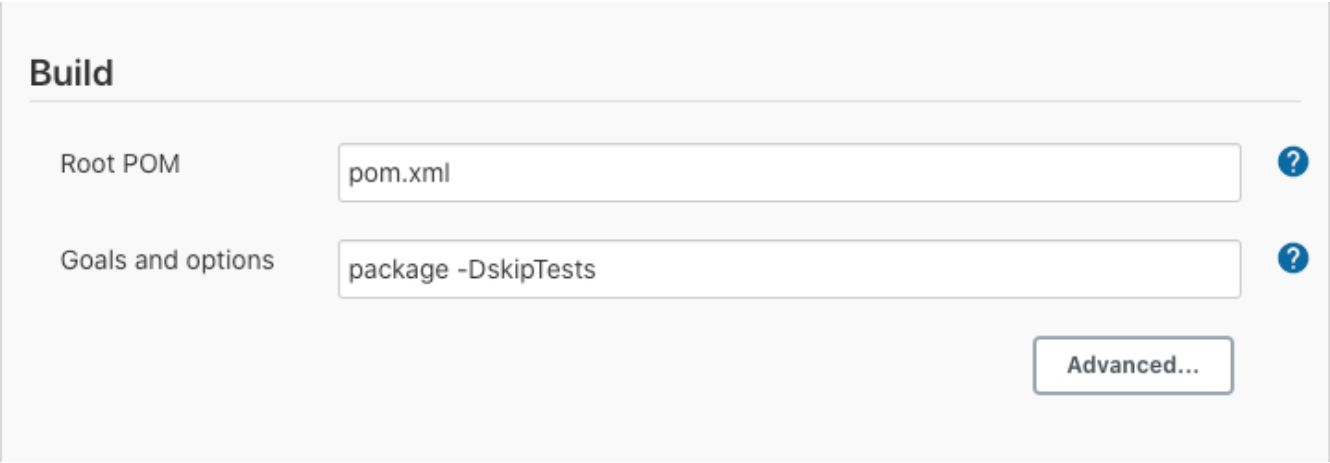

- In `test` job , change your description as `test sysfoo java app`. Source code management ,repository is same .

- under build step, change the goals and option as `clean test` and remaining will be same, so save the job and build.


- 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.

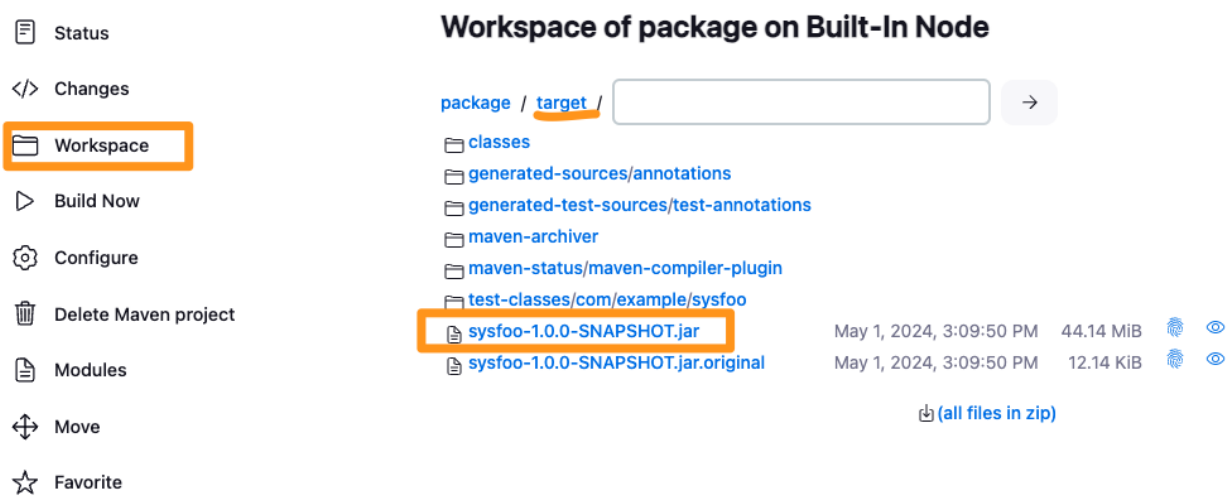
- Update the description as `package sysfoo java app, create jar` and only change in the configuration is build step.

- In the build step for the package job, change the goal to `package -DskipTests`, save the changes and build.

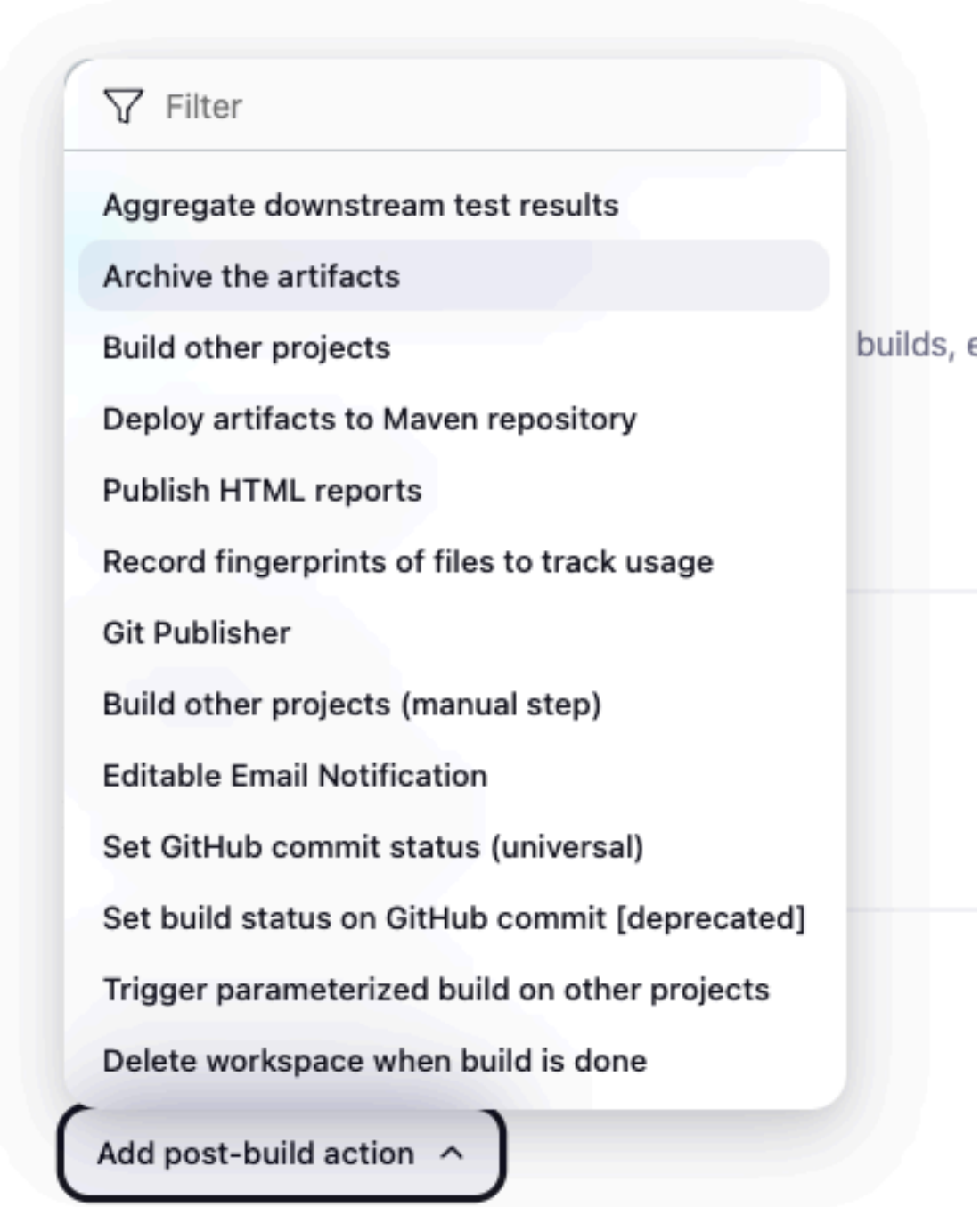


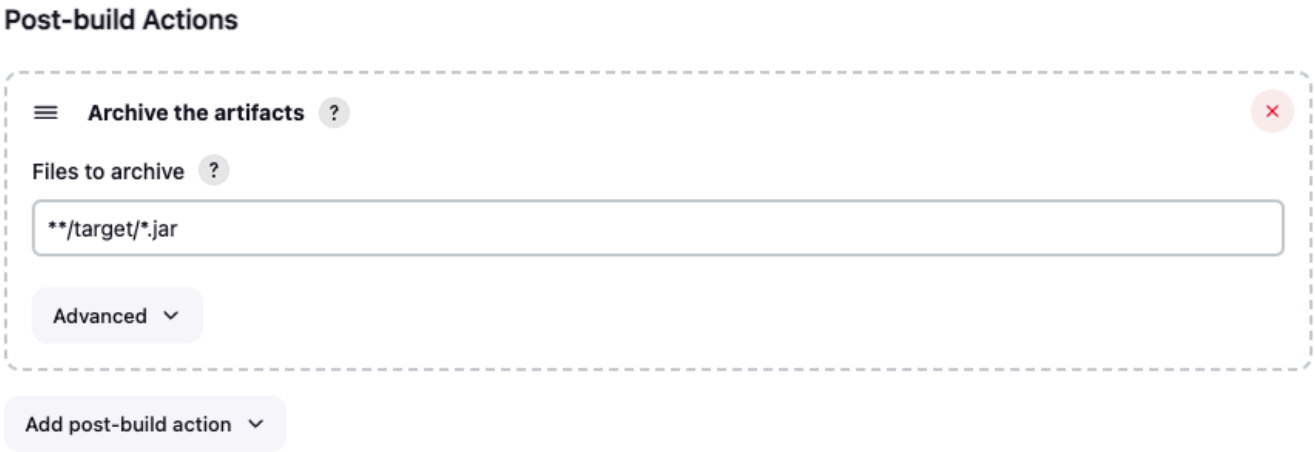

## Archiving the  Artifacts

- 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.




- 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, e


Post-build Actions
Archive the artifacts
Files to archive
**/target/*.jar
Advanced
Add post-build action

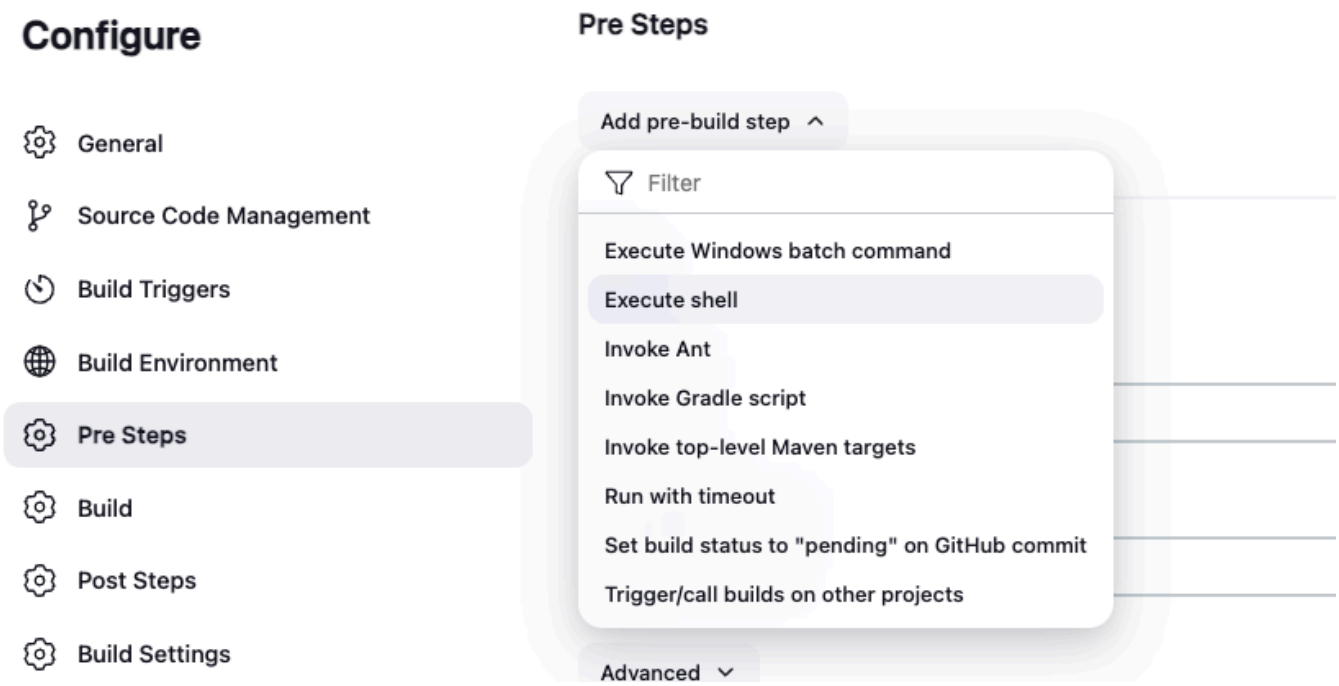
- Save the changes and build the job. Once build successful, check the project page to find out your artifacts right out there.

## Configuring Artifact Version

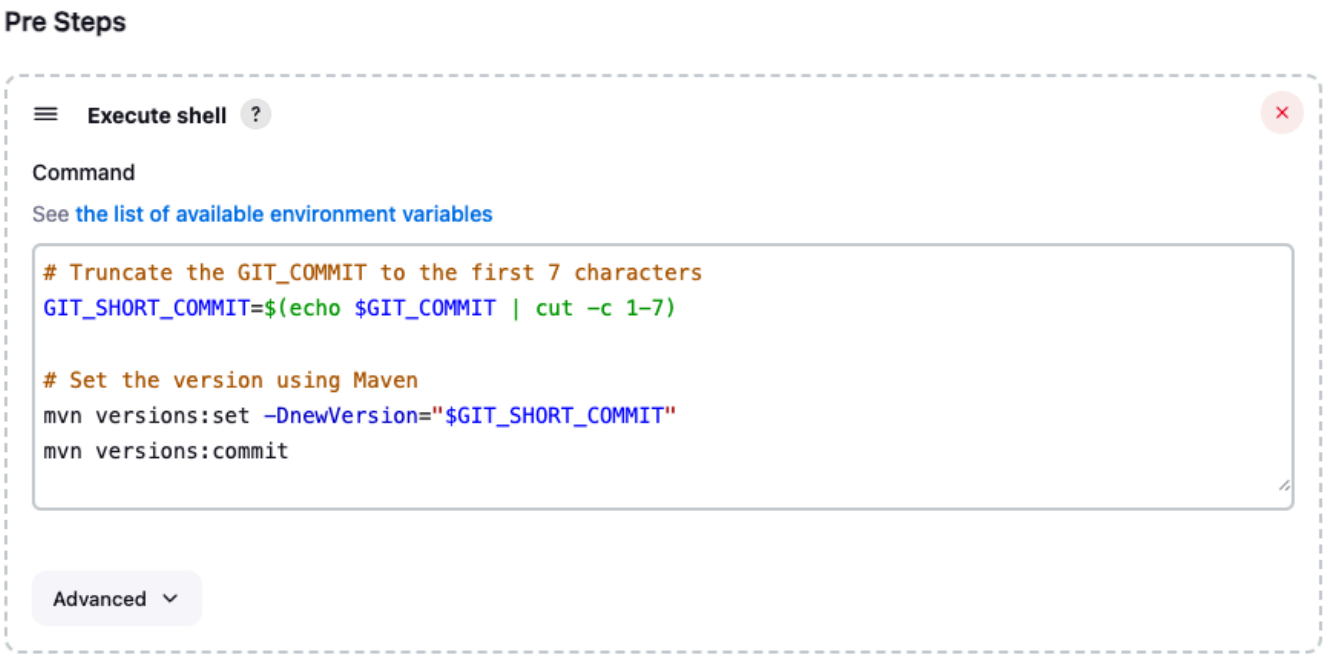
Instead of you needing to manually generating the artifact version e.g. sysfoo-1.0.0-SNAPSHOT.jar , you could have it generated using a git commit id which is unique for every change. To achieve that, you could add a script such as follows:

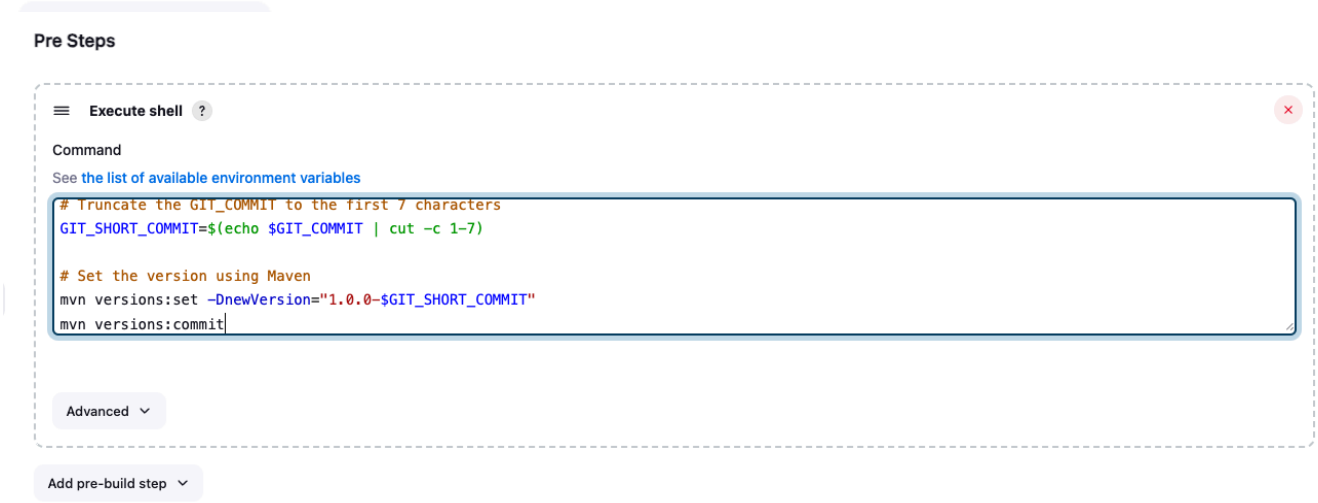
```
# Truncate the GIT_COMMIT to the first 7 characters
GIT_SHORT_COMMIT=$(echo $GIT_COMMIT | cut -c 1-7)

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

Add it using Pre Steps => Execute shell

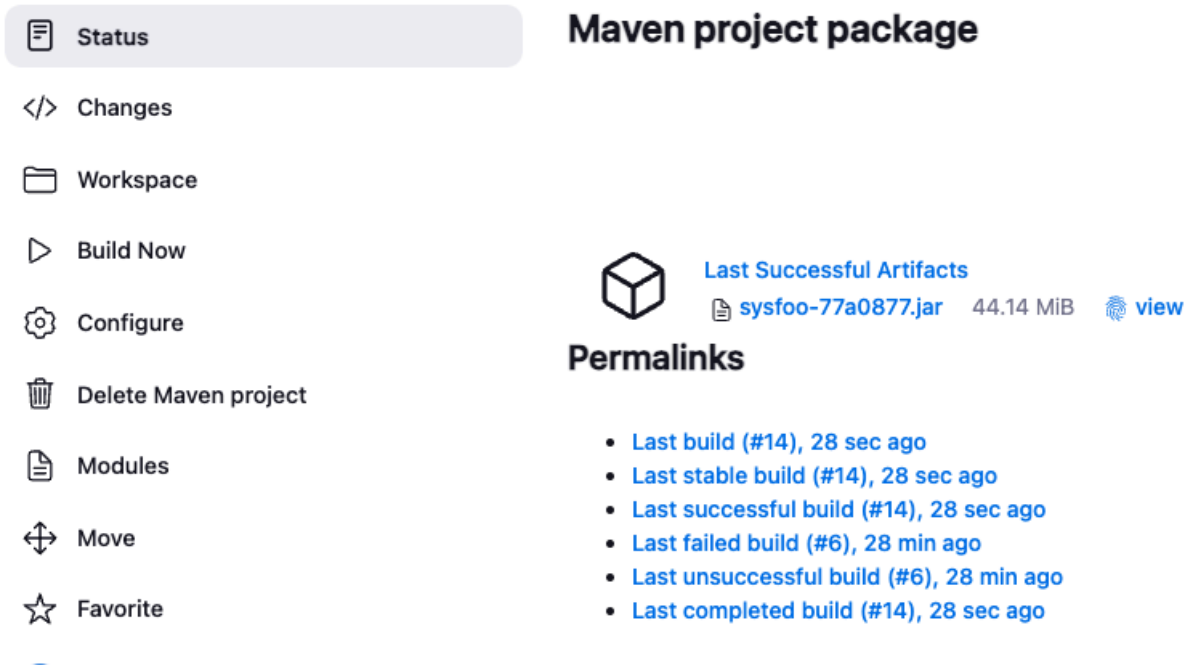



and copy over the script above in the command box




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

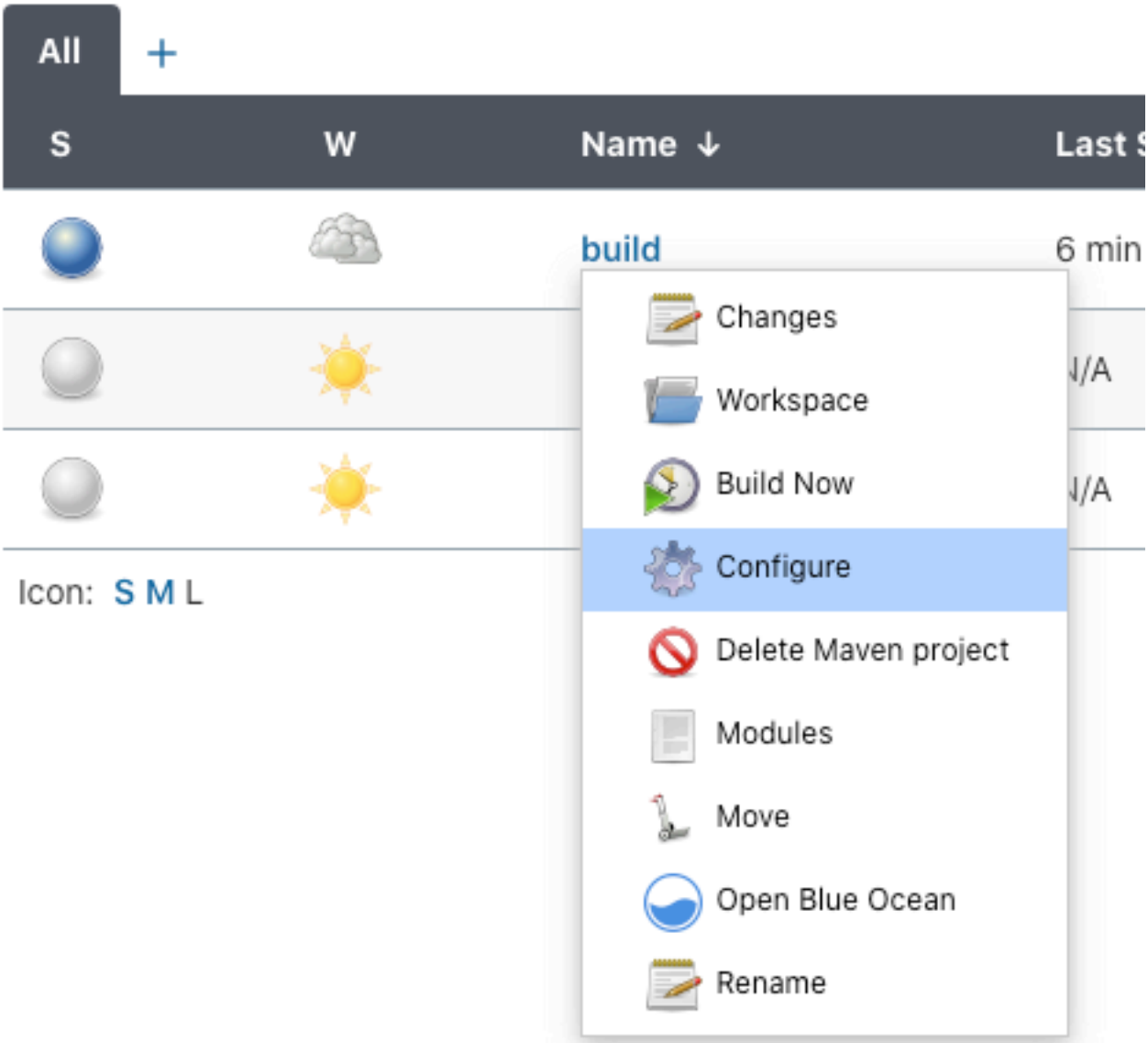


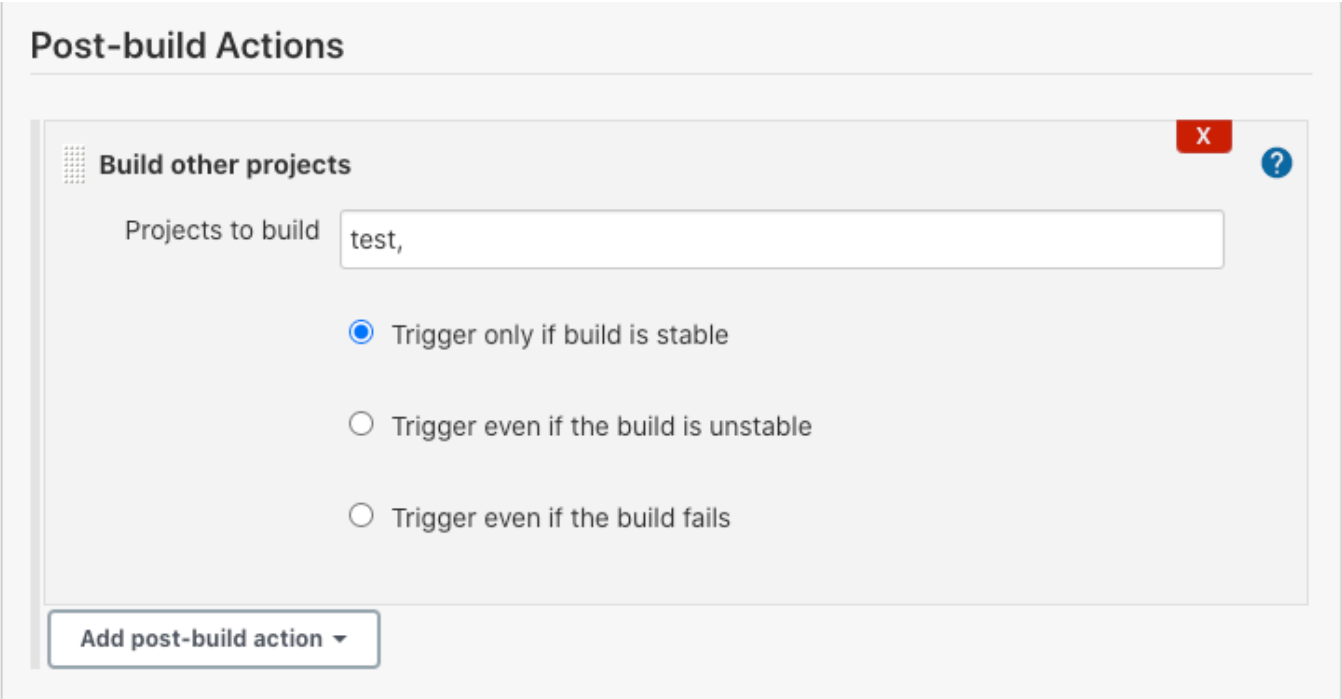

## Connecting Jobs with Upstreams and Downstreams

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.

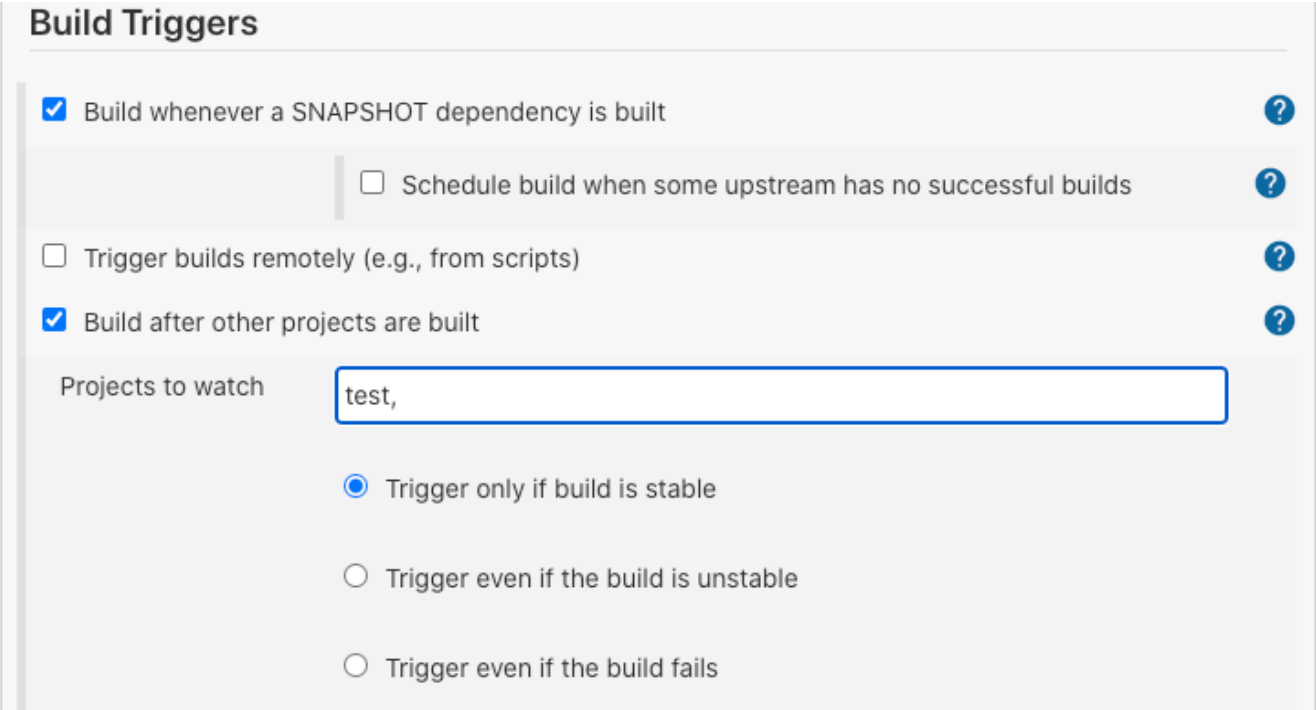
Follow the following steps to setup upstream and downstream

- 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




- 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



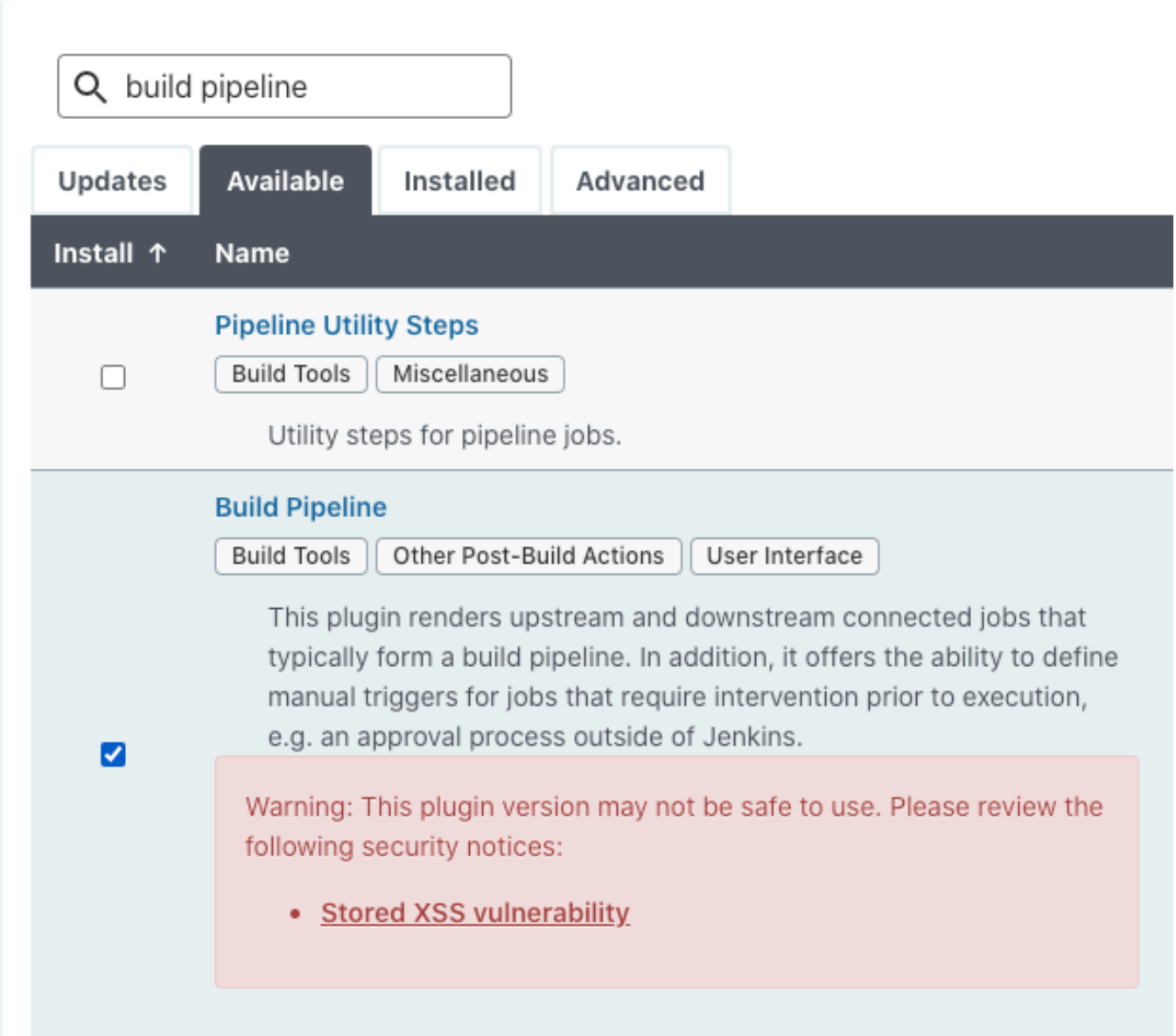

this defines the upstream for package.

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.

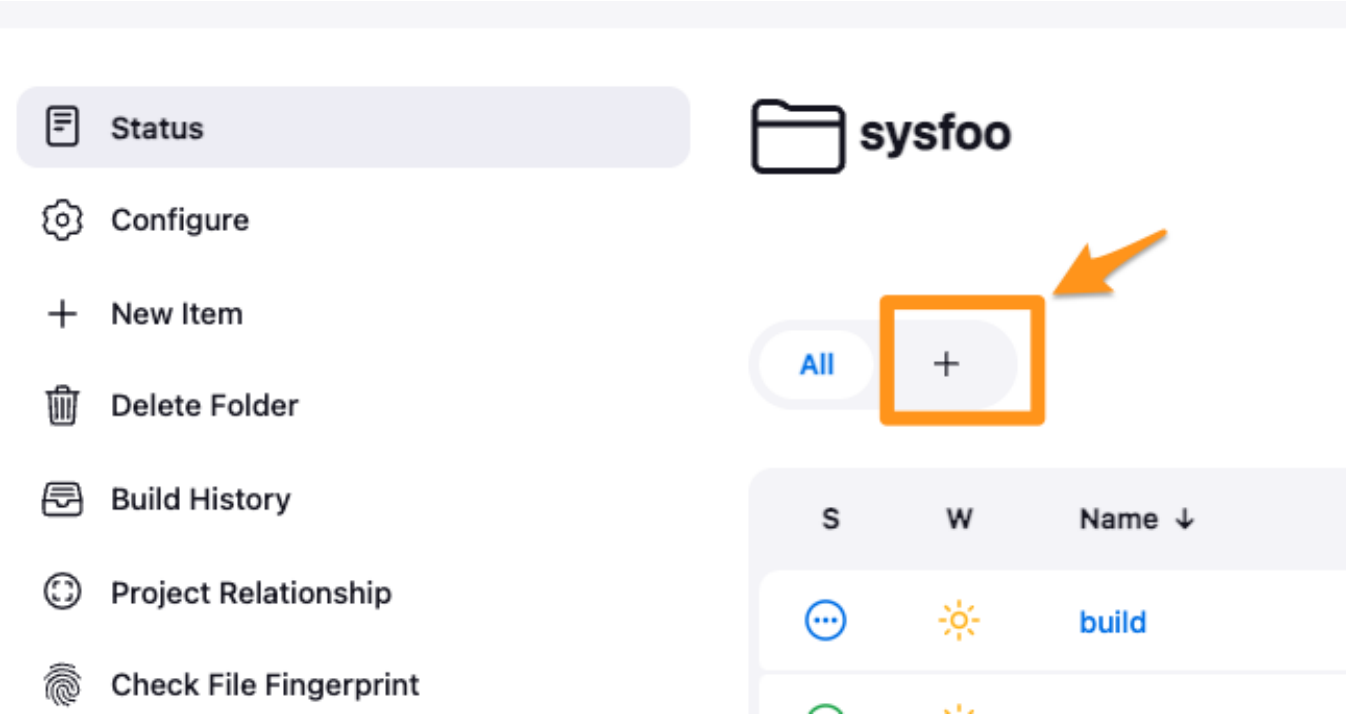


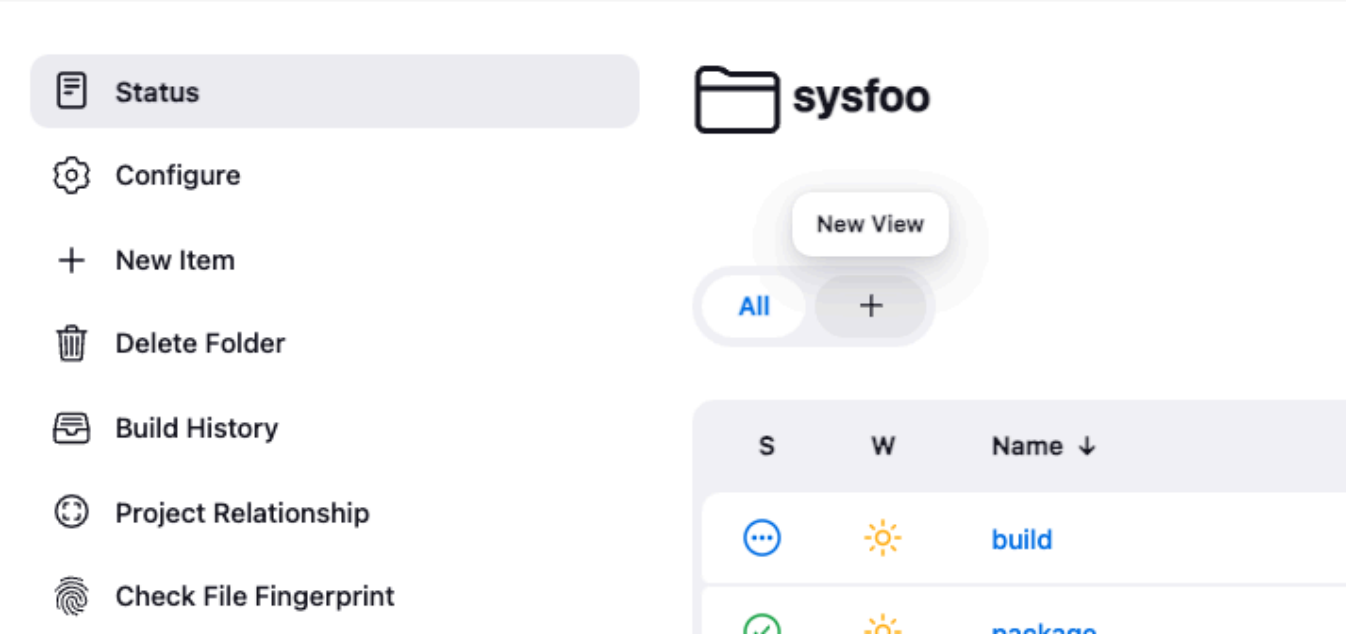

You could install this plugin without a restart.

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

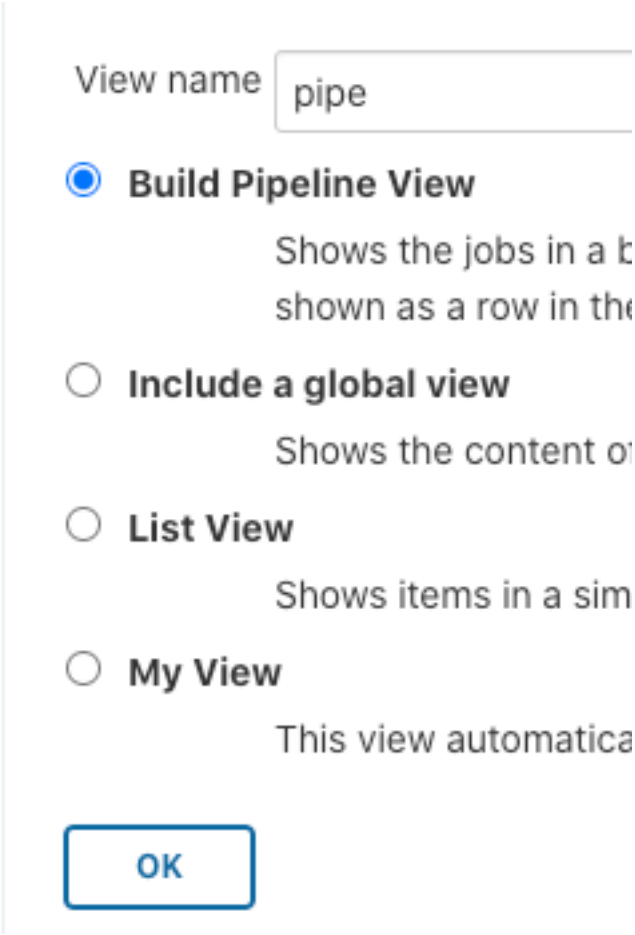
> you are not going to use this in a production setup and only to understand the pipeline concept visually, its ok to proceed. Do not use this plugin in a live/production environment.

- 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.

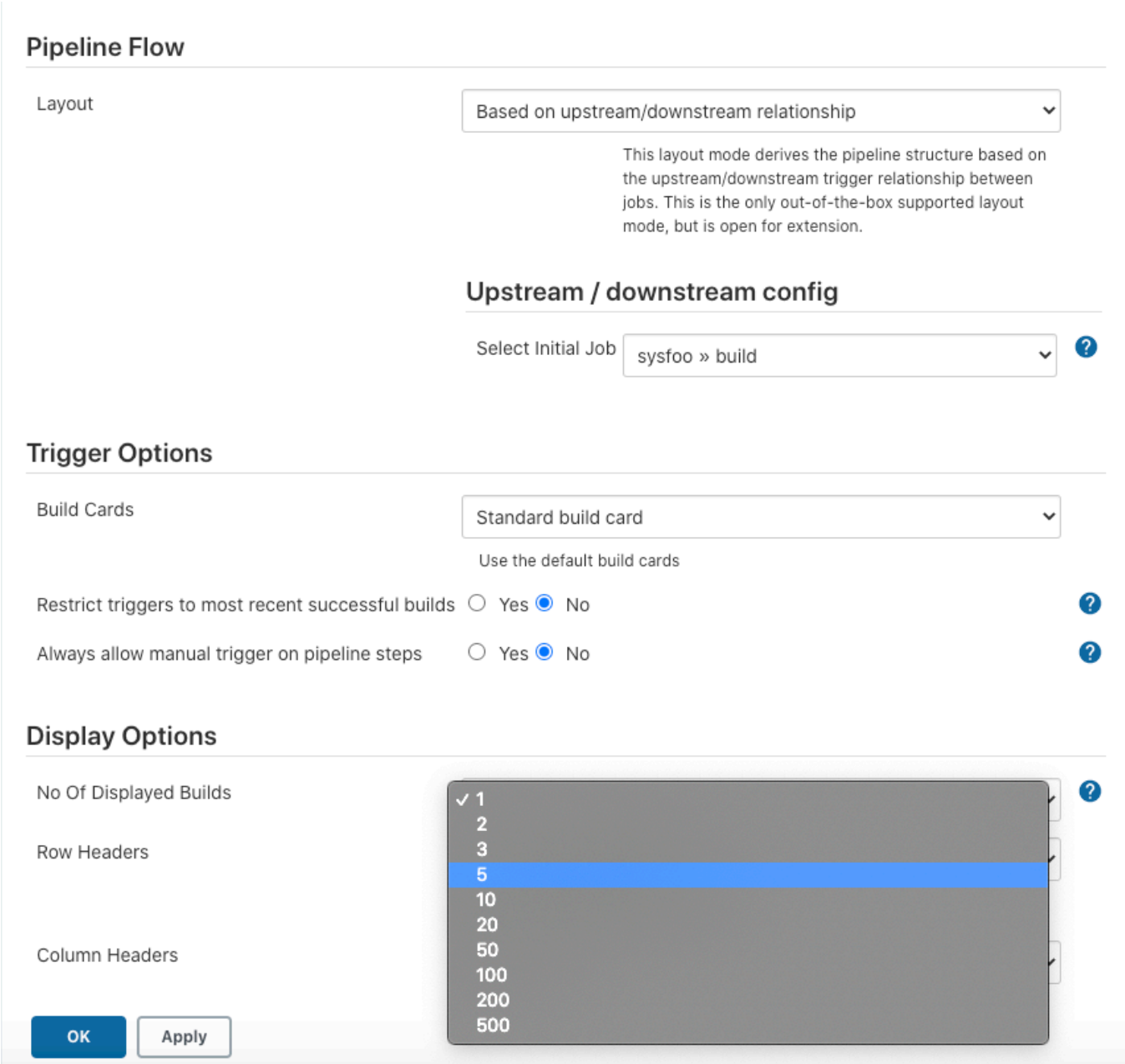


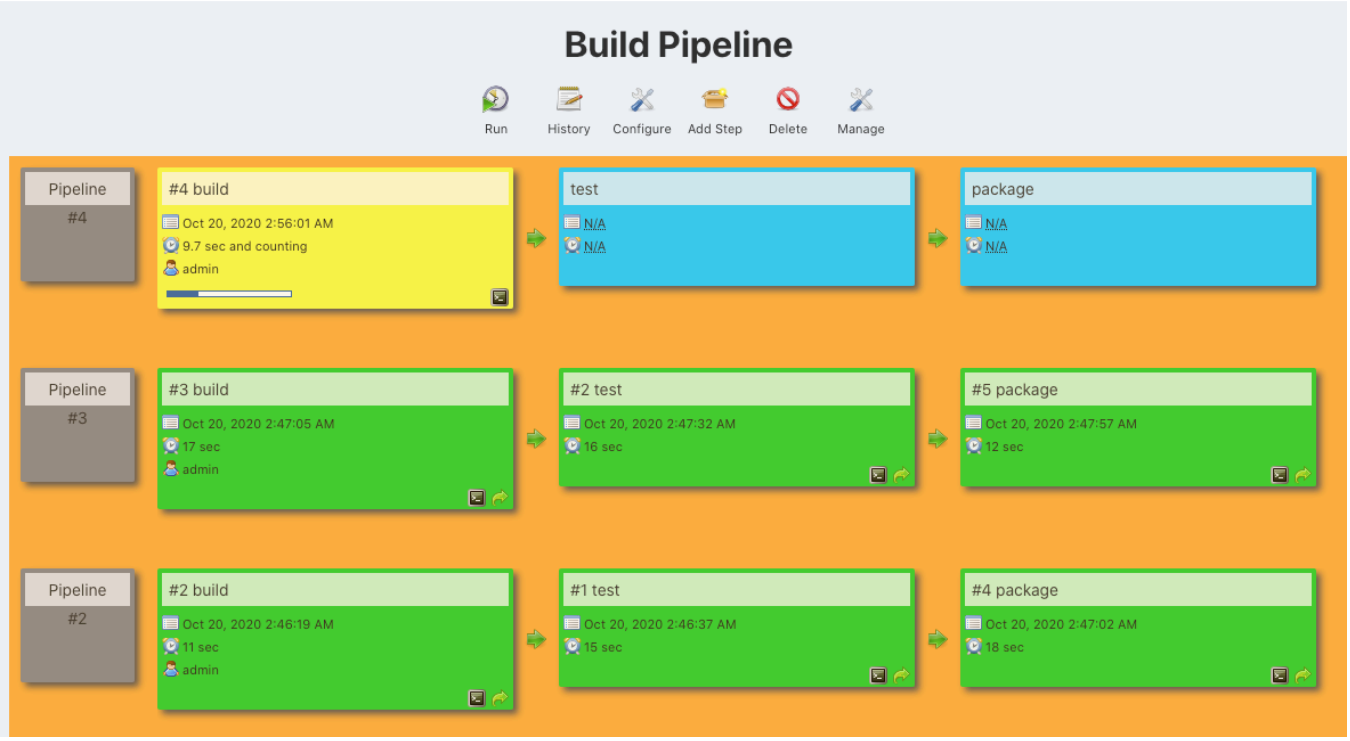

Provide a name and select **Build Pipeline View**




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




- 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.

Jenkins 2 search

Jenkins › instavote › worker-build › 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