Lab 4. Setting Up Continuous Integration with Jenkins

By the end of this lab exercise, you should be able to:

- Set up Jenkins using Docker.
- Take a walkthrough of the Jenkins web interface and essential configurations and start creating freestyle and maven jobs.
- Integrate with tools such as NodeJS and Maven.
- Integrate with GitHub and set up build triggers.
- Set up pipelines which run automated builds and unit tests.

Install Docker Compose

Follow the directions at https://docs.docker.com/compose/install/ to install Docker Compose on your machine.

Set Up Jenkins with Docker

Here you are going to learn how to set up Jenkins using Docker. Docker must be installed prior to this lab.

You will be running a Jenkins container on your Docker host by using the Jenkins image with version `jenkins/jenkins:2.375-jdk11`. Use the following commands to clone the `devops-repo` directory and launch the Jenkins container:

```
git clone https://github.com/lftraining/devops-repo.git
```

```
cd devops-repo/setup
```

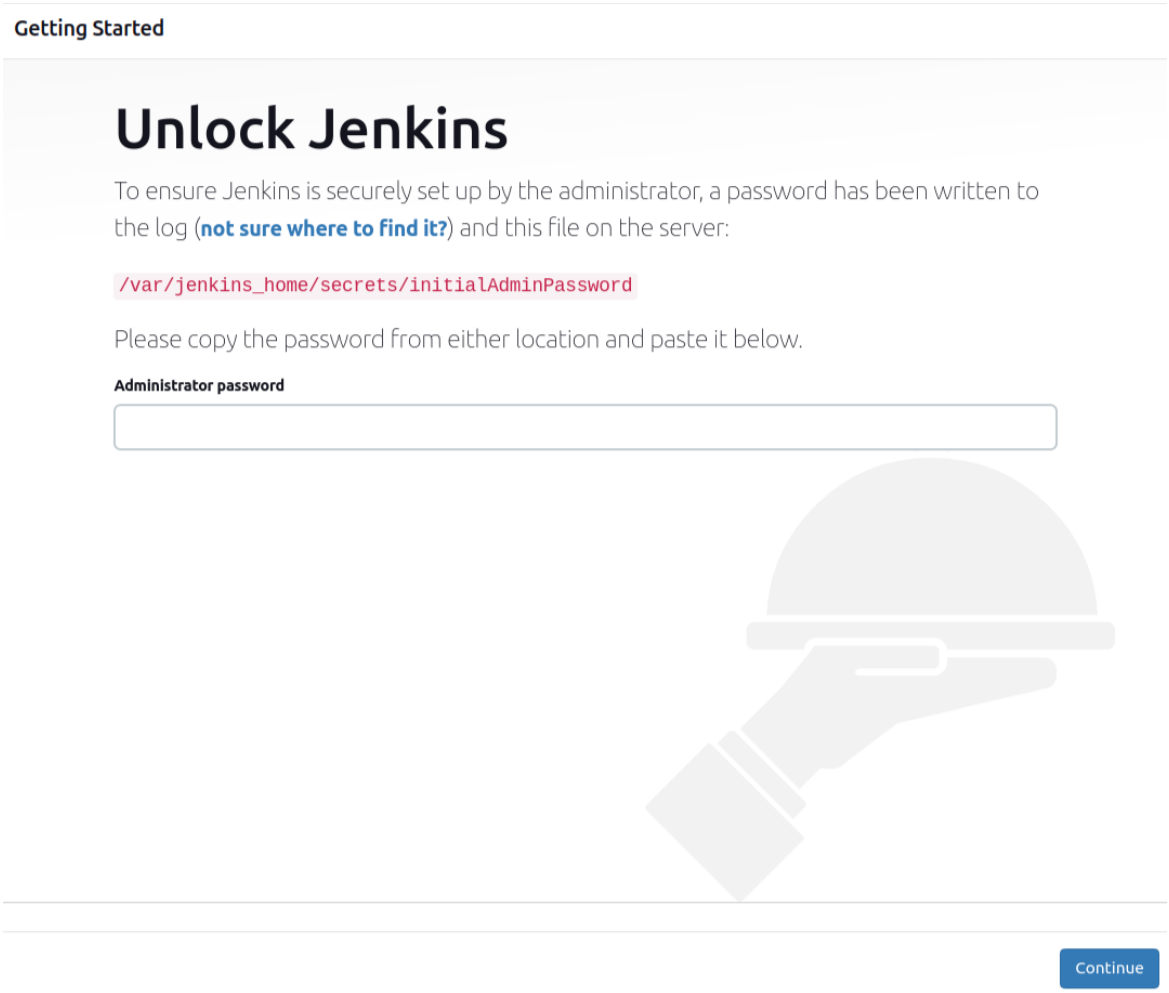
Inside the `devops-repo/setup` directory is the `docker-compose.yml` and the Dockerfile that we will be working with. The Jenkins container image is

`jenkins/jenkins:2.375-jdk11`.

```
docker compose build
```

```
docker compose up -d
```

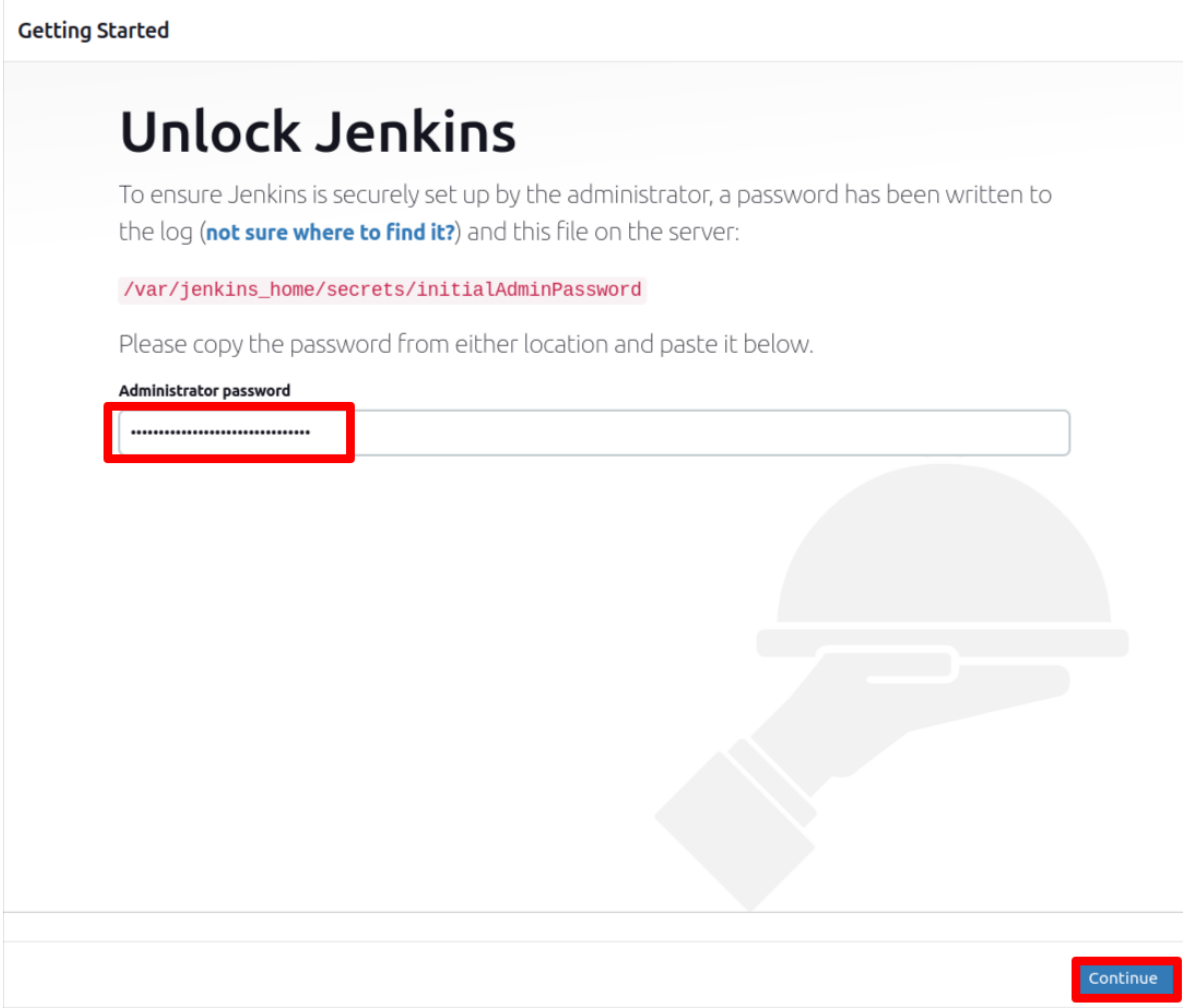
Access the Jenkins UI by browsing to **`http://IPADDRESS:8080`**. In this case, it will most likely be **`http://localhost:8080`** if you are on your own machine. If you are in the cloud, you will need to find the public IP of your cloud machine.



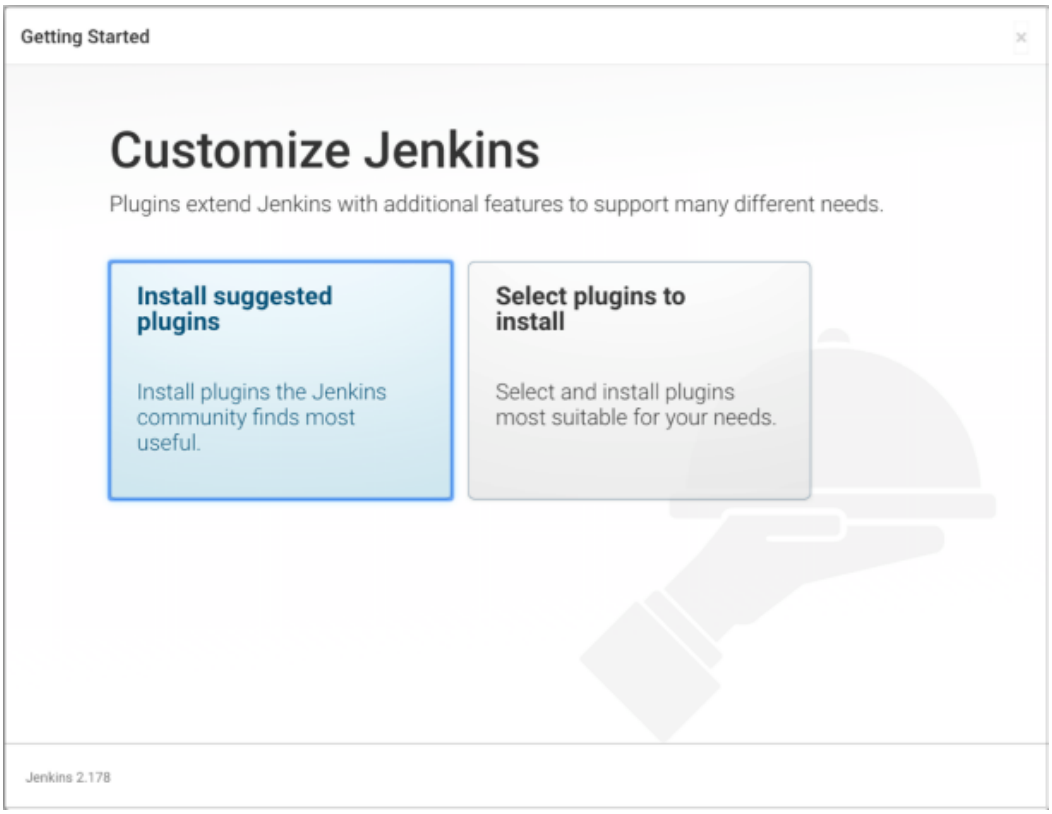
To fetch the **`initialAdminPassword`** use the following command:

```
docker exec -it setup-docker-1 cat /var/jenkins_home/secrets/initialAdminPassword
```

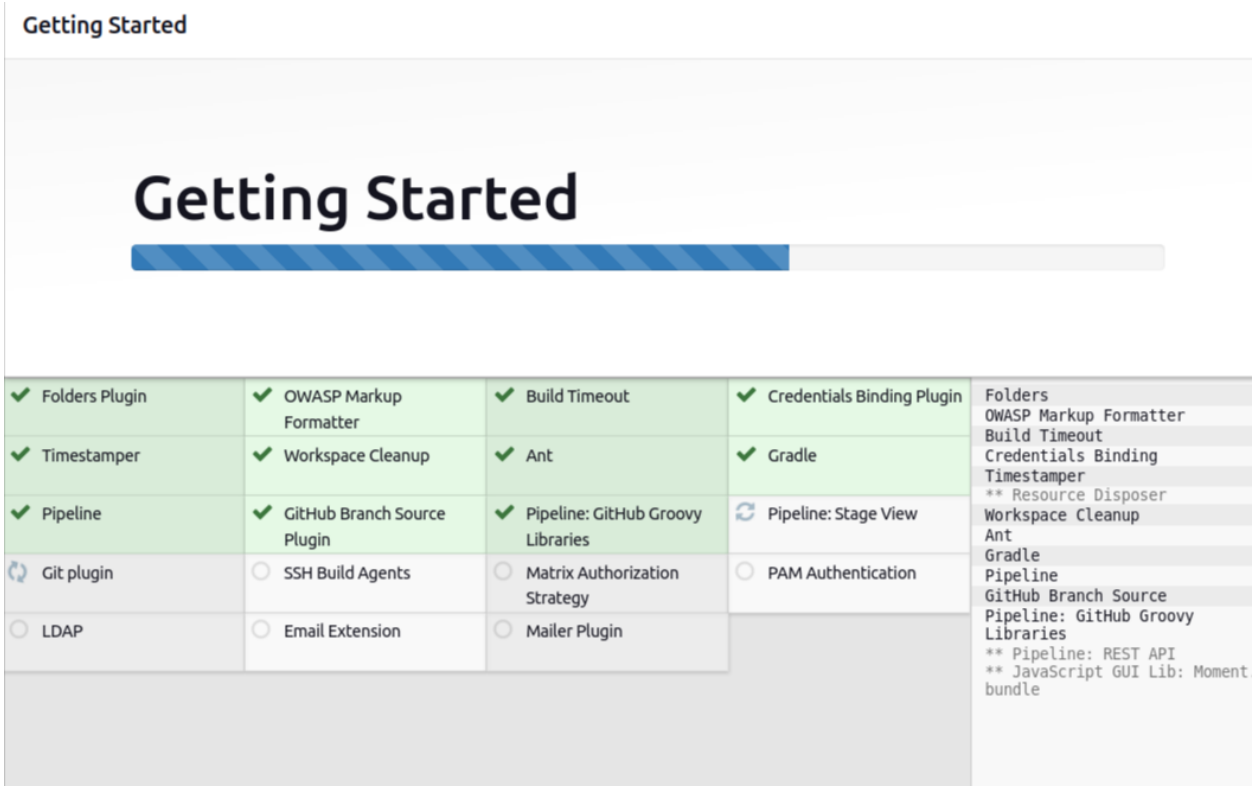
The output will be the initial Jenkins password. Paste it into the Jenkins UI to unlock and click **Continue**.



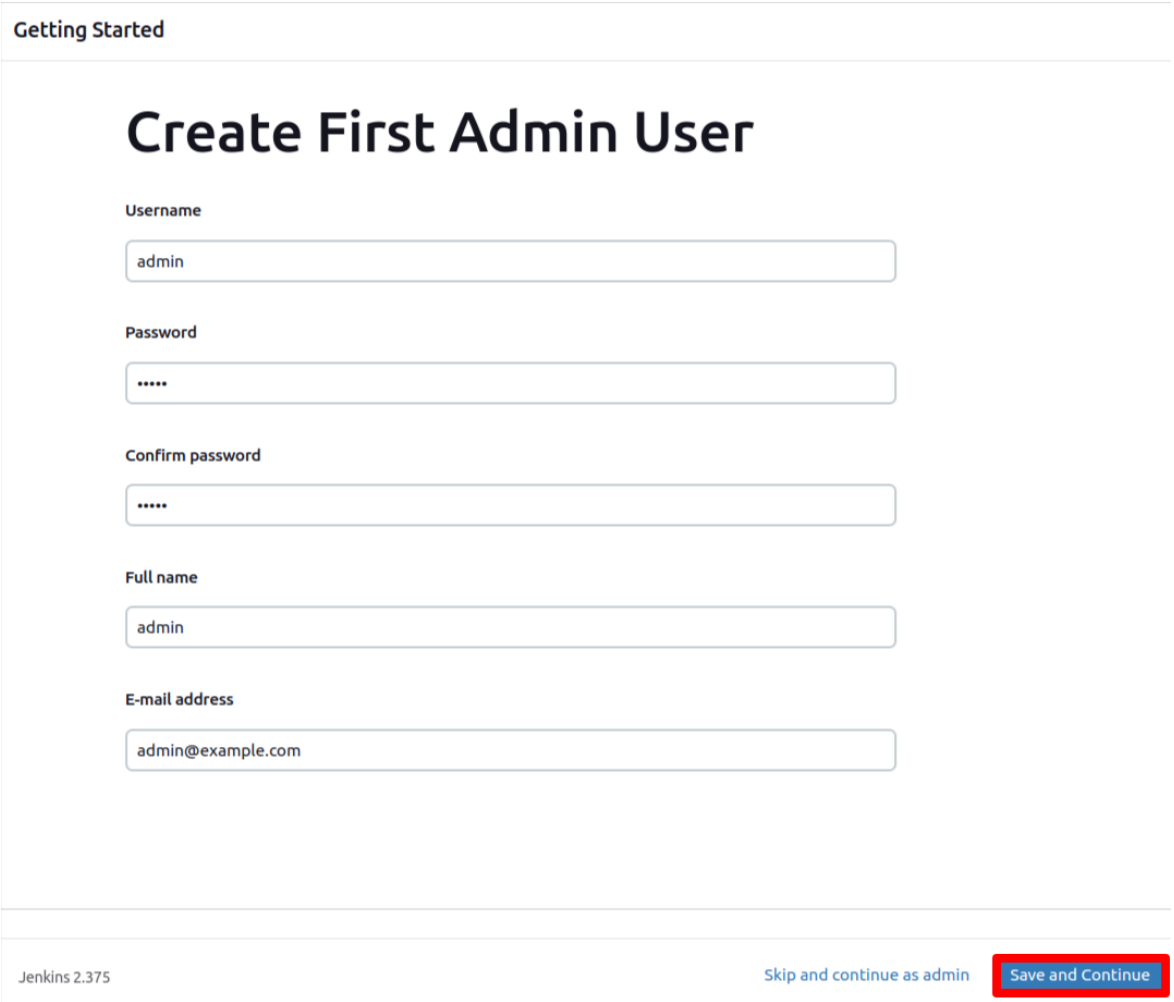
In the next step, choose **Install suggested plugins** to configure the default plugins automatically.



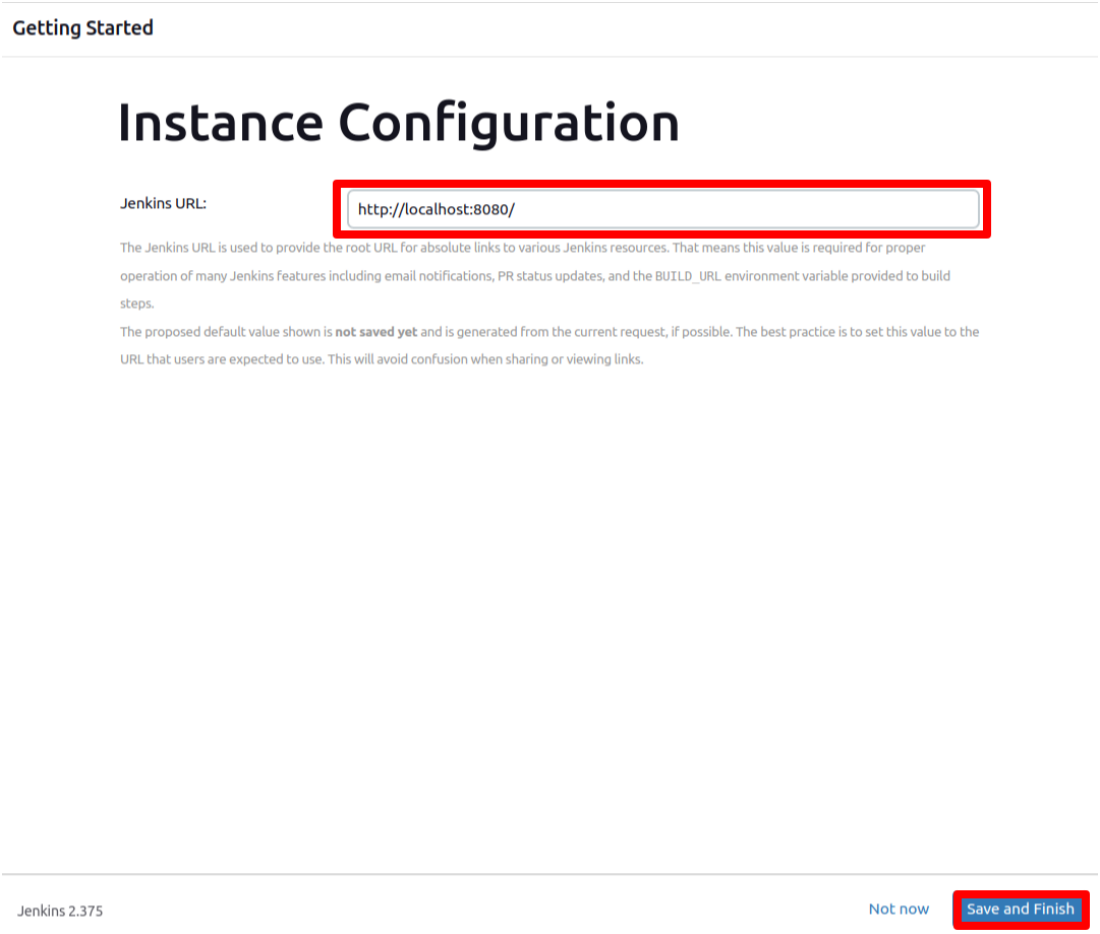
You will be able to observe the progress of the plugin installation process as follows:



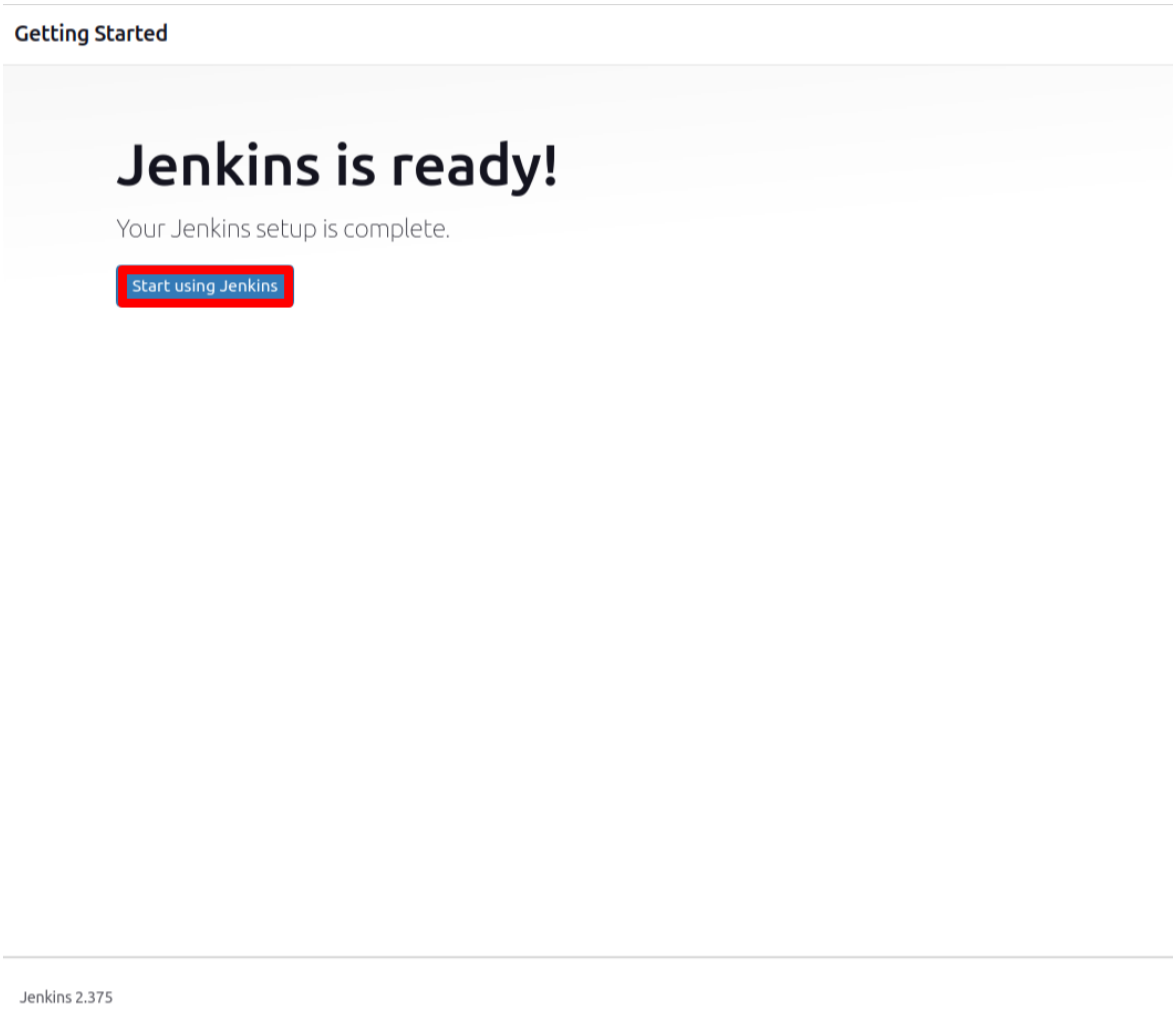
Once the plugins are installed, create the admin user using the form presented. Fill out the page and click **Save and Continue**. You can use whatever username and password you wish.

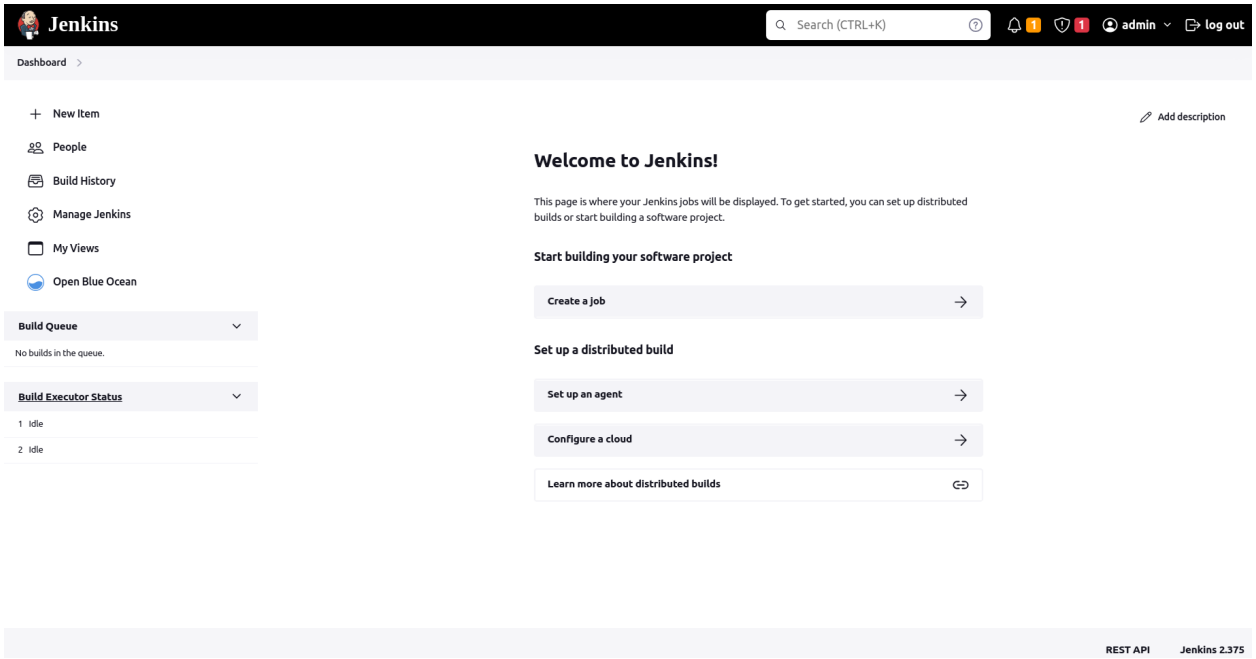


On the **Instance Configuration** page, the URL will depend on if you are on a local machine or a cloud machine. The default should be fine for our purposes. Click **Save and Finish**.



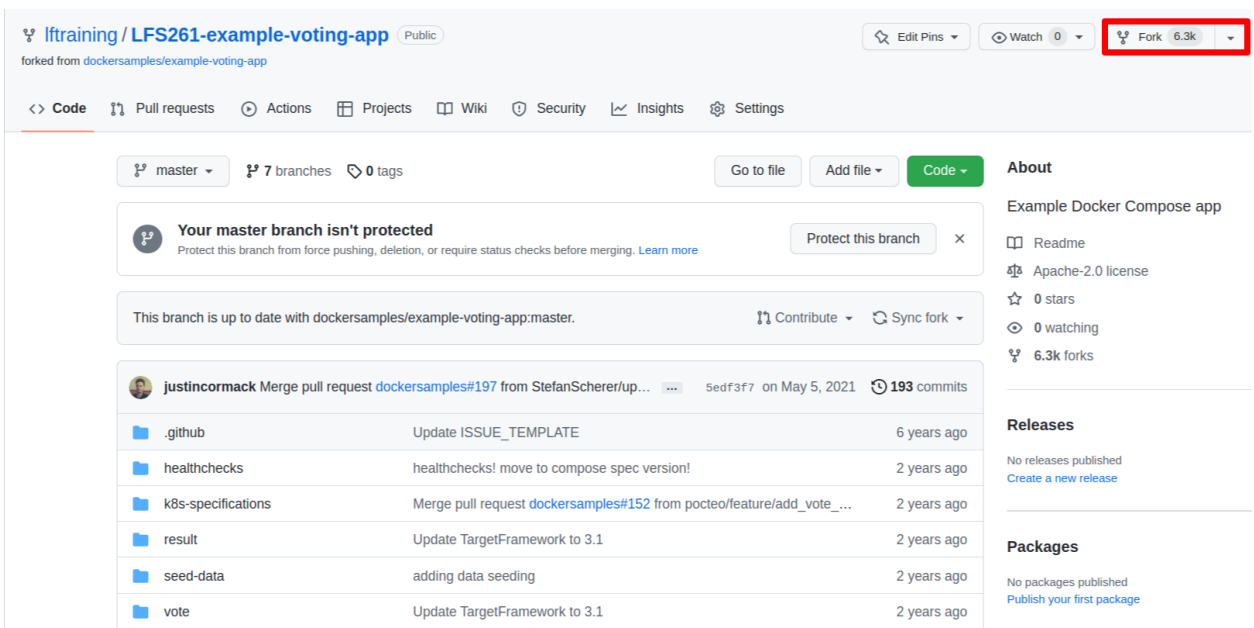
You should see a confirmation page. Click **Start using Jenkins**.





Fork the Voting App

Visit [example-voting-app on GitHub]() and fork the repository onto your Git account.



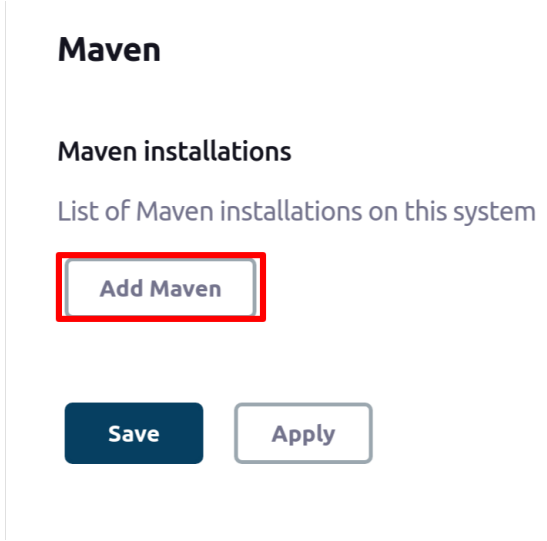
Clone the repository onto your own machine.

Configuring a Maven Build Job

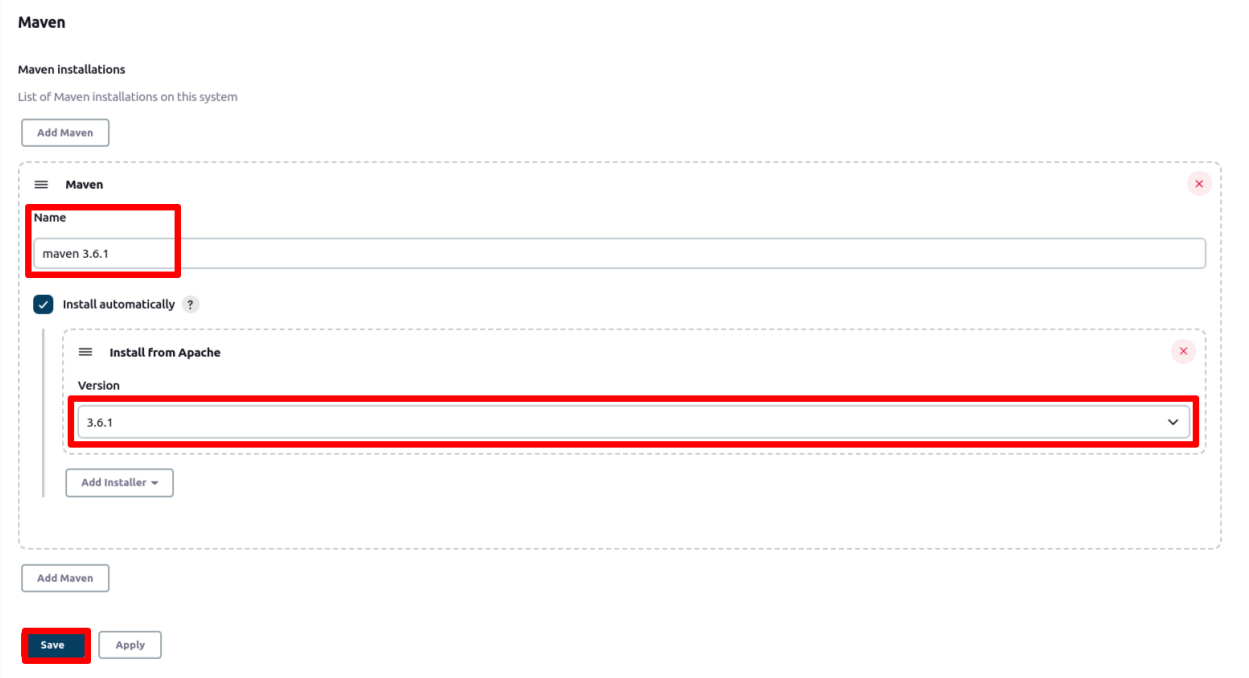
We went over a simple `job-01` in Chapter 6. Now you will build the `worker` application as part of the `example-voting-app` project. This is a Java application that uses Maven as a build tool.

First we will configure a Maven build job.

Go to **Manage Jenkins > Global Tool Configuration** and under the **Maven** section, click **Add Maven**.

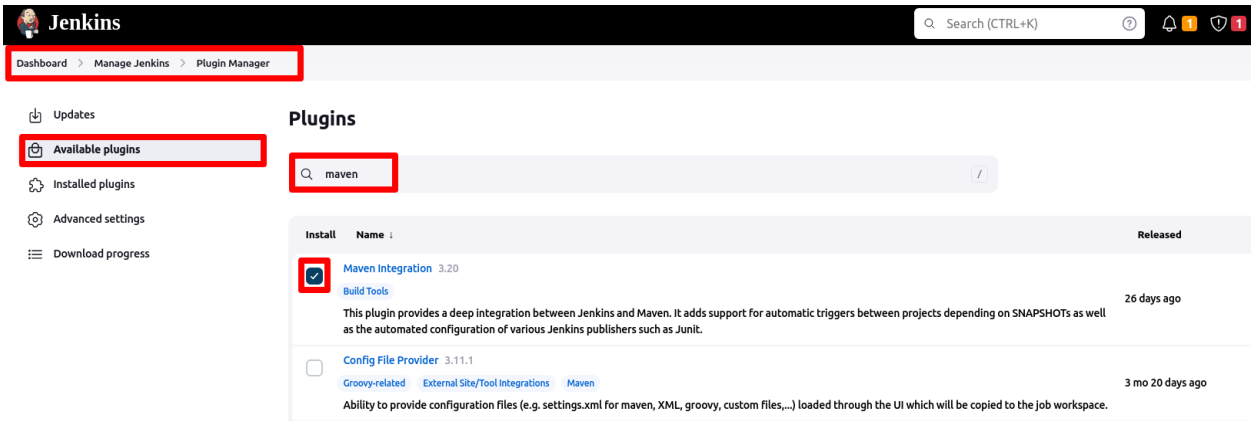


Paste `maven 3.6.1` into the name box and select maven version **3.6.1** from the **Version** dropdown menu under **Install from Apache**. Save the changes:

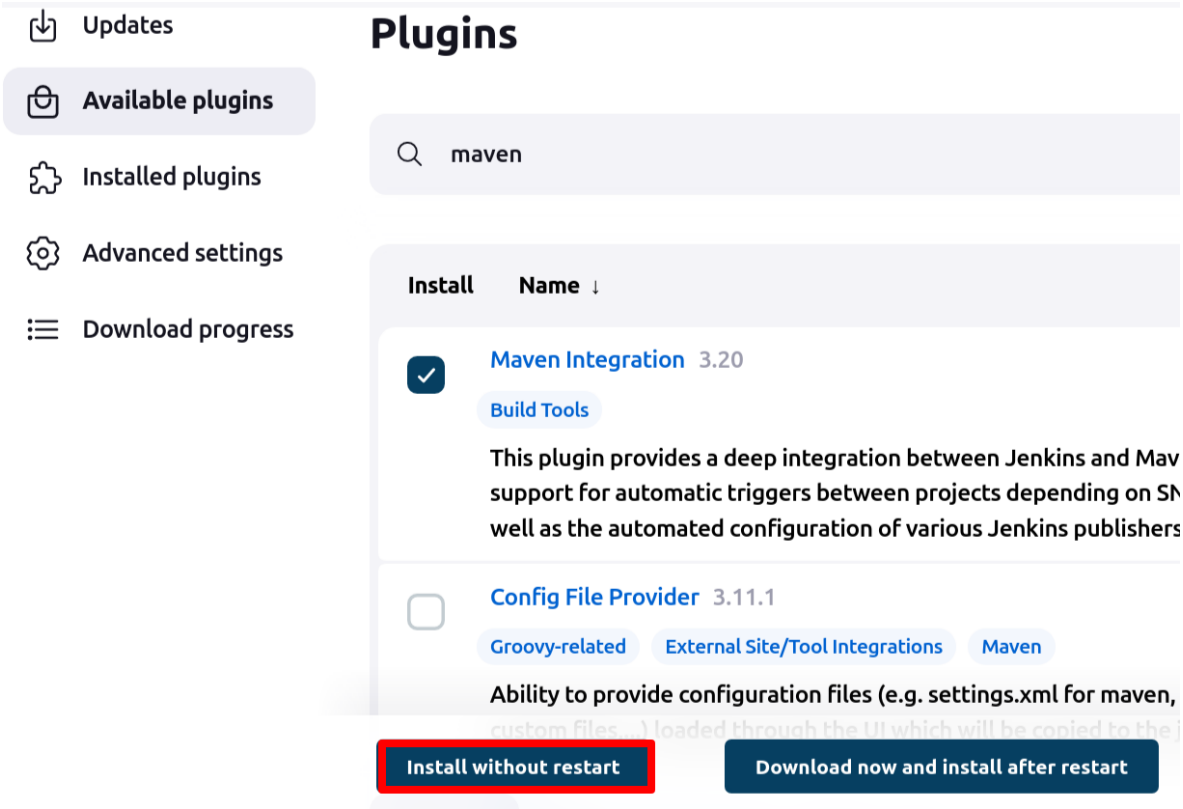


***NOTE**: Remember to use the exact name in this screenshot. This is the name you are going to use to reference the Maven installation in your Jenkinsfile later. If you provide a different name, be aware of it in future labs.*

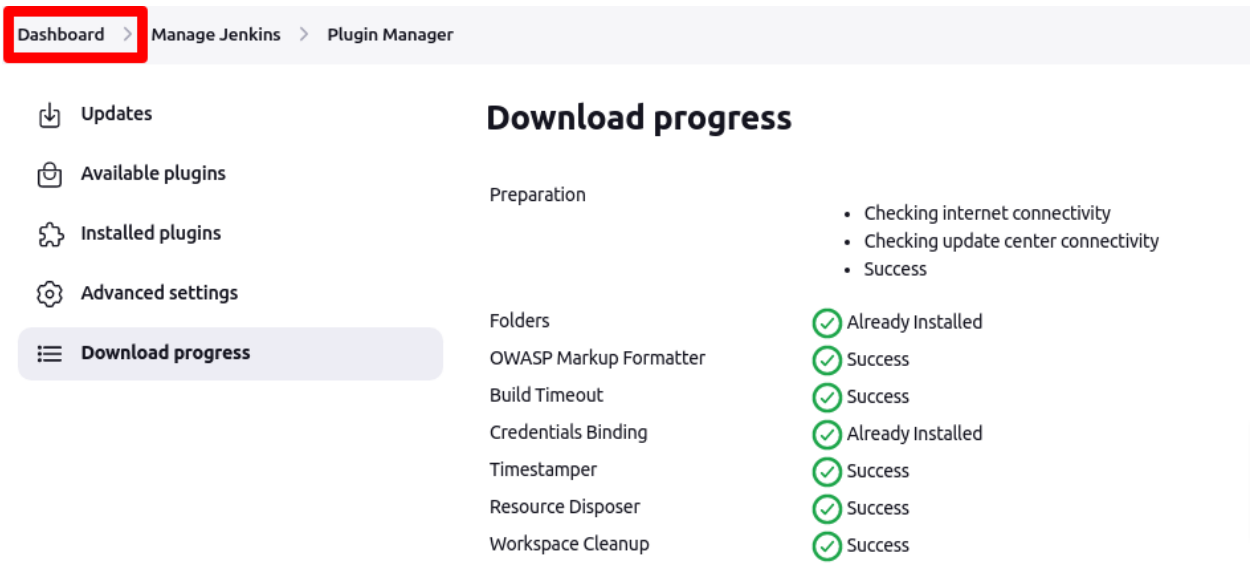
Install the Maven integration plugin. Go to **Manage Jenkins > Manage Plugins > Available**, search for **Maven Integration** plugin and install it without a restart.



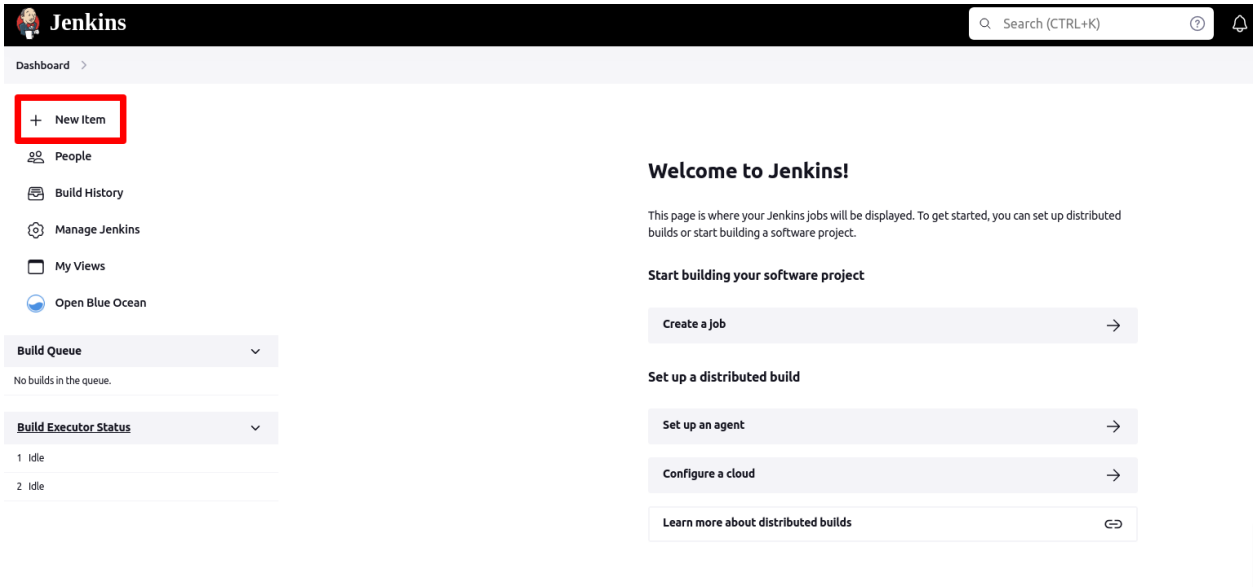
Click **Install without restart**.



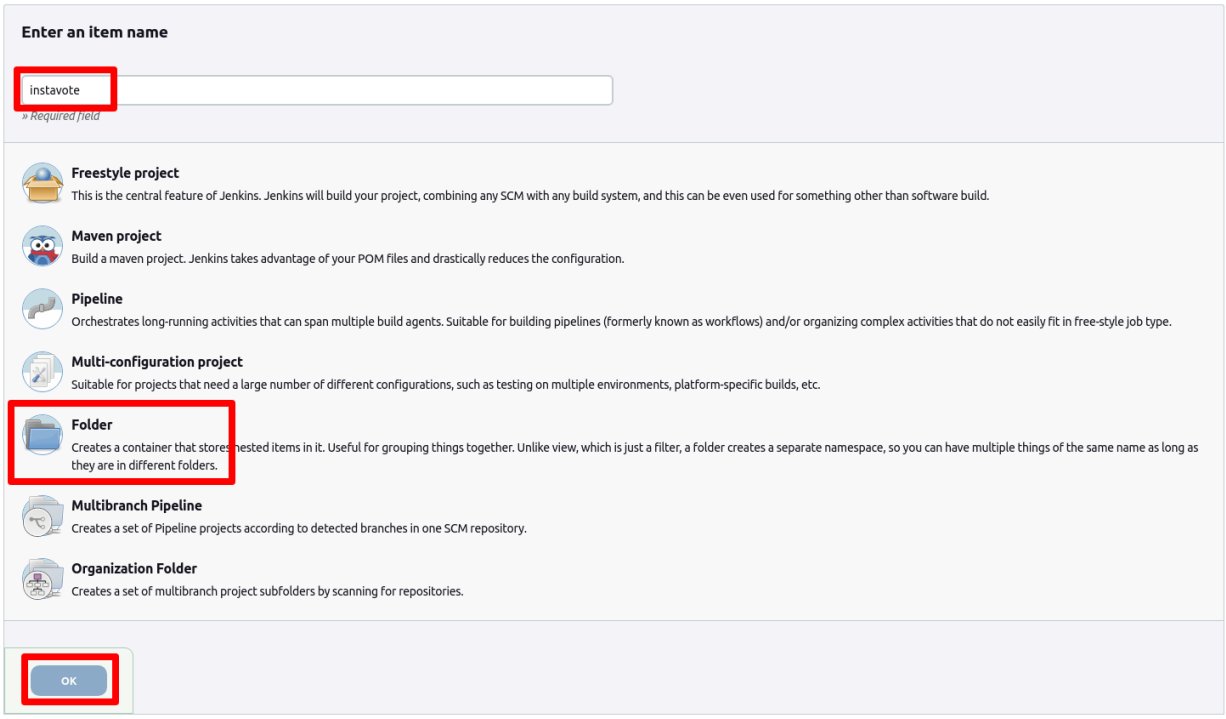
Click **Dashboard** in the top left of the page to go back to the home page.



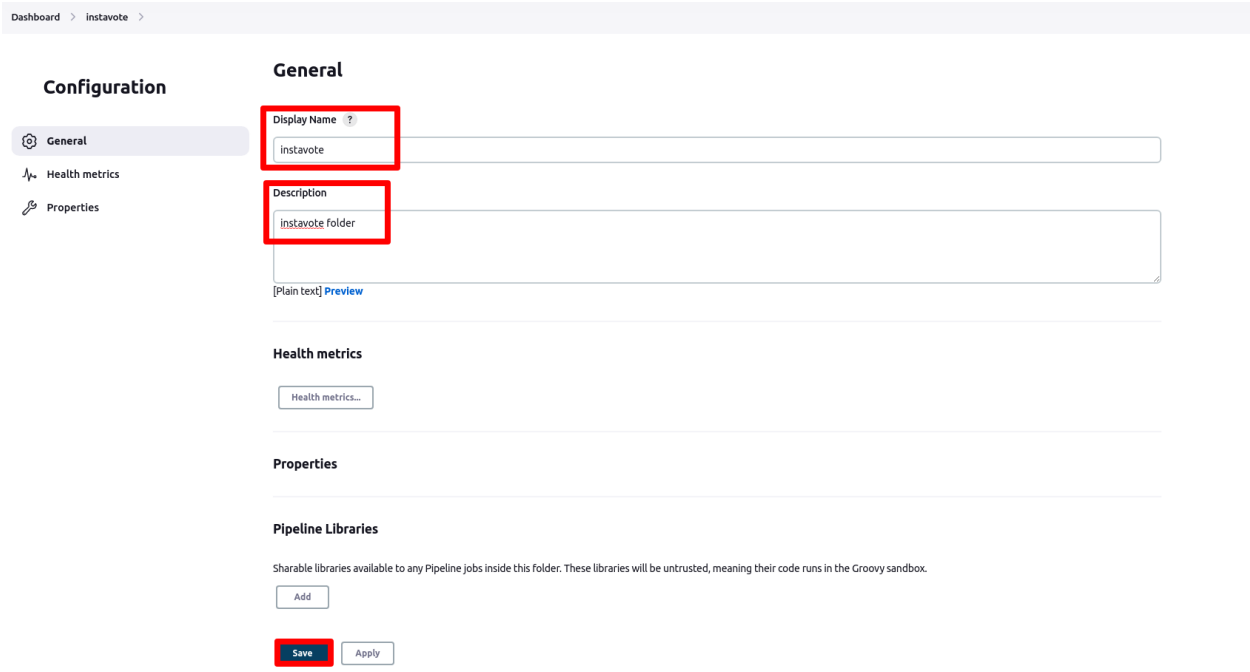
Create an `instavote` folder for your project. To do this, click **New item**:



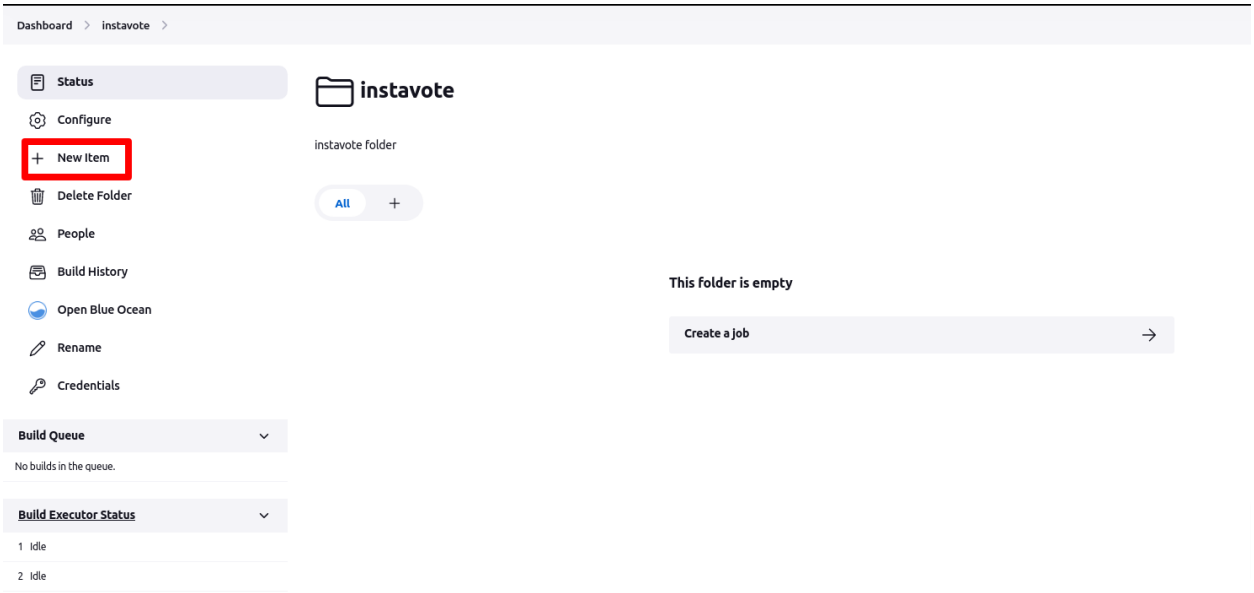
On the item creation page enter the name as “instavote” and select type **Folder**. Click **OK**:



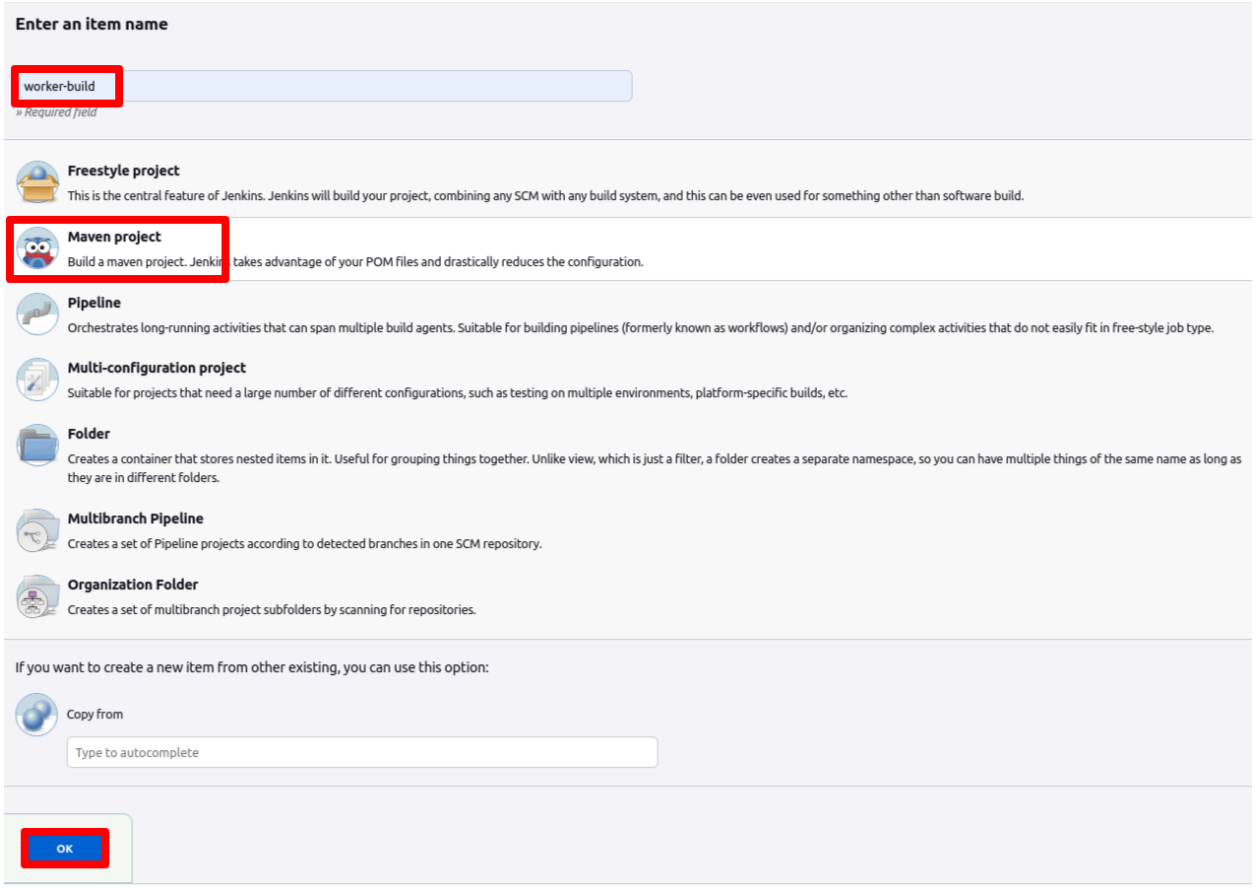
On the configuration page that appears fill it out as follows and click **Save**:



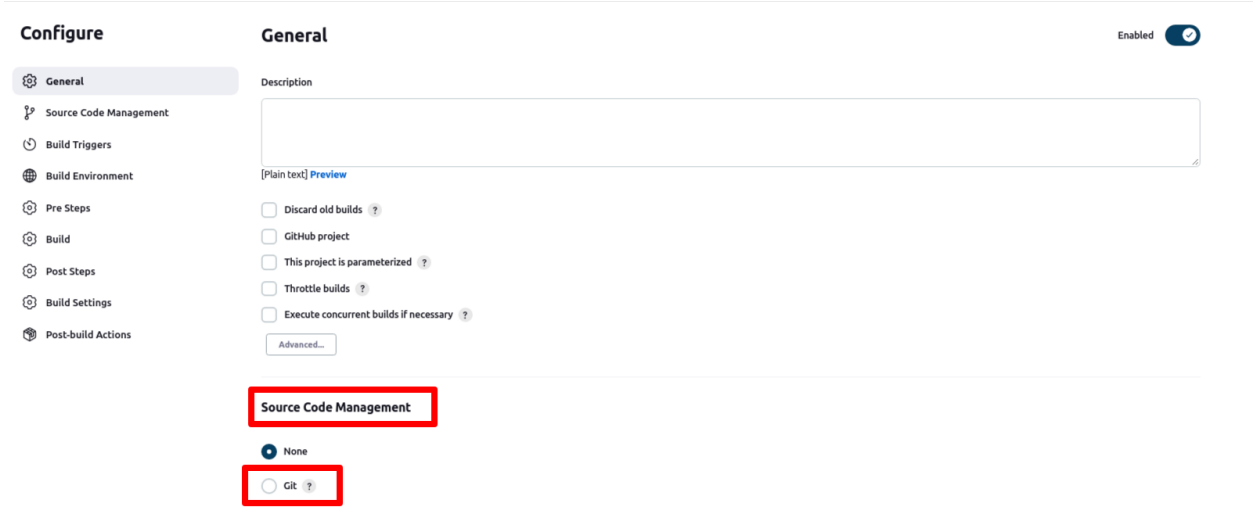
Inside the `instavote` folder, create a new item:



Select **Maven project** as the project type. Name the job `worker-build` and click **OK**:

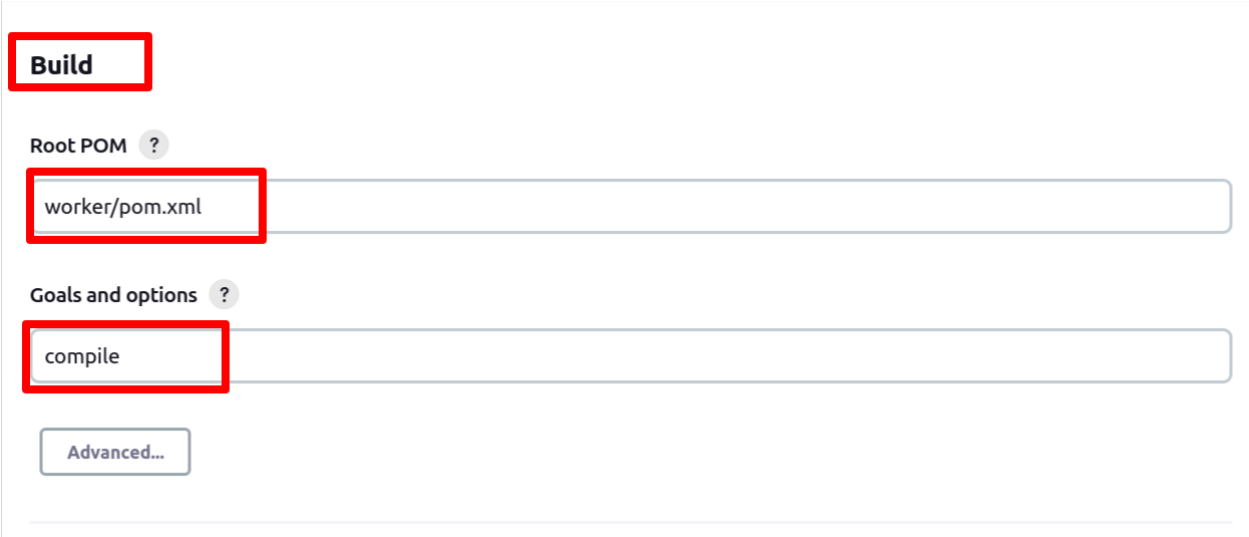


The **Configuration** page will appear. Scroll to the **Source Code Management** section and check the **Git** option.



Provide the URL to your `example-voting-app` repository that you forked earlier.

Go to the **Build** section. Provide the path to `pom.xml`, which is in the `worker` subdirectory of the repo, i.e. `worker/pom.xml`. Type or paste **compile** into the **Goals and options** box.



Save the job and build. Observe the job status, console output, etc.

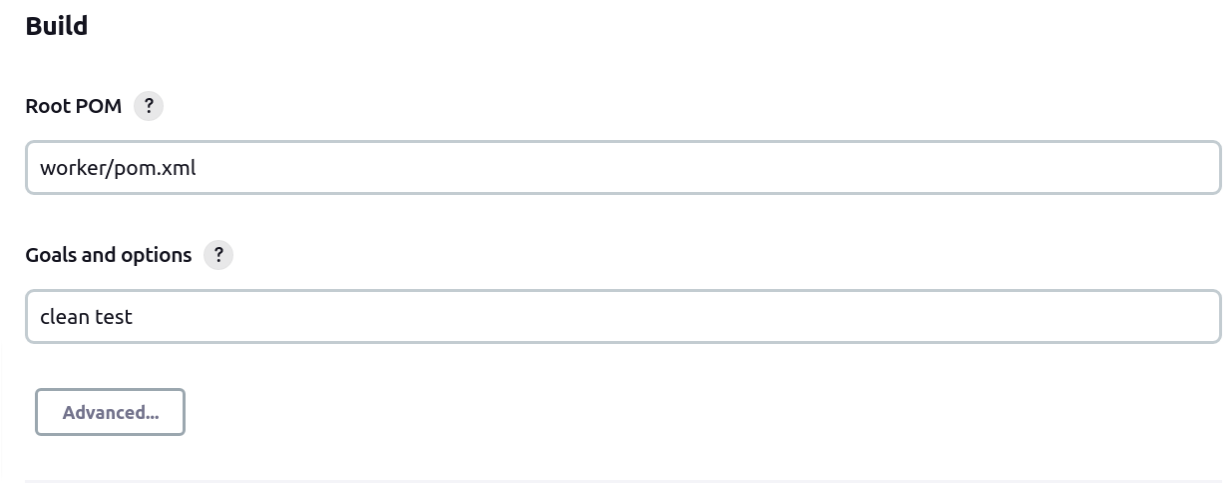
Adding Unit Test and Packaging Jobs

You will now add test and package jobs for the worker application.

You need to create one more job in the `instavote` folder and name it “worker-test”. You can copy the `worker-build` job. Follow these steps to complete the configuration.

In the `worker-test` job, change your description to “test worker java app”. Source code management repository is the same.

Under the **Build** section, change the **Goals and options** field to “clean test” and leave the rest. Save the job and build.

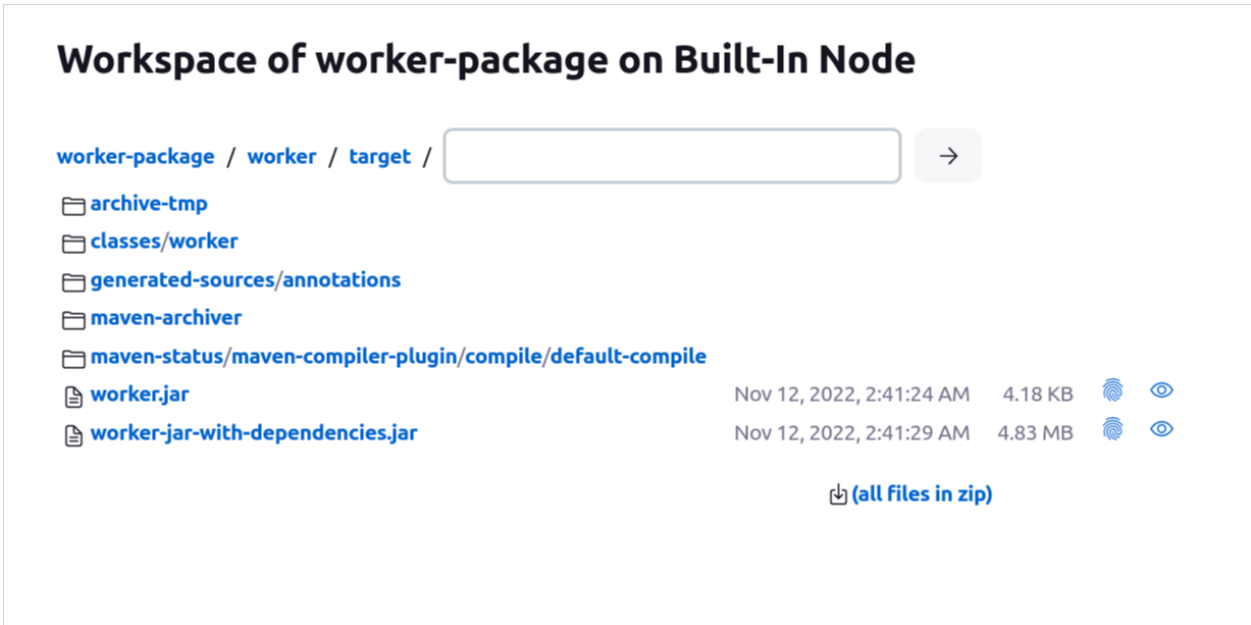


The next job will be **`worker-package`**. This will compile the application and then generate the .jar file. Create a job in the same folder with the name of "worker-package", copying the **`worker-test`** or the **`worker-build`** job.

Update the description to "package worker java app, create jar". The only change in the configuration is in the Build step.

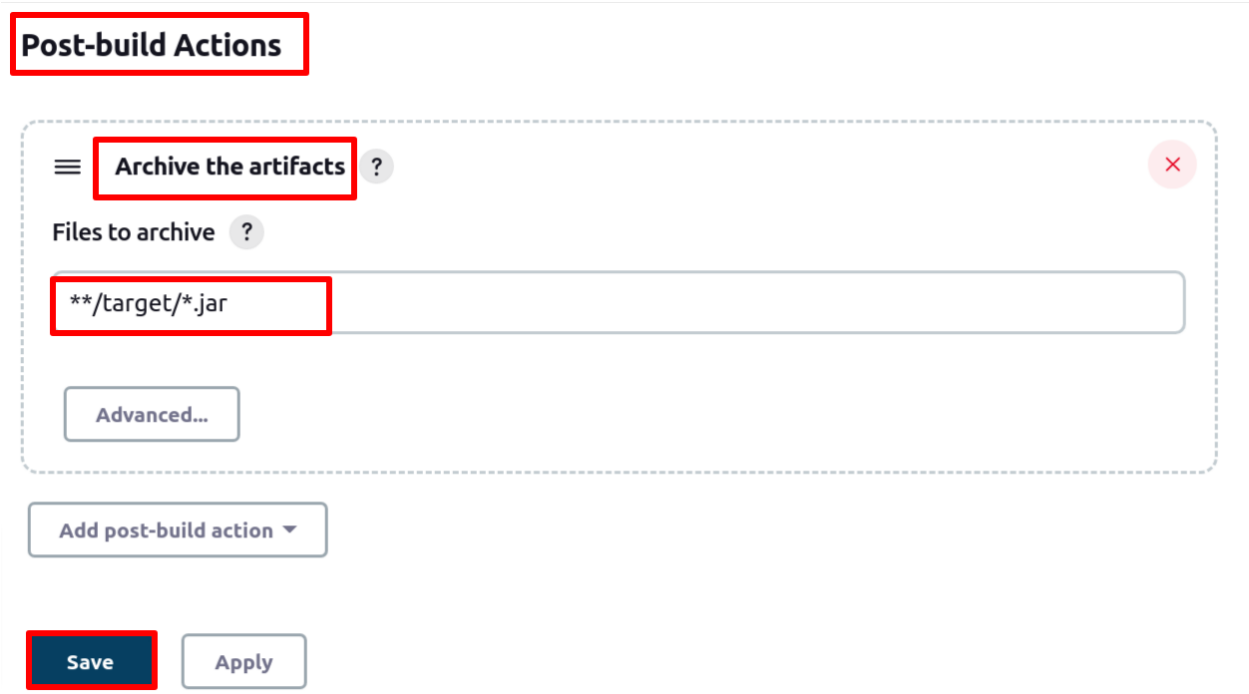
In the **Build** section for the package job, change the goal to "package -DskipTests". Save the changes and build.

After the build is successful, you can see the .jar file created in the workspace under the **`worker/target/`** directory. You can verify your workspace by comparing it to the picture below.

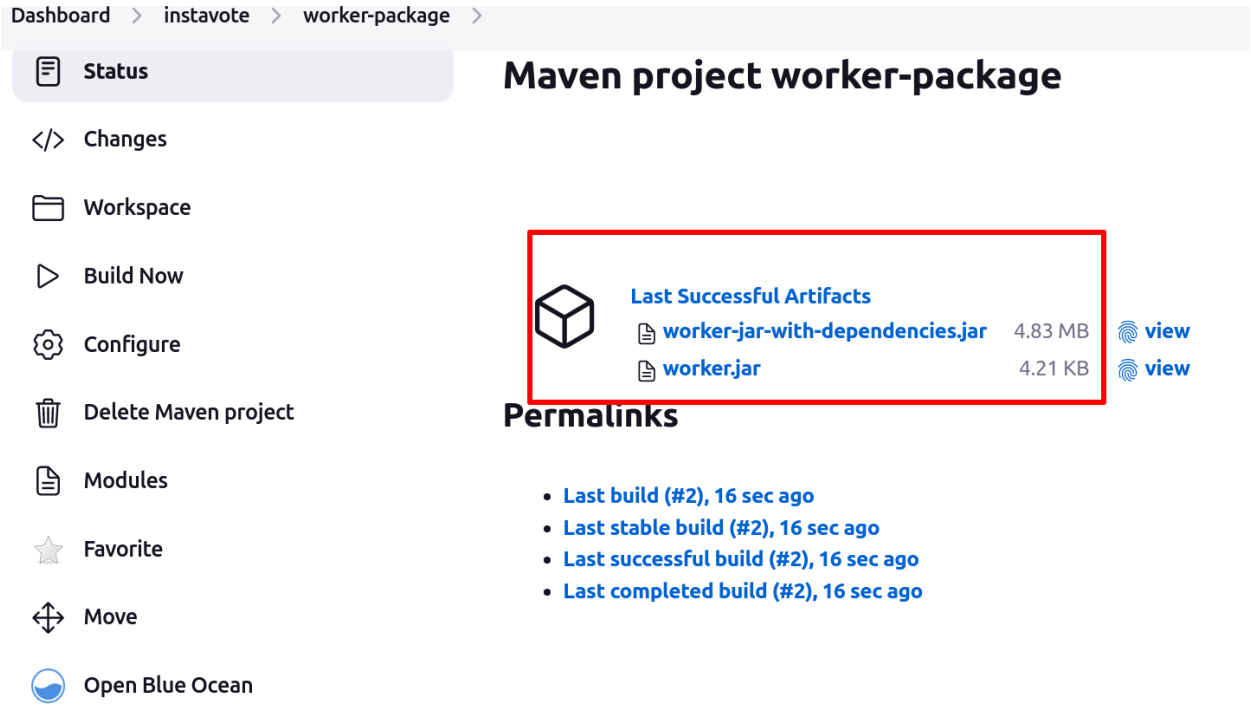


Click the **Configure** option in the left panel.

From the post-build actions, choose **Archive the artifacts** and provide the path `**/target/*.jar` to store the .jar file.



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



Configuring Build Triggers

You can use anything under **Build Triggers** to trigger automatic builds, but for now you are going to use **Poll SCM** under **Build Triggers**.

Go to the `worker-build` job configuration page and choose **Poll SCM** under **Build Triggers**. To periodically poll the Git repository, type or paste the following in the **Poll SCM** schedule:

```
H/2 * * * *
```

Once you have made changes, save the job and go to your job page where you will find the **Git polling log** in the left panel. Check your polling logs by clicking **Git Polling log**. It may take a minute to run.

Dashboard > instavote > worker-build > Git Polling Log

- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Maven project
- Modules
- Favorite
- Git Polling Log
- Move

Git Polling Log

```
Started on Nov 12, 2022, 3:40:00 PM
Using strategy: Default
[poll] Last Built Revision: Revision 4f256b896204526d787c804600380f192815f6cc
(refs/remotes/origin/master)
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
 > git --version # timeout=10
 > git --version # 'git version 2.30.2'
 > git ls-remote -h -- https://github.com/eeganlf/LFS261-example-voting-app.git #
timeout=10
Found 2 remote heads on https://github.com/eeganlf/LFS261-example-voting-app.git
[poll] Latest remote head revision on refs/heads/master is:
4f256b896204526d787c804600380f192815f6cc - already built by 2
Done. Took 0.52 sec
No changes
```

Now you will use **Trigger builds remotely** in **Build Triggers**. You can type any characters randomly into the token box and save it. Use the following example to trigger the build using the browser.

Build Triggers

- [x] Build whenever a SNAPSHOT dependency is built ?
  - [ ] Schedule build when some upstream has no successful builds ?
- [x] Trigger builds remotely (e.g., from scripts) ?

  Authentication Token

  adgsnfgnjsfhs

  Use the following URL to trigger build remotely: JENKINS_URL/job/instavote/job/worker-build/build?token=TOKEN_NAME or /buildWithParameters?token=TOKEN_NAME

  Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.
- [ ] Build after other projects are built ?
- [ ] Build periodically ?
- [ ] GitHub hook trigger for GITScm polling ?

A trigger URL is displayed just below where you defined the token. Your custom trigger URL will be similar to the following:

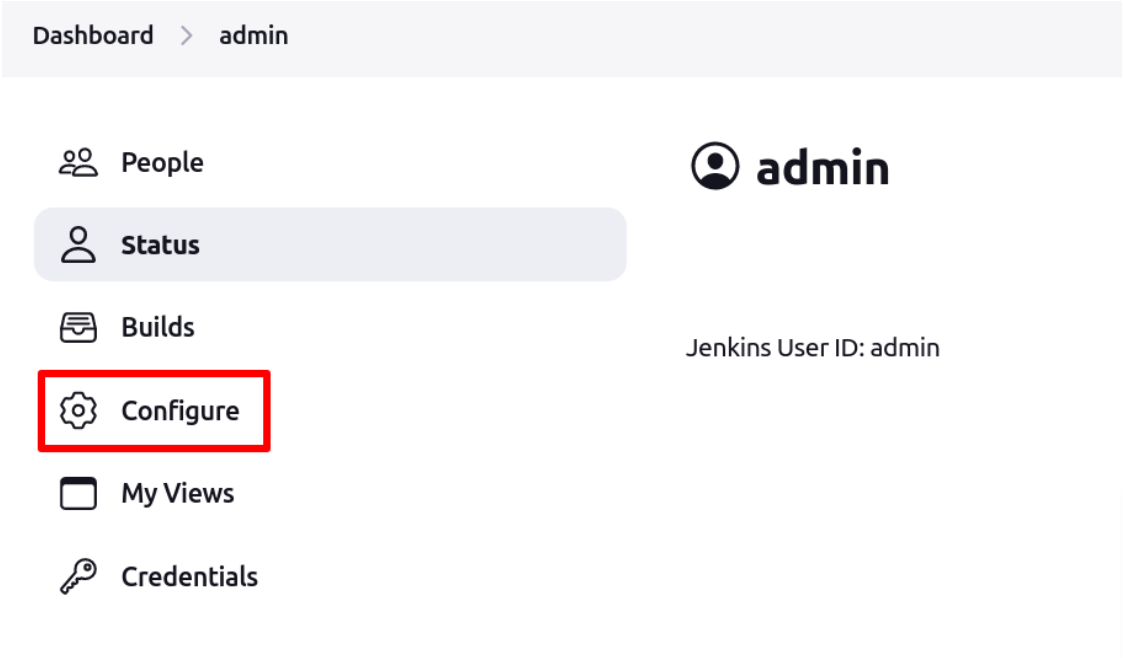
```
http://localhost:8080/job/instavote/job/worker-build/build?
token=yourauthenticationtoken
```

If you paste this URL in the browser and press **Enter**, you *may* see a page asking you to proceed, but you may see no indication that anything has occurred. You can verify that the URL ran the build by checking **Build History** at the bottom of the left pane in your Maven `worker-build` project page:

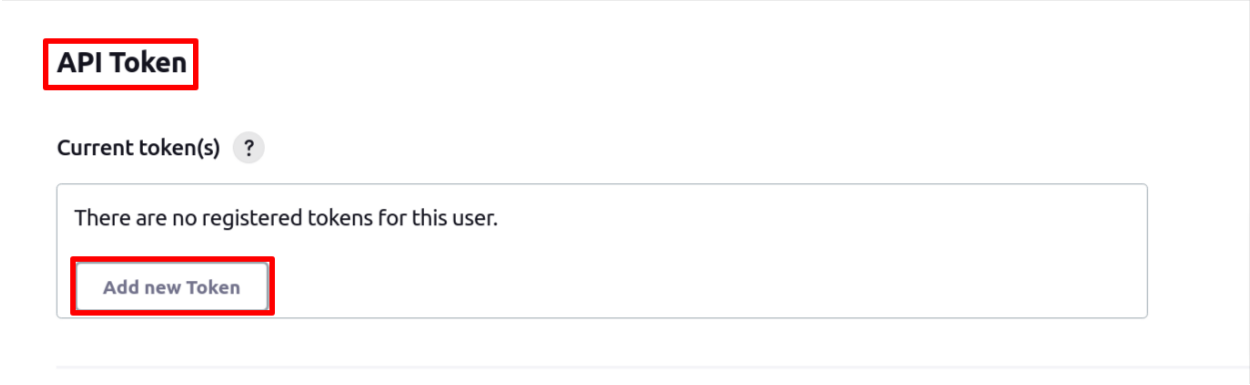
Click on **Proceed** if the UI appeared and, since you are already authenticated, you should see the job launched automatically. Verify that from the project page.

You can also trigger the build using a command line interface or programmatically from external code by providing the API token.

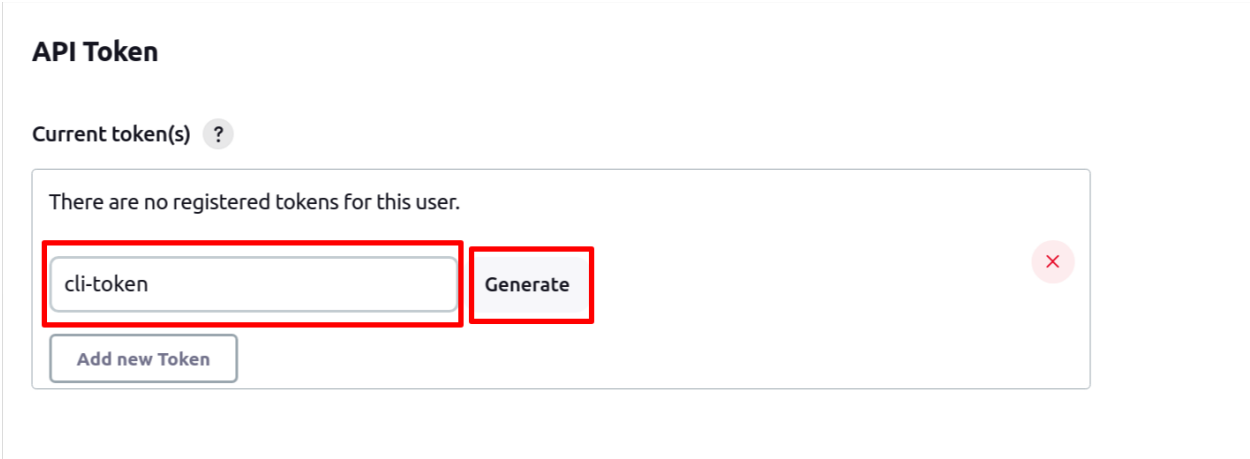
To create an API token, browse to **Jenkins -> People -> Admin -> Configure** as shown in the following image.

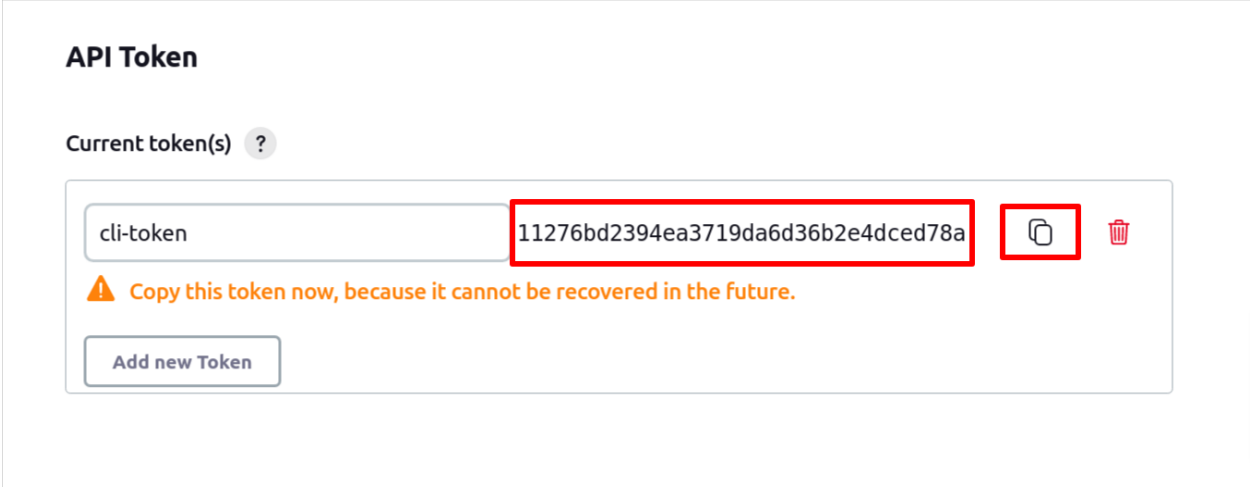


Scroll to the **API Token** section and click **Add new Token**:



Name the token “cli-token” and click **Generate**:





Copy the API token and put together a remote trigger URL similar to the following:

```
http://admin:TOKEN@localhost:8080/job/instavote/job/worker-build/build
?token=yourauthenticationtoken
```

Test trigger the build using a `curl` command or equivalent:

```
curl
http://admin:yourapitoken@localhost:8080/job/instavote/job/worker-buil
d/build?token=yourauthenticationtoken
```

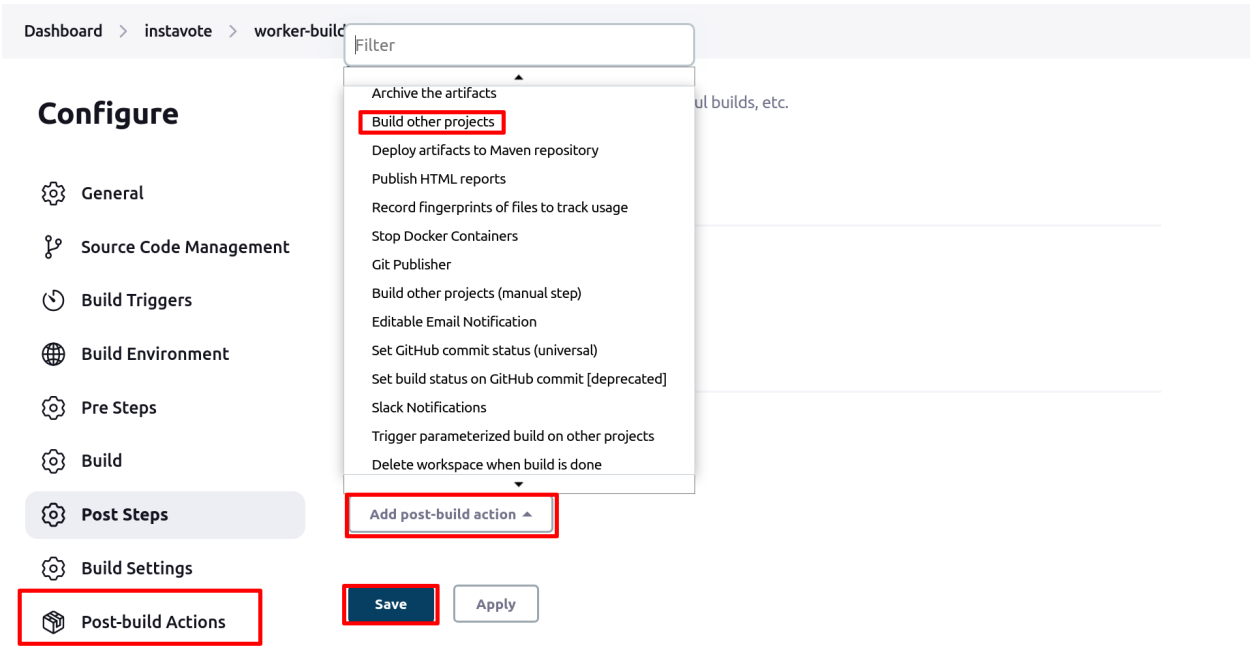
The above instructions demonstrate how to trigger builds remotely.

Creating a Job Pipeline

Now you will link jobs by defining upstreams and downstreams. You will also create a pipeline view using a plugin.

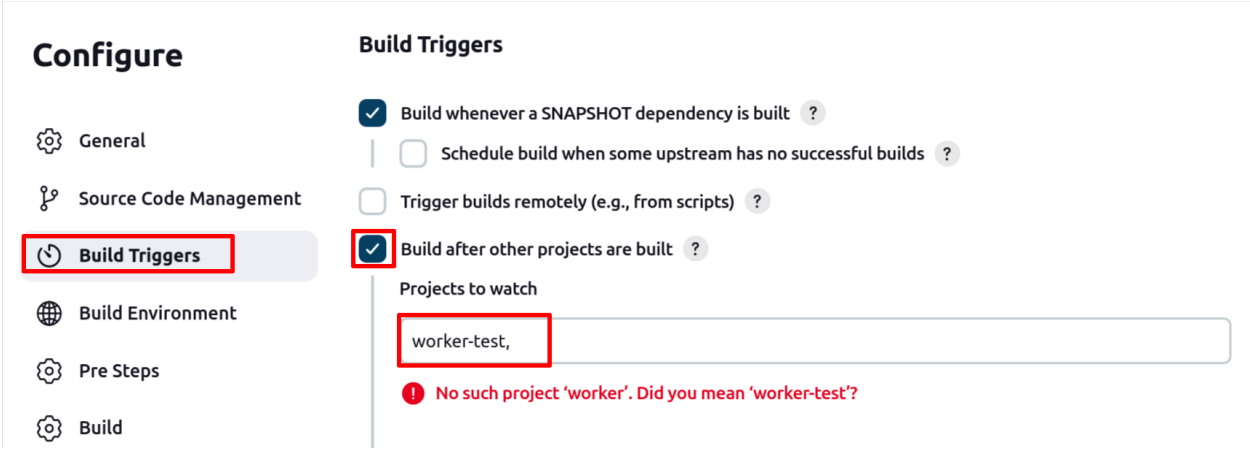
Follow these steps to set up upstream and downstream jobs.

From the `worker-build` job configuration page, scroll all the way to **Post-build Actions,** click **Add post-build action** and select **Build other projects**.



This is where you will define the downstream job. Provide `worker-test` as the project to build and save.

Now you are going to set up the upstream for `worker-package`. Go to the `worker-package` configuration page. From **Build Triggers**, check the box for **Build after other projects are built**. Put `worker-test` in the **Build after other projects are built** input box.



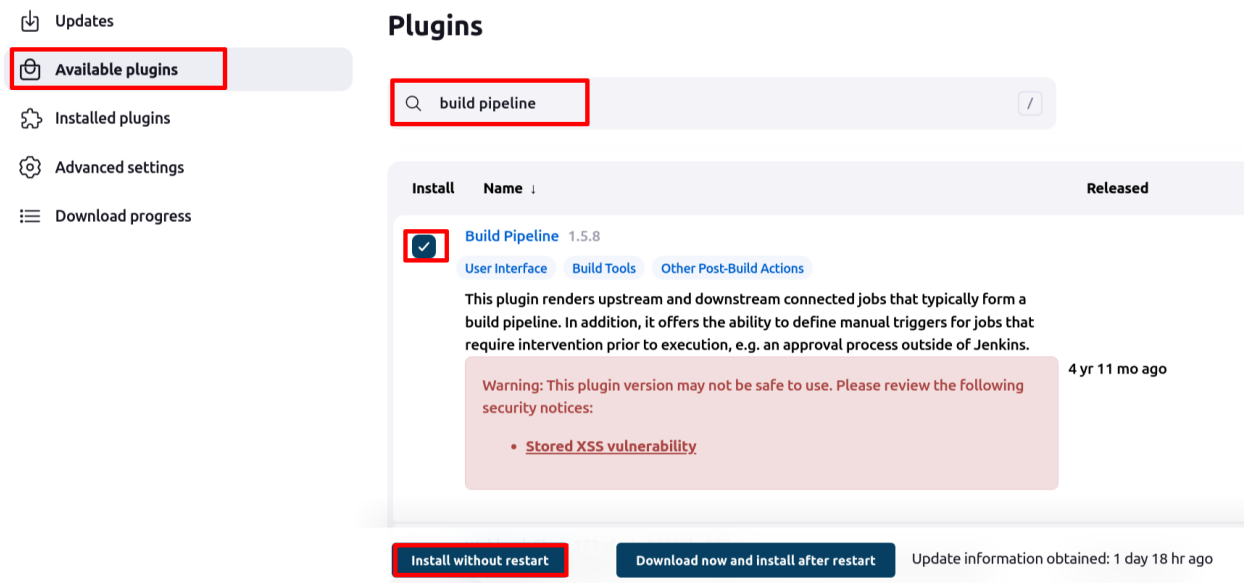
This defines the upstream for `worker-package`.

Once upstreams and downstreams are defined, run `worker-build` and it will automatically run `worker-test` and `worker-package`.

Set Up the Pipeline View

Now you are going to set up a pipeline view for this build job.

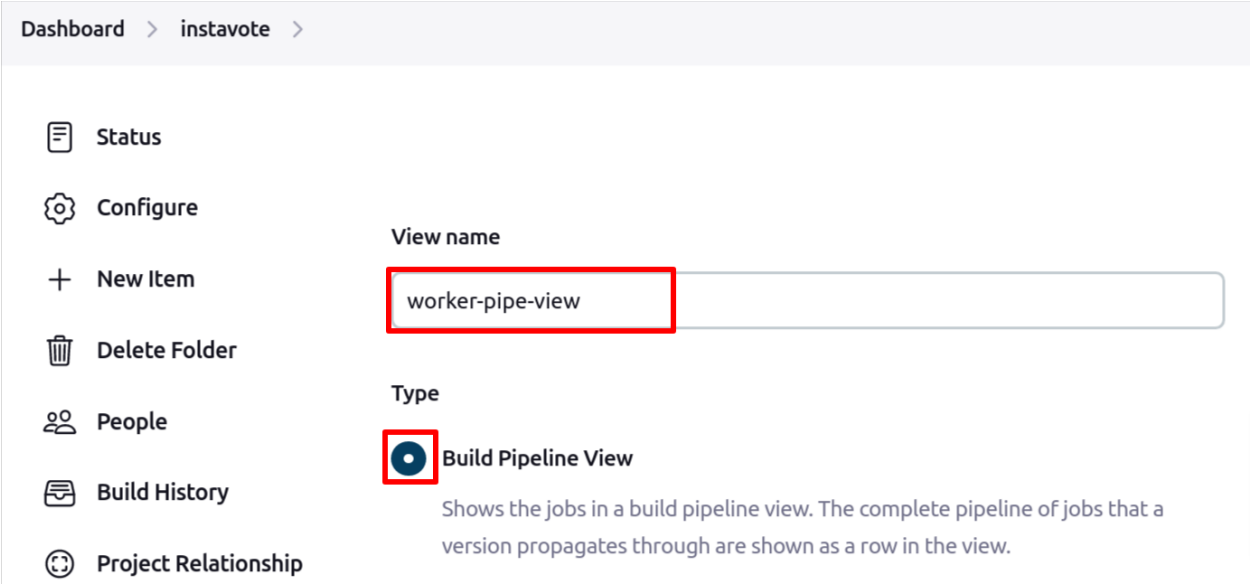
Begin by installing the Build Pipeline plugin from the **Manage Jenkins > Manage Plugins** page. Type *build pipeline* in the search box, select the **Build Pipeline** plugin and click **Install without restart**.



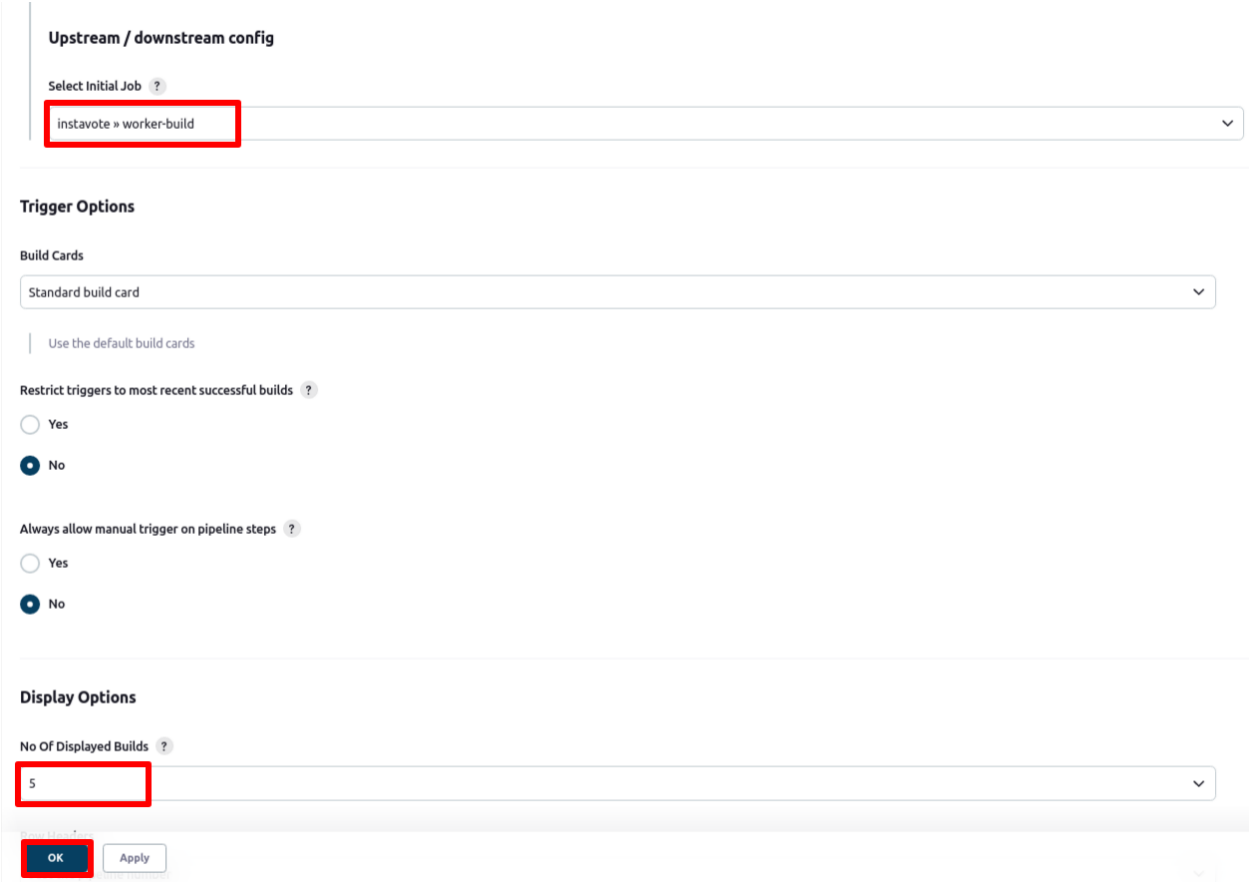
***NOTE**: You may see a vulnerability warning while installing the Build Pipeline plugin. It's not recommended that you install this plugin in a production environment. You are using this plugin only during this lab to aid your understanding of the process of creating pipelines. Starting with the next chapter, you will start using the Jenkinsfile, which does not need this plugin to provide you with a pipeline view. Jenkinsfiles are what you will use in a real production environment.*

Click on the `instavote` folder and you will now notice a **+** sign under the title. Click on it:

Select **Build Pipeline View** and provide a name for it, e.g. “worker-pipe-view”, then click **Create**:



From the configuration page, select the first job in the pipeline, **`worker-build`**, and select the number of displayed builds as 5, then save it.

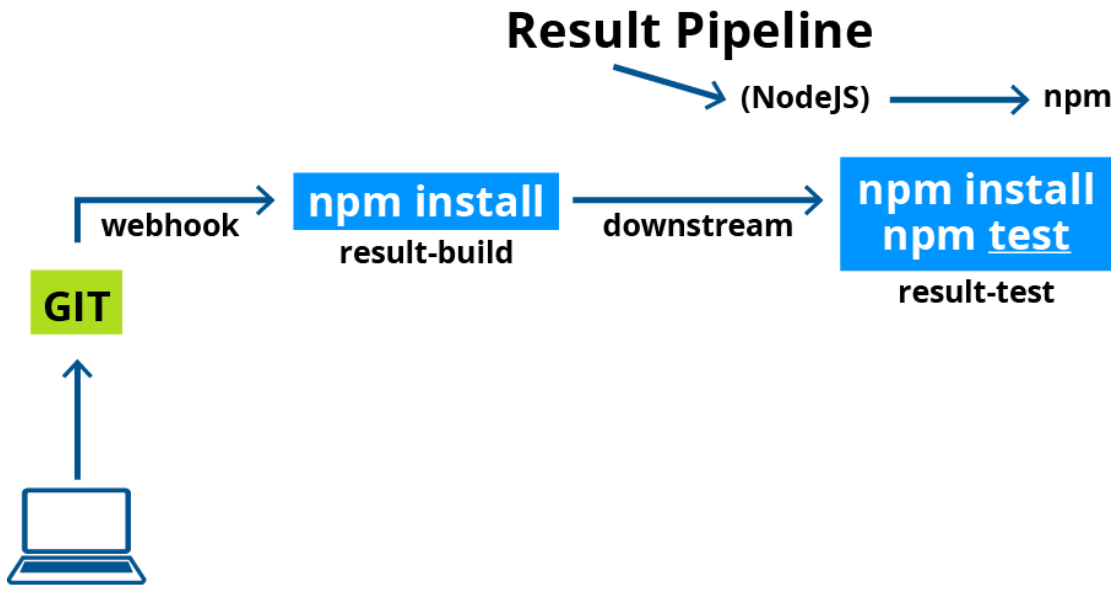


Once you complete, you will see the pipeline of your job. Green is successful, red is failed, blue is to do, yellow is in progress.

You have just created a pipeline for a Java project.

Exercise: Create a Pipeline for the Node.js Application

Now that you are familiar with creating CI pipelines, try creating one for the Node.js `result` application with `npm`.



Steps:

- Install **NodeJS** plugin for Jenkins.
- Configure **Global Tools** with a NodeJS installation with version 8.9.0.
- Create two jobs `result-build` and `result-test`, this time as freestyle projects.
- Define the Git repository you have forked in the **Source Code Management** section.
- Define the build trigger as **PollSCM** with an interval of 2 mins.
- In the build environment, choose to add NodeJS configurations with the version you have selected in **Global Tools**.
- Select the **Execute Shell** option from build and provide the commands to build the Node application from the `result` subdirectory. For example:

```
cd result

npm install

npm test
```

Summary

In this lab, we set up Jenkins using Docker Compose. We forked the `instavote` app so that we could work with our own version of the app. We then used Jenkins to set up an integration

pipeline for our `instavote` app on GitHub. Finally, we added a visual representation of what the build pipeline looks like.