



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.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password



Continue

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**.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

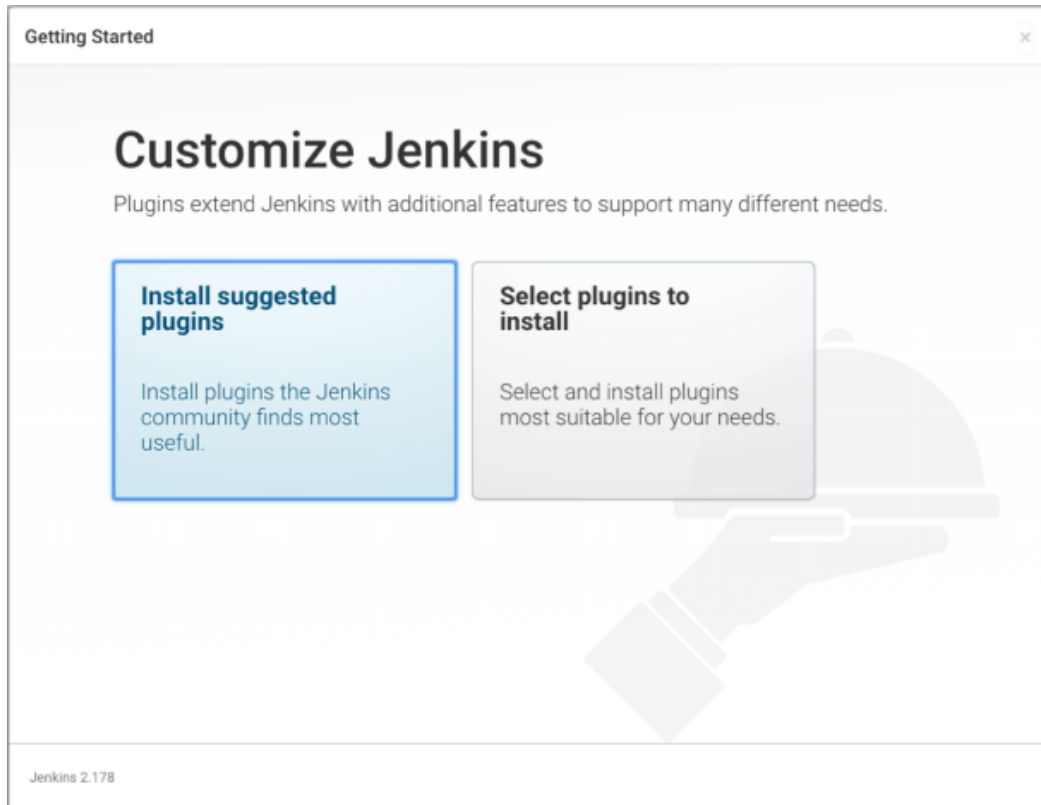
```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

[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:

Getting Started

Getting Started



<input checked="" type="checkbox"/> Folders Plugin	<input checked="" type="checkbox"/> OWASP Markup Formatter	<input checked="" type="checkbox"/> Build Timeout	<input checked="" type="checkbox"/> Credentials Binding Plugin	Folders
<input checked="" type="checkbox"/> Timestampers	<input checked="" type="checkbox"/> Workspace Cleanup	<input checked="" type="checkbox"/> Ant	<input checked="" type="checkbox"/> Gradle	OWASP Markup Formatter
<input checked="" type="checkbox"/> Pipeline	<input checked="" type="checkbox"/> GitHub Branch Source Plugin	<input checked="" type="checkbox"/> Pipeline: GitHub Groovy Libraries	<input checked="" type="checkbox"/> Pipeline: Stage View	Build Timeout
<input type="checkbox"/> Git plugin	<input type="checkbox"/> SSH Build Agents	<input type="checkbox"/> Matrix Authorization Strategy	<input type="checkbox"/> PAM Authentication	Credentials Binding
<input type="checkbox"/> LDAP	<input type="checkbox"/> Email Extension	<input type="checkbox"/> Mailer Plugin		Timestampers
				** Resource Disposer
				Workspace Cleanup
				Ant
				Gradle
				Pipeline
				GitHub Branch Source
				Pipeline: GitHub Groovy Libraries
				** Pipeline: REST API
				** JavaScript GUI Lib: Moment bundle

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.

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.375

[Skip and continue as admin](#)[Save and Continue](#)

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**.

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.375

[Not now](#)

[Save and Finish](#)

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

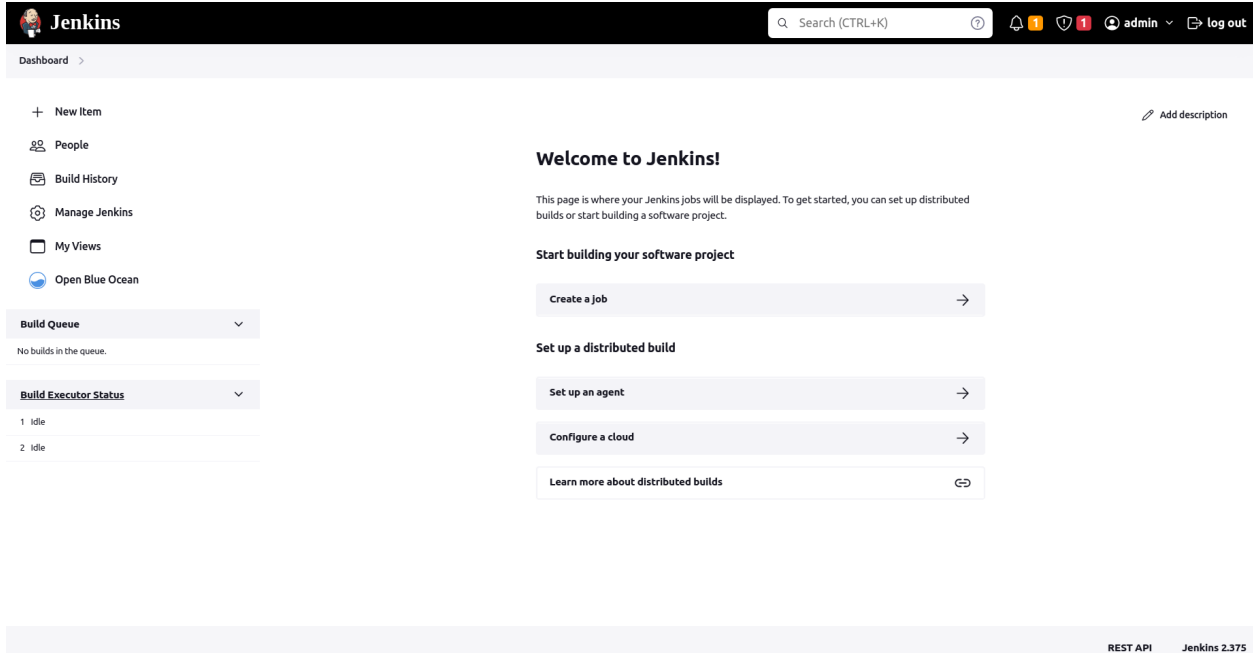
Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

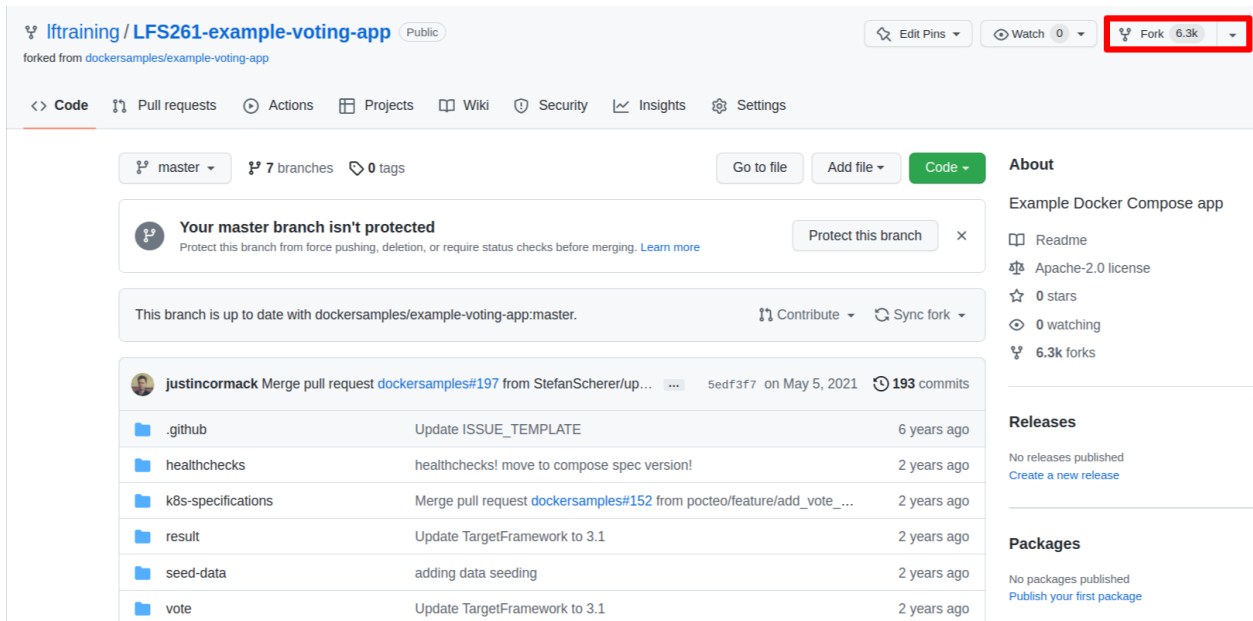
Jenkins 2.375



The screenshot shows the Jenkins Dashboard. At the top, there's a header with the Jenkins logo, a search bar, and user information (admin, log out). The main content area is divided into a left sidebar and a main panel. The sidebar contains links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Open Blue Ocean'. Below these are two expandable sections: 'Build Queue' (showing 'No builds in the queue') and 'Build Executor Status' (showing two idle executors). The main panel has a 'Welcome to Jenkins!' message, a brief explanation of the page's purpose, and a section titled 'Start building your software project' with a 'Create a job' button. Below that is a 'Set up a distributed build' section with buttons for 'Set up an agent', 'Configure a cloud', and a link to 'Learn more about distributed builds'.

Fork the Voting App

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



The screenshot shows the GitHub repository page for 'lftraining / LFS261-example-voting-app'. The repository is public and was forked from 'dockersamples/example-voting-app'. The top navigation bar includes links for 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main content area shows the repository's status: 'master' branch, 7 branches, 0 tags. A warning message states 'Your master branch isn't protected'. Below this, a message indicates the branch is up to date with the upstream repository. A list of recent commits is shown, including updates to the ISSUE_TEMPLATE, healthchecks, k8s-specifications, result, seed-data, and vote files. The right sidebar contains information about the repository: 'Example Docker Compose app', 'Readme', 'Apache-2.0 license', '0 stars', '0 watching', '6.3k forks', 'Releases' (no releases published), and 'Packages' (no packages published).

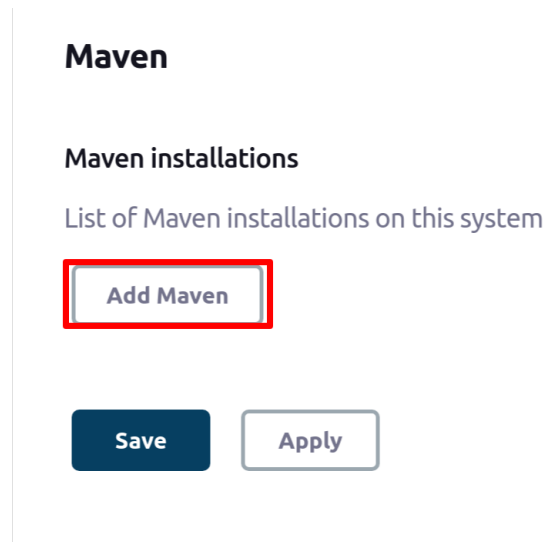
Clone the repository onto your own machine.

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:

Maven

Maven Installations

List of Maven installations on this system

Add Maven

Maven

Name

maven 3.6.1

☒ Install automatically ?

Install from Apache

Version

3.6.1

Add Installer

Add Maven

Save Apply

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.

Jenkins

Dashboard > Manage Jenkins > Plugin Manager

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

Plugins

Search: maven

Install	Name	Released
<input checked="" type="checkbox"/>	Maven Integration 3.20 Build Tools This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTS as well as the automated configuration of various Jenkins publishers such as Junit.	26 days ago
<input type="checkbox"/>	Config File Provider 3.11.1 Groovy-related External Site/Tool Integrations Maven 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.	3 mo 20 days ago

Click **Install without restart**.

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

Plugins

Search: maven

Install	Name ↓
<input checked="" type="checkbox"/>	Maven Integration 3.20 Build Tools This plugin provides a deep integration between Jenkins and Maven support for automatic triggers between projects depending on SN as well as the automated configuration of various Jenkins publishers
<input type="checkbox"/>	Config File Provider 3.11.1 Groovy-related External Site/Tool Integrations Maven Ability to provide configuration files (e.g. settings.xml for maven, custom files...) loaded through the UI which will be copied to the

Install without restart **Download now and install after restart**

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

Dashboard > Manage Jenkins > Plugin Manager

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

Download progress

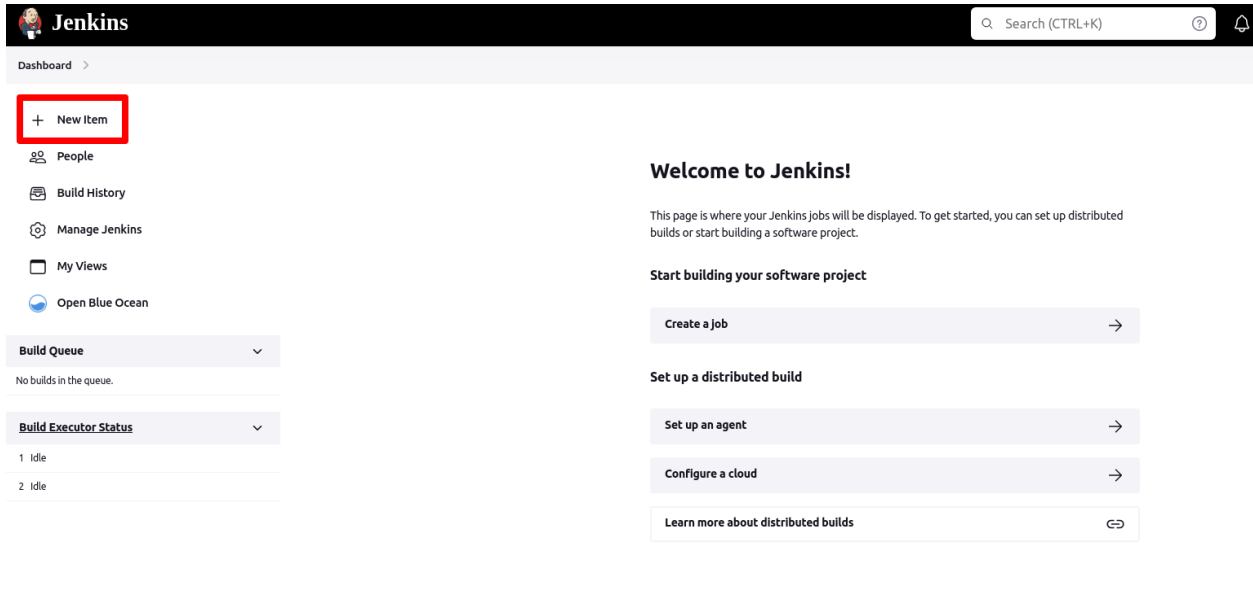
Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

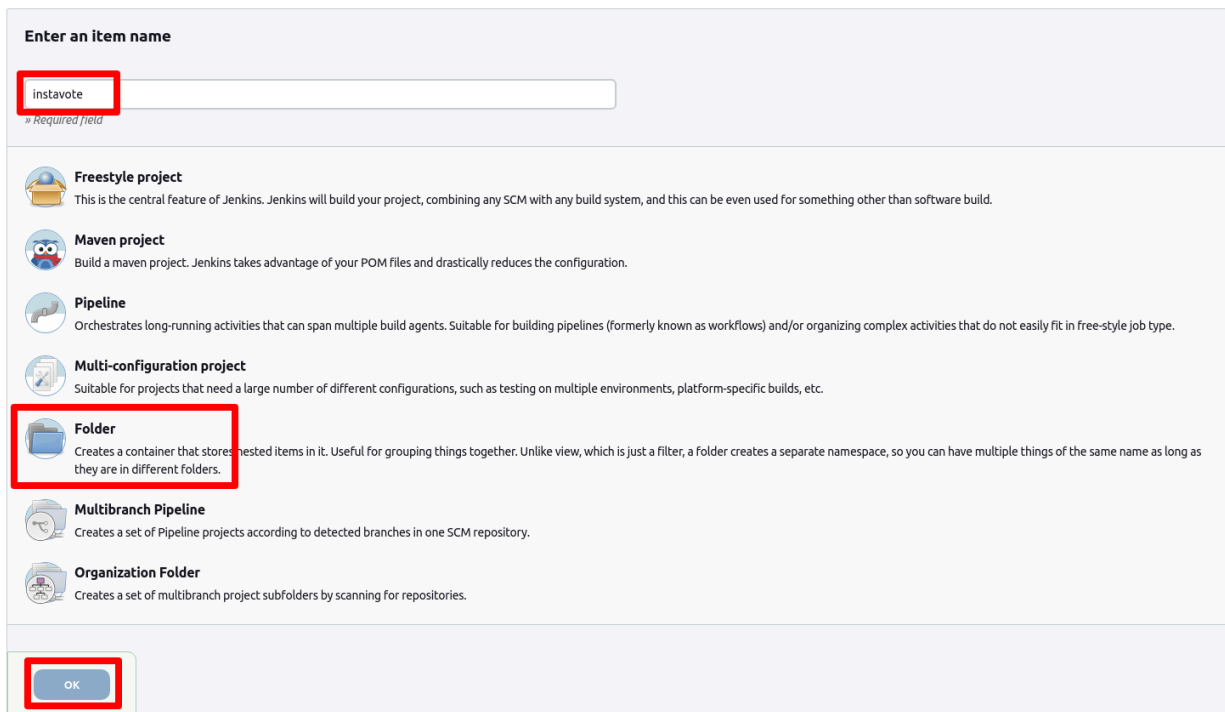
Folders

OWASP Markup Formatter	Success
Build Timeout	Success
Credentials Binding	Already Installed
Timestamp	Success
Resource Disposer	Success
Workspace Cleanup	Success

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**:

Dashboard > instavote >

Configuration

- General
- Health metrics
- Properties

General

Display Name 

instavote

Description

instavote folder

[Plain text] [Preview](#)

Health metrics

Health metrics...

Properties

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

[Add](#)

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

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

Dashboard > instavote >

Status

- Configure
- New Item**
- Delete Folder
- People
- Build History
- Open Blue Ocean
- Rename
- Credentials

instavote

instavote folder

[All](#) [+](#)

This folder is empty

[Create a job](#) →

Build Queue

No builds in the queue.

Build Executor Status


- 1 Idle
- 2 Idle


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


Enter an item name


worker-build


» 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.


 **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:

 Copy from

Type to autocomplete

OK

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

Configure

General Enabled

General

Source Code Management

Build Triggers

Build Environment

Pre Steps

Build

Post Steps

Build Settings

Post-build Actions

Description

[Plain text] [Preview](#)

☐ Discard old builds ?

☐ GitHub project

☐ This project is parameterized ?

☐ Throttle builds ?

☐ Execute concurrent builds if necessary ?

[Advanced...](#)

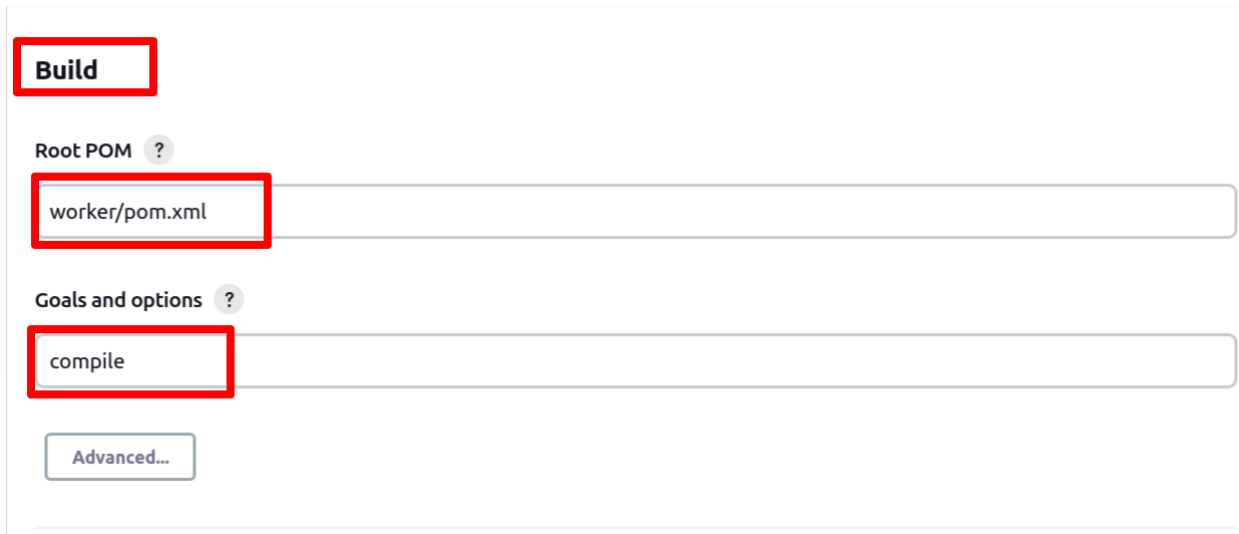
Source Code Management

☒ None

☐ Git ?

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.



Build

Root POM ?

worker/pom.xml

Goals and options ?

compile

Advanced...

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.

Build

Root POM ?

Goals and options ?

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.

Workspace of worker-package on Built-In Node

worker-package / worker / target /

- archive-tmp
- classes/worker
- generated-sources/annotations
- maven-archiver
- maven-status/maven-compiler-plugin/compile/default-compile
- worker.jar Nov 12, 2022, 2:41:24 AM 4.18 KB  
- worker-jar-with-dependencies.jar Nov 12, 2022, 2:41:29 AM 4.83 MB  

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

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.

Post-build Actions

≡ **Archive the artifacts** ?

Files to archive ?

****/target/*.jar**

Advanced...

Add post-build action ▼

Save Apply

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

Dashboard > instavote > worker-package >

Status

Changes

Workspace

Build Now

Configure

Delete Maven project


Modules

Favorite

Move

Open Blue Ocean

Maven project worker-package



Last Successful Artifacts

worker-jar-with-dependencies.jar 4.83 MB [view](#)

worker.jar 4.21 KB [view](#)

Permalinks

- [Last build \(#2\), 16 sec ago](#)
- [Last stable build \(#2\), 16 sec ago](#)
- [Last successful build \(#2\), 16 sec ago](#)
- [Last completed build \(#2\), 16 sec ago](#)

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.

Copyright The Linux Foundation, 2019-2022. All rights reserved.

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

☒ Build whenever a SNAPSHOT dependency is built ?

☐ Schedule build when some upstream has no successful builds ?

☒ **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:

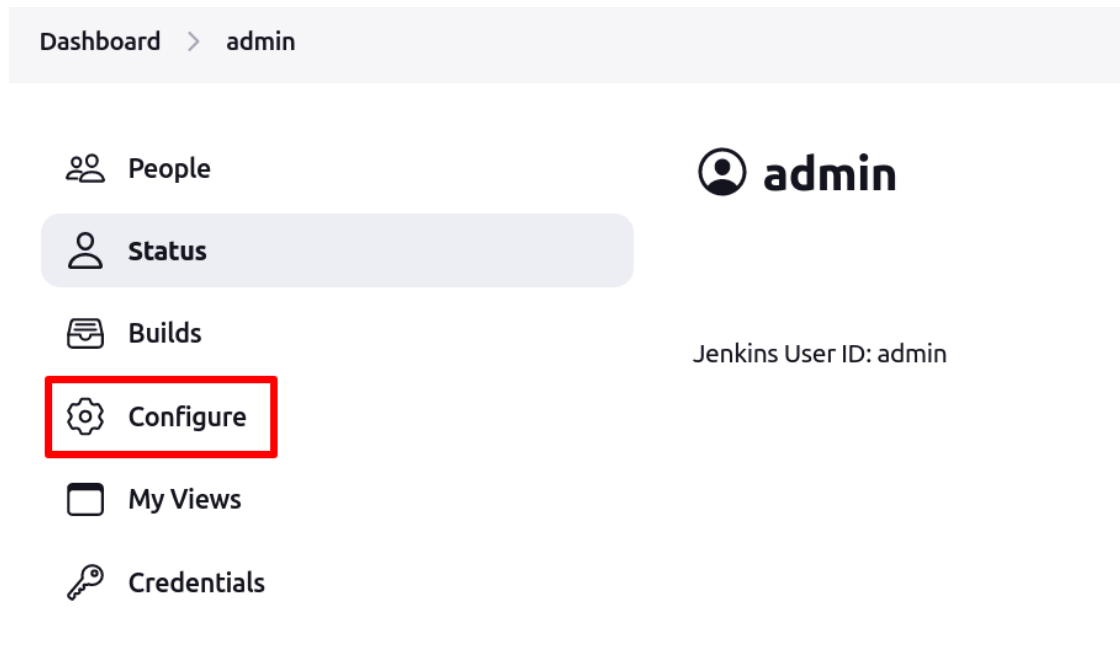
[http://localhost:8080/job/instavote/job/worker-build/build?
token=yourauthenticationtoken](http://localhost:8080/job/instavote/job/worker-build/build?token=yourauthenticationtoken)

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**:

API Token

Current token(s) ?

There are no registered tokens for this user.

Add new Token

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

API Token

Current token(s) ?

There are no registered tokens for this user.

cli-token

Generate**Add new Token****API Token**

Current token(s) ?

cli-token

11276bd2394ea3719da6d36b2e4dced78a

⚠ Copy this token now, because it cannot be recovered in the future.

Add new Token

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-build/build?token=yourauthenticationtoken
```

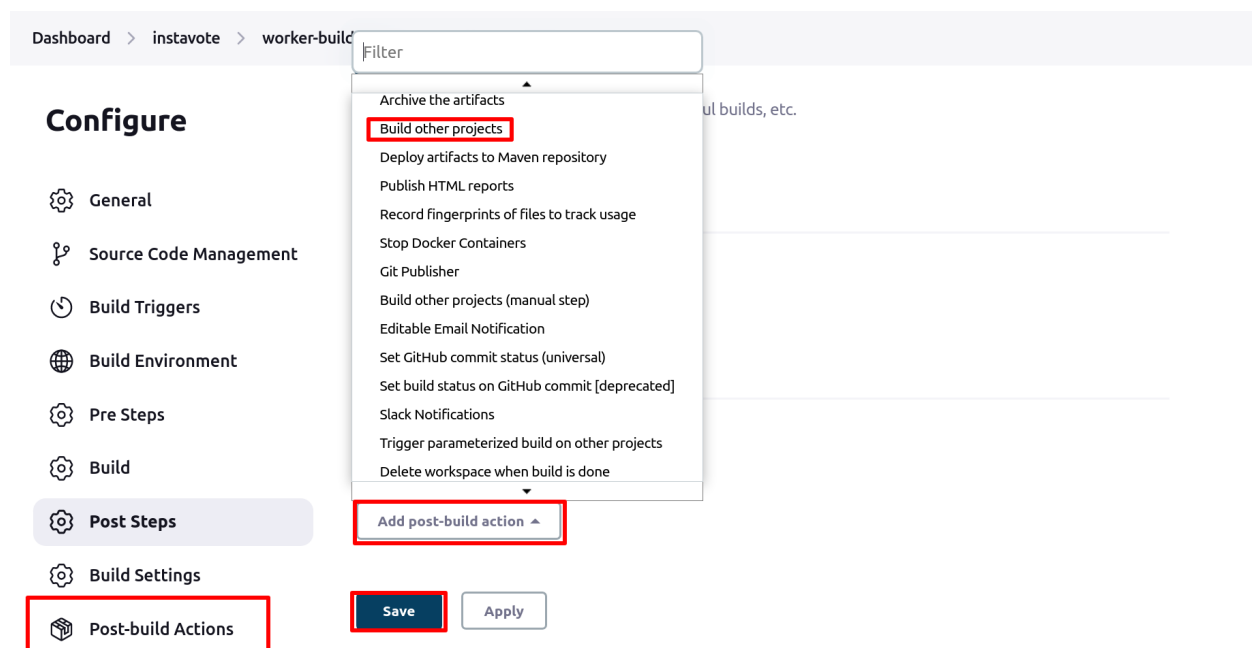
The above instructions demonstrate how to trigger builds remotely.

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.

Post-build Actions

≡ Build other projects ?

×

Projects to build

worker-test,

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

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.

Configure

⚙️ General

🔑 Source Code Management

🕒 Build Triggers

🌐 Build Environment

⚙️ Pre Steps

⚙️ Build

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

worker-test,

❗

 No such project 'worker'. Did you mean 'worker-test'?

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**.

The screenshot shows the Jenkins 'Manage Plugins' interface. On the left sidebar, 'Available plugins' is selected. The main area is titled 'Plugins' and contains a search bar with 'build pipeline' entered. Below the search bar, the 'Build Pipeline' plugin (version 1.5.8) is listed. It has a checkmark in the 'Install' column. 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.' A warning box indicates: 'Warning: This plugin version may not be safe to use. Please review the following security notices: • Stored XSS vulnerability'. At the bottom, the 'Install without restart' button is highlighted with a red box. Other buttons include 'Download now and install after restart' and 'Update information obtained: 1 day 18 hr ago'.



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:

instavote


instavote folder


All 


S	W	Name ↓	Last Success	Last Failure
		worker-build	21 hr #1	N/A


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


Dashboard > instavote >


 Status


 Configure

 New Item

 Delete Folder


 People

 Build History

 Project Relationship

View name

Type

 **Build Pipeline View**

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

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.

Upstream / downstream config

Select Initial Job ?
instavote » worker-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 ?
5

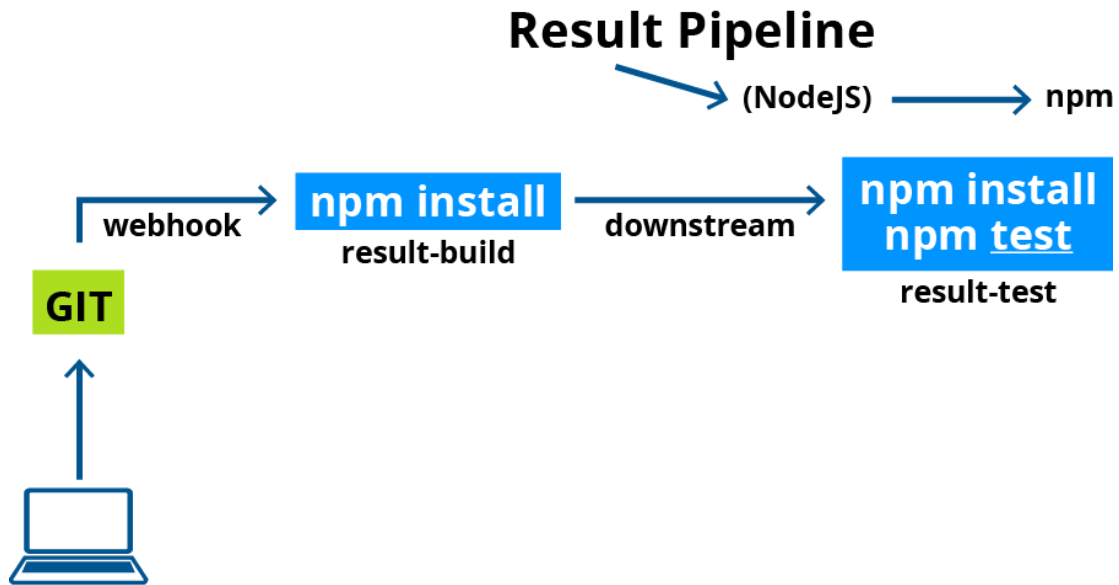
OK Apply

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.