

## Tasks

0/11

### 1. Add Maven to Jenkins

Log in to Jenkins and make sure it has the Maven installation.

1. Open Jenkins by clicking the link on top right (the red rectangle below)

```
README.md ×

1 # Welcome to Udemy Jenkins Labs!
2
3 ## Access
4
5 To open Jenkins, click on the "Jenkins" link at the bottom of the page at the the status bar.
6
7 When you load Jenkins for the first time, to gain access you are going to need the initial admin password. The username and
8 password has been generated for you.
9
10 ...
11
12
13 Please check the DevOps Workspace Guide for additional documentation.
14
15
16 ### Java / Spring or Tomcat Access (optional)
```

2. Login to Jenkins using the provided username / password (green rectangle above)



# Welcome to Jenkins!

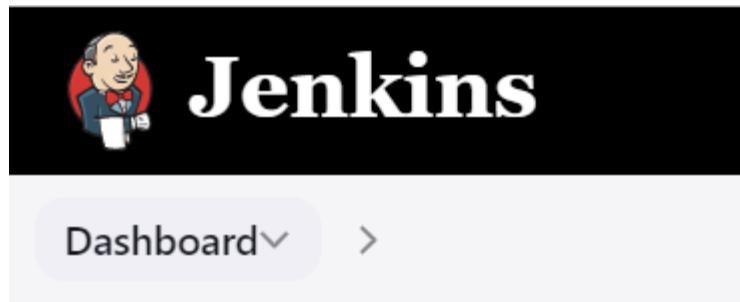
Username:

Password:

Keep me signed in

**Sign in**

3. You will build a Java project using Maven, so make sure the Jenkins has Maven installed. From the Jenkins home screen, go to Menu **Manage Jenkins > Global Tool Configuration** (under the **System Configuration** section)



+ New Item

People

Build History

Manage Jenkins

1

My Views

Lockable Resources

#### System Configuration



Configure System

Configure global settings and paths.

2



Global Tool Configuration

Configure tools, their locations and automatic installers.



Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



Manage Nodes and Clouds

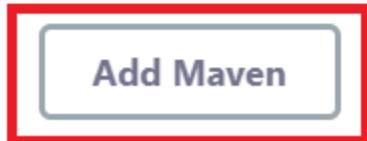
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

4. Scroll down and click **Add Maven** button

# Maven

## Maven installations

List of Maven installations on this system



### 5. Installation details:

- Name: my-maven
- Select the checkbox *Install automatically*
- Install the latest version from Apache
- Save the configuration

A screenshot of a configuration dialog for adding a Maven installation. The dialog has a dashed border and contains the following fields:

- Maven Name:** A text input field containing "my-maven" (labeled 'a').
- Install automatically:** A checkbox that is checked (labeled 'b').
- Install from Apache:** A section with a dropdown menu set to "Version 3.8.6" (labeled 'c').
- Add Installer:** A button to add more installers.

At the bottom of the dialog are two buttons: "Add Maven" and "Save" (labeled 'd'), with "Save" being the primary action button.

## Resources

### Documentation

- Build Java Maven Application Using Jenkins

Mark task as complete

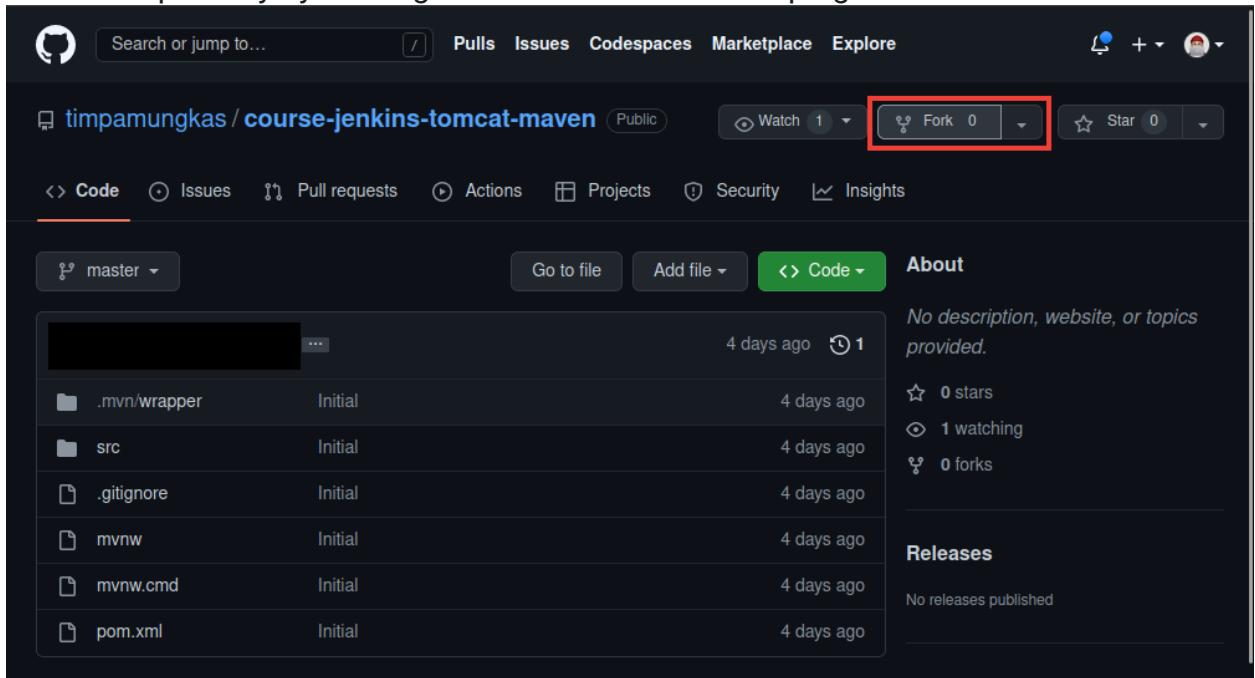
Report issue

### 2. Configure Credentials for the Tomcat Deployment

### 3. Initialize the Java Source Code

Fork [this Github repository](#) to your own [Github](#) account.

1. Create a [Github account](#) using your email if you do not have one. Log in if you already have an account.
2. Go to [this repository](#).
3. Fork the repository by clicking the **Fork** button on the top right corner.



4. Enter the details for your fork, and then click **Create fork**

## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Owner \*



Repository name \*



course-jenkins-tomcat-maven



By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

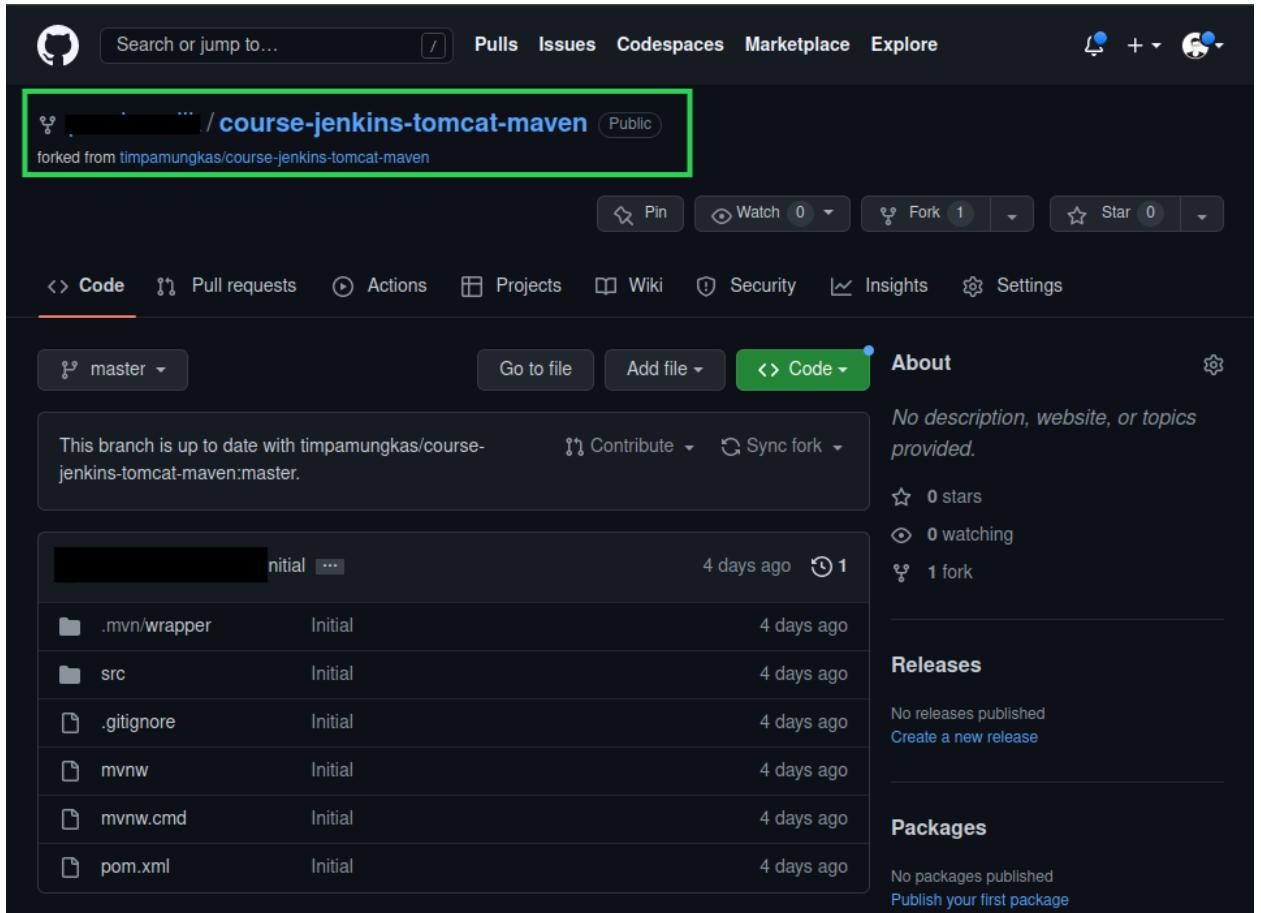
Copy the master branch only

Contribute back to [timparamungkas/course-jenkins-tomcat-maven](#) by adding your own branch. [Learn more](#).

(i) You are creating a fork in your personal account.

Create fork

5. You should be redirected to your own repository, meaning you have successfully forked the repository. This repository is a Java project configured for the Maven build tool (you can tell from the file structure that contains mvn... files and a pom.xml).



## Resources

### Documentation

- Fork Github Repository
- Apache Maven

Mark task as complete

Report issue

### 4. Configure the Jenkins Task to Build & Deploy the Java Project

Install the Jenkins plugin with the name: ***Deploy To Container***

Create a Jenkins project to build the Java application from the forked repository, and deploy the built app into Tomcat 9.

1. From the Jenkins home screen, go to ***Manage Jenkins***:



### Open *Manage Plugins*

The image shows the 'System Configuration' page. It includes sections for 'Configure System' (Configure global settings and paths), 'Global Tool Configuration' (Configure tools, their locations and automatic installers), 'Manage Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins, which is highlighted with a red rectangular border), and 'Manage Nodes and Clouds' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on).

System Configuration

Configure System  
Configure global settings and paths.

Global Tool Configuration  
Configure tools, their locations and automatic installers.

Manage Plugins  
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds  
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Search for the **Deploy to Container** plugin under the **Available** tab. This plugin allows you to deploy an application into the defined container (server). Select **Install it without restart**:

The image shows the 'Plugin Manager' page. It has tabs for 'Updates', 'Available' (which is selected and highlighted with a red rectangular border), 'Installed', and 'Advanced'. A search bar contains the text 'Q. Deploy to container'. Below the search bar is a table listing the plugin. The table columns are 'Name', 'Version', 'Last Published', and 'Released'. The first row in the table is for the 'Deploy to container' plugin, version 1.16, last published 2 yr 2 mo ago, and released 2 yr 2 mo ago. It has a brief description: 'This plugin allows you to deploy a war to a container after a successful build. Glassfish 3.x remote deployment'. Below the table are two buttons: 'Install without restart' (which is highlighted with a red rectangular border) and 'Download now and install after restart'. There is also a note: 'Update information obtained: 38 min ago' and a 'Check now' button.

Plugin Manager

Updates Available Installed Advanced

Q. Deploy to container

Name	Version	Last Published	Released
Deploy to container	1.16	2 yr 2 mo ago	2 yr 2 mo ago

Install Name : Deploy to container 1.16  
Artifact Uploader  
This plugin allows you to deploy a war to a container after a successful build.  
Glassfish 3.x remote deployment

Released

Install without restart Download now and install after restart Update information obtained: 38 min ago Check now

2. When installation is done, restart Jenkins by checking the checkbox shown below.

# Installing Plugins/Upgrades

## Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Loading plugin extensions  Success

Deploy to container  Success

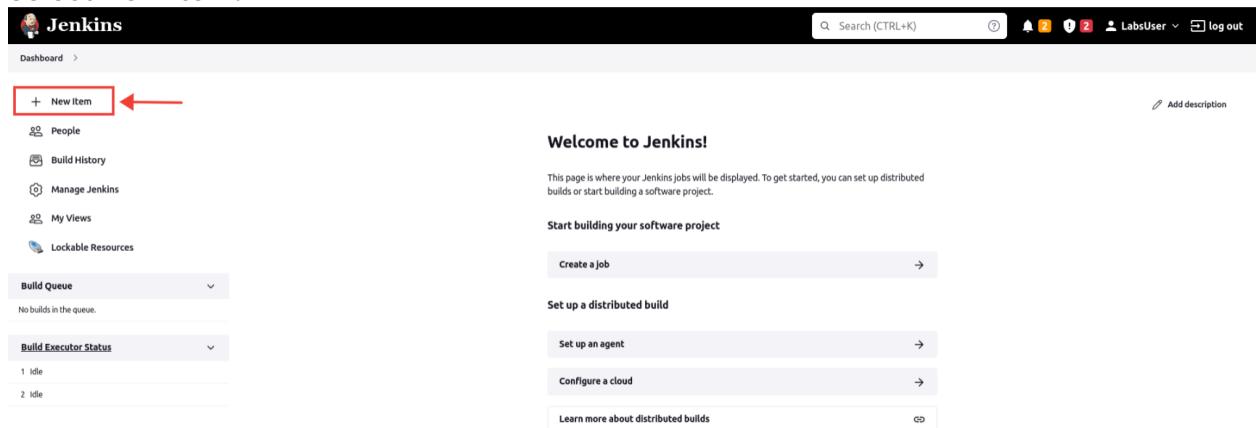
Loading plugin extensions  Success

→ [Go back to the top page](#)

(you can start using the installed plugins right away)

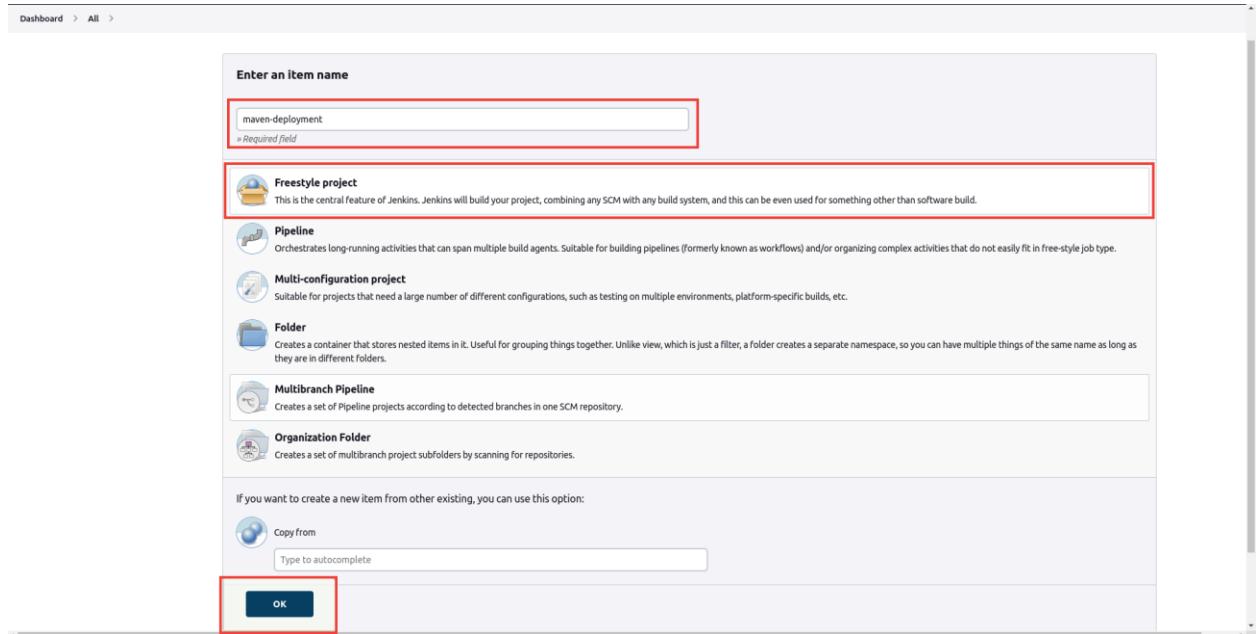


3. Wait until Jenkins restarts. Refresh your browser if Jenkins does not restart automatically after 2 minutes. Go back to the Jenkins home screen, and select **New Item**.



A screenshot of the Jenkins dashboard. A red box highlights the '+ New Item' button in the top-left corner, with a red arrow pointing to it. The dashboard shows various Jenkins management links like Dashboard, People, Build History, Manage Jenkins, My Views, and Lockable Resources. It also displays build queue and executor status. The main area is titled 'Welcome to Jenkins!' with instructions to start building a software project. Buttons for 'Create a job', 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds' are present.

4. Create a Jenkins project using the **Freestyle project** type. Give a name to your project, and then click **OK**.



You will be redirected to the project details page. In the **General** section, enter a description for your project (optional).

**General** Enabled

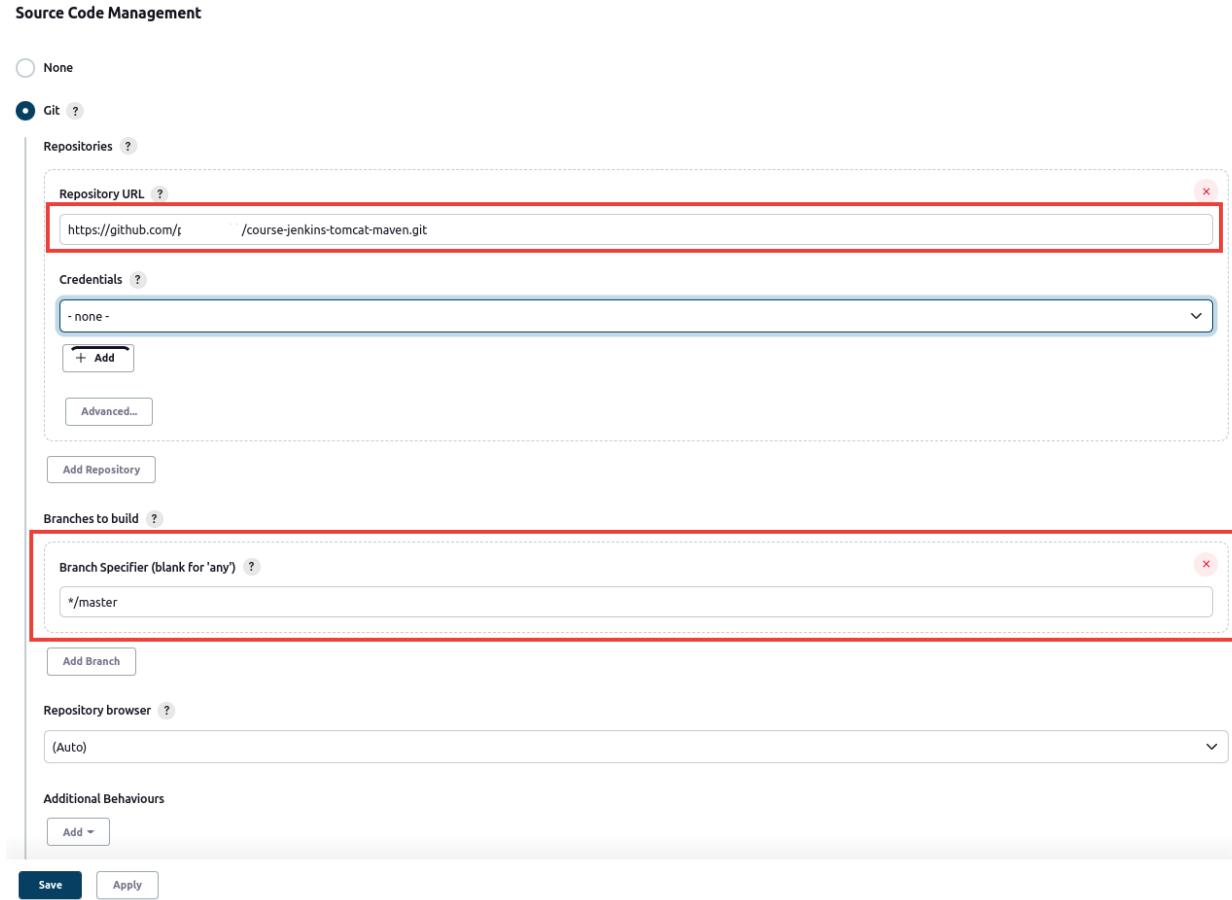
Description  
`maven project deployment.`

[Plain text] [Preview](#)

Discard old builds ?  
 GitHub project  
 This build requires lockable resources  
 This project is parameterized ?  
 Throttle builds ?  
 Execute concurrent builds if necessary ?

[Advanced...](#)

5. In the **Source Code Management** section, choose **Git** and then fill in the details of your repository:
  - a. **Repository url** :URL of your forked repository (from the task 3)
  - b. **Selected branch** : `*/master`, meaning that you will build the application from the `master` branch of the repository.



You don't need to fill in the **credentials** section since the repository is public.

6. In the **Build Steps** section, select **Invoke top-level Maven's target** from the **Add build step dropdown menu** and then type `clean install` in the Goals section. Use **my-maven** installation (Task 1). You will do this since forked repository contains a Java project configured with the Maven build tools. This will clean the previous built application (if there is any), then trigger the new build, using Maven.

## 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

≡ **Invoke top-level Maven targets** ? ×

Maven Version

my-maven

Goals

clean install

Advanced...

7. The built application will generate a Java war to be deployed. Select **Deploy war/ear to a Container** from the **Post-Build Actions** dropdown menu. Click **Save**.

Build Steps

≡ **Invoke top-level Maven targets** ? ×

Goals

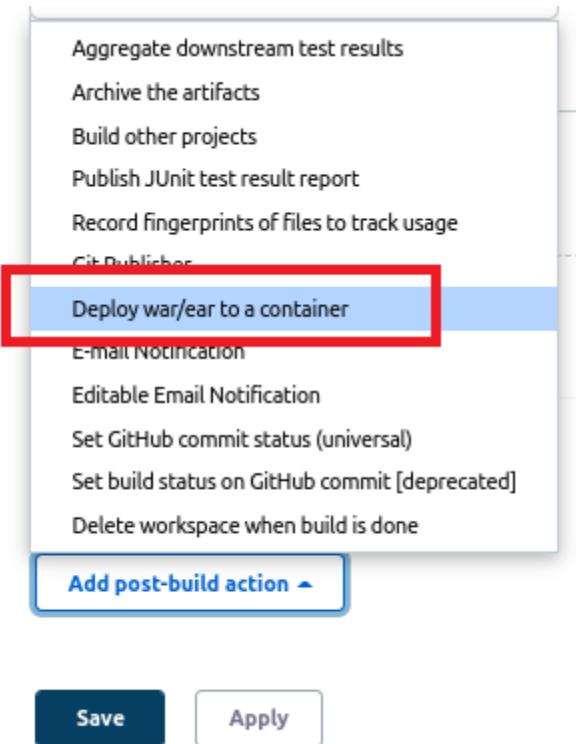
clean install

Advanced...

Add build step ▾

Post-build Actions

Add post-build action ▾



8. After selecting the post build actions, you will see the below screen.

**Post-build Actions**

≡ Deploy war/ear to a container

WAR/EAR files ?

Context path ?

Containers

Deploy on failure

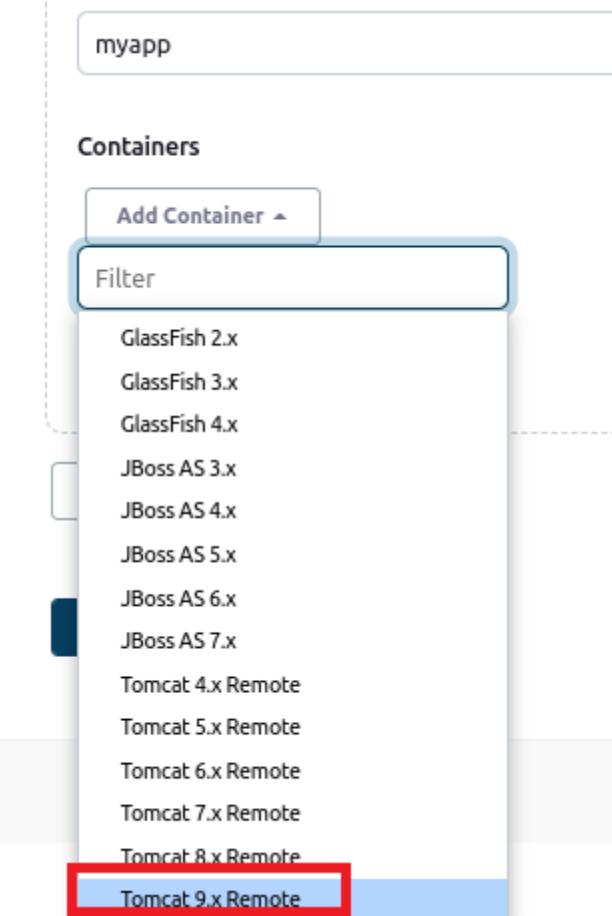
Fill in the details for the **WAR/EAR files:** `**/*.war` and **Context Path:** `myapp`

≡ Deploy war/ear to a container

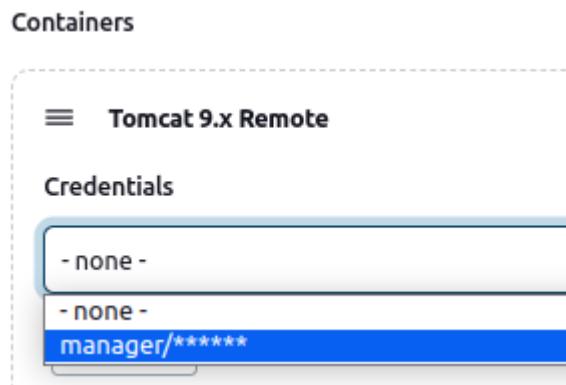
WAR/EAR files ?

Context path ?

9. In the **Container** sub section, select the version of your Tomcat server (Version 9)



10. For the credentials, use the Jenkins credential that you already set in the previous steps (with the username `manager/*****`).



11. Set **Tomcat URL** to `http://localhost:8081`, and your final result of **Post-build Actions** should look like this:

#### Post-build Actions

The screenshot shows the 'Post-build Actions' configuration screen. It includes fields for 'WAR/EAR files' (set to '\*\*/\*.war'), 'Context path' (set to 'myapp'), and a 'Containers' section for 'Tomcat 9.x Remote'. In the 'Tomcat 9.x Remote' section, 'Credentials' are set to 'manager/\*\*\*\*\*\*\*\*' and 'Tomcat URL' is set to 'http://localhost:8081'. There are 'Save' and 'Apply' buttons at the bottom.

WAR/EAR files ?  
\*\*/\*.war

Context path ?  
myapp

Containers

Tomcat 9.x Remote

Credentials  
manager/\*\*\*\*\*\*\*\*

Tomcat URL ?  
http://localhost:8081

Advanced...

Add Container ▾

Save Apply

Save your project to finish your configurations.

## Resources

### Documentation

- Deploy Java War to Tomcat
- Create Deployable War File

Mark task as complete

Report issue

## 5. Test your Jenkins Project

Run the freestyle Jenkins project that you have just created to test.

1. From the Jenkins dashboard, open your project (maven-deployment)

The screenshot shows the Jenkins dashboard with the 'maven-deployment' project listed in the build history. The project details are as follows:

S	W	Name	Last Success	Last Failure	Last Duration
		maven-deployment	N/A	N/A	N/A

Icon legend: S M L

Atom feed for all Atom feed for failures Atom feed for just latest builds

2. Test your build by selecting the **Build Now** button :

Dashboard > maven-deployment >

**Status**

</> Changes

Workspace

Build Now

Configure

Delete Project

Rename

**Project maven-deployment**

maven project deployment.

**Permalinks**

- Last build (#1), 1 min 5 sec ago
- Last stable build (#1), 1 min 5 sec ago
- Last successful build (#1), 1 min 5 sec ago
- Last completed build (#1), 1 min 5 sec ago

**Build History**

trend

Filter builds...

- Wait until the build succeeds

**Jenkins**

Dashboard > maven-deployment > #1

Status

Build #1 (Success)

No changes.

Started by user LabsUser

git Revision: e5c5ccb245f6412e4956e8d0c2f88e2a2146ec0  
Repository: https://github.com/[REDACTED]/course-jenkins-tomcat-maven.git  
refs/remotes/origin/master

Keep this build forever

Started 1 min 38 sec ago  
Took 26 sec

Add description

- After successfully building the project, go back to your workspace terminal and test the deployment using the below terminal command :

```
curl http://localhost:8081/myapp/api/now
```

```
labsuser@udemy-labs(jenkins):~/code$ curl http://localhost:8081/myapp/api/now
Now is 2023-01-03T15:15:23.698456925 labsuser@udemy-labs(jenkins):~/code$
```

You will get the result of current date and time as an API response, meaning the application is deployed successfully.

## Resources

## Documentation

- Curl Manual

Mark task as complete

Report issue

## 6. Schedule a Regular Build & Deploy

Configure a Jenkins task to automatically run every 5 minutes.

1. Curl to `http://localhost:8081/myapp/api/random` (you will get a 404 at this point since there's no code to handle this functionality). Use the terminal command `curl -i http://localhost:8081/myapp/api/random`:

```
labsuser@udemy-labs(jenkins) :~/code$ curl -i http://localhost:8081/myapp/api/random

HTTP/1.1 404
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 03 Jan 2023 15:22:02 GMT
```

2. You will add a Jenkins scheduler to run the Jenkins project periodically (every 5 minutes). From the Jenkins home screen, select your project (**maven-deployment**):

The screenshot shows the Jenkins home page. At the top, there is a navigation bar with a Jenkins logo and the word "Jenkins". Below the navigation bar, there is a "Dashboard" link. In the center, there is a "New Item" button with a plus sign. To the right of the "New Item" button, there are links for "People" (with a user icon), "Build History" (with a history icon), "Project Relationship" (with a relationship icon), "Check File Fingerprint" (with a fingerprint icon), "Manage Jenkins" (with a gear icon), "My Views" (with a user icon), and "Lockable Resources" (with a lock icon). On the far right, there is a search bar with the text "All" and a "+" button. Below the search bar, there is a table with columns "S", "W", and "Name ↓". A red arrow points from the "Check File Fingerprint" link towards the "maven-deployment" project in the table. The "maven-deployment" project is highlighted with a red border. It has a green checkmark icon and a blue Jenkins icon next to its name. The table also contains other projects like "maven-deployment-test" and "maven-deployment-test2". At the bottom of the table, there is a "Icon:" dropdown menu with options "S", "M", and "L".

The screenshot shows the Jenkins project overview for 'maven-deployment'. The top navigation bar includes the Jenkins logo and the project name. Below the navigation, there are several links: 'Status' (highlighted in blue), 'Changes', 'Workspace', 'Build Now', 'Configure' (highlighted with a red box), 'Delete Project', and 'Rename'.

3. From the project configuration, select the **Build Triggers** section, then select the **Build Periodically** option and fill the schedule using a cron notation for running the build every 5 minutes `H/5 * * * *`:

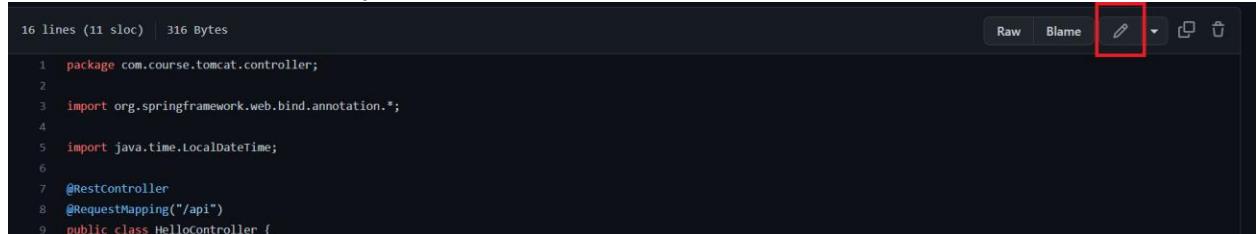
The screenshot shows the Jenkins configuration page for the 'maven-deployment' project. On the left, a sidebar lists configuration sections: General, Source Code Management (with 'Build Triggers' selected and highlighted with a red box), Build Environment, Build Steps, and Post-build Actions. On the right, the 'Build Triggers' section is expanded, showing options like 'Trigger builds remotely' and 'Build periodically'. The 'Build periodically' option is checked, and its 'Schedule' field contains the cron expression `H/5 * * * *`. At the bottom of the configuration page, there are 'Save' and 'Apply' buttons, with the 'Save' button highlighted with a red box.

Save your configuration by clicking the **Save** button on the bottom of the page.

4. Create a new functionality controller to handle the new endpoint (`/random`) . Go to your forked Github repository, and edit the file `src/main/java/com/course/tomcat/controller/HelloController.java` by copy-paste the source code below:

```
1. package com.course.tomcat.controller;
2.
3. import org.springframework.web.bind.annotation.*;
4.
5. import java.time.LocalDateTime;
6. import java.util.concurrent.ThreadLocalRandom;
7.
8. @RestController
9. @RequestMapping("/api")
10. public class HelloController {
11.
12.     @GetMapping(value = {"", "/", "/now"})
13.     String hello() {
14.         return "Now is " + LocalDateTime.now();
15.     }
16.
17.     @GetMapping(value = "/random")
18.     int random(){
19.         return ThreadLocalRandom.current().nextInt(0,1000);
20.     }
21. }
```

Choose to edit file directly



16 lines (11 sloc) | 316 Bytes

```
1. package com.course.tomcat.controller;
2.
3. import org.springframework.web.bind.annotation.*;
4.
5. import java.time.LocalDateTime;
6.
7. @RestController
8. @RequestMapping("/api")
9. public class HelloController {
```

Copy paste the source code above

Commit changes directly to the `master` branch

The screenshot shows a GitHub commit interface. At the top, it displays the repository path: `course-jenkins-tomcat-maven / src / main / java / com / course / tomcat / controller / HelloController.java` in the `master` branch. Below this is a code editor window showing `HelloController.java`. A red box highlights the following code block:

```
1 package com.course.tomcat.controller;
2
3 import org.springframework.web.bind.annotation.*;
4
5 import java.time.LocalDateTime;
6 import java.util.concurrent.ThreadLocalRandom;
7
8 @RestController
9 @RequestMapping("/api")
10 public class HelloController {
11
12     @GetMapping(value = "", "/", "/now")
13     String hello() {
14         return "Now is " + LocalDateTime.now();
15     }
16
17     @GetMapping(value = "/random")
18     int random(){
19         return ThreadLocalRandom.current().nextInt(0,1000);
20     }
21 }
```

Below the code editor is a commit message input field containing `feat: adding new random controller`, which is also highlighted with a red box. The commit message area includes an optional extended description field and two radio button options for committing:  Commit directly to the `master` branch and  Create a **new branch** for this commit and start a pull request. The "Commit changes" button at the bottom left is also highlighted with a red box.

Commit your change directly to the `master` branch of your forked repository

5. Wait for 5 minutes for the scheduled Jenkins build to start:

The screenshot shows the Jenkins interface for the project 'maven-deployment'. The top navigation bar includes the Jenkins logo and the project name. Below the navigation, there's a sidebar with various project management options like Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. The main content area is titled 'Project maven-deployment' and describes it as a 'maven project deployment'. A 'Permalinks' section lists recent builds. The 'Build History' section displays a list of previous builds, with build #7 highlighted and a red box around it. The build history table has columns for build number, date, and time.

Build	Date
#7	Jan 3, 2023, 4:00 PM
#6	Jan 3, 2023, 3:55 PM
#5	Jan 3, 2023, 3:50 PM
#4	Jan 3, 2023, 3:45 PM
#3	Jan 3, 2023, 3:40 PM
#2	Jan 2, 2023, 2:38 PM
#1	Jan 1, 2023, 1:20 PM

[Atom feed for all](#) [Atom feed for failures](#)

6. After the scheduled build finishes, go back to your workspace terminal and try to access the new `/random` endpoint using the following terminal command :

```
curl -i http://localhost:8081/myapp/api/random
```

```

labsuser@udemy-labs (jenkins) :~/code$ curl -i http://localhost:8081/myapp/api/random
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 03 Jan 2023 16:03:00 GMT

180 labsuser@udemy-labs (jenkins) :~/code$ 

```

You will get a random number as the result.

## Resources

### Documentation

- How to use cron

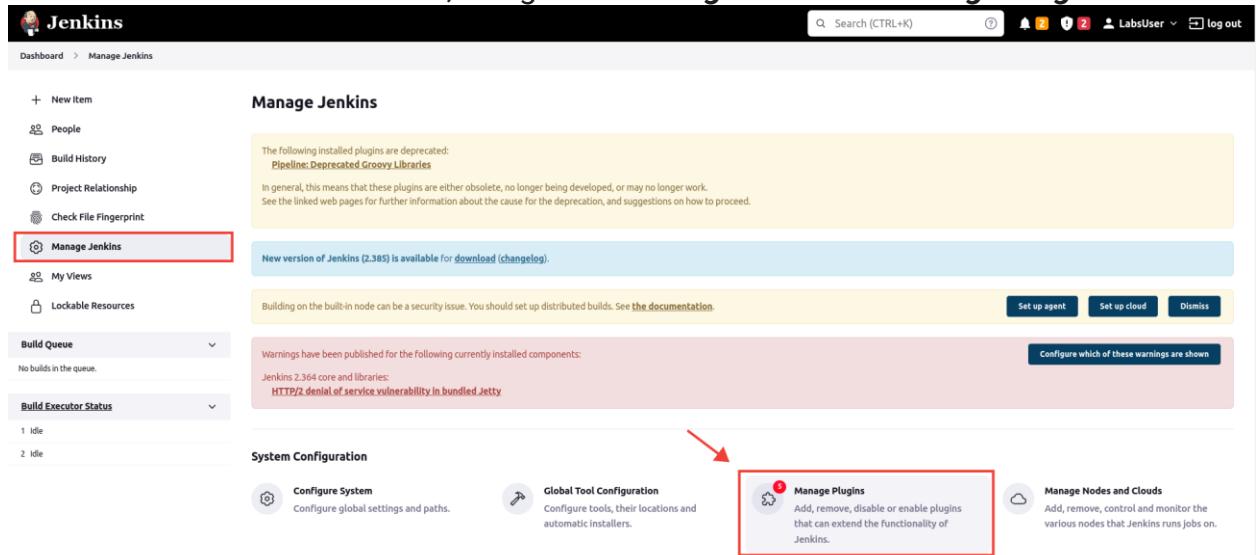
Mark task as complete

Report issue

## 7. Create Users for the Jenkins Task

Create two Jenkins users with different permissions:

- First user can only view the Jenkins project and its execution result, but cannot run the project
  - Second user can view and run the Jenkins project
1. From the Jenkins home screen, navigate to **Manage Jenkins > Manage Plugins**.



Under the **Available** tab, search for **Role-based Authorization Strategy**. This plugin allows you to manage users, giving each user specific access permissions.

## Plugin Manager

The screenshot shows the Jenkins Plugin Manager interface. At the top, there are tabs for 'Updates', 'Available' (which is highlighted with a red border), 'Installed', and 'Advanced'. Below these, a search bar contains the text 'Role-based Authorization Strategy'. A plugin card for 'Role-based Authorization Strategy' is displayed, showing its name, version (584.vf8e515397ecd), category ('Security Authentication and User Management'), release date ('1 day 12 hr ago'), and a brief description: 'Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.' At the bottom of the card are three buttons: 'Install without restart' (highlighted with a red box), 'Download now and install after restart', and 'Check now'.

Choose **Install without restart**

2. Back on the Jenkins home screen, navigate to **Manage Jenkins > Configure Global Security**

The screenshot shows the Jenkins Manage Jenkins screen. The left sidebar includes links for 'Check File Fingerprint', 'Manage Jenkins' (highlighted with a red box), 'My Views', and 'Lockable Resources'. The main area has sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 idle, 2 idle). The 'System Configuration' section contains several links: 'Configure System' (Configure global settings and paths), 'Global Tool Configuration' (Configure tools, their locations and automatic installers), 'Manage Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), 'Manage Nodes and Clouds' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on), 'Configure Global Security' (Secure Jenkins; define who is allowed to access/use the system, highlighted with a red box and indicated by a red arrow), 'Manage Credentials' (Configure credentials), 'Configure Credential Providers' (Configure the credential providers and types), and 'Manage Users' (Create/delete/modify users that can log in to this Jenkins). A message at the top right says 'New version of Jenkins (2.385) is available for download (changelog)'.

And then select **Role-Based Strategy** from the **Authorization** dropdown menu:

The screenshot shows the Jenkins 'Configure Global Security' configuration page. Under the 'Authorization' section, the 'Role-Based Strategy' option is selected from a dropdown menu. The 'Save' button at the bottom of the page is highlighted with a red box.

**Save** the configuration.

- Back on the Jenkins home screen, navigate to **Manage Jenkins > Manage User > select *Create User***. Create two Jenkins users:

The screenshot shows the Jenkins home screen. The 'Manage Jenkins' link in the left sidebar is highlighted with a red box. In the 'System Configuration' section, the 'Manage Users' link in the 'Security' category is highlighted with a red box and has a red arrow pointing to it.

Create 2 new users by selecting the ***Create User*** button:

**Jenkins**

Dashboard > Manage Jenkins > Jenkins' own user database

**Users**

+ Create User

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User ID ↓	Name
labuser	LabsUser

Username: **usertrigger** (can view and run the job)

**Jenkins**

Dashboard > Manage Jenkins > Jenkins' own user database

**Create User**

+ Create User

Username: **usertrigger**

Password: **\*\*\*\*\***

Confirm password: **\*\*\*\*\***

Full name: **User Trigger**

E-mail address: **usertrigger@mail.com**

**Create User**

Username: **userreadonly** (can only view)

**Jenkins**

Dashboard > Jenkins' own user database

**Create User**

+ Create User

Username: **userreadonly**

Password: **\*\*\*\*\***

Confirm password: **\*\*\*\*\***

Full name: **User Read-only**

E-mail address: **userreadonly@mail.com**

**Create User**

The user list will be like the following:

The screenshot shows the Jenkins 'Users' management interface. At the top, there's a search bar and a log out link. Below it, a table lists users with columns for User ID and Name. The user 'userreadonly' is highlighted with a red border. Each user row has edit and delete icons.

User ID	Name	
labuser	LabsUser	
userreadonly	User Read-only	
usertrigger	User Trigger	

4. Back on the Jenkins home screen, navigate to **Manage Jenkins > Manage and Assign Roles > Manage Roles**.

The screenshot shows the Jenkins 'Manage Jenkins' configuration page. On the left, a sidebar lists various management options. The 'Manage Jenkins' link is highlighted with a red box. The main content area includes sections for system configuration, security, and global tool configuration. In the 'System Configuration' section, there are links for 'Configure System', 'Global Tool Configuration', 'Manage Plugins', and 'Manage Nodes and Clouds'. In the 'Security' section, there are links for 'Configure Global Security', 'Manage Credentials', and 'Configure Credential Providers'. A red arrow points from the 'Configure Credential Providers' link to the 'Manage and Assign Roles' link in the 'Manage and Assign Roles' section, which is also highlighted with a red box.

The screenshot shows the Jenkins 'Manage and Assign Roles' page. At the top, there's a navigation bar with 'Dashboard > Manage Jenkins > Manage and Assign Roles'. Below this is a sidebar with various Jenkins management links like 'New Item', 'People', 'Build History', etc. The main area is titled 'Manage and Assign Roles' and contains three main sections: 'Manage Roles' (with a red box around it), 'Assign Roles', and 'Role Strategy Macros'. Each section has a brief description and a link to its details.

5. Create the role as follows:

- Role name: `trigger` make sure to check the options below:
  - Overall: Read**
  - Job: Build / Cancel / Configure / Create / Read**
  - View: Read**

The screenshot shows the Jenkins 'Manage Roles' page. It lists 'Global roles' and allows users to add new roles. A search bar at the top right contains 'LabsUser'. In the 'Role to add' section, the role name `trigger` is entered into a text input field, which is highlighted with a red box. Below the input field is an 'Add' button.

- Role name: `read-only` make sure to check the options below:
  - Overall: Read**
  - Job: Read**
  - View: Read**

## 6. Your final settings will look like this:

7.  
8.

Save your configuration

- Back on the Jenkins home screen, navigate **Manage Jenkins > Manage and Assign Roles > Assign Roles** > and assign your created users into the created roles from the previous step.

The screenshot shows the Jenkins 'Manage Jenkins' page. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', and 'Manage Jenkins'. The 'Manage Jenkins' link is highlighted with a red box. The main content area has several sections: one about deprecated plugins (Pipeline: Deprecated Groovy Libraries), a notice for version 2.385, a warning about building on the built-in node, and a section about installed components (Jenkins 2.364 core and libraries: HTTP/2 denial of service vulnerability in bundled Jetty). At the bottom, there's a 'System Configuration' section with links for 'Configure System', 'Global Tool Configuration', 'Manage Plugins', 'Manage Nodes and Clouds', 'Configure Global Security', 'Manage Credentials', 'Configure Credential Providers', and 'Manage and Assign Roles'. A red arrow points from the 'Manage Jenkins' link in the sidebar to the 'Manage and Assign Roles' link in the configuration section.

The screenshot shows the Jenkins 'Manage and Assign Roles' page. The top navigation bar includes 'Dashboard', 'Manage Jenkins', and 'Manage and Assign Roles'. The main content area has a title 'Manage and Assign Roles' and a sidebar with links: '+ New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', and 'Lockable Resources'. To the right, there are four main sections: 'Manage Roles' (fingerprint icon), 'Assign Roles' (two people icon, highlighted with a red box), 'Role Strategy Macros' (puzzle piece icon), and 'Provides info about macro usage and available macros'. A red box highlights the 'Assign Roles' link.

- Assign the `User Read Only` to role `read-only`
- Assign the `User Trigger` to role `trigger`

The screenshot shows the Jenkins 'Assign Roles' configuration page. In the 'Global roles' section, 'LabsUser' is assigned to 'Read-Only' and 'Trigger'. In the 'Item roles' section, 'Anonymous' is assigned to 'Trigger'. A red box highlights the 'Save' button at the bottom.

Save your configurations

## 10. Log out from your current session:

The screenshot shows the Jenkins dashboard. The 'log out' link in the top right corner is highlighted with a red arrow.

and then log in as the user `usertrigger` that should be able to execute the job:



The image shows the Jenkins project dashboard for "maven-deployment". The top navigation bar includes "Dashboard", "User Trigger", and "log out". The main area displays the project name "Project maven-deployment" and a status bar with "Status", "Changes", "Build Now" (which is highlighted with a red box), "Configure", and "Rename". Below this is a "Build History" section with a table showing two builds: #112 (Jan 4, 2013, 10:48 AM) and #111 (Jan 4, 2013, 10:40 AM). A red box highlights the "Build Now" button. On the right side, there are links for "Edit description" and "Disable Project".

Test and confirm that the `usertrigger` have the permissions to trigger the build.

## 11. Log out from your current session:

The image shows the Jenkins log out screen. It features the Jenkins logo and navigation bar with "Dashboard". On the right side, there is a "User Trigger" dropdown and a "log out" button. A red arrow points to the "log out" button.

and then log in as the user `userreadonly`:



A screenshot of the Jenkins project dashboard for "maven-deployment". The dashboard includes a sidebar with "Status" and "Build History" sections, and a main area titled "Project maven-deployment" with a description of "maven project deployment". The "Build History" section shows five recent builds: #13 (Jan 4, 2023, 10:48 AM), #12 (Jan 4, 2023, 10:45 AM), #11 (Jan 4, 2023, 10:40 AM), and #10 (Jan 4, 2023, 10:35 AM). A red box highlights the "Build History" section. The top right of the dashboard shows search, user status, and log out links.

Confirm that the `userreadonly` can only view the build status and progress. The **Build Now** button should **not** available.

## Resources

### Documentation

- Jenkins Role Based Authorization

Mark task as complete

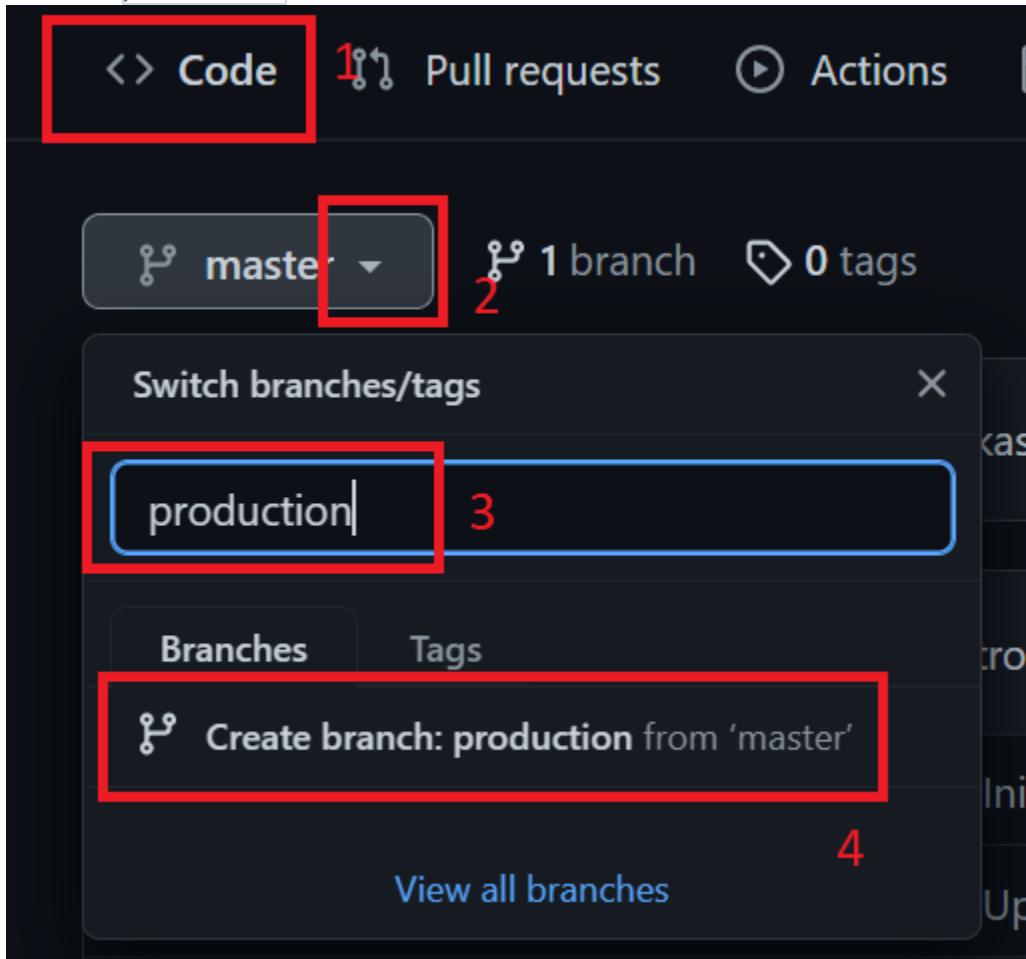
Report issue

## 8. Configure Jenkins to Automatically Trigger the Jenkins Task

Create a `production` branch on your forked repository.

Configure Jenkins to scan the repository for code change every  $n$  minutes and build the task automatically when there is a code change on `production` branch.

1. Go to your Github forked repository. Create a new branch from master: `production`



2. In your workspace terminal, curl to /hero (you will get a 404 at this point):

```
curl -i http://localhost:8081/myapp/api/hero
```

```
oabsuser@udemylabs(jenkins):~/code$ curl -i http://localhost:8081/myapp/api/hero
```

```
HTTP/1.1 404
```

```
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date:
```

3. Open your Jenkins Dashboard, navigate to your project (maven-deployment):

The screenshot shows the Jenkins dashboard interface. At the top, there is a navigation bar with a Jenkins logo and the word "Jenkins". Below this is a header bar with the text "Dashboard >". The main content area contains several links: "New Item", "People", "Build History", "Project Relationship", "Check File Fingerprint", "Manage Jenkins", "My Views", and "Lockable Resources". To the right of these links is a search bar with the text "All" and a "+" button. Below the search bar is a table with three columns: "S", "W", and "Name ↓". A row in this table is highlighted with a red border, containing a green checkmark icon, a blue Jenkins icon, and the text "maven-deployment". At the bottom of the table, there is a "Icon:" label followed by "S", "M", and "L" buttons.

Reconfigure your project to configure `production` config:

The screenshot shows the Jenkins interface for a project named 'maven-deployment'. The top navigation bar includes the Jenkins logo and the word 'Jenkins'. Below it, the breadcrumb navigation shows 'Dashboard > maven-deployment >'. A horizontal menu bar contains several options: 'Status' (selected), 'Changes', 'Workspace', 'Build Now', 'Configure' (highlighted with a red box), 'Delete Project', 'Git Polling Log', and 'Rename'. The 'Configure' option is specifically highlighted with a red border.

Status

</> Changes

Workspace

Build Now

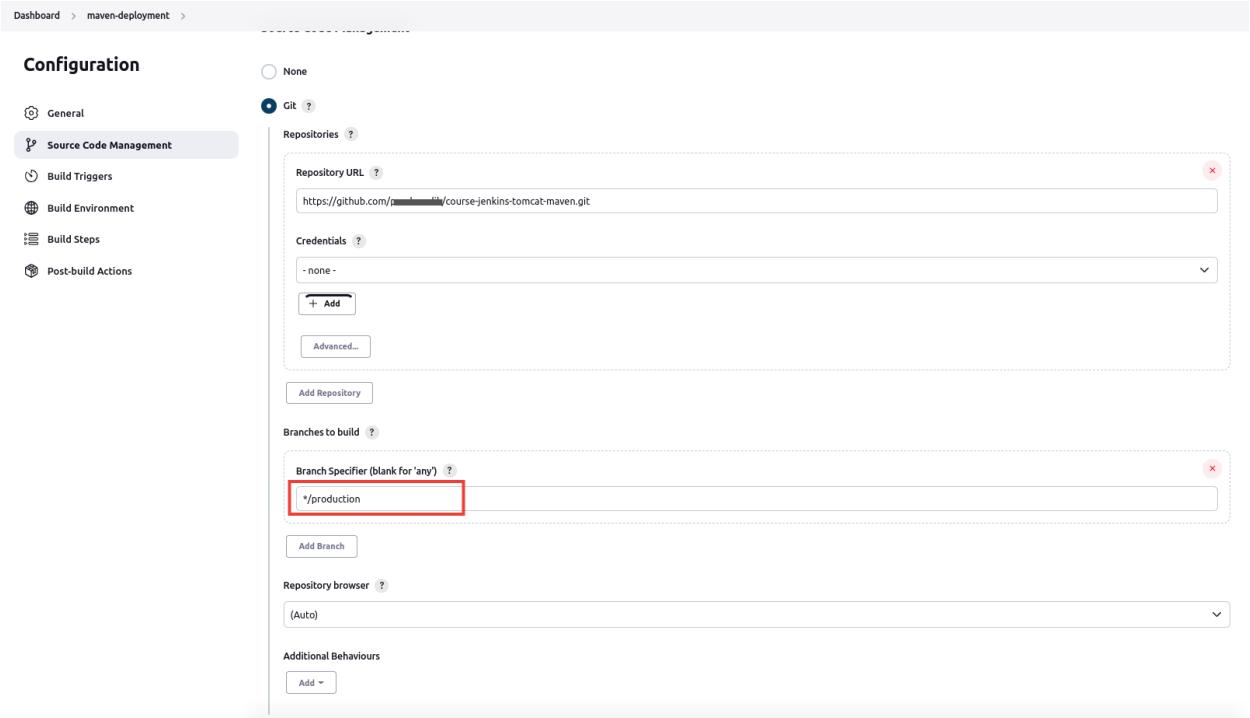
Configure

Delete Project

Git Polling Log

Rename

In the **Source Code Management** section, enter `*/production` into the **Branch Specifier** field under **Branches to build**:



In the **Build Trigger** section, unselect the **Build periodically** option and select the **Poll SCM** instead. Add cron annotation to scan the repository every 5 minutes (`H/5 * * * *`), **Poll SCM** option allows the Jenkins services to scan and build the project if there are any changes on the working branch (`production`). Unlike the **Build periodically** option that will always run the Jenkins task, **Poll SCM** will scan the repository for changes in the source code every 5 minutes, but will only run the Jenkins task if it detects a change in the source code.

The screenshot shows the Jenkins 'Configuration' page for a job named 'maven-deployment'. On the left, there are several tabs: General, Source Code Management, Build Triggers (which is selected and highlighted with a red box), Build Environment, Build Steps, and Post-build Actions. In the 'Build Triggers' section, there are four options: 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically' (which is checked and highlighted with a red box), and 'GitHub hook trigger for GITScm polling'. Below these is a 'Schedule' field containing 'H/5 \* \* \* \*'. At the bottom of the 'Build Triggers' section are two buttons: 'Save' (highlighted with a red box) and 'Apply'.

4. Go to your forked Github repository, and add a new endpoint on the file **/src/main/java/com/course/tomcat/controller/HelloController.java**

Add this to the `master` branch

The screenshot shows a GitHub repository page for 'course-jenkins-tomcat-maven'. The 'master' branch is selected (highlighted with a red box). The file 'HelloController.java' is open (also highlighted with a red box). The code editor shows the following Java code:

```

1 package com.course.tomcat.controller;
2
3 import org.springframework.web.bind.annotation.*;
4
5 import java.time.LocalDateTime;
6 import java.util.concurrent.ThreadLocalRandom;

```

At the bottom right of the code editor, there is a red box highlighting the edit icon (pencil symbol).

Copy and paste the entire source code below.

1. `package com.course.tomcat.controller;`
- 2.
3. `import org.springframework.web.bind.annotation.*;`
- 4.

```
5. import java.time.LocalDateTime;
6. import java.util.concurrent.ThreadLocalRandom;
7.
8. @RestController
9. @RequestMapping("/api")
10. public class HelloController {
11.
12.     @GetMapping(value = {"", "/", "/now"})
13.     String hello() {
14.         return "Now is " + LocalDateTime.now();
15.     }
16.
17.     @GetMapping(value = "/random")
18.     int random(){
19.         return ThreadLocalRandom.current().nextInt(0,1000);
20.     }
21.
22.     @GetMapping(value = "/hero")
23.     String hero(){
24.         return "My favorite superhero is Batman";
25.     }
26.
27. }
```

Add any commit message you want.

Commit directly to the `master` branch

Click **Commit changes**

```

course-jenkins-tomcat-maven / src / main / java / com / course / tomcat / controller / HelloController.java in master
Cancel changes

<> Edit file ⌂ Preview changes Spaces 4 No wrap

1 package com.course.tomcat.controller;
2
3 import org.springframework.web.bind.annotation.*;
4
5 import java.time.LocalDateTime;
6 import java.util.concurrent.ThreadLocalRandom;
7
8 @RestController
9 @RequestMapping("/api")
10 public class HelloController {
11
12     @GetMapping(value = {"", "/", "/now"})
13     String hello() {
14         return "Now is " + LocalDateTime.now();
15     }
16
17     @GetMapping(value = "/random")
18     int random(){
19         return ThreadLocalRandom.current().nextInt(0,1000);
20     }
21
22     @GetMapping(value = "/hero")
23     String hero(){
24         return "My favorite superhero is Batman";
25     }
26
27 }
28

```

**Commit changes**

feat: adding hero endpoint

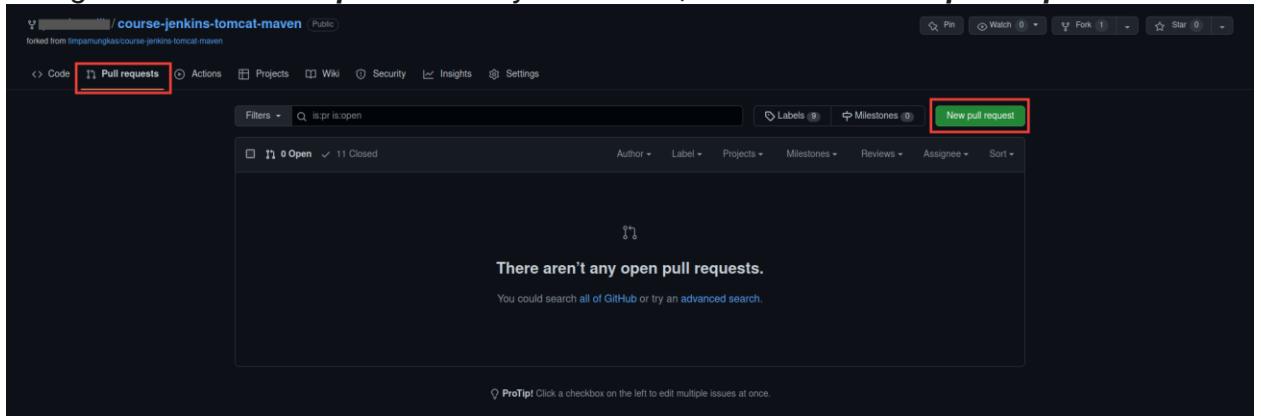
Add an optional extended description...

Commit directly to the `master` branch.

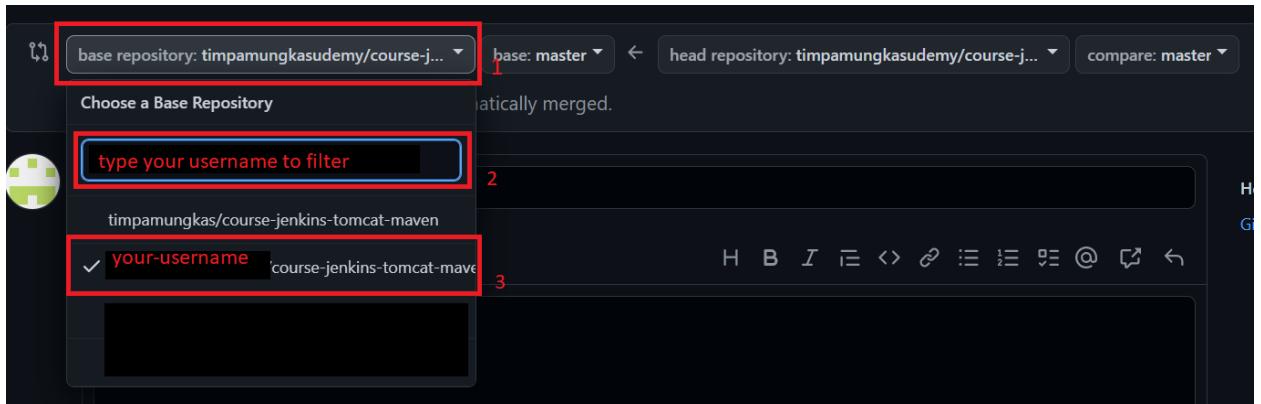
Create a [new branch](#) for this commit and start a pull request. [Learn more about pull requests.](#)

**Commit changes** Cancel

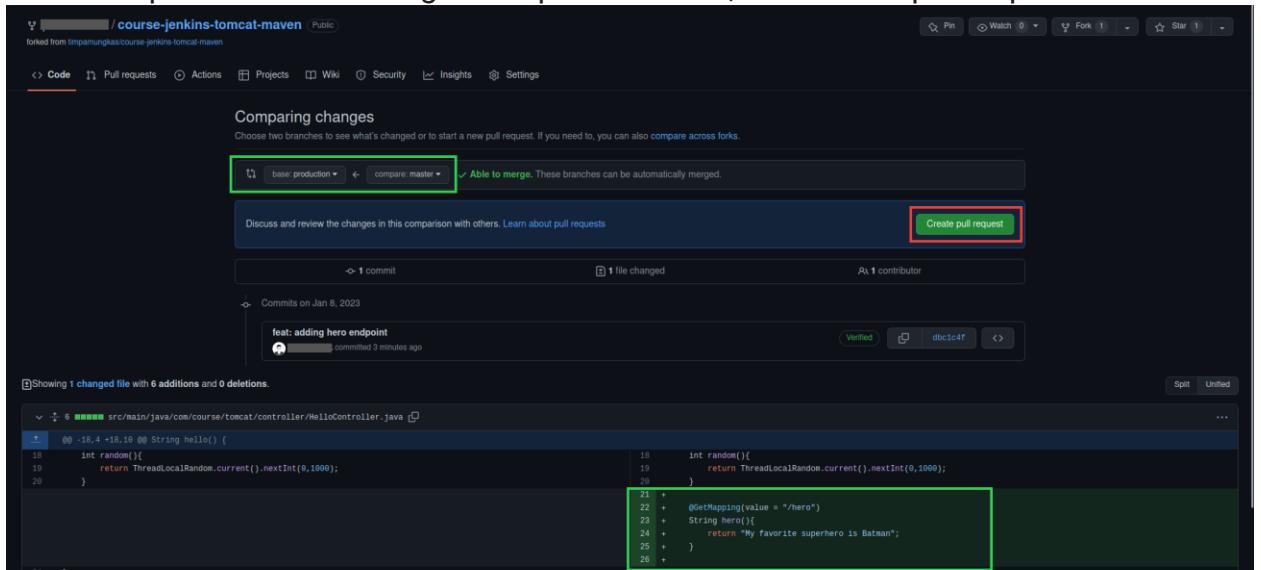
## 5. Navigate to the **Pull Request** tab on your GitHub, and create a **New pull request**:



Create a Pull request from the `master` branch into the `production` branch. Select your own forked repository from the dropdown

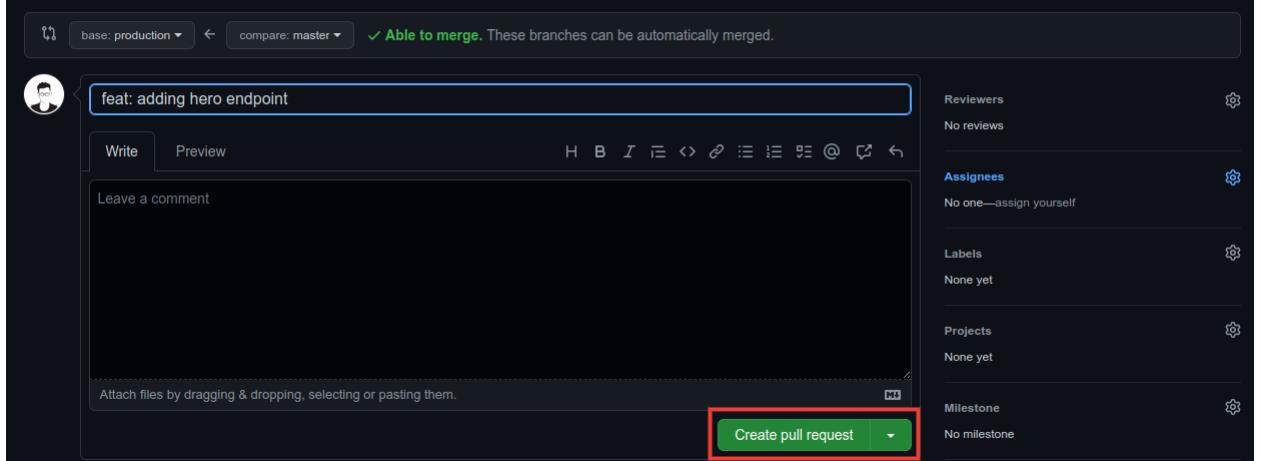


so the dropdown will be changed like picture below, then create pull request :

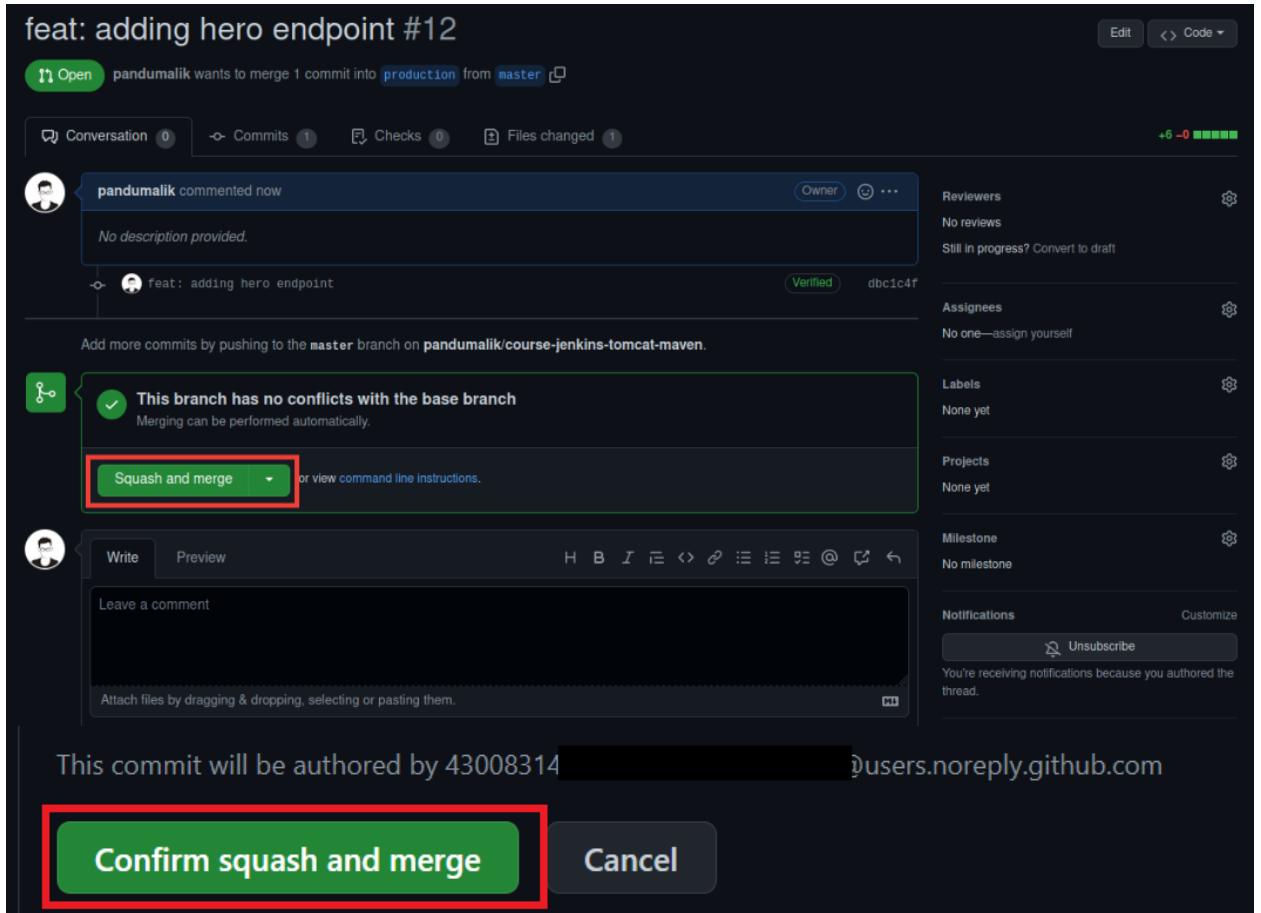


### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.



6. Approve the created Pull Request. Optionally, choose **Squash and merge** to merge multiple commits into one and simplify the commit list:



- At this point, Jenkins will scan the repository every 5 minutes. Since there is a change on the code, the build task will be executed.

The screenshot shows the Jenkins build history for build #7. The build status is "Build #7 (Jan 8, 2023, 3:45:07 PM)" with a green checkmark icon. A green box highlights this status. The build log shows "Changes" and "1. feat: adding hero endpoint (details / githubweb)".

- Test the new deployment using curl in your workspace terminal. It should now work. Use the following terminal command:

```
curl -i http://localhost:8081/myapp/api/hero
```

```
labsuser@udemy-labs(jenkins):~/code$ curl -i http://localhost:8081/myapp/api/hero
HTTP/1.1 200
Content-Type: text/plain; charset=UTF-8
Content-Length: 31
Date: [REDACTED]

My favorite superhero is Batman[REDACTED]labsuser@udemy-labs(jenkins):~/code$
```

## Resources

### Documentation

- Git pull request
- Mark task as complete  
Report issue

### 9. Add Jenkins CI to Slack for Notifications

Register to [Slack](#) messaging, and add Jenkins CI Slack application.

1. Register for a free [Slack](#) account and sign in.

# Sign in to Slack

We suggest using the email address you use at work.



[Sign in with Google](#)



[Sign in with Apple](#)

OR

name@work-email.com

[Sign In with Email](#)

2. Create a new Slack workspace, name it as you like then navigate to your newly created workspace.

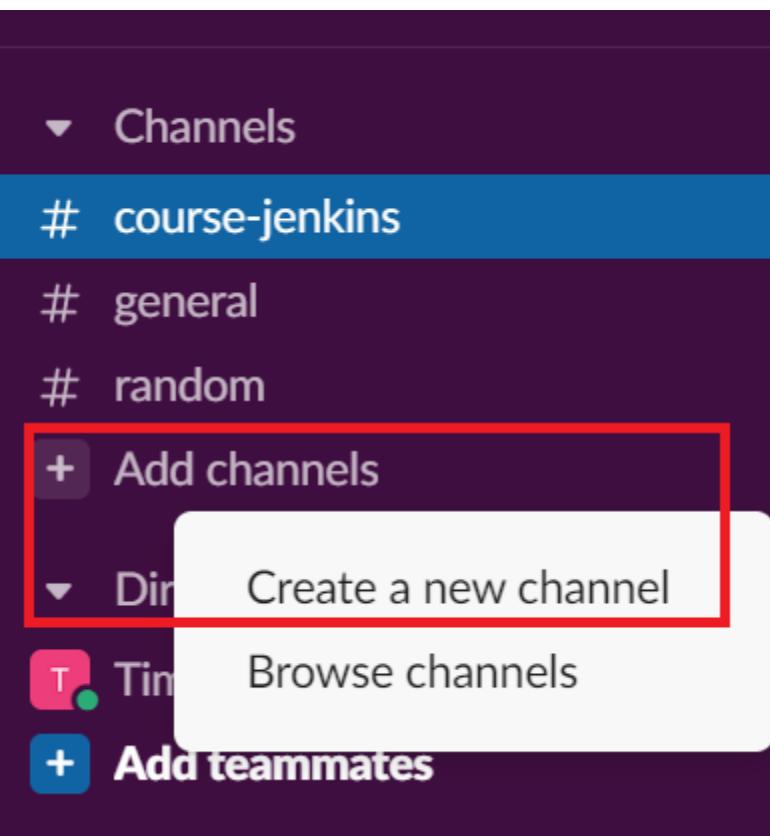
# Create a new Slack workspace

Slack gives your team a home – a place where they can talk and work together. To create a new workspace, click the button below.

[Create a Workspace →](#)

By continuing, you're agreeing to our Customer Terms of Service, User Terms of Service, Privacy Policy, and Cookie Policy.

3. Create a channel to receive the Jenkins notifications. For example, name it ***jenkins-maven-deployment***



# Create a channel

X

Channels are where your team communicates. They're best when organized around a topic — #marketing, for example.

Name

# jenkins-maven-deployment

56

Description (optional)

What's this channel about?

Make private

When a channel is set to private, it can only be viewed or joined by invitation.

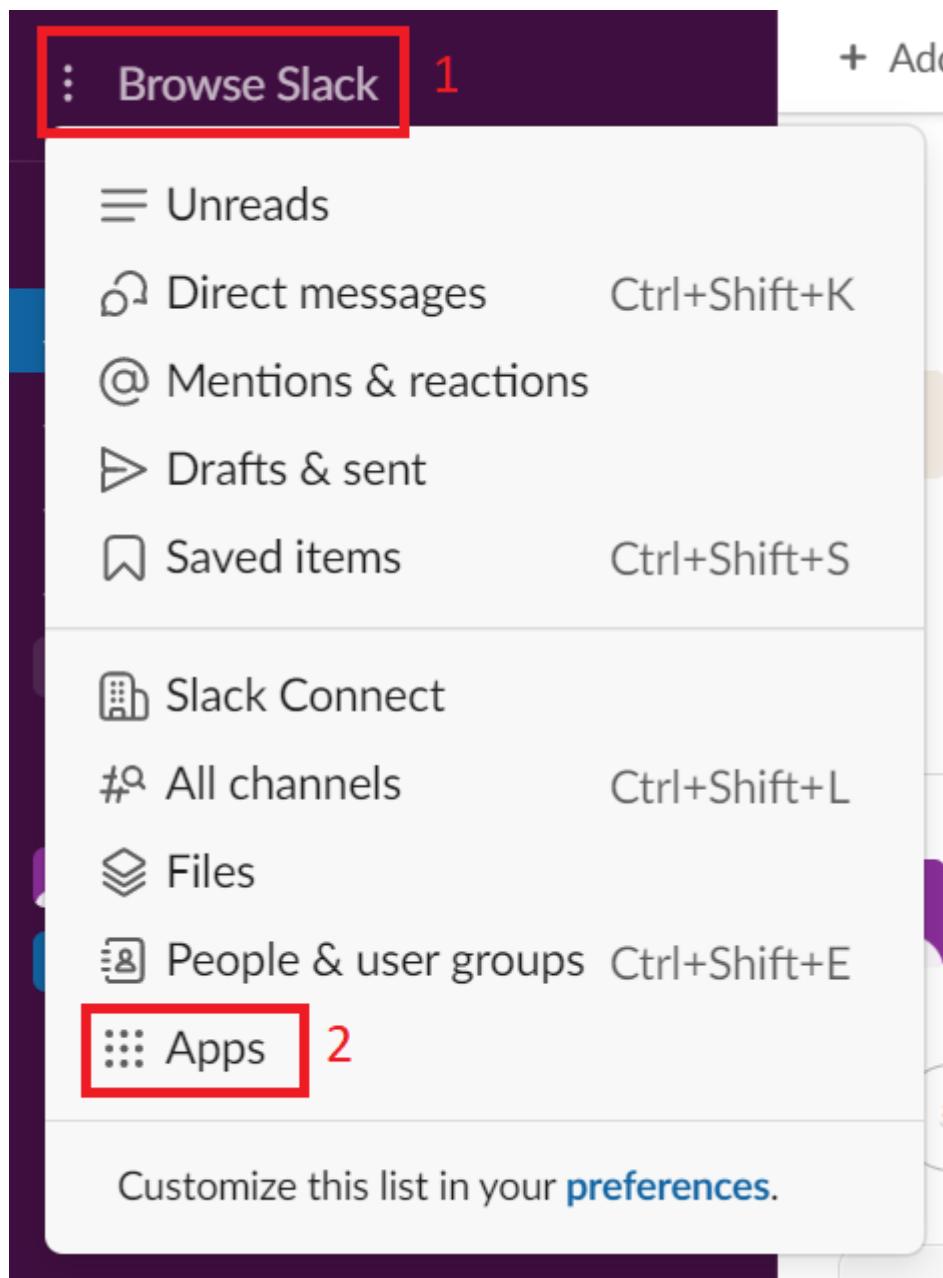


Share outside course-jenkins ⓘ

PREMIUM

Create

4. Navigate to the Slack apps directory from your Slack sidebar menu

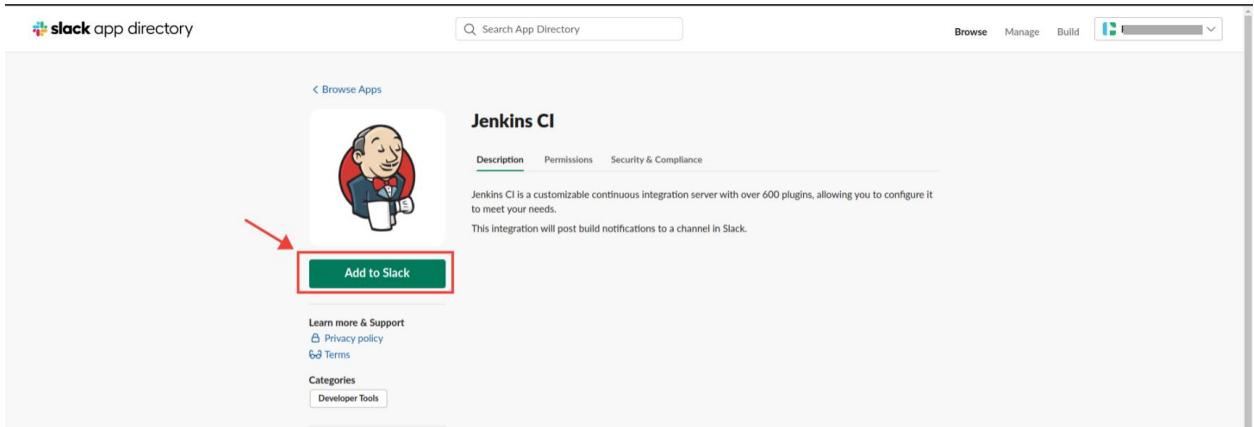


The screenshot shows the Slack App Directory search interface. At the top, there is a dark purple header with the text "course-jenkins" and a user icon. To the right of the header are the words "Apps PRO" and "App Directory" (which is highlighted with a red box). Below the header is a search bar with the placeholder text "Search by name or category (e.g. productivity, sales)". Underneath the search bar, there is a "Recommended apps" section. At the bottom right of the header area is a "Filter" icon.

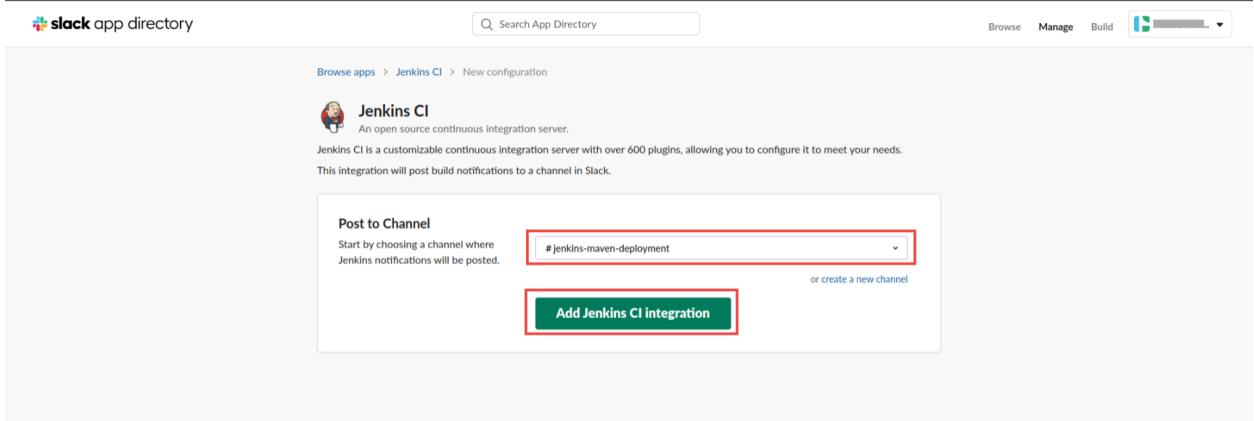
- From the Slack App Directory, click **Browse** (1) and search for [Jenkins CI](#) (2)

The screenshot shows the Jenkins CI app page in the Slack App Directory. At the top left, there is a "slack app directory" logo. To the right of the logo are three buttons: "1 Browse" (highlighted with a red box), "Manage", and "Build". On the far right, there is a "course-jenkins" user icon and a dropdown arrow. Below the top navigation, there is a "Staff Picks" section with categories: "Featured", "Enterprise-ready apps", and "Working from home". To the right of the Staff Picks section is a search bar with the placeholder text "Q Jenkins CI" (highlighted with a red box). Below the search bar, there is a brief description of the Jenkins CI app: "Jenkins CI An open source continuous integration server.".

- Add Jenkins CI to Slack



Enter your channel name from the previous step and then press **Add Jenkins CI Integration** to connect Jenkins CI to your Slack channel



7. After finishing adding the Jenkins CI, you will be redirected to the integrations guide page:

The screenshot shows the Jenkins CI integration page in the Slack App Directory. It includes setup instructions, step-by-step guides for managing Jenkins and installing the plugin, and a configuration form for Global Slack Notifier Settings.

**Save your Team Subdomain and Integration Token Credential ID for later use.**

### Step 3

After it's installed, click on **Manage Jenkins** again in the left navigation, and then go to **Configure System**. Find the **Global Slack Notifier Settings** section and add the following values:

- **Team Subdomain:** `privateworkspace`
- **Integration Token Credential ID:** Create a secret text credential using `o4suEU1` as the value

The other fields are optional. You can click on the question mark icons next to them for more information. Press **Save** when you're done.

**Note:** Please remember to replace the Integration Token in the screenshot below with your own.

The configuration page shows the following settings:

- Slack compatible app URL (optional): `privateworkspace`
- Team Subdomain: `jenkins-slack-plugin`
- Integration Token: `o4suEU1` (with a warning message: "⚠️ Exposing your Integration Token is a security risk. Please use the Integration Token Credential ID")
- Integration Token Credential ID: `some text (bot user slack token)`
- Is Bot User?:
- Channel or Slack ID: `slack-plugin-testing`

A "Test Connection" button is at the bottom right.

## Resources

### Documentation

- Slack Jenkins App

Mark task as complete

Report issue

## 10. Configure Jenkins for Slack Notifications

Configure Jenkins to send a notification to [Slack](#) every time a build task is finished regardless of the result.

1. Go to your Jenkins home screen, then navigate to **Manage Jenkins > Manage Credentials** then add your Slack integration credentials using your saved **Integration Token Credential ID** from the previous steps:

The screenshot shows the Jenkins 'Manage Jenkins' configuration page. The 'Manage Jenkins' link in the sidebar is highlighted with a red box. In the main content area, under 'System Configuration', there is a 'Security' section with a 'Manage Credentials' link, which is also highlighted with a red box. A red arrow points from the text 'Stores scoped to Jenkins' below to this 'Manage Credentials' link.

### Stores scoped to Jenkins

The screenshot shows the Jenkins 'Global credentials (unrestricted)' page. The 'System' store is selected, and the '(global)' scope is highlighted with a red box. A red box also highlights the '+ Add Credentials' button in the top right corner.

### Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

And then enter your credential details:

- a. Kind:
- b. Scope:
- c. Secret: the **Integration Token Credential ID** you saved from the previous steps
- d. ID:

New credentials

Kind

Secret text

Scope ? Global (Jenkins, nodes, items, all child items, etc.)

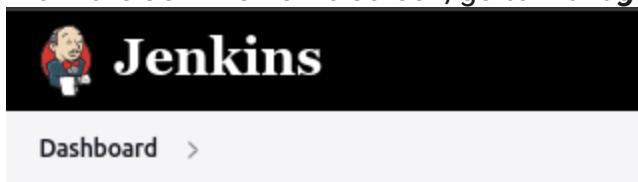
Secret

ID ? slack-integration

Description ?

Create

- From the Jenkins home screen, go to **Manage Jenkins** :



+ New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Lockable Resources

## Open **Manage Plugins**

### System Configuration



Configure System

Configure global settings and paths.



Global Tool Configuration

Configure tools, their locations and automatic installers.



Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



Manage Nodes and Clouds

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Install the **Slack Notifications** plugin from the **Available** tab

## Plugin Manager

Updates Available Installed Advanced

Q slack

Install Name ↓ Released

**Slack Notification** 631.v40deea\_40323b  
slack Build Notifiers 2 mo 0 days ago  
Integrates Jenkins with Slack, allows publishing build statuses, messages and files to Slack channels.

**Global Slack Notifier** 1.5 slack 3 yr 10 mo ago  
This plugin post to slack after any build completed without any job setting.

**Install without restart** **Download now and install after restart** Update information obtained: 8 hr 37 min ago **Check now**

3. Select the checkbox at the bottom of the page to restart Jenkins.

# Installing Plugins/Upgrades

## Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Loading plugin extensions ✓ Success

Loading plugin extensions ✓ Success

Slack Notification ✓ Success

Loading plugin extensions ✓ Success

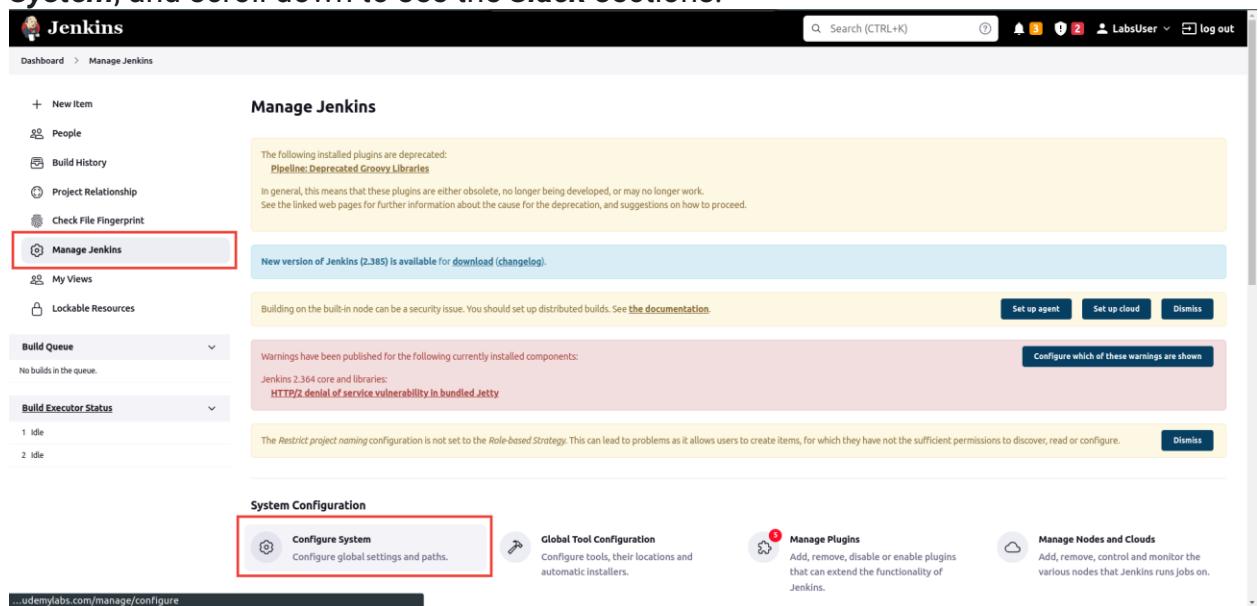
→ [Go back to the top page](#)

(you can start using the installed plugins right away)

→  Restart Jenkins when installation is complete and no jobs are running

Wait for Jenkins to restart. Refresh your browser if you are not redirected to Jenkins in 2 minutes.

4. Back on your Jenkins home screen, navigate to **Manage Jenkins > Configure System**, and scroll down to see the **Slack** sections:



The screenshot shows the Jenkins 'Manage Jenkins' configuration page. The left sidebar has a 'Manage Jenkins' link that is highlighted with a red box. The main content area contains several sections:

- Manage Jenkins:** A warning about deprecated plugins (Pipeline: Deprecated Groovy Libraries) and a note about a new Jenkins version (2.385).
- System Configuration:** This section is highlighted with a red box. It includes links for 'Configure System' (Configure global settings and paths), 'Global Tool Configuration' (Configure tools, their locations and automatic installers), 'Manage Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), and 'Manage Nodes and Clouds' (Add, remove, control and monitor the various nodes that Jenkins runs on).
- Global Tool Configuration:** Configure tools, their locations and automatic installers.
- Manage Plugins:** Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- Manage Nodes and Clouds:** Add, remove, control and monitor the various nodes that Jenkins runs on.

Fill the details of your Slack configurations:

- a. For workspace use the saved **Team subdomain** from the previous steps
- b. For credential, type in **slack-integration**
- c. Select channel name to send the notification
- d. Save it

The screenshot shows the Jenkins 'Configure System' page under 'Manage Jenkins'. The 'Slack' section is expanded. Three input fields are highlighted with red boxes: 'Workspace' containing 'steworkspace', 'Credential' containing 'slack-integration', and 'Default channel / member id' containing '#jenkins-maven-deployment'. At the bottom left, the 'Save' button is also highlighted with a red box.

Save your configurations.

5. Back on the Jenkins home, select your project and configure your project (**maven-deployment**):



# Jenkins

Dashboard >

+ New Item

People

All

+

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Lockable Resources

S W Name ↓



maven-deployment

Icon: S M L

The screenshot shows the Jenkins interface for a project named 'maven-deployment'. At the top, there's a navigation bar with the Jenkins logo and the word 'Jenkins'. Below it, a breadcrumb navigation shows 'Dashboard > maven-deployment >'. The main content area has a tab bar with 'Status' selected. Below the tabs, there are several options: 'Changes' (with a file icon), 'Workspace' (with a folder icon), 'Build Now' (with a play icon), 'Configure' (with a gear icon, which is highlighted with a red border), 'Delete Project' (with a trash bin icon), 'Git Polling Log' (with a clipboard icon), and 'Rename' (with a pencil icon). A horizontal line separates this from the bottom section.

Dashboard > maven-deployment >

Status

</> Changes

Workspace

Build Now

Configure

Delete Project

Git Polling Log

Rename

Go to **Post-Build Actions** and select **Slack Notifications** from the **Add post-build action** dropdown menu.

Dashboard > maven-deployment >

## Configuration

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

**Deploy war/ear to a container**

WAR/EAR Files: `**/*.war`

Context path: `myapp`

**Containers**

**Tomcat 9.x Remote**

Credentials: `manager/*****`

Add Container

Tomcat URL: `http://localhost:8081`

Advanced...

Add Container

Deploy on failure

Add post-build action

Save Apply

REST API Jenkins

=>

## Configuration

 General

 Source Code Management

 Build Triggers

 Build Environment

 Build Steps

 Post-build Actions

### Deploy war/ear to a container

WAR/EAR files 

\*\*/\*.war

Context path 

myapp

### Containers

#### Tomcat 9.x Remote

##### Credentials

manager/\*\*\*\*\*\*\*\*

 Add

##### Filter

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Publish JUnit test result report
- Record Fingerprints of files to track usage
- Git Publisher
- Deploy war/ear to a container
- E-mail Notification
- Editable Email Notification
- Set GitHub commit status (universal)
- Set build status on GitHub commit [deprecated]
- Slack Notifications**
- Delete workspace when build is done

 Add post-build action ▾

Select every notification you want to receive on your Slack channel:

The screenshot shows the Jenkins configuration page for a job named 'maven-deployment'. The left sidebar has sections like General, Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. 'Post-build Actions' is selected and highlighted with a grey background. In the main panel, there's a 'Configuration' section with fields for 'http://localhost:8081', 'Advanced...', 'Add Container', and a 'Deploy on failure' checkbox. Below these is a 'Slack Notifications' section, which is also highlighted with a red box. It contains seven checkboxes, all of which are checked: 'Notify Build Start', 'Notify Success', 'Notify Aborted', 'Notify Not Built', 'Notify Unstable', 'Notify Regression', and 'Notify Every Failure'. At the bottom of the configuration panel are 'Advanced...', 'Add post-build action ▾', 'Save' (which is highlighted with a red box), and 'Apply' buttons.

Save your configuration.

## Resources

### Documentation

- Jenkins Slack Plugin
- Mark task as complete  
Report issue

## 11. Test the Slack Notifications

Test the Jenkins-Slack integration for:

- builds that failed
- builds that succeeded

Notifications must be sent to your Slack channel

1. Log in to Jenkins as the user that can execute the Jenkins tasks (E.g. as `labsuser`). Select your project (**maven-deployment**), and build it.

The screenshot shows the Jenkins dashboard interface. At the top, there is a dark header with the Jenkins logo and the word "Jenkins". Below the header, the word "Dashboard" is followed by a right-pointing arrow. On the left side, there is a vertical sidebar with several options: "+ New Item", "People", "Build History", "Project Relationship", "Check File Fingerprint", "Manage Jenkins", "My Views", and "Lockable Resources". To the right of the sidebar, there is a main content area. At the top of this area, there is a button labeled "All" with a "+" sign next to it. Below this, there is a table-like structure with columns labeled "S", "W", and "Name ↓". A red box highlights the first row of this table, which contains a green checkmark icon, a blue Jenkins icon, and the text "maven-deployment". At the bottom of the main content area, there is a small "Icon:" label followed by three letters: "S", "M", and "L", with "L" being highlighted in a light gray box.

 Status

</> Changes

 Workspace

 Build Now

 Configure

 Delete Project

 Git Polling Log

 Rename

2. Wait until the build is finished and check your Slack channel. You will have a notification indicating the job was finished successfully.



**jenkins** APP 5:25 PM

maven-deployment - #16 Started by user LabsUser ([Open](#))

maven-deployment - #16 Success after 41 sec ([Open](#))

3. Go to your forked repository file `src/main/java/com/course/tomcat/controller/HelloController.java` and remove a "," character from your code that is in the `master` branch. This will cause a build error later.

The screenshot shows a GitHub code editor interface. At the top, there are navigation links: Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below these, a dropdown menu shows "master" with a red box around it, and the path "course-jenkins-tomcat-maven / src / main / java / com / course / tomcat / controller / HelloController.java". To the right of the path is a "Jump to" dropdown and a "Go to file" button. A red box highlights the file name "HelloController.java". On the far right, there's a "History" button. Below the header, a message says "Latest commit e4a04e7 16 minutes ago". The main area displays the code for HelloController.java:

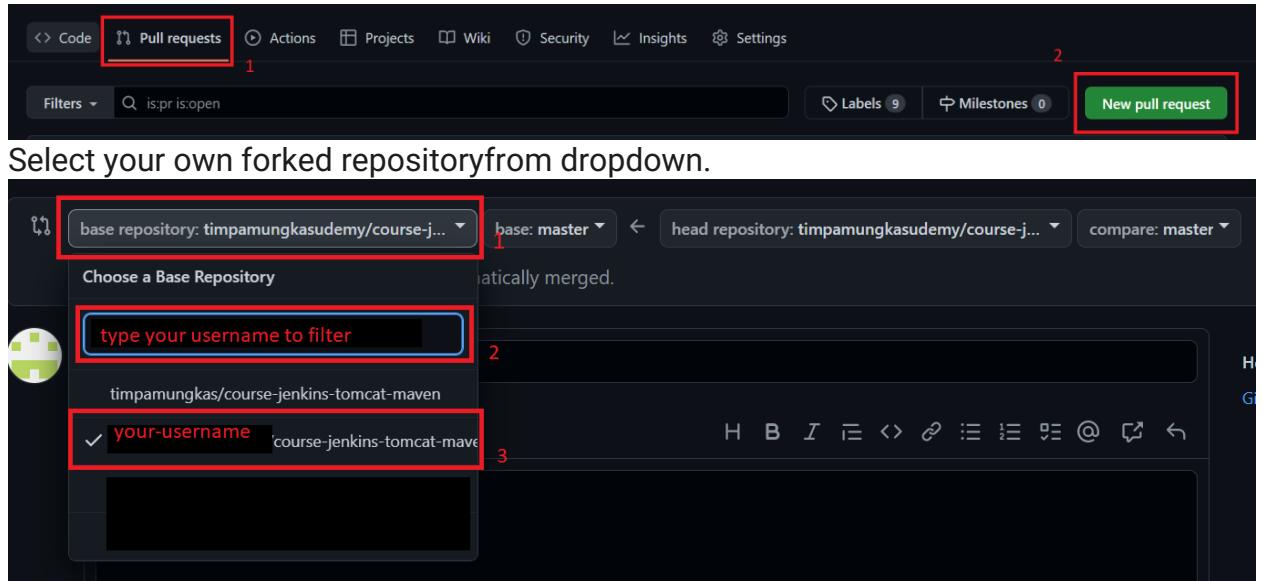
```
1 package com.course.tomcat.controller;
2
3 import org.springframework.web.bind.annotation.*;
4
5 import java.time.LocalDateTime;
6 import java.util.concurrent.ThreadLocalRandom;
```

The screenshot shows a GitHub commit changes dialog. At the top, it says "course-jenkins-tomcat-maven / src / main / java / com / course / tomcat / controller / HelloController.java in master". There are "Edit file" and "Preview changes" buttons. On the right, there are "Cancel changes", "Spaces", "4", and "No wrap" buttons. The main area shows the Java code for HelloController.java with some changes highlighted by red boxes. The last line of code is highlighted with a red box:

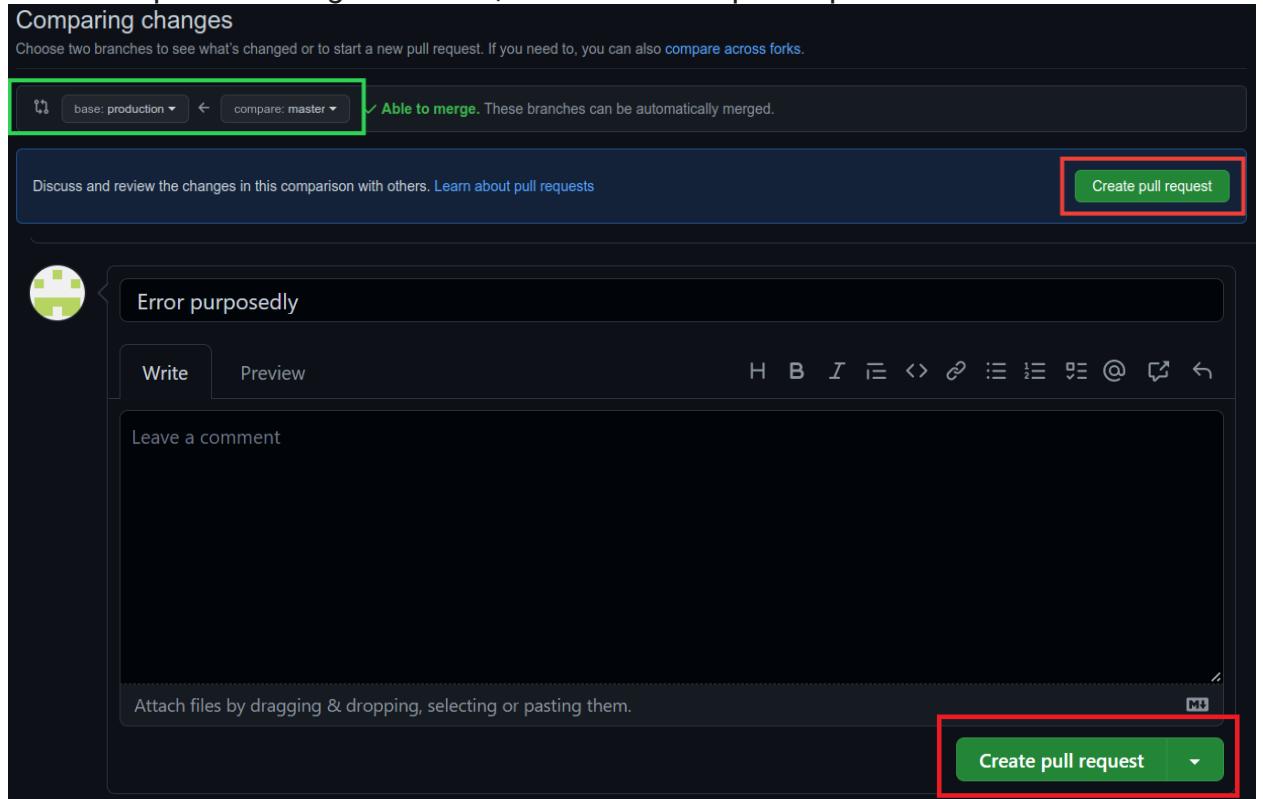
```
1 package com.course.tomcat.controller;
2
3 import org.springframework.web.bind.annotation.*;
4
5 import java.time.LocalDateTime;
6 import java.util.concurrent.ThreadLocalRandom;
7
8 @RestController
9 @RequestMapping("/api")
10 public class HelloController {
11
12     @GetMapping(value = {"", "/", "/now"})
13     String hello() {
14         return "Now is " + LocalDateTime.now();
15     }
16
17     @GetMapping(value = "/random")
18     int random(){
19         return ThreadLocalRandom.current().nextInt(0,1000);
20     }
21
22     @GetMapping(value = "/hero")
23     String hero(){
24         return "My favorite superhero is Batman"
25     }
26
27 }
```

At the bottom, there's a "Commit changes" dialog. It has a "Commit changes" button at the bottom left, which is also highlighted with a red box. The dialog includes fields for a commit message ("feat: error testing for slack notification") and an optional extended description, and radio buttons for committing directly to the master branch or creating a new branch.

4. Create a Pull Request from the `master` branch to the `production` branch.



Select your own forked repository from dropdown.



5. Approve the created Pull Request, then merge it.

Add more commits by pushing to the `master` branch on [https://github.com/mpandugalang/course-jenkins-tomcat-maven](#).

This branch has no conflicts with the base branch  
Merging can be performed automatically.

Squash and merge ▾ or view command line instructions.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close pull request Comment

Add more commits by pushing to the `master` branch on [https://github.com/mpandugalang/course-jenkins-tomcat-maven](#).

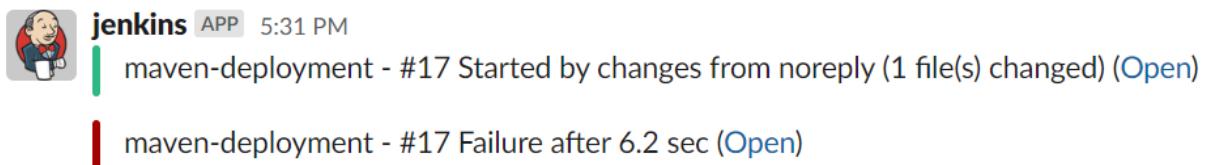
feat: error testing for slack notification (#15)

Add an optional extended description...

This commit will be authored by mpandugalang@gmail.com

Confirm squash and merge Cancel

6. Wait until your Jenkins scheduler runs and it now will display an error message. Check Slack for the notification. The notification will still be sent although the build failed.



Mark task as complete

Report issue