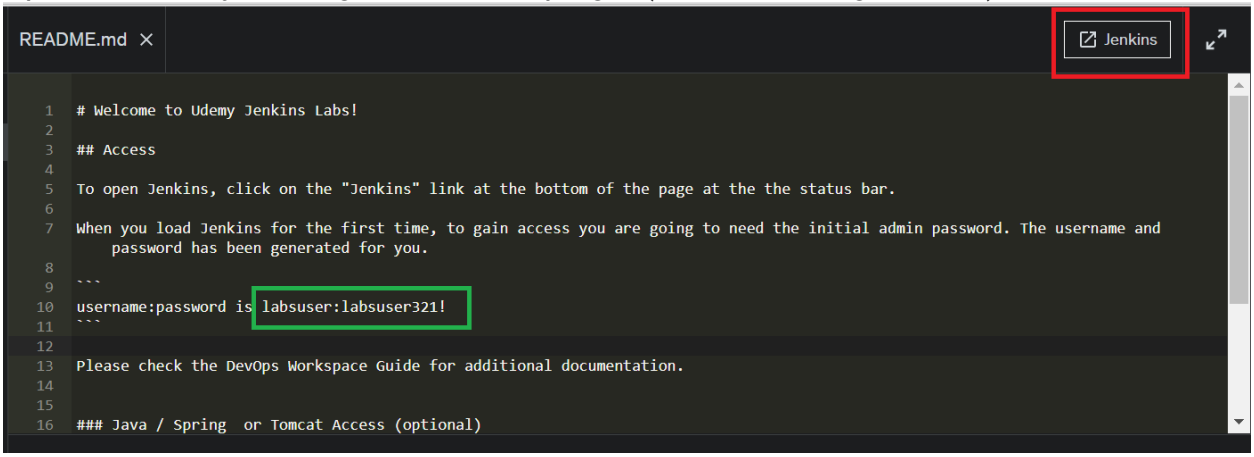
Tasks
0/11
1. Add Maven to Jenkins

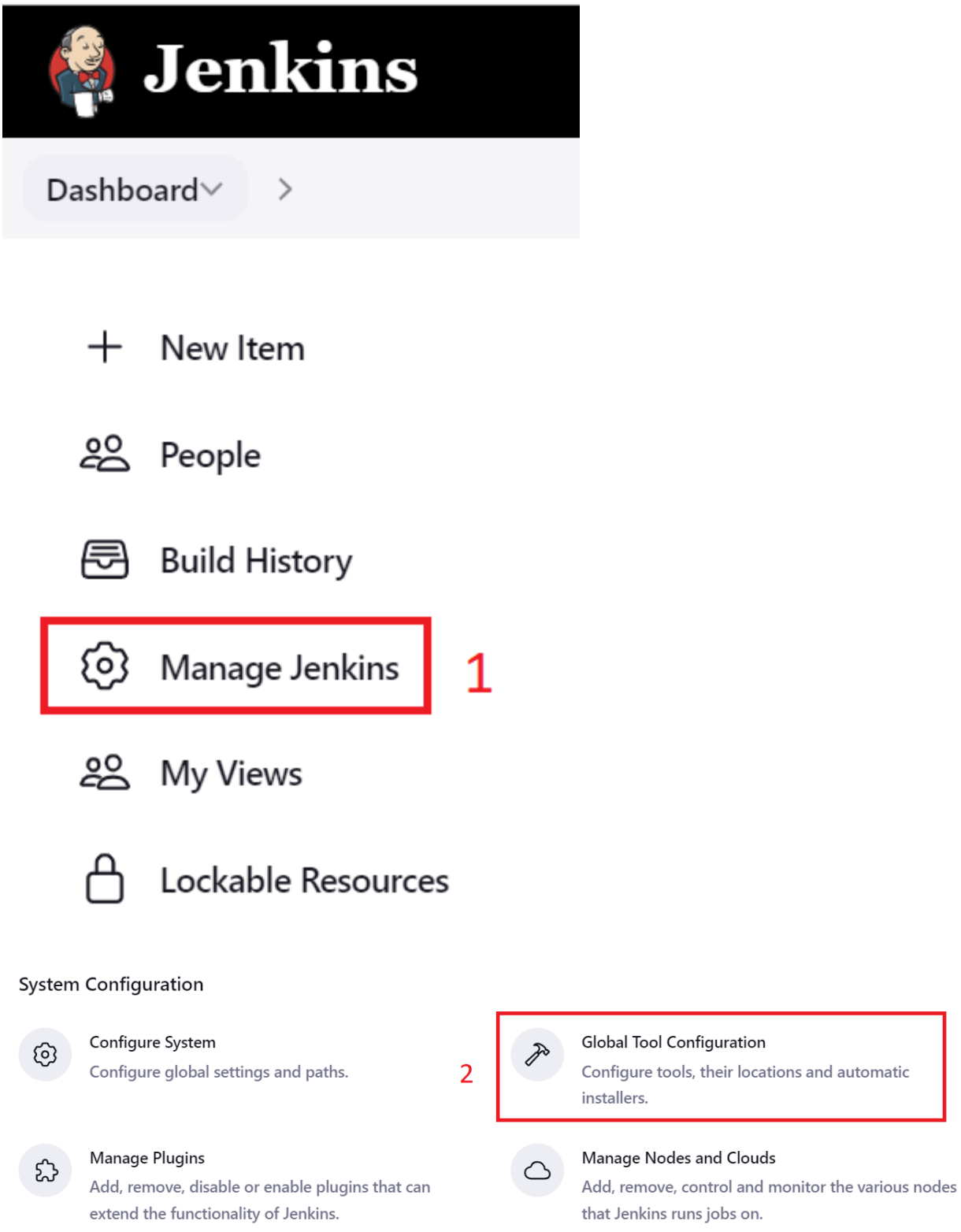
Log in to Jenkins and make sure it has the Maven installation.

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



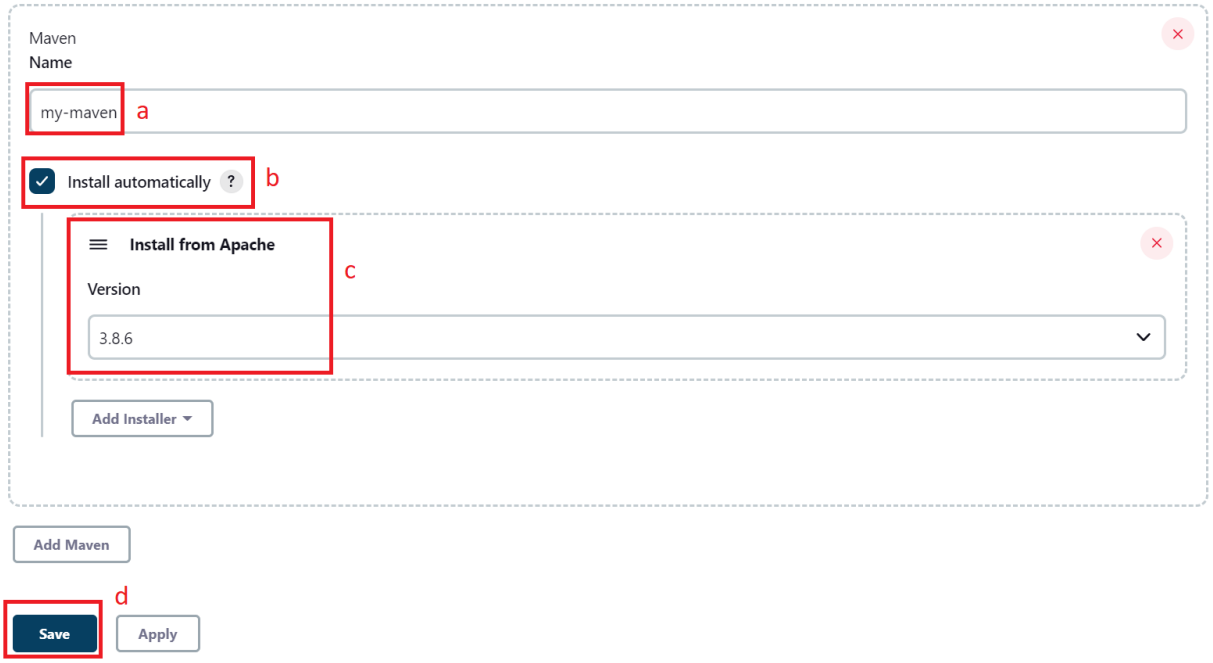
2. Login to Jenkins using the provided username / password (green rectangle above)

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)



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

5. Installation details:
    a. Name: `my-maven`
    b. Select the checkbox *Install automatically*
    c. Install the latest version from Apache
    d. Save the configuration



Resources

Documentation

- Build Java Maven Application Using Jenkins

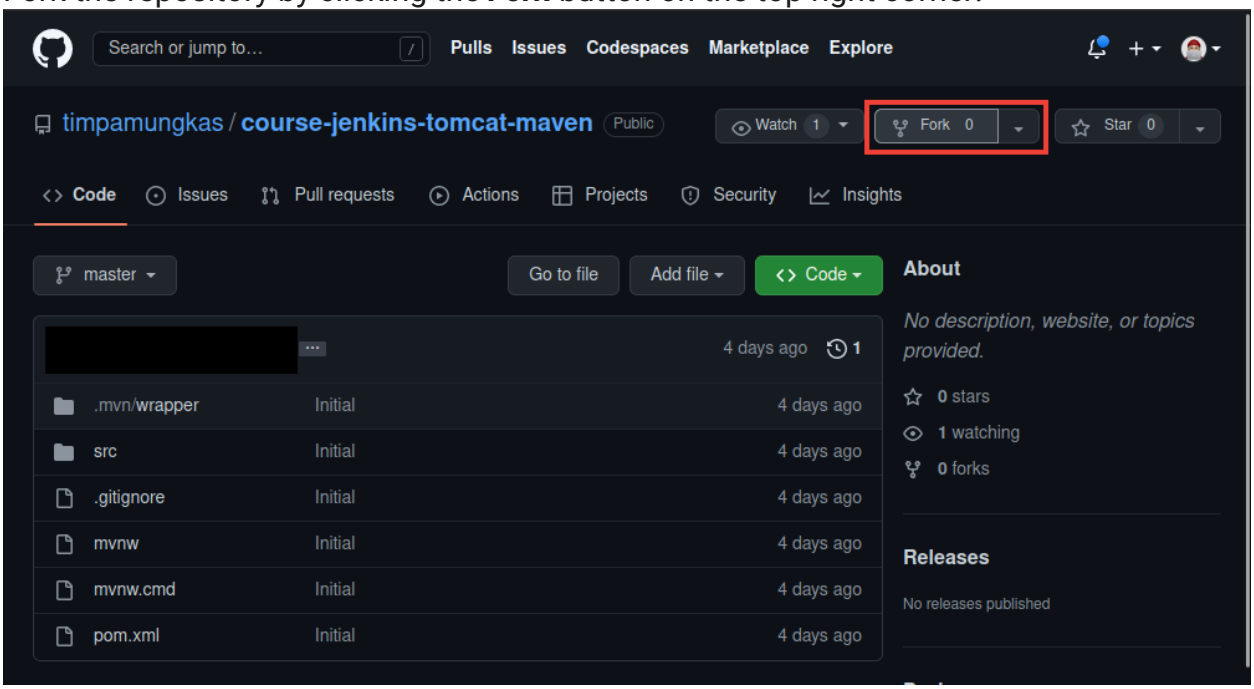
Mark task as complete
Report issue

2. Configure Credentials for the Tomcat Deployment

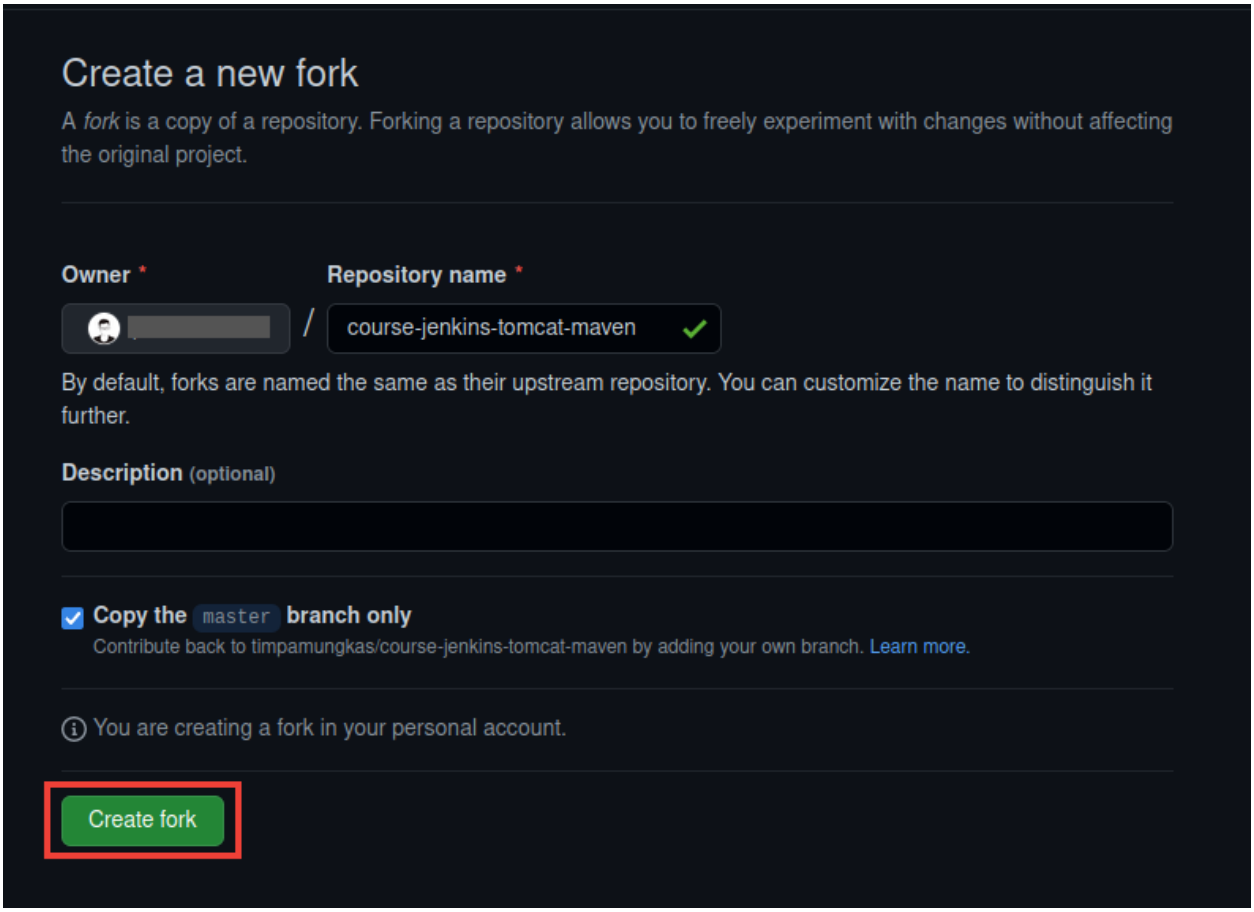
3. Initialize the Java Source Code

Fork this Github repository to your own Github account.

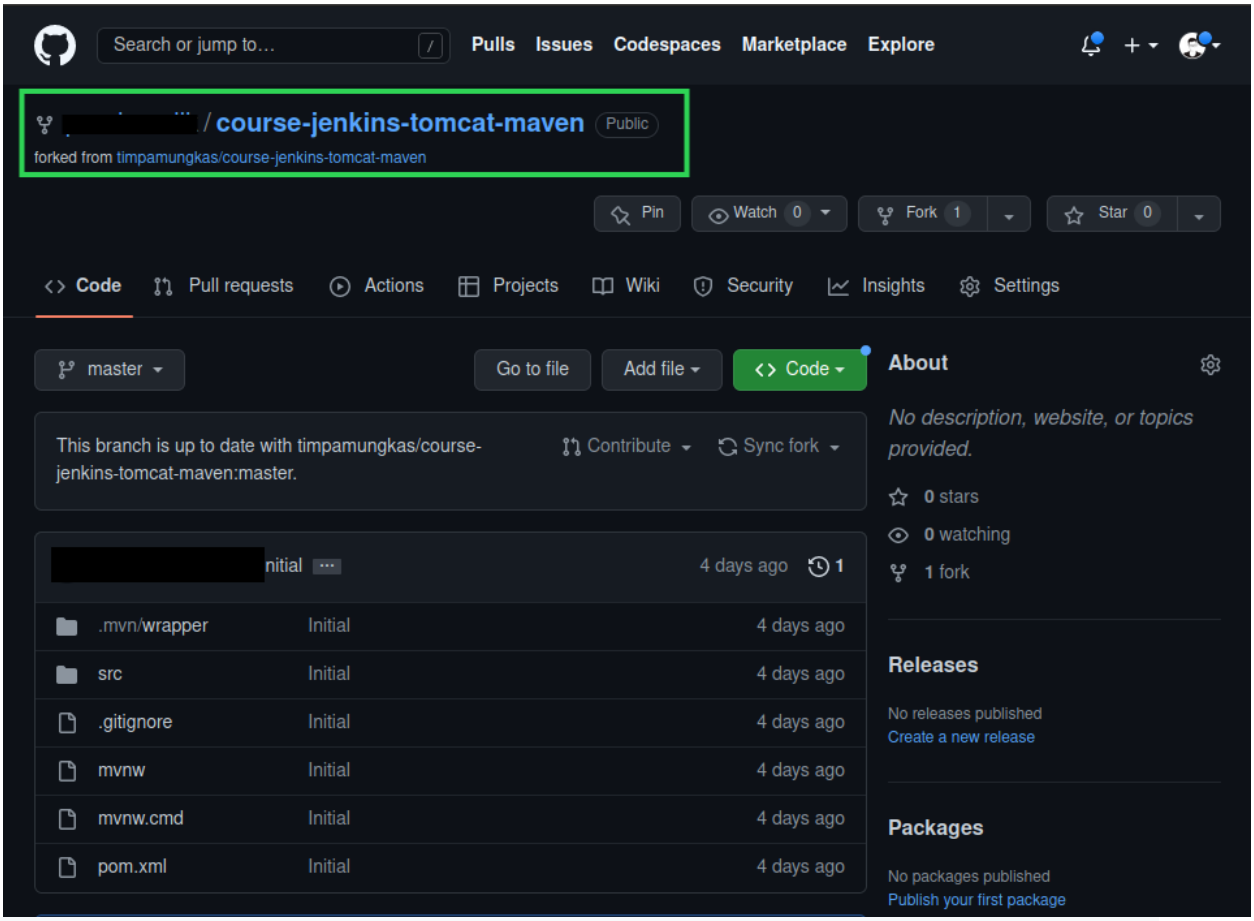
1. Create a Github account using your email if you do not have one. Log in if you already have an account.
2. Go to this repository.
3. Fork the repository by clicking the ***Fork*** button on the top right corner.



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



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

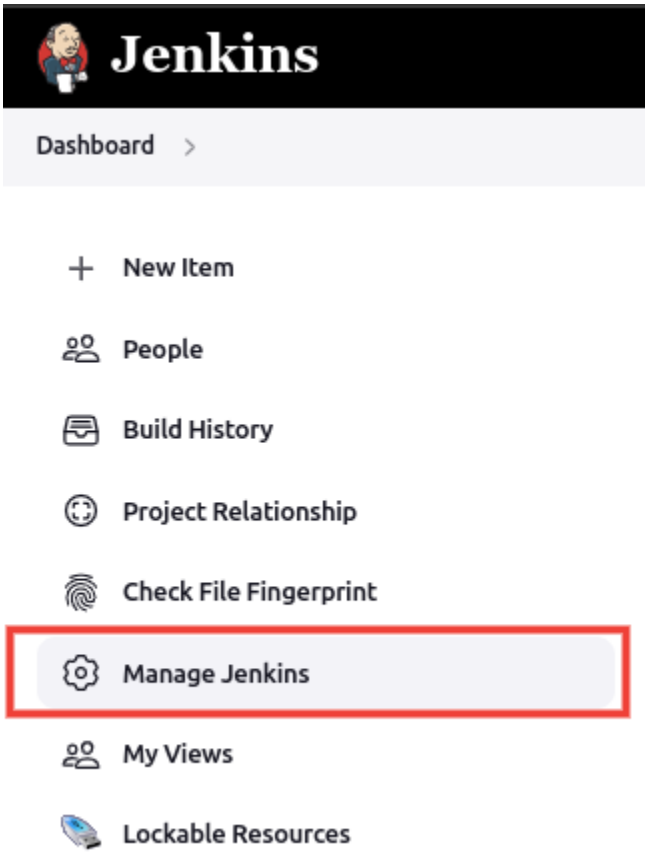
Mark task as complete
Report issue

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

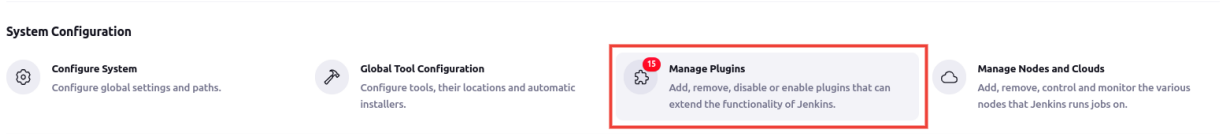
Install the Jenkins plugin with the name: ***Deploy To Container***

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

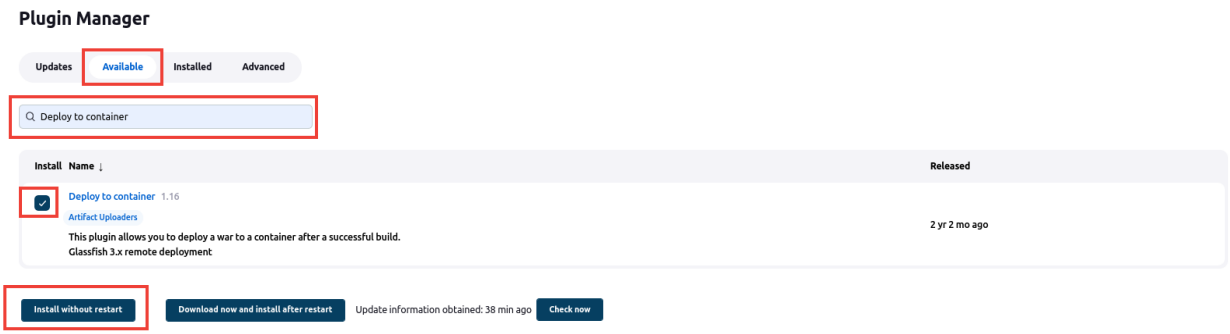
1. From the Jenkins home screen, go to ***Manage Jenkins*** :



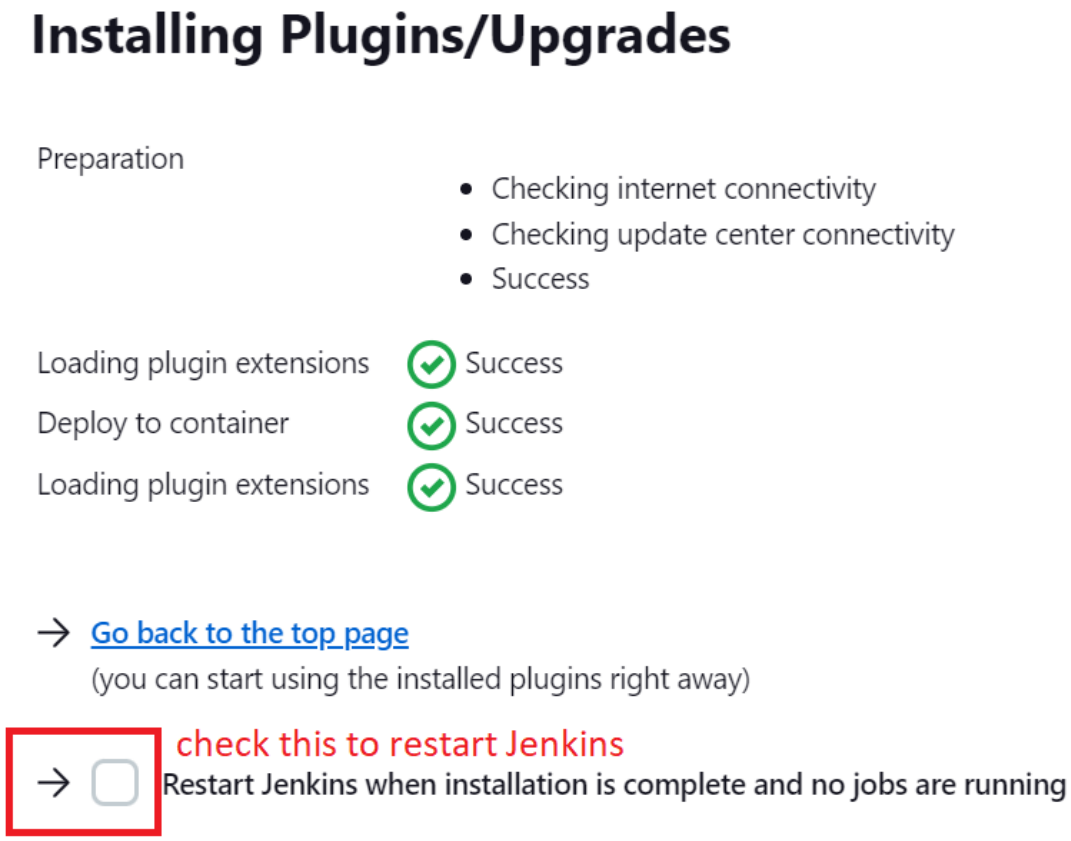
Open ***Manage Plugins***



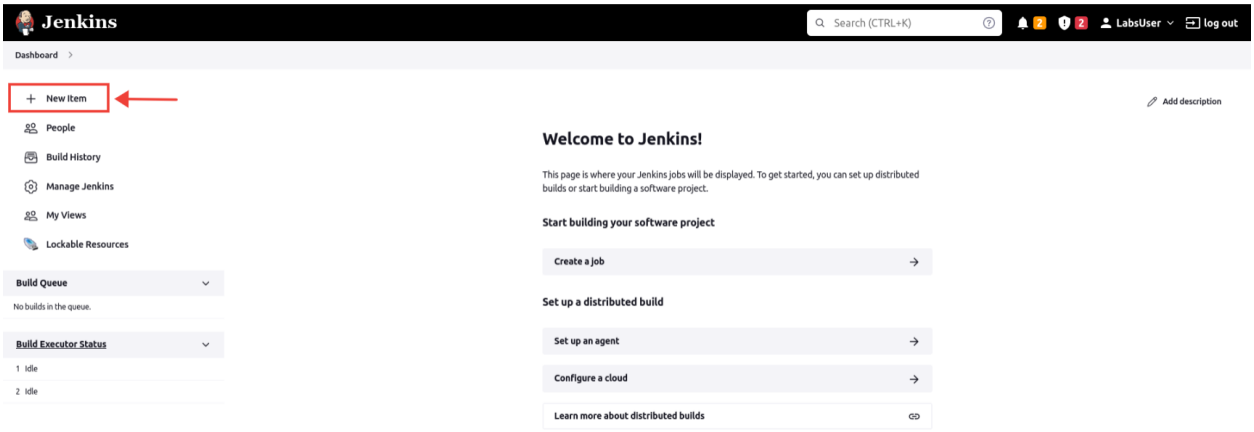
Search for the ***Deploy to Container*** plugin under the ***Available*** tab. This plugin allows you to deploy an application into the defined container (server).
Select ***Install it without restart***:



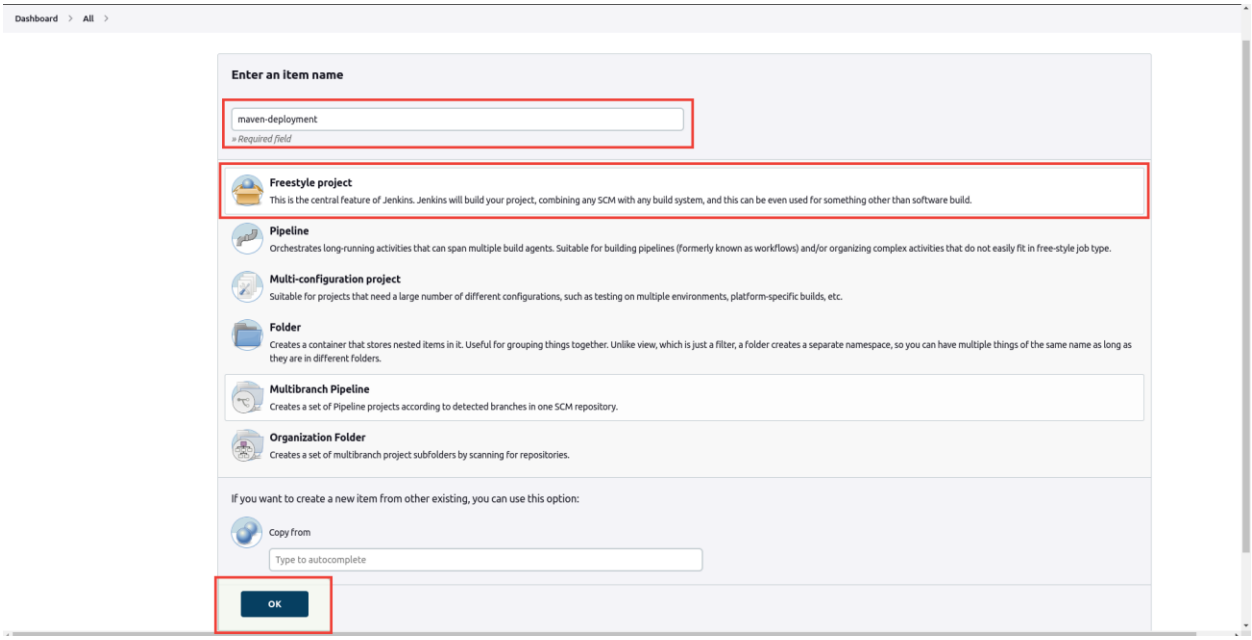
2. When installation is done, restart Jenkins by checking the checkbox shown below.



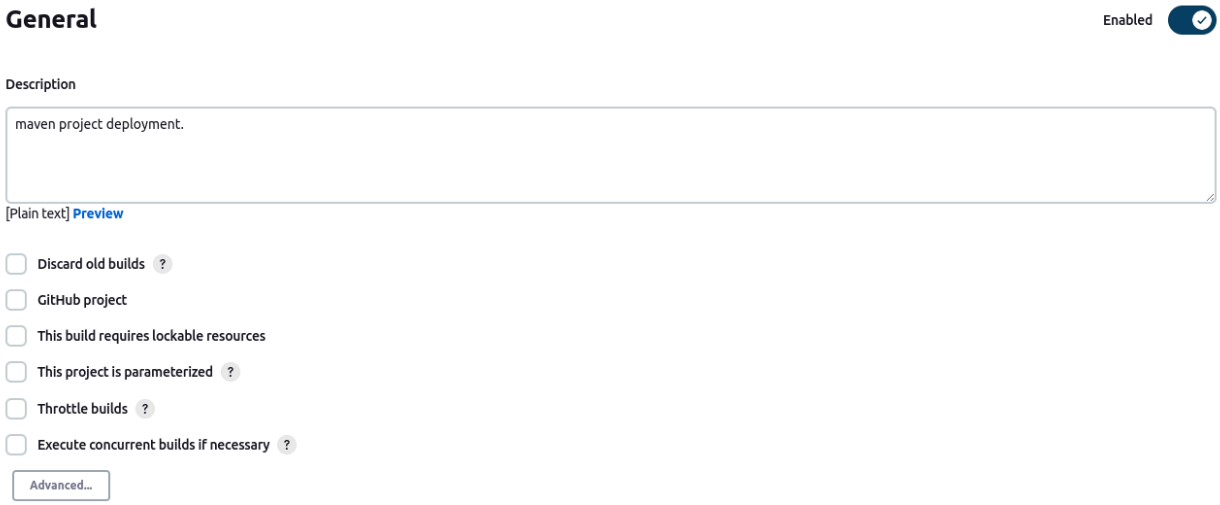
3. Wait until Jenkins restarts. Refresh your browser if Jenkins does not restart automatically after 2 minutes. Go back to the Jenkins home screen, and select ***New Item***.



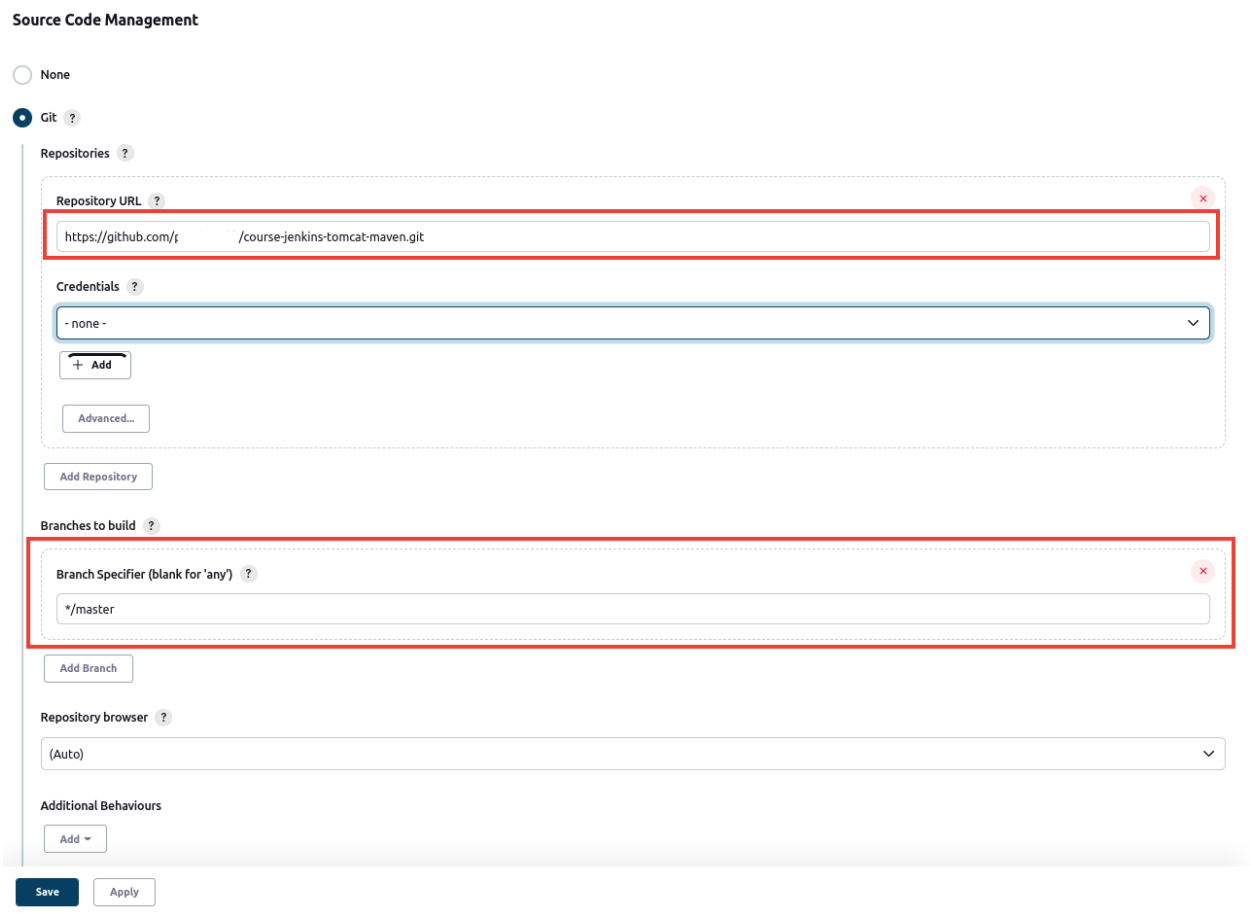
4. Create a Jenkins project using the ***Freestyle project*** type. Give a name to your project, and then click ***OK.***



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

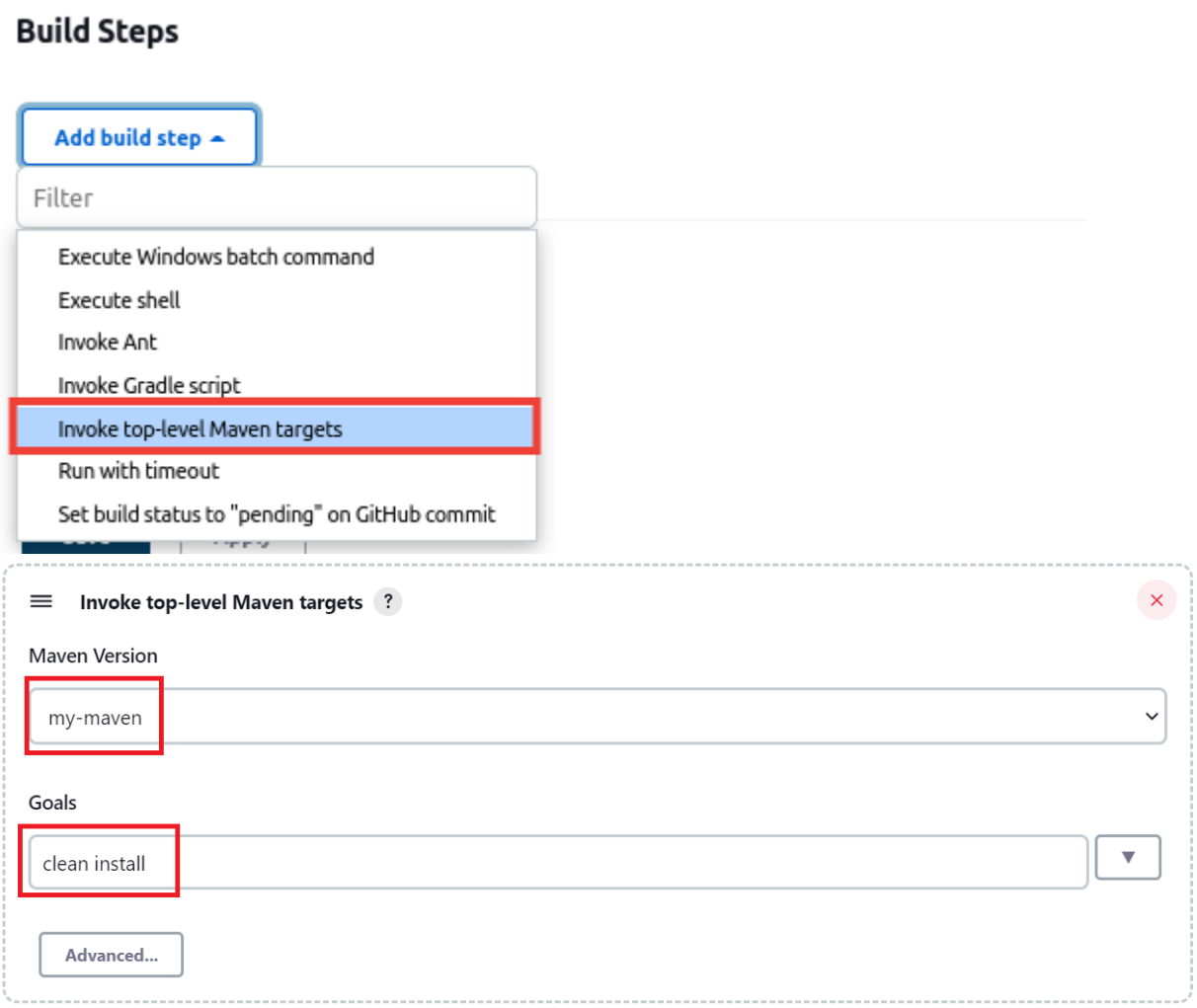


5. In the ***Source Code Management*** section, choose ***Git*** and then fill in the details of your repository:
    a. ***Repository url*** :URL of your forked repository (from the task 3)
    b. ***Selected branch*** : `*/master`, meaning that you will build the application from the `master` branch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

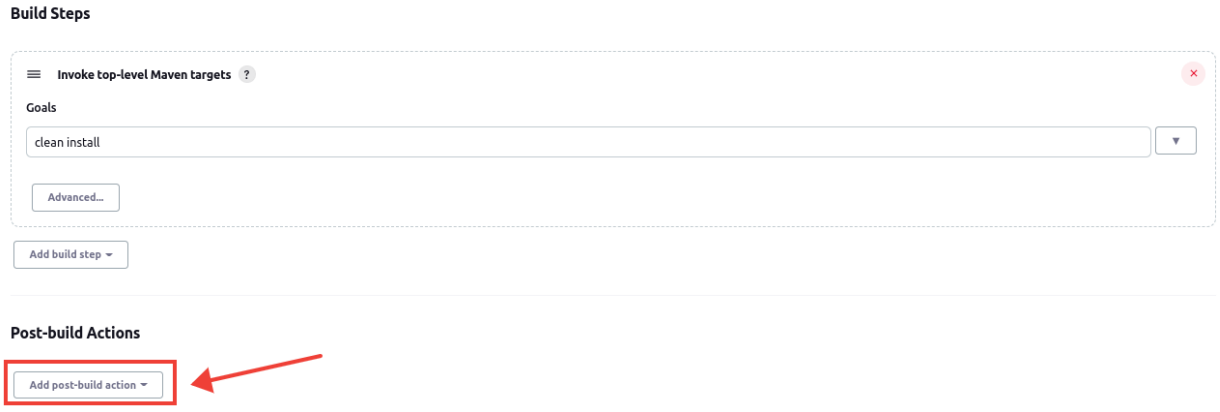


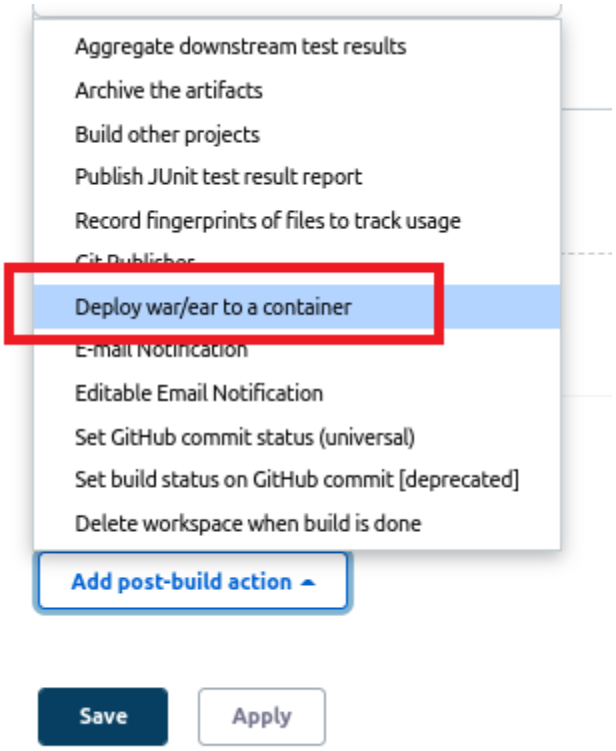
You don't need to fill in the ***credentials*** section since the repository is public.

6. In the ***Build Steps*** section, select ***Invoke top-level Mavens 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.

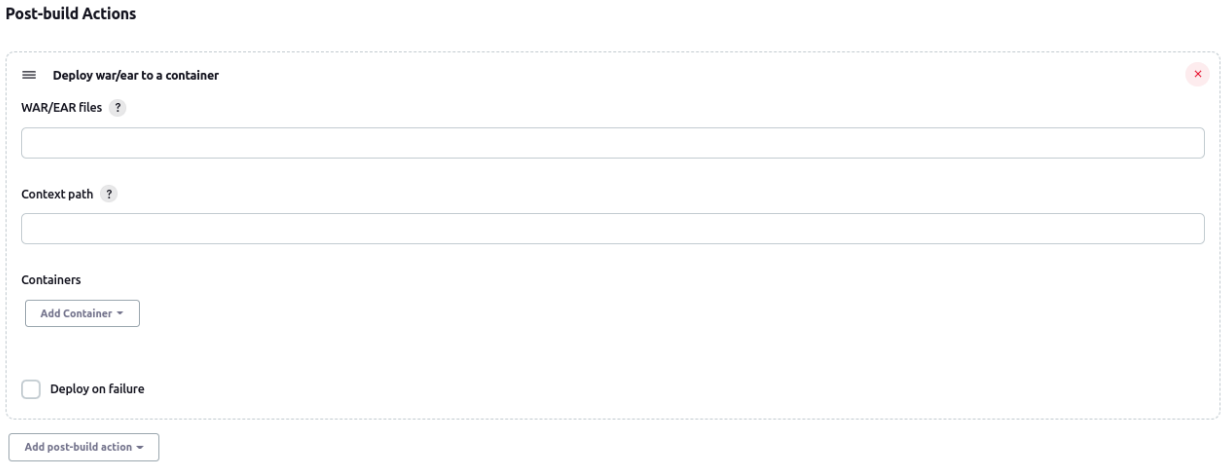


7. The built application will generate a Java war to be deployed. Select ***Deploy war/ear to a Container*** from the ***Post-Build Actions*** dropdown menu. Click **Save**.

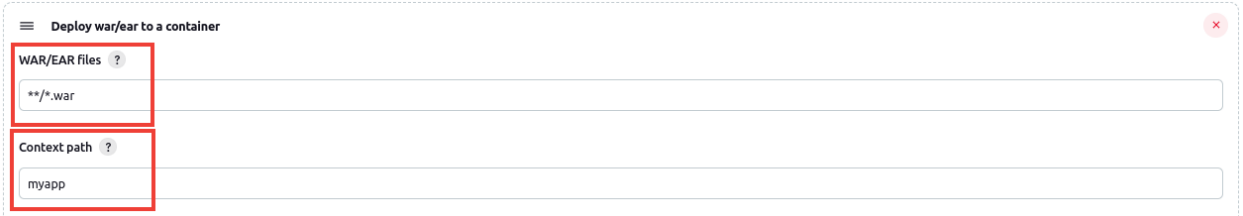




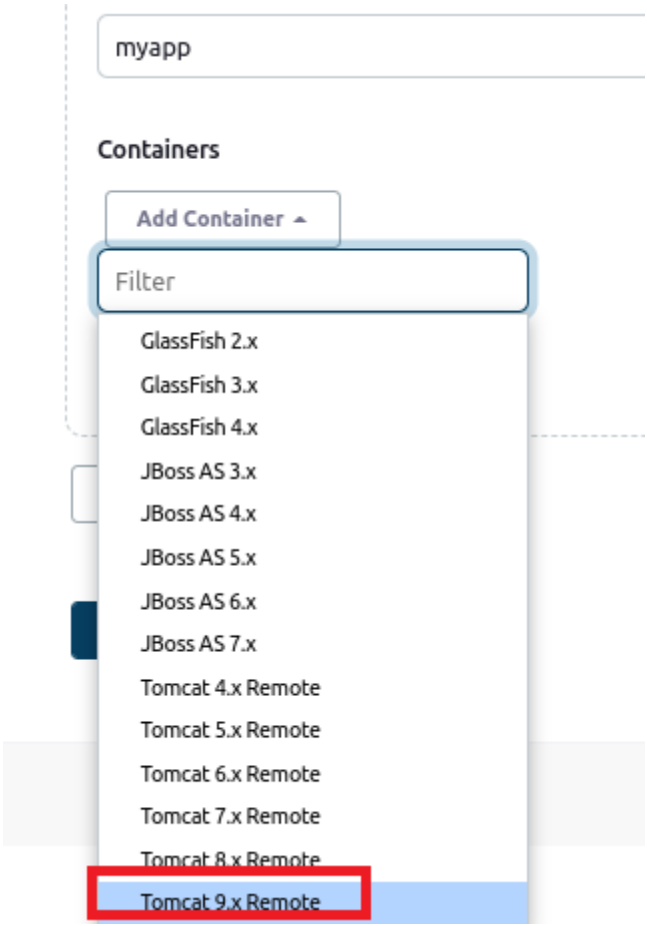
8. After selecting the post build actions, you will see the below screen.



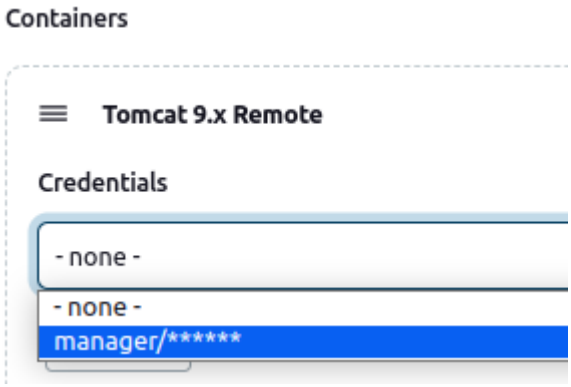
Fill in the details for the ***WAR/EAR files***: `**/*.war` and ***Context Path***: `myapp`



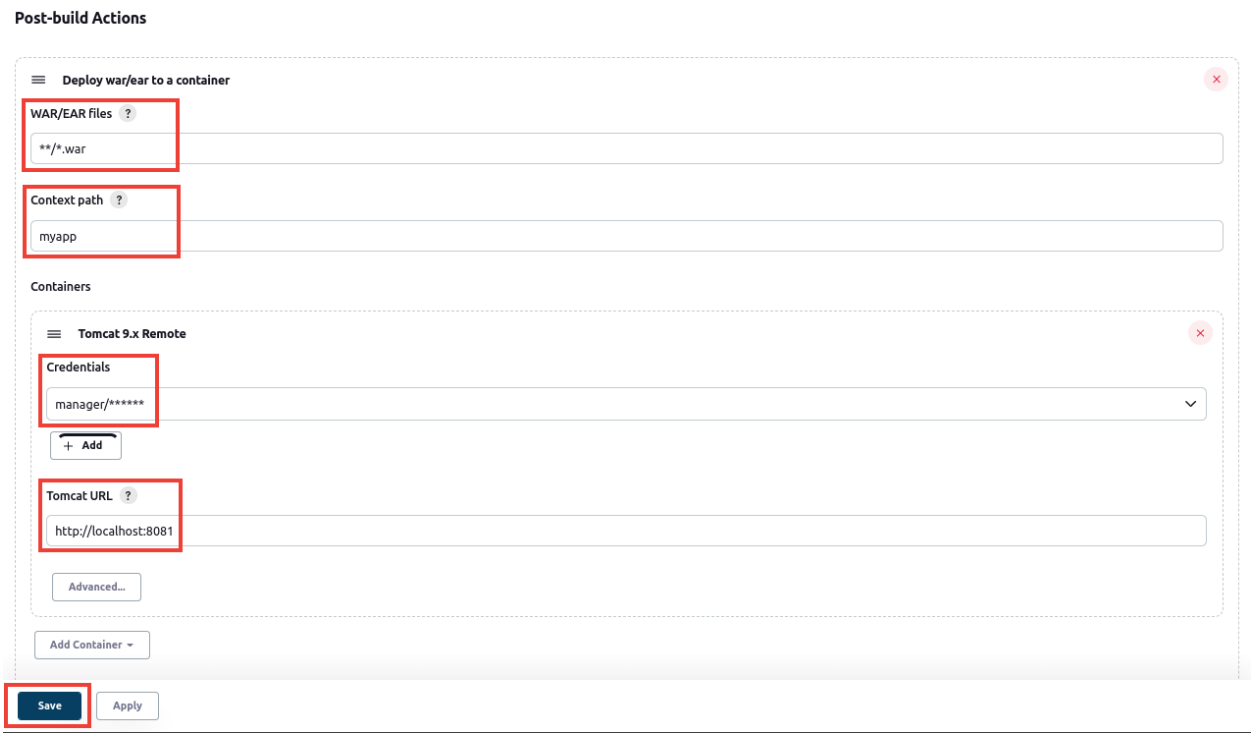
9. In the **Container** sub section, select the version of your Tomcat server (Version 9)



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:



Save your project to finish your configurations.

Resources

Documentation

- Deploy Java War to Tomcat
- Create Deployable War File

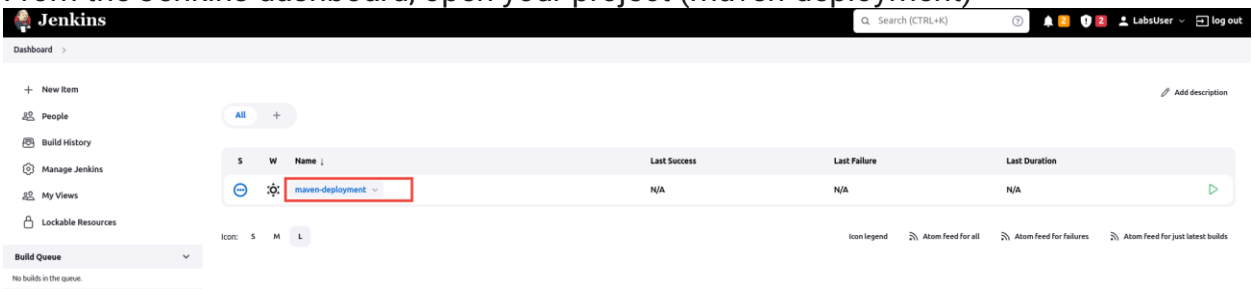
Mark task as complete

Report issue

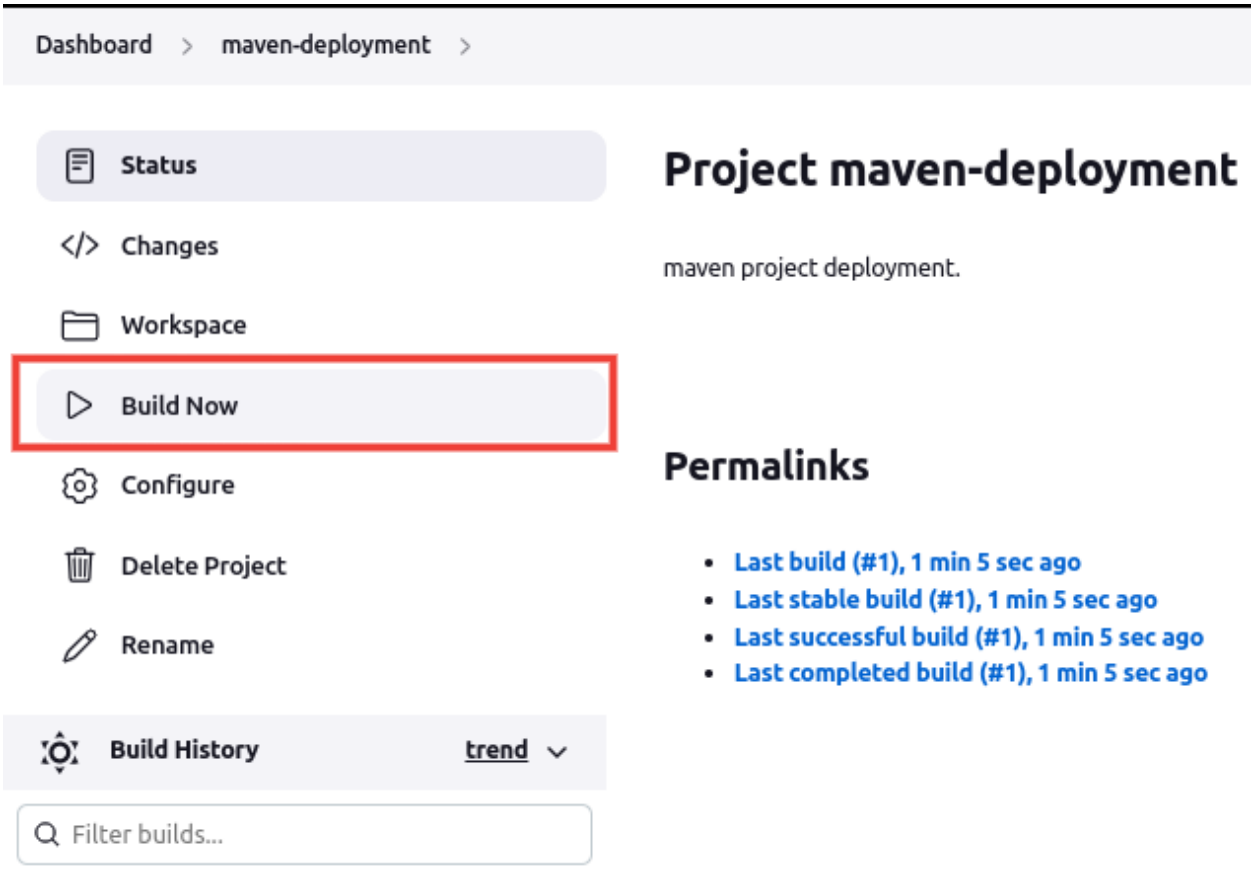
5. Test your Jenkins Project

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

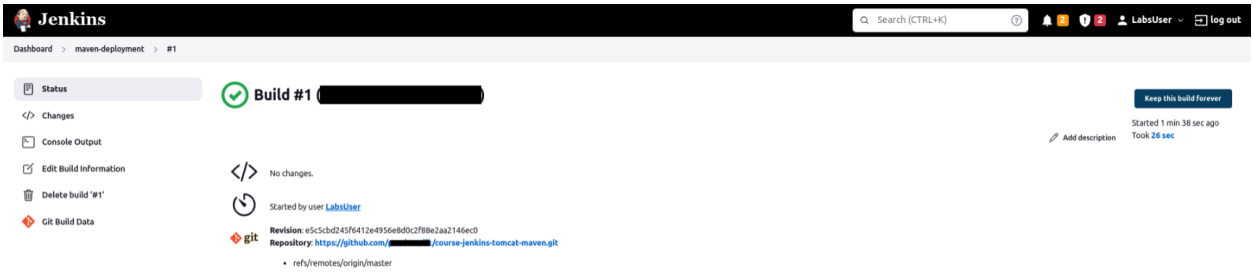
1. From the Jenkins dashboard, open your project (maven-deployment)



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

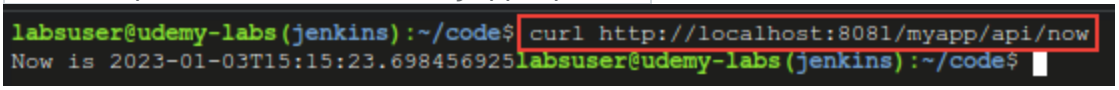


3. Wait until the build succeeds



4. 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
```



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

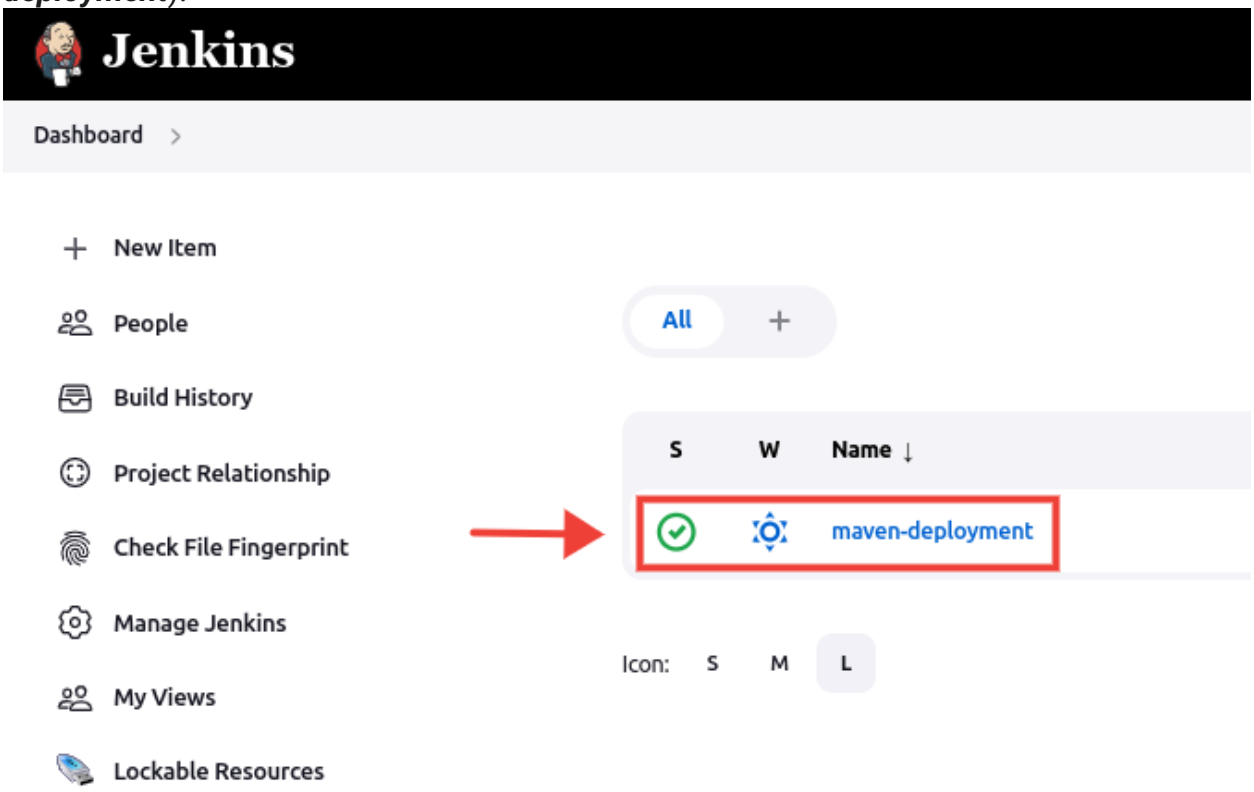
6. Schedule a Regular Build & Deploy

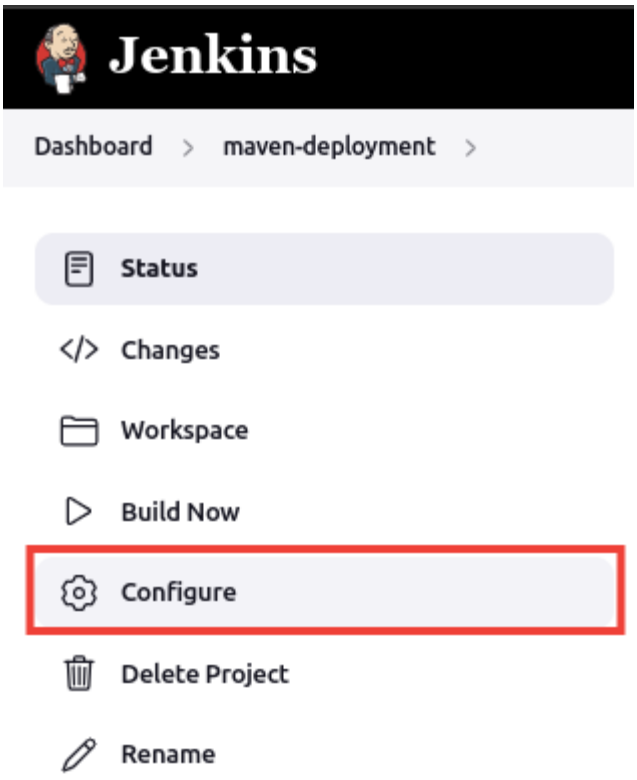
Configure a Jenkins task to automatically run every 5 minutes.

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`:

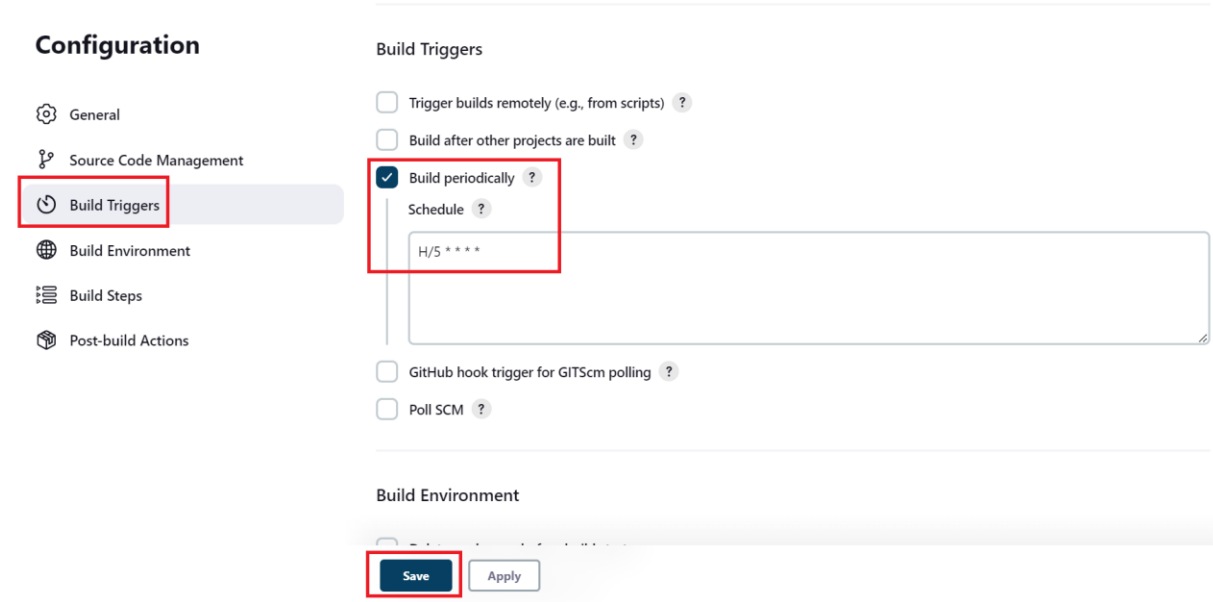
```
labsuser@udemy-labs(jenkins):~/code$ curl -i http://localhost:8081/myapp/api/random

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***):





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 * * * *`:

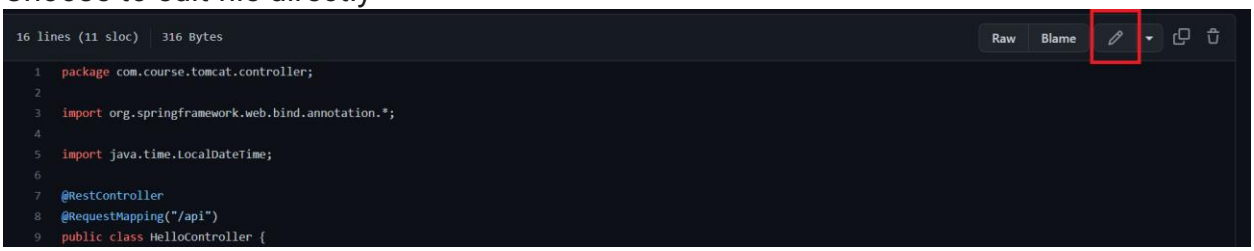


Save your configuration by clicking the **Save** button on the bottom of the page.

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:

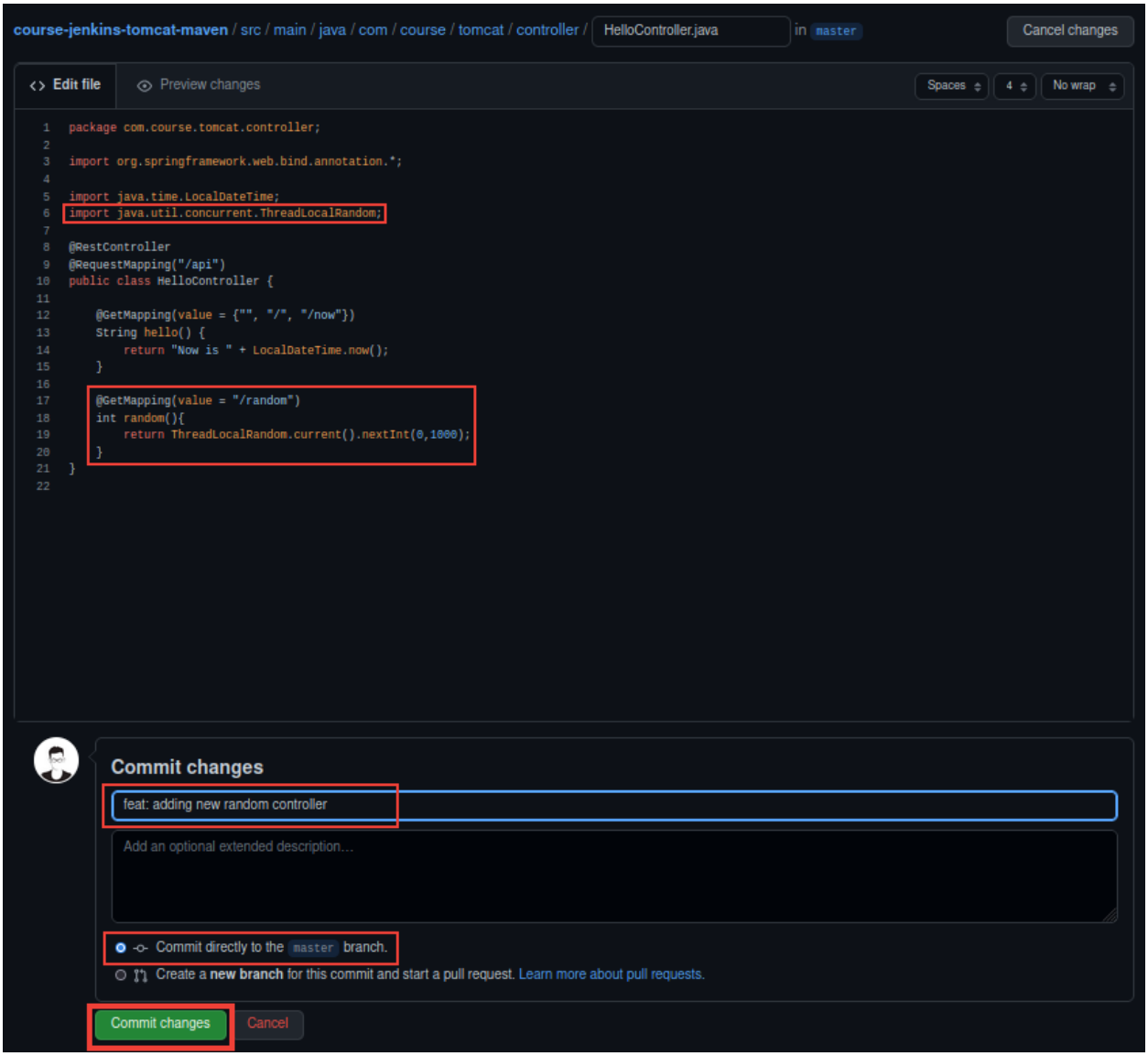
```java
package com.course.tomcat.controller;

import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.concurrent.ThreadLocalRandom;

@RestController
@RequestMapping("/api")
public class HelloController {

    @GetMapping(value = {"", "/", "/now"})
    String hello() {
        return "Now is " + LocalDateTime.now();
    }

    @GetMapping(value = "/random")
    int random(){
        return ThreadLocalRandom.current().nextInt(0,1000);
    }
}
```

Choose to edit file directly



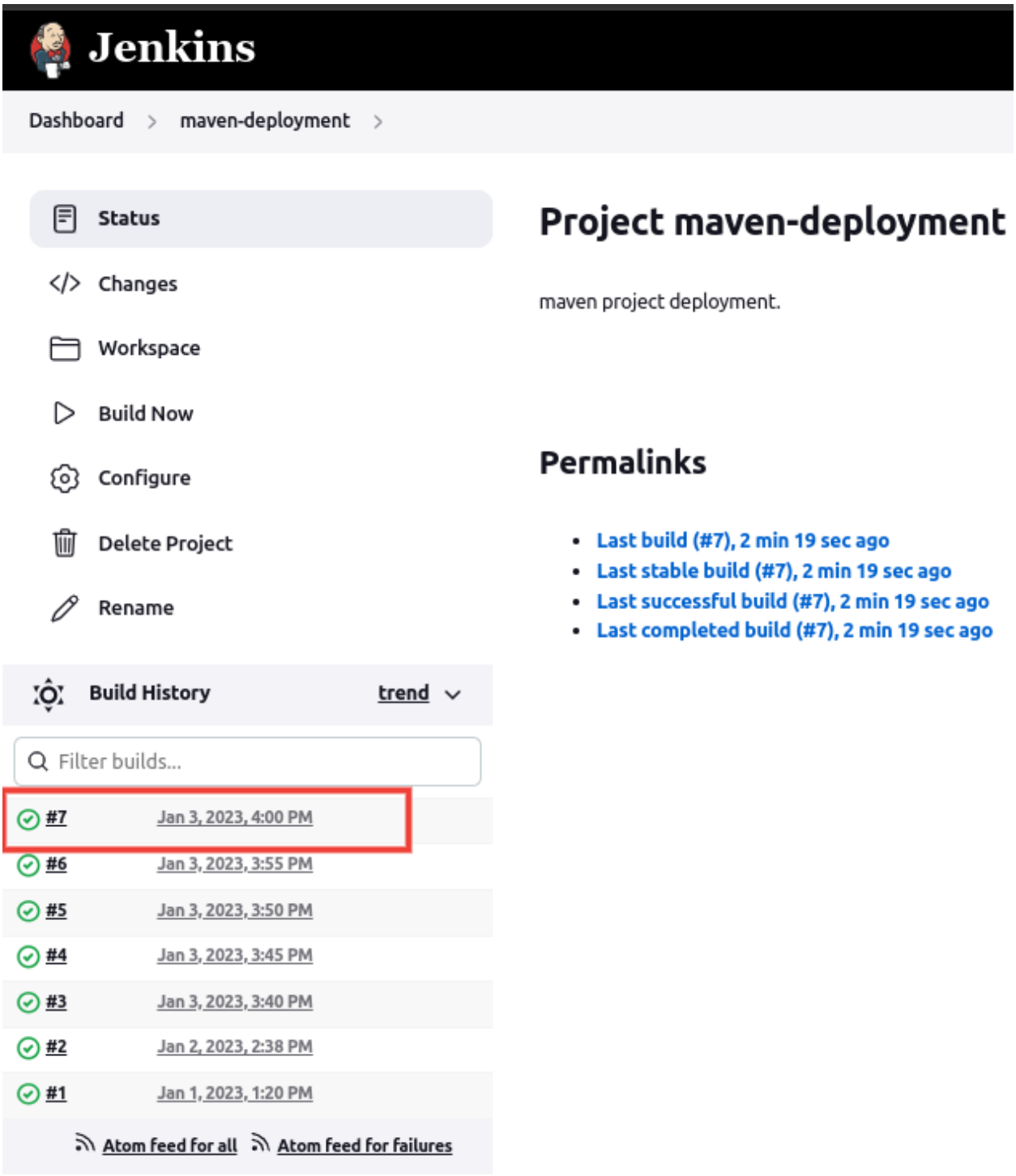
Copy paste the source code above

Commit changes directly to the `master` branch

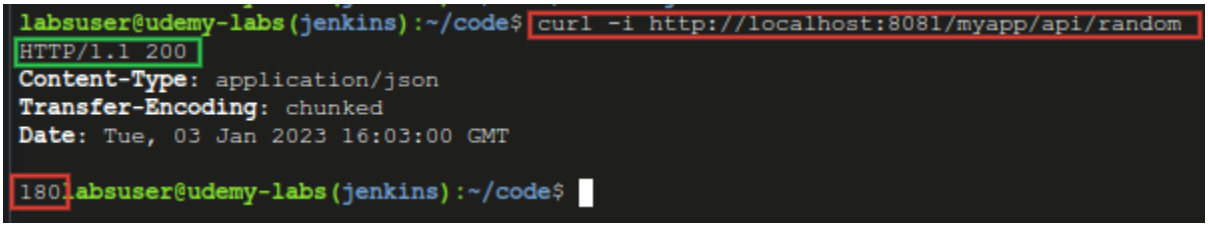


Commit your change directly to the `master` branch of your forked repository

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



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`



You will get a random number as the result.

Resources

Documentation

- How to use cron

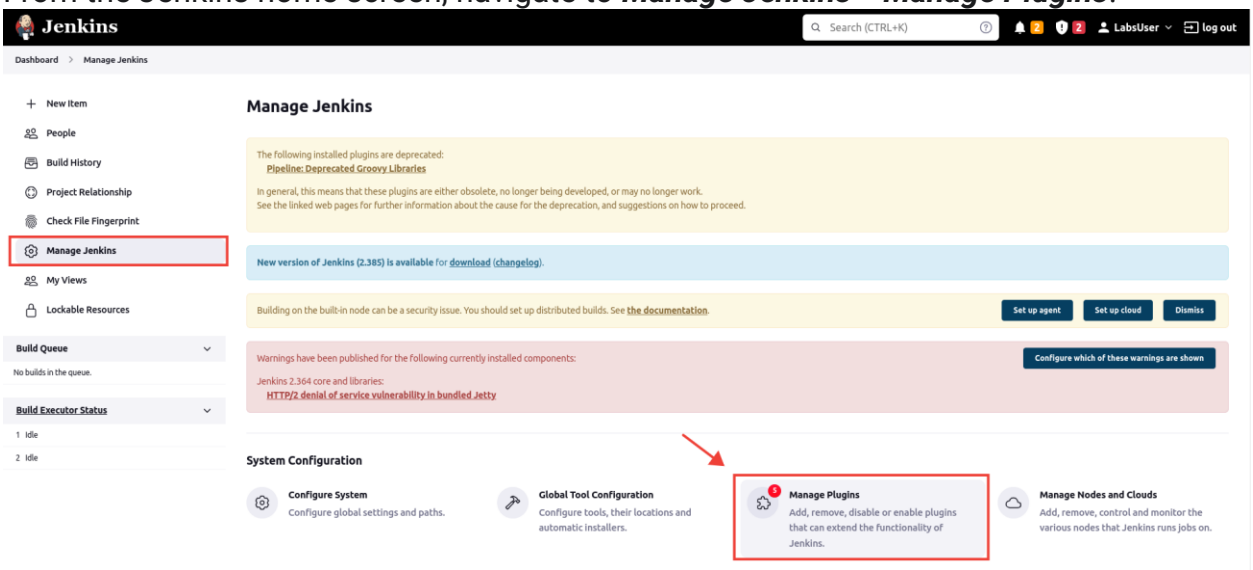
Mark task as complete
Report issue

7. Create Users for the Jenkins Task

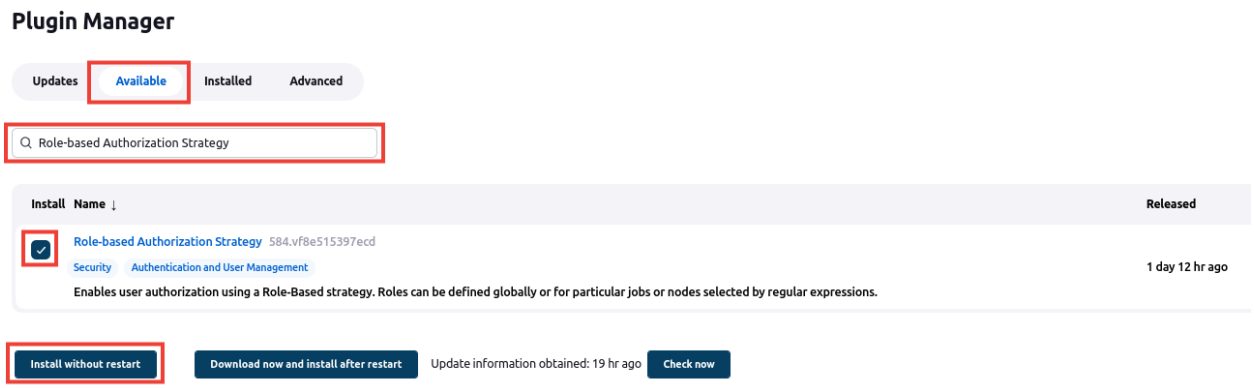
Create two Jenkins users with different permissions:

- First user can only view the Jenkins project and it's 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***.

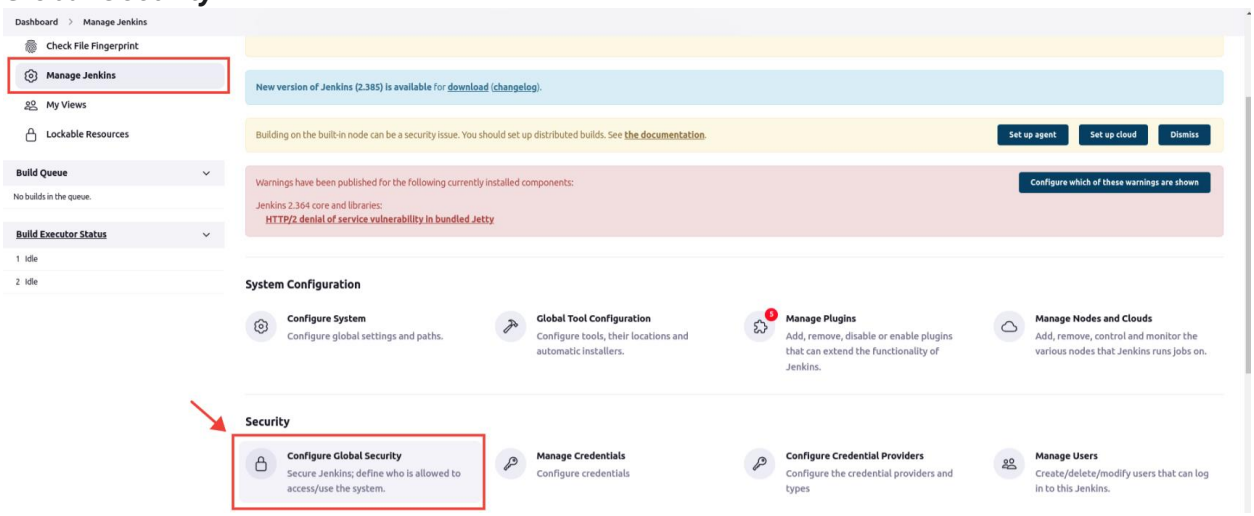


Under the ***Available*** tab, search for ***Role-based Authorization Strategy***. This plugin allows you to manage users, giving each user specific access permissions.

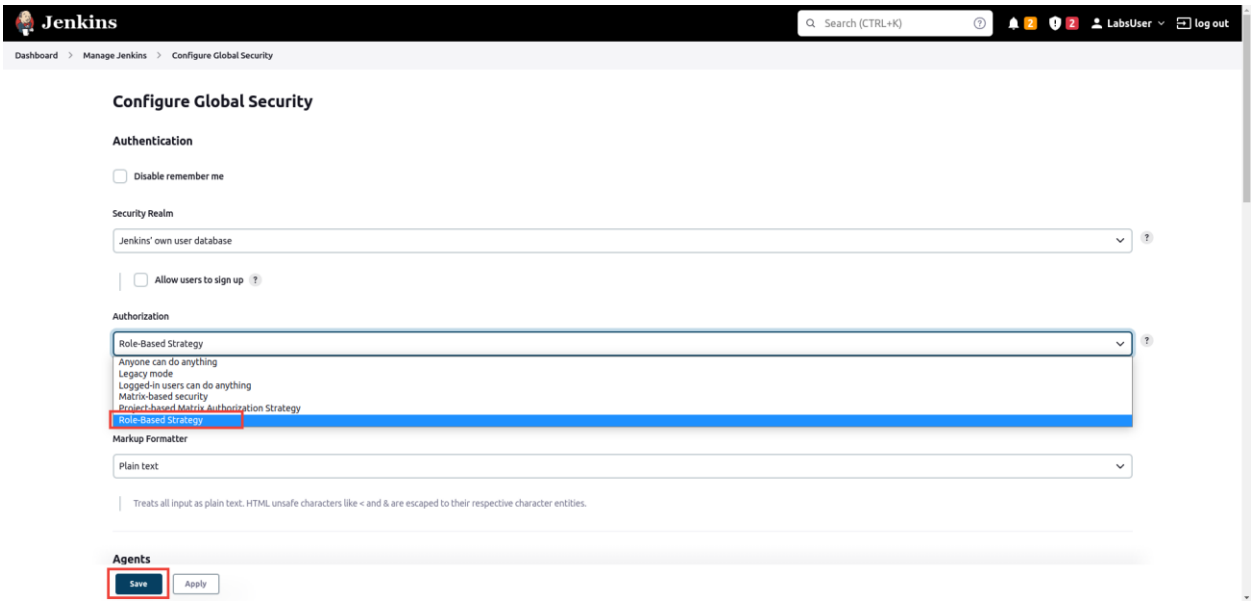


Choose ***Install without restart***

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

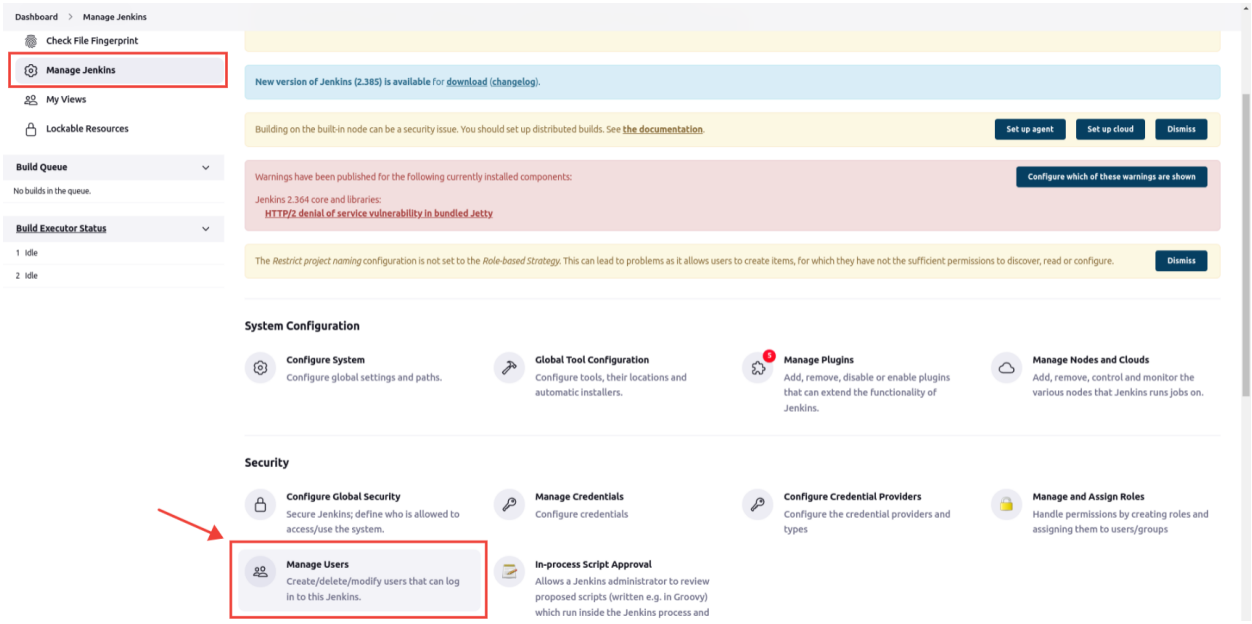


And then select ***Role-Based Strategy*** from the ***Authorization*** dropdown menu:

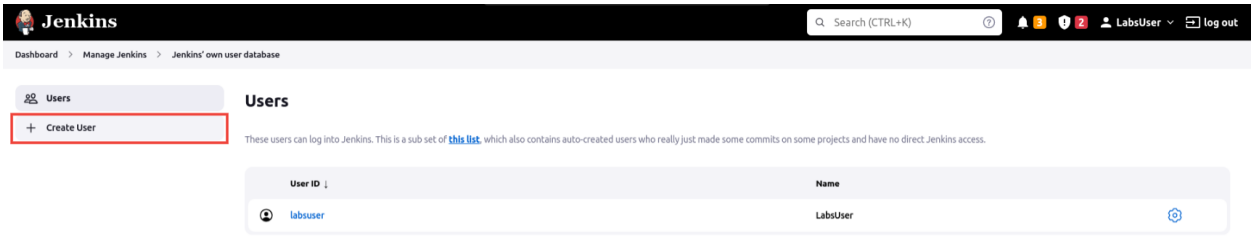


Save the configuration.

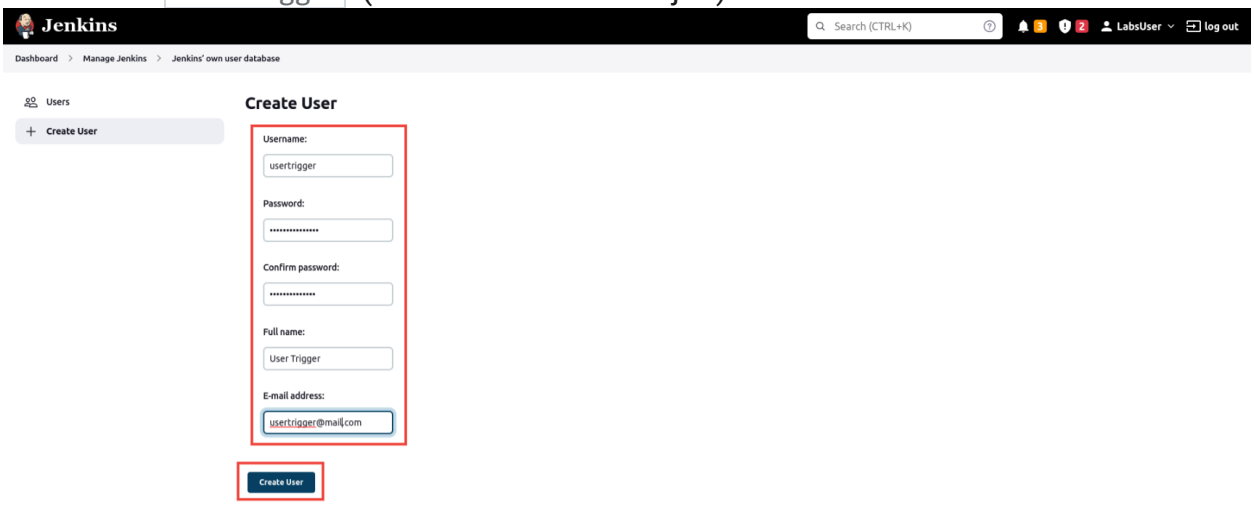
3. Back on the Jenkins home screen, navigate to ***Manage Jenkins*** > ***Manage User*** > select ***Create User*.** Create two Jenkins users:



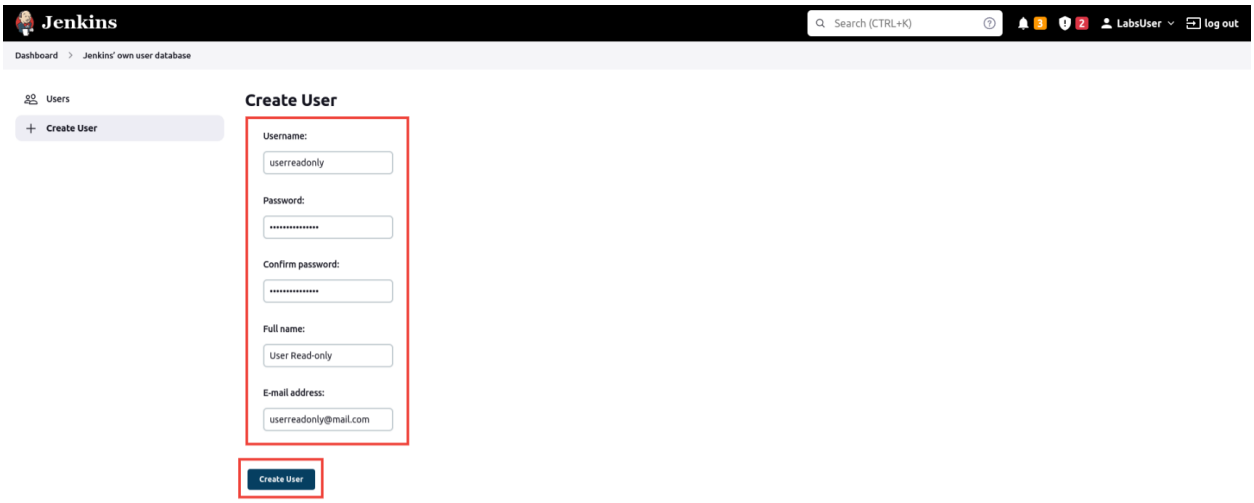
Create 2 new users by selecting the ***Create User*** button:



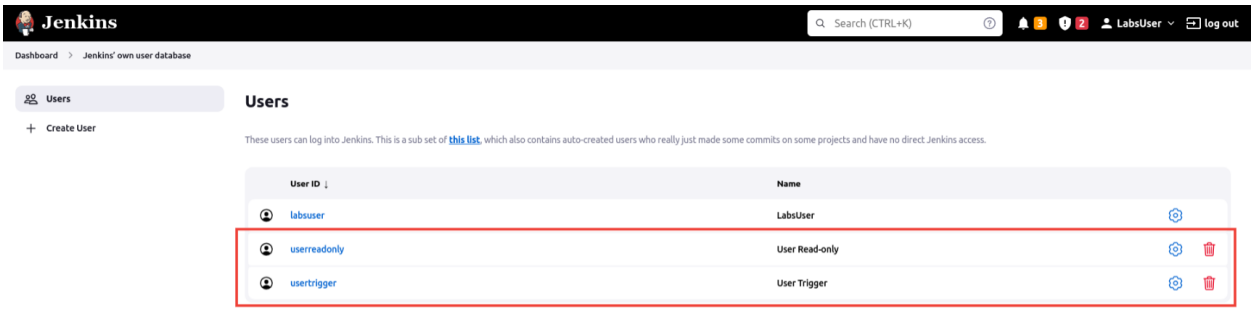
Username: `usertrigger` (can view and run the job)



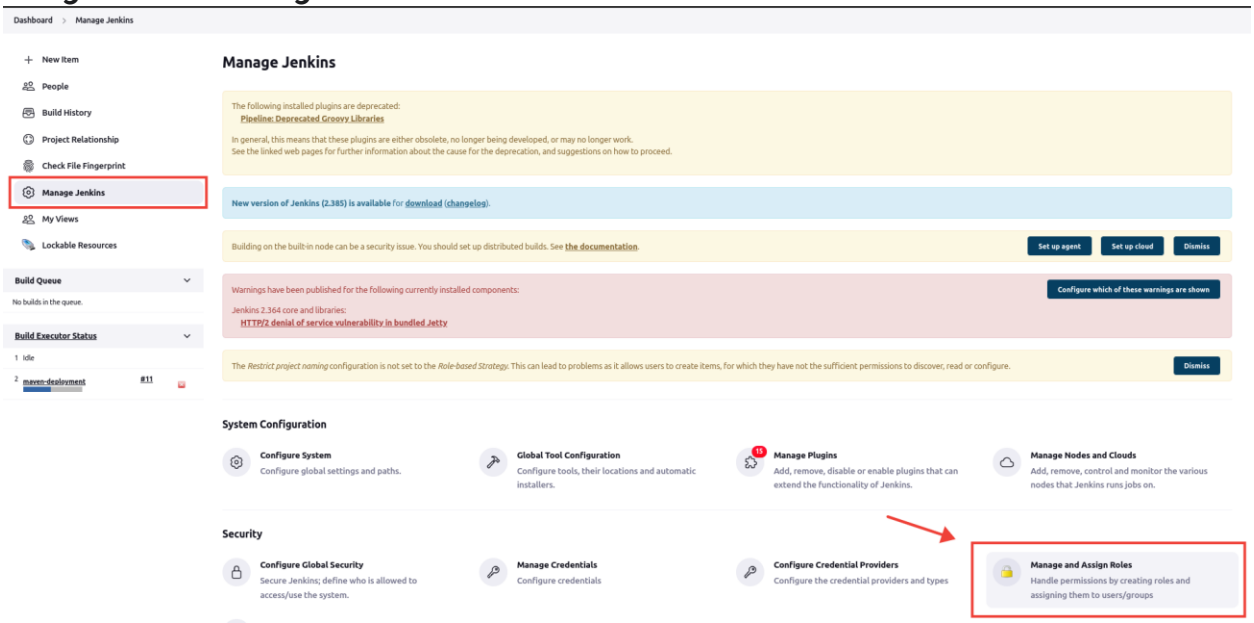
Username: `userreadonly` (can only view)

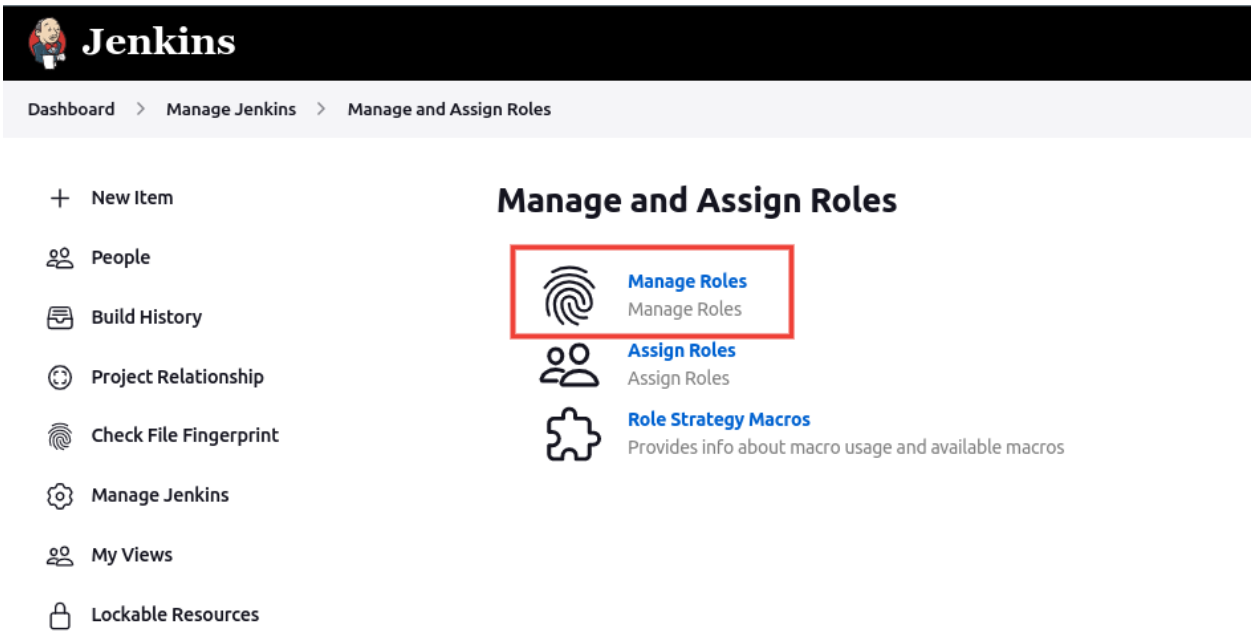


The user list will be like the following:

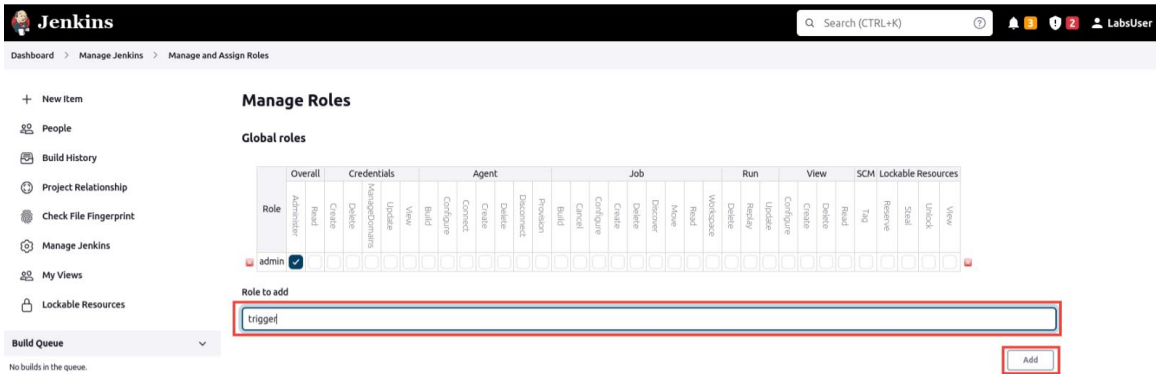


4. Back on the Jenkins home screen, navigate to ***Manage Jenkins*** > ***Manage and Assign Roles*** > ***Manage Roles***.

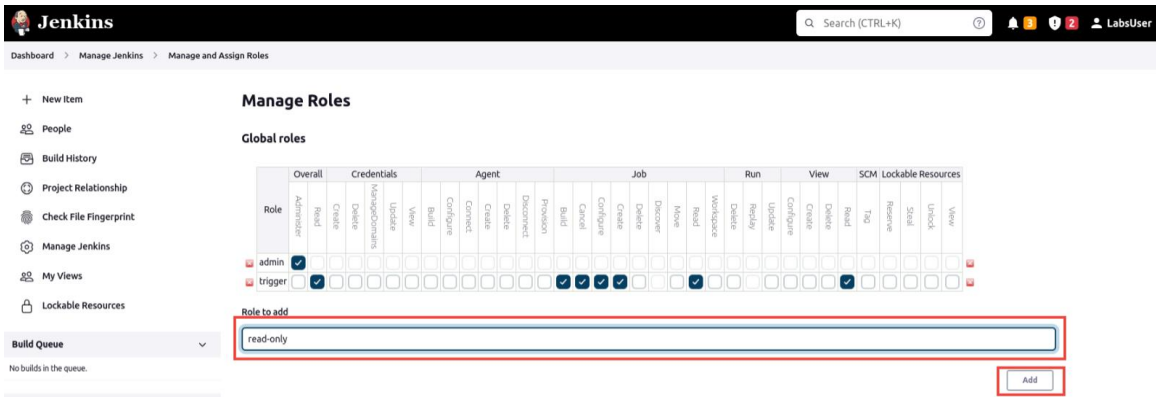




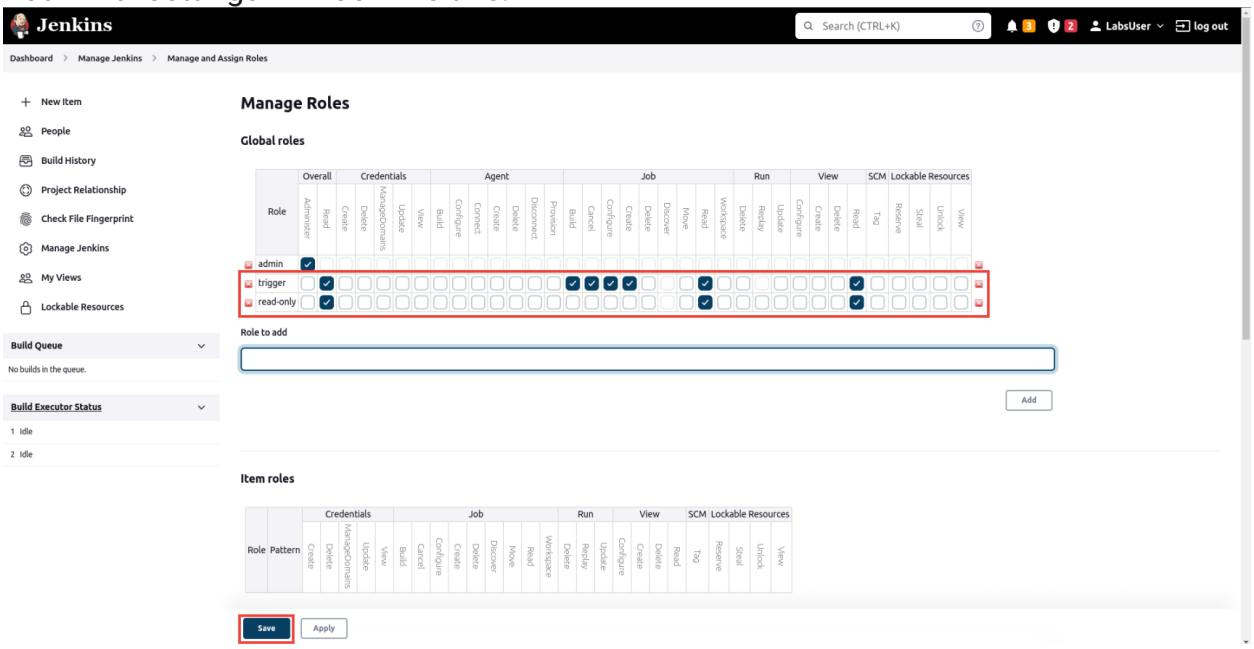
5. Create the role as follows:
    a. Role name: `trigger` make sure to check the options below:
        i. ***Overall: Read***
        ii. ***Job: Build / Cancel / Configure / Create / Read***
        iii. ***View: Read***



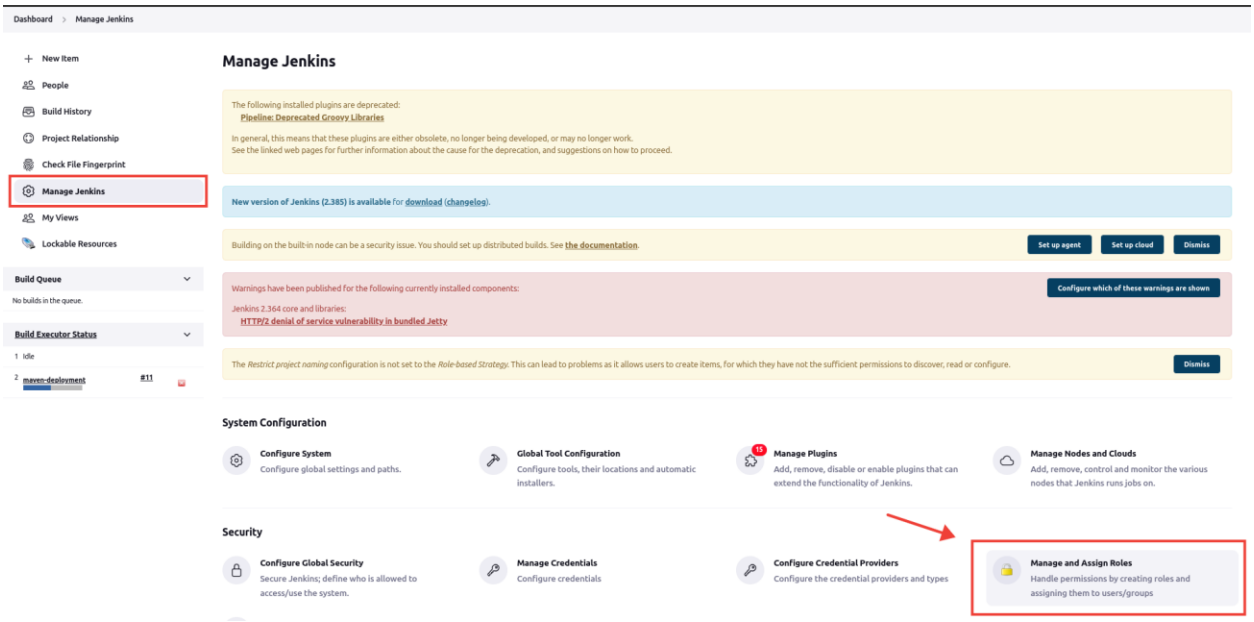
    b. Role name: `read-only` make sure to check the options below:
        i. ***Overall: Read***
        ii. ***Job: Read***
        iii. ***View: Read***

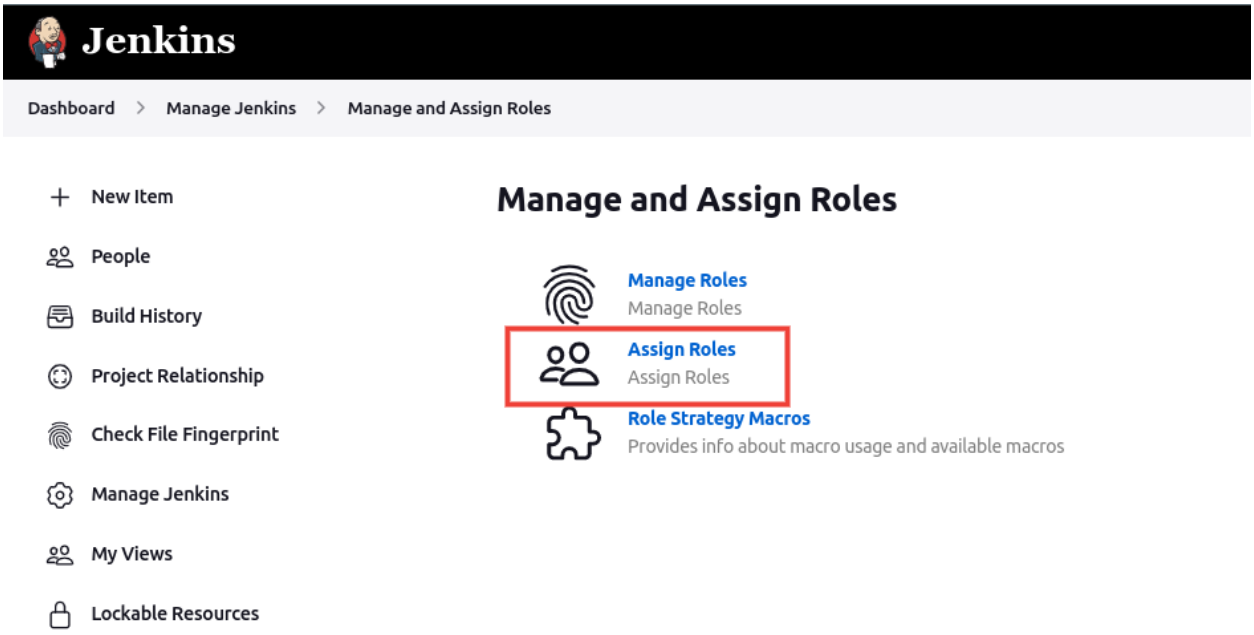


6. Your final settings will look like this:

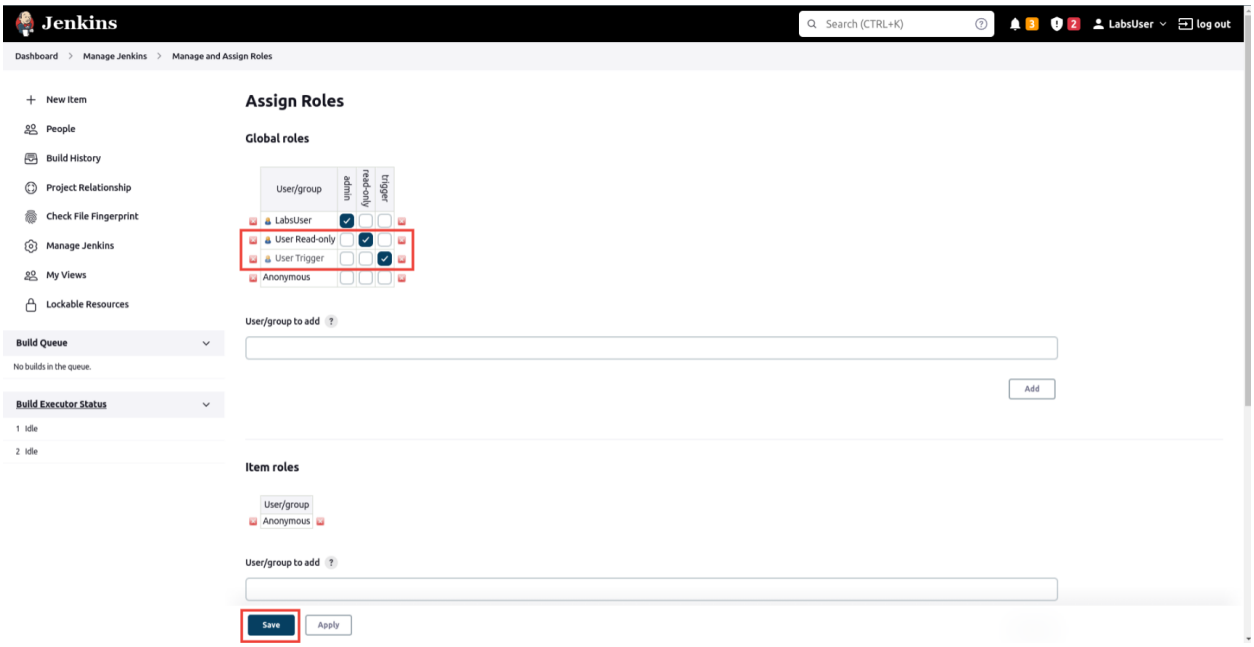


7.
8. Save your configuration
9. Back on the Jenkins home screen, navigate ***Manage Jenkins*** > ***Manage and Assign Roles*** > ***Assign Roles*** > and assign your created users into the created roles from the previous step.



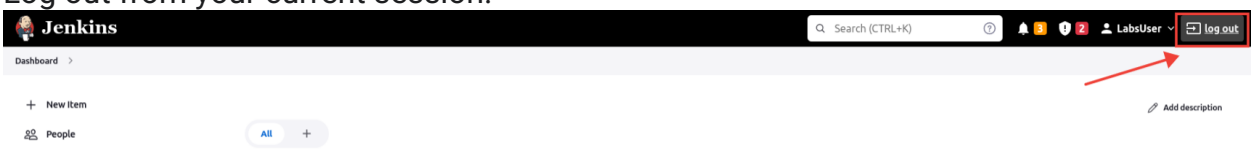


a. Assign the `User Read Only` to role `read-only`
b. Assign the `User Trigger` to role `trigger`

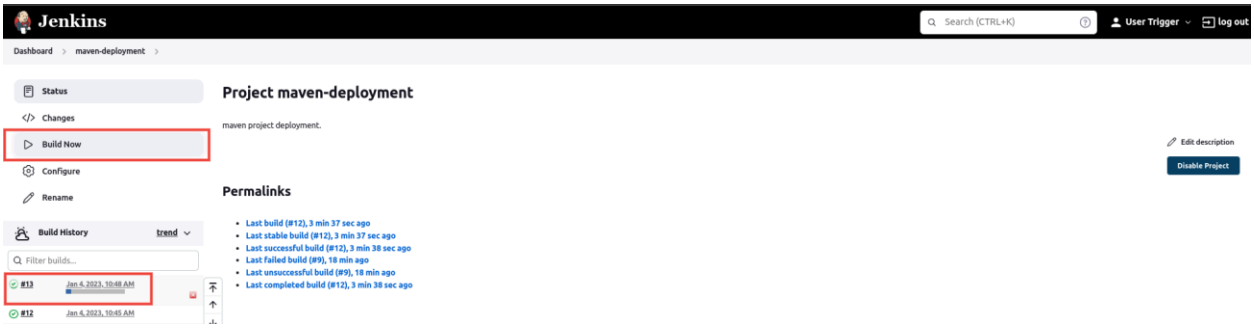


Save your configurations

10. Log out from your current session:



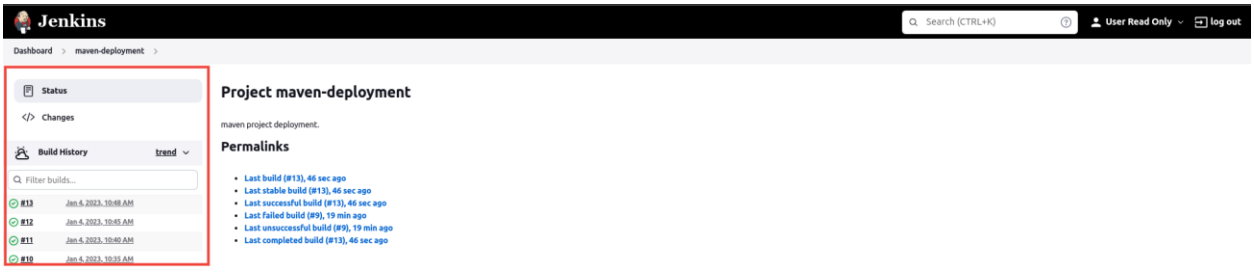
and then log in as the user `usertrigger` that should be able to execute the job:



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

11. Log out from your current session:

and then log in as the user `userreadonly`:



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

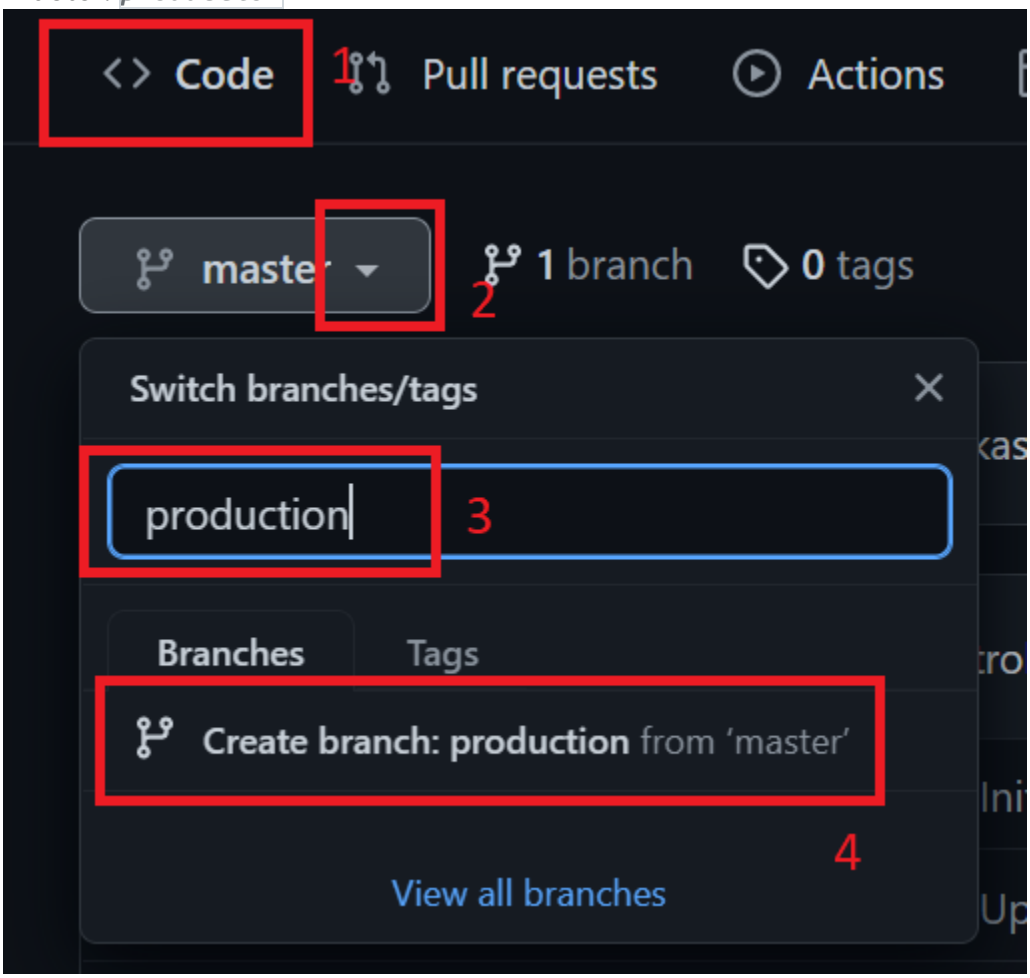
Mark task as complete
Report issue

8. Configure Jenkins to Automatically Trigger the Jenkins Task

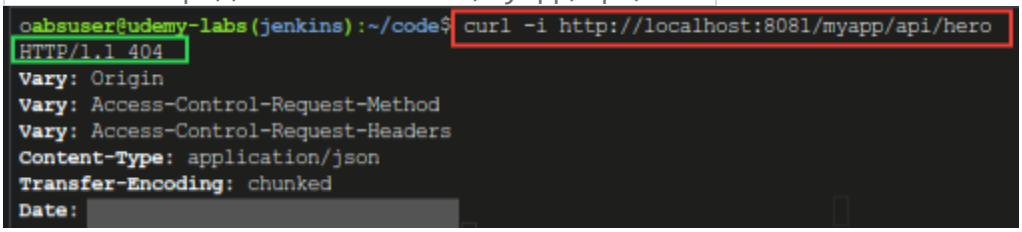
Create a `production` branch on your forked repository.

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.

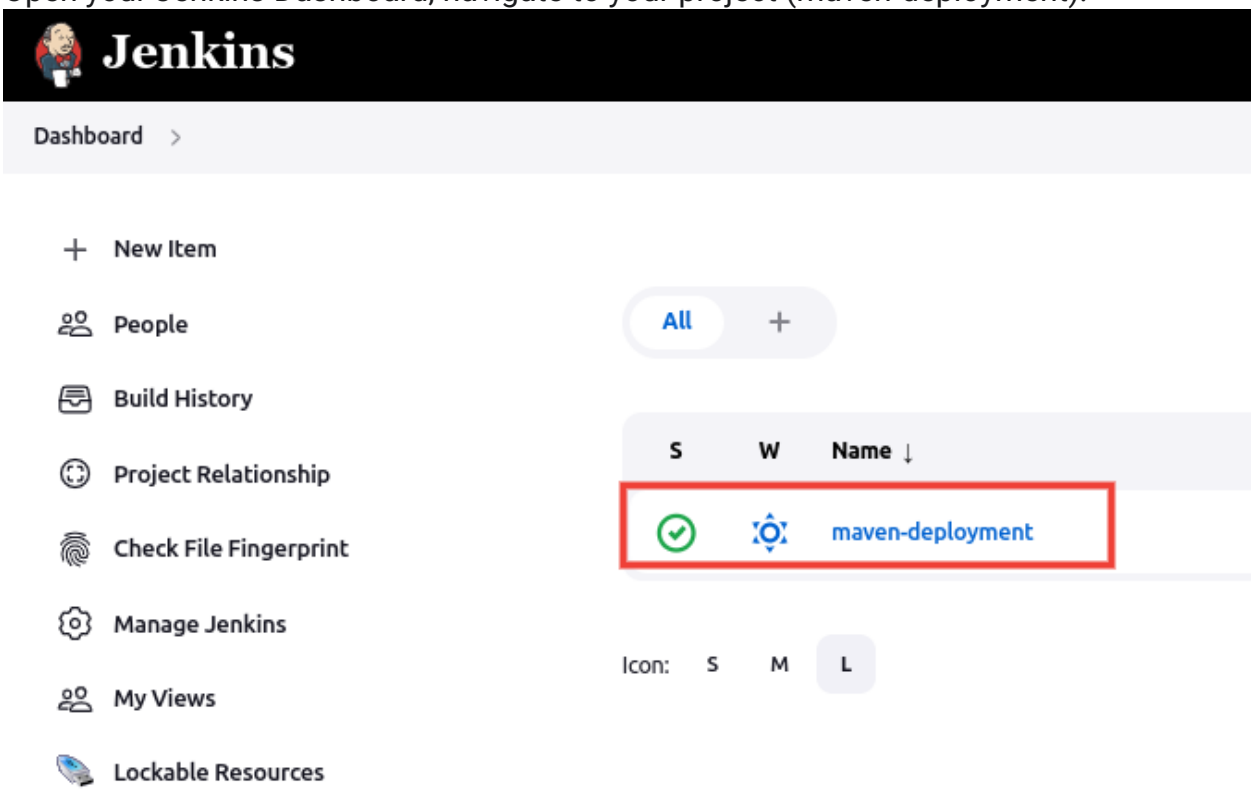
1. Go to your Github forked repository. Create a new branch from master: ***`production`***



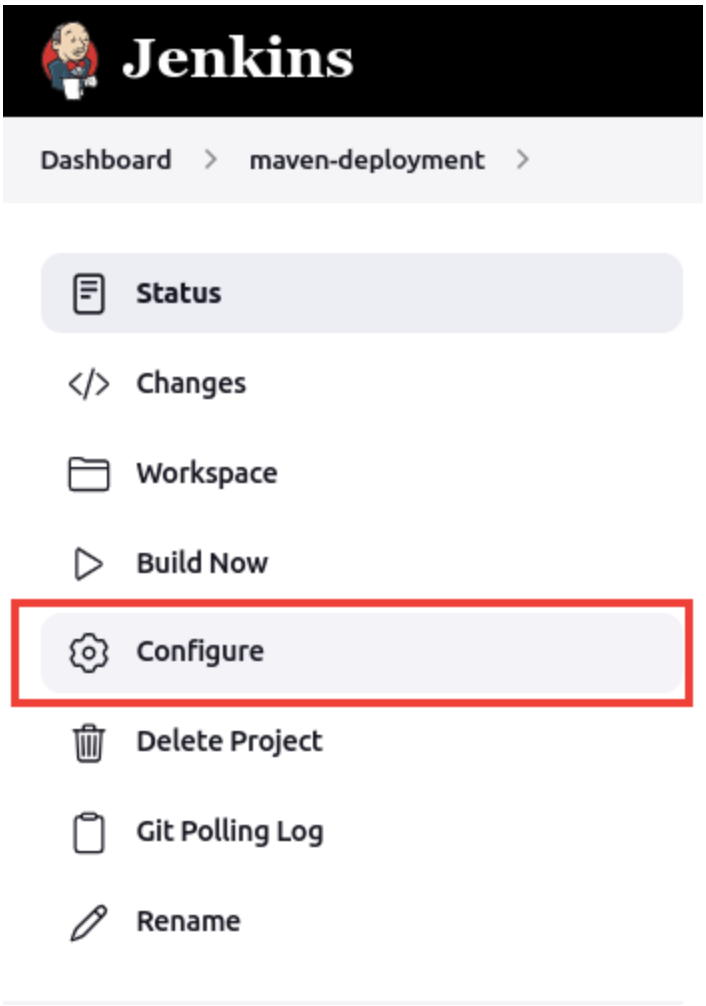
2. In your workspace terminal, curl to /hero (you will get a 404 at this point): `curl -i http://localhost:8081/myapp/api/hero`



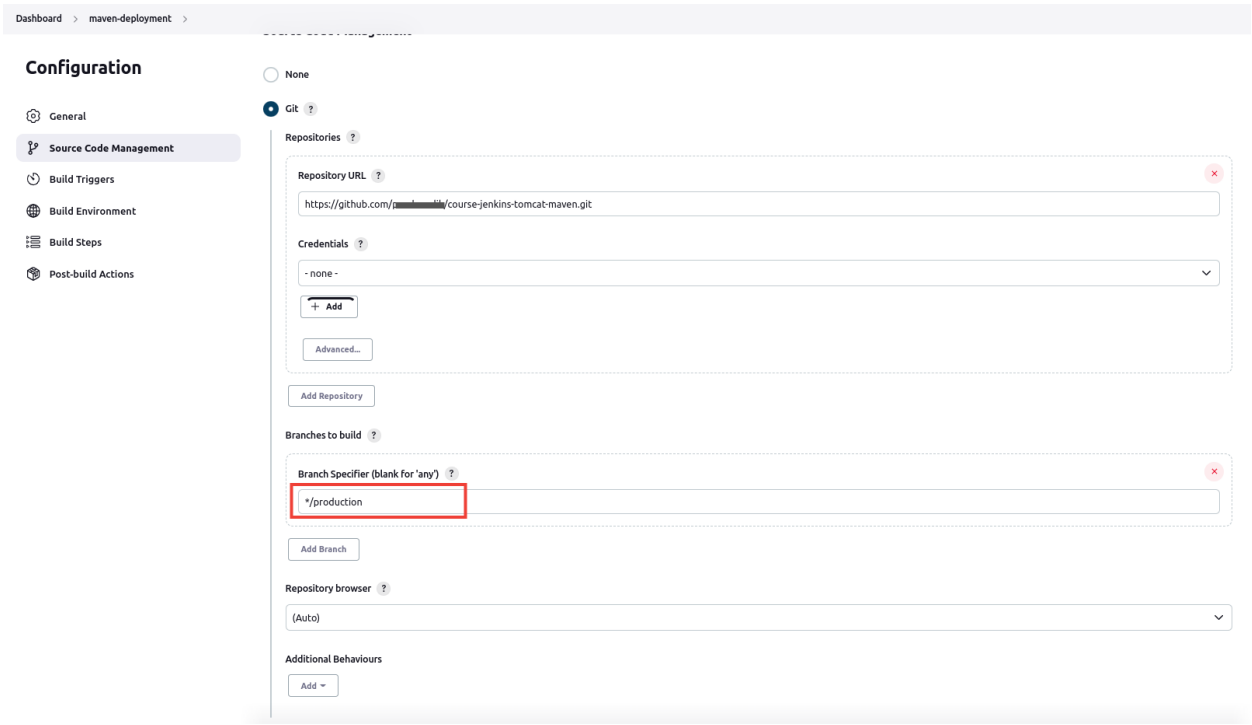
3. Open your Jenkins Dashboard, navigate to your project (maven-deployment):



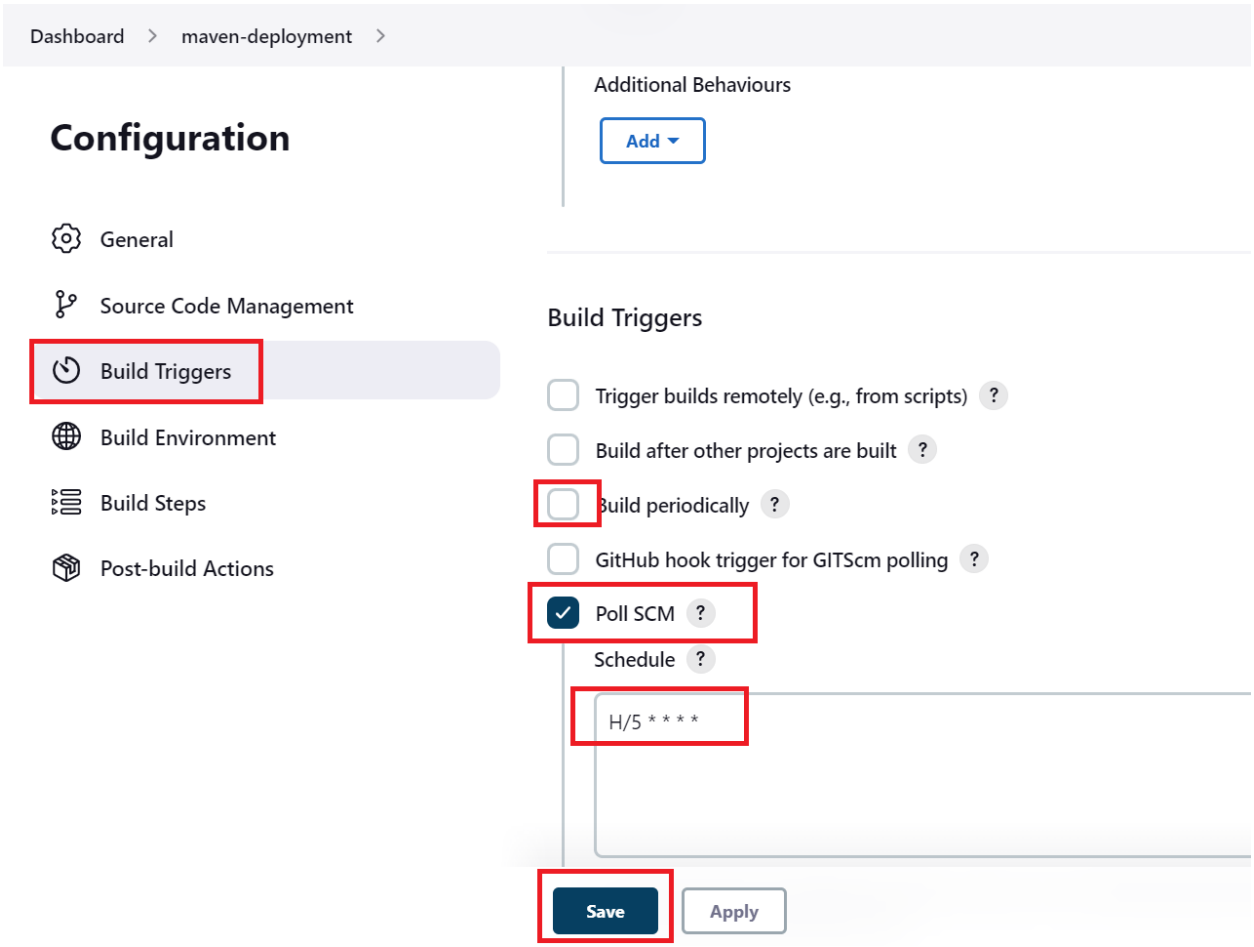
Reconfigure your project to configure `production` config:



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

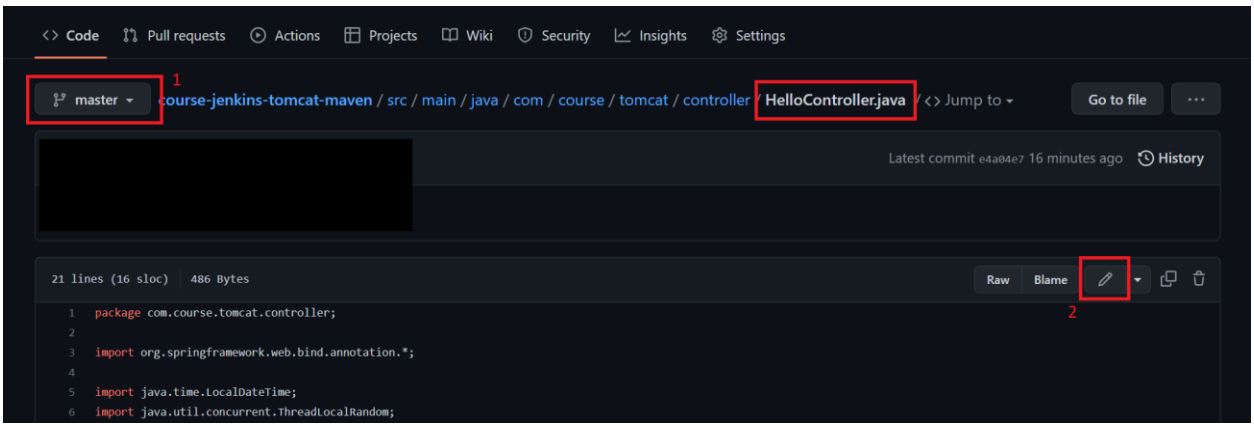


In the ***Build Trigger*** section, unselect the ***Build periodically*** option and select the ***Poll SCM*** instead. Add cron annotation to scan the repository every 5 minutes (`H/5 * * * *`), ***Poll SCM*** option allows the Jenkins services to scan and build the project if there are any changes 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.



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



Copy and paste the entire source code below.

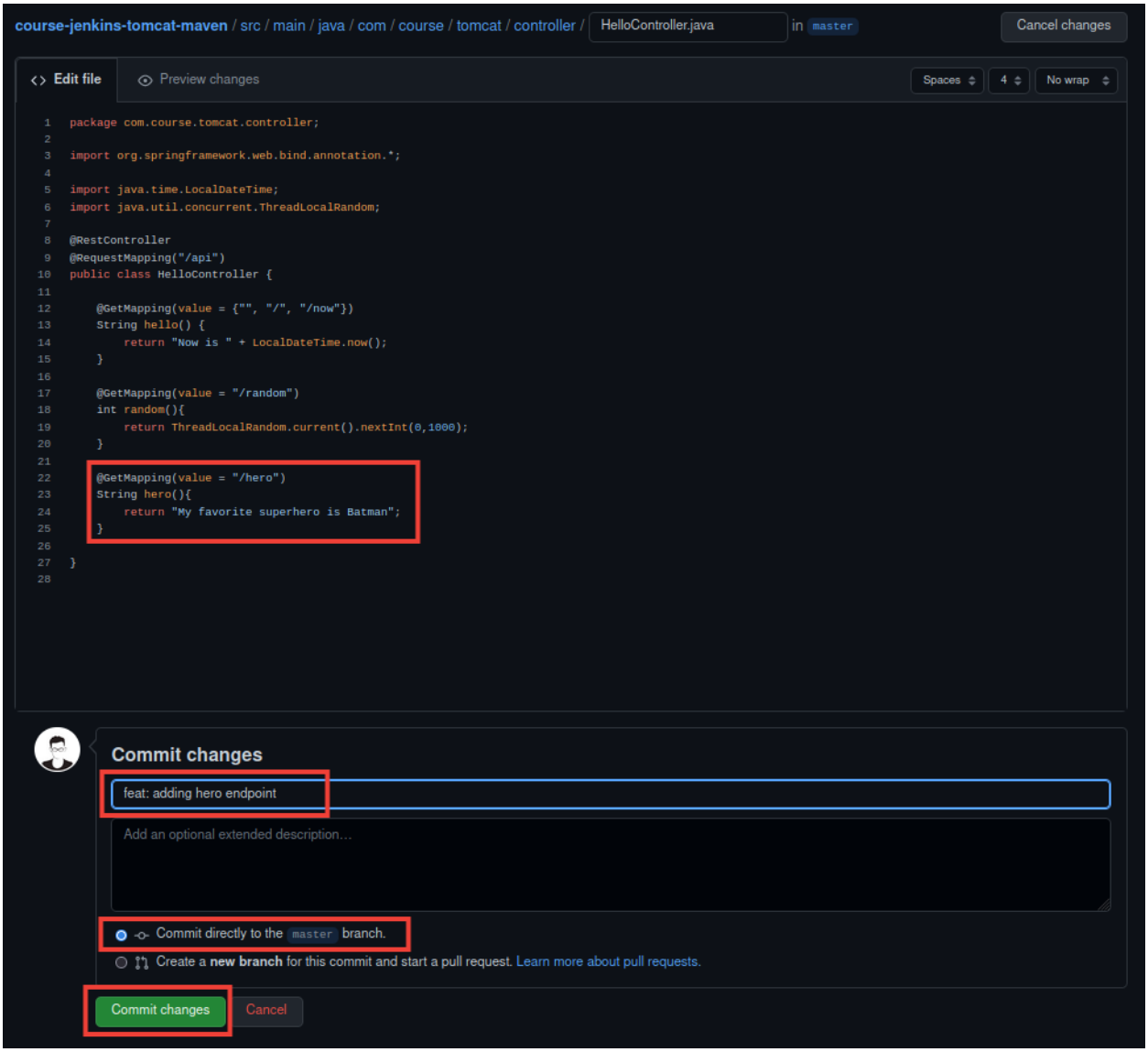
```
1. package com.course.tomcat.controller;
2.
3. import org.springframework.web.bind.annotation.*;
4.
```

```
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.}
```

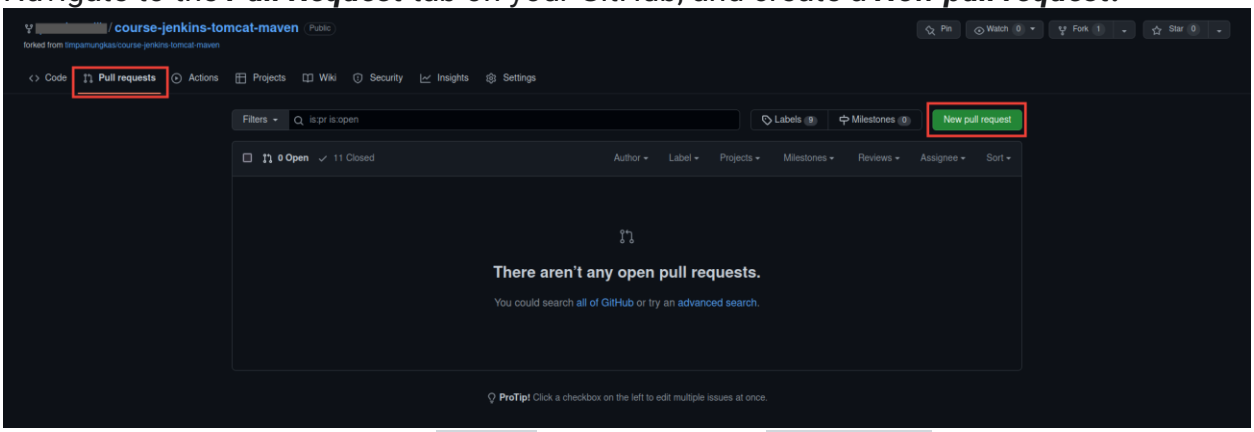
Add any commit message you want.

Commit directly to the `master` branch

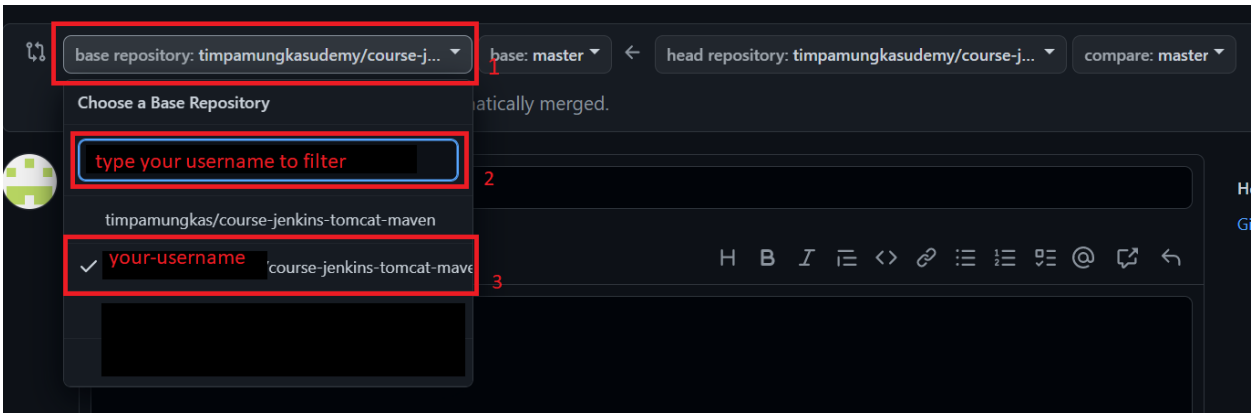
Click ***Commit changes***



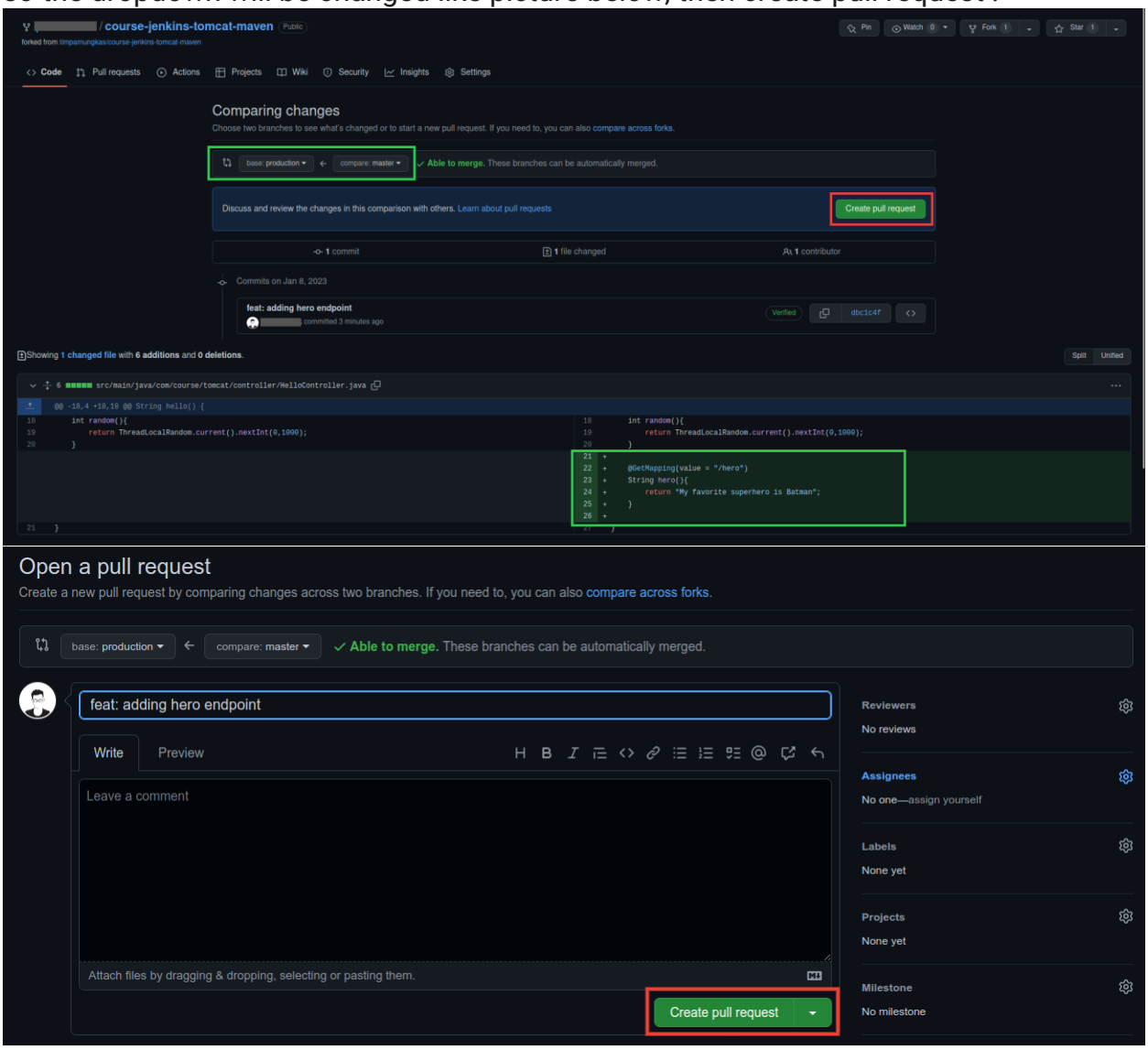
5. Navigate to the ***Pull Request*** tab on your GitHub, and create a ***New pull request***:



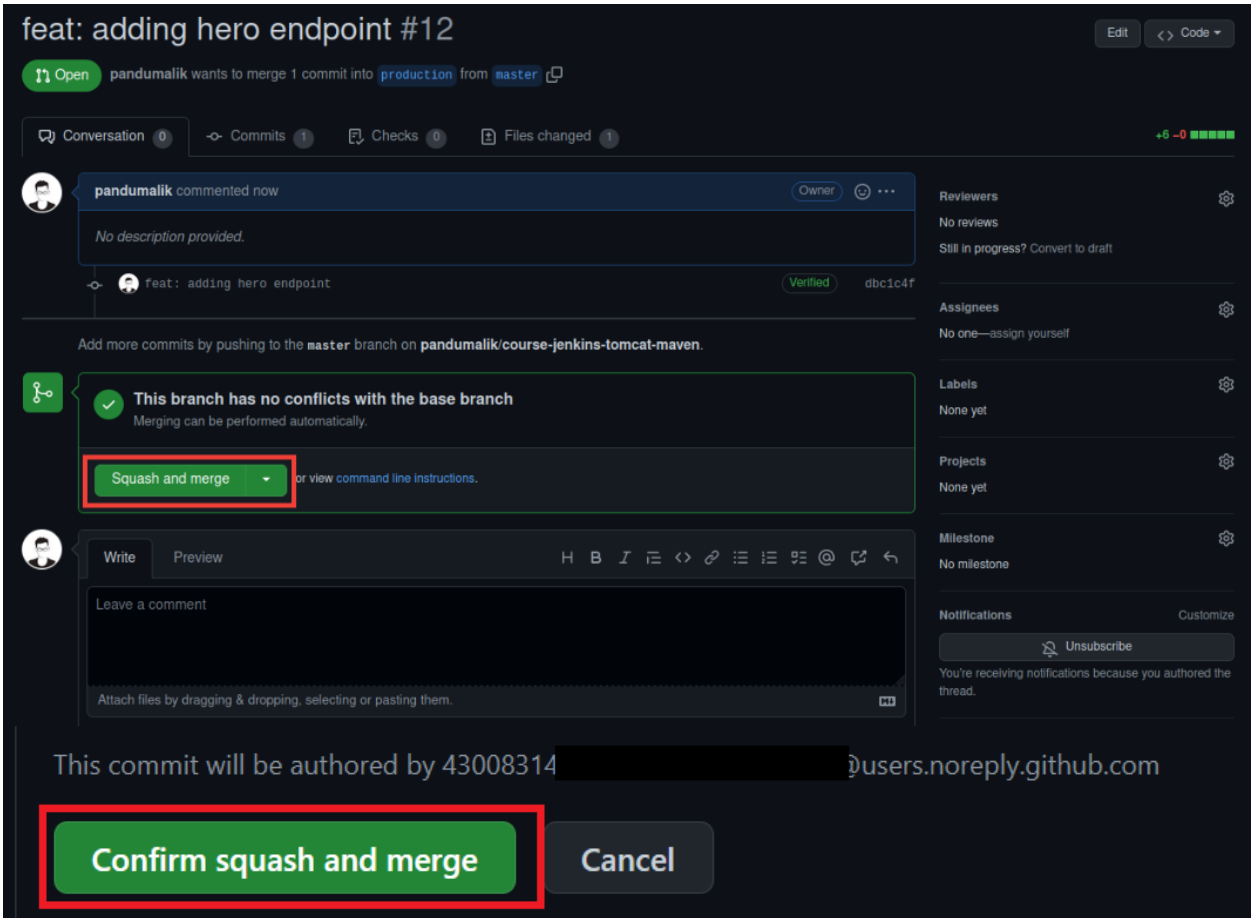
Create a Pull request from the `master` branch into the `production` branch. Select your own forked repository from the dropdown



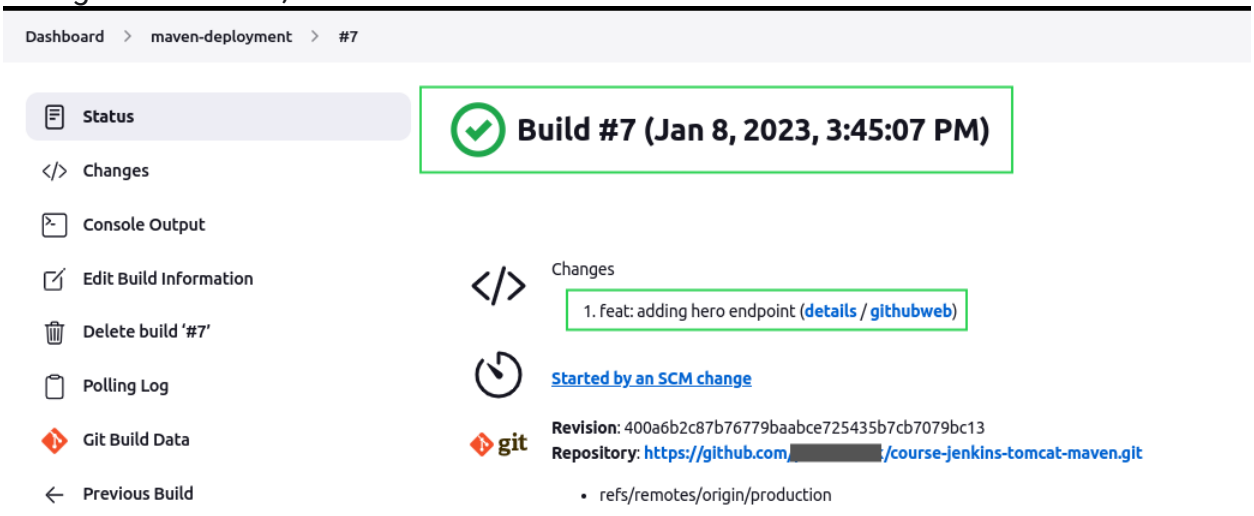
so the dropdown will be changed like picture below, then create pull request :



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

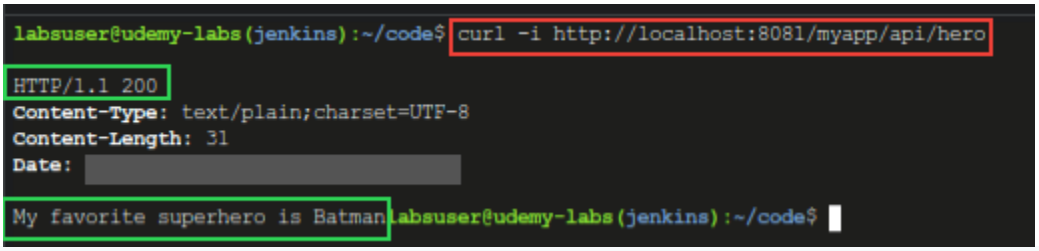


7. At this point, Jenkins will scan the repository every 5 minutes. Since there is a change on the code, the build task will be executed.



8. 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
```



Resources

Documentation

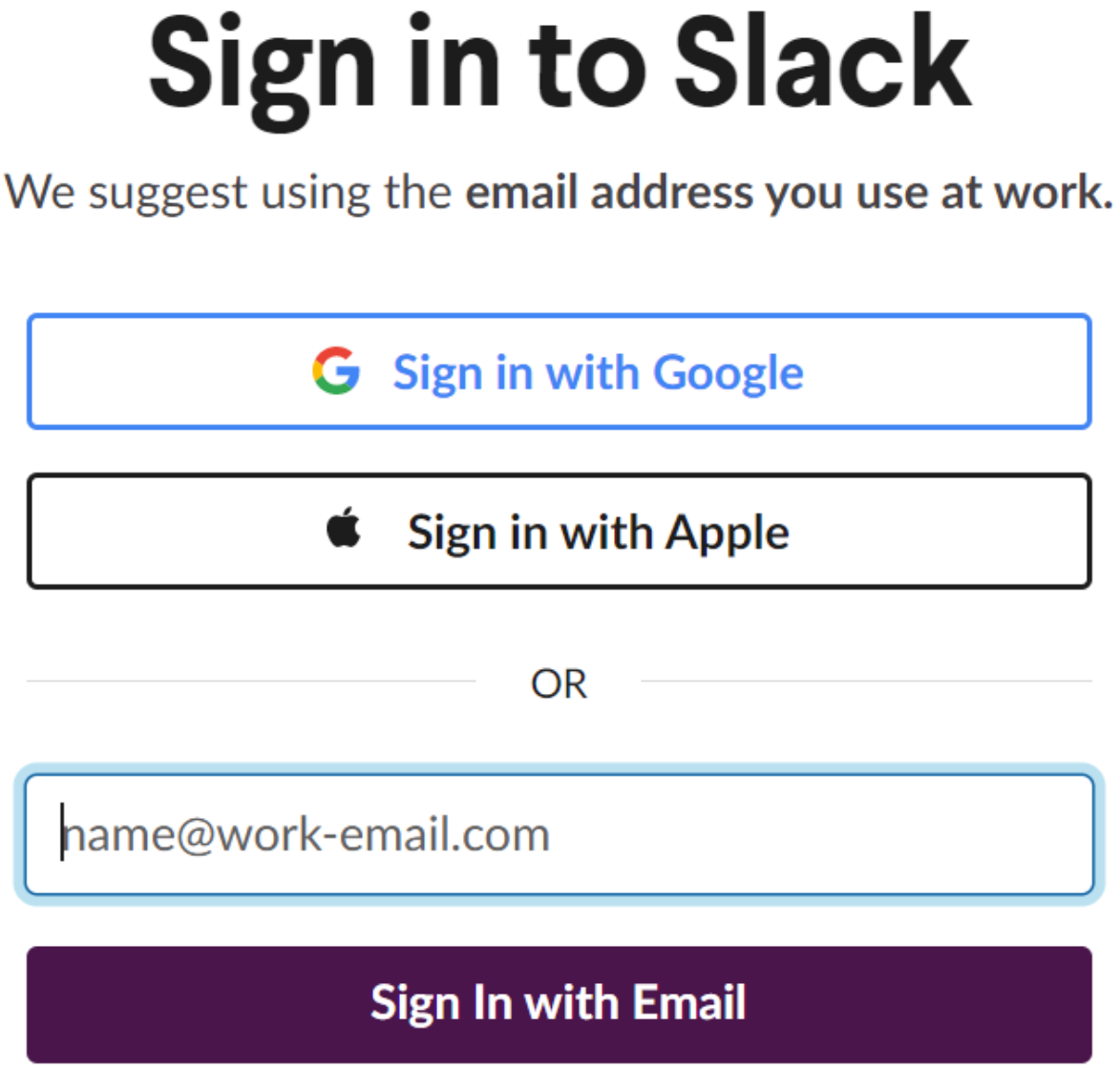
- Git pull request

Mark task as complete
Report issue

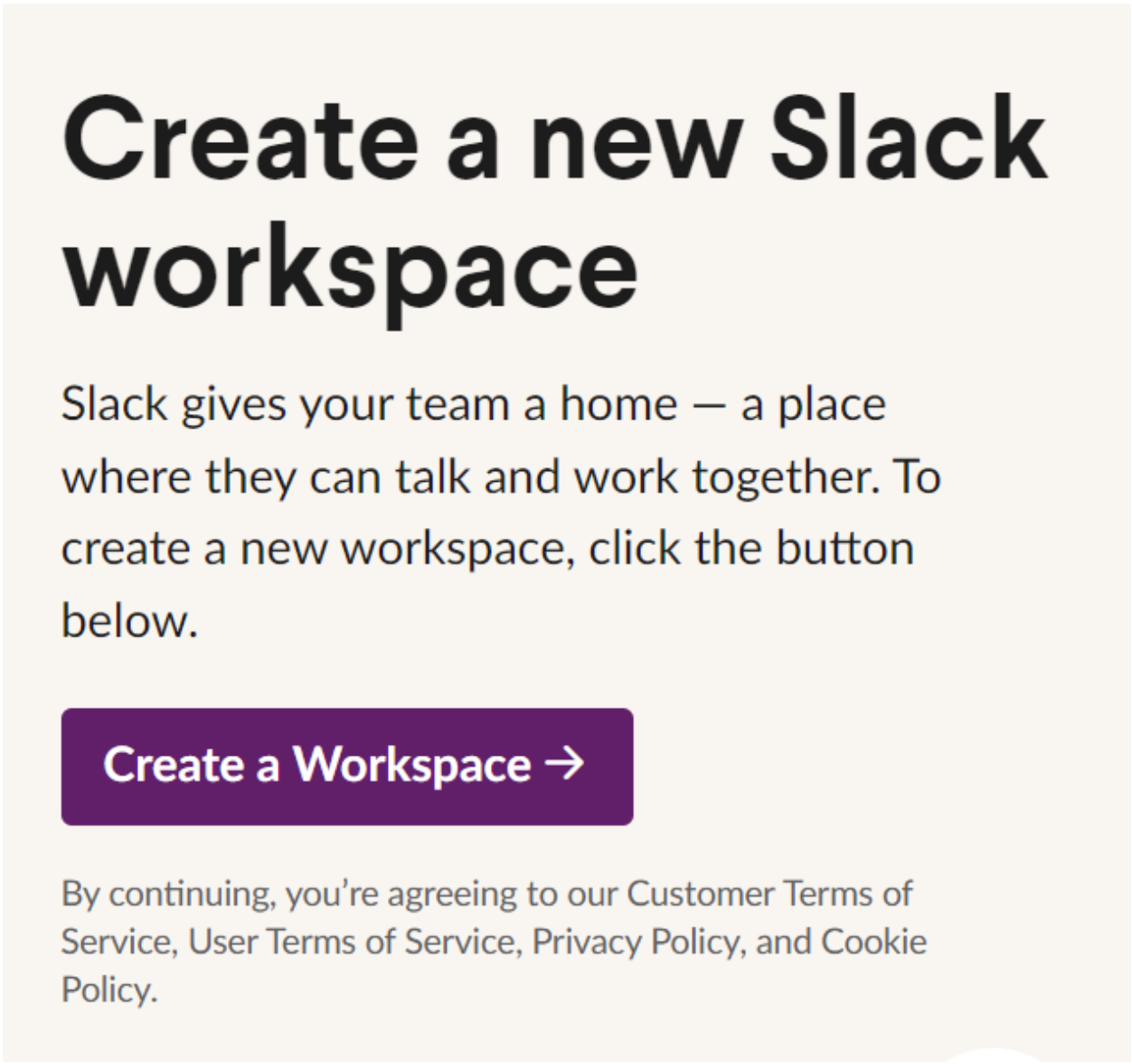
9. Add Jenkins CI to Slack for Notifications

Register to Slack messaging, and add Jenkins CI Slack application.

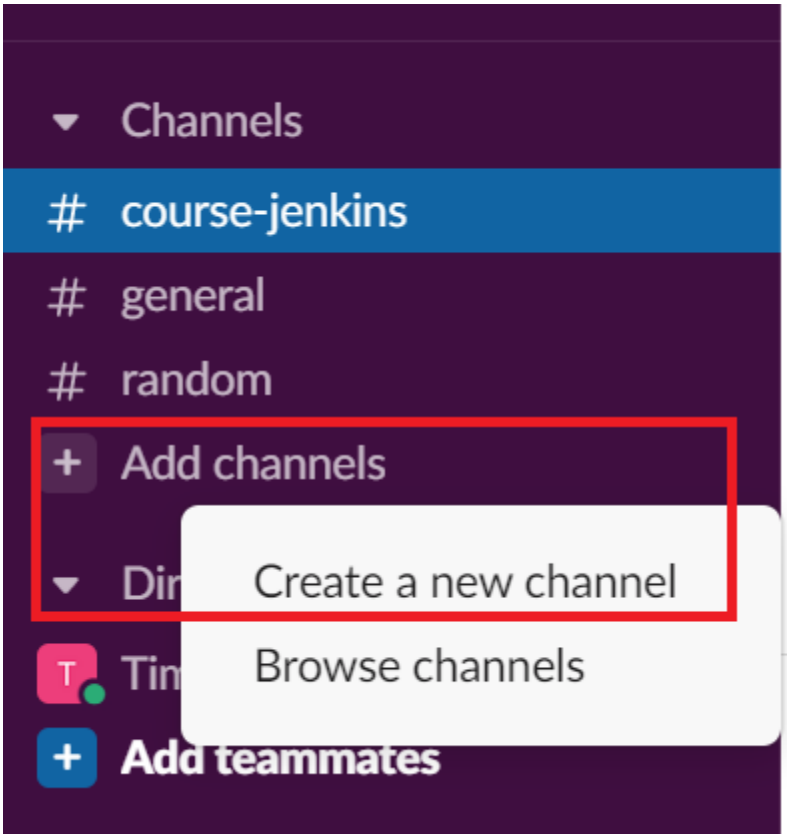
1. Register for a free Slack account and sign in.

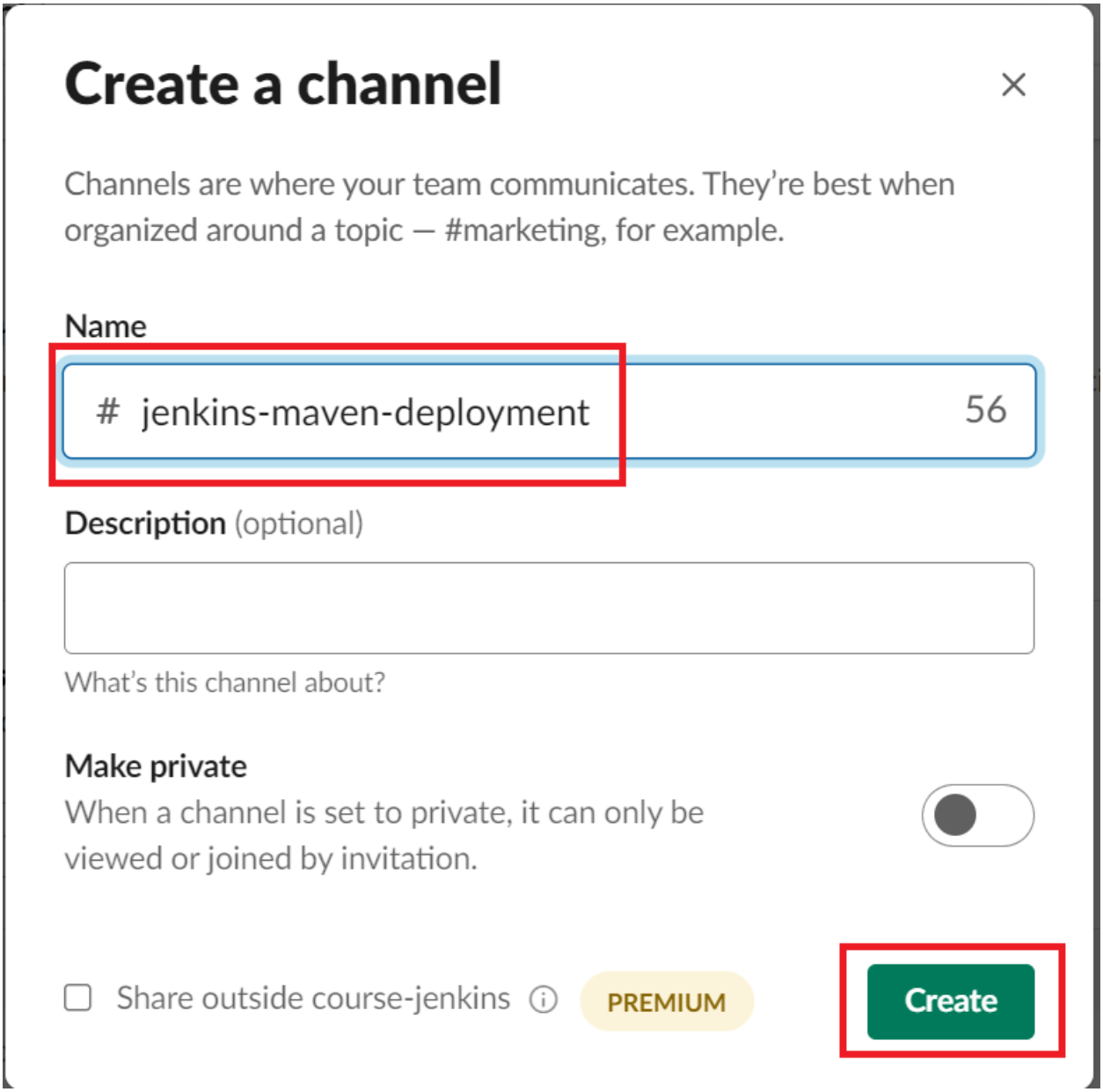


2. Create a new Slack workspace, name it as you like then navigate to your newly created workspace.

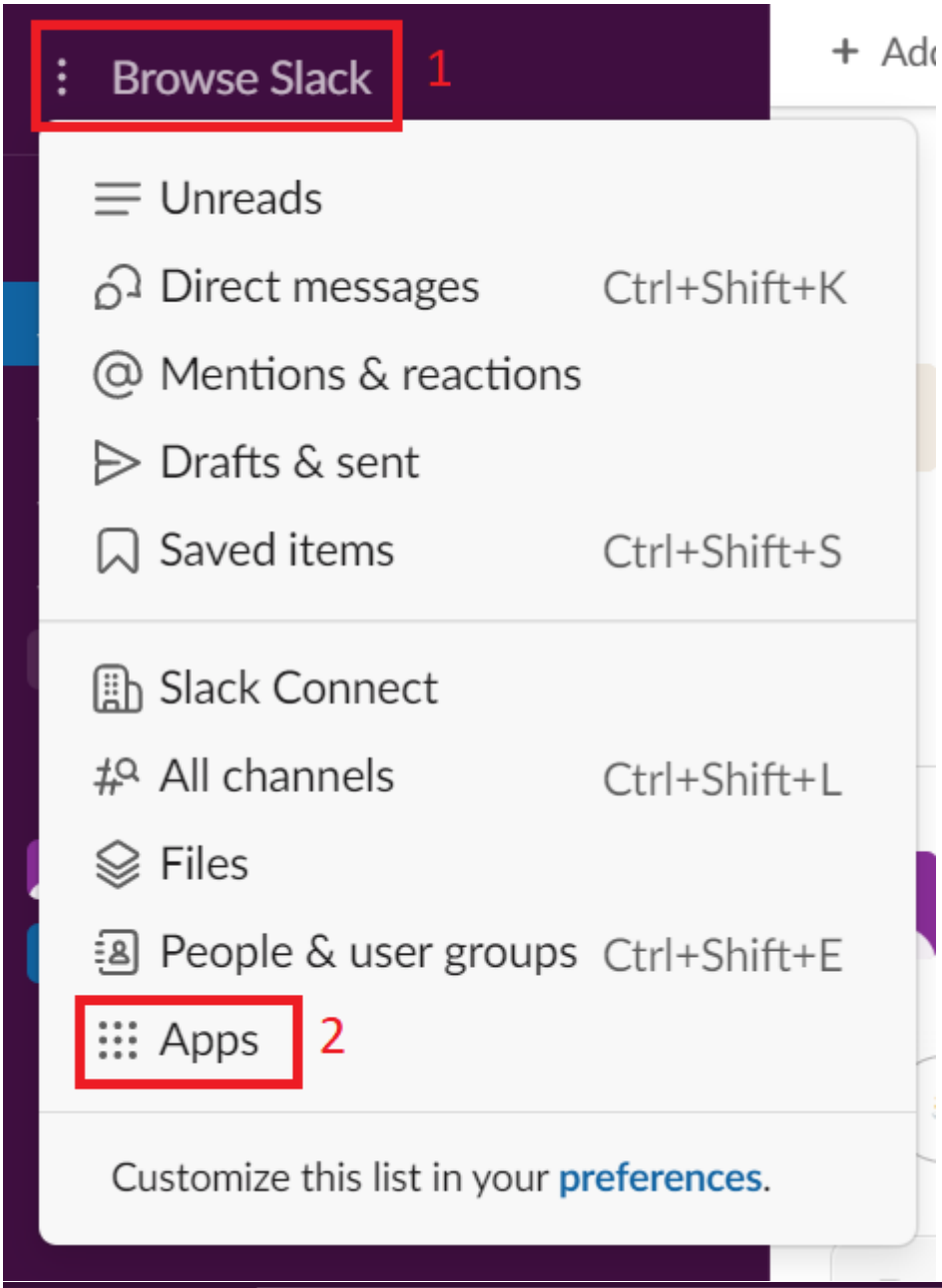


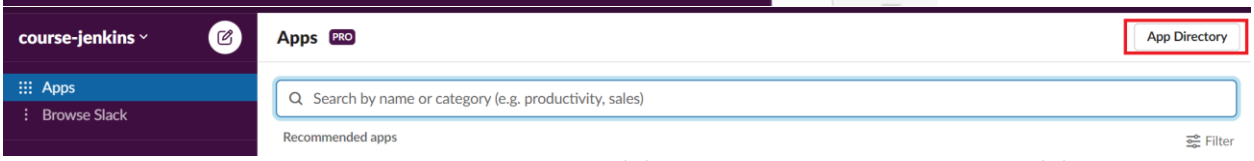
3. Create a channel to receive the Jenkins notifications. For example, name it ***jenkins-maven-deployment***


Channels
course-jenkins
general
random
+ Add channels
Create a new channel
Browse channels
Add teammates

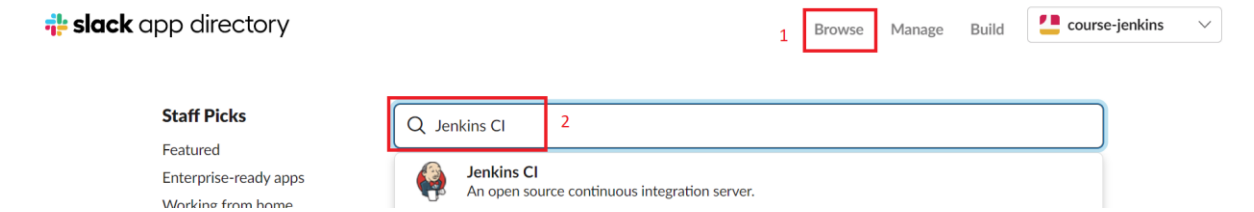


4. Navigate to the Slack apps directory from your Slack sidebar menu

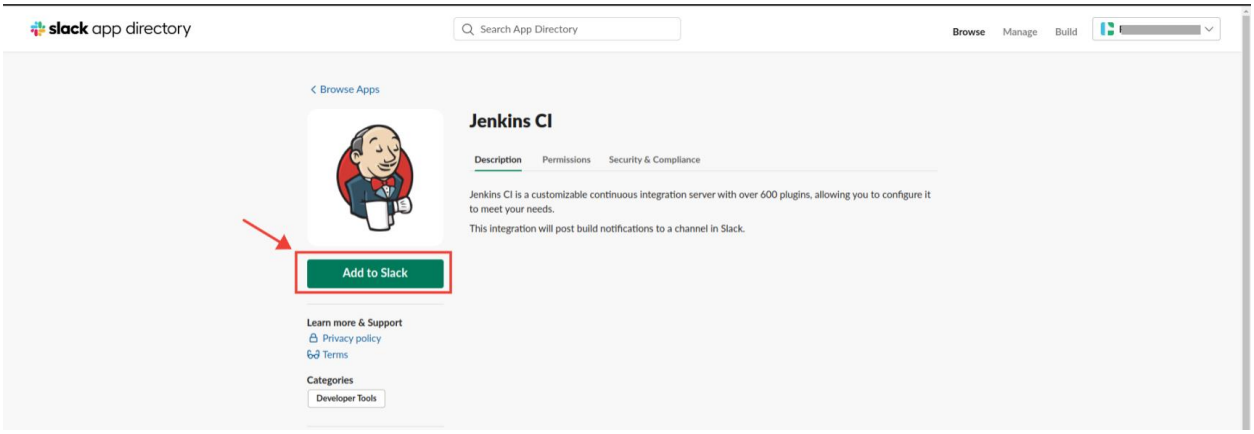




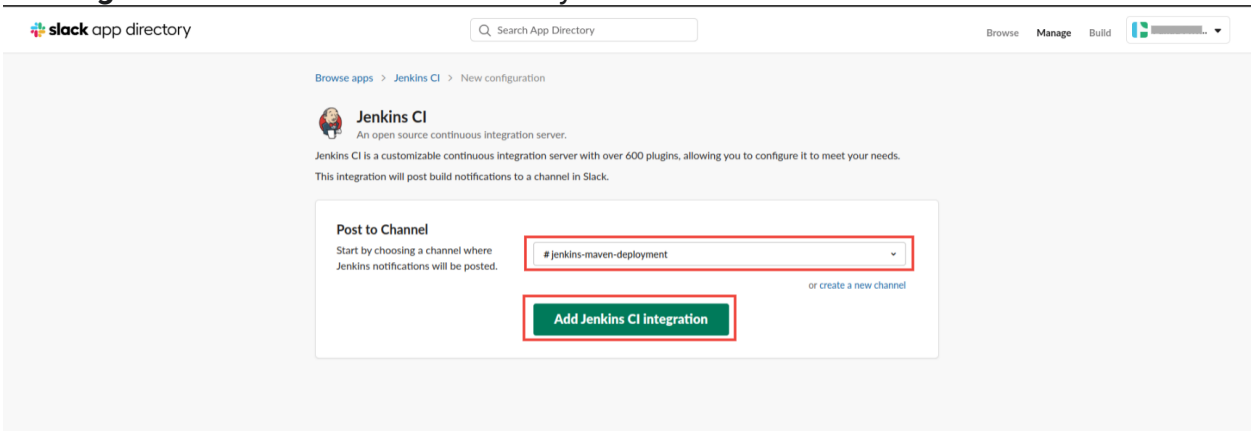
5. From the Slack App Directory, click ***Browse*** (1) and search for Jenkins CI (2)



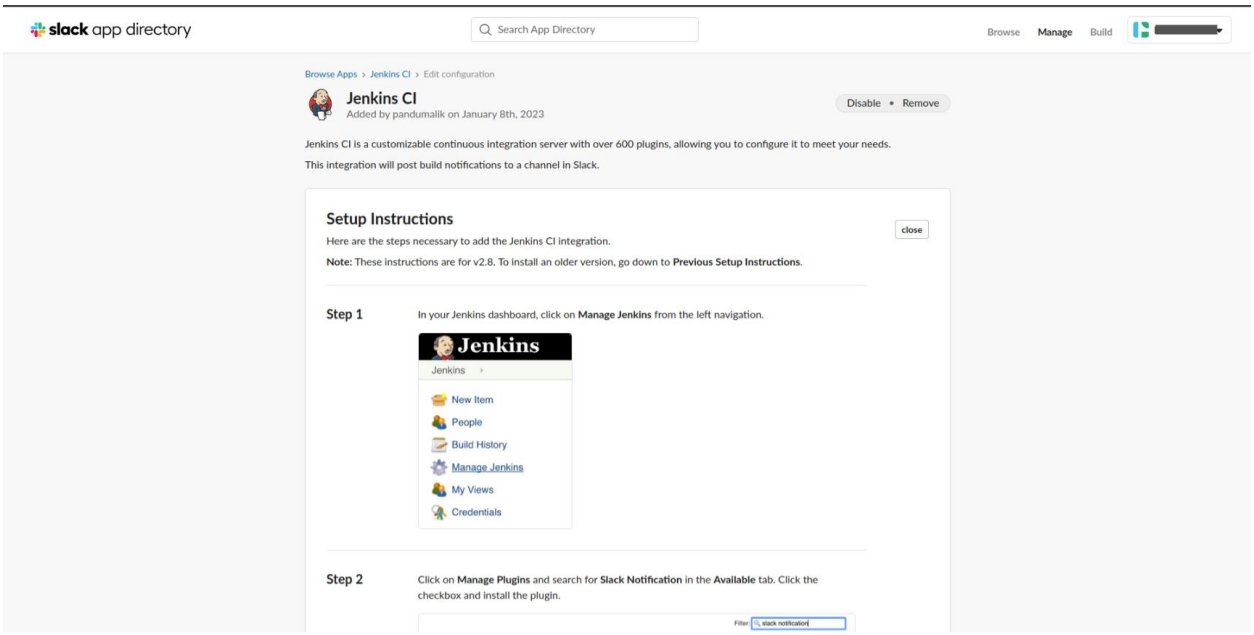
6. Add Jenkins CI to Slack



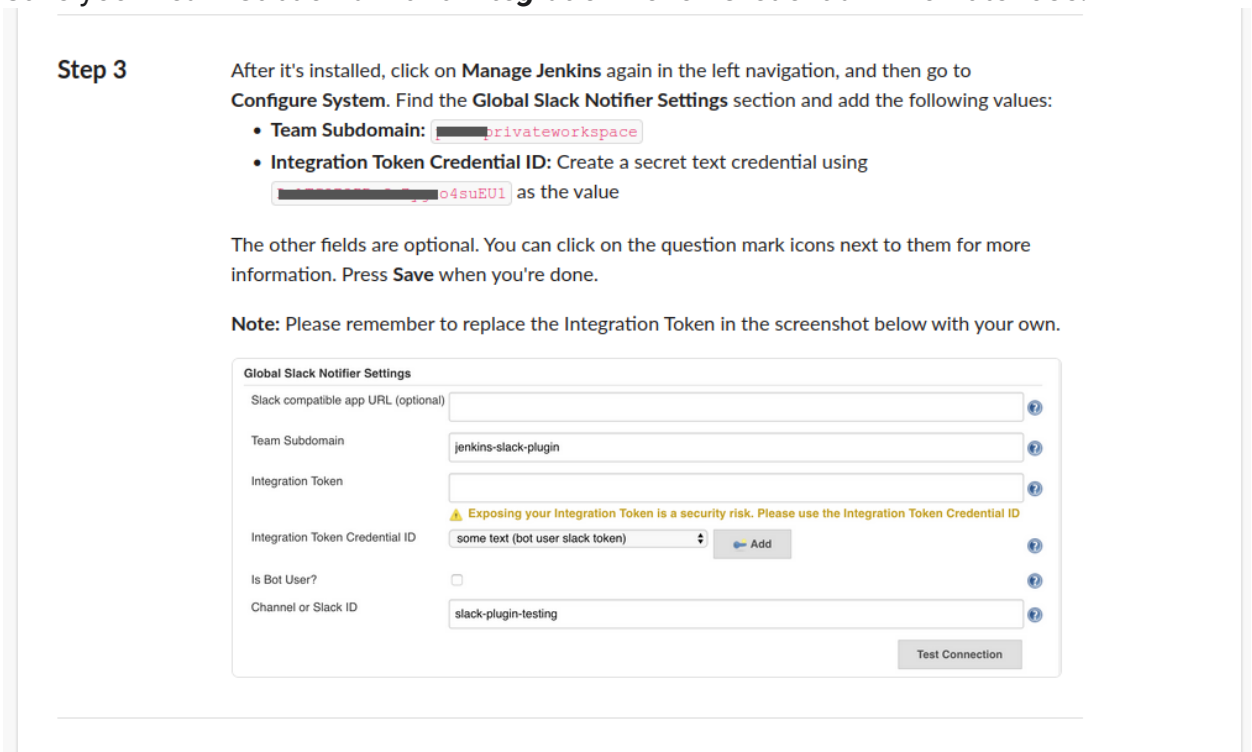
Enter your channel name from the previous step and then press **Add Jenkins CI Integration** to connect Jenkins CI to your Slack channel



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



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



Resources

Documentation

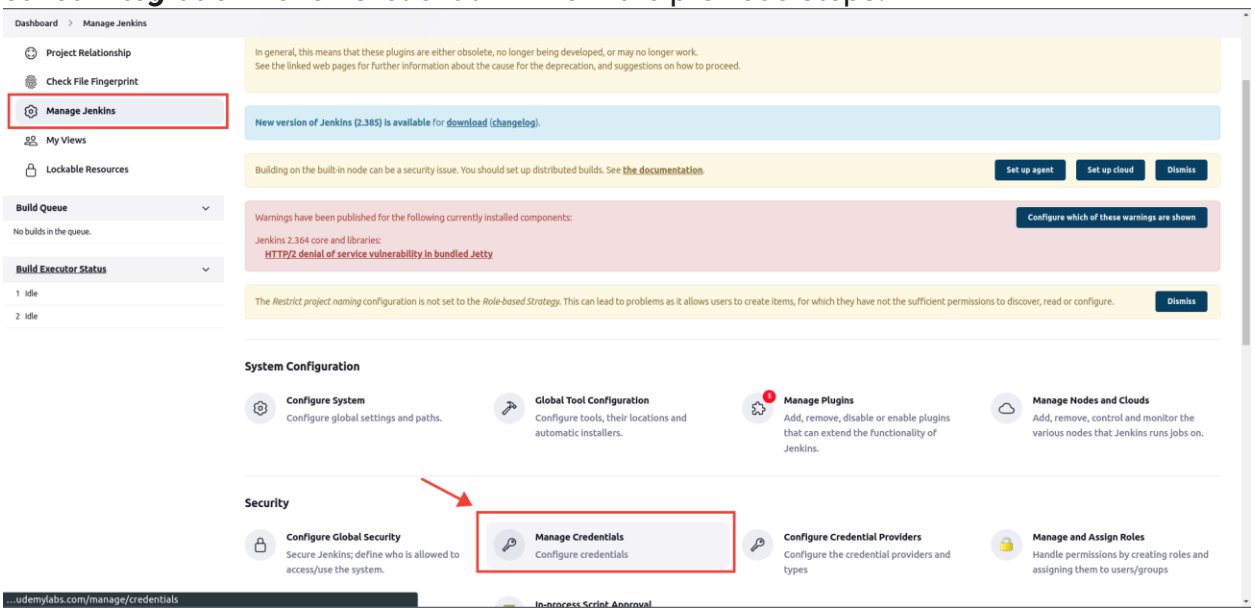
- Slack Jenkins App

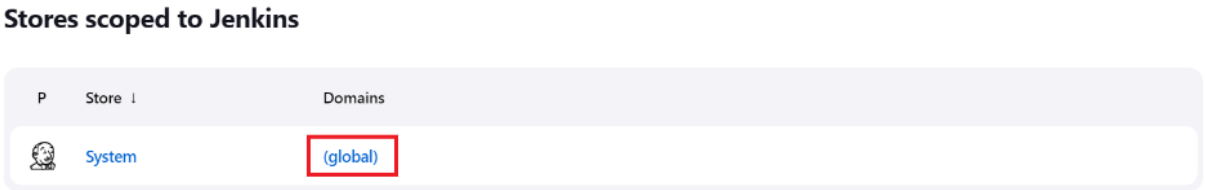
Mark task as complete
Report issue

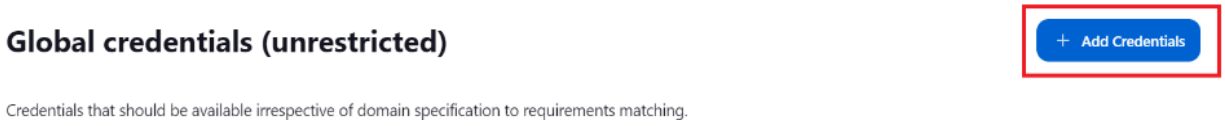
10. Configure Jenkins for Slack Notifications

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:

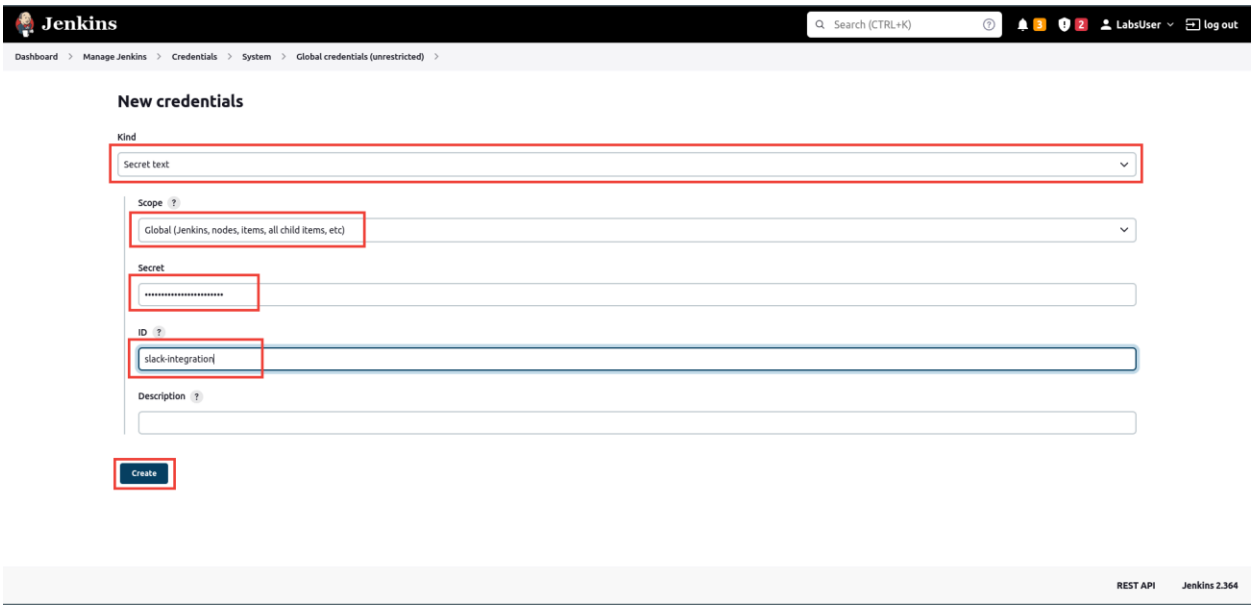




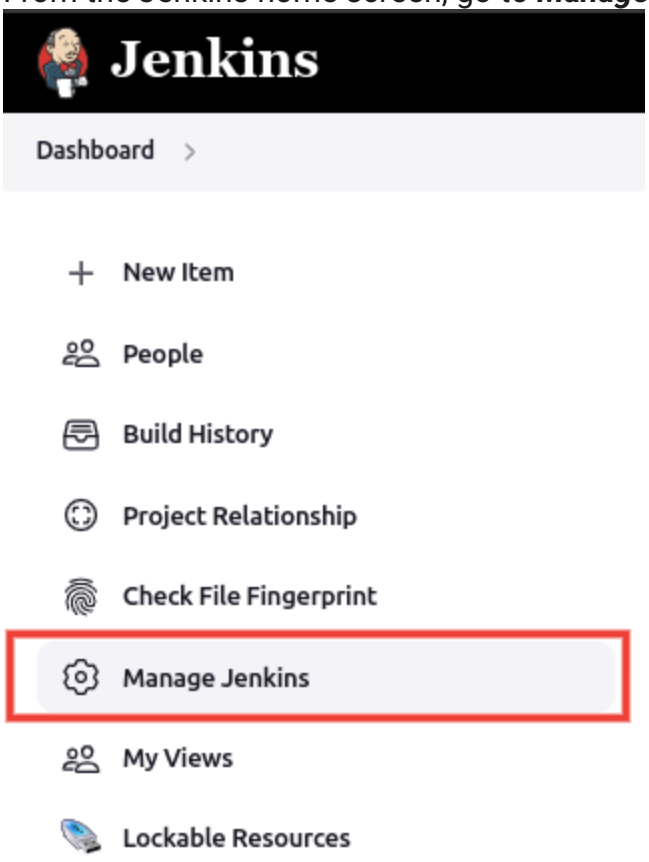


And then enter your credential details:

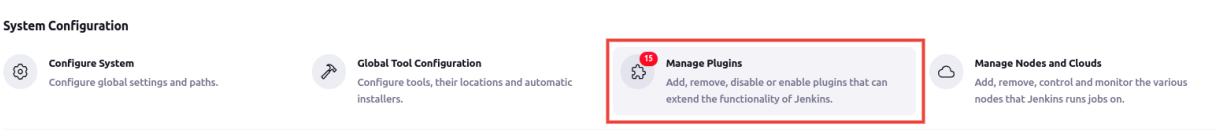
a. Kind: `Secret Text`
b. Scope: `Global`
c. Secret: the ***Integration Token Credential ID*** you saved from the previous steps
d. ID: `slack-integration`



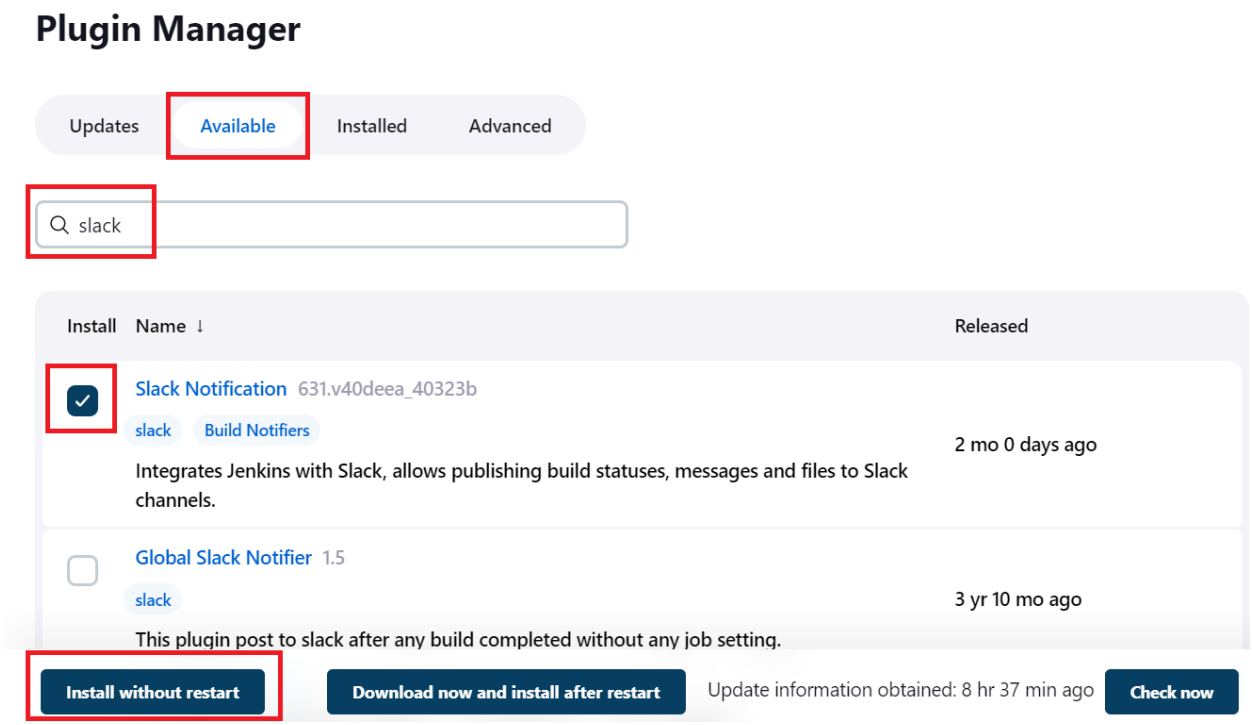
2. From the Jenkins home screen, go to ***Manage Jenkins*** :



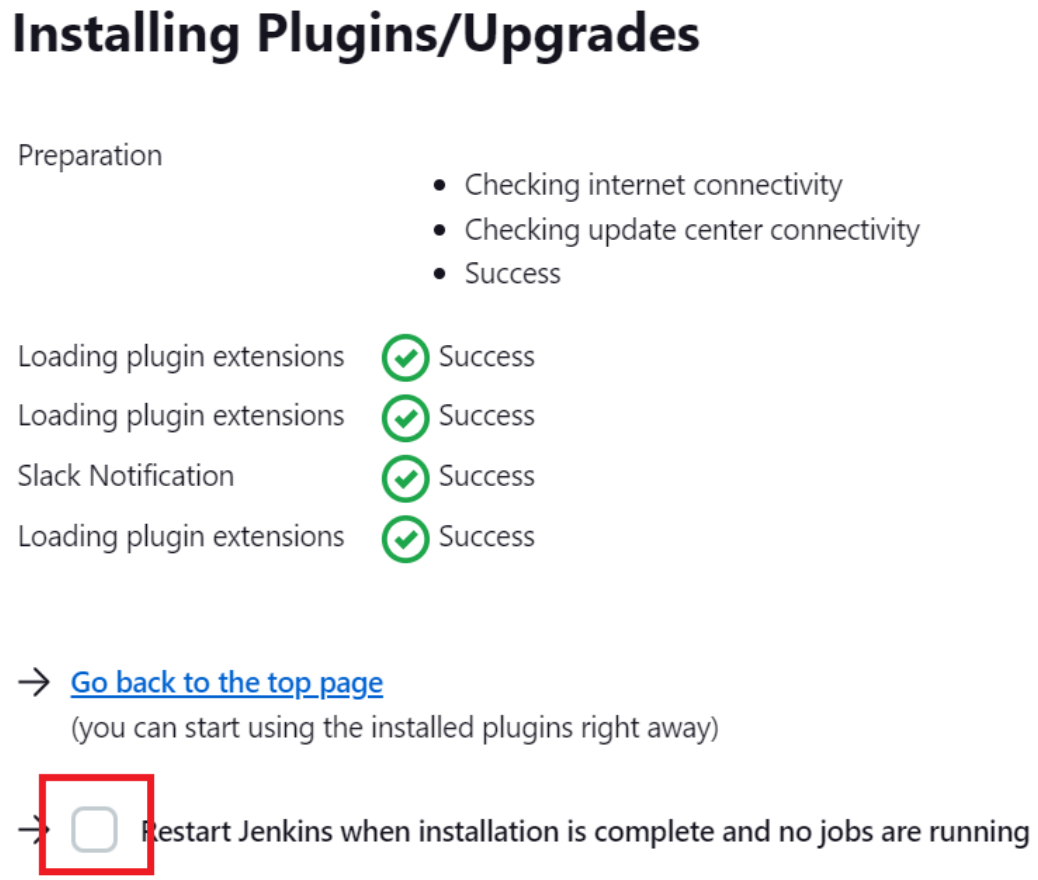
Open ***Manage Plugins***



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

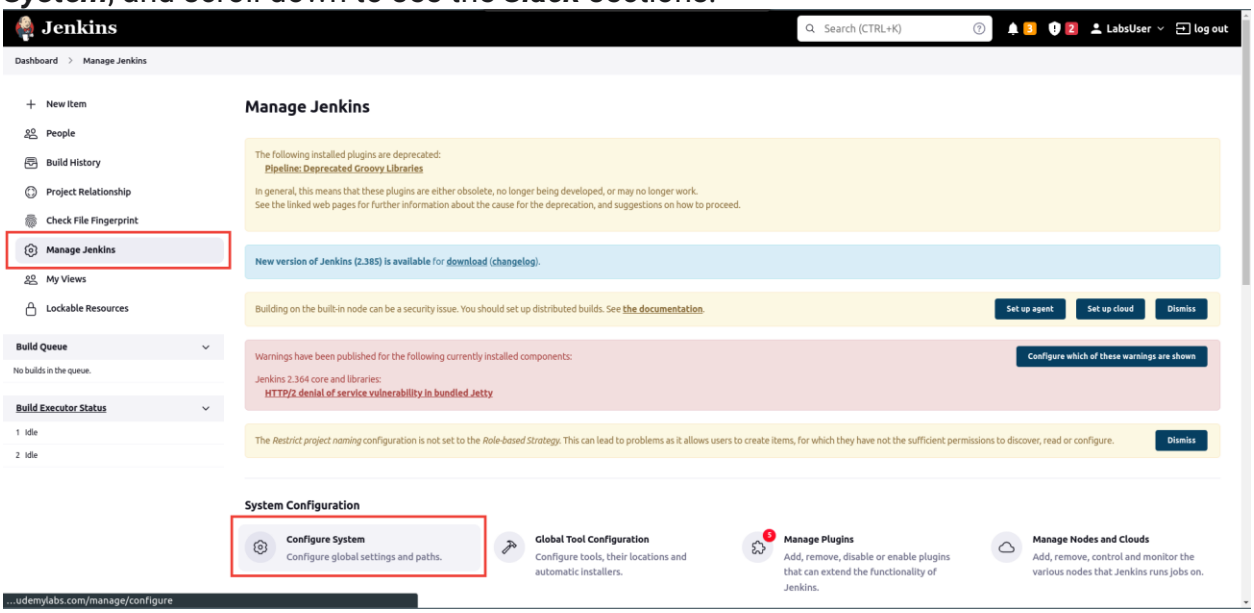


3. Select the checkbox at the bottom of the page to restart Jenkins.



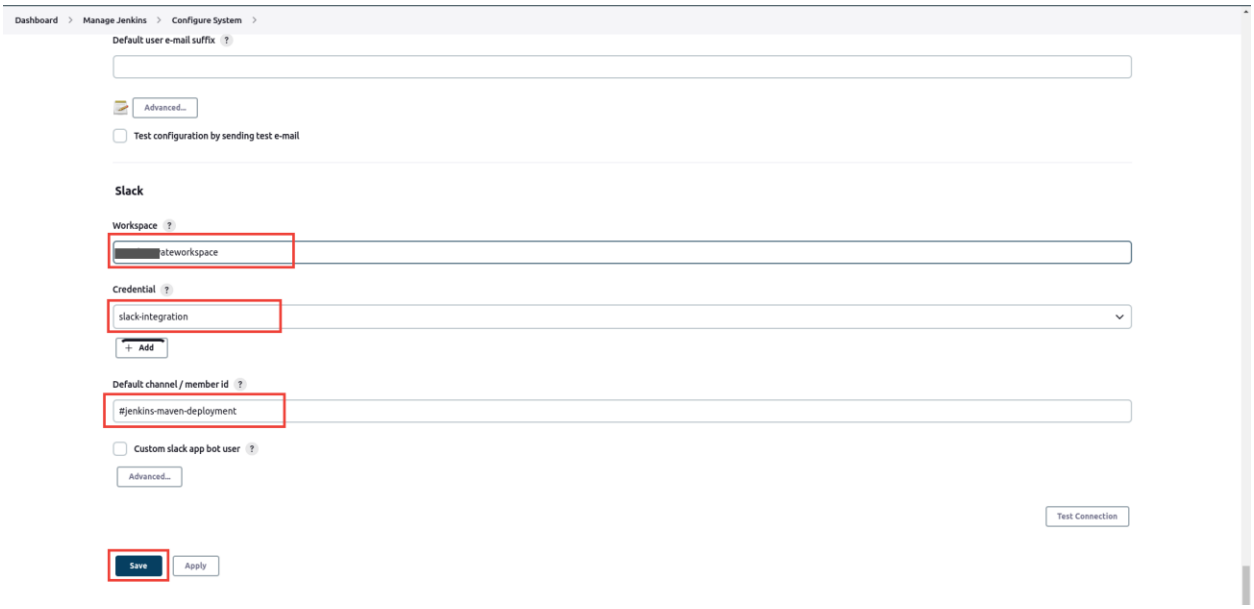
Wait for Jenkins to restart. Refresh your browser if you are not redirected to Jenkins in 2 minutes.

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



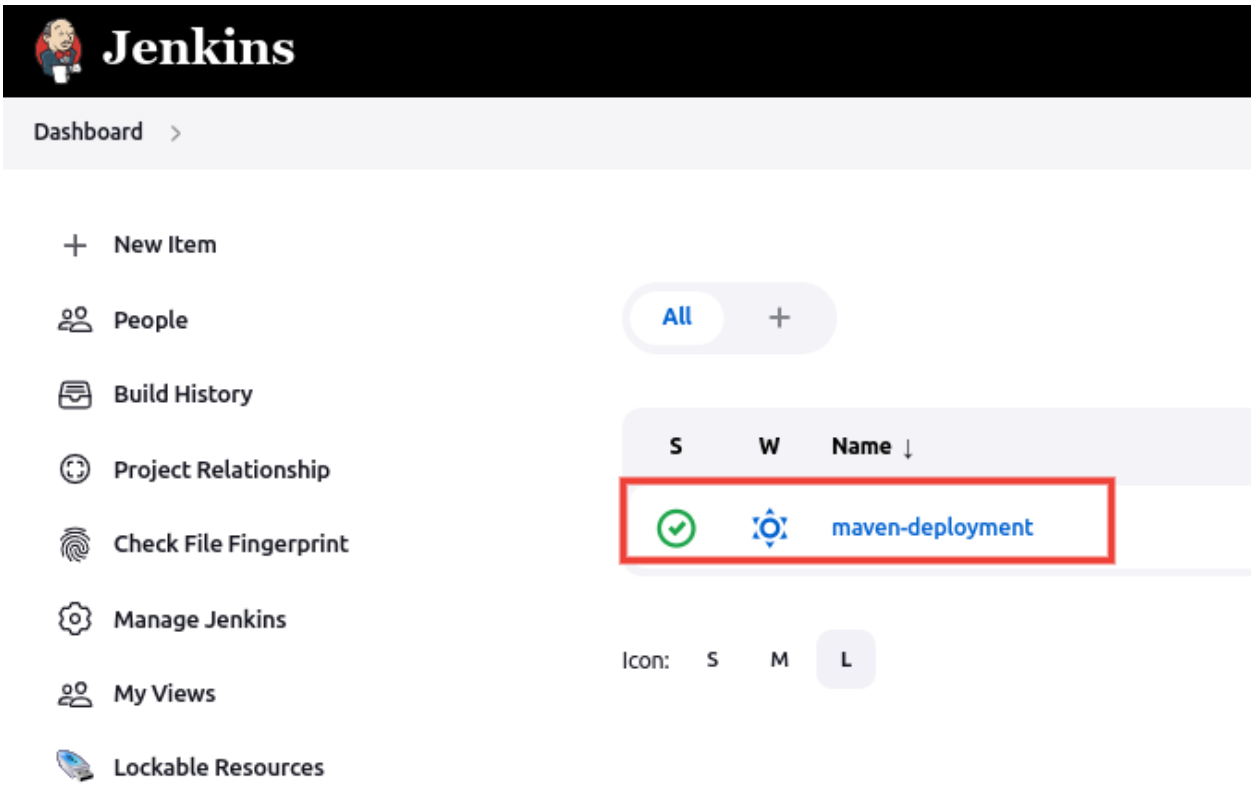
Fill the details of your Slack configurations:

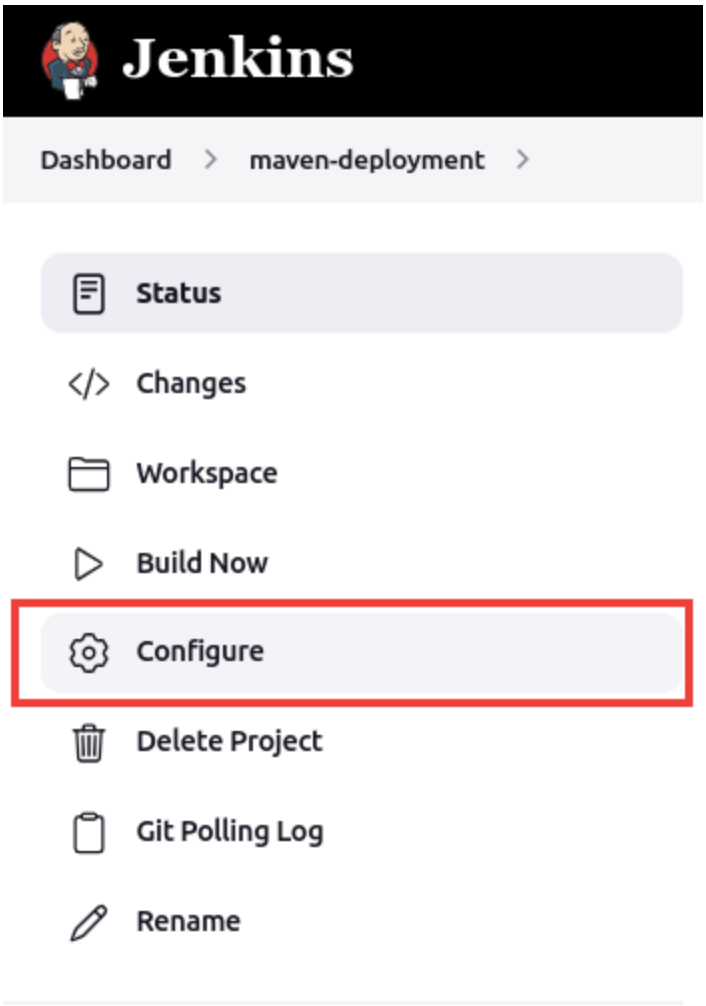
a. For workspace use the saved ***Team subdomain*** from the previous steps
b. For credential, type in ***slack-integration***
c. Select channel name to send the notification
d. Save it



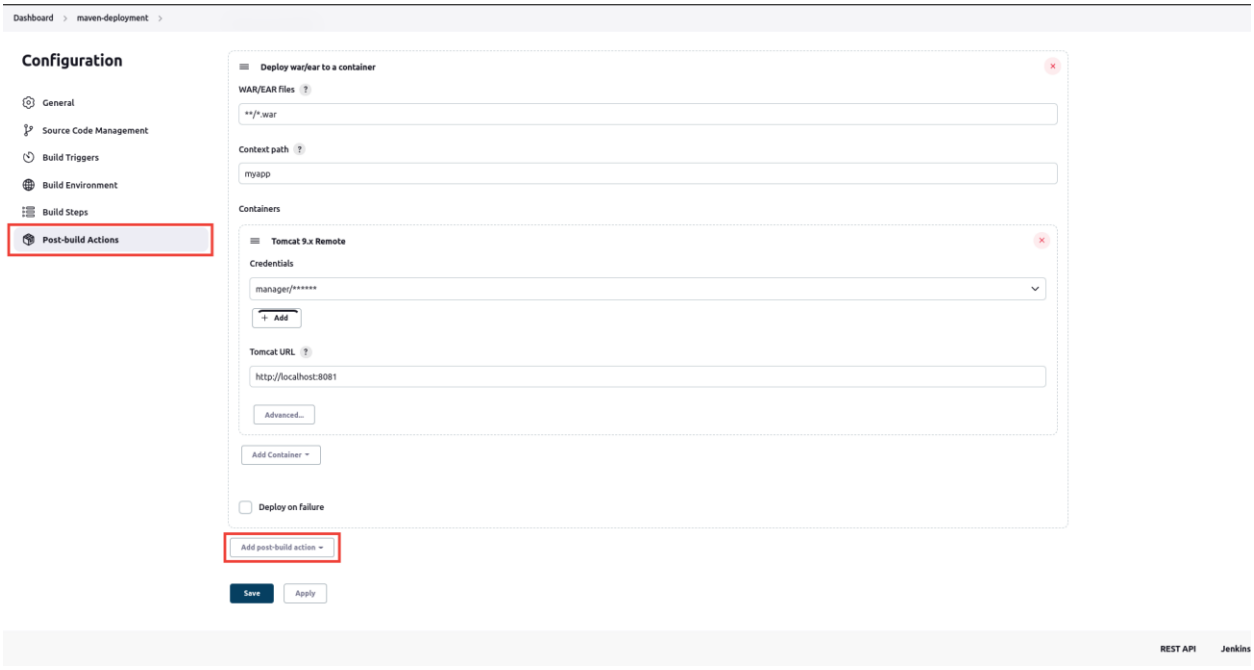
Save your configurations.

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

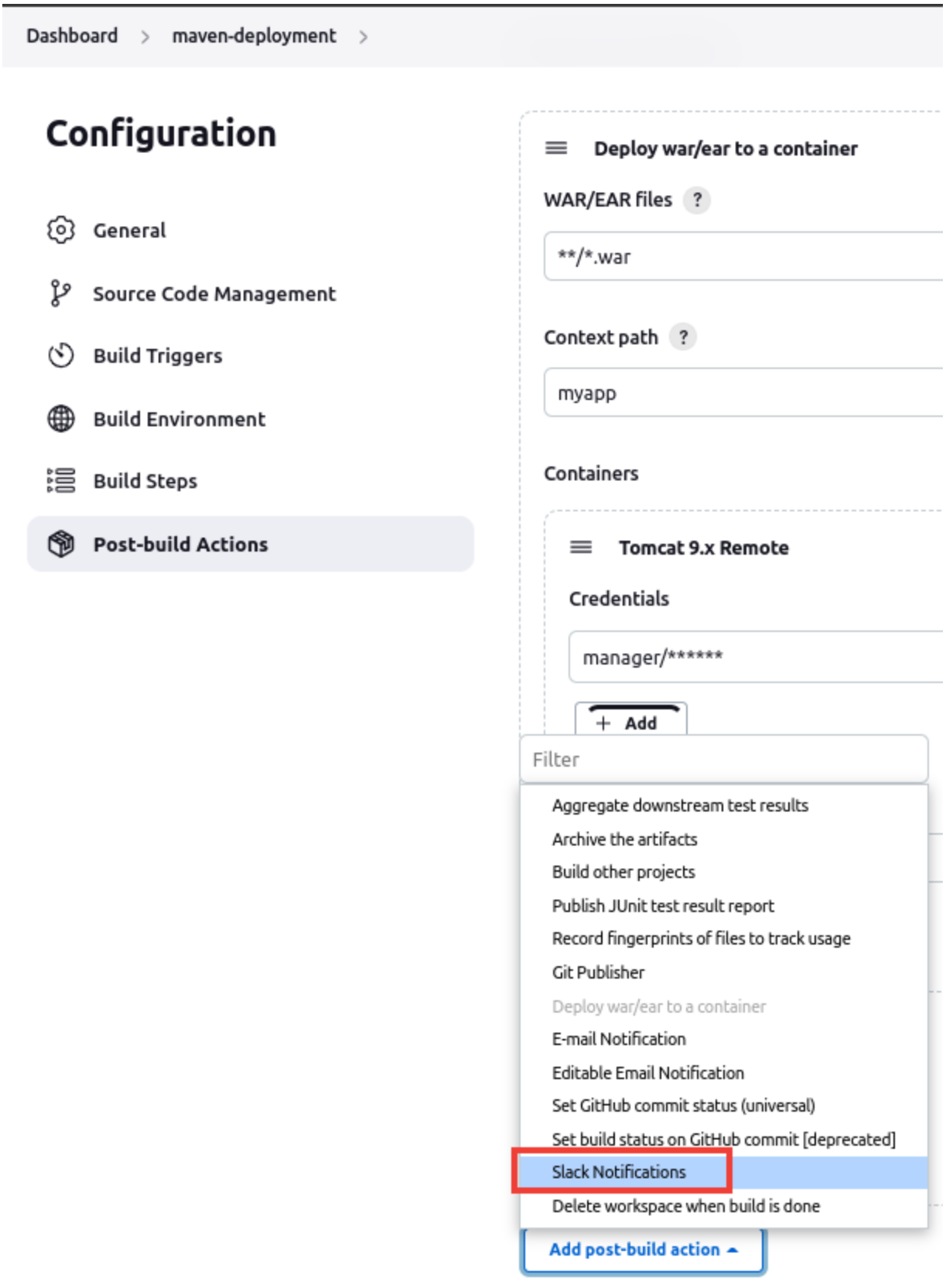

Jenkins
Dashboard
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Lockable Resources
All
S
W
Name
maven-deployment
Icon:
S
M
L



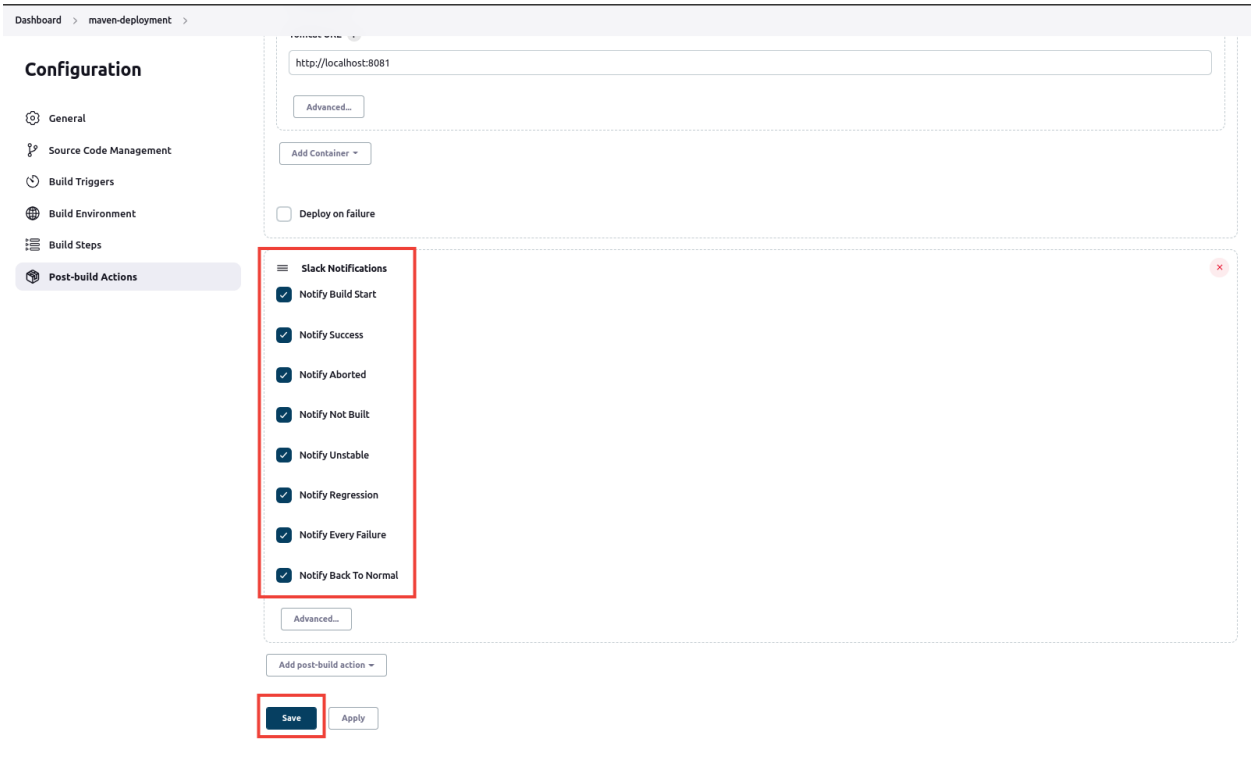
Go to ***Post-Build Actions*** and select **Slack Notifications** from the ***Add post-build action*** dropdown menu.


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
Tomcat URL
http://localhost:8081
Advanced...
Add Container
Deploy on failure
Add post-build action
Save
Apply
REST API
Jenkins

=>


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
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:



Save your configuration.

Resources

Documentation

- Jenkins Slack Plugin

Mark task as complete
Report issue

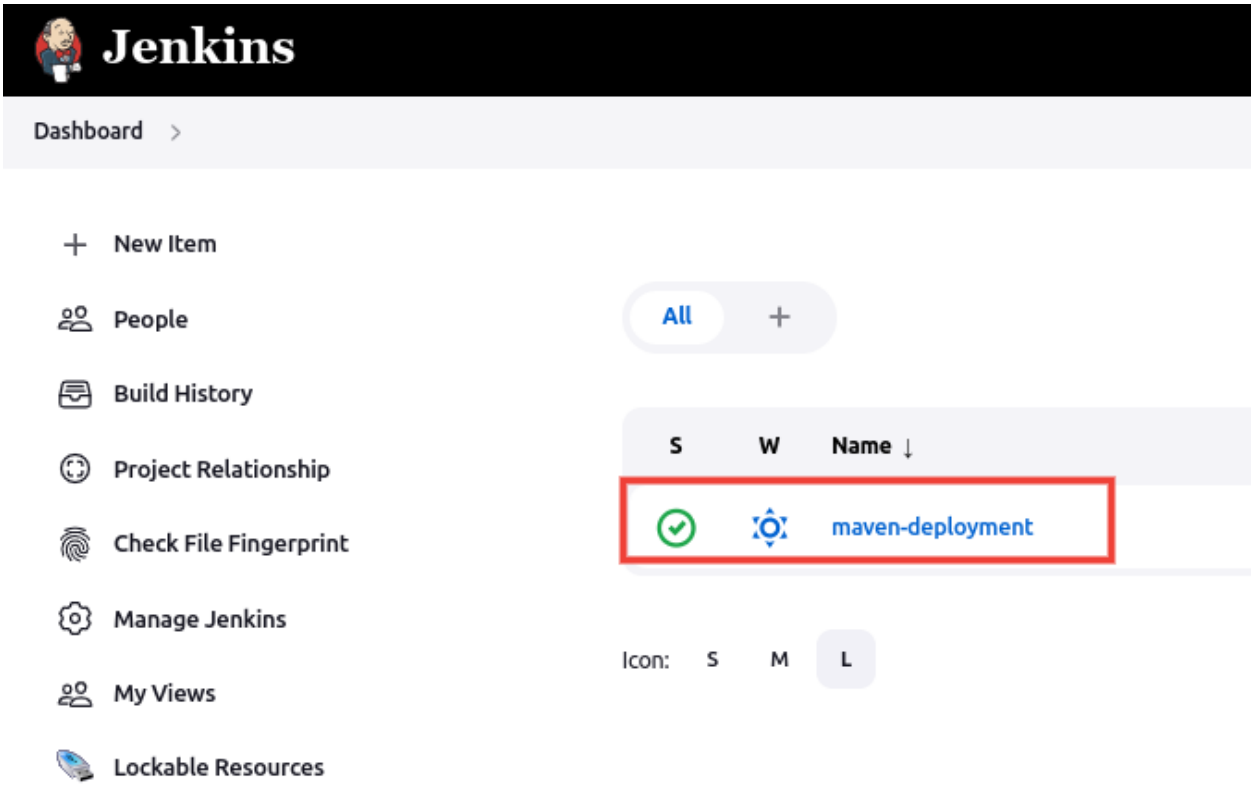
11. Test the Slack Notifications

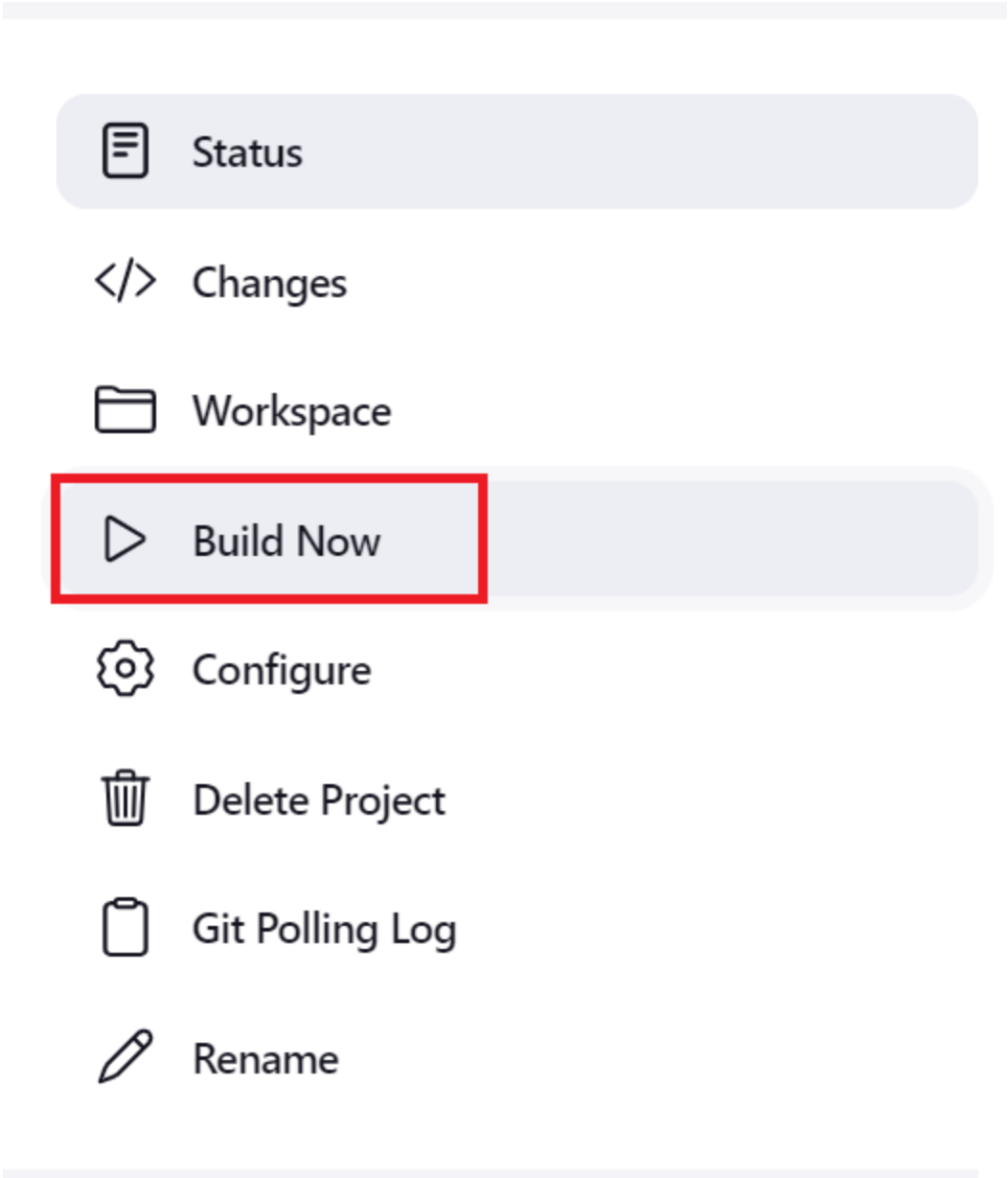
Test the Jenkins-Slack integration for:

- builds that failed
- builds that succeeded

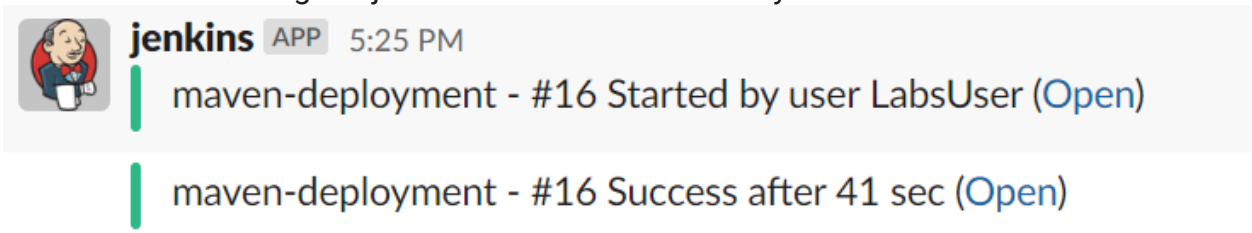
Notifications must be sent to your Slack channel

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.

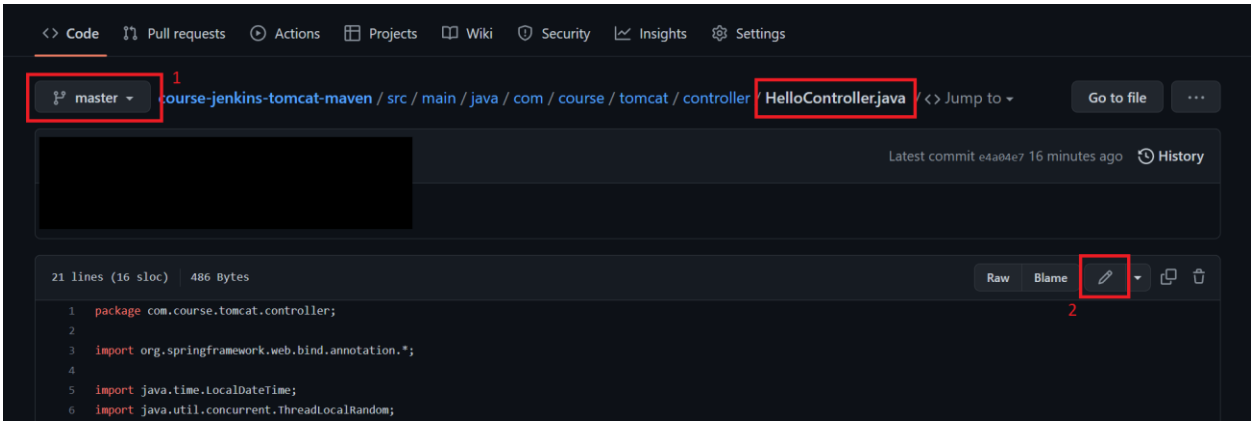


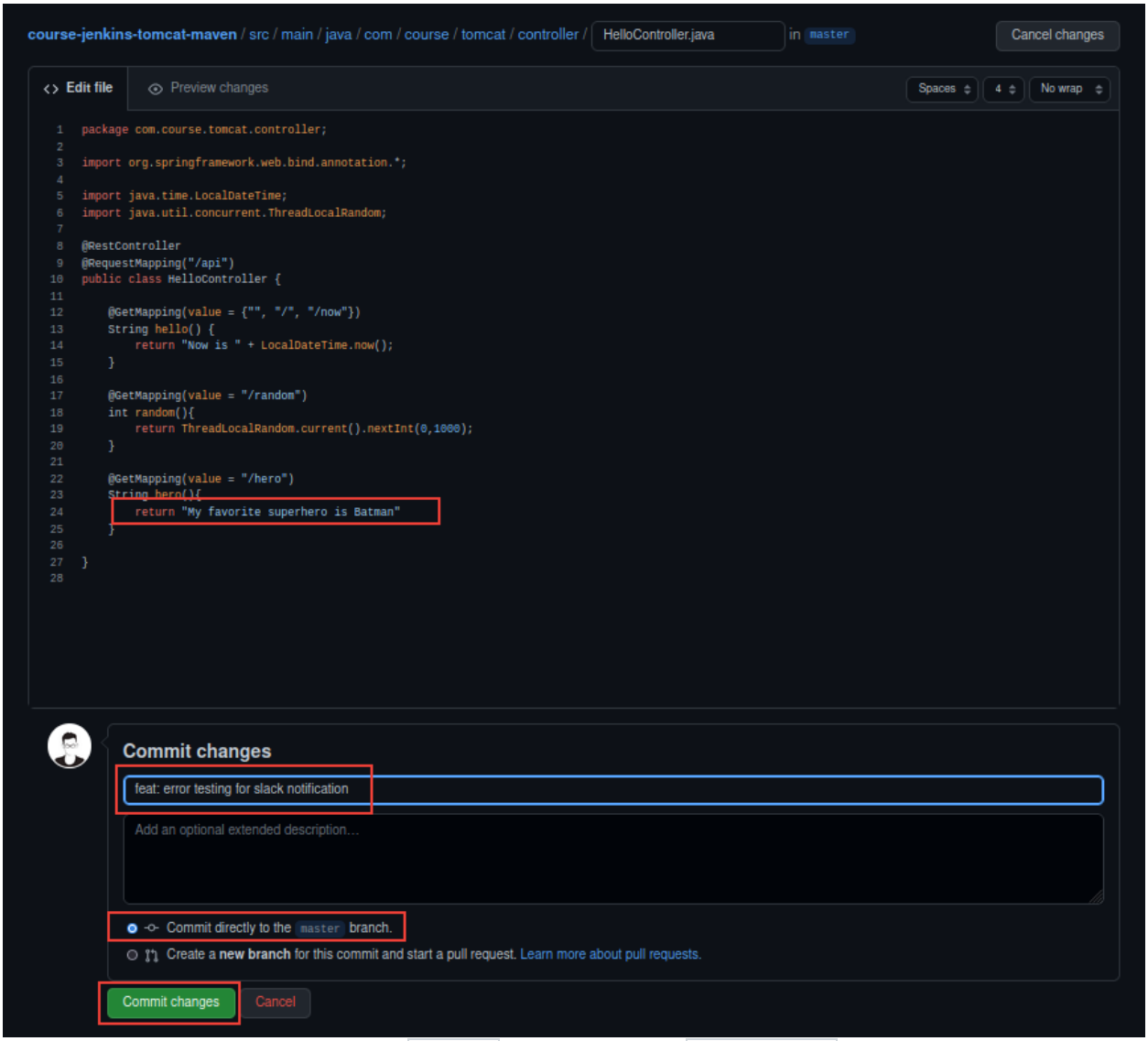


2. Wait until the build is finished and check your Slack channel. You will have a notification indicating the job was finished successfully.

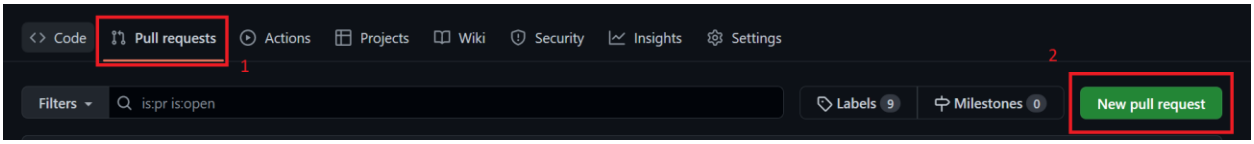


3. Go to your forked repository file ***src/main/java/com/course/tomcat/controller/HelloController.java*** and remove a ";" character from your code that is in the `master` branch. This will cause a build error later.

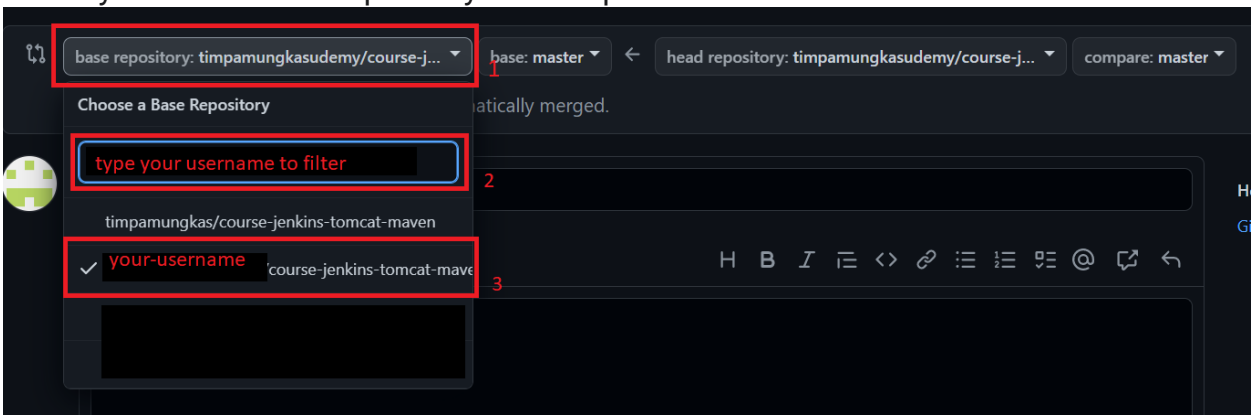




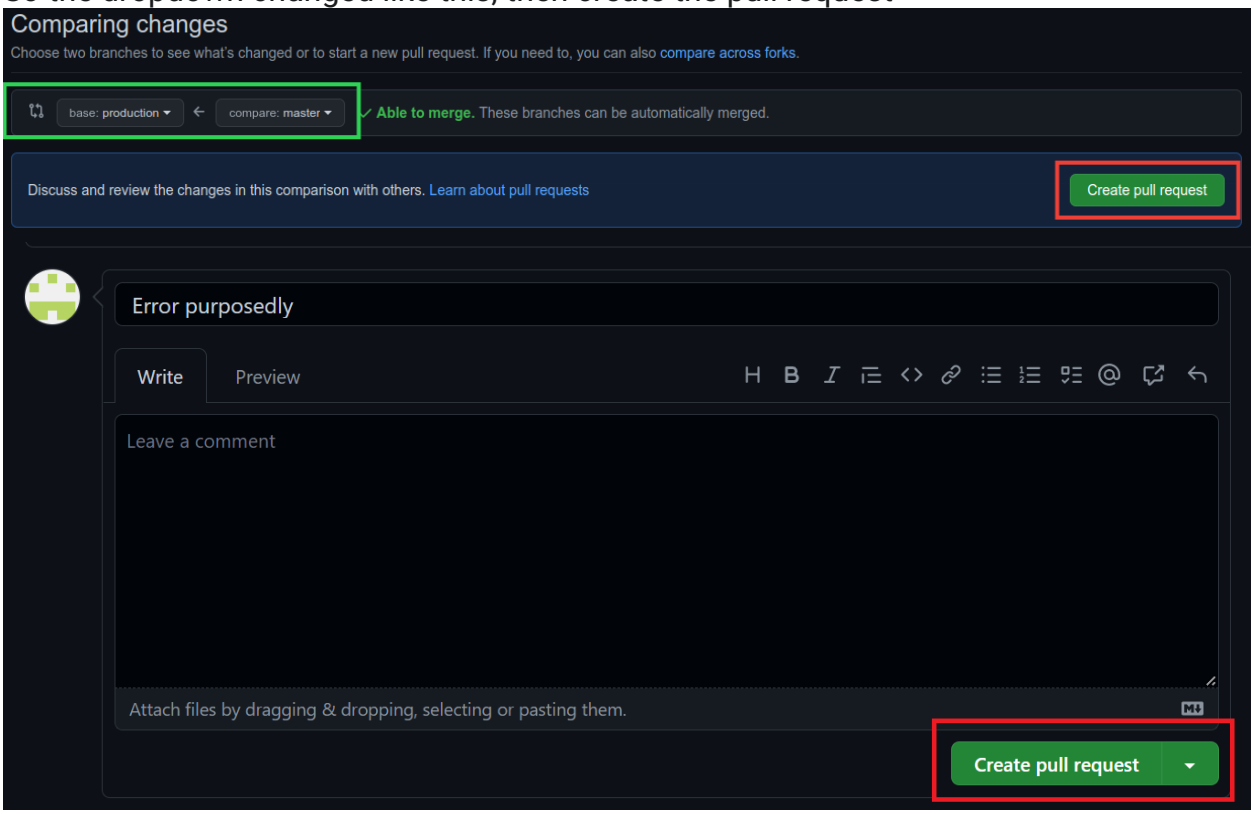
4. Create a Pull Request from the `master` branch to the `production` branch.



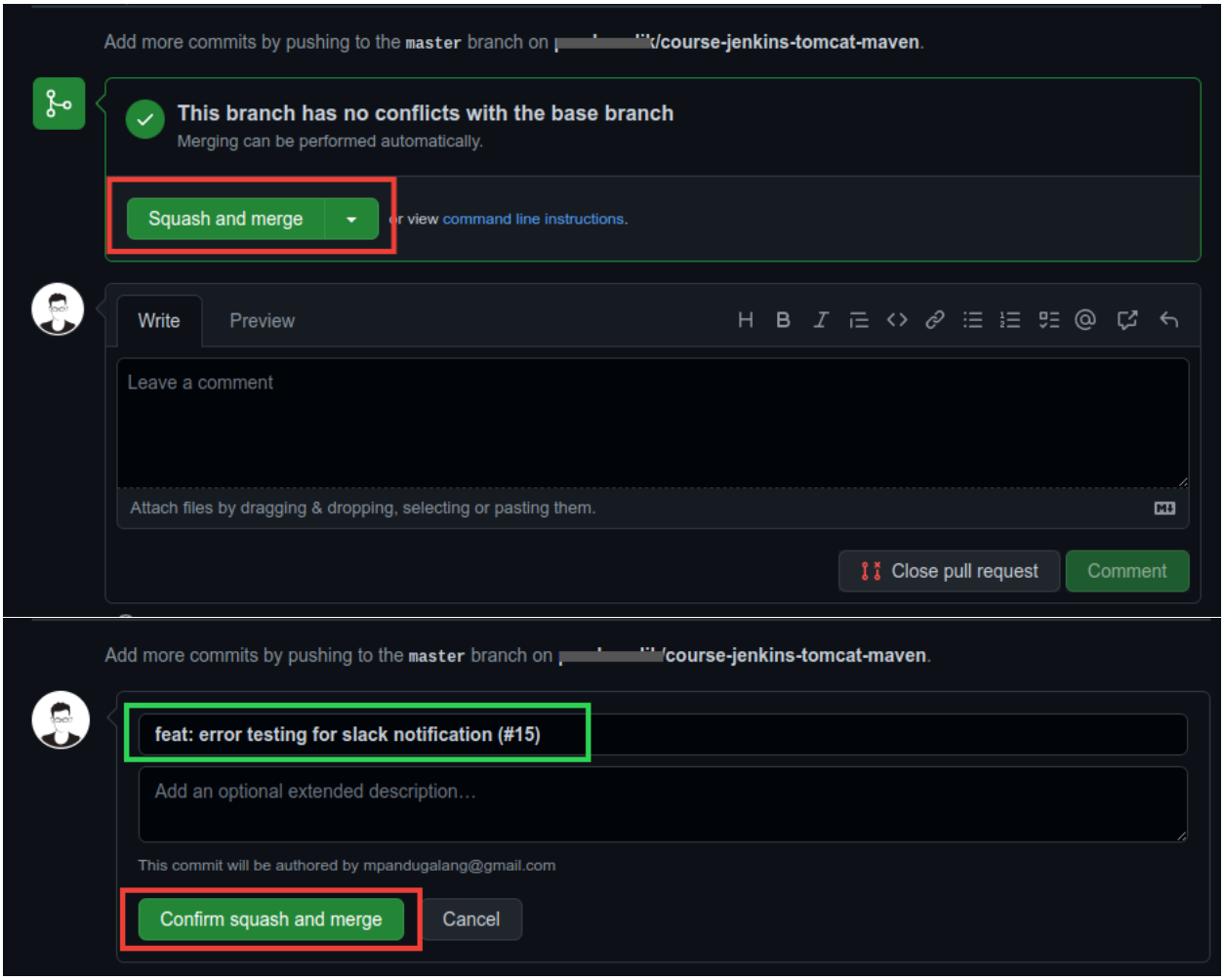
Select your own forked repositoryfrom dropdown.



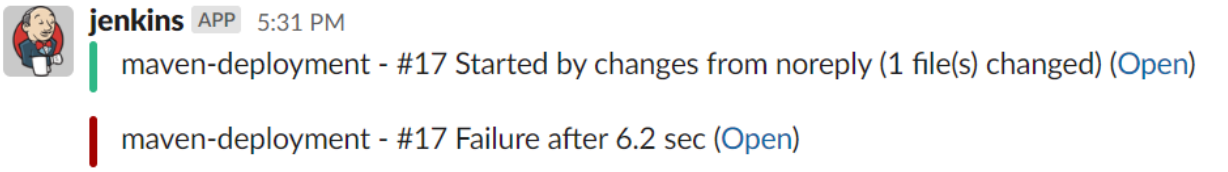
So the dropdown changed like this, then create the pull request



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



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