# Intro to Jenkins
Lab 3

1   Lab setup

We will be working with the Tomcat server in addition to the Jenkins Server in this lab.
Apache Tomcat is a servlet container which is used to run servlet and JavaServer Pages (JSP) web applications, which is part of the J2EE specifications. Most modern Java web frameworks (for e.g. Spring, JavaServer Faces, Struts) are based on servlet technology. Tomcat can also be used as a web server, although it is typically deployed inside more conventional and well known web servers such as Apache or Nginx

Ensure that you can access the Jenkins UI in the usual manner.

Next, verify that you can access the main Tomcat UI at your designated server:

http://server-ip-address:port-number
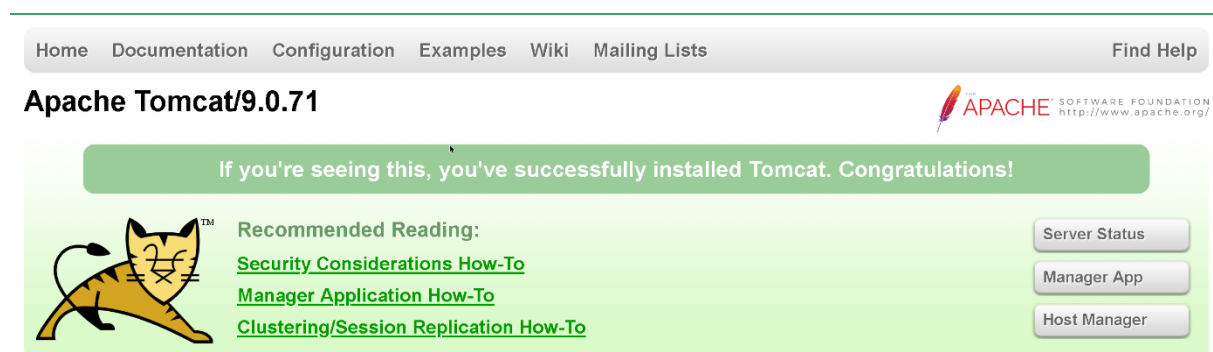
Click on Manager App and login with the provided username / passwd combination. After successful login, the main UI should look something like this:

In all the labs that follow, *userx* refers to your assigned username in the lab guide.

2 Building and deploying a Java web app with Maven

On the Bash shell in the Main Server, navigate to the project folder with the code base for building a basic java web app:

```
cd ~/userx/githubprojects/basic-java-web-app
```

The first thing we will typically do in a standard project is to clean up the artifacts (classes, JARs, etc) from a previous build operation. To do that we can use the `clean` phase from the `clean` life cycle. Type:

```
mvn clean
```

As before in the case of `simple-java-app`, if this is the first time you are running Maven, a whole bunch of related plugins and dependencies will be downloaded from the central Maven repository and stored in the local cache. This may take some time to complete, depending on the capacity of your broadband connection. Once they are available in the local cache, Maven will reference them there for future executions of this phase, thus removing the need for costly network access.

This project contains a single file `index.jsp` which will render the single page for this web app with some very basic HTML.

```
webapp/src/main/webapp/index.jsp
```
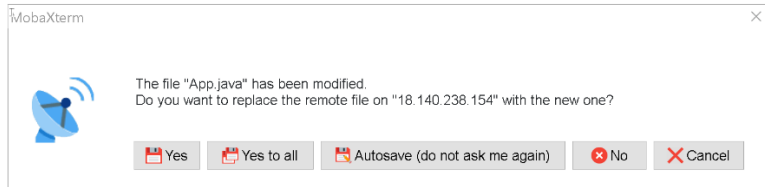
As usual, we can use MobaXTerm's default editor to open it for editing:
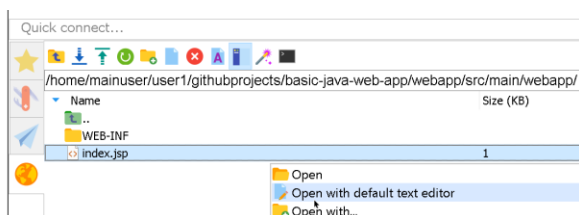
Change the text between the tags `<h2>` `</h2>` to include your name (to distinguish your app from the apps of all other participants later when you all deploy it on the same Tomcat server):

```
<h2>Interesting things to do for today, Lee Chong Wei ! </h2>
```

Save the changes (Ctrl+S) and select Yes to the prompt that appears.

You can verify that the changes have actually been made by directly viewing the content of this file from the CLI:

```
cat webapp/src/main/webapp/index.jsp
```

Now we can produce a build artifact (a WAR file) that we can subsequently deploy to a servlet container such as Tomcat. To accomplish this, we can type:

```
mvn clean package
```

The final part of messages from the Maven build console output shows the generation of the WAR artifact (`webapp/target/webapp.war`). This WAR artifact contains the Java web app (in the form of a servlet) which is now ready to be deployed into Tomcat.

To deploy this WAR via the Tomcat Manager UI, we will first download the WAR to our local Windows machines using MobaXTerm. Navigate to the directory holding this WAR file and select Download.
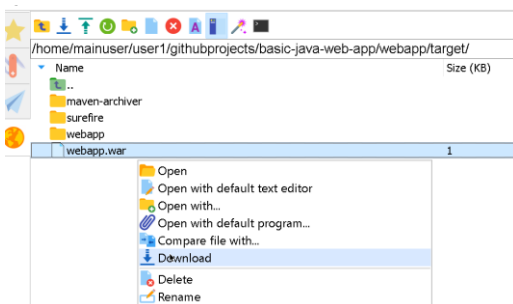
Download it to a folder or location on your Windows machine that you can easily navigate to, for e.g. Desktop.

Rename the file `webapp.war` to *userx*`.war`. This is important to distinguish your WAR file from the those of the other users and prevent conflicts when everyone deploys to the same Tomcat server.

In the Tomcat Manager UI, choose the WAR file to deploy, then navigate to the folder containing *userx*`.war` in the file dialog box and then select it

WAR file to deploy

Select WAR file to upload [Choose File] No file chosen

[Deploy]

Once deployed, you should see its path in the list of applications. The name of the path should match the name of the WAR file.

/user1	None specified	Webapp	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Click on the path to navigate to the app

Welcome to the coolest Tomcat webapp ever !

Interesting things to do for today

- Learn JavaScript
- Learn Python
- Learn Java

Go back to the Manager UI and Stop and Undeploy this app

Start | Stop | Reload | Undeploy

Expire sessions with idle ≥ 30 minutes

It should now disappear from the list of applications on Tomcat Manager UI.

3   Creating a GitHub repo for the Java web app project

Login to your GitHub account.
Create a new repository and give it the same name as the Java project that we have been working with so far: `basic-java-web-app`

Leave it as a public repository (default setting)

🔘 🖥 **Public**
Anyone on the internet can see this repository. You choose who can commit.

⚪ 🔒 **Private**
You choose who can see and commit to this repository.

DO NOT initialize the repository with anything. Simply click Create.

Back in the Bash shell, make sure you are in the folder `userx/githubprojects/basic-java-web-app`

If not, you can change to it with:

```
cd ~/userx/githubprojects/basic-java-web-app
```

First, we remove any previously existing generated artifacts

```
mvn clean
```

Git is already installed on the server. First, we check that this folder is not a proper Git repo yet with

```
git status
```

An error should indicate that this project folder is not yet a Git repo. If it is, you can simply delete the `.git` folder in the project folder.

Next initialize it as a Git repo with all the files in the project folder with the following sequence of command in the Git Bash shell (you can type them in one at a time):

```
git init
git add .
git status
```

Before creating the first commit in this newly initialized repository, you need provide an identify for the individual creating the commit (the commit author). Since we are going to be pushing our local repo to populate the GitHub repo, we will use our GitHub identity (name and email). Go to

https://github.com/settings/profile

Public profile

Name

Victor Tan

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

victor.tan.33@gmail.com   ✕ Remove

You can manage verified email addresses in your email settings.

and note down the name and email address there. Use them in the configuration commands below:

```
git config --local user.name "your name"
```

```
git config --local user.email "your email address"
```

We are configuring these details at the local level so that they apply only to this repository, rather than to the user account since all users are now using this user account `mainuser.`

Then we can create our first commit.

```
git commit -m "Added all project files in first commit"
git status
```

Return to the main page of your new GitHub repo that you just created. Copy the HTTPS URL shown there:

We will refer to this URL as `remote-url` in the commands to follow.

Back in the Git Bash shell in `basic-java-web-app`, and type:

```
git remote add origin remote-url
```

Check that the origin handle is set to point to the correct repo URL with:

```
git remote --verbose
```

We can also optionally configure auto color scheme for the Git Bash shell

```
git config --global color.ui auto
```

As usual, in order to authenticate the newly initialized repo to your remote GitHub repo, you need to first obtain a classic personal access token (PAT). We can still keep using the PAT we generated from the previous lab, but if you have lost access to that, then just generate a new PAT following the instructions from the previous lab.

Finally, push the entire contents of the local repo (including all its branches) to the remote repo with:

```
git push -u origin --all
```

You will be prompted for your username and password:

```
Username for 'https://github.com': yourusername
Password for 'https://githubaccountname@github.com': PAT
```

Type in the PAT at the password section. You can copy and paste the PAT, but be very careful to do this slowly and correctly.

Assuming you have entered the correct GitHub user account name and PAT, the command should succeed with appropriate success messages in the Git Bash shell.

```
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
….
…
branch 'master' set up to track 'origin/master'.
```

Refresh the main repo page, you should see the contents that you have just pushed up.
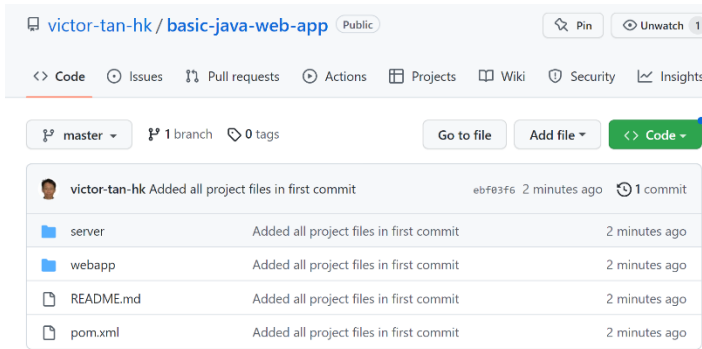
Back in the Bash shell, type the following commands to verify that the remote tracking branches have been set up and that the local and upstream master are both in sync with each other.

```
git remote show origin

git status
```

Other users (for e.g. team members of a project team) can now clone this remote repo to a local repo on their local machines and work on it via a branching workflow.

4   Set up credentials and plugin to integrate with Tomcat container

As we have mentioned before in a previous lab, plugins are key to the functionality of Jenkins. When a plugin is installed, it provides functionality to Jenkins by adding available steps that can be used in the actions / options to be selected in the key stages of a standard free style project (General, SCM, Build Triggers, Build Environment, Build Steps, Post Build Actions). These steps / options represent a fundamental unit of action that is performed by Jenkins.

The action below of installing a plugin only needs to be performed once by a user with Admin Privilege.

In Jenkins main dashboard, go to Manage Jenkins -> Manage Plugins, and in Available Plugins search for Deploy to Container
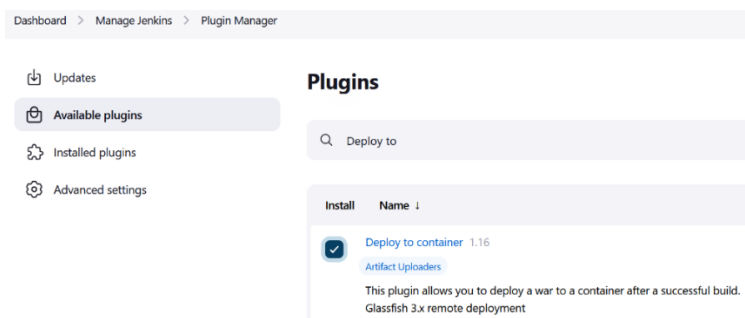
Select Install without restart.

After the Download progress is complete, select Restart Jenkins when installation is complete and no jobs are running. Wait for Jenkins to restart.

Please wait while Jenkins is restarting .

Your browser will reload automatically when Jenkins is ready.

The restart may take some time to complete or may occasionally fail. In Linux, you can start / stop the Jenkins service from the CLI via:

```
sudo systemctl start / stop jenkins
```

You can check the status of the Jenkins service using the command:

```
sudo systemctl status jenkins
```

You may have to log back in again once the restart is complete.

The action below of setting up a Global Credential only needs to be performed once by a user with Admin Privilege.

In Jenkins main dashboard, go to Manage Jenkins - Manage Credentials -> System (or Jenkins) -> Global Credentials (Unrestricted), and then Add Credentials.
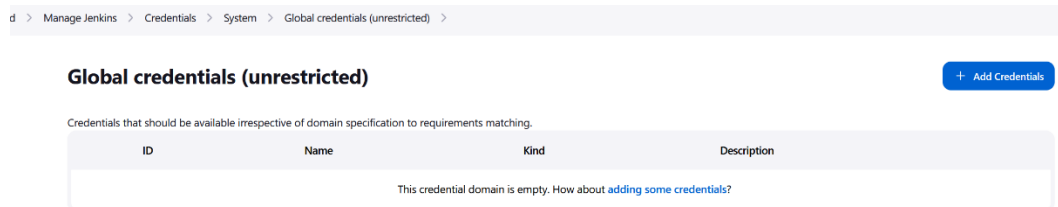
Here we can configure any user that has the access rights to deploy a WAR file directly into the Tomcat server. We can use the admin user account, and we will configure that username / password combination here. We will provide an ID to identify this username / password combination: `tomcat-deployer`

Click Create.

As this credential is a global credential, its use is unrestricted and any job created by any valid user registered in the Jenkins server can it for purposes of authenticating to the Tomcat server.

5   Basic Jenkins job to build and deploy a Java Web app

From the main Jenkins dashboard, create a new Item and name it: `userx-Basic-JavaWebApp`
Make this a Freestyle Project and select Ok.

In the Configure section, go to Source Code Management and select Git

Return to the main page of your GitHub repo, click on the Green Code button and copy the HTTPs URL:
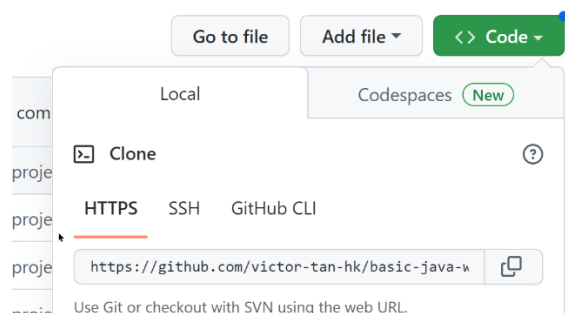
Paste the URL into the Repository URL on the Jenkins configuration page.

Click on Save and perform a build.

The Console Output for this build attempt will show that the remote repo contents have been cloned into the workspace folder for this current job. You can navigate to this folder using the Workspace item in the navigation pane of the main area for the job to verify this.

In the main area for the job, select Configure, go to Build Steps and add a Build Step -> Invoke Top Level Maven Targets:

Select the Maven Version that you configured previously and add the goals: `clean package`

Click Save, then perform a Build. This will take a while to complete due to the downloading of all the Maven dependencies and plugins again.

The build should be a success, and the Console output shows the output from Maven, with the WAR artifact being generated successfully at the end.

We will add another Build step to rename the generated WAR `webapp.war` to *userx*`.war`, to avoid conflicts with other users when they deploy their WAR to the same Tomcat server at the same time.

Add an Execute Shell command:

```
mv webapp/target/webapp.war webapp/target/userx.war
```

Save and run another build. Navigate in the Workspace folder and verify that the WAR file in `webapp/target` has been renamed appropriately.

Finally, we will configure the job again by adding a Post-build action, and add Choose deploy war/ear to container.
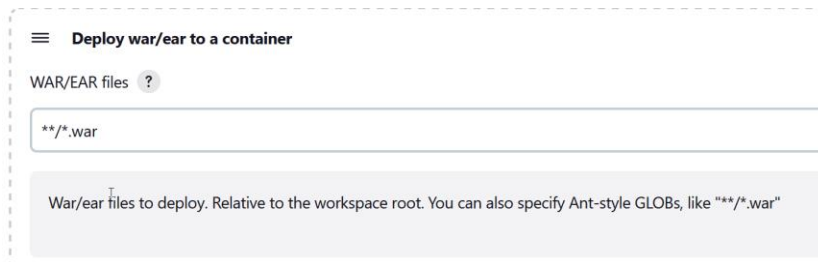
The actual generated WAR file in the workspace folder would be at: `webapp/target/userx.war`

We only need to specify the path relative from the workspace root folder, i.e.
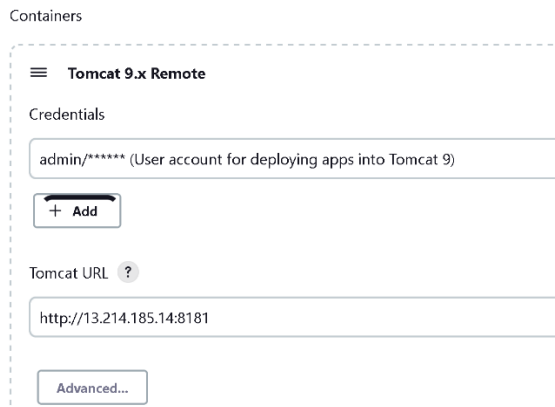
```
webapp/target/userx.war
```

We can also use an Ant-style GLOB to specify the location of the WAR file: `**/*.war`
https://www.bojankomazec.com/2019/07/apache-ant-patterns.html

For containers, select Tomcat 9.x remote

Select the credentials you added earlier and provide the complete URL to the Tomcat server, including the correct port number.

Click Save and run the build. Make sure your Tomcat server is up and running before this.

If you return to your Tomcat manager, you should see that the web app is now deployed successfully in the list of applications in the UI, and you should be able to click on the link to view it.

Once you have confirmed this, stop and undeploy it as we have done previously.

Return to the project folder `basic-java-web-app` in `githubprojects`
Let's make a minor modification to the HTML for the main page of the web app at:

```
webapp/src/main/webapp/index.jsp
```

Change the content of the <h1> header to some other random value, for e.g.

```
<h1>Welcome to the most boring Tomcat webapp ever ... boo </h1>
```

Save and exit.

In the Git Bash shell in this project folder, check that the modifications have being made:

```
git status
```

Add the modifications to a new commit with an appropriate message:

```
git commit -am "My first change in the local project folder"
```
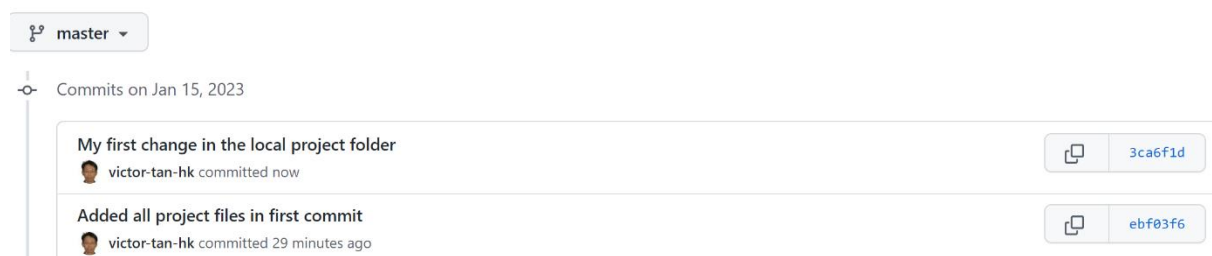
And if we check the status again

```
git status
```

We can see that the latest commit on the master branch is ahead of the remote master by one commit, as we expect.
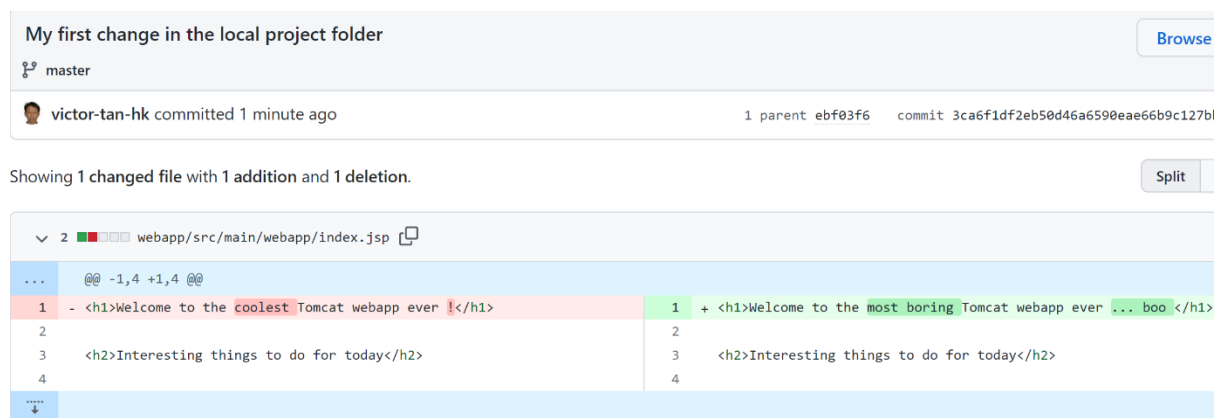Push this latest commit to the master on the remote GitHub repo with:

```
git push
```

Back on the GitHub repo main page, click on the Commits link to zoom in to the main commits page

Then click on the hash of the particular commit to see its details.

We can see the latest commit and how it has changed from the previous (initial commit)

Returning back to Jenkins, perform another build of the same job. The latest changes to the remote repo will pulled down to the local workspace folder, and a build will be run to generate a new build artifact. This will again be deployed in the Tomcat server, and you should be able to see the latest changes once you navigate to the app again.

Continue to make a few more changes on the local code base and push the commits to update the remote repo and run a build again.

As a matter of practice, you can configure tests, periodic builds and polling and email notification for this job as well, the same way that you have done for the previous basic Java job.

6 Basic pipeline to build and deploy a Java Web App

We will now replicate all the functionality that we accomplished previously in a freestyle project using a pipeline.

From the main Jenkins dashboard, create a new Item and name it: `userx-Pipeline-JavaWebApp`
Make this a Pipeline and select Ok.

Select the Pipeline section and copy and paste the script from: `pipeline-javawebapp`

Modify the script to specify the HTTPS URL for the GitHub repo for `basic-java-web-app` that you had created earlier:
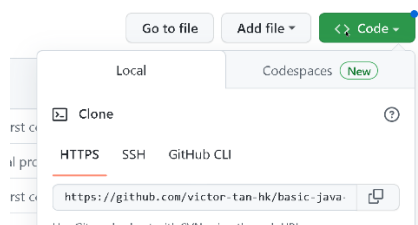
Also remember to
- Change `userx` to your assigned username
- Specify the Tomcat URL correctly in the `Deploy` stage

Note that the pipeline snippet corresponding to the `Deploy` stage could have been generated from the Pipeline Syntax Snippet generator, as we discussed earlier.
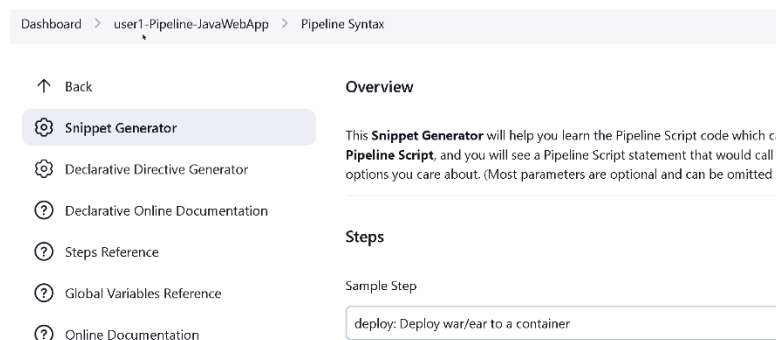
Before running a build on this pipeline, make sure you stop and undeploy all existing Java web apps on the destination Tomcat Server.

Now run a build and verify that your web app is now successfully deployed to the Tomcat server.

Lets make a few more simple changes in the project folder `basic-java-web-app` in `githubprojects`
Let's make a minor modification to the HTML for the main page of the web app at:

`webapp/src/main/webapp/index.jsp`

Change the content of the <h1> header to some other random value, for e.g.

`<h1>Welcome to an interesting Tomcat webapp ... cool </h1>`

Save and exit.

In the Git Bash shell in this project folder, check that the modifications have being made:

`git status`

Add the modifications to a new commit with an appropriate message:

`git commit -am "Pipeline related change in the local project folder"`

Push this latest commit to the master on the remote GitHub repo with:

`git push`

Run a build again on the pipeline job and verify that it has been deployed correctly on the associated path on the Tomcat server.

Now that you have a complete CI / CD pipeline set up, you can convert this pipeline script into a Jenkinsfile which you can place in the root of your project folder and check it into the GitHub repo, the same way that we have done in a previous lab.