Jenkins
Lab 2

1 Lab setup

Make sure you have a suitable text editor installed for your OS (e.g. Notepad++) and a suitable SSH / SCP client such as MobaXTerm.

You will continue using the lab code that you created in the `jenkinslabs` directory from the previous lab, as well as the remote code repos that you created on GitHub in your GitHub account, so don't delete this yet.

You will each be assigned a group of 2 AWS EC2 instances running a specific Linux distro. The name of these instances are:

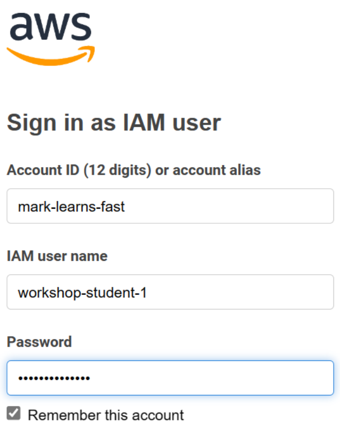
```
JenkinsServer-x
GeneralServer-x
```

where x is a number assigned to you in accordance to a specified listing.

You will also be given a private key file in the form of *`keyfilename`*`.pem`.
Create a subfolder `labkeys` in your top-level `jenkinslabs` directory, and store the *`keyfilename`*`.pem` file there.

You will also be provided with a URL to login to the AWS Console, as well as an individual username and password combination.

At the specified URL, login with your assigned username / PW combination



Sign in as IAM user

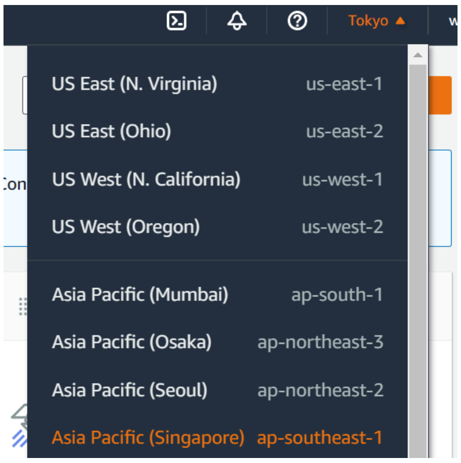
Account ID (12 digits) or account alias: mark-learns-fast

IAM user name: workshop-student-1

Password: ••••••••••••••

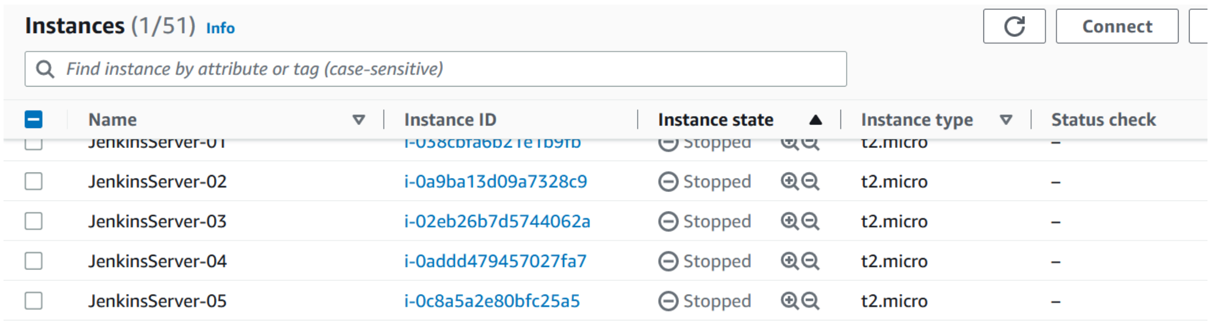
☑ Remember this account

Change your region (upper right hand corner) to `Asia Pacific (Singapore) ap-southeast-1`.



US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1

Navigate to the EC2 main dashboard (click on the symbol or search in the top navigation bar), then click on instances. You should see a list of all instances available in this region.



Instances (1/51) Info

Find instance by attribute or tag (case-sensitive)

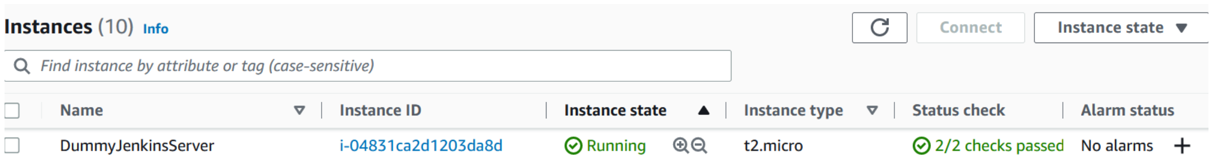
Name	Instance ID	Instance state	Instance type	Status check
JenkinsServer-01	i-038cbfa6b21e1b9fb	Stopped	t2.micro	–
JenkinsServer-02	i-0a9ba13d09a7328c9	Stopped	t2.micro	–
JenkinsServer-03	i-02eb26b7d5744062a	Stopped	t2.micro	–
JenkinsServer-04	i-0addd479457027fa7	Stopped	t2.micro	–
JenkinsServer-05	i-0c8a5a2e80bfc25a5	Stopped	t2.micro	–

In the subsequent lab sessions, you will be asked to start up the various EC2 instances one at a time. Please **ONLY** operate on EC2 instances assigned to you, to avoid interfering with the work of other students in the lab sessions.
DO NOT start the EC2 instances assigned to you until the particular lab session when they are required to avoid incurring additional costs on my credit card. For the same reason as well, avoid starting or using other AWS instances that are not mentioned in this lab manual.

Your cooperation is much appreciated to help with my high costs of living 😊

2 Install and configure Jenkins on a Linux machine

Start up the JenkinsServer instance and wait for it to transition to a running state. This should be indicated in the instance state and status check columns.

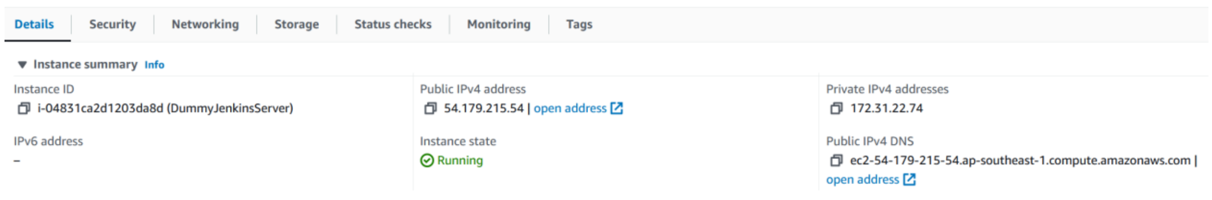


Instances (10) Info

Find instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
DummyJenkinsServer	i-04831ca2d1203da8d	Running	t2.micro	2/2 checks passed	No alarms

We will open an SSH terminal connection to a bash shell in this instance.
To do this, we will use either the public IPv4 address or Public IPv4 DNS of the running instance.
Select the instance and in the details tab below, note down either value in somewhere suitable where you can easily copy and paste it (for e.g. Notepad++)



Instance summary Info

Instance ID: i-04831ca2d1203da8d (DummyJenkinsServer)
Public IPv4 address: 54.179.215.54 | open address
Private IPv4 addresses: 172.31.22.74
IPv6 address: –
Instance state: Running
Public IPv4 DNS: ec2-54-179-215-54.ap-southeast-1.compute.amazonaws.com | open address

In the subfolder `labkeys` in your top-level `jenkinslabs` directory where you earlier stored your *.pem file, open a Windows Powershell.

To create the SSH connection using the private key in the *`keyfilename`*`.pem` file, the command format is:

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

Put this down somewhere in Notepad++ so you can easily and quickly copy and paste it for frequent use. Paste this into Powershell and press enter to attempt the SSH connection.

You may encounter an error message similar to the following:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for './DevOpsProjectKey.pem' are too open.
It is required that your private key files are NOT accessible by
others.
This private key will be ignored.
```

To resolve this problem, you will need to modify the security permissions on the *.pem file to make it as restrictive as possible (for e.g. only accessible to the current active Windows user).

There are two ways to do this:

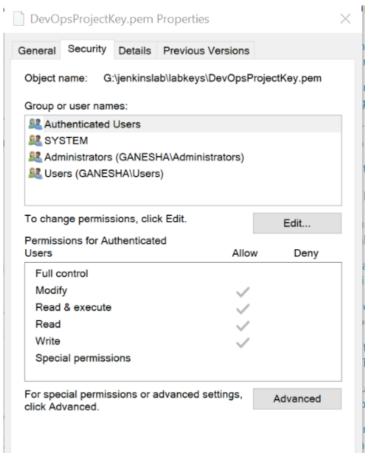
Approach 1:

Open a Git Bash shell in `labkeys`
Type:
```
chmod 0400 keyfilename.pem
```

Then attempt the connection again with:
```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

Approach 2:

In File Explorer, right click on the file -> Properties, go Security Tab. Click on Advanced at the bottom.



DevOpsProjectKey.pem Properties

General | Security | Details | Previous Versions

Object name: G:\jenkinslab\labkeys\DevOpsProjectKey.pem

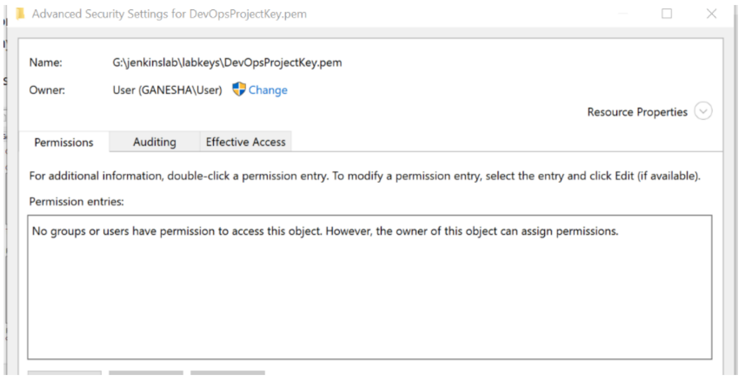
Group or user names:
- Authenticated Users
- SYSTEM
- Administrators (GANESHA\Administrators)
- Users (GANESHA\Users)

To change permissions, click Edit. [Edit...]

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced. [Advanced]

IN the next dialog box, select Disable Inheritance.
Select Remove all inherited permissions from this object in the dialog box.
The permissions entries box will now be empty.



Advanced Security Settings for DevOpsProjectKey.pem

Name: G:\jenkinslab\labkeys\DevOpsProjectKey.pem
Owner: User (GANESHA\User) Change

Resource Properties

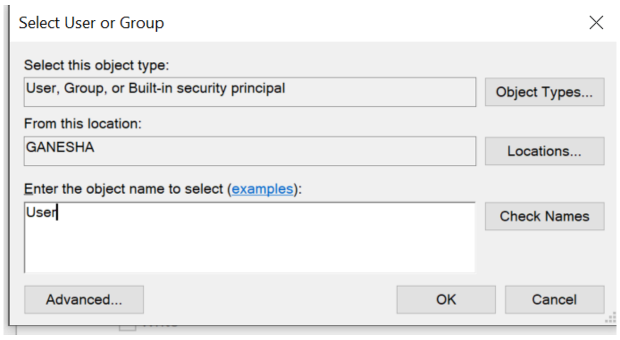
Permissions | Auditing | Effective Access

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Permission entries:

No groups or users have permission to access this object. However, the owner of this object can assign permissions.

Click Add. In the next dialog box, select a principal. In the Select User or Group, type your Windows username.

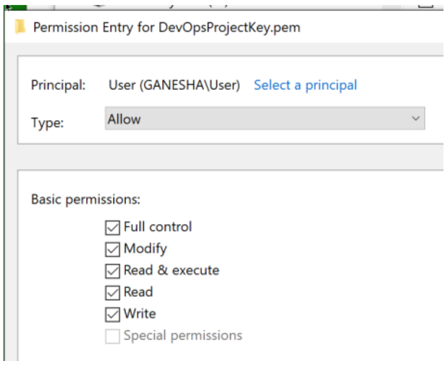

Select User or Group

Select this object type:
User, Group, or Built-in security principal

From this location:
GANESHA

Enter the object name to select (examples):
User

Click Ok. Then click on all the boxes under Basic Permissions:

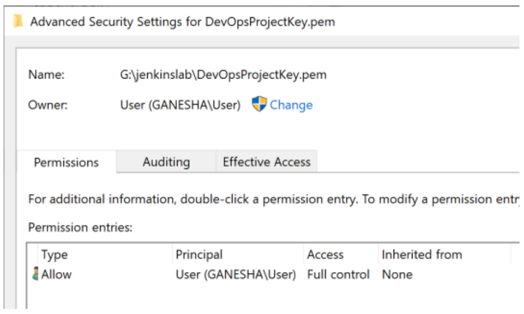

Permission Entry for DevOpsProjectKey.pem

Principal: User (GANESHA\User) Select a principal

Type: Allow

Basic permissions:
- [✓] Full control
- [✓] Modify
- [✓] Read & execute
- [✓] Read
- [✓] Write
- [ ] Special permissions

Then click Ok.
In the final Security Settings, you should see your Windows username as the principal in the Permission entries section, with full control access.


Advanced Security Settings for DevOpsProjectKey.pem

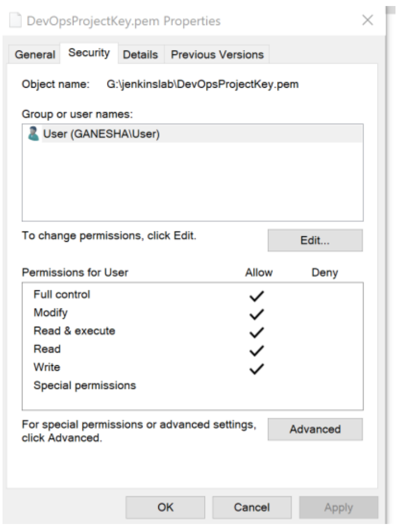
Name: G:\jenkinslab\DevOpsProjectKey.pem
Owner: User (GANESHA\User) Change

For additional information, double-click a permission entry. To modify a permission entr

Permission entries:

Type	Principal	Access	Inherited from
Allow	User (GANESHA\User)	Full control	None

Click Apply and Ok. The final Security tab should also only show your username with complete permissions for the file.


DevOpsProjectKey.pem Properties

General | Security | Details | Previous Versions

Object name: G:\jenkinslab\DevOpsProjectKey.pem

Group or user names:
User (GANESHA\User)

To change permissions, click Edit. Edit...

Permissions for User	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced. Advanced

OK Cancel Apply

Click Ok and attempt the connection again in Powershell with:

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

The initial Bash prompt upon a successful SSH connection established will look something like:

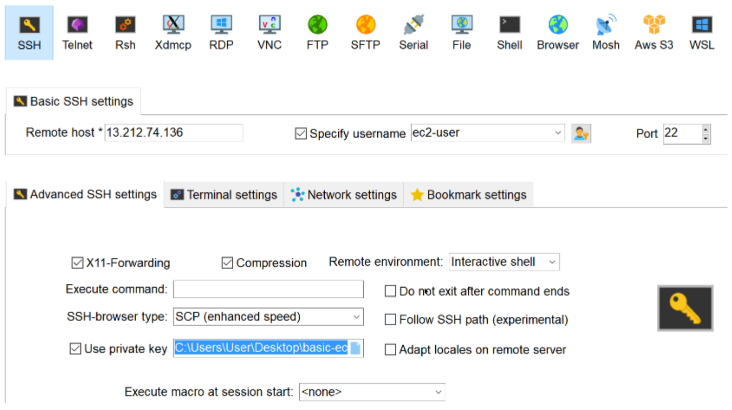
```
ubuntu@ip-172-31-22-74:~$
```

This `ubuntu` account is the default account on all Ubuntu AMI instances and it has sudo privileges to perform standard admin functionality. `root` user password access has been disabled.

On Windows, you can also connect to the instance using any client that offers a SSH client, for e.g. MobaXTerm.

The main settings on MobaXTerm are:
Remote host: *`instance-public-dns-name / instance-public IP address`*
Username: `ubuntu` (or the appropriate user name for the default user account on that instance)
Port: `22`

SSH-browser type: `SCP`
Specify the `private key file`



Basic SSH settings

Remote host * 13.212.74.136 ☑ Specify username ec2-user Port 22

Advanced SSH settings | Terminal settings | Network settings | Bookmark settings

☑ X11-Forwarding ☑ Compression Remote environment: Interactive shell

Execute command: ☐ Do not exit after command ends

SSH-browser type: SCP (enhanced speed) ☐ Follow SSH path (experimental)

☑ Use private key C:\Users\User\Desktop\basic-ec ☐ Adapt locales on remote server

Execute macro at session start: <none>

NOTE: There appears to be a bug/issue with MobaXTerm. Occasionally, the MobaXTerm terminal will freeze temporarily for about 3-10 seconds when you are typing in it - the connection remains stable but the terminal will not appear to respond to key presses for a certain period of time. If this is irritating for you, switch back to working from Powershell.

As we will be working with many instances at once, it is useful to change the hostname of the instances we are connected to avoid any confusion.

Change the hostname of the current JenkinsServer instance with:

```
sudo hostnamectl set-hostname jenkinsserver
```

Then type

```
exit
```

To close the SSH connection and return to Powershell. Repeat the previous command to connect

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

This time you should see the new hostname `jenkinsserver` in the shell prompt.

Before performing any software installation on an existing Linux machine, it is always good to check the Linux distro installed since the installation approach will vary depending on the Linux distro (`apt` package manager for Ubuntu/Debian family of distros and `yum` for RHEL/Fedora/CentOS family of distros).

```
cat /etc/*-release
```

Here we can see that we are on an Ubuntu Linux machine.

We follow the instructions for installing Jenkins on a Ubuntu/Debian family of distros

https://pkg.jenkins.io/debian-stable/
Copy and paste the instructions above one at a time into the shell to install Jenkins

Note
- The \ stands for a line break, so you can copy and paste multiple lines at once
- You can add a -y option to all the install commands of apt-get to avoid question prompts while installing (otherwise you will just need to type y) for all questions.

When you are done, check the Java version

```
java -version
```

Verify that Jenkins is installed and initially inactive with

```
sudo systemctl status jenkins
```

and then start / stop it with the following commands:

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

You can then access the main Jenkins UI in a browser tab:

```
http://public-IP-address:8080
```

The location of the file containing the password to unlock the server is stated, and you can access its contents with a suitable command, for e.g.

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Provide the password

Install all suggested plugins. This will take some time to complete.

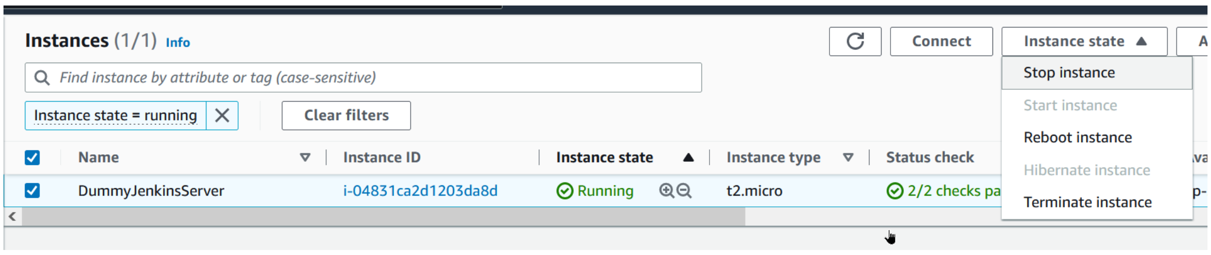
At the Create First Admin User page, enter `admin` for the username and password and some dummy full name and email address.
Click Save and Continue.
Click Save and Finish for the Instance Configuration, and move on to start using Jenkins.

You should see the main Jenkins dashboard.

If at this point, or any at other point in the lab, you wish to stop / break for a significant period of time, you can first shutdown the Jenkins server with:

```
sudo systemctl stop jenkins
```

Then stop the EC2 instance by selecting your instance and choosing Stop instance from the drop down menu.



Name	Instance ID	Instance state	Instance type	Status check
DummyJenkinsServer	i-04831ca2d1203da8d	Running	t2.micro	2/2 checks pa

When you start the EC2 instance again and connect via SSH, you can start up the Jenkins server again with:

```
sudo systemctl start jenkins
```

IMPORTANT: The public IP address or DNS name will dynamically change when an instance is stopped and restarted, so remember to include the latest IP address /DNS name (available from the EC2 instance dashboard) in the ssh command that you type.

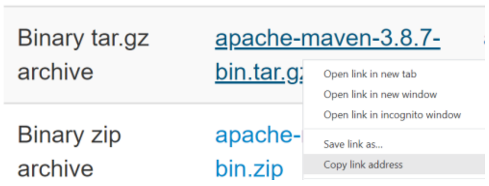
3 Install Maven on the Linux machine

Ensure that you have connected to the Bash shell of your JenkinsServer via SSH.

Go to the main download page of Maven

https://maven.apache.org/download.cgi

Copy the link address for binary tar.gz archive


Binary tar.gz archive	apache-maven-3.8.7-bin.tar.gz
Binary zip archive	apache-maven-3.8.7-bin.zip

and paste this link address to the end of `wget -c` command as below, for e.g.

```
wget -c https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.71/bin/apache-
tomcat-9.0.71.tar.gz
```

This will start the download of the binary achive. When the download is complete, extract this into the current directory (which should be the home directory of the user account ubuntu)

```
tar xvf apache-maven-3.8.7-bin.tar.gz
```

Move the extracted directory to `/opt`
https://www.baeldung.com/linux/opt-directory

```
sudo mv apache-maven-3.8.7 /opt
```

Create a symbolic link to this directory

```
sudo ln -s /opt/apache-maven-3.8.7 /opt/maven
```

Next, we need to get the installation directory of Java/JDK on our system with:

```
readlink -f $(which java)
```

The part of the path in the result before the `/bin/java` will be the actual installation directory, for e.g: `/usr/lib/jvm/java-11-openjdk-amd64`

We will be using this installation directory path to set the environment variable `JAVA_HOME` in the startup script that we will be creating next in `/etc/profile.d`
https://unix.stackexchange.com/questions/64258/what-do-the-scripts-in-etc-profile-d-do

Create a new script `maven.sh` in this location with:

```
sudo nano /etc/profile.d/maven.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

Save and exit nano.

Exit the EC2 instance and connect back in again using the standard SSH command.

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

Check that Maven is on the system path with

```
mvn -version
```

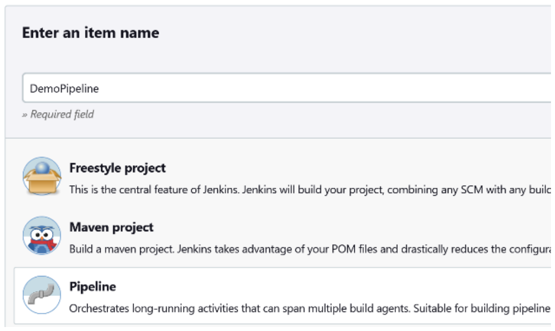
4 Basic Jenkins pipeline with GitHub integration

https://www.jenkins.io/doc/pipeline/tour/hello-world/
https://www.jenkins.io/doc/book/pipeline/

On the JenkinsServer, start up Jenkins if you have not already done so with:

```
sudo systemctl start jenkins
```

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



Enter an item name

DemoPipeline

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configura

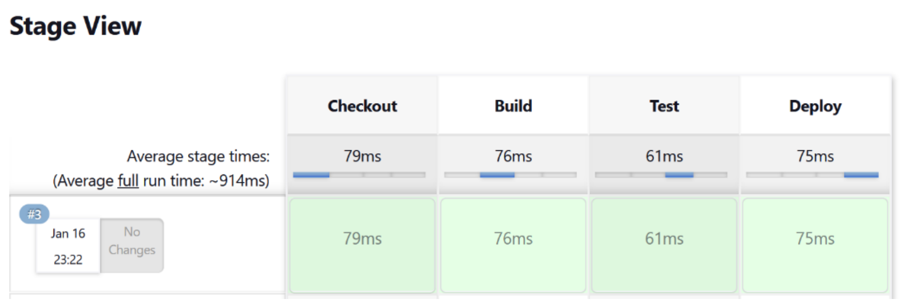
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipeline

In the Pipeline section of Configure, enter the following in the script box

```
pipeline {
    agent any
    stages {

        stage('Checkout') {
            steps {
                echo  'Functionality  to  checkout  code  base  from
remote repo'
            }
        }

        stage('Build') {
            steps {
                echo 'Use Maven or some other build tool to compile'
            }
        }

        stage('Test') {
            steps {
                echo 'Use Maven or some other build tool to run unit
tests on successfully compiled code'
            }
        }

        stage('Deploy') {
            steps {
                echo  'Use  scripts  to  deploy  tested  code  to  staging
and / or production servers'
            }
        }

    }
}
```

Click Save and perform a build.

The main dashboard for this job should show clearly the stages defined in the pipeline and time taken to execute each one.



Stage View

	Checkout	Build	Test	Deploy
Average stage times: (Average full run time: ~914ms)	79ms	76ms	61ms	75ms
#3 Jan 16 23:22 No Changes	79ms	76ms	61ms	75ms

Next, we will implement the previous basic Jenkins GitHub integration freestyle project job from Lab 1 as a pipeline instead.

To prepare, we will repeat the previous steps from Lab 1 which we performed on the Jenkins server installed on our local machines.

First, start off with the lab on installing and configuring Maven plugin

When this is done, perform the lab on Checking / installing Git / GitHub plugins for Jenkins

Finally, return to the `DemoPipeline` go to Configure -> Pipeline, and enter the following updated Pipeline script:
NOTE: Remember to replace the Git HTTPS URL below with the one for your GitHub repo instead.

```
pipeline {

    agent any

    tools {
      maven 'jenkins-maven'
    }

    stages {

        stage("Checkout") {
             steps {
                  git     url:     'https://github.com/victor-tan-
hk/simple-java-maven-app.git', branch: 'master'
             }
        }


        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }

        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }

        stage('Deploy') {
            steps {
              echo 'Executing the generated build artifact'
              sh 'java -jar target/my-app-1.0-SNAPSHOT.jar'
```

```
                    }
                }
            }
}
```

Click Save and perform a build.

Notice the final output reflects the latest code changes from the remote repo. You can perform some more changes to `App.java` in `githubprojects\simple-java-maven-app`, commit them and push to the remote repo, then run this build again to verify.

Compare this with the original Freestyle project that accomplished the exact same functionality in Lab 1. Notice that with a pipeline, we accomplish all our build functionality through command line scripts (executed via the `sh`) as opposed to pre-built in options / functions in a Freestyle project. A Pipeline project gives us a greater degree of freedom in crafting our build process but requires more in-depth knowledge of executing the various steps in detail, while a Freestyle project does a lot of the work in the background for us.

Note that we can also execute this pipeline script on a Jenkins server running on a Windows machine: but remember to replace all the `sh` (which is for a Linux system) with `bat` (which is for a Window system). Try and replicate the creation of this pipeline on a Jenkins server installed on a Windows machine, to verify this for yourself.

A more suitable way to work with a pipeline is to create a Jenkinsfile that contains the pipeline script and include that with our project (typically in the root folder of the project) and maintain that in VCS as well. In this approach, we can also keep track of the changes that we make to our build process (defined in the Jenkinsfile) over the lifetime of the project.

Return to the project folder `simple-java-maven-app` in `githubprojects\simple-java-maven-app`

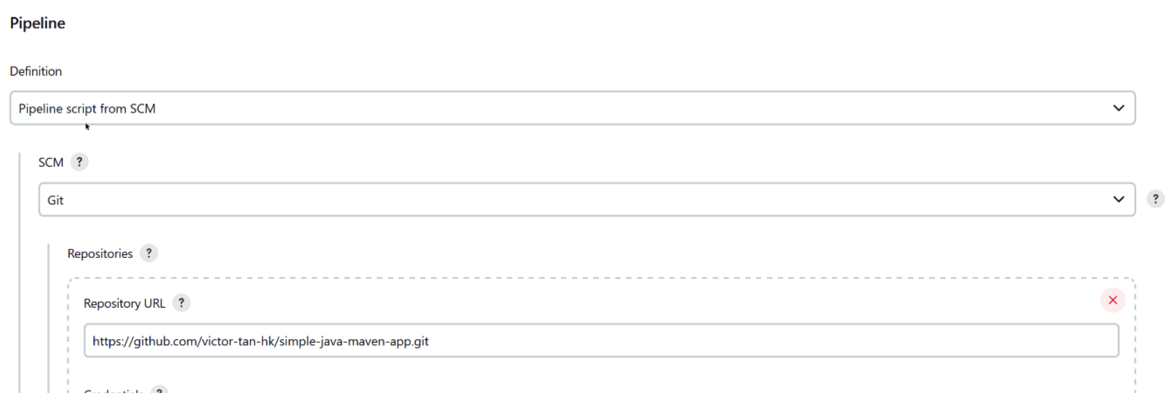
In the root folder of this project, create a single file `Jenkinsfile` (make sure there is no extension like *.txt, etc if you are working in Windows) and place the script above into that file. Save the file. Make a new commit with an appropriate message:

```
git commit -am "Added a new Jenkinsfile to control the build"
```

And push it to the remote repo:

```
git push
```

Return to the `DemoPipeline` go to Configure -> Pipeline, and change the definition to Pipeline script from SCM.



Pipeline

Definition

Pipeline script from SCM

SCM

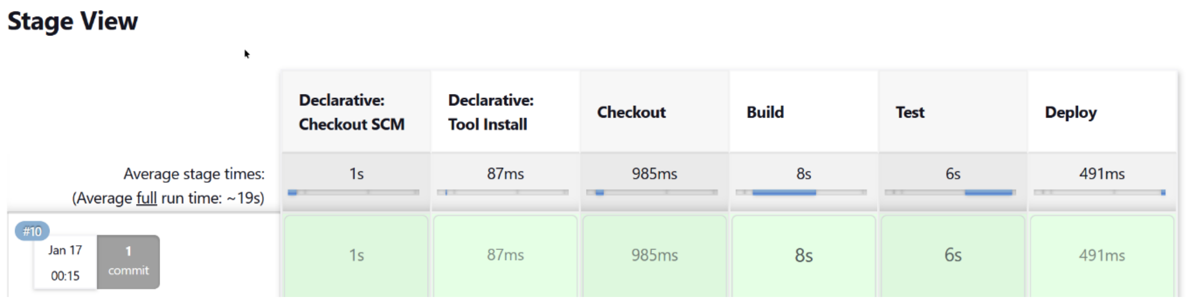
Git

Repositories

Repository URL

https://github.com/victor-tan-hk/simple-java-maven-app.git

Click Save and perform a build. Now we can see there is a declarative checkout SCM stage (indicating that the pipeline script that is being executed is from a Jenkinsfile that is in the GitHub repo).



Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Checkout	Build	Test	Deploy
Average stage times: (Average full run time: ~19s)	1s	87ms	985ms	8s	6s	491ms
#10 Jan 17 00:15 1 commit	1s	87ms	985ms	8s	6s	491ms

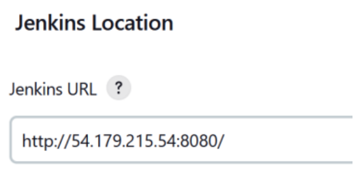
5 Configuring Webhooks with GitHub

In a previous lab, we looked at 2 different build triggers: Periodical builds and Poll SCM. Another popular and useful trigger is webhooks.

When a specific event occurs (for e.g. a new commit is pushed to some branch on a remote repo, a pull request is opened to merge a feature branch back into the master branch, etc), the remote repo sends a signal (typically a HTTP POST request) to the Jenkins server to trigger the build. This feature is known as a webhook and is available with most Git cloud hosting services (GitHub, BitBucket, etc)

https://docs.github.com/en/developers/webhooks-and-events/webhooks/about-webhooks
https://docs.github.com/en/get-started/customizing-your-github-workflow/exploring-integrations/about-webhooks

At the Jenkins UI, go to Manage Jenkins -> Configure System -> Jenkins Location
Make sure the Jenkins URL reflects the correct public IP address of the JenkinsServer EC2 instance as well as port number (check the address bar to verify this or check the EC2 Instance details main dashboard).


Jenkins Location

Jenkins URL

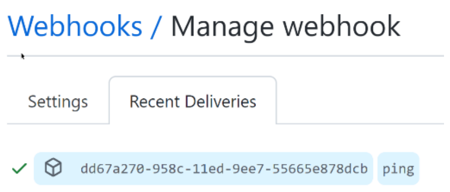
http://54.179.215.54:8080/

Click Save when done.

To setup a WebHook for the GitHub repo `simple-java-maven-app`, follow the guide below:
https://www.blazemeter.com/blog/how-to-integrate-your-github-repository-to-your-jenkins-project

Make sure you type the webhook URL with an ending / , for e.g.
http://13.229.52.22:8080/github-webhook/
or else error will occur when processing the HTTP POST request sent from GitHub on the Jenkins server end.

After the setup is complete on the GitHub end, it will send out a test ping to the Jenkins server at the specified URL and you should receive a success message if everything has been configured correctly.


Webhooks / Manage webhook

Settings | Recent Deliveries

✓ dd67a270-958c-11ed-9ee7-55665e878dcb ping

In the main area for the `BasicGitHubDemo` job, select Configure, Build Triggers -> GitHub hook trigger for GitScm polling.

Click Save.

Wait for a few minutes. Notice that nothing happens.

Return to the project folder in `githubprojects\simple-java-maven-app`
Let's make a minor modification to the source code of the main class for this simple Java project at:

```
src\main\java\com\mycompany\app\App.java
```

using Notepad++ (or any other suitable editor)
Change the parameter of the System.out.println to some other random value, for e.g. `"Modified from local project !"`
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 "Some changes to test webhooks"
```

Check the status again

```
git status
```

Return to the main job page for `BasicGitHubDemo` in the Jenkins UI, and keep your eye on the Build History

Back in the Git Bash shell, push this latest commit to the master on the remote GitHub repo with:

```
git push
```

Notice now that a build is automatically triggered as a result of the webhook that you have setup.

6 Install and configure Tomcat on a Linux machine

Start up the GeneralServer instance and wait for it to transition to a running state. This should be indicated in the instance state and status check columns.

We will open an SSH terminal connection to a bash shell in this instance. The steps here are pretty much identical to the section on installing Jenkins on a Linux machine.

To do this, we will use either the public IPv4 address or Public IPv4 DNS of the running instance. Select the instance and in the details tab below, note down either value in somewhere suitable where you can easily copy and paste it (for e.g. Notepad++)

In the subfolder `labkeys` in your top-level `jenkinslabs` directory where you earlier stored your *.pem file, open a Windows Powershell.

To create the SSH connection using the private key in the *.pem file, the command format is:

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

Put this down somewhere in Notepad++ so you can easily and quickly copy and paste it for frequent use.

On Windows, you can also connect to the instance using any client that offers a SSH client, for e.g. MobaXTerm, as we have already explained earlier in the section on installing Jenkins on a Linux machine.
As we will be working with many instances at once, it is useful to change the hostname of the instances we are connected to avoid any confusion.

Change the hostname of the current GeneralServer instance with:

```
sudo hostnamectl set-hostname generalserver
```

Then type

```
exit
```

to close the SSH connection and return to Powershell. Repeat the previous command to connect

```
ssh ubuntu@public-IP-address-or-DNS -i ./keyfilename.pem
```

This time you should see the new hostname `generalserver` in the shell prompt.

Before performing any software installation on an existing Linux machine, it is always good to check the Linux distro installed since the installation approach will vary depending on the Linux distro (`apt` package manager for Ubuntu/Debian family of distros and `yum` for RHEL/Fedora/CentOS family of distros).

```
cat /etc/*-release
```

Here we can see that we are on a Ubuntu Linux machine.

There are a variety of tutorials for installing Tomcat on Linux:

https://www.digitalocean.com/community/tutorials/install-tomcat-on-linux

https://www.hostinger.my/tutorials/how-to-install-tomcat-on-ubuntu/

First, we will install the default JDK version for our Linux distro.

```
sudo apt update
```

```
sudo apt -y install default-jdk
```

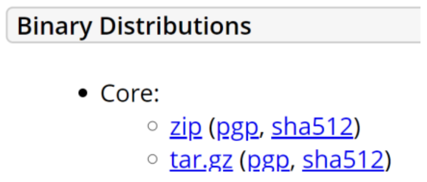
If you have difficulties performing the install for `default-jdk`, then you may additionally need to run:

```
sudo apt upgrade
```

Go to the main download page of Tomcat version that you want to use (here we will use 9 because Jenkins is only able to deploy to that version, although the latest version is 11).

https://tomcat.apache.org/download-90.cgi

Copy the link address for tar.gz



Binary Distributions

- Core:
  - zip (pgp, sha512)
  - tar.gz (pgp, sha512)

And paste this link address to the end of `wget -c` command as below:

```
wget -c https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.71/bin/apache-
tomcat-9.0.71.tar.gz
```

Rename and move this installation directory to `/opt`

```
sudo mv apache-tomcat-9.0.71 /opt/tomcat
```

Provide execution permissions for the startup and shutdown scripts in the bin folder of the main installation directory in `/opt/tomcat`

```
chmod +x /opt/tomcat/bin/startup.sh
chmod +x /opt/tomcat/bin/shutdown.sh
```

To startup and shutdown the Tomcat server, we simply access these 2 scripts with either:

```
/opt/tomcat/bin/shutdown.sh
```

```
/opt/tomcat/bin/startup.sh
```

Once Tomcat is active and running, we install `net-tools` in order to get access to the `netstat` command to check for active ports. This is useful for trouble shooting purposes in the event we cannot determine whether the Tomcat server is actively running.

```
sudo apt install -y net-tools
```

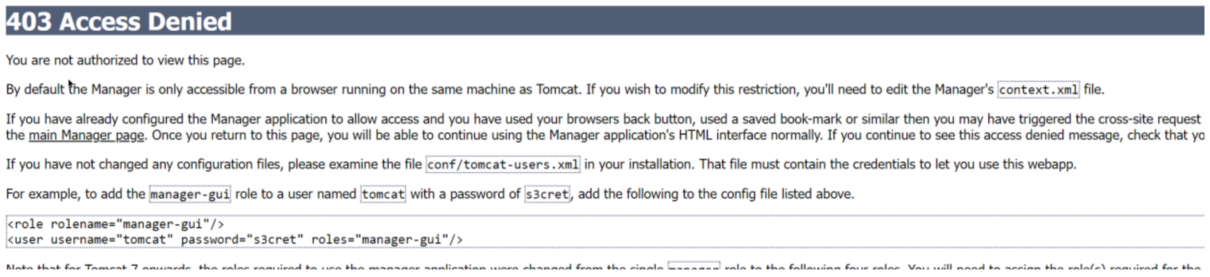
```
sudo netstat -tuplan
```

You should see a Java process running on port 8080 (this would be the Tomcat server)

Once the Tomcat server is up and running, access the main UI:

```
http://public-IP-address:8080
```

At the moment, you will not be able to gain access to the Manager App since it is be default configured to only allows access from a browser on the same machine as Tomcat. If you attempt to do so, you will get this error message:



403 Access Denied

You are not authorized to view this page.

By default the Manager is only accessible from a browser running on the same machine as Tomcat. If you wish to modify this restriction, you'll need to edit the Manager's `context.xml` file.

If you have already configured the Manager application to allow access and you have used your browsers back button, used a saved book-mark or similar then you may have triggered the cross-site request
the main Manager page. Once you return to this page, you will be able to continue using the Manager application's HTML interface normally. If you continue to see this access denied message, check that yo

If you have not changed any configuration files, please examine the file `conf/tomcat-users.xml` in your installation. That file must contain the credentials to let you use this webapp.

For example, to add the `manager-gui` role to a user named `tomcat` with a password of `s3cret`, add the following to the config file listed above.

```
<role rolename="manager-gui"/>
<user username="tomcat" password="s3cret" roles="manager-gui"/>
```

Note that for Tomcat 7 onwards, the roles required to use the manager application were changed from the single `manager` role to the following four roles. You will need to assign the role(s) required for the

To permit access, you must

a) allow access from browsers on any machine (remote IP addresses) rather than a browser running on the same machine as Tomcat
b) register one or more users with an appropriate role (typically manager-gui) in `$CATALINA_BASE/conf/tomcat-users.xml.`

https://clients.javapipe.com/knowledgebase/129/How-to-access-Tomcat-Manager.html

https://tomcat.apache.org/tomcat-9.0-doc/manager-howto.html
Topic: Configuring Manager Application Access

To edit files from the CLI in Linux, there are a variety of text editors we can use. Vim and Emacs are older and more established with extensive functionality, but are very difficult to use. We will be using nano instead, as this is perfect for beginners performing basic text editing.

https://linuxhint.com/nano-editor-beginner-guide/
https://linuxize.com/post/how-to-use-nano-text-editor/

Create a few basic files in your home directory and make sure you know how to remove lines, add lines, save changes and exit from the editor. You can copy and paste stuff from external sources into nano editor in the Powershell by simply right-clicking.

First shutdown Tomcat:

```
/opt/tomcat/bin/shutdown.sh
```

Next, modify this configuration file for Tomcat with nano:

```
nano /opt/tomcat/webapps/manager/META-INF/context.xml
```

Comment out this line

```
<!--  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
```

Save and exit.

Next, modify this configuration file for Tomcat with nano:

```
nano /opt/tomcat/webapps/host-manager/META-INF/context.xml
```

and comment out the same previous line as well.
Save and exit.

Finally, modify this configuration file for Tomcat with nano:

```
nano /opt/tomcat/conf/tomcat-users.xml
```

Add in the snippet below in any empty location above or between the comments in the file before the final closing `</tomcat-users>`
NOTE: Make sure you do not have any space between the starting < and the start of each line in the file to avoid parsing issues in this file by Tomcat.

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user   username="admin"   password="admin"   roles="manager-gui,
manager-script, manager-jmx, manager-status"/>
<user   username="deployer"   password="deployer"   roles="manager-
script"/>
<user username="user" password="user" roles="manager-gui"/>
```

Save and exit.

Now startup the Tomcat server again:

```
/opt/tomcat/bin/startup.sh
```

Access the main UI at:

```
http://public-IP-address:8080
```

You should be able to login using the username/pw combination of any user that has the `manager-gui` role (in this case `admin` and `user`).

7 EXERCISE: Basic Jenkins job to build and deploy a Java Web app

Repeat the same lab session for Lab 1, with the difference that this time, when we configure the container, we will provide the URL that includes the public IP address and port number that the Tomcat server is running on the GeneralServer that you configured and completed in a previous lab session. Remember to also setup credentials and plugin to integrated with Tomcat container.

If you now add in the integration of webhooks as well with your GitHub repo, you will have implemented a full proper CI / CD pipeline workflow.

8 EXERCISE: Configuring Webhooks with BitBucket

In a previous lab, we have configured a webhook to automatically trigger a build of a Jenkins job when a push is made to a GitHub repo. The exact same thing can also be done for a BitBucket repo.

In the main `jenkinslab` folder, create a new folder called `bitbucketprojects`. Copy `simple-java-maven-app` from the `labcode` of your downloaded zip into the `bitbucketprojects` folder

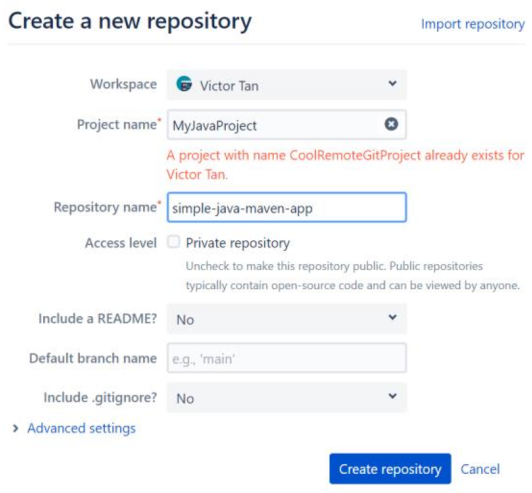
Open a Git bash in `simple-java-maven-app` and initialize this as a new Git repo with:

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

Login to your Bitbucket account. Go through the process of [creating a new repository](), but create a bare repository with no content to facilitate the process of pushing the contents of our local repository to it:

Enter the values for the following fields.
Project Name: `MyJavaProject`
Repository Name: `simple-java-maven-app`

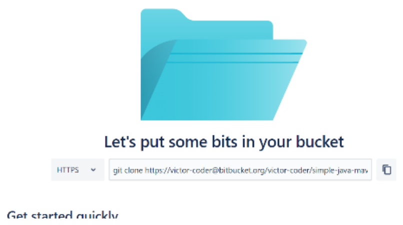
Make sure this is NOT a private repository, and DON'T either a readme file or .gitignore. This is to ensure that the repository is a bare repository that we can push new content into from our local repo.


Create a new repository

Import repository

Workspace	Victor Tan
Project name*	MyJavaProject
	A project with name CoolRemoteGitProject already exists for Victor Tan.
Repository name*	simple-java-maven-app
Access level	☐ Private repository
	Uncheck to make this repository public. Public repositories typically contain open-source code and can be viewed by anyone.
Include a README?	No
Default branch name	e.g., 'main'
Include .gitignore?	No

When you are done specifying the values for the fields as shown above, click Create Repository. You will be transitioned to the Source view for the newly created repo, where some instructions will be provided on how to get started.


Let's put some bits in your bucket

HTTPS

```
git clone https://victor-coder@bitbucket.org/victor-coder/simple-java-mav
```

Click on the Clone button to copy the URL (e.g. https://xxx@bitbucket.org/yyy/simple-java-maven-app.git) for this new remote repo to an empty document. We will refer to this URL as `remote-url` in the commands to follow.

Open a Git Bash shell in `firstlab`, and type:

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

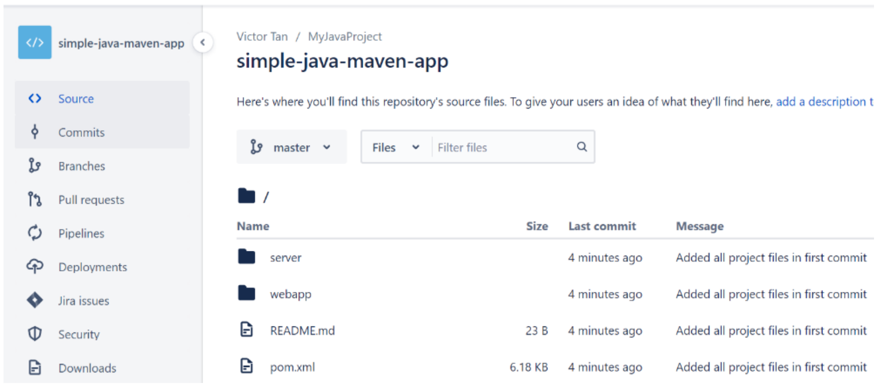
Check that the origin handle is set to point to the correct repo URL with:

```
git remote --verbose
```

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

```
git push -u origin --all

Enumerating objects: 39, done.
….
…..
* [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Return back to the browser and check through the Source, Commits and Branches main view to verify that these mirrors the status of the local repo that you just pushed up into it. Note that you may need to navigate out to the main Repositories tab in the main menu, and then click on the `simple-java-maven-app` repo entry in order for it to refresh properly and show the uploaded content. This may take some time to complete correctly.



Here's where you'll find this repository's source files. To give your users an idea of what they'll find here, add a description t

Name	Size	Last commit	Message
server		4 minutes ago	Added all project files in first commit
webapp		4 minutes ago	Added all project files in first commit
README.md	23 B	4 minutes ago	Added all project files in first commit
pom.xml	6.18 KB	4 minutes ago	Added all project files in first commit

Back in the Git 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
```

As an exercise on your own, set up a new Jenkins job to integrated with a webhook on this Bitbucket repo. Test it out by pushing new commits from your local `simple-java-maven-app`
The sample guides to do this are shown below.

https://hevodata.com/learn/bitbucket-webhook-jenkins-integration/

https://medium.com/ampersand-academy/integrate-bitbucket-jenkins-c6e51103d0fe

https://plugins.jenkins.io/bitbucket/

9 EXERCISE: Creating a Jenkins job / pipeline for a language / platform of your choice

So far in this workshop, we have focused on using Jenkins for a Java web app project.
Of course, Jenkins is programming language and platform neutral, and can be used to automate the build as well as CI/CD pipeline for any language / framework / platform.

If you are a developer, try to create a simple build job / pipeline for the language / framework that you are familiar with. Some guides to get you started (obviously feel free to search for more of your own😊

C# / .NET Core

https://www.c-sharpcorner.com/article/continuous-integration-for-net-projects-with-jenkins/

https://www.oneoddsock.com/2022/04/08/ci-cd-building-a-c-project-in-jenkins/

https://referbruv.com/blog/cicd-getting-started-automating-aspnet-core-build-using-jenkins/

https://www.swtestacademy.com/jenkins-dotnet-integration/

Python

https://www.jenkins.io/solutions/python/

https://medium.com/nerd-for-tech/python-job-using-jenkins-35dd0afa396b

https://joachim8675309.medium.com/jenkins-ci-pipeline-with-python-8bf1a0234ec3

https://comquent.de/de/de-how-to-build-a-python-project-in-jenkins/

https://mdyzma.github.io/2017/10/14/python-app-and-jenkins/

https://skamalakannan.dev/posts/jenkins-pipeline-python/