Setting up the EC2 Server

When starting I had an EC2 already set up with the AWS Ubuntu 22.04 jammy image.

```
ubuntu@ip-[redacted]: ~
richard@Tokyo:~/jenkins $ screenfetch
                           ./+o+-       richard@Tokyo
                   yyyyy- -yyyyyy+      OS: Ubuntu 22.04 jammy
                 ://+//////-yyyyyyo     Kernel: x86_64 Linux 5.15.0-1017-aws
           .++ .:/++++++/-.+sss/`       Uptime: 14d 3h 14m
         .:++o:  /++++++++/:--:/-       Packages: 681
        o:+o+:++.`..```.-/oo+++++/      Shell: bash 5.1.16
       .:+o:+o/.          `+sssoo+/     Disk: 3.5G / 7.7G (45%)
  .++/+:+oo+o:`             /sssooo.    CPU: Intel Xeon E5-2676 v3 @ 2.4GHz
 /+++//+:`oo+o               /::--:.    RAM: 707MiB / 966MiB
 \+/+o+++`o++o               ++////.
  .++.o+++oo+:`              /dddhhh.
       .+.o+oo:.          `oddhhhh+
        \+.++o+o``-```.:ohdhhhhh+
         `:o+++ `ohhhhhhhhyo++os:
           .o:`.syhhhhhhh/.oo++o`
               /osyyyyyyo++ooo+++/
                    ``` +oo+++o\:
                           `oo++.
richard@Tokyo:~/jenkins $
```

Then I edited the port rules of the EC2 making sure 22, 443, 8080 and 22 are open.

**Inbound rules** Info

| Security group rule ID | Type Info | Protocol Info | Port range Info | Source Info | Description - optional Info | |
|---|---|---|---|---|---|---|
| sgr-03fdf7d94e86ffd47 | SSH | TCP | 22 | Custom 0.0.0.0/0 | SSH Traffic | Delete |
| sgr-097abfcbc953e7936 | HTTPS | TCP | 443 | Custom 0.0.0.0/0 | HTTPS Traffic | Delete |
| sgr-01d394d4ca6afe2e0 | Custom TCP | TCP | 8080 | Custom 0.0.0.0/0 | Jenkins Traffic | Delete |
| sgr-006ae1bda2c51382e | HTTP | TCP | 80 | Custom 0.0.0.0/0 | HTTP Traffic | Delete |

Add rule
```

Installing Jenkins on the EC2

After logging on to the EC2 with SSH I ran the script I made below(also on my github as **runinstalljenkins.sh**) that I transferred over to the EC2 earlier. This script **runs a script to install Jenkins(installjenkins.sh)**:

```
#!/bin/bash

#This script is to run the Install Jenkins script

#Run as admin only check
if [ $UID != 0 ]; then
    echo "Run again with admin permissions"
    exit 1
fi

echo "Installing Jenkins"

#Command to run the install script and log everything
/bin/bash installjenkins.sh &> installjenkins.log

#Check if the install had a error
if [ $? -ne 0 ]; then
    echo "Installation error"
    exit 1
else
    echo "Installation successful"
    echo "Installation Logs in 'installjenkins.log'"
fi

#Wait 5 seconds then check status
sleep 5

#Check the status of the service
systemctl status jenkins --no-pager

#Print out secret password to login and setup Jenkins
echo "Password to Unlock and Setup Jenkins Below"
cat /var/lib/jenkins/secrets/initialAdminPassword

#successful
echo "Run successful"
exit 0
```

Below is the script(also on my github as **installjenkins.sh**) that **runinstalljenkins.sh** runs. Also transferred to the EC2 earlier. The **output is stored in a log file**.

```bash
#!/bin/bash

#This script is to install Jenkins and it's dependencies and start the service

#Run as admin only check
if [ $UID != 0 ]; then
    echo "Run again with admin permissions"
    exit 1
fi

#Adding the Keyrings without user interaction
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | gpg --batch --yes --dearmor -o /usr/share/keyrings/jenkins.gpg && echo "Jenkins Keyring Added"

#Adding the repo to the sources of apt
sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list' && echo "Jenkins Repo Added"

#Update local repo database
apt-get update

#Install java, Jenkins, pip and venv in that order
apt-get install default-jre -y && echo "Installed Java Runtime Engine" && apt-get install jenkins -y && echo "Installed Jenkins" && apt-get install python3-pip -y && echo "Installed Python pip" && apt-get install python3.10-venv -y && echo "Installed Python venv"

#Start the Jenkins service
systemctl start jenkins && echo "Jenkins Started"

#successful
echo "Installation successful"
exit 0
```

Below is an example of the output(censored) from the **runinstalljenkins.sh** script.

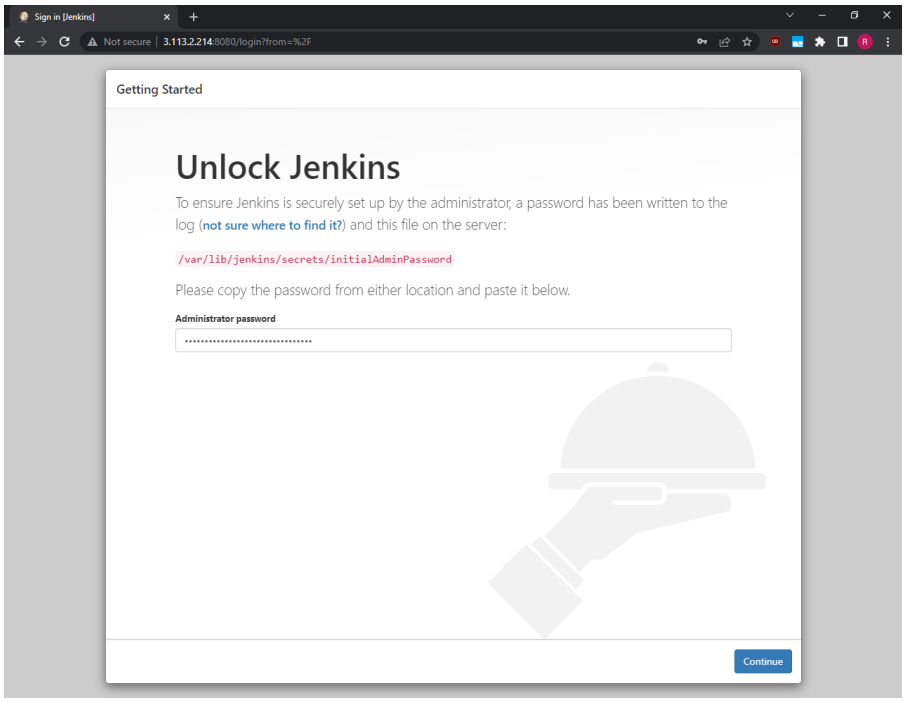
```
Ubuntu 20.04 on Windows
richard@Tokyo:~ $ sudo ./runinstalljenkins.sh
Installing Jenkins
Installation successful
Installation Logs in 'installjenkins.log'
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-08-31 20:59:31 UTC; 16s ago
   Main PID: 137227 (java)
      Tasks: 43 (limit: 1143)
     Memory: 232.8M
        CPU: 20.536s
     CGroup: /system.slice/jenkins.service
             └─137227 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot

Aug 31 20:59:16 Tokyo jenkins[137227]: *************************************************************
Aug 31 20:59:16 Tokyo jenkins[137227]: *************************************************************
Aug 31 20:59:16 Tokyo jenkins[137227]: WARNING: An illegal reflective access operation has occurred
Aug 31 20:59:16 Tokyo jenkins[137227]: WARNING: Illegal reflective access by org.codehaus.groovy.vmplug
Aug 31 20:59:16 Tokyo jenkins[137227]: WARNING: Please consider reporting this to the maintainers of or
Aug 31 20:59:16 Tokyo jenkins[137227]: WARNING: Use --illegal-access=warn to enable warnings of further
Aug 31 20:59:16 Tokyo jenkins[137227]: WARNING: All illegal access operations will be denied in a futur
Aug 31 20:59:31 Tokyo jenkins[137227]: 2022-08-31 20:59:31.617+0000 [id=28]        INFO        jenkins.
Aug 31 20:59:31 Tokyo jenkins[137227]: 2022-08-31 20:59:31.657+0000 [id=22]        INFO        hudson.I
Aug 31 20:59:31 Tokyo systemd[1]: Started Jenkins Continuous Integration Server.
Hint: Some lines were ellipsized, use -l to show in full.
Password to Unlock and Setup Jenkins Below

Run successful
richard@Tokyo:~ $
```

Below is an example of the logs produced from the **installjenkins.sh** script.

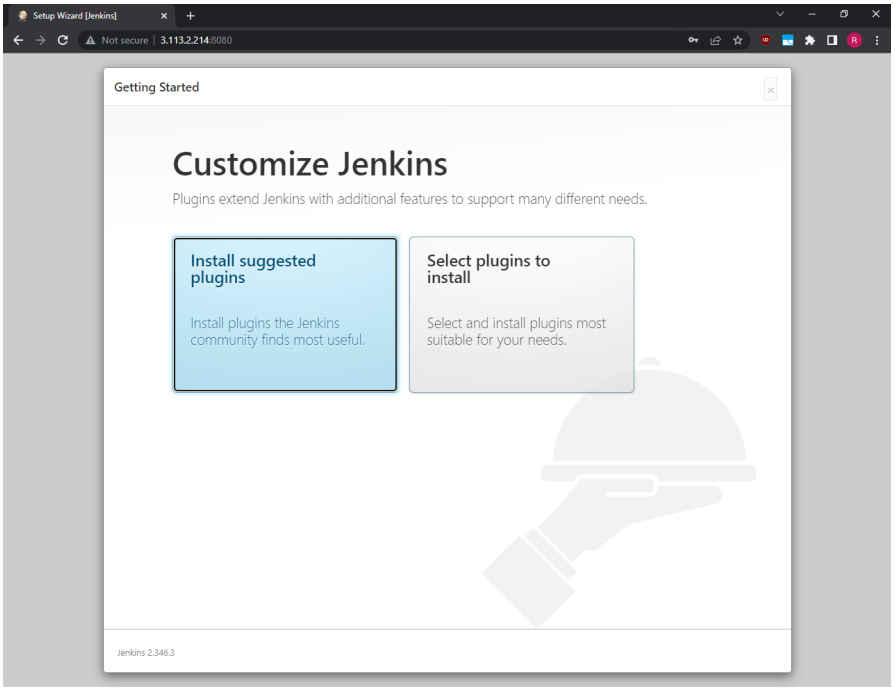
```
Ubuntu 20.04 on Windows
  GNU nano 6.2
Selecting previously unselected package libpython3.10-dev:amd64.
Preparing to unpack .../06-libpython3.10-dev_3.10.4-3ubuntu0.1_amd64.deb ...
Unpacking libpython3.10-dev:amd64 (3.10.4-3ubuntu0.1) ...
Selecting previously unselected package libpython3-dev:amd64.
Preparing to unpack .../07-libpython3-dev_3.10.4-0ubuntu2_amd64.deb ...
Unpacking libpython3-dev:amd64 (3.10.4-0ubuntu2) ...
Selecting previously unselected package python3.10-dev.
Preparing to unpack .../08-python3.10-dev_3.10.4-3ubuntu0.1_amd64.deb ...
Unpacking python3.10-dev (3.10.4-3ubuntu0.1) ...
Selecting previously unselected package python3-dev.
Preparing to unpack .../09-python3-dev_3.10.4-0ubuntu2_amd64.deb ...
Unpacking python3-dev (3.10.4-0ubuntu2) ...
Selecting previously unselected package python3-wheel.
Preparing to unpack .../10-python3-wheel_0.37.1-2_all.deb ...
Unpacking python3-wheel (0.37.1-2) ...
Selecting previously unselected package python3-pip.
Preparing to unpack .../11-python3-pip_22.0.2+dfsg-1_all.deb ...
Unpacking python3-pip (22.0.2+dfsg-1) ...
Setting up javascript-common (11+nmu1) ...
Setting up python3-wheel (0.37.1-2) ...
Setting up libexpat1-dev:amd64 (2.4.7-1) ...
Setting up python3-pip (22.0.2+dfsg-1) ...
Setting up zlib1g-dev:amd64 (1:1.2.11.dfsg-2ubuntu9) ...
Setting up libjs-jquery (3.6.0+dfsg+~3.5.13-1) ...
Setting up libjs-underscore (1.13.2~dfsg-2) ...
Setting up libpython3.10-dev:amd64 (3.10.4-3ubuntu0.1) ...
Setting up libjs-sphinxdoc (4.3.2-1) ...
Setting up python3.10-dev (3.10.4-3ubuntu0.1) ...
Setting up libpython3-dev:amd64 (3.10.4-0ubuntu2) ...
Setting up python3-dev (3.10.4-0ubuntu2) ...
Processing triggers for man-db (2.10.2-1) ...
Installed Python pip
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  python3-pip-whl python3-setuptools-whl
The following NEW packages will be installed:
  python3-pip-whl python3-setuptools-whl python3.10-venv
0 upgraded, 3 newly installed, 0 to remove and 6 not upgraded.
Need to get 0 B/2473 kB of archives.
After this operation, 2882 kB of additional disk space will be used.
Selecting previously unselected package python3-pip-whl.
(Reading database ... ^M(Reading database ... 5%^M(Reading database ... 10%^M(Readin
Preparing to unpack .../python3-pip-whl_22.0.2+dfsg-1_all.deb ...
Unpacking python3-pip-whl (22.0.2+dfsg-1) ...
Selecting previously unselected package python3-setuptools-whl.
Preparing to unpack .../python3-setuptools-whl_59.6.0-1.2_all.deb ...
Unpacking python3-setuptools-whl (59.6.0-1.2) ...
Selecting previously unselected package python3.10-venv.
Preparing to unpack .../python3.10-venv_3.10.4-3ubuntu0.1_amd64.deb ...
Unpacking python3.10-venv (3.10.4-3ubuntu0.1) ...
Setting up python3-setuptools-whl (59.6.0-1.2) ...
Setting up python3-pip-whl (22.0.2+dfsg-1) ...
Setting up python3.10-venv (3.10.4-3ubuntu0.1) ...
Installed Python venv
Jenkins Started
Installation successful

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify
```

Configuring Jenkins on the EC2

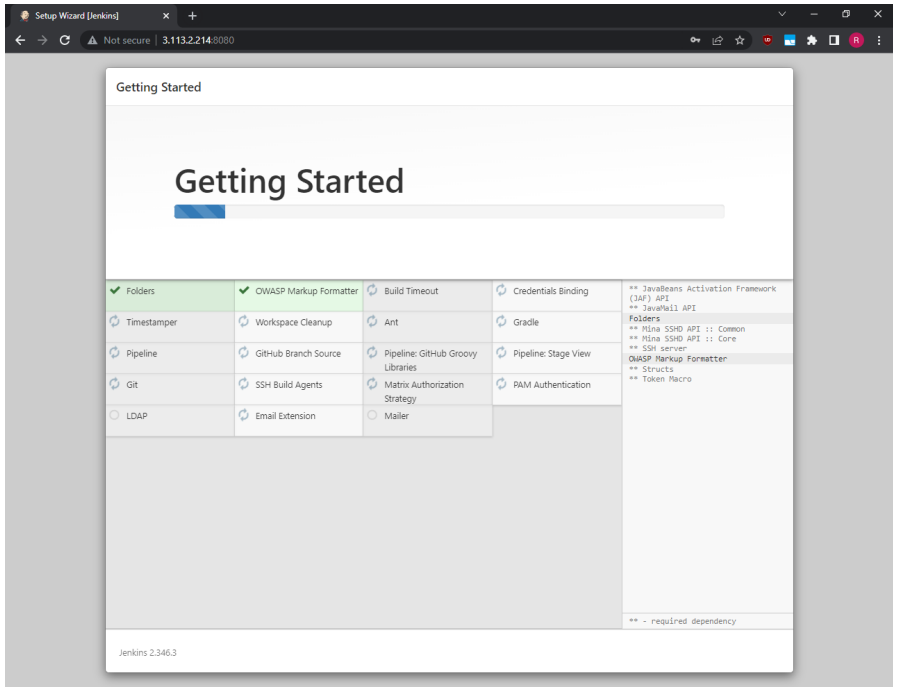
I navigated to the webpage of Jenkins and port 8080 such as **http://3.113.2.214:8080/** and entered the **secret password** located in the file **/var/lib/jenkins/secrets/initialAdminPassword** and clicked **Continue**.



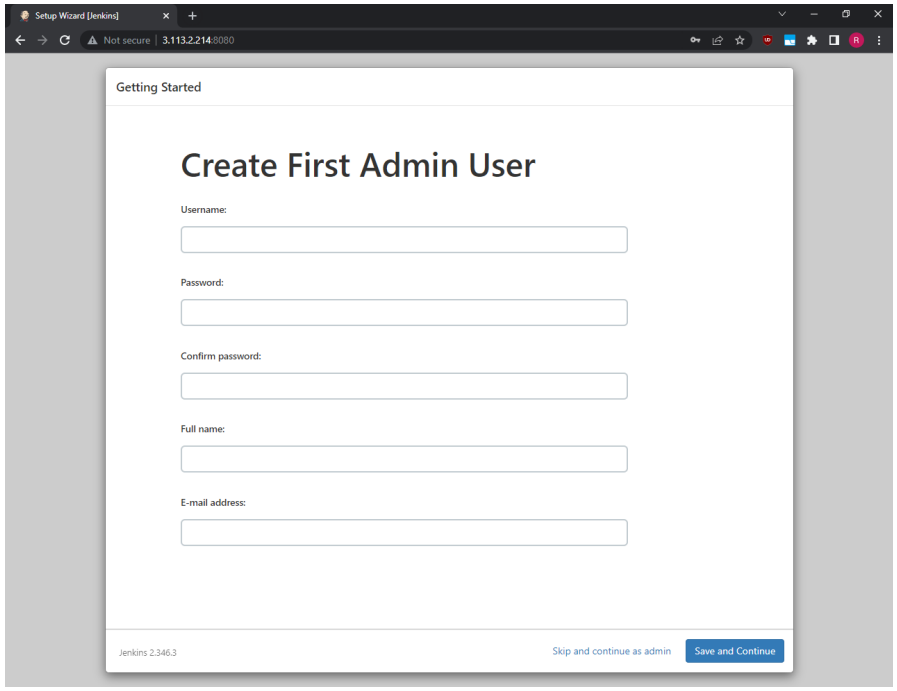
Then I needed to install plugins so I selected **install suggested plugins**.



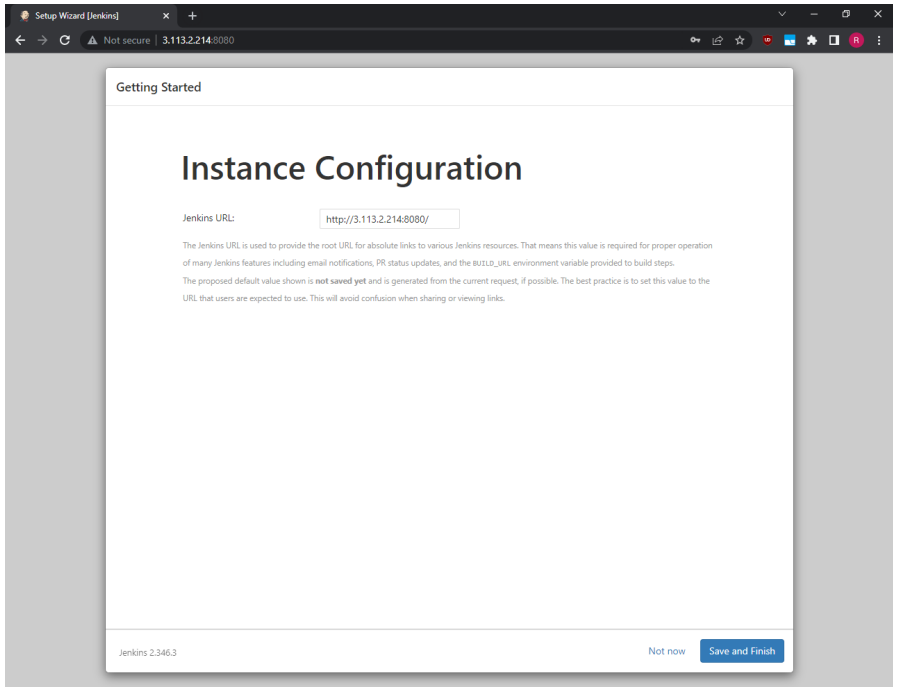
I had to **wait** for the plugins to download and install themself. **No action is needed**.



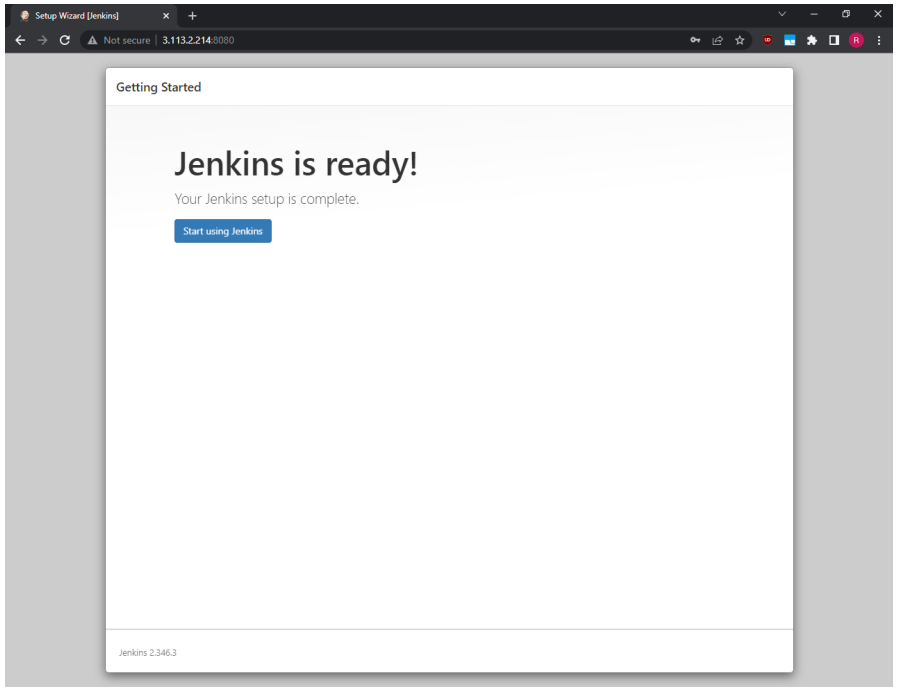
After that I created the **first admin user** by inputting my information and clicking **Save and Continue**.



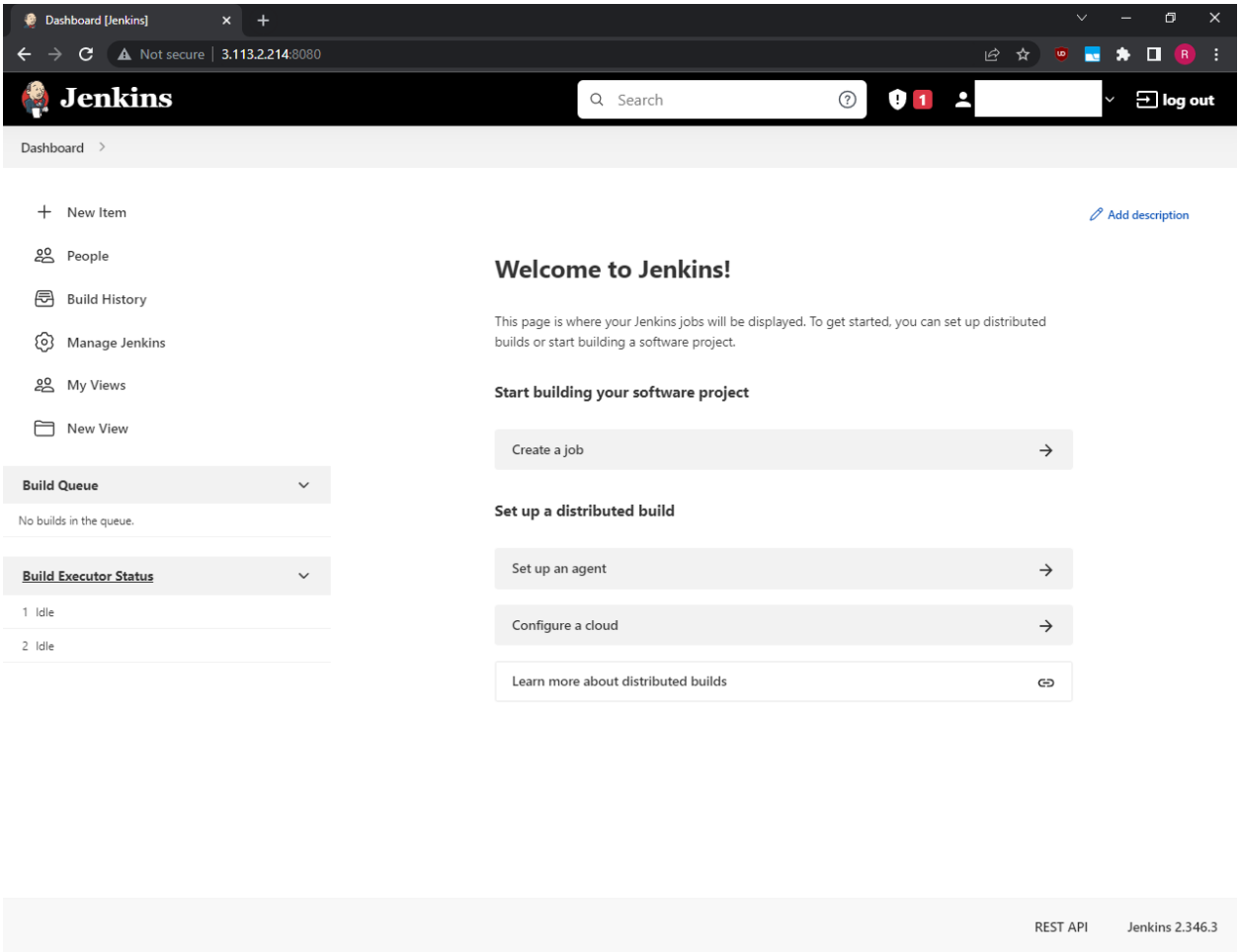
When the user is created then I had to configure the instance specifically the **Jenkins URL** which is the root URL. The default is good so I left it alone and clicked **Save and Finish**.



This marks the e**nd of configuring** Jenkins. I clicked **Start using Jenkins**.

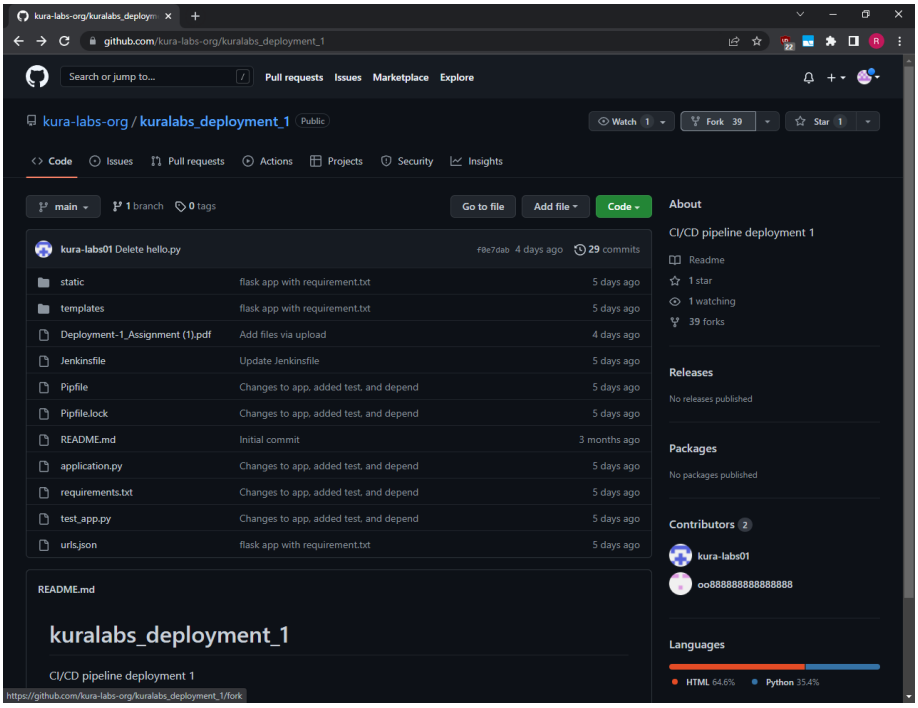


After that is this page which is the **main screen of Jenkins**. This part is completed for now. **Optionally** I could have set up the **Amazon EC2 plugin** and **Configure a Cloud** but I felt it was **unnecessary**.

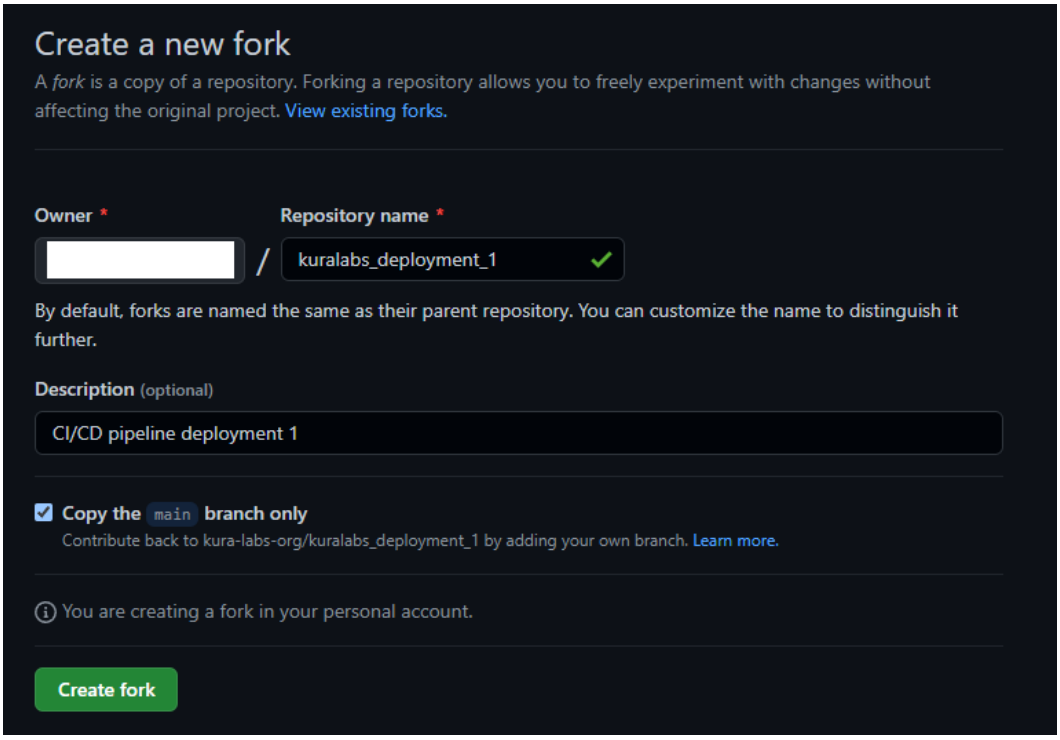


Forking the Deployment Repository

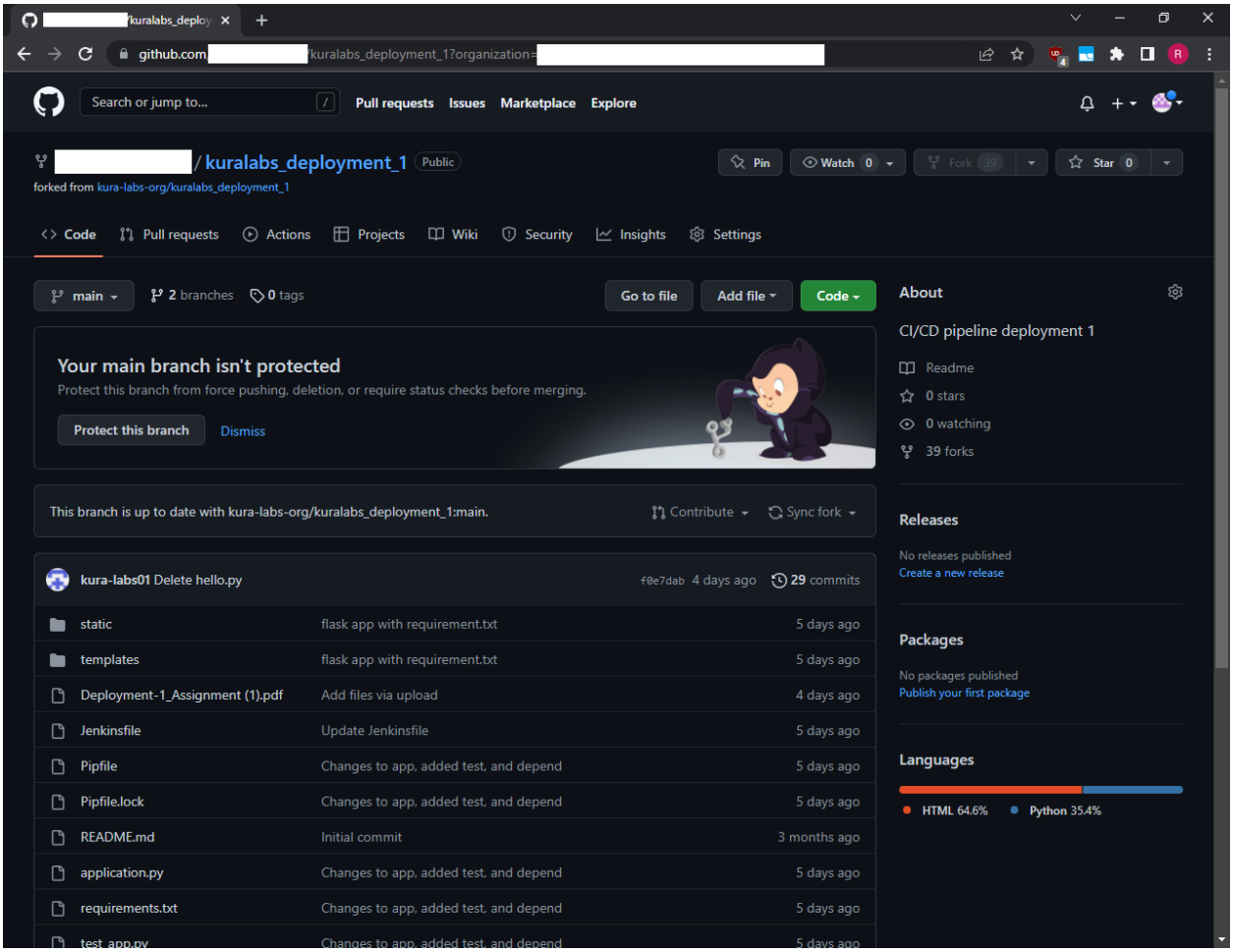
I navigated to the repository page (https://github.com/kura-labs-org/kuralabs_deployment_1) and clicked **Fork** on the right.



Then after all the information was correct I clicked **Create fork**.

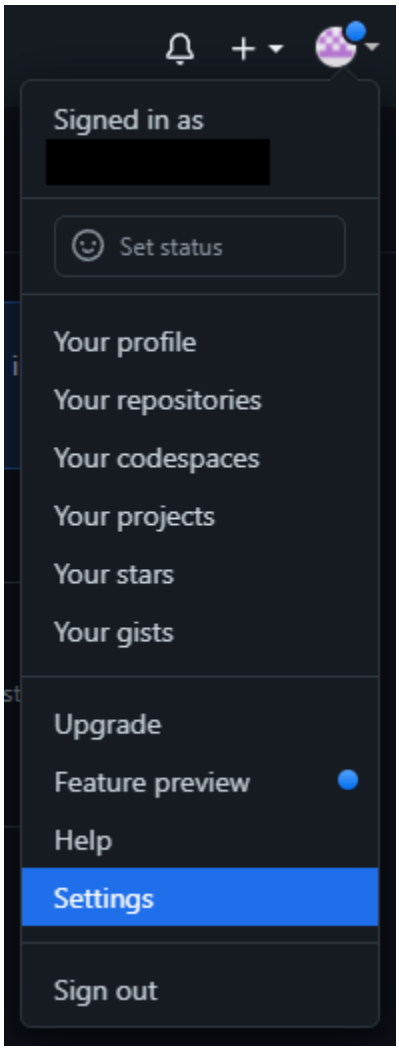


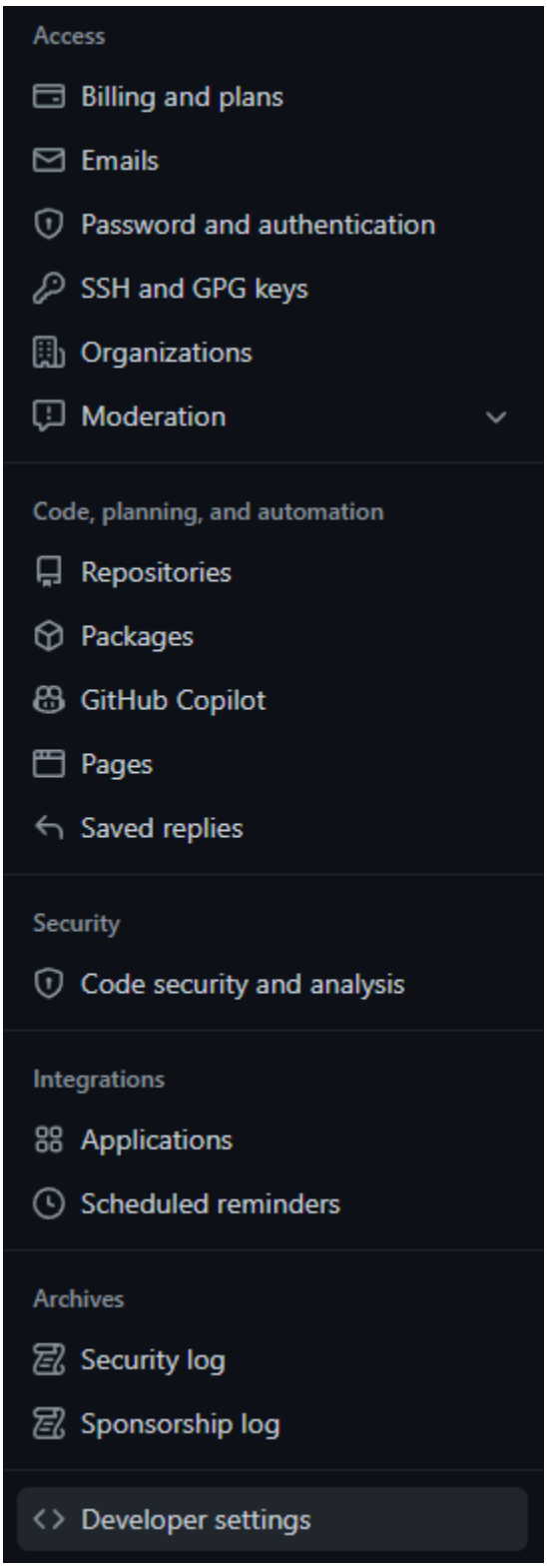
When the fork was created I had **my own copy of the repository**. This is the end of this part.



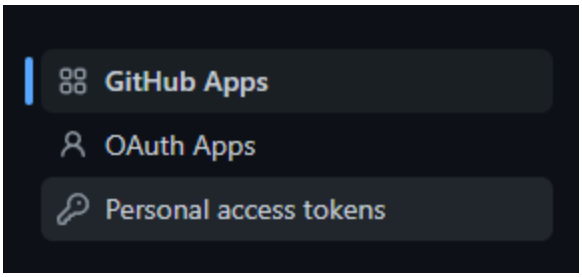
Github Personal Access Token

I went to **Github** and then clicked my **Account Icon** on the right for the options. Then click **Setting** then on the new page click **Developer settings** on the left.

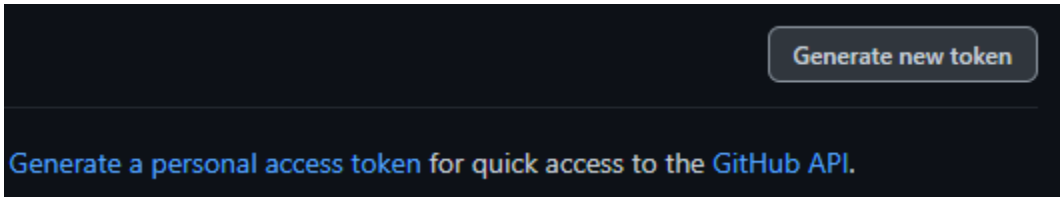




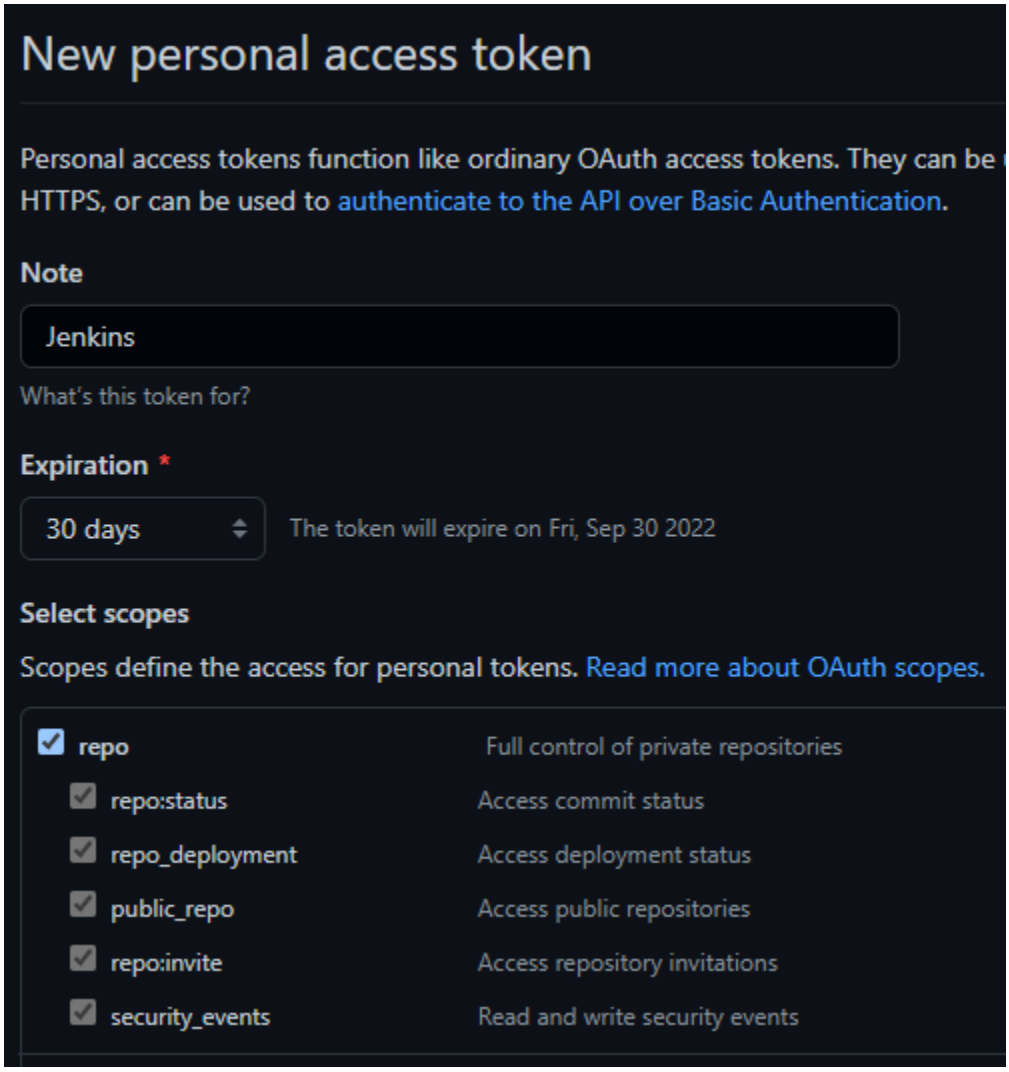
On the left of the new page click **Personal access tokens**.



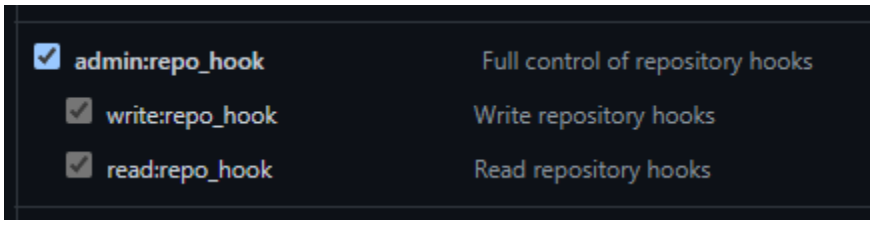
In the middle of the new page click **Generate new token**.



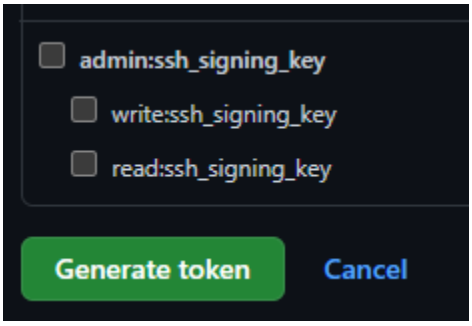
Write a **Note** on what the token is for then click **repo** for the scope.



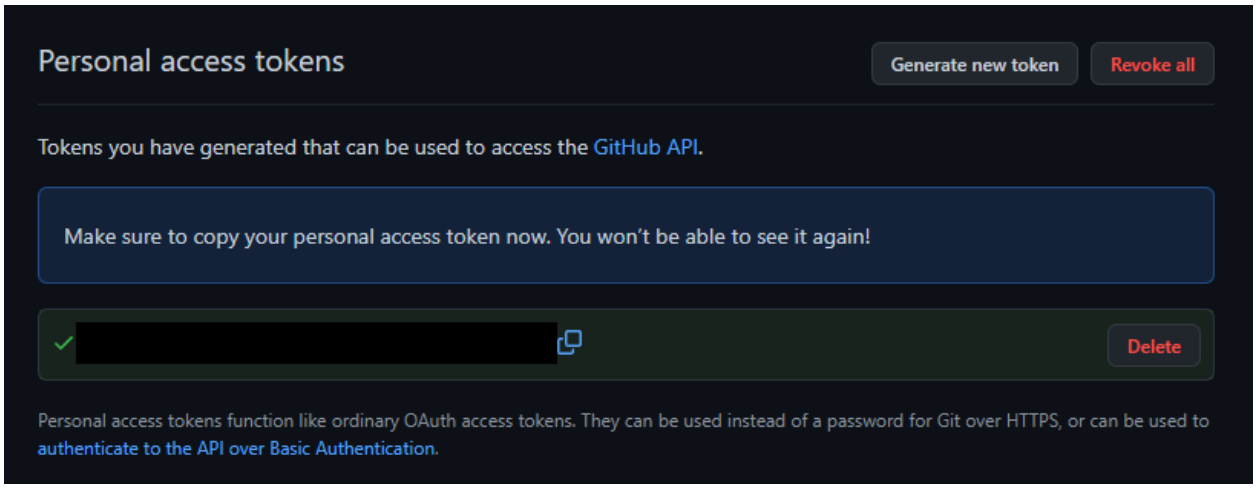
Further down click **admin:repo_hook** to add to the scope.



After that click **Generate token** to generate the Personal access token.

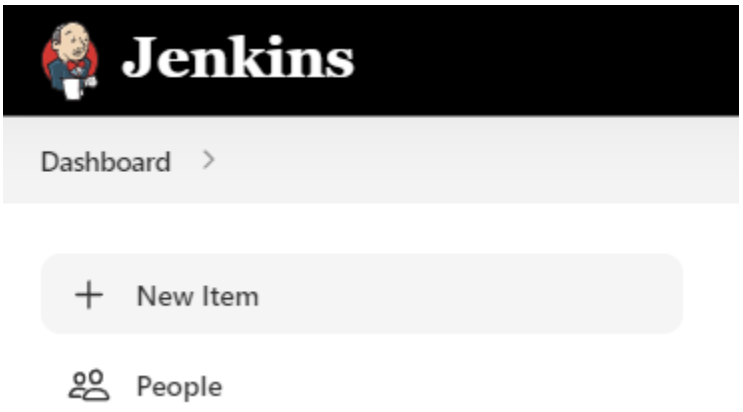


When it was generated I had the following(censored) **Personal access token**.

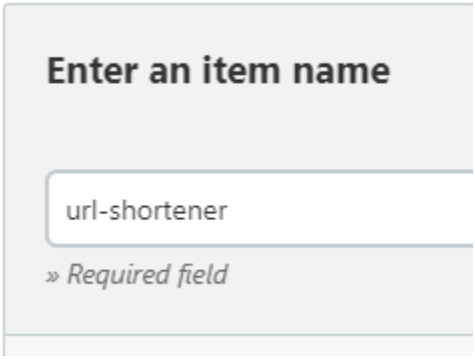


Create a Multibranch Build on Jenkins

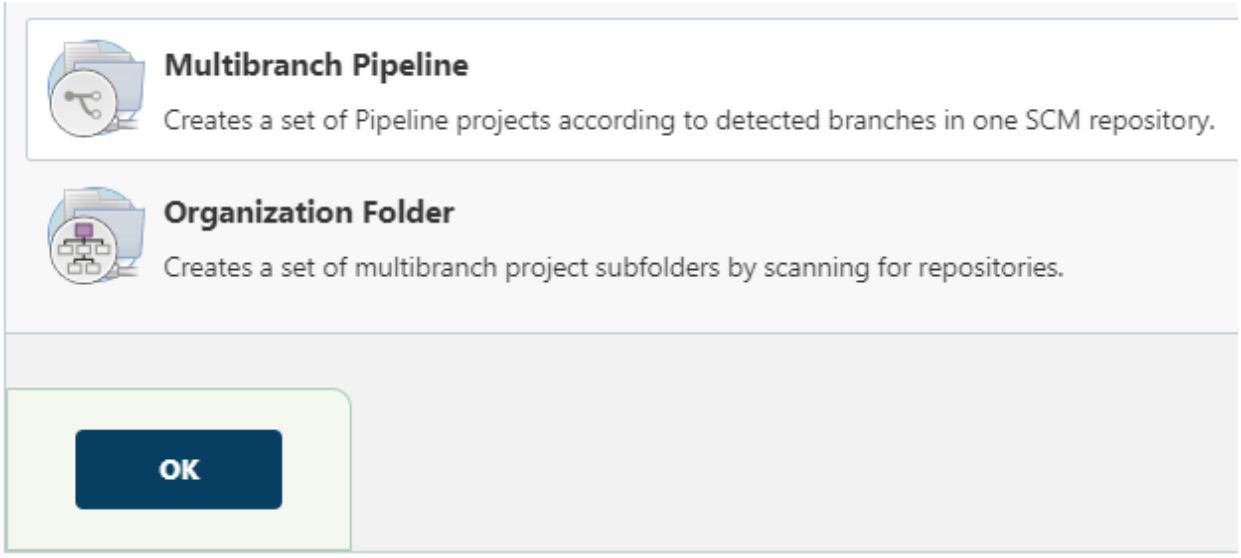
I logged into Jenkins and clicked **New Item** on the left.



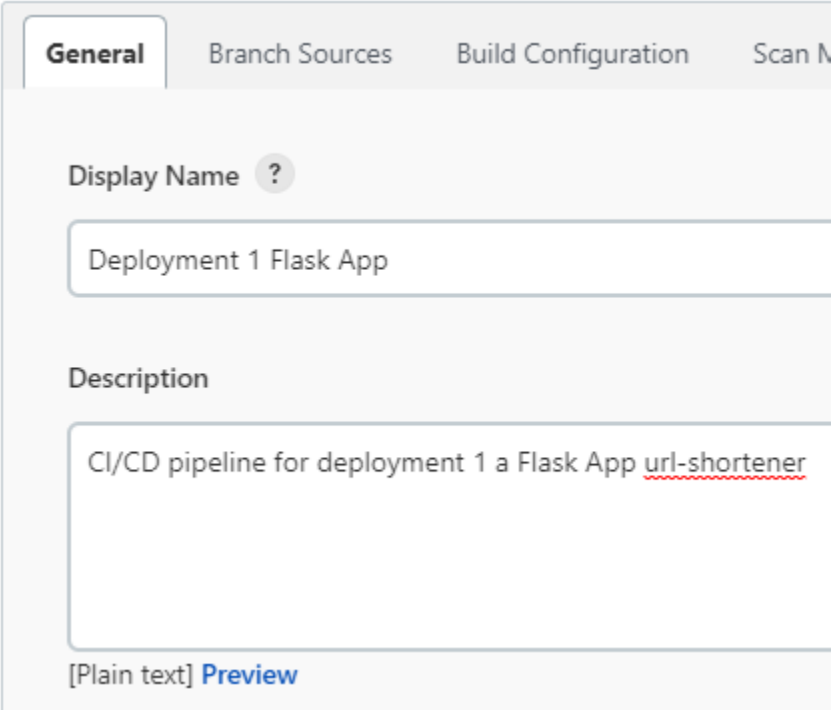
Then I entered a **name(url-shortener)**.



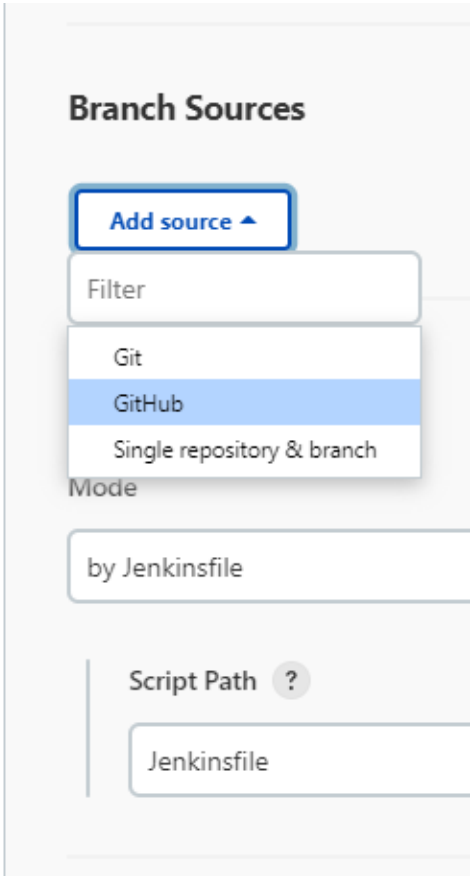
After that I selected the **Multibranch Pipeline** option and clicked **OK**.



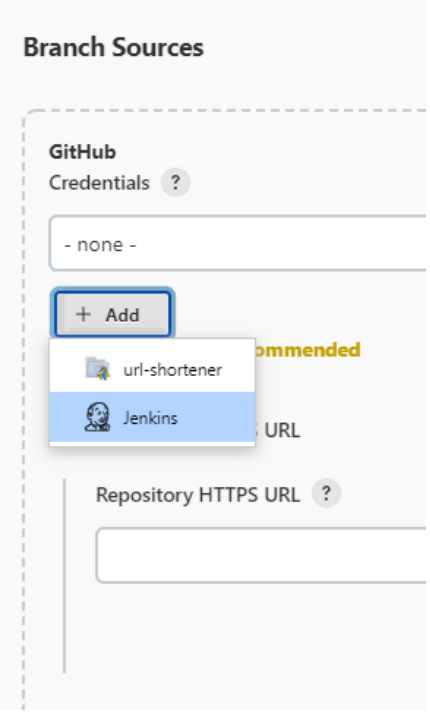
Now I had to select a **Display Name** and **Description**.



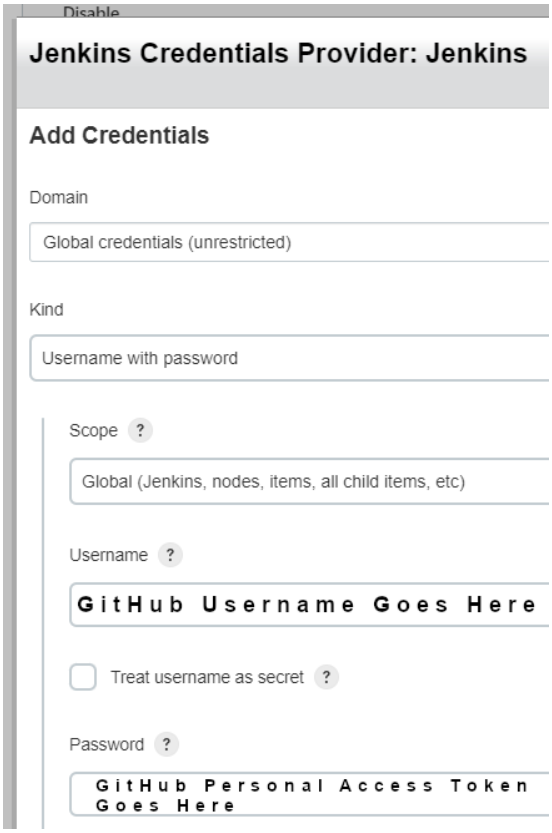
With that done I went to the **Branch Sources** section and clicked **Add source** then **GitHub** to add the deployment GitHub Repository(https://github.com/kura-labs-org/kuralabs_deployment_1)



This next step seems optional because I don't believe I need **credentials** to grab a **public project** from Github. The original and my fork are **both public** but I set the credentials anyway as **practice**. I clicked **Add** then **Jenkins**.

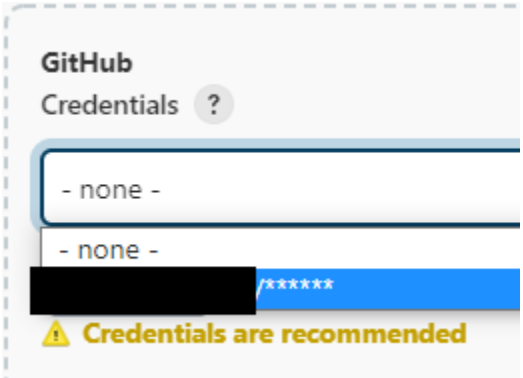


The Jenkins Credentials Provider popped up and I left it on **Username and password for Kind** and set the **Username** as my **GitHub username** and **Password** as the **GitHub Personal access token** I previously generated.

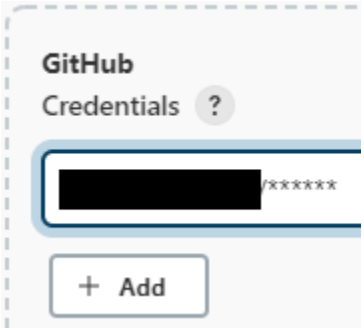


When all the information was entered I clicked **Add** to add the credentials.

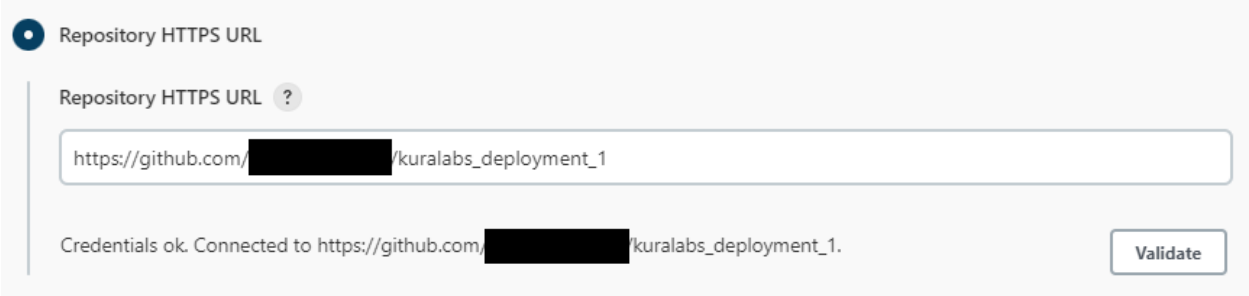
Then once it was added I saw it as an **option to pick which credentials to use**.



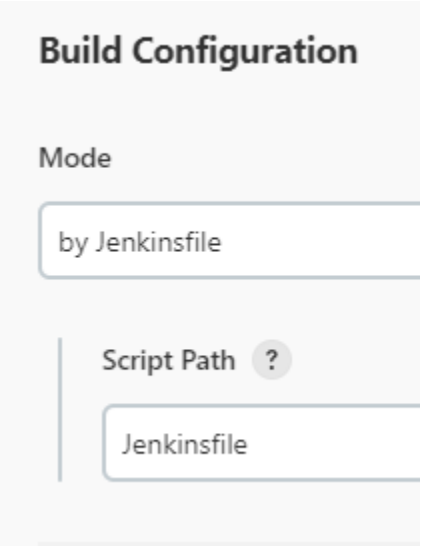
I **selected** the credentials I just added.



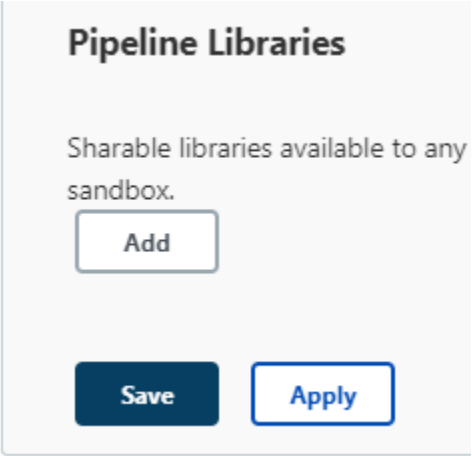
After that I entered the **GitHub url of the repository to build**. I could have used https://github.com/kura-labs-org/kuralabs_deployment_1 or my fork of that, I used **my fork** of it then clicked **Validate**.



I scrolled down and made sure the **Build Configuration** was left at the default(**Jenkinsfile**) below.

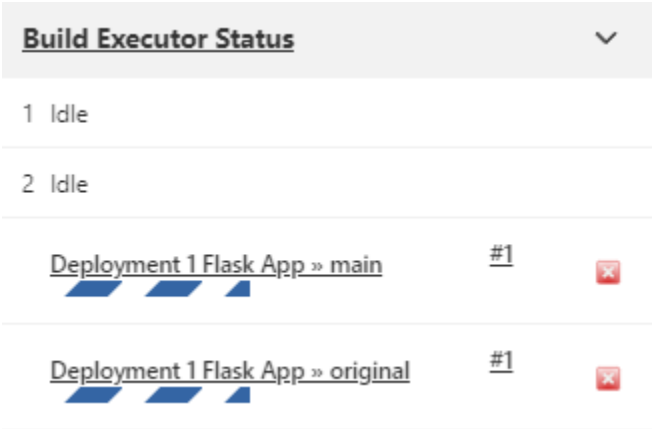


Next I scrolled to the bottom and clicked **Apply** and **Save** to save everything inputted.

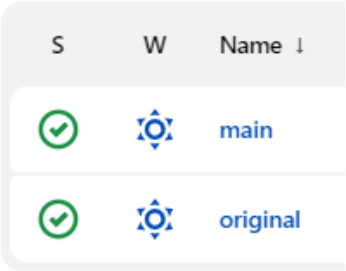


The Build on Jenkins

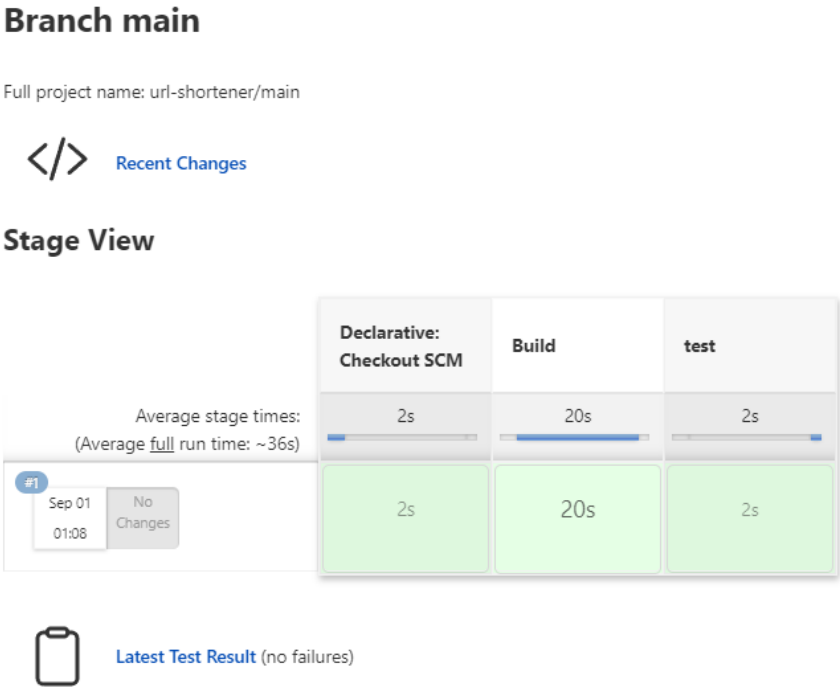
When I finished creating the multibranch build it started **building**(I had two branches).



When it was done building it. It **passed the test** and I got the green check mark.



The **url-shortener** built successfully and passed the test stage. This part is done.



Deploy to Elastic Beanstalk

The first step I did was to **prepare the zip file** with the code by cloning the repository and zipping the files myself.

```
Ubuntu 20.04 on Windows
richard@Executor:/mnt/f/Desktop/Test $ git clone git@github.com:[redacted]/kuralabs_deployment_1.git
Cloning into 'kuralabs_deployment_1'...
remote: Enumerating objects: 115, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 115 (delta 12), reused 35 (delta 12), pack-reused 79
Receiving objects: 100% (115/115), 9.19 MiB | 3.40 MiB/s, done.
Resolving deltas: 100% (59/59), done.
Updating files: 100% (30/30), done.
richard@Executor:/mnt/f/Desktop/Test $ ls
kuralabs_deployment_1
richard@Executor:/mnt/f/Desktop/Test $ cd kuralabs_deployment_1/
richard@Executor:/mnt/f/Desktop/Test/kuralabs_deployment_1 main $ ls
'Deployment-1_Assignment (1).pdf'   Jenkinsfile   Pipfile   Pipfile.lock   README.md   application.py   requirements.txt
   static   templates   test_app.py   urls.json
richard@Executor:/mnt/f/Desktop/Test/kuralabs_deployment_1 main $ zip -q ../kuralabs_deployment_1 -r * .[^.]*
richard@Executor:/mnt/f/Desktop/Test/kuralabs_deployment_1 main $ cd ..
richard@Executor:/mnt/f/Desktop/Test $ ls
kuralabs_deployment_1  kuralabs_deployment_1.zip
richard@Executor:/mnt/f/Desktop/Test $
```

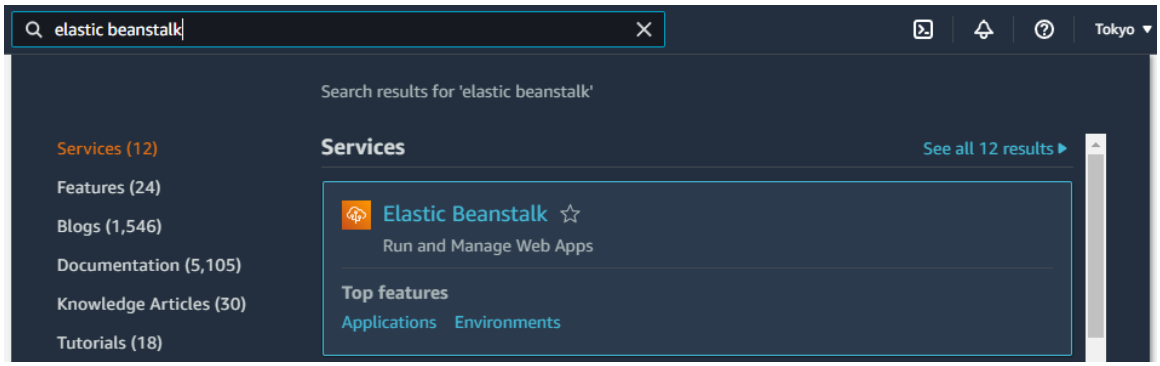
I also wrote a script to do the **same thing** (also on my github as **ziprepo.sh**).

```bash
#!/bin/bash

#This script is to clone the deployment repository and zip it for the elastic beanstalk

#The Url of the GitHub repository
GitGubUrl="https://github.com/kura-labs-org/kuralabs_deployment_1.git"

#The name of the repository folder
RepoFolderName="kuralabs_deployment_1"

#The Name of the Zip file we create
ZipFileName="kuralabs_deployment_1"

#Command to clone the GitHub repository
GitCloneCMD="git clone $GitGubUrl"

#Command to zip the cloned repository
ZipCMD="zip -q ../"$ZipFileName" -r * .[^.]*"

#Check if the zip command is installed
which zip > /dev/null 2>&1

#If it's not installed exit
if [ $? -ne 0 ]; then
    echo "Zip command is not installed"
    exit 1
fi

#Clone the repository and zip it
$GitCloneCMD && cd $RepoFolderName && $ZipCMD && cd ..

#Check if the commands worked
if [ $? -ne 0 ]; then
    echo "Clone and Zip Error"
    exit 1
fi

#successful
echo "Zip successful"
exit 0
```

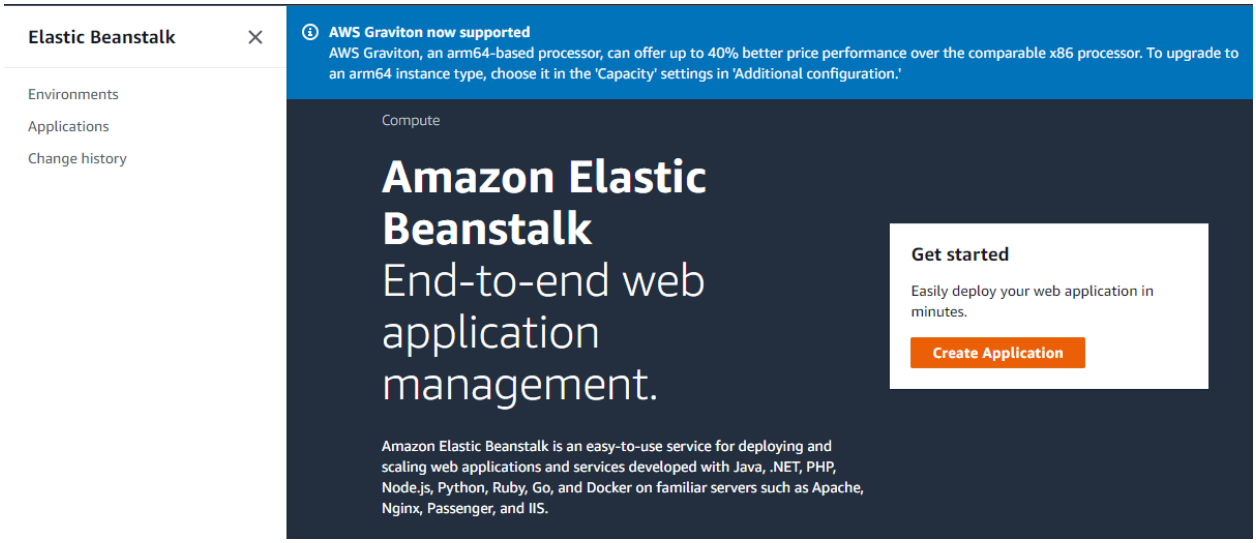
Below is an example of the **output** from the above script.

```
Ubuntu 20.04 on Windows
richard@Tokyo:~/Jenkins-Setup main $ ls
README.md  installjenkins.log  installjenkins.sh  runinstalljenkins.sh  ziprepo.sh
richard@Tokyo:~/Jenkins-Setup main $ ./ziprepo.sh
Cloning into 'kuralabs_deployment_1'...
remote: Enumerating objects: 115, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 115 (delta 12), reused 35 (delta 12), pack-reused 79
Receiving objects: 100% (115/115), 9.19 MiB | 20.10 MiB/s, done.
Resolving deltas: 100% (59/59), done.
Zip successful
richard@Tokyo:~/Jenkins-Setup main $ ls
README.md           installjenkins.sh      kuralabs_deployment_1.zip  ziprepo.sh
installjenkins.log  kuralabs_deployment_1  runinstalljenkins.sh
richard@Tokyo:~/Jenkins-Setup main $
```

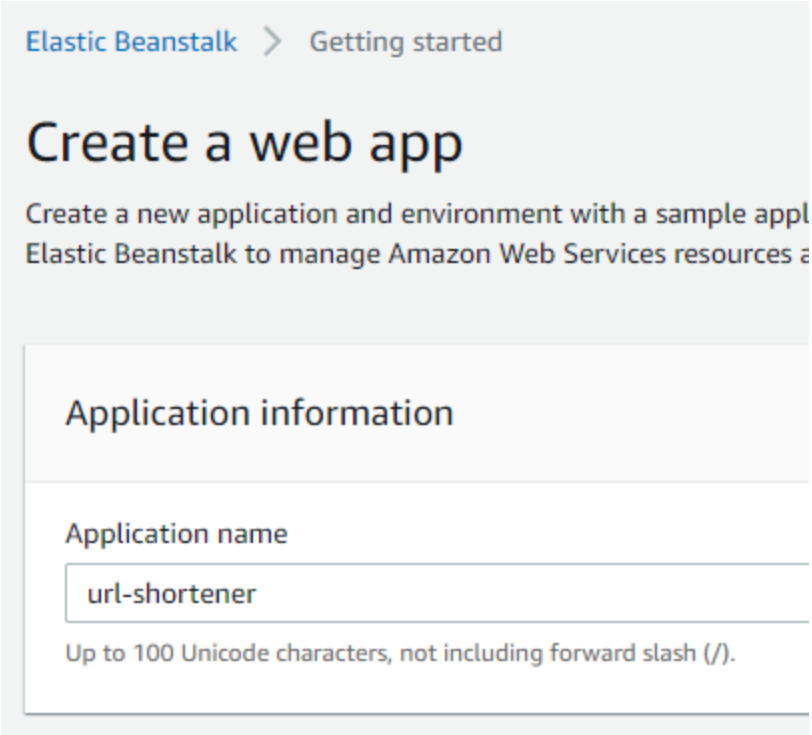
Then I went to the **AWS website**, logged in and searched for **Elastic Beanstalk** and clicked on **Elastic Beanstalk** on the **Services** list.



After the page loaded I clicked on **Create Application** on the right of the page.

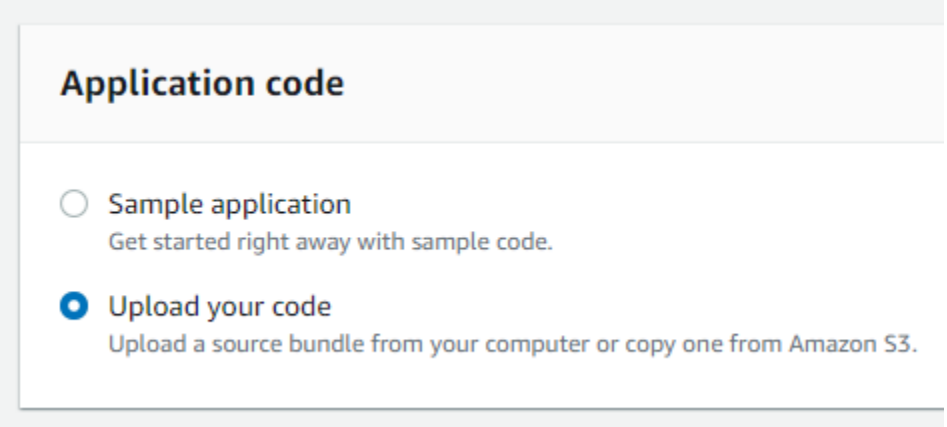


One the new page it loaded I entered an **Application name**(url-shortener).

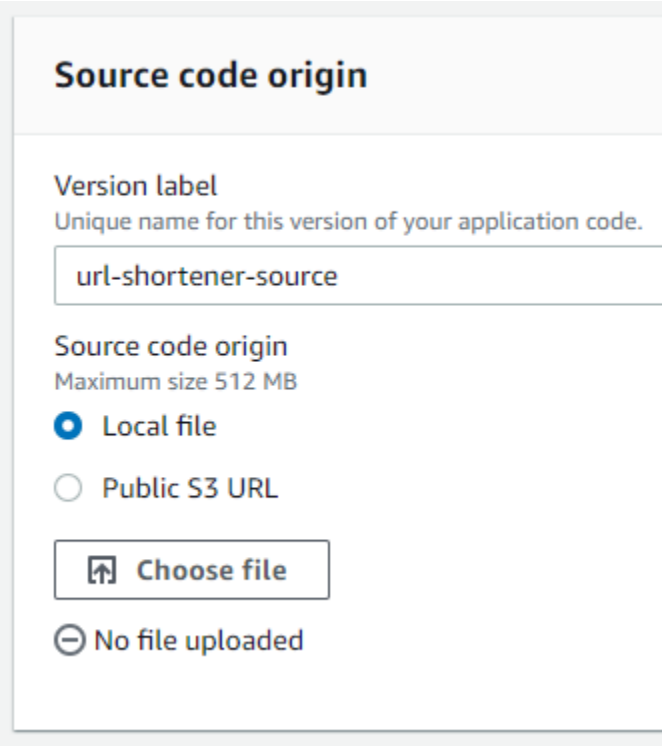


Then I scrolled down to **Platform** and selected **Python** leaving the default options it fills.

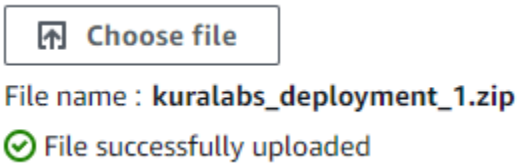
Scrolling further down to **Application code** I selected **Upload your code**.



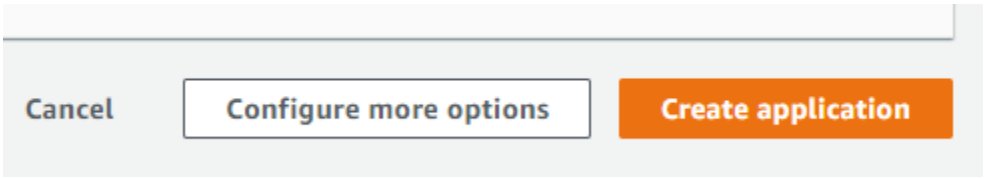
When selected it showed some options to upload the code I selected **Local file** and clicked **Choose file** then selected the correct zip file from the pop up and waited for it to upload the zip file.



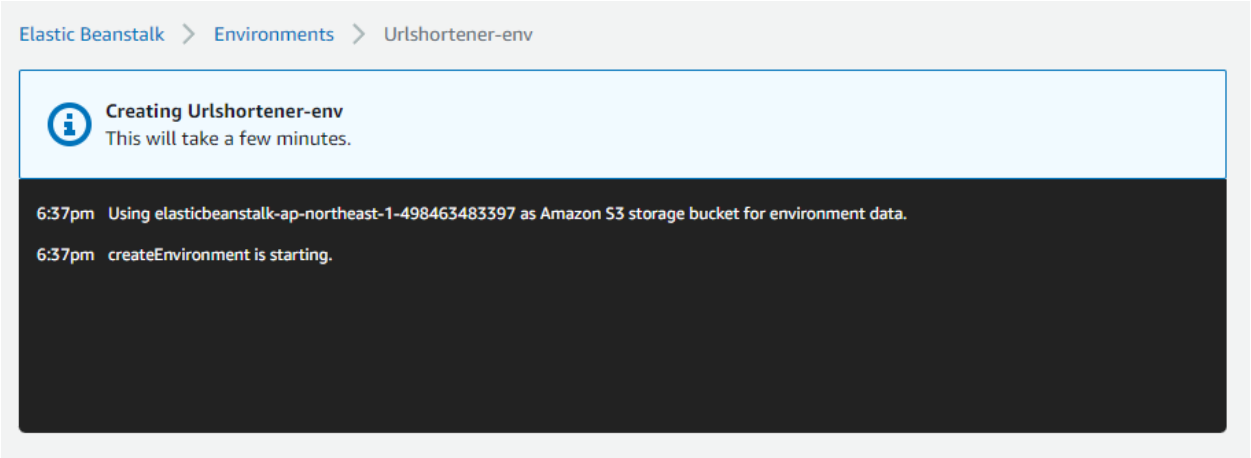
After it finished uploading it showed the **filename with a green checkmark**.



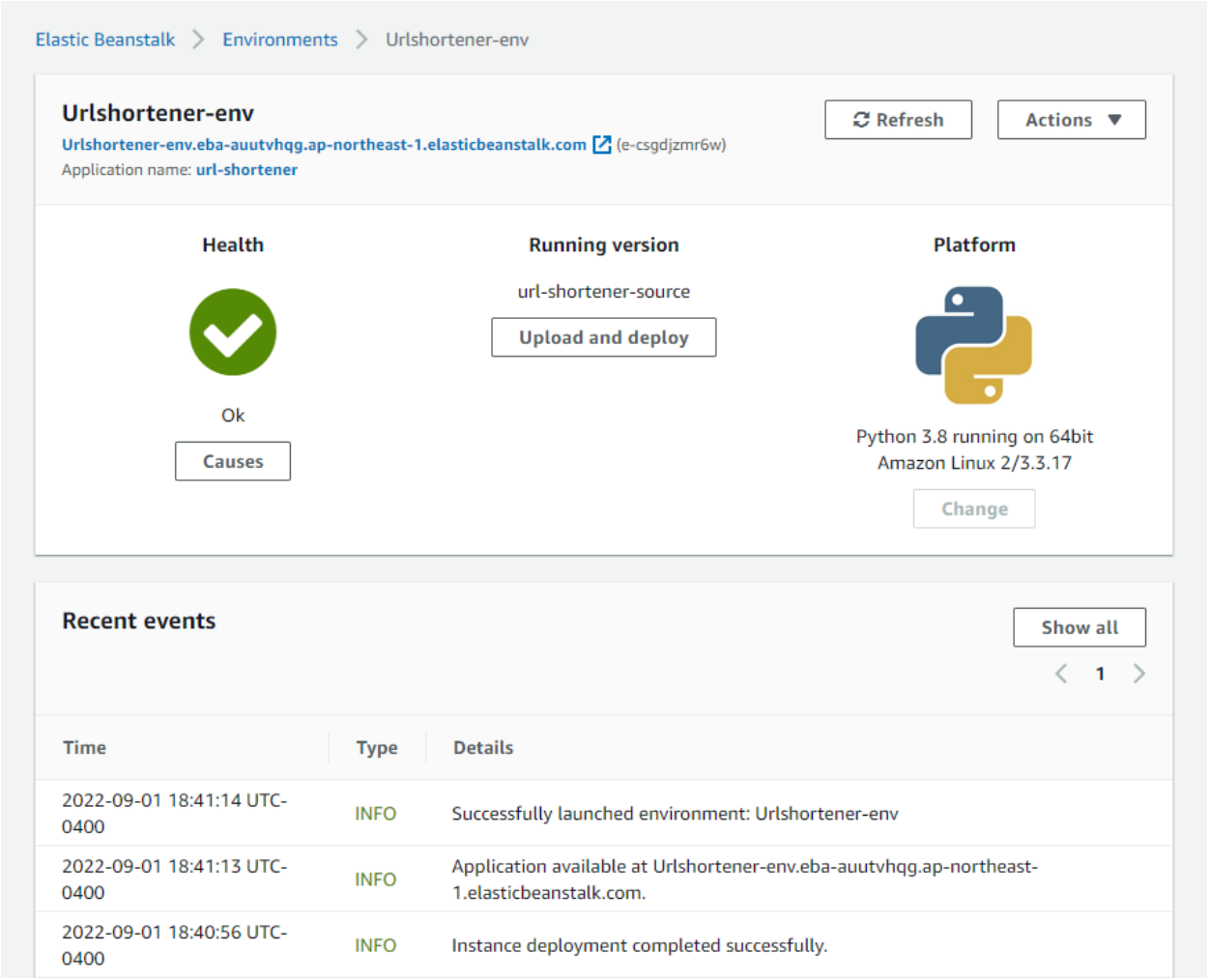
With everything correct I clicked **Create application** at the bottom of the page.



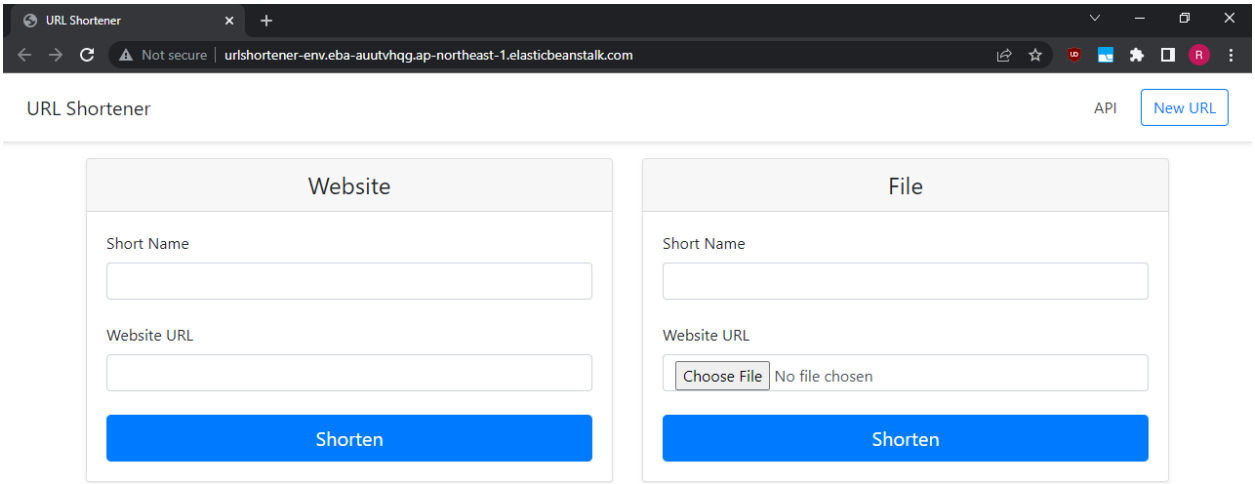
When the application was created it made an **environment** for it and showed the progress so I waited for it to complete.



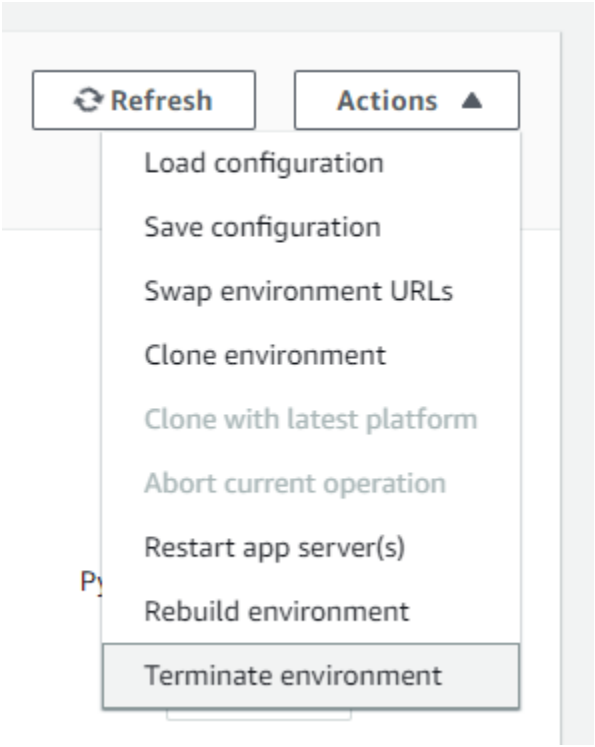
When everything was built it showed the status of **Health** as **Ok**.



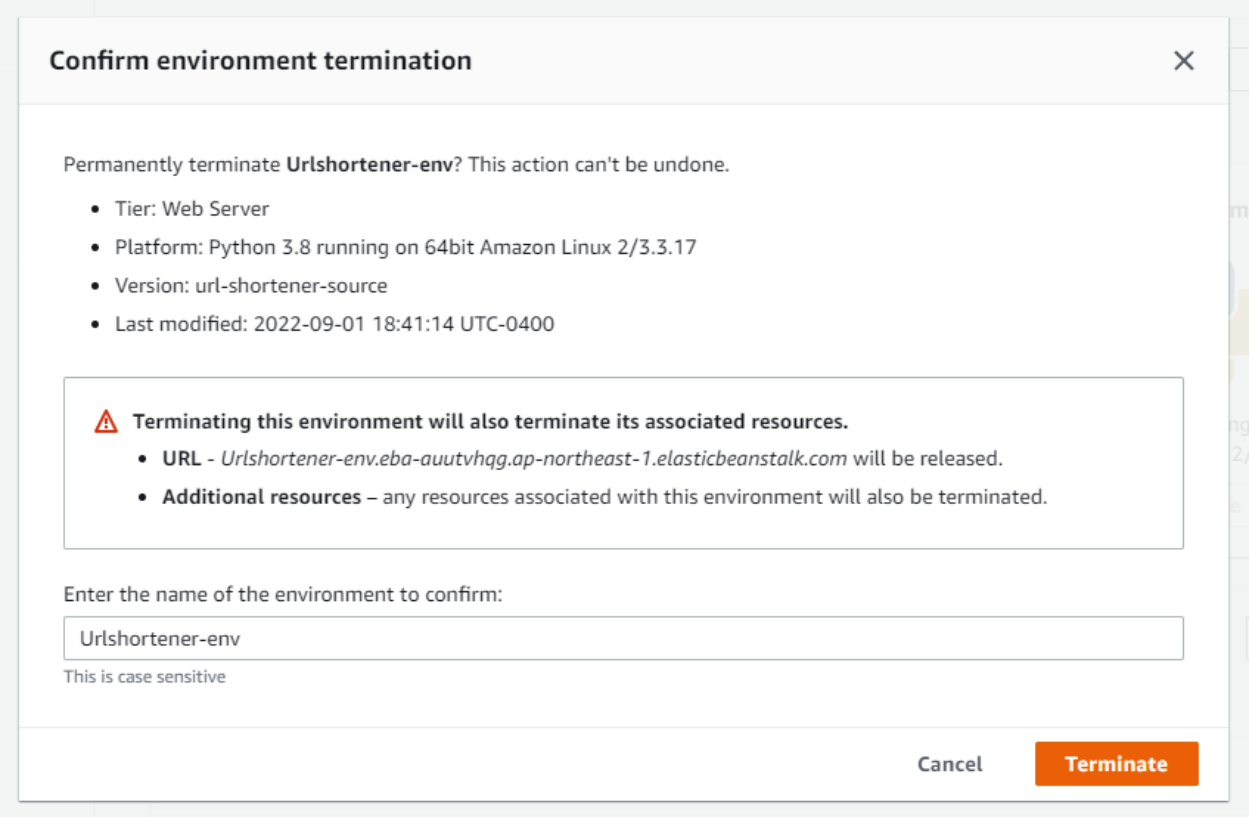
When I opened the **link to the url-shortener app** it created, it worked.



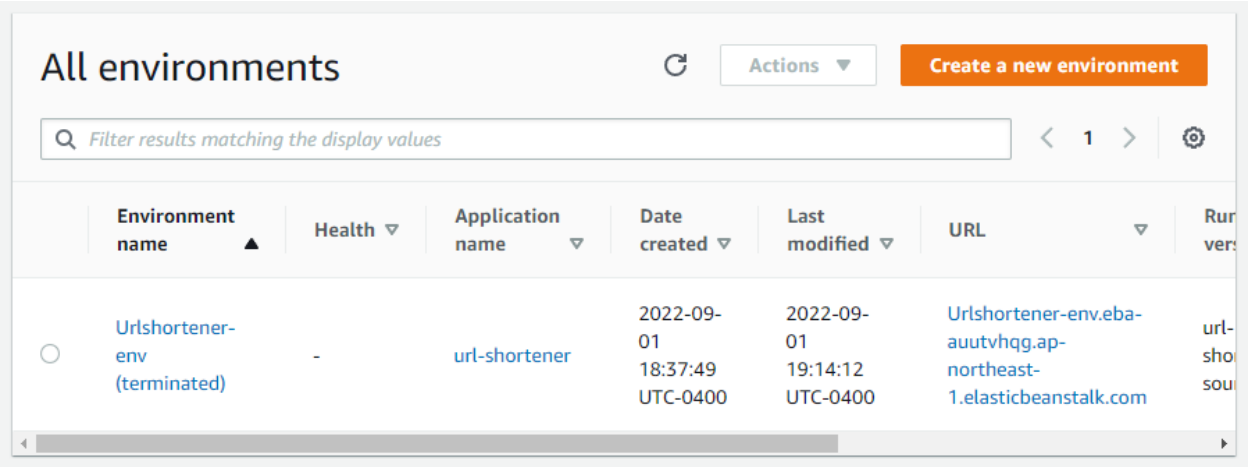
When I was done I **terminated** the environment.



It asked to **confirm** by typing the name of the environment.

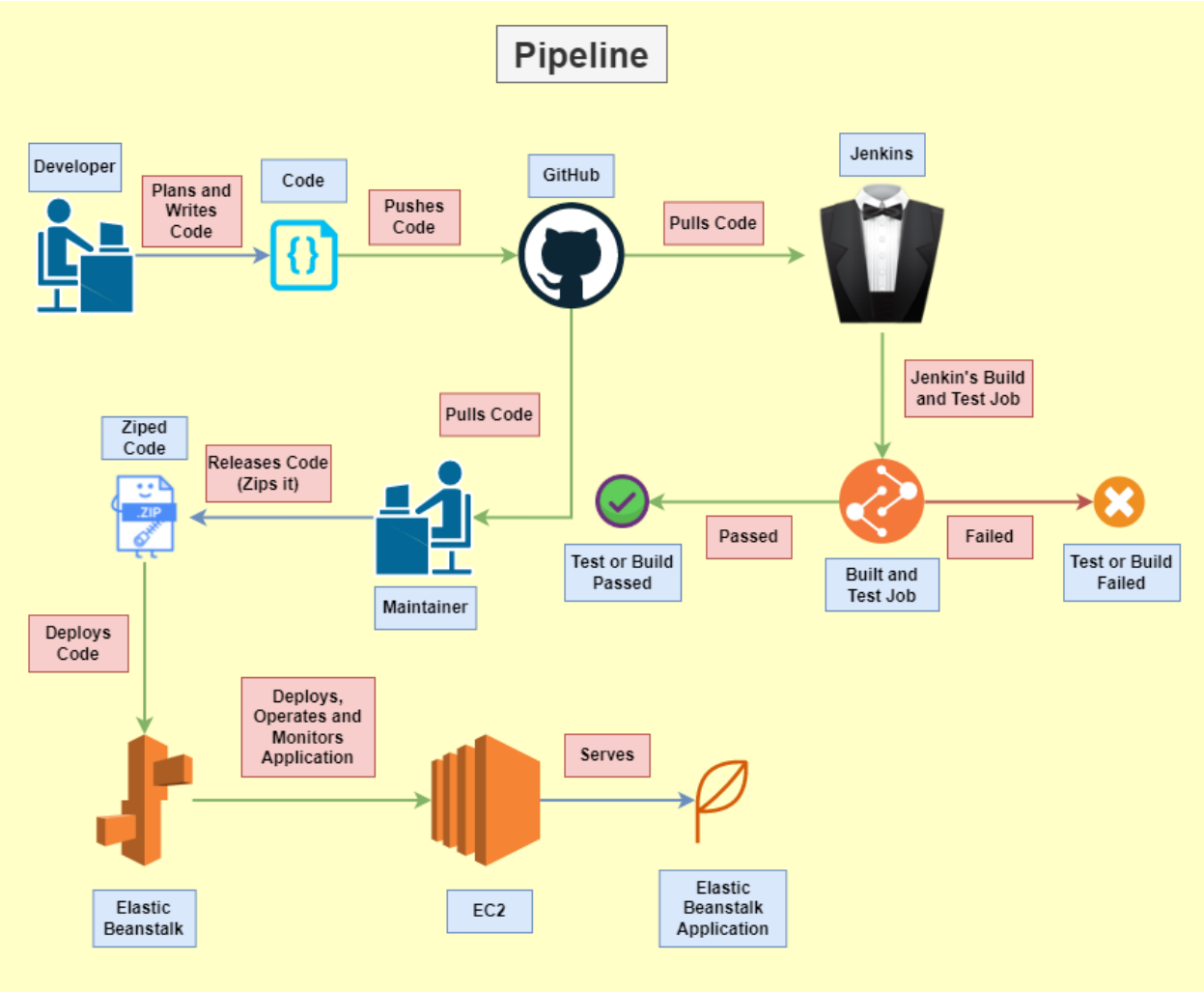


Once that was done it started to terminate it and **eventually it was terminated.** The deployment was **successfully completed**.



Pipeline Diagram

This is how I understand the process I **experienced** would look as a pipeline.



What Could I Improve

If I could learn the **AWS CLI** I assume I could automate creating an EC2 for the Jenkins server to run.

Another part that could be automated is the **configuration of the Jenkins server**, there probably exists a way to do it from the command line instead of using the web page.

There is probably a way to set it up so that Jenkins automatically creates or updates the Elastic Beanstalk application and environment **after a build passes**. I think this might be achieved by adding another stage for deployment or release and using a script that clones the repo, zips it and uses AWS CLI to create or update the Elastic Beanstalk.