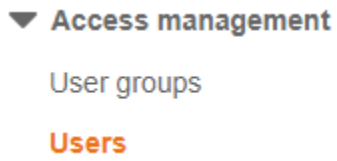
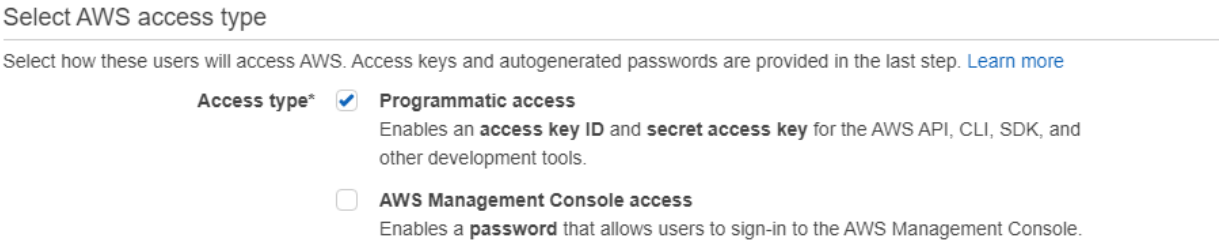
Deployment 4 - Documentation

First I started my EC2 Instance that has Jenkins

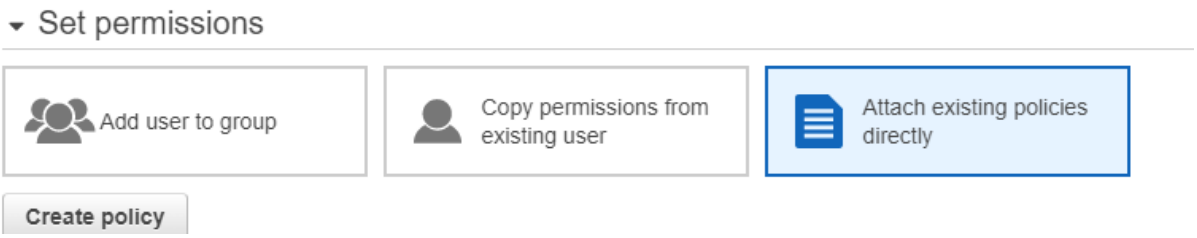
I then logged into my AWS account w/ Admin Privileges and **created an IAM user**.
I went to IAM aws service and selected users section



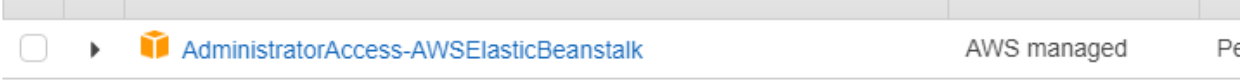
I then added a new user
I called it "**Jenkins-user**" and gave it **Programmatic access**



I then selected Attach existing policies directly

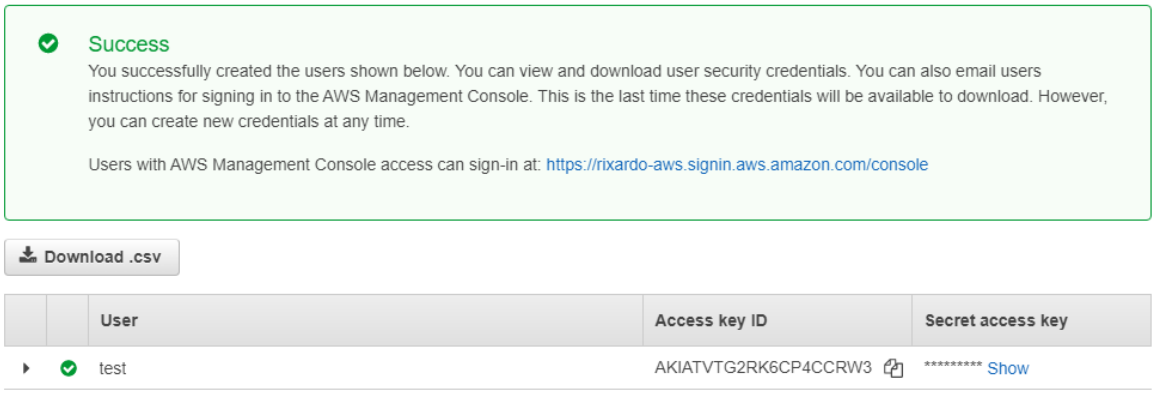


And Selected

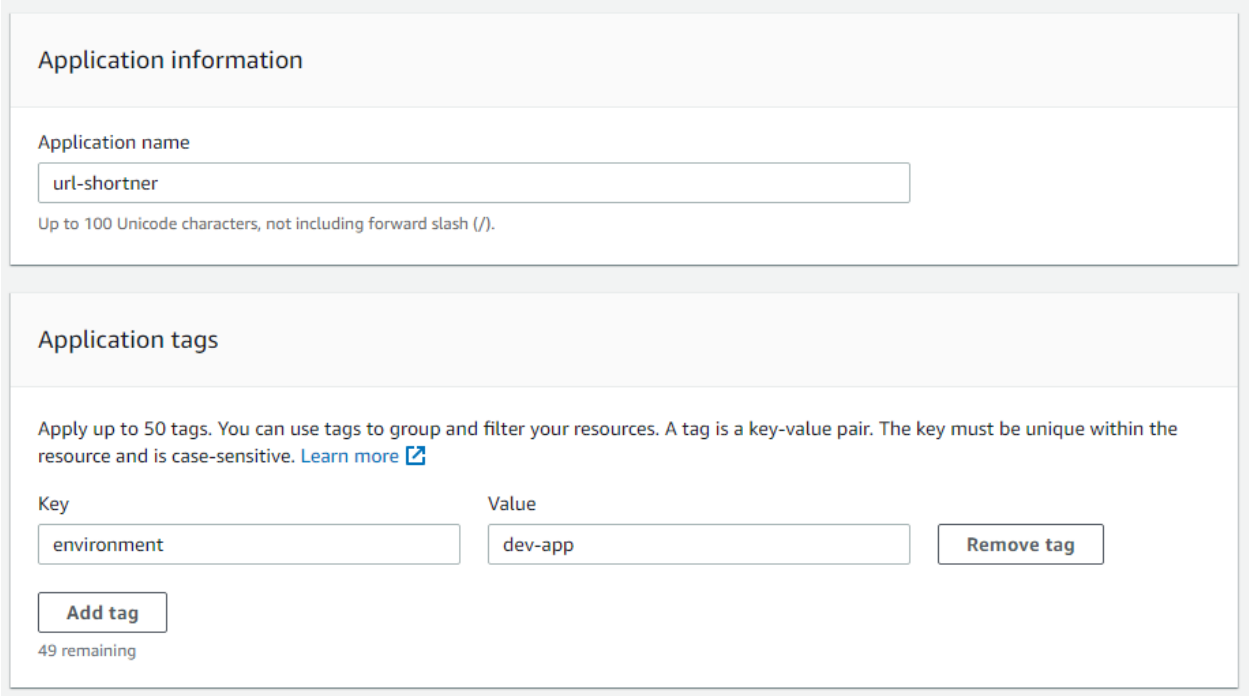


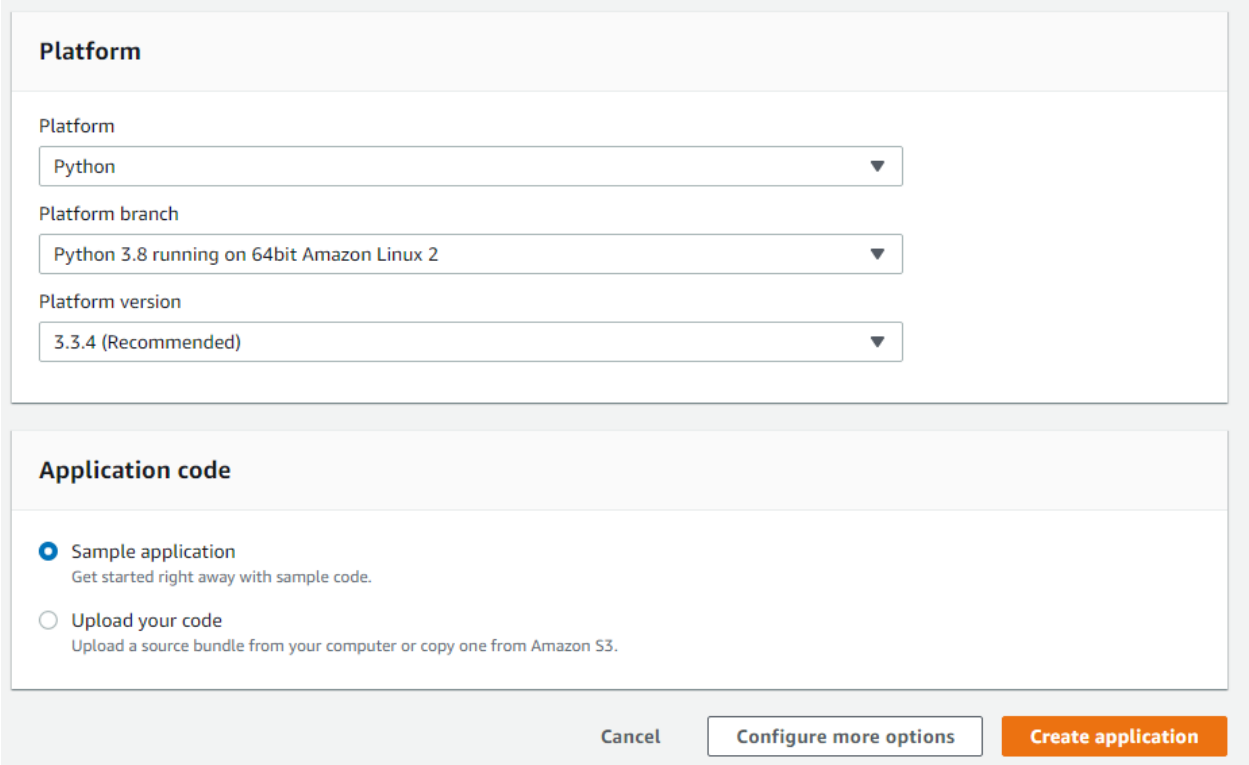
Tags are not necessary

Once you create the user, **download the .csv** that has the user name, Access Key ID and secret access key



Create an Elastic Beanstalk next

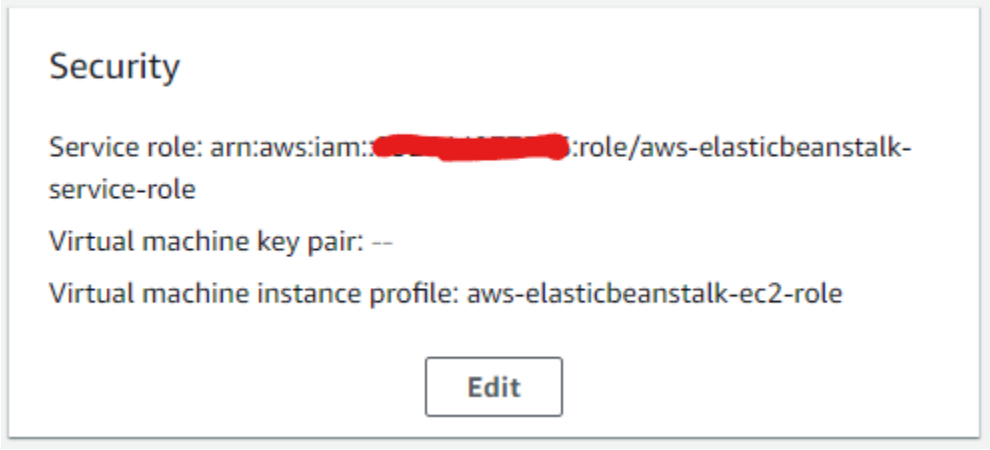


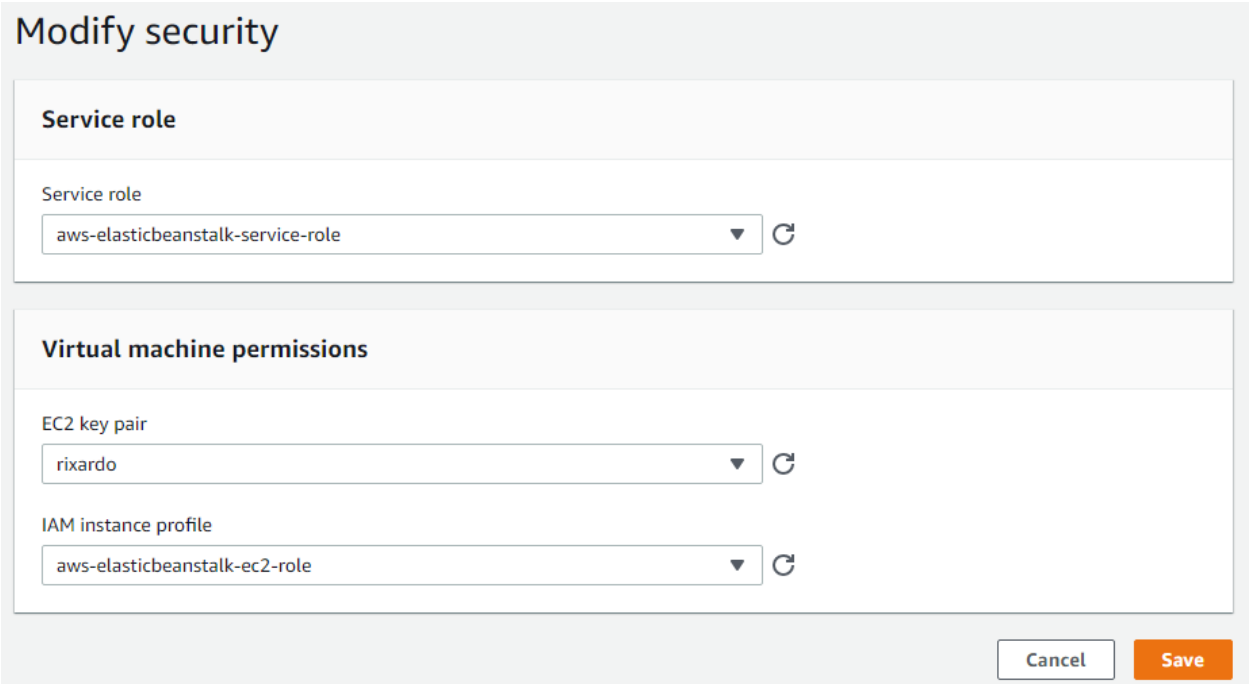


OPTIONAL

Configure more options

Select configure more options and select Security. Add your Key pair to SSH into beanstalk EC2





Save and then press create APP

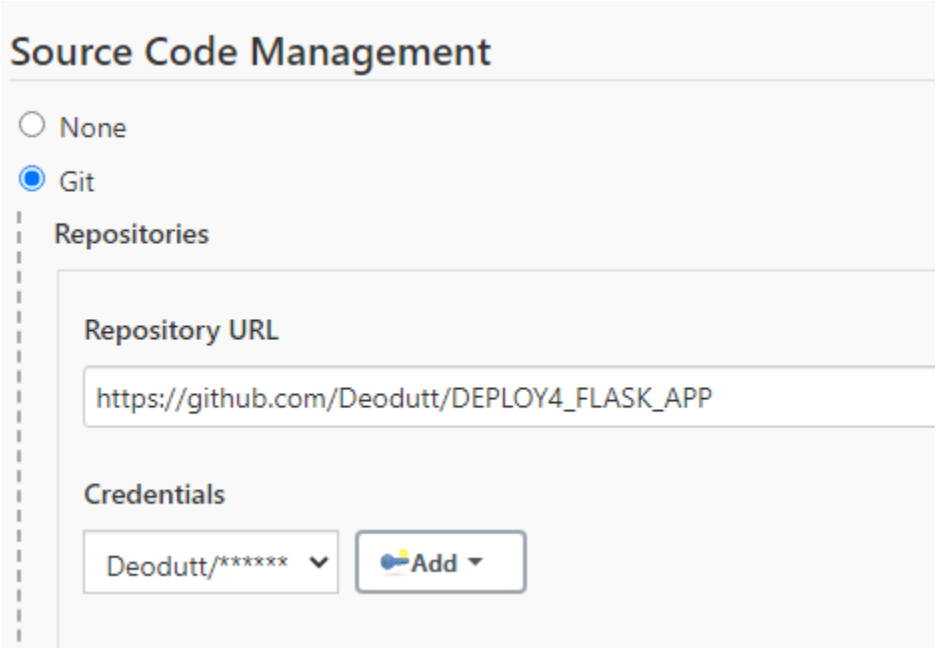
Go into Jenkins and manage Jenkins -> manage Plugins
Install the following plugins
AWSEB Deployment Plugin
CloudBees Credentials Plugin

I then created a Deployment 4 folder that has 3 separate folders inside with each name based on the application.

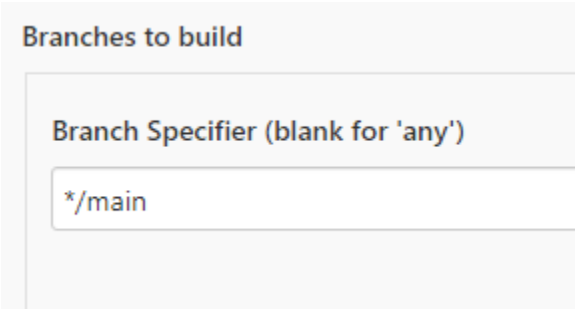
Useful Commands
python3 -m venv venv
pip freeze > requirements.txt
source ./Scripts/Activate

FLASK_APP=app.py flask run

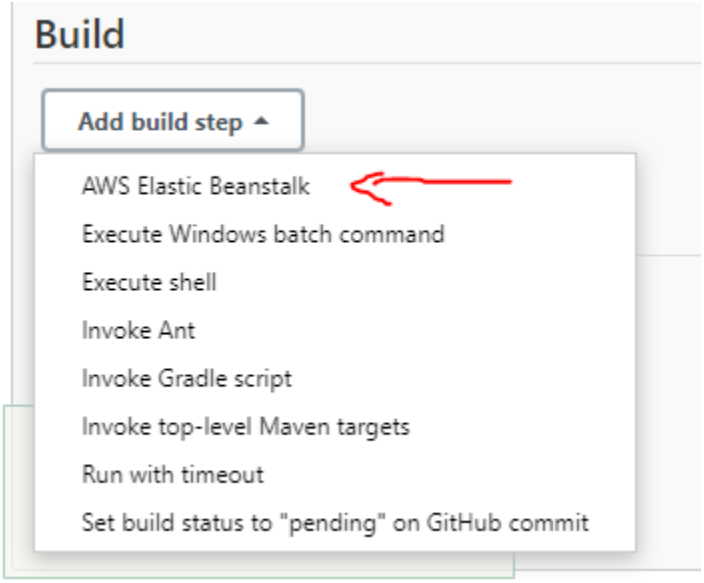
Now create a Freestyle Project inside the corresponding application folder.



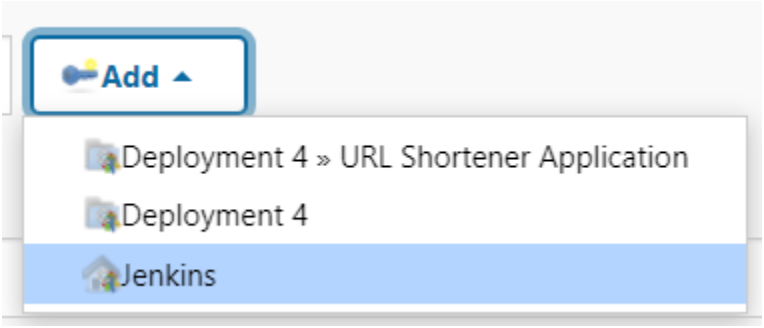
Make sure the branch is set correctly. (If its master, change it to master)



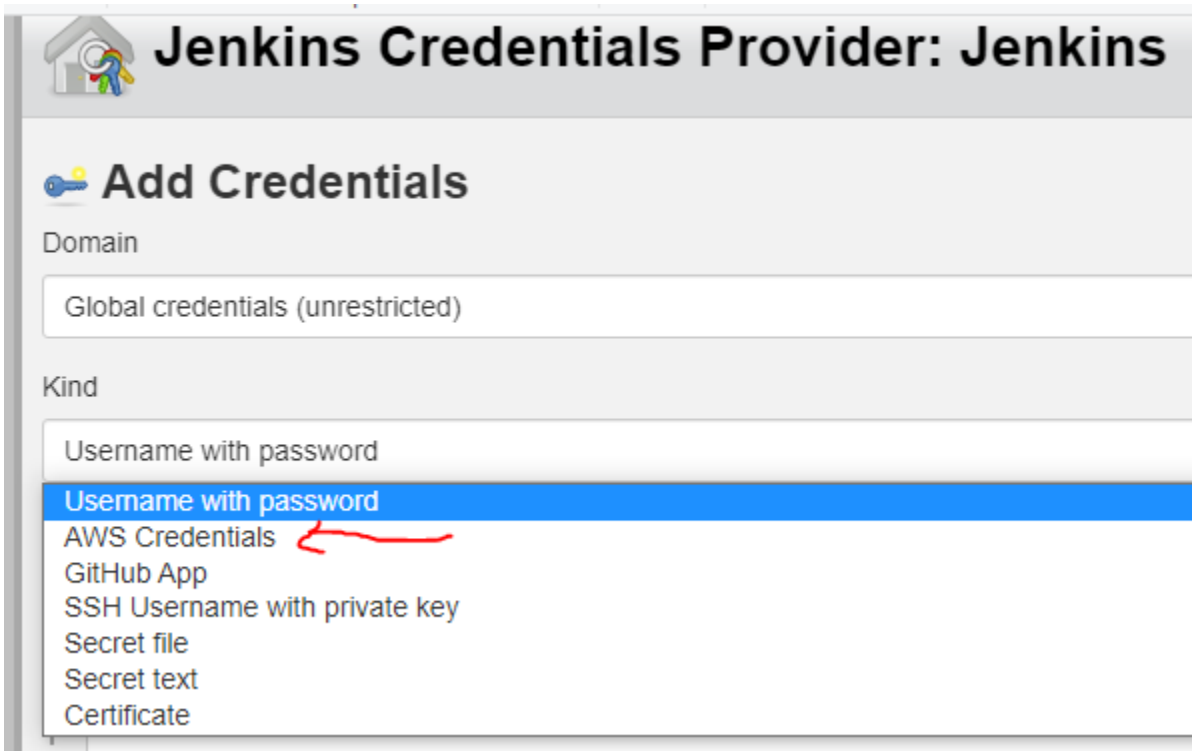
Scroll down to Build and select AWS Elastic Beanstalk



To create a new credentials select Jenkins



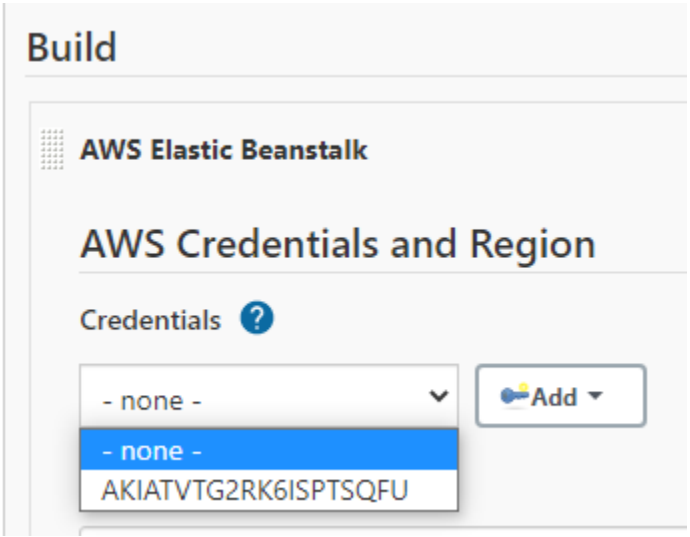
Under kind, select AWS Credentials



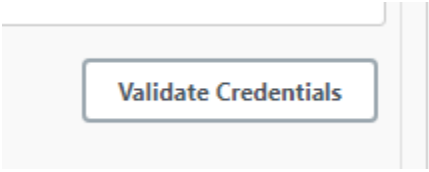
For ID just put a "**Jenkins-user**"

For **Access KEY ID** and **Secret Access Key** - Use the credentials from the IAM user you downloaded.

Press Add and now select the credentials from the dropdown



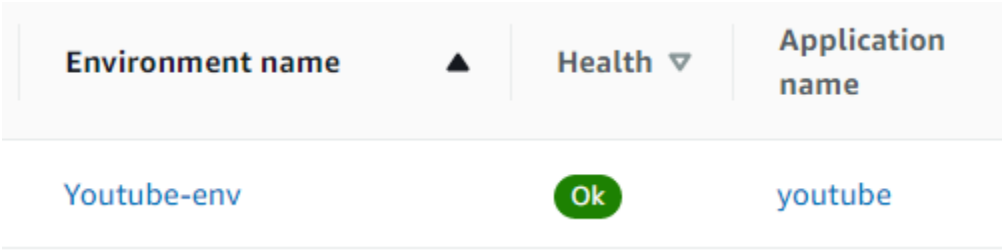
Select the AWS Region where the Elastic BeanStalk was created
Then press **Validate Credentials** in the bottom right



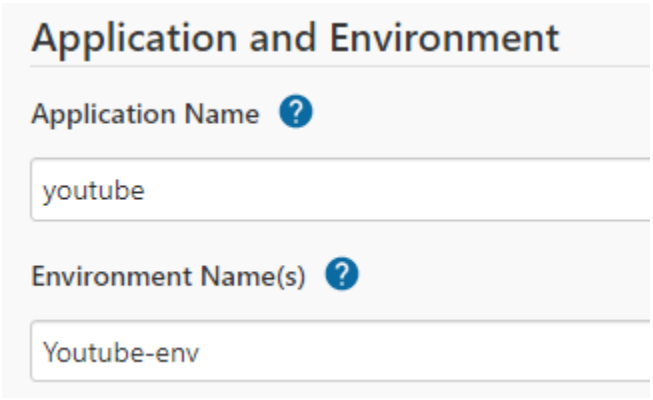
You should see a response like this…

- Building Client (credentialId: 'Jenkins-user', region: 'us-east-1')
- Testing Amazon S3 Service (endpoint: https://s3.amazonaws.com)
- Buckets Found: 4
- Testing AWS Elastic Beanstalk Service (endpoint: https://elasticbeanstalk.us-east-1.amazonaws.com)
- Applications Found: 1 (youtube)

Go back to AWS Beanstalk and find the environment you created. Make a note of the **Environment name** and **application name**

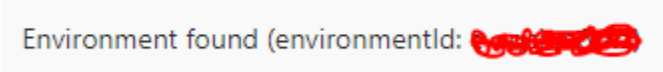


Scroll down to Application and Environment and enter the correct info from AWSEB,

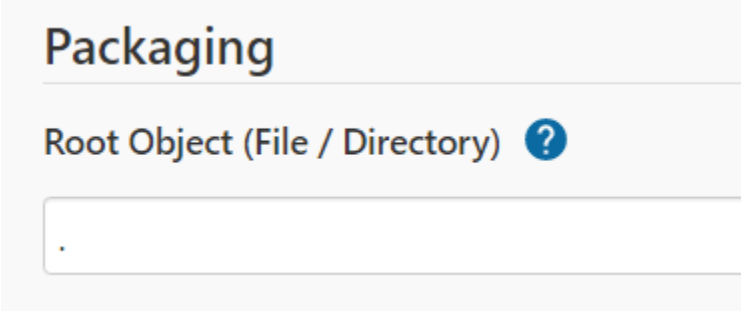


Then press validate coordinates

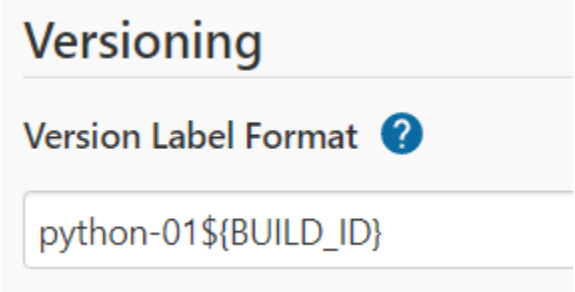
You should see something like this..



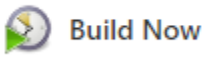
In Packaging, add a “.” to the Root object



In versioning put **python-01${BUILD_ID}**



Then press SAVE and **BUILD NOW**



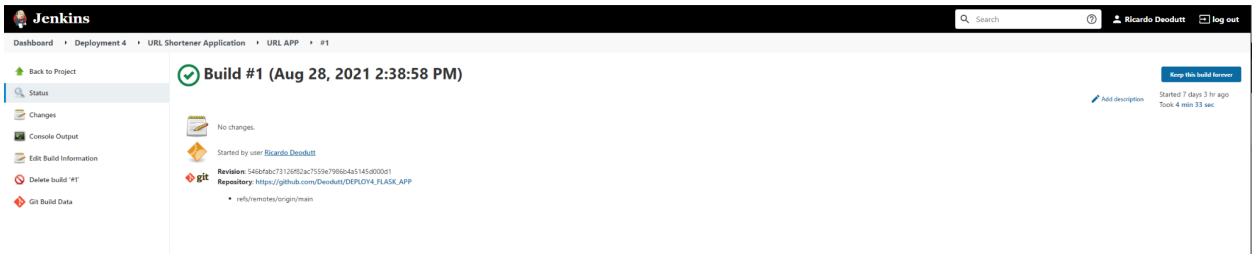
Should take about ~5 minutes to build successfully

Once you have a successful build, Go back to AWS Elastic Bean and click on the **URL** for your environment.

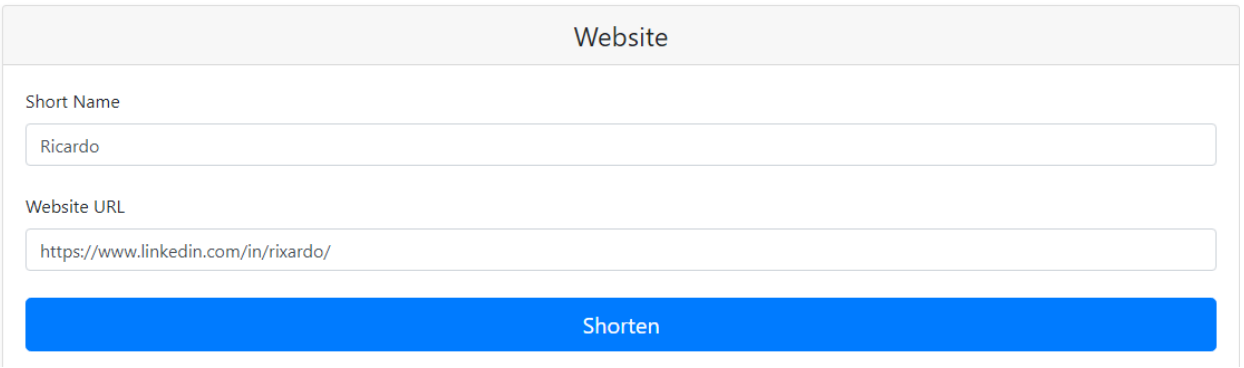
URL Shortener

Souce: https://github.com/Deodutt/DEPLOY4_FLASK_APP

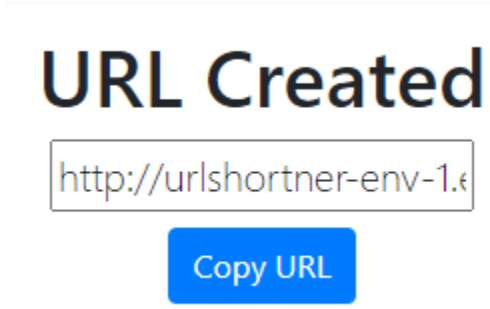
Successful Build:



Default Web Page:



Web page once I enter information and press shorten

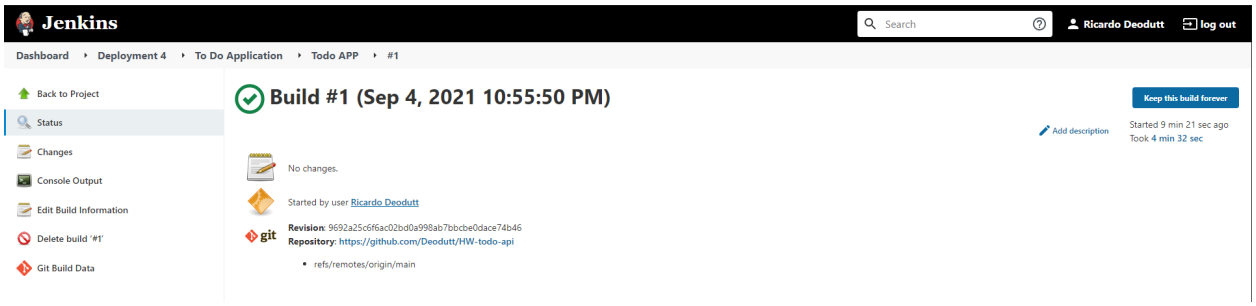


This takes me to my Linkedin Page

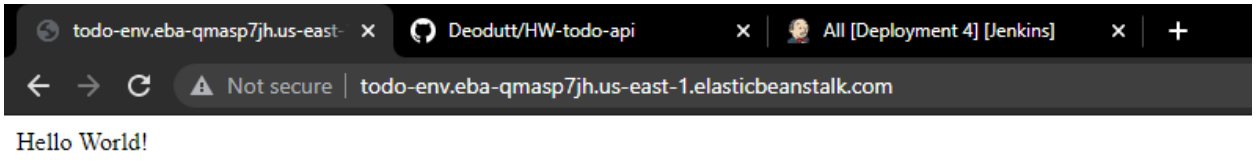
Todo

Source: https://github.com/Deodutt/HW-todo-api

Successful Build:

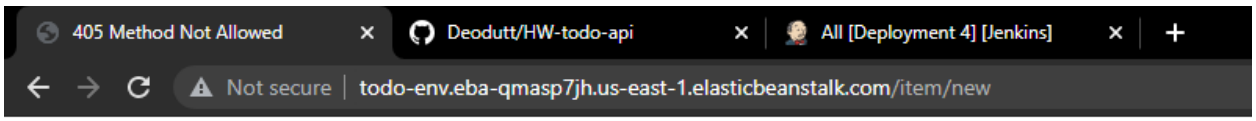


Route: /



Hello World!

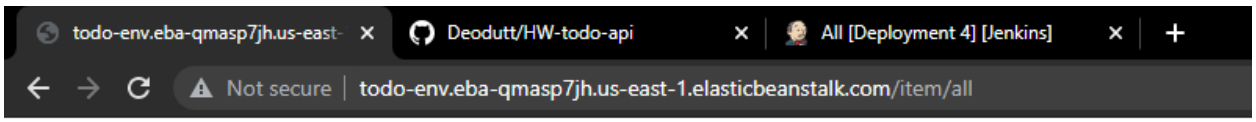
Route: /item/new



Method Not Allowed

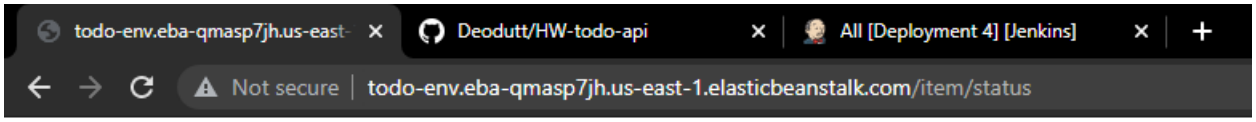
The method is not allowed for the requested URL.

Route: /item/all




```
{
    "count": 0,
    "items": []
}
```

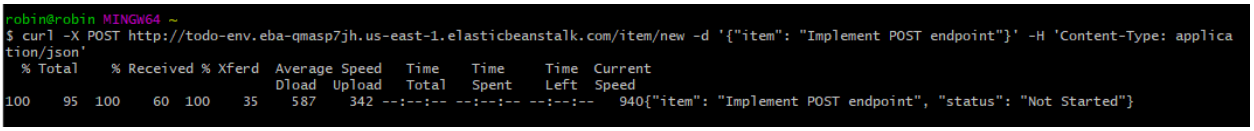
Route: /item/status



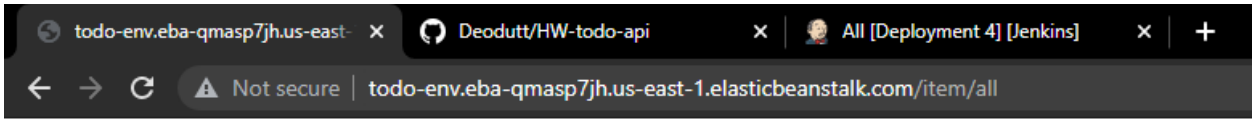

```
{'error': 'Item Not Found - None'}
```


To update the database run the following commands...

```
curl -X POST
http://todo-env.eba-qmasp7jh.us-east-1.elasticbeanstalk.com/item/new -d
'{"item": "Implement POST endpoint"}' -H 'Content-Type: application/json'
```

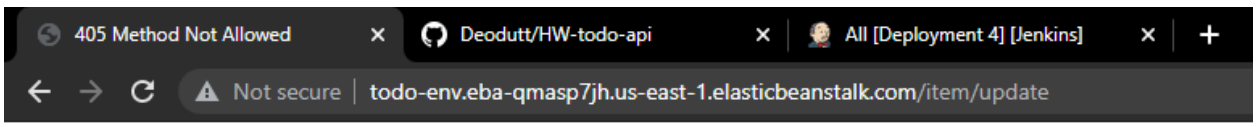


Route: /item/all




```
{
    "count": 1,
    "items": [
        [
            "Implement POST endpoint",
            "Not Started"
        ]
    ]
}
```

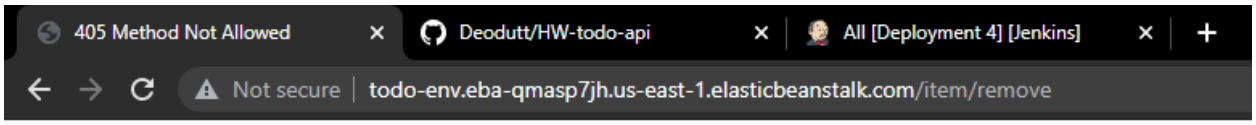

Route: /item/update



Method Not Allowed

The method is not allowed for the requested URL.

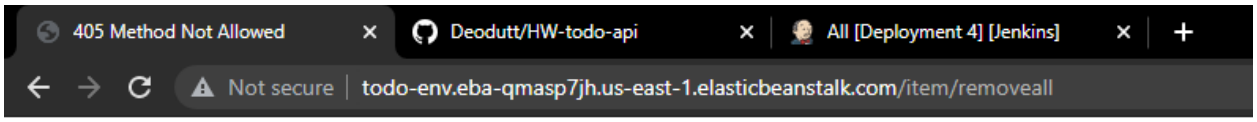
Route: /item/remove



Method Not Allowed

The method is not allowed for the requested URL.

Route: /item/removeall



Method Not Allowed

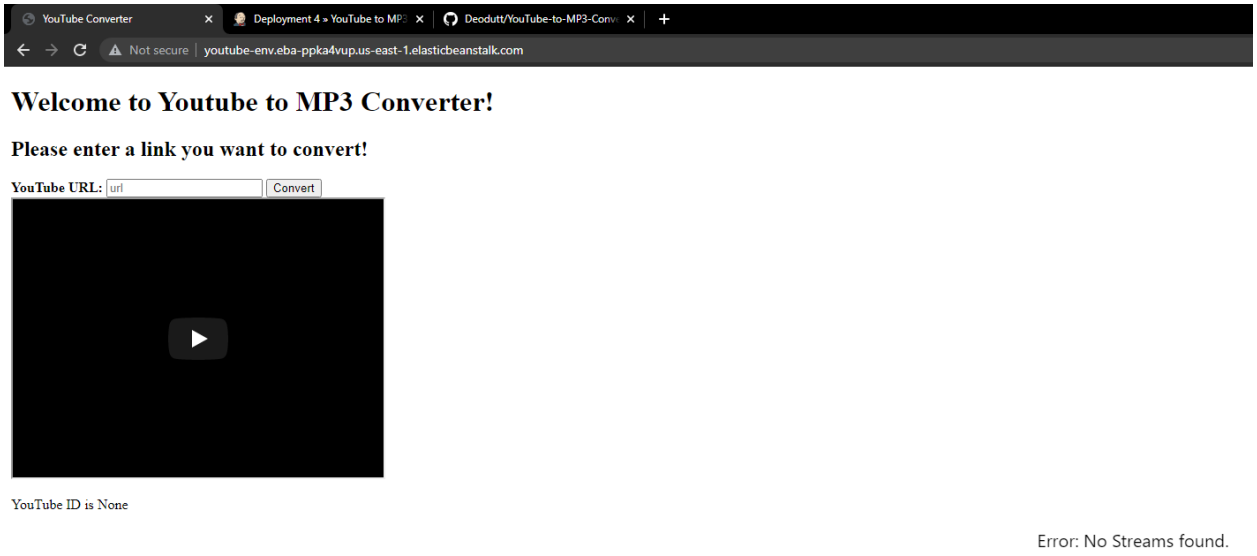
The method is not allowed for the requested URL.

YouTube

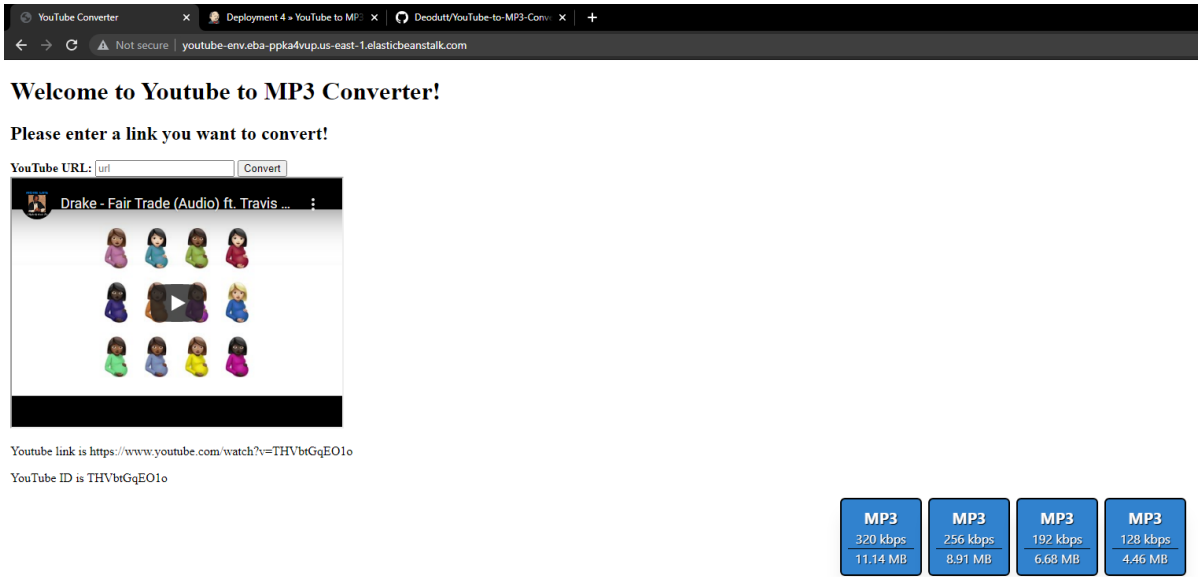
Source: https://github.com/Deodutt/YouTube-to-MP3-Converter-API

Successful Build

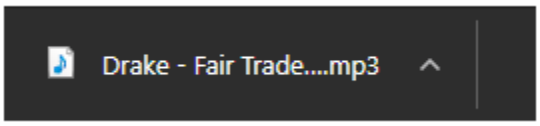
Default web page when I click on the URL from Elastic Beanstalk



Web page once I enter an URL and press convert.



Successful download after selecting a download option



YouTube Converter

(This is for educational purposes)

As a person who listens to music a lot, I wanted to make this application to make my life a little easier. I prefer to work on projects that benefit me. For example, I made a VPN application so I can use it. I prefer to download music online and put it on my phone rather than using streaming services. I do not like paywalls so I thought this was a great project for me. It's a weird way of listening to music but I have been doing it for a long time since Limewire days.

While looking up ideas, I faced some issues.I tried looking up official YouTube APIs to download videos but I found out it's not supported because it's against their terms of services so I had to look for different options. I came across this repository and thought it would be great to add on. The repo creator supplies an API that I could use to embed into a website. It gives the user different options to download videos in different qualities. While creating the application, I used Ibrahima's repository as a template. His repository was a great way to understand how to write to HTML files and use variables in them. Everything was simple and easy to understand. While creating the application, I found this really great way of getting the YouTube ID and thought it would be great to incorporate it.

For my application, once a user enters the URL inside the web page and presses convert, it would do a GET method. This would then call the converter() function inside of application.py and that would get the string the user entered. The application will then call a id_grabber function from the helper.py file, and that will use urlparse to parse the URL and extract the youtube ID from the URL. It goes to many different cases to incase the user enters a different youtube link format. Once it gets the ID it will return the ID in a variable called youtube_id. The youtube_id will then concat with a string to the API and be assigned to a variable called api_converter_link. This is then returned into a function called render_templates() which basically renders a template from the template folder which has an index.html file. This will basically let the HTML page use variables.

example

```
{% if youtube_id %}
<p>YouTube ID is  {{youtube_id}}</p>
{% endif %}
```

URL Shortener Application: https://github.com/Deodutt/DEPLOY4_FLASK_APP
To Do Application: https://github.com/Deodutt/HW-todo-api
YouTube Converter Application: https://github.com/Deodutt/YouTube-to-MP3-Converter-API