Azubi Project 2: Vividart Studio - Beanstalk2

Author	Olatunji Olayinka Oluwaseun
Contributors	Brian Mathenge, Adade Sedom Percy, and Pauline Andege Omondi
GitHub Repository	https://github.com/olatunji-weber/MineVividarts_studio.git

Vividart Studio Project

This project is to build an image processor utilizing python, lambda, S3 and docker.

I collaborated with Brian Mathenge, Adade Sedom Percy, and Pauline Andege Omondi in doing the project.

First of all, I forked the repo from: https://github.com/olatunji-weber/Vividarts_studio.git

Then cloned it to workstation via CLI

```
> git clone https://github.com/olatunji-weber/Vividarts_studi
```

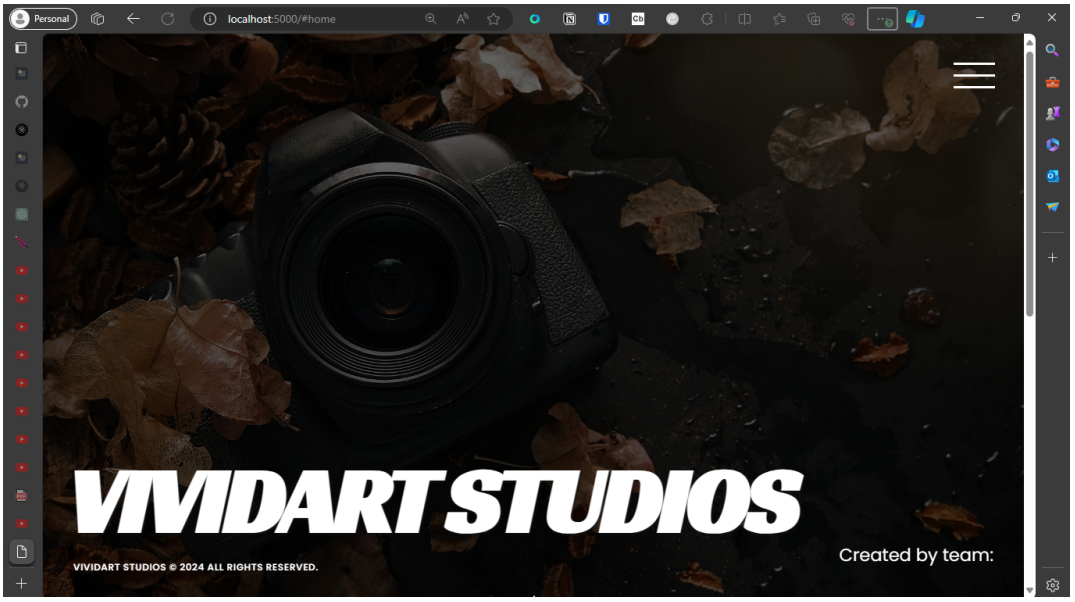
Then created “requirements.txt” file and entered the line:

```
boto3==1.34.16
Flask==3.0.0
```

Then used “pip install requirements.txt” at the command prompt.

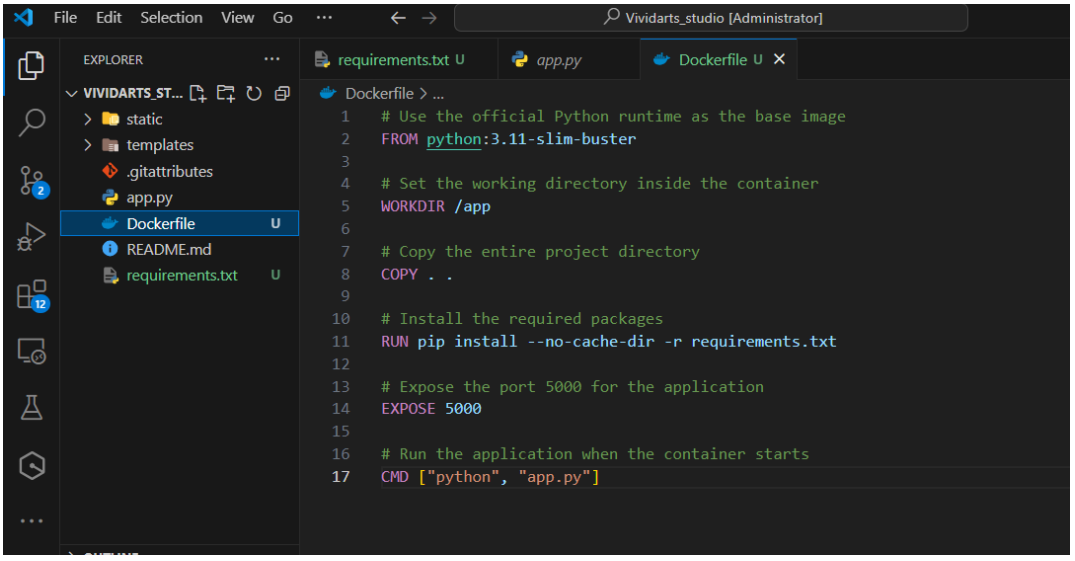
```
PS C:\Users\Olatunji Olayinka\Documents\FormerHDD\Programming\Node-React-NPM&More\IntroToNode&Express\AzubiParole\Azubi2023Work-CloudClass\AzubiProj
ects\Vividarts_studio-Project2\Vividarts_studio> pip install -r requirements.txt
Collecting Flask==2.1.1 (from -r requirements.txt (line 1))
  Downloading Flask-2.1.1-py3-none-any.whl.metadata (3.9 kB)
Collecting Werkzeug>=2.0 (from Flask==2.1.1->-r requirements.txt (line 1))
  Downloading werkzeug-3.0.1-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.0 (from Flask==2.1.1->-r requirements.txt (line 1))
  Downloading Jinja2-3.1.3-py3-none-any.whl.metadata (3.3 kB)
Collecting itsdangerous>=2.0 (from Flask==2.1.1->-r requirements.txt (line 1))
  Downloading itsdangerous-2.1.2-py3-none-any.whl.metadata (2.9 kB)
Collecting click>=8.0 (from Flask==2.1.1->-r requirements.txt (line 1))
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: colorama in c:\users\olatunji olayinka\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\loca
lcache\local-packages\python311\site-packages (from click>=8.0->Flask==2.1.1->-r requirements.txt (line 1)) (0.4.4)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.0->Flask==2.1.1->-r requirements.txt (line 1))
  Downloading MarkupSafe-2.1.5-cp311-cp311-win_amd64.whl.metadata (3.1 kB)
Downloading Flask-2.1.1-py3-none-any.whl (95 kB)
     ---------------------------------------- 95.2/95.2 kB 194.0 kB/s eta 0:00:00
Downloading click-8.1.7-py3-none-any.whl (97 kB)
     ---------------------------------------- 97.9/97.9 kB 267.4 kB/s eta 0:00:00
Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
     ---------------------------------------- 133.2/133.2 kB 187.5 kB/s eta 0:00:00
Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
     ---------------------------------------- 226.7/226.7 kB 223.6 kB/s eta 0:00:00
Downloading MarkupSafe-2.1.5-cp311-cp311-win_amd64.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2, Flask
  WARNING: The script flask.exe is installed in 'C:\Users\Olatunji Olayinka\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p
0\LocalCache\local-packages\Python311\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed Flask-2.1.1 Jinja2-3.1.3 MarkupSafe-2.1.5 Werkzeug-3.0.1 click-8.1.7 itsdangerous-2.1.2
PS C:\Users\Olatunji Olayinka\Documents\FormerHDD\Programming\Node-React-NPM&More\IntroToNode&Express\AzubiParole\Azubi2023Work-CloudClass\AzubiProj
ects\Vividarts_studio-Project2\Vividarts_studio>
```

Ran “python app.py” in order to launch the application. Then observed the following page when accessing localhost via port 5000 in a web browser:



Containerized the cloned application by creating Dockerfile for Flask application, via these steps:

Created a file called Dockerfile in the root directory of the project in order to containerize the application (right next to app.py, requirements.txt, etc.).



Used the official Python runtime as the base image

```
FROM python:3.11-slim-buster
```

Set the working directory inside the container

```
WORKDIR /app
```

Copied the entire project directory

```
COPY . /app
```

Installed the required packages

```
RUN pip install --no-cache-dir -r requirements.txt
```

Exposed the port 5000 for the application

```
EXPOSE 5000
```

Added the command to run the application when the container starts

```
CMD ["python", "app.py"]
```

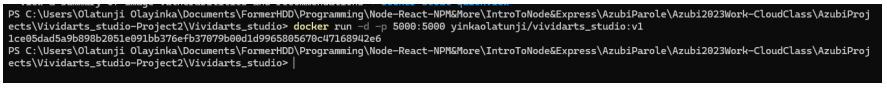
After creating the Dockerfile, I built and ran the Docker container using the following commands:

Build the Docker image:

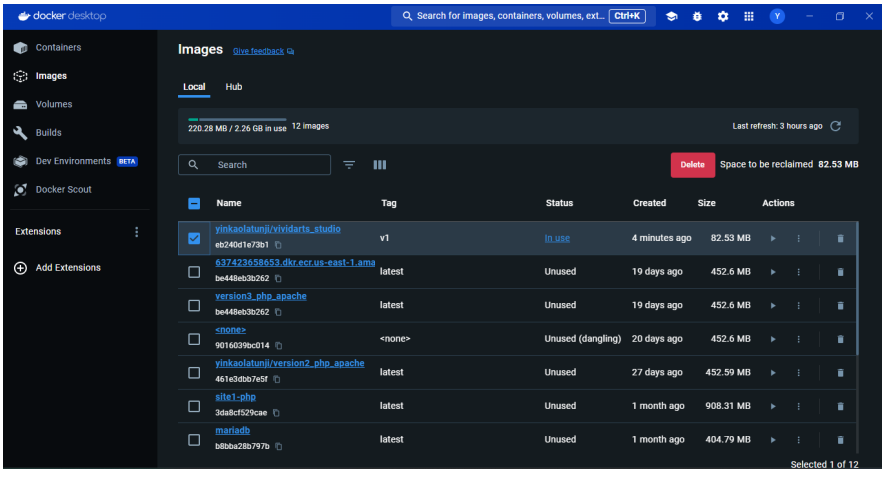
```
docker build -t yinkaolatunji/vividarts_studio:v1 .
```

Run the Docker image:

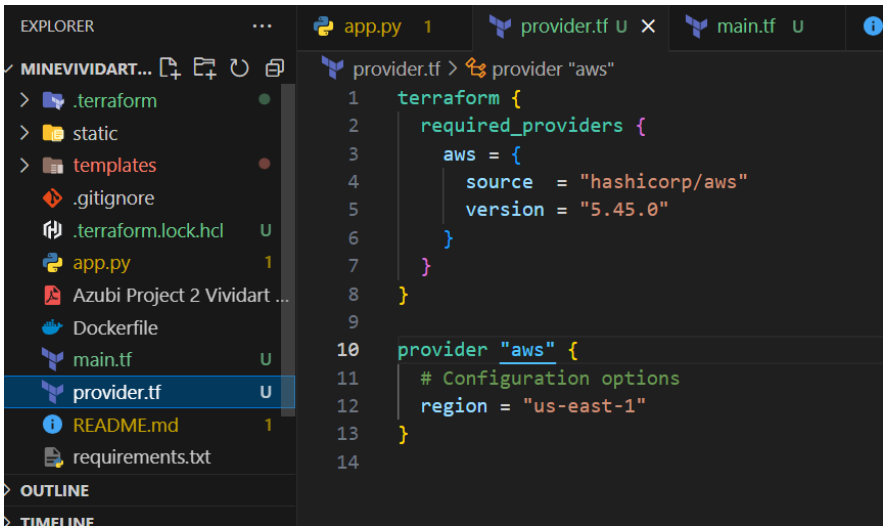
```
docker run -d -p 5000:5000 yinkaolatunji/vividarts_studio:v1
```



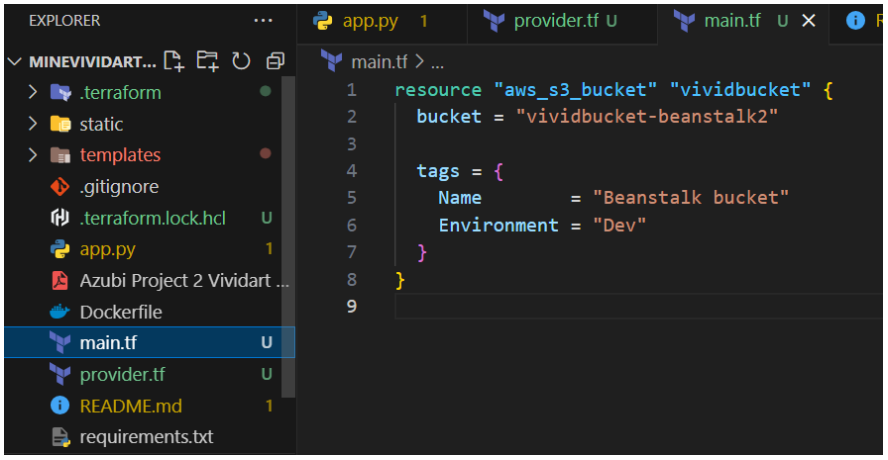
Here is the created docker image in Docker Desktop:



Then I created an S3 bucket named “vividbucket-beanstalk2” which is where we’ll be uploading the image to using Terraform. Here is the “provider.tf” file specifying the provider to be used to create the infrastructure.



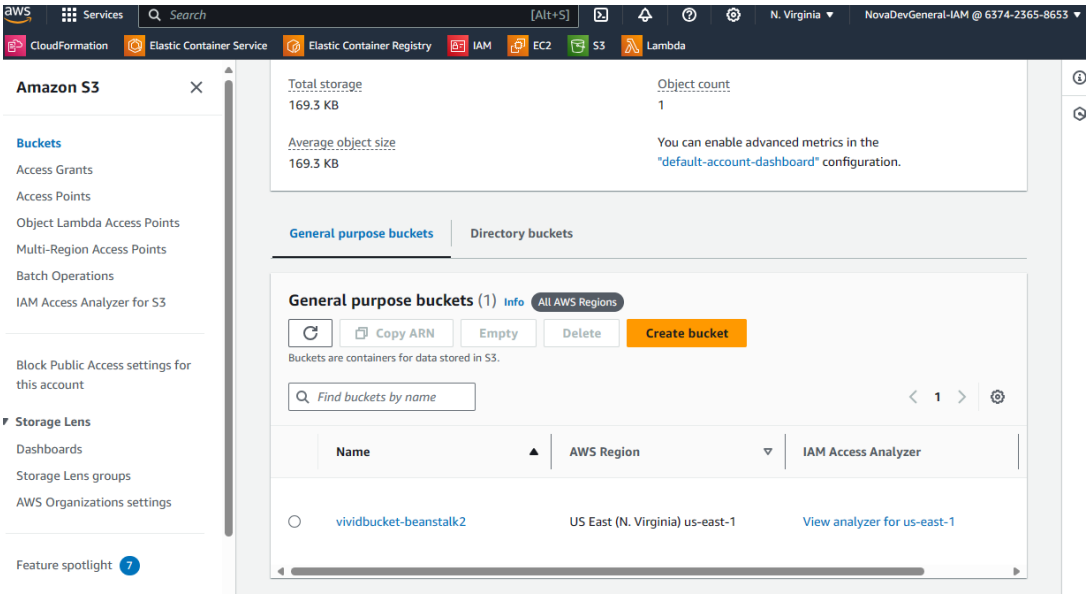
Here is the “main.tf” file specifying the s3 bucket to create.



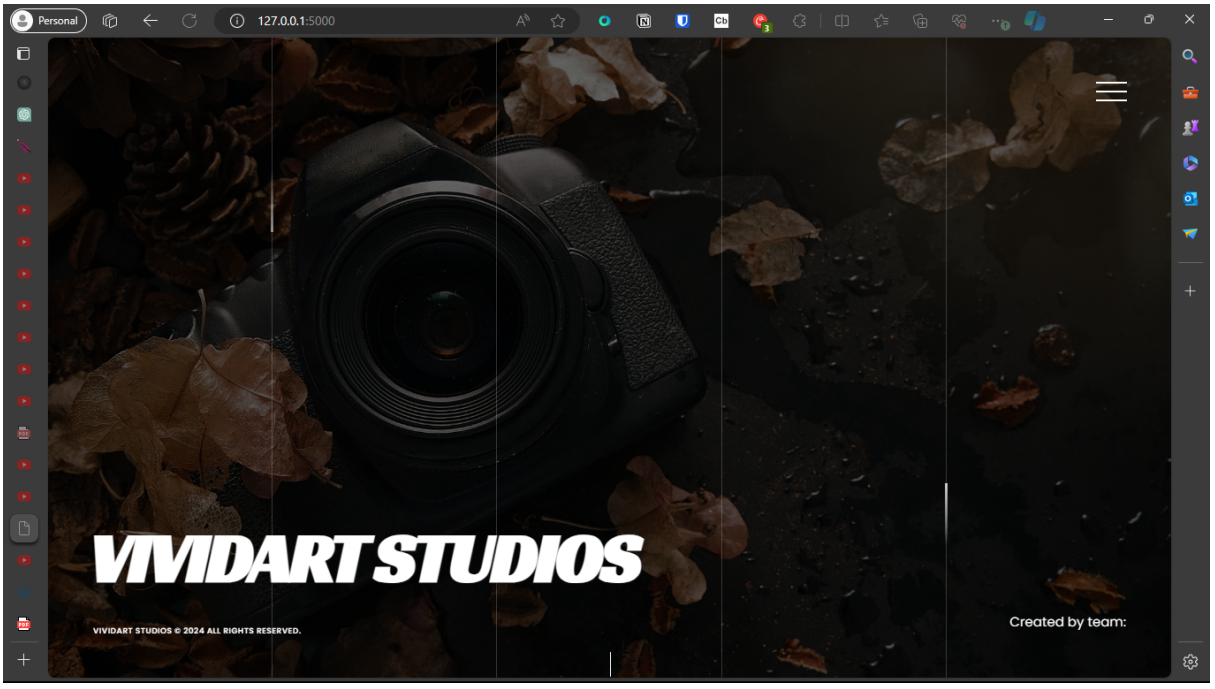
Then, run “terraform init” and “terraform apply -auto-approve” after you ensure that you shell prompt is configured with AWS credentials.

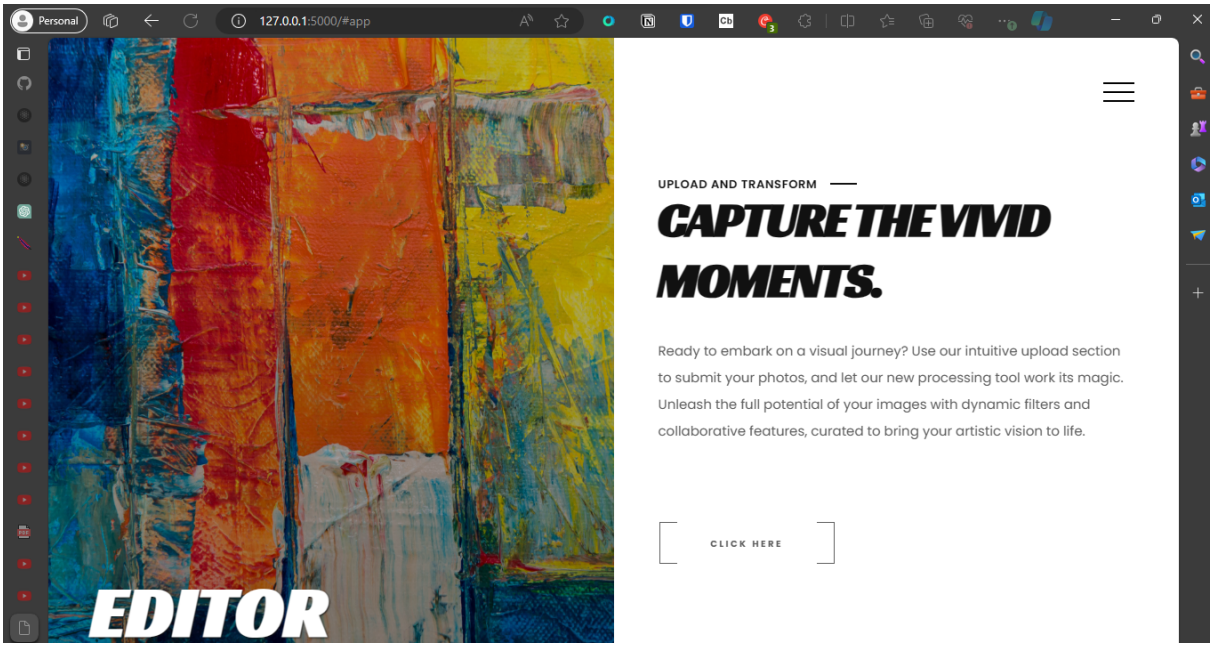
```
    profile                <not set>             None    None
access_key     ****************S5W0 shared-credentials-file
secret_key     ****************jYZ/ shared-credentials-file
    region                 us-east-1           config-file    ~/.aws/config
PS C:\Users\Olatunji Olayinka\Documents\FormerHOD\Programming\Node-React-NPM&More\IntroToNode&Express\AzubiParole\Azubi2023Work-CloudClass\AzubiProjects\VividArtsStud
io-2ndProject\MineVividarts_studio> terraform apply -auto-approve

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.vividbucket will be created
  + resource "aws_s3_bucket" "vividbucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "vividbucket-beanstalk2"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "Beanstalk bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "Beanstalk bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_s3_bucket.vividbucket: Creating...
aws_s3_bucket.vividbucket: Creation complete after 8s [id=vividbucket-beanstalk2]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

And this is the S3 bucket that was created as a result.



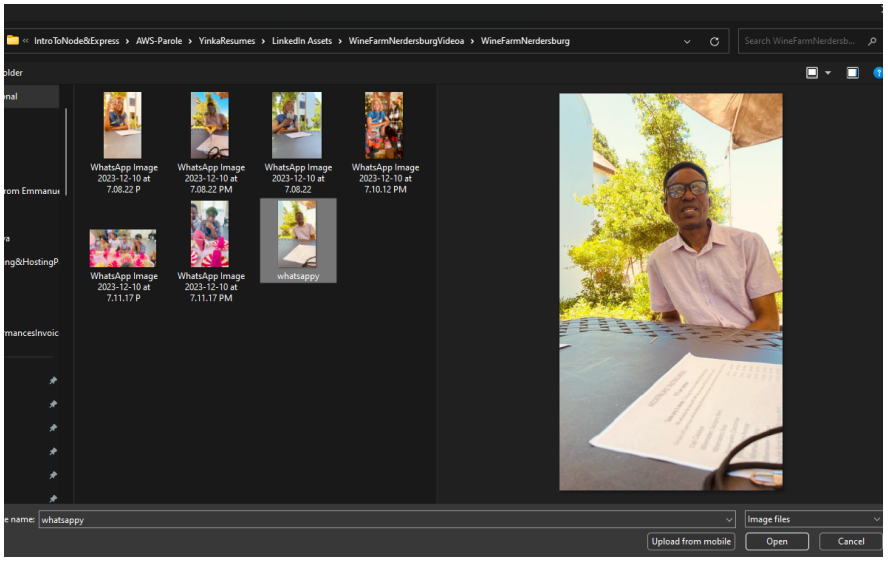
Now, you can access the Containerized Flask application via your web browser by visiting http://localhost:5000

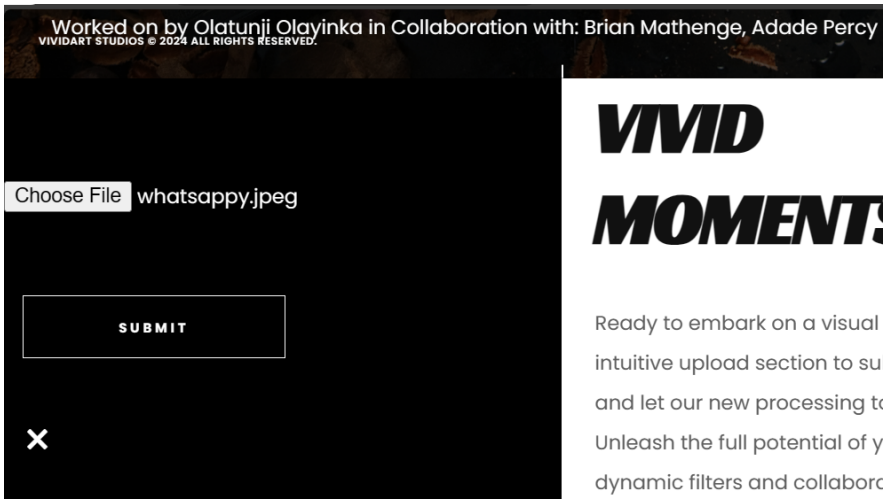




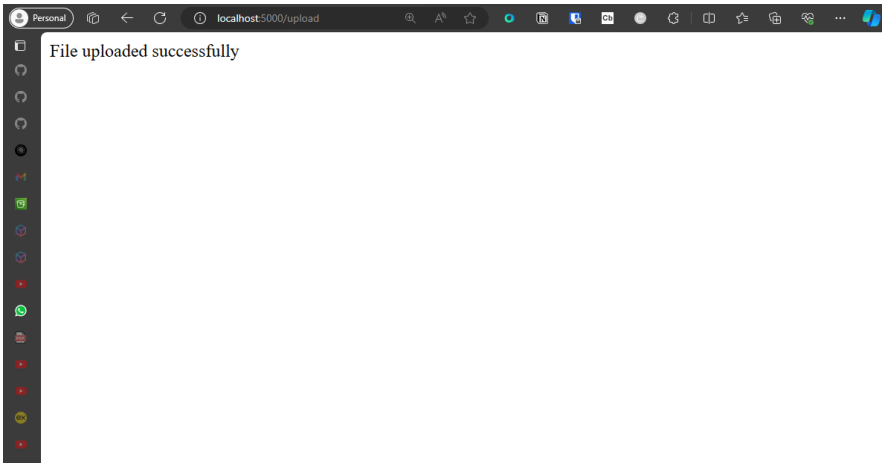
Added link to be able to go to the upload page in the navigation menu:

Selected an image to upload:

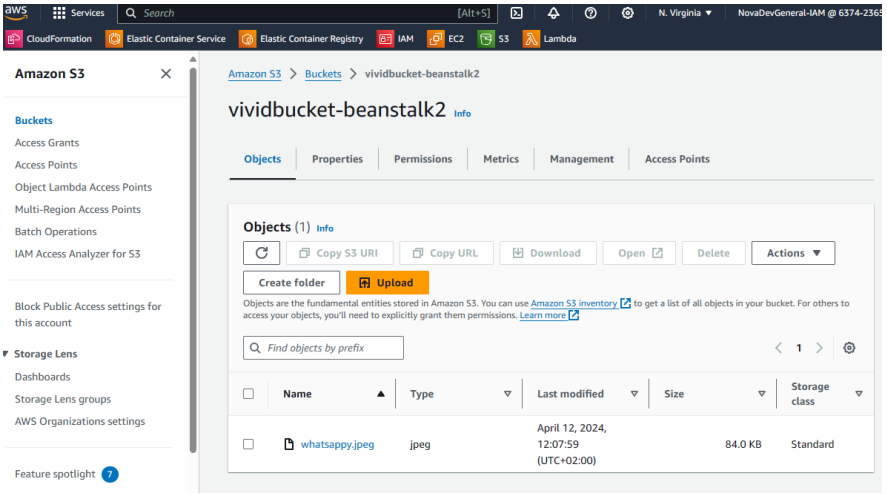




And then Clicked the SUBMIT button which then displayed a message saying that the image is uploaded successfully.

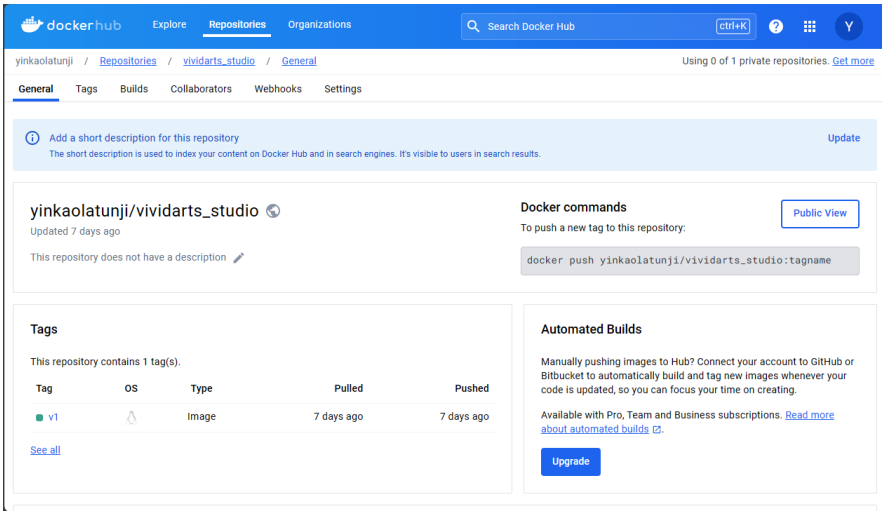


Then I checked to confirm that the S3 bucket named "**vividbucket-beanstalk2**" really contains the uploaded image.



Then I pushed the container image to Docker Hub using this command:

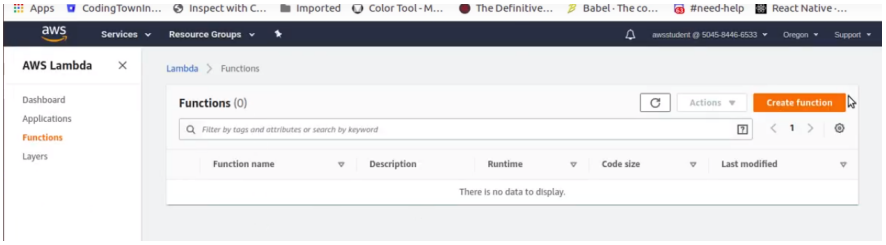
```
>docker push yinkaolatunji/vividarts_studio:v1
```

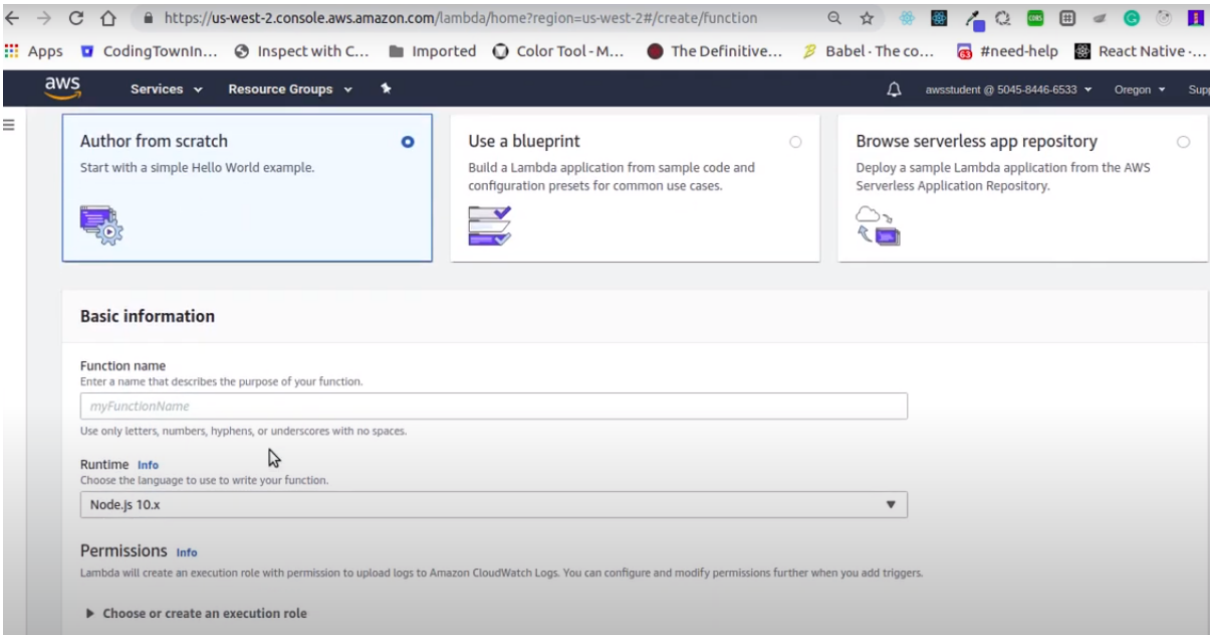


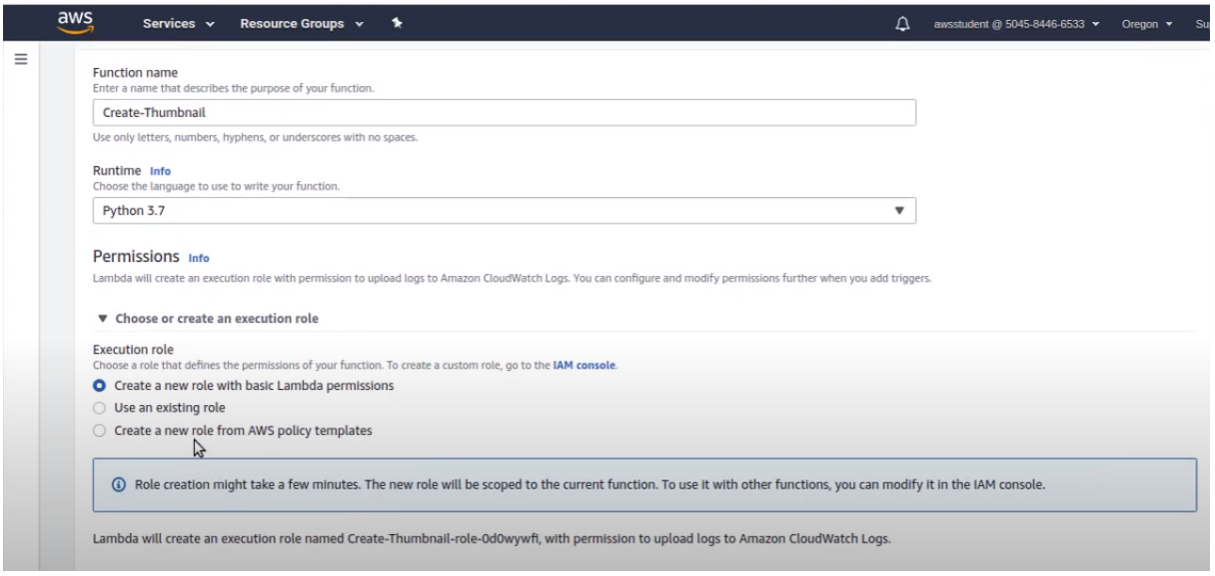
Creating Lambda function to process the uploaded image.

In order to create a Lambda Function to automatically process the image that is uploaded by resizing it using some Python Code:

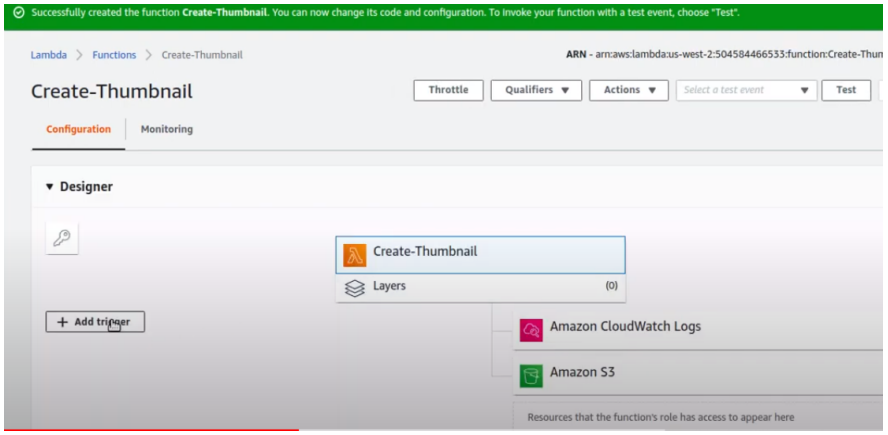
Go to the Lambda service in the AWS management console and click “Create Function”.



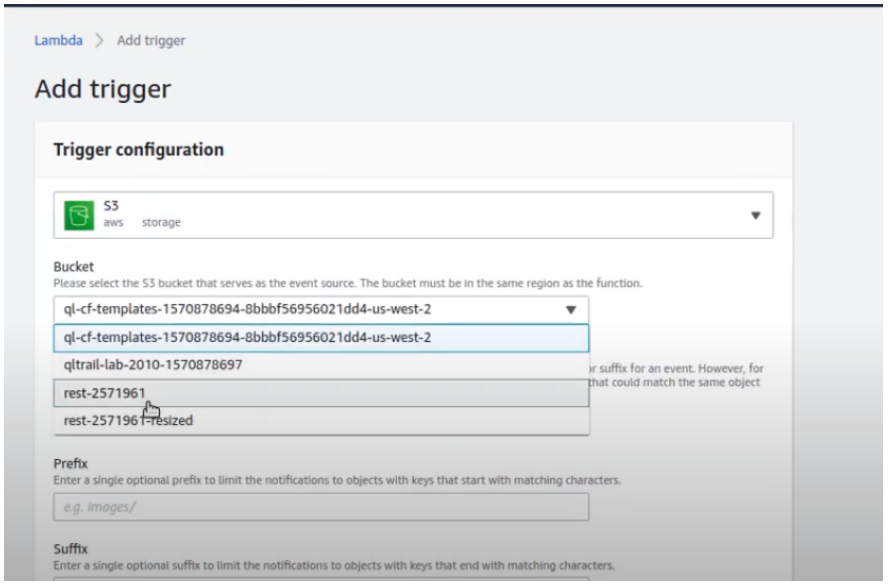




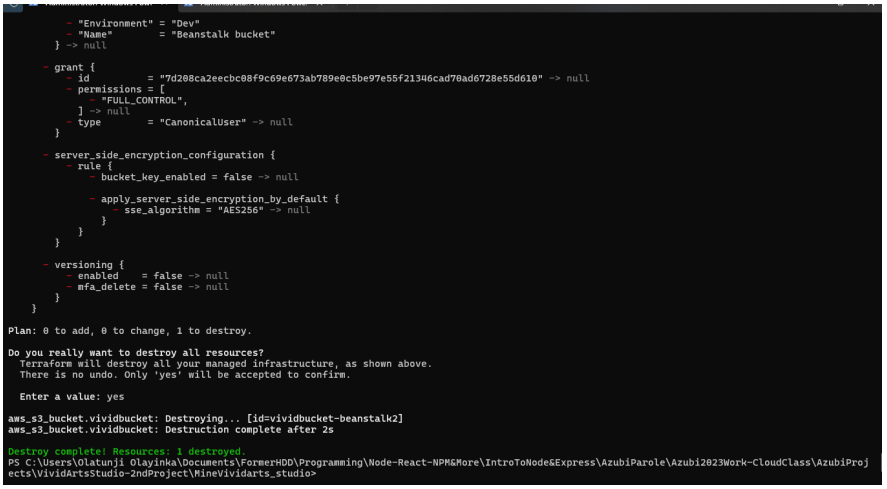
Then I added the Trigger



Selected S3 service and the Bucket name. Then clicked Add.



Then, I ran “terraform destroy” in order to deprovision the S3 bucket that was created using Terraform earlier:


- "Environment" = "Dev"
- "Name" = "Beanstalk bucket"
} -> null
- grant {
- id = "7d208ca2eecbc08f9c69e673ab789e0c5be97e55f21346cad70ad6728e55d610" -> null
- permissions = [
- "FULL_CONTROL",
] -> null
- type = "CanonicalUser" -> null
}
- server_side_encryption_configuration {
- rule {
- bucket_key_enabled = false -> null
- apply_server_side_encryption_by_default {
- sse_algorithm = "AES256" -> null
}
}
}
- versioning {
- enabled = false -> null
- mfa_delete = false -> null
}
}
Plan: 0 to add, 0 to change, 1 to destroy.
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
aws_s3_bucket.vividbucket: Destroying... [id=vividbucket-beanstalk2]
aws_s3_bucket.vividbucket: Destruction complete after 2s
Destroy complete! Resources: 1 destroyed.
PS C:\Users\Olatunji Olayinka\Documents\FormerHDD\Programming\Node-React-NPM&More\IntroToNode&Express\AzubiParole\Azubi2023Work-CloudClass\AzubiProjects\VividArtsStudio-2ndProject\MineVividarts_studio>