CREATING A WEBSITE USING FLASK+DYNAMODB+DOCKER

CLASS:
INSTRUCTOR:

Introduction:

In this tutorial you will be learning how to create a simple website using flask web framework and integrate aws dynamodb database to it using python programming language.

- Flask

  Flask is a micro web framework which is written in python which will help us to create web applications. At the most basic level it provides a development server and debugger.Flask depends on jinja template engine for template creation.

- DynamoDB

  Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data export tools.

- Docker

  We use docker container to package our website and deploy it in aws environment

Prerequisites:

- Linux CLI (bash)
- Docker
- Python3 programming
- pip
- Flask
- AWS and IAM
- AWS elastic beanstalk cli

If you do not have docker installed in your machine please follow a guide to how to install docker on your machine. This is a guide on how you could install docker on a ubuntu machine.

You need python3 installed in your local machine if it is already there you need to download it from here and install it.

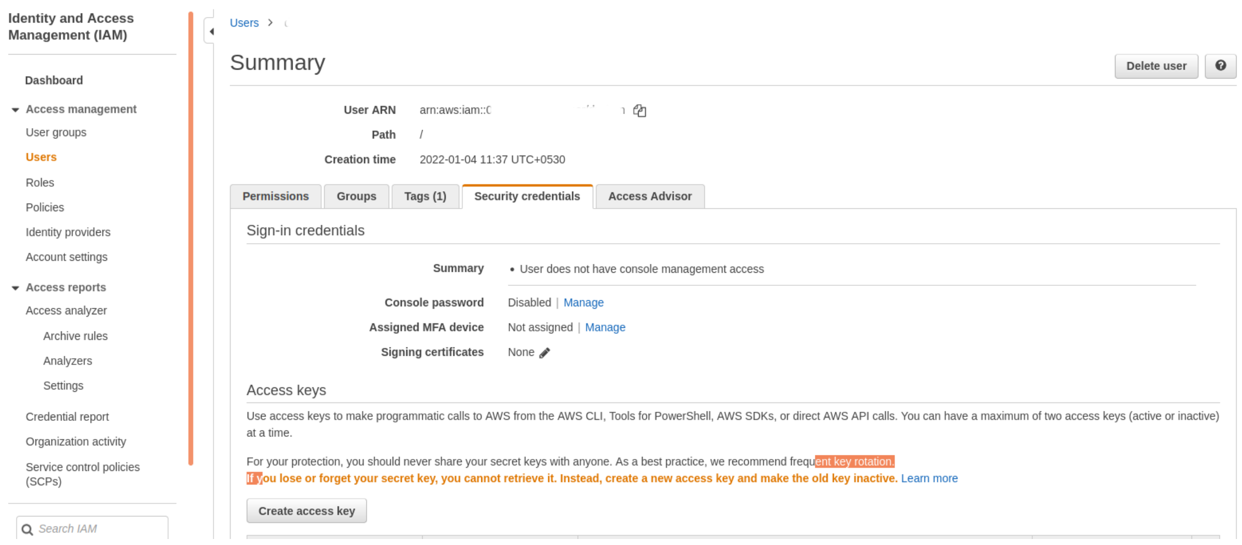
Pip will be available once you have installed python. After that you will need to install aws cli if you have not already configured the aws cli in your local machine.In a terminal try to paste below and install aws cli.

pip install awscli --upgrade --user

After that you need to configure the aws then past below command and provide correct values which are related to your aws account

aws configure

You will get below to add the data. For your user which is used to login to aws console must have an access key and secret. You have to remember the secret value you have entered when you are creating the access key. Please login to aws console and search for IAM and open IAM console navigate to your user and you will find the plase to create a key if you have not already done it.Please follow this [guide](#) to get more details.



Summary

User ARN arn:aws:iam::0

Path /

Creation time 2022-01-04 11:37 UTC+0530

Sign-in credentials

Summary • User does not have console management access

Console password Disabled | Manage

Assigned MFA device Not assigned | Manage

Signing certificates None

Access keys

Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time.

For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation.

If you lose or forget your secret key, you cannot retrieve it. Instead, create a new access key and make the old key inactive. Learn more

AWS Access Key ID [None]: ***PASTE ACCESS KEY ID HERE***

AWS Secret Access Key [None]: ***PASTE SECRET ACCESS KEY HERE***

Default region name [None]: ***TYPE YOUR PREFERRED REGION***

Default output format [None]: json

Then we need to install elastic beanstalk.First clone the files from git using below command

git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git

Then install eb using this command

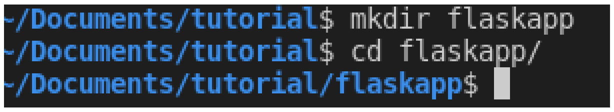
python3.7 ./aws-elastic-beanstalk-cli-setup/scripts/ebcli_installer.py

After installation output will instruct you to export environment variable do as it mentioned.Command will look similar to this

echo 'export PATH="/home/xxxxx/.ebcli-virtual-env/executables:$PATH"' >> ~/.bash_profile && source ~/.bash_profile

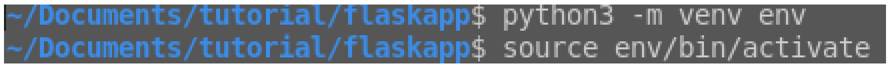
Project Setup:

Ok well done. Now we are ready to go with our project setup. First let's create a directory to hold our project data.open up a terminal and create a directory and navigate to it.



```
~/Documents/tutorial$ mkdir flaskapp
~/Documents/tutorial$ cd flaskapp/
~/Documents/tutorial/flaskapp$ 
```

Now let's create a python virtual environment to install dependencies which we require to develop our app. In same terminal just paste below commands



```
~/Documents/tutorial/flaskapp$ python3 -m venv env
~/Documents/tutorial/flaskapp$ source env/bin/activate
```

Now we could use pip command to install flask and boto3 libs for our project run below command in your terminal where you have activated the virtual environment.This is required for local testing later will add these dependencies in a requirements file for docker image creation.

pip install flask

pip install boto3

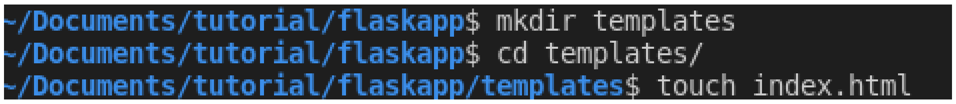
Ok, Then we need to instruct flask which file needs to be used as the main file so let's create the environment variable.Please add the code below in the terminal.

export FLASK_APP=application.py

Next create a directory named templates to keep our html files.

mkdir templates

Create a index.html file inside the templates directory.



```
~/Documents/tutorial/flaskapp$ mkdir templates
~/Documents/tutorial/flaskapp$ cd templates/
~/Documents/tutorial/flaskapp/templates$ touch index.html
```

Add below code in the index.html file.

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>FlaskDynamodb</title>

</head>

<body>

  <h1>Welcome to FlaskSite</h1>

</body>

</html>
```

Now let’s go to the root directory and create an application.py file.

CMD: touch application.py

Try to add this code in application.py file

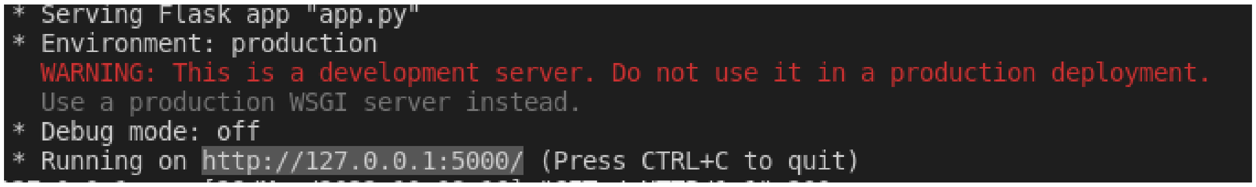
```
from flask import Flask, render_template
```

```
application = Flask(__name__)


@application.route('/')

def index():

   return render_template('index.html')
```

@app.route('/') indicate that if the server get an request for root ('/') render the index.html file.

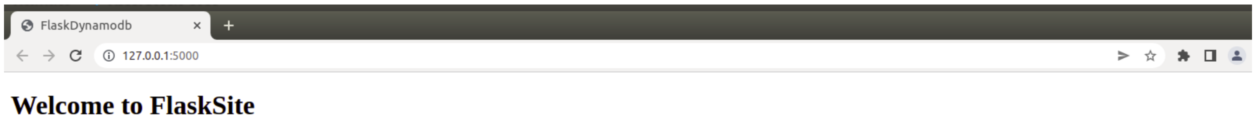
Ok now we can run our very first flask web app. In terminal run:

flask run



```
 * Serving Flask app "app.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now navigate to the link http://127.0.0.1:5000/ and you will see a web page as below.



Welcome to FlaskSite

Database creation:

Create a json file named db-table.json and add this content in it.

```
{

   "TableName": "FilmInfoTable",

   "KeySchema": [

       { "AttributeName": "Director", "KeyType": "HASH" },
       { "AttributeName": "FilmTitle", "KeyType": "RANGE" }

   ],

   "AttributeDefinitions": [

       { "AttributeName": "Director", "AttributeType": "S" },

       { "AttributeName": "FilmTitle", "AttributeType": "S" }

   ],

   "ProvisionedThroughput": {

       "ReadCapacityUnits": 5,

       "WriteCapacityUnits": 5

   }

}
```

Here we are creating a dynamodb table named FilmInfoTable and it's partition key is Director and the sort key is the FilmTitle

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

Sort key

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

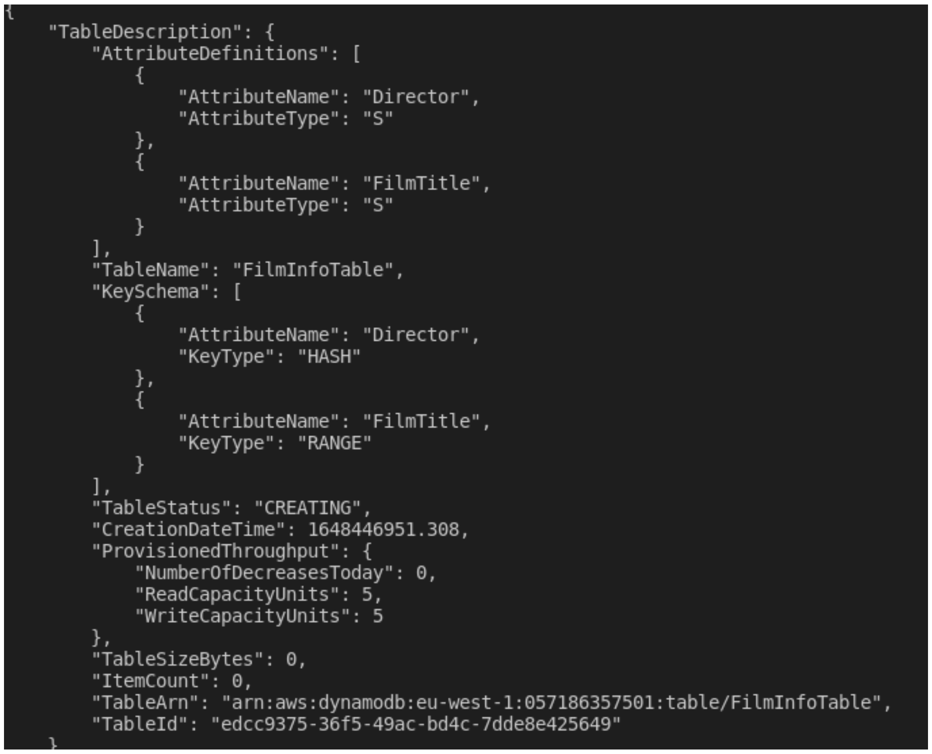
More about the provisioned information can be found here.

Let's create the table and add the data to the table now.

Run the below command in the terminal where your json file exists.

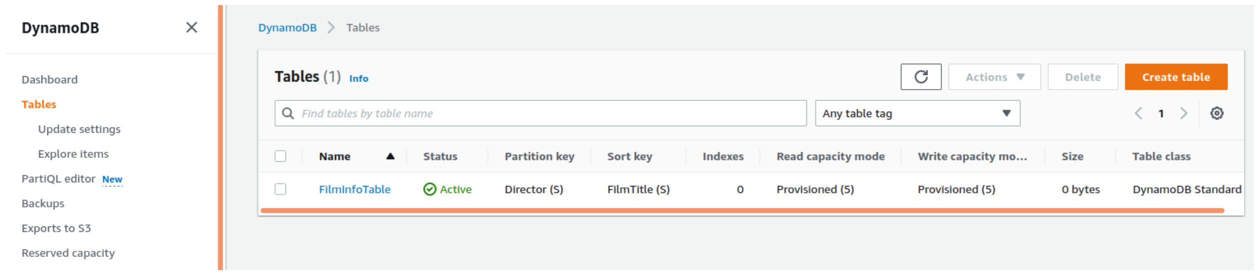
aws dynamodb create-table --cli-input-json file://db-table.json

You will see a output like this:



```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Director",
                "AttributeType": "S"
            },
            {
                "AttributeName": "FilmTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "FilmInfoTable",
        "KeySchema": [
            {
                "AttributeName": "Director",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "FilmTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1648446951.308,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:eu-west-1:057186357501:table/FilmInfoTable",
        "TableId": "edcc9375-36f5-49ac-bd4c-7dde8e425649"
    }
```

You can navigate to aws console and search for the dynamodb and go dynamodb and see the table which you created.



	Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mo...	Size	Table class
	FilmInfoTable	Active	Director (S)	FilmTitle (S)	0	Provisioned (5)	Provisioned (5)	0 bytes	DynamoDB Standard

Let's add some data to this table now. Create a json file named test-data.json and add the below content in it.

```
{
  "FilmInfoTable": [{
         "PutRequest": {
                "Item": {
                    "Director": {
                        "S": "ADAM"
                    },
                    "FilmTitle": {
                        "S": "Sleep less"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Director": {
                        "S": "Stefan"
                    },
                    "FilmTitle": {
                        "S": "Paint ball"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Director": {
                        "S": "Lisa"
                    },
                    "FilmTitle": {
                        "S": "Fedup"
                    }
                }
            }
        }
    ]
}
```

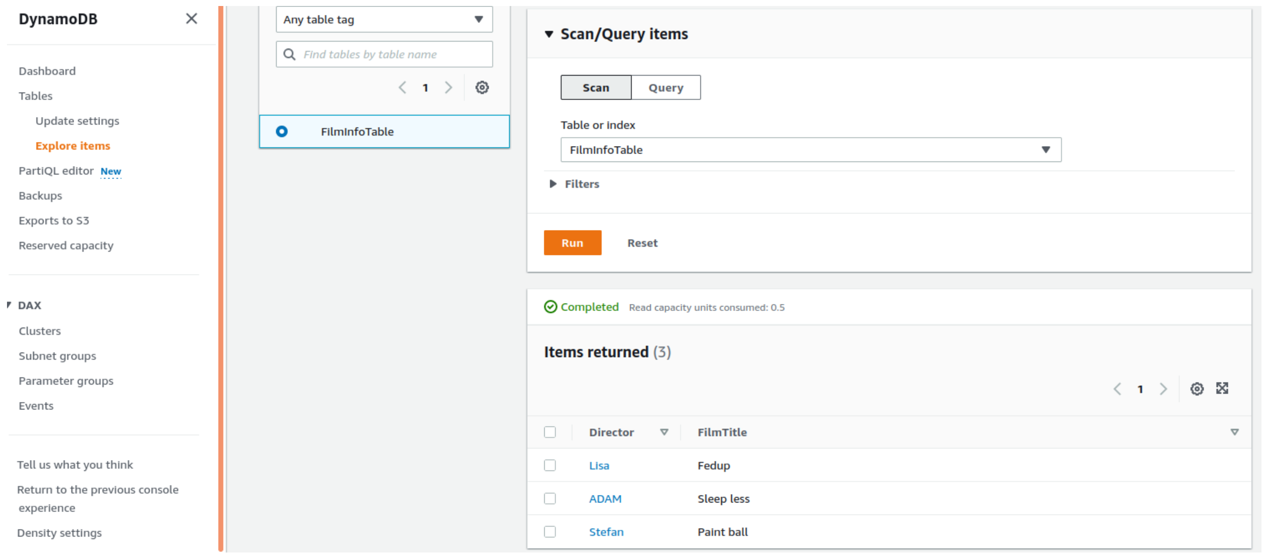
Now execute below command.

```
aws dynamodb batch-write-item --request-items file://test-data.json
```

You should see a result like this

```
{"UnprocessedItems": {}}
```

If you explore the items in the newly created table you will see the inserted data.



Director	FilmTitle
Lisa	Fedup
ADAM	Sleep less
Stefan	Paint ball

Let’s read these data from our flask code and visualize it in our web template.Create a file named aws-controller.py and add this code

```
import boto3


dynamo_client = boto3.client('dynamodb')


def get_items():

    response=dynamo_client.scan(

        TableName='FilmInfoTable'

    )

    return response['Items']
```

In this code we are using boto3 to connect our dynamodb database and fetch the table data.

After this we need to pass the fetched values to web page via app.py. So you need to change the app.py like this

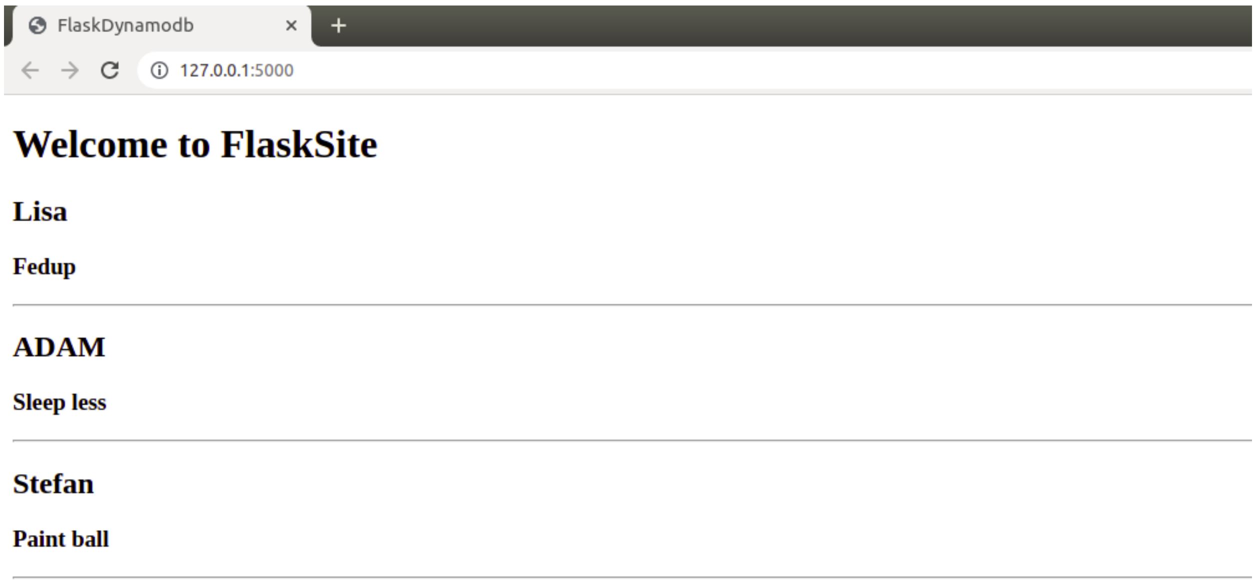
```
from flask import Flask, render_template

import aws_controller



application = Flask(__name__)



@application.route('/')

def index():

   data=aws_controller.get_items()

   return render_template('index.html',films=data)
```

So here we are passing fetched film data to our template render_template('index.html',films=data) and now we need to read those data via our index.html file.

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>FlaskDynamodb</title>

</head>

<body>
```

```
  <h1>Welcome to FlaskSite</h1>

  {% for film in films %}

       <h2>{{ film['Director']['S'] }}</h2>

       <h3>{{ film['FilmTitle']['S'] }}</h3>

       <hr>

   {% endfor %}

</body>

</html>
```

We use jinja template capabilities to loop “{% for film in films %}” through the film data we passed and visualize those. Run again the application and see the results.



FlaskDynamodb

127.0.0.1:5000

Welcome to FlaskSite

Lisa

Fedup

ADAM

Sleep less

Stefan

Paint ball

Ok if you see the above result you have done a great job. Now let’s create a docker container out of this code and deploy it in AWS using aws Elastic Beanstalk.

Docker Container Creation:

First let’s create a requirements.txt file and add flask and boto3 in it.

```
flask
boto3
```

Create dockerfile in the root directory and add below content

```
# syntax=docker/dockerfile:1

FROM python:3.8-slim-buster

WORKDIR /python-docker

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
ENV FLASK_APP=application.py

COPY . .
EXPOSE 80
EXPOSE 8000
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0", "--port=8000"]
```

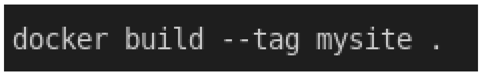
We use python:3.8-slim-buster as the base image of our solution.Then we copy the requirement.txt file install requirements. We are copying the rest of the files using COPY .. and expose the port 8000 in the container.Then change the aws_controller.py code to this.
Make sure you set AWS_SERVER_PUBLIC_KEY and AWS_SERVER_SECRET_KEY values to your values.It is not recommended to hard code these values but for learning purposes we hardcoded those.

```
from urllib import response
import boto3
import os
AWS_SERVER_PUBLIC_KEY="xxxxxxxxxxxxx"
AWS_SERVER_SECRET_KEY="xxxxxxxxxxxxxxxxxxxxxxxxxxx"

dynamo_client = boto3.client('dynamodb',
                           aws_access_key_id=AWS_SERVER_PUBLIC_KEY,
                           aws_secret_access_key=AWS_SERVER_SECRET_KEY,
                           region_name='eu-west-1')
```

```
def get_items():
   response=dynamo_client.scan(
       TableName='FilmInfoTable'
   )
   return response['Items']
```

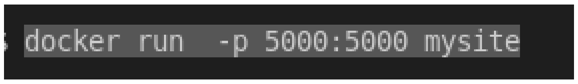
Here we are passing the credentials to access aws services via environment variables which we pass through the docker file.

Let's build our first docker file and run it locally first.
Open a terminal in the root folder and run Docker build command. Here we are tagging our image as mysite



```
docker build --tag mysite .
```

After you successfully build the image you could run the image as below.Here we are mapping port 5000 to exposed port 5000.



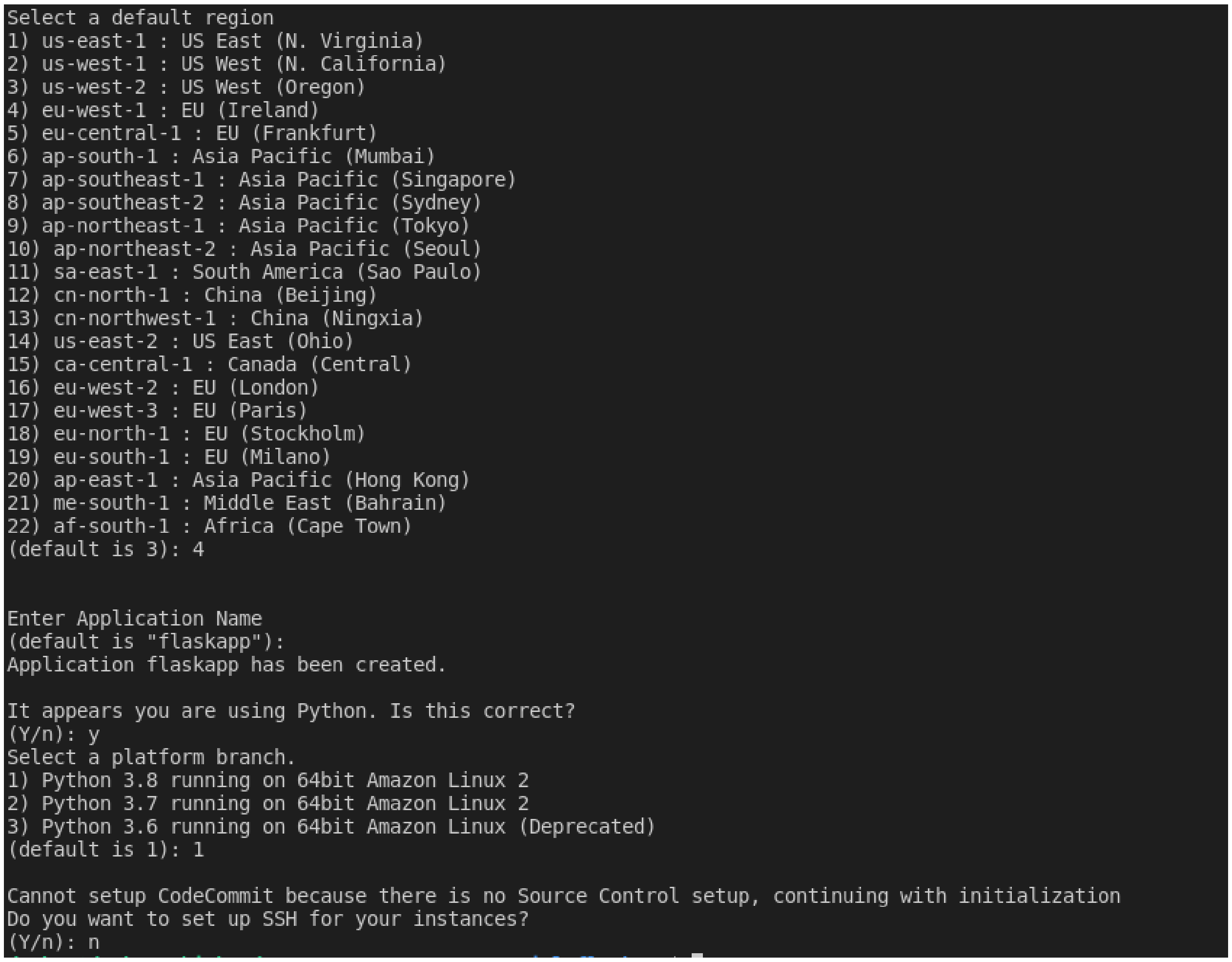
```
docker run  -p 5000:5000 mysite
```

Deploy using elastic beanstalk:

Open a new terminal and navigate to your root directory and run eb init.

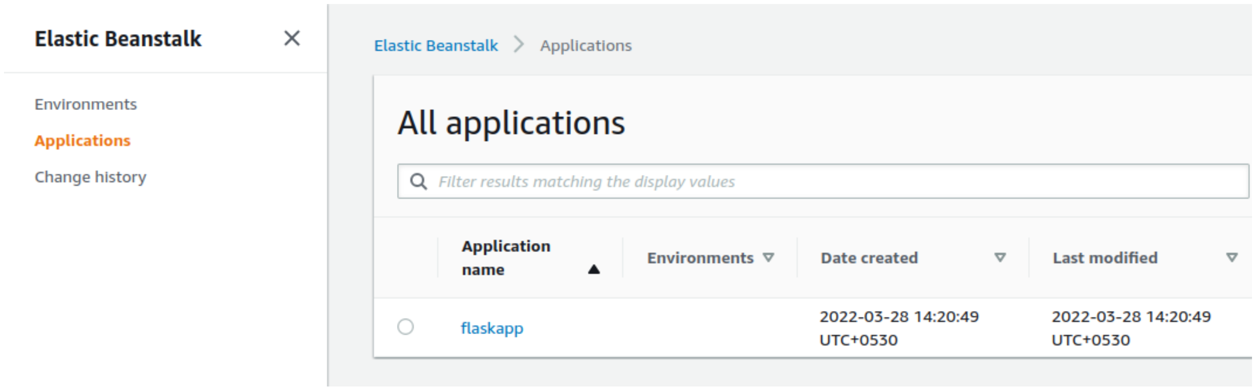
eb init

It will ask to select a region, add the number of your region. In my case it is 4 because I'm using eu-west-1.It will ask for the app name, keep it as it is and select the python env as instruct finally type n for ssh access because we do not need to access the ec2 instance we create.See below image what I have entered.



```
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
8) ap-southeast-2 : Asia Pacific (Sydney)
9) ap-northeast-1 : Asia Pacific (Tokyo)
10) ap-northeast-2 : Asia Pacific (Seoul)
11) sa-east-1 : South America (Sao Paulo)
12) cn-north-1 : China (Beijing)
13) cn-northwest-1 : China (Ningxia)
14) us-east-2 : US East (Ohio)
15) ca-central-1 : Canada (Central)
16) eu-west-2 : EU (London)
17) eu-west-3 : EU (Paris)
18) eu-north-1 : EU (Stockholm)
19) eu-south-1 : EU (Milano)
20) ap-east-1 : Asia Pacific (Hong Kong)
21) me-south-1 : Middle East (Bahrain)
22) af-south-1 : Africa (Cape Town)
(default is 3): 4


Enter Application Name
(default is "flaskapp"):
Application flaskapp has been created.

It appears you are using Python. Is this correct?
(Y/n): y
Select a platform branch.
1) Python 3.8 running on 64bit Amazon Linux 2
2) Python 3.7 running on 64bit Amazon Linux 2
3) Python 3.6 running on 64bit Amazon Linux (Deprecated)
(default is 1): 1

Cannot setup CodeCommit because there is no Source Control setup, continuing with initialization
Do you want to set up SSH for your instances?
(Y/n): n
```

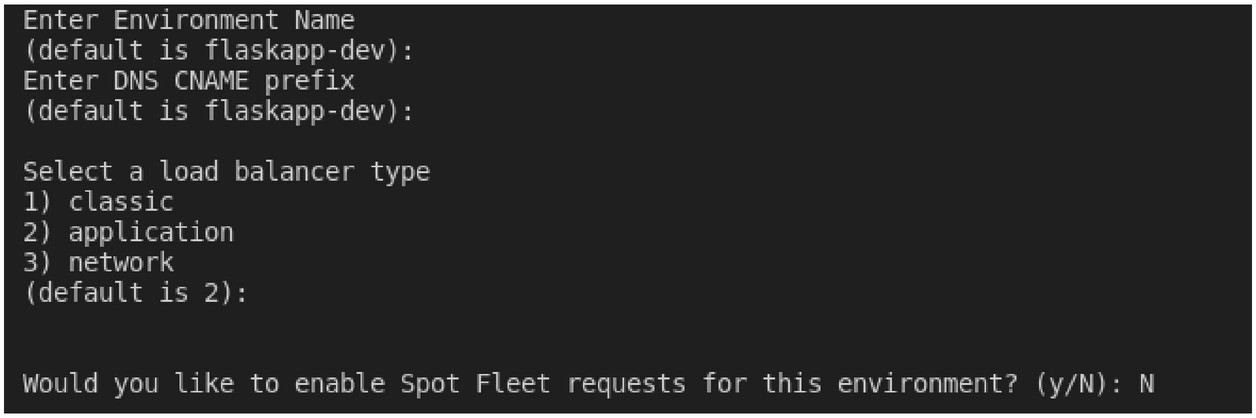
Now if you got elastic beanstalk in aws console you will see the application you created.


Elastic Beanstalk
Environments
Applications
Change history
Elastic Beanstalk > Applications
All applications
Filter results matching the display values

Application name	Environments	Date created	Last modified
flaskapp		2022-03-28 14:20:49 UTC+0530	2022-03-28 14:20:49 UTC+0530

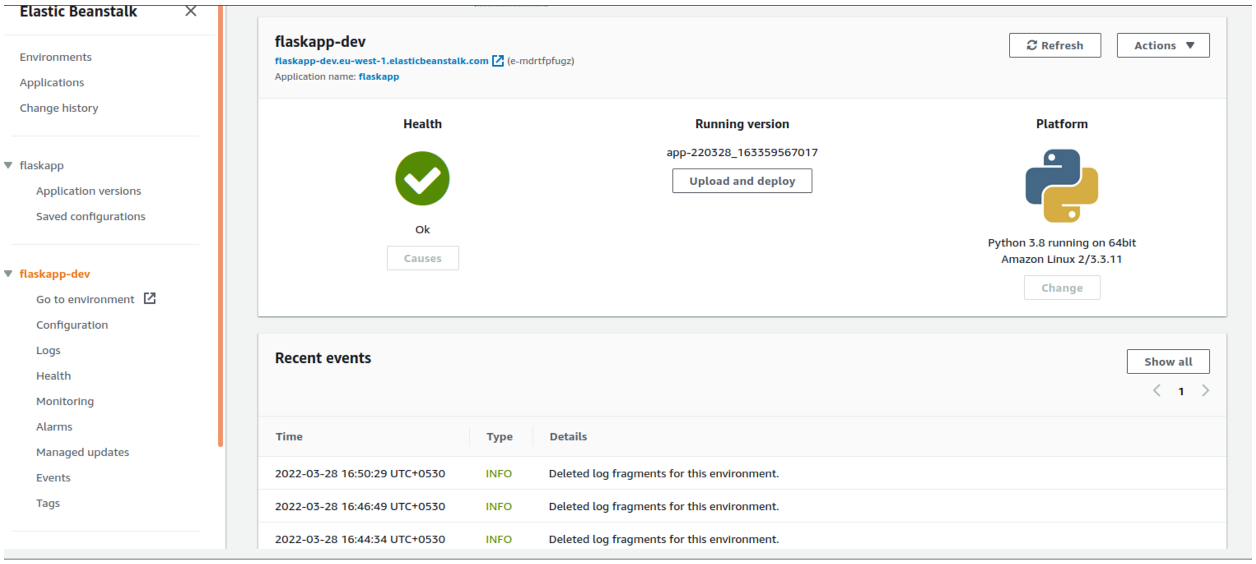
Now lets create an environment. In terminal type eb create.
eb create

I kept most of the configurations to default as in the image just set spot fleet request to N.And it will create a set of aws resources such as security groups, load balancers, ec2 instances,s3 bucket for keeping the code .ect for you. It will take a few minutes to create all the resources.



```
Enter Environment Name
(default is flaskapp-dev):
Enter DNS CNAME prefix
(default is flaskapp-dev):

Select a load balancer type
1) classic
2) application
3) network
(default is 2):


Would you like to enable Spot Fleet requests for this environment? (y/N): N
```

Once you get all setup go to aws console and navigate to aws beanstalk and see the application status.



flaskapp-dev
flaskapp-dev.eu-west-1.elasticbeanstalk.com (e-mdrtfpfugz)
Application name: flaskapp

Health: Ok

Running version: app-220328_163359567017

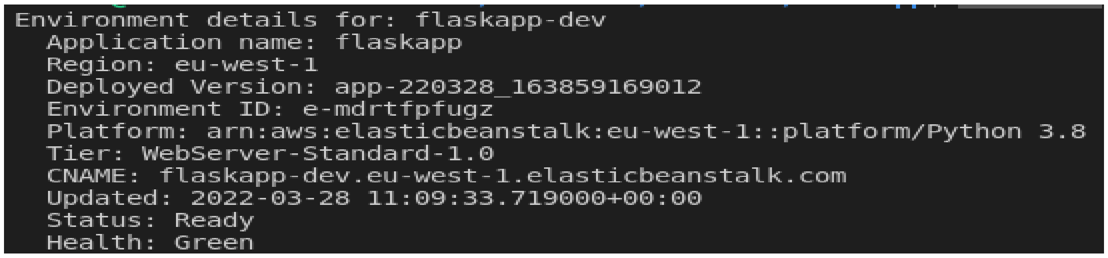
Platform: Python 3.8 running on 64bit Amazon Linux 2/3.3.11

Recent events

Time	Type	Details
2022-03-28 16:50:29 UTC+0530	INFO	Deleted log fragments for this environment.
2022-03-28 16:46:49 UTC+0530	INFO	Deleted log fragments for this environment.
2022-03-28 16:44:34 UTC+0530	INFO	Deleted log fragments for this environment.

And also you can see the status via command eb status in terminal

eb status

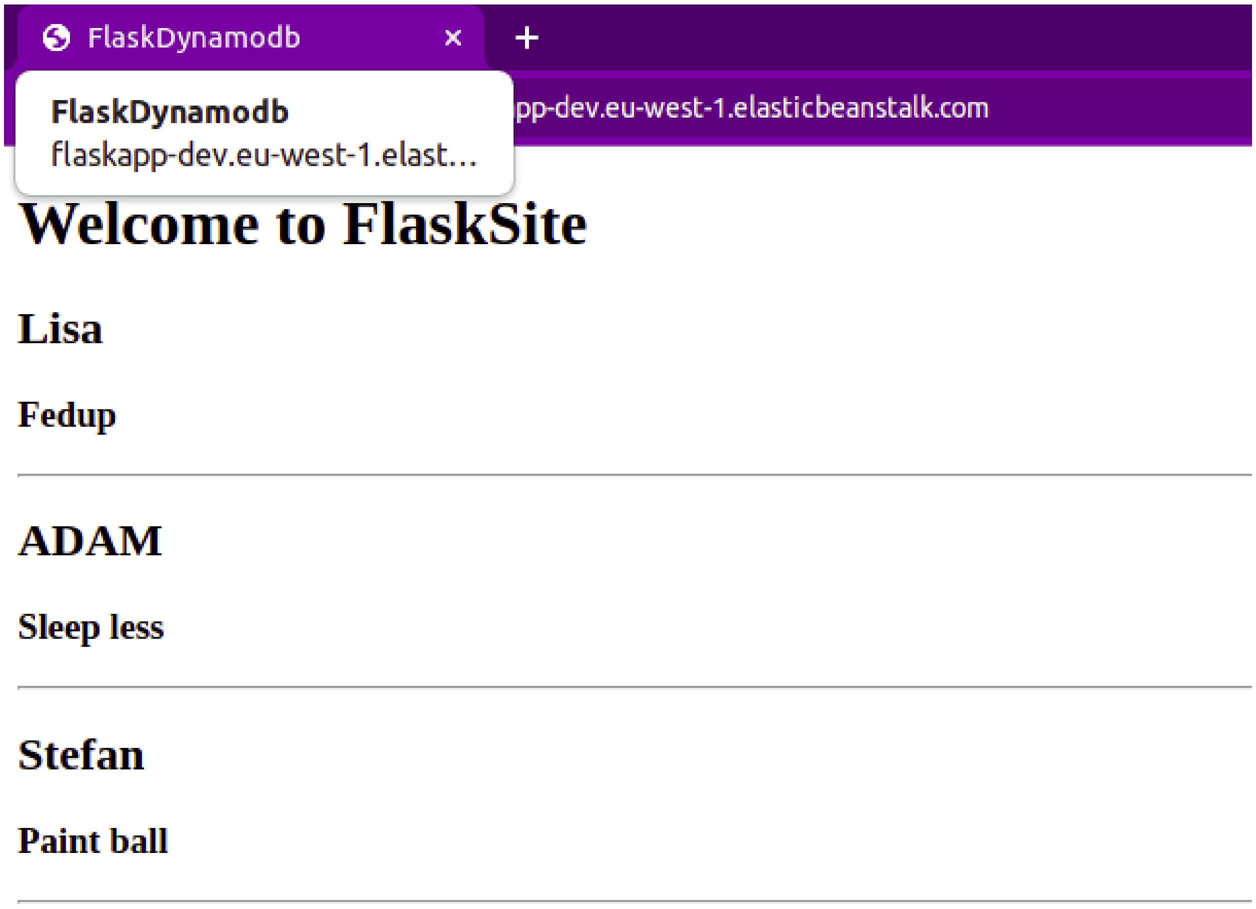


```
Environment details for: flaskapp-dev
  Application name: flaskapp
  Region: eu-west-1
  Deployed Version: app-220328_163859169012
  Environment ID: e-mdrtfpfugz
  Platform: arn:aws:elasticbeanstalk:eu-west-1::platform/Python 3.8
  Tier: WebServer-Standard-1.0
  CNAME: flaskapp-dev.eu-west-1.elasticbeanstalk.com
  Updated: 2022-03-28 11:09:33.719000+00:00
  Status: Ready
  Health: Green
```

Now you can open the web page by typing eb open

eb open

It will open a web browser with output like this.


FlaskDynamodb
FlaskDynamodb
flaskapp-dev.eu-west-1.elast...
pp-dev.eu-west-1.elasticbeanstalk.com

Welcome to FlaskSite

Lisa

Fedup

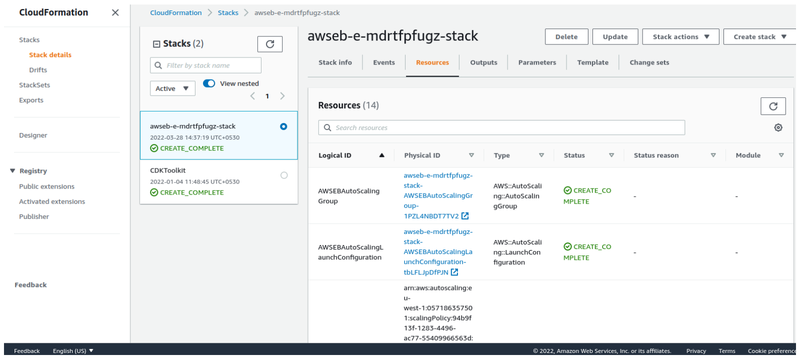
ADAM

Sleep less

Stefan

Paint ball

If you navigate to cloudformation in aws console you will see all the resources created by the beanstalk.



awseb-e-mdrtfpfugz-stack
2022-03-28 14:37:19 UTC+0530
CREATE_COMPLETE

CDKToolkit
2022-01-04 11:48:45 UTC+0530
CREATE_COMPLETE

awseb-e-mdrtfpfugz-stack

Resources (14)

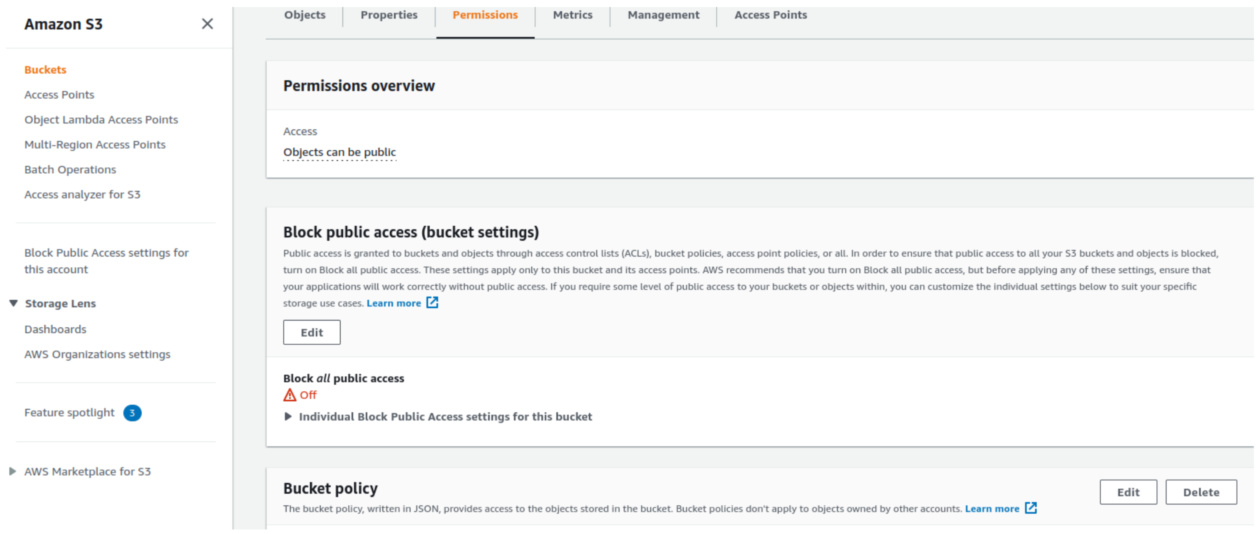
Logical ID	Physical ID	Type	Status	Status reason	Module
AWSEBAutoScalingGroup	awseb-e-mdrtfpfugz-stack-AWSEBAutoScalingGroup-1PZL4NBDT7TV2	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE	-	-
AWSEBAutoScalingLaunchConfiguration	awseb-e-mdrtfpfugz-stack-AWSEBAutoScalingLaunchConfiguration-tbLFLJpDfPJN	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE	-	-
	arn:aws:autoscaling:eu-west-1:057186357501:scalingPolicy:94b9f13f-1283-4496-ac77-55409966563d:				

Remove Deployment:

If you need to terminate the deployed service you can run below command

```
eb terminate –force
```

This will remove all the resources except s3 bucket.You have to remove the attached bucket policy and delete the bucket from aws console.
Navigate to s3 and there is a tab named permission in there you have the option to delete the bucket policy.



Permissions overview

Access
Objects can be public

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access
⚠ Off

▶ Individual Block Public Access settings for this bucket

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

After that just delete the bucket.