National College of Ireland

Configuration Manual For AWS Instance Management

MSc Research Project
Cloud Computing

Sarath Ravichandran
Student ID: x18174311

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sarath Ravichandran
Student ID:	x18174311
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual For AWS Instance Management
Word Count:	2209
Page Count:	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	16th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual For AWS Instance Management

Sarath Ravichandran
x18174311

Abstract

AWS Instance Management is a web application that is created to achieve business continuity and cost optimization in the AWS spot instance. This report would provide the configurations of each technology that is used while implementing the web application.

1 Introduction

The main objective of this report is to provide the configurations of each technology that is used while creating the web application.

2 Configuration Specifications

This section of the document would provide the configuration specification of each module that is present in the AWS instance management application. Since this web application contains two major modules in it. This section is further categorized into two sub-sections namely Configuration of Forecast application, and Configurations of AWS instance management application

2.1 Configuration and Implementation of Forecast Module

Various technologies were used in order to predict the spot price of the instance in advance. Those technology specifications are mentioned below.

Table 1: Overview of Tools and Technologies

Tools and Technologies		
Technologies	Version	Purpose
Anaconda Navigator	1.9.12	Forecast Module
Jupyter Notebook	6.0.3	Forecast Module
Python	3.7.6	Forecast Module

2.1.1 Anaconda Navigator

Anaconda is python package management tool. For the forecasting module 1.9.12 version of anaconda is used. In which an environment is created for the forecast module as shown below:

From the above diagram, it could be clearly seen that an environment was created with the name of TensorFlow. In that, all the required python libraries are installed for

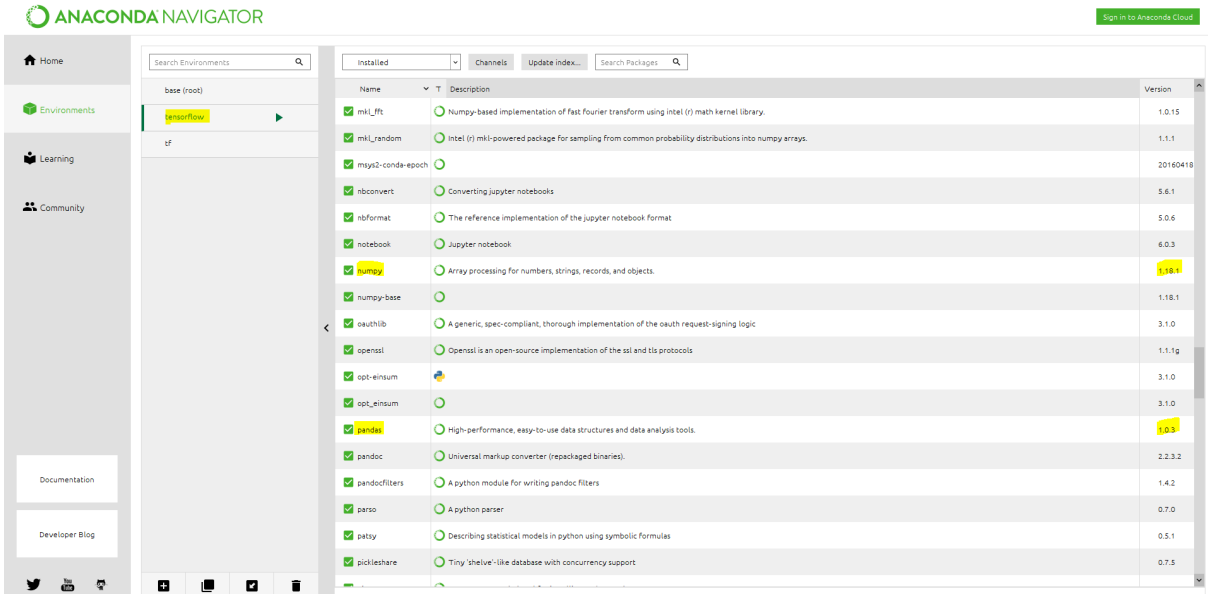

Figure 1: Proposed Life Cycle

the forecast model. This environment would have all the predefined packages with it but the following libraries are added for the forecast model.

Table 2: Python Libraries for Forecast Model

Python Libraries	
Library Name	Version
Tensorflow	2.1.0
Pandas	1.0.3
Keras	2.3.1
Numpy	1.18.1
Scikit-Learn	0.22.1
matplotlib	3.1.3

The above-mentioned libraries are installed in the TensorFlow environment for the forecast model to make the forecast of the spot price in advance. In which pandas and NumPy python libraries are used to clean the dataset. Scikit learn, and Keras are used to make use to create the forecast models. Matplotlib is used to produce graphs of the result. Two models were created with two different machine learning algorithms namely ARIM and LSTM as shown below:

2.1.2 Implementation of Forecast Model Using LSTM

The below figure shows the code of the created LSTM forecast model. All the libraries that are configured in the anaconda are used in this Jupyter Notebook file. All the required libraries are imported for this forecast module namely Keras, Scikit Learn, Matplotlib, and etc. From the code, it could be seen that 70 percent of the dataset is used for training and the remaining 30 percent are used for the testing purpose. The forecast is measured in terms of Mean Absolute Error, and Root Mean Squared Error.

```
df =pd.read_csv('t2-micro-linux-unix.csv')
df.columns = ["1", "2","3","price","5"]

dataset = df.price.values #numpy.ndarray
dataset = dataset.astype('float32')
dataset = np.reshape(dataset, (-1, 1))
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
train_size = int(len(dataset) * 0.70)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 24
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0],  X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

model = Sequential()
model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(X_train, Y_train, epochs=50, batch_size=70, validation_data=(X_test, Y_test),
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=1, shuffle=False)

model.summary()
```

```
pre_lis =[]

df_n =pd.read_csv('t2-micro-linux-unix.csv')
df_n.columns = ["1", "2","3","4","5"]
#Create a new dataframe

predict=[]
new_df = df_n.filter(['4'])
last_60_days = new_df[-24:].values
#Scale the data to be values between 0 and 1
last_60_days_scaled = scaler.transform(last_60_days)
#Create an empty list
X_test = []
#Append teh past 60 days
X_test.append(last_60_days_scaled)
#Convert the X_test data set to a numpy array
X_test = np.array(X_test)
#Reshape the data
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
batch = X_test

for i in range(24):
    pre_lis.append(model.predict(batch)[0])
    batch = np.append(batch[:, 1:,:], [[pre_lis[i]]], axis=1
#undo the scaling
pre_lis = scaler.inverse_transform(pre_lis)
print(pre_lis)
```


Figure 2: Forecast Model Using ARIMA

2.1.3 Implementation of Forecast Model Using ARIMA

```
from pandas import read_csv
import pandas as pd
import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error

t2micro =pd.read_csv('t2-micro-linux-unix-us-east-1a.csv')
t2micro.columns = ["1", "2","3","4","5"]
#Create a new dataframe
predict=[]
new_df = t2micro.filter(['4'])

X = new_df.values
    size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
for t in range(len(test)):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
print(predictions)
```

Figure 3: Forecast Model Using ARIMA

From the above figure, it could be clearly seen that the jupyter notebook is used for coding the forecast model. All the libraries that are configured in the anaconda environments are used here to construct the corresponding model. In this case, the model is created with the ARIMA. In which the dataset is divided into 66% for the training purpose and the remaining percentage is used for the testing. The forecasts are made and it is measured in terms of MSE.

2.2 Configuration and Implementation of Failure Recovery

A web application was created as a part of this research to implement the proposal and make it available for the end-users. The name of the created web application is

AWS Instance Management(AIM). Overview of tools and technologies that are used are mentioned below.

Table 3: Overview of Tools and Technologies

Tools and Technologies		
Technologies	Version	Purpose
PyCharm	11.0.5	Development Tool
Python	Python 3.8.3	Programming Language
Postgres	12.3	DataBase
PgAmin	4.21	DataBaseAdmin
Boto3	1.14.12	Python Library
Django	3.0.7	Python Library
Django-heroku	0.3.1	Python Library
Jquery	3.4.1	Frontend Technology
Bootstrap	4.3.1	Frontend Technology
Ajax	1.14.7	Frontend Technology

2.2.1 PyCharm

PyCharm would provide the python development environment to develop the python program. This tool was used throughout the web application development phase. The following figure would provide more details on the python version and PyCharm.

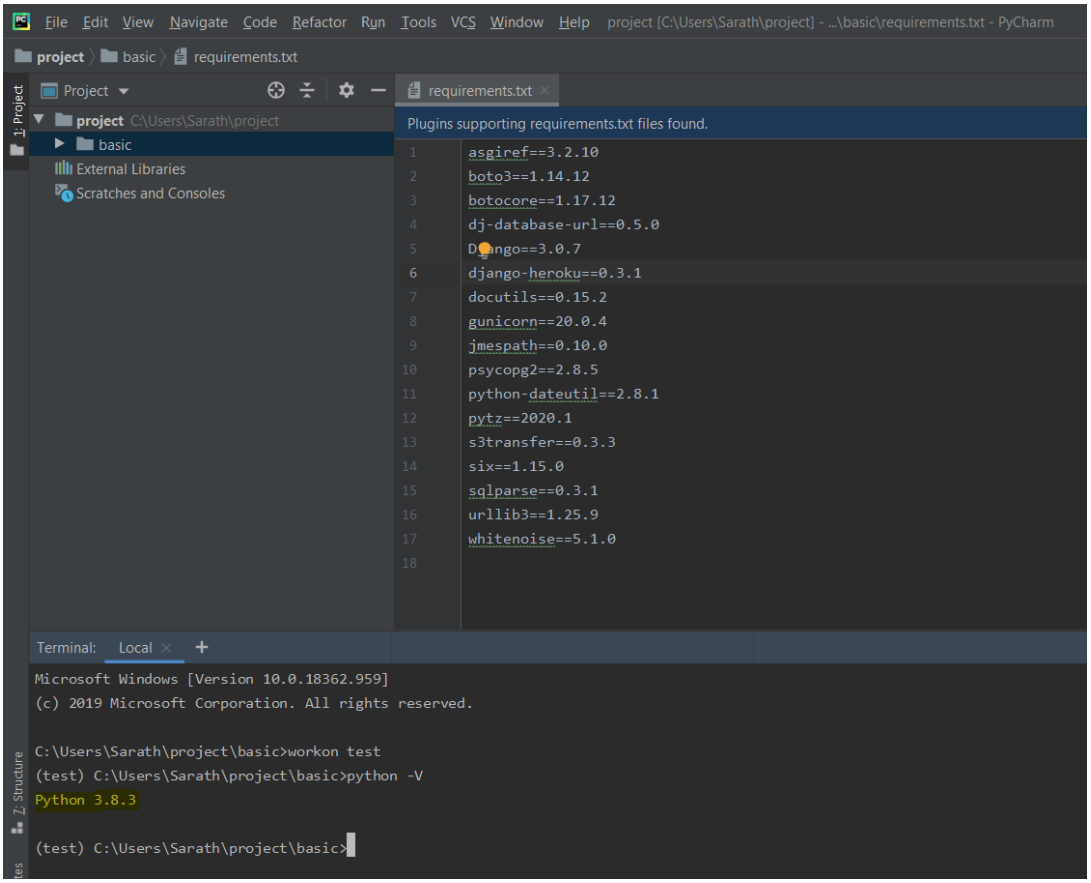

Figure 4: PyCharm and Python Version

A virtual environment was created with the name of the test in pycharm and all the

dependent python libraries are installed in that variable. Python 3.8.3 version is used in this application. From the above figure, we could verify that. It also provides a list of dependent python libraries and their version that is used in this project. By default, the Django framework would have predefined libraries on top of we for this project various libraries like boto3, and Django-Heroku are installed manually. In which boto3 is used to make the communication with the user's AWS account. Django-Heroku library is used to deploy the created web application in the cloud platform.

2.2.2 Postgres

Postgresql is used as the database to store the content of our application. This database is used to store the data of the registered user, forecast-ed spot price, newly created instance, and AMI creation. These details could be accessed from the admin functionality of the proposed web application. PgAdmin is a database administration tool that is used to verify the entries in the database.

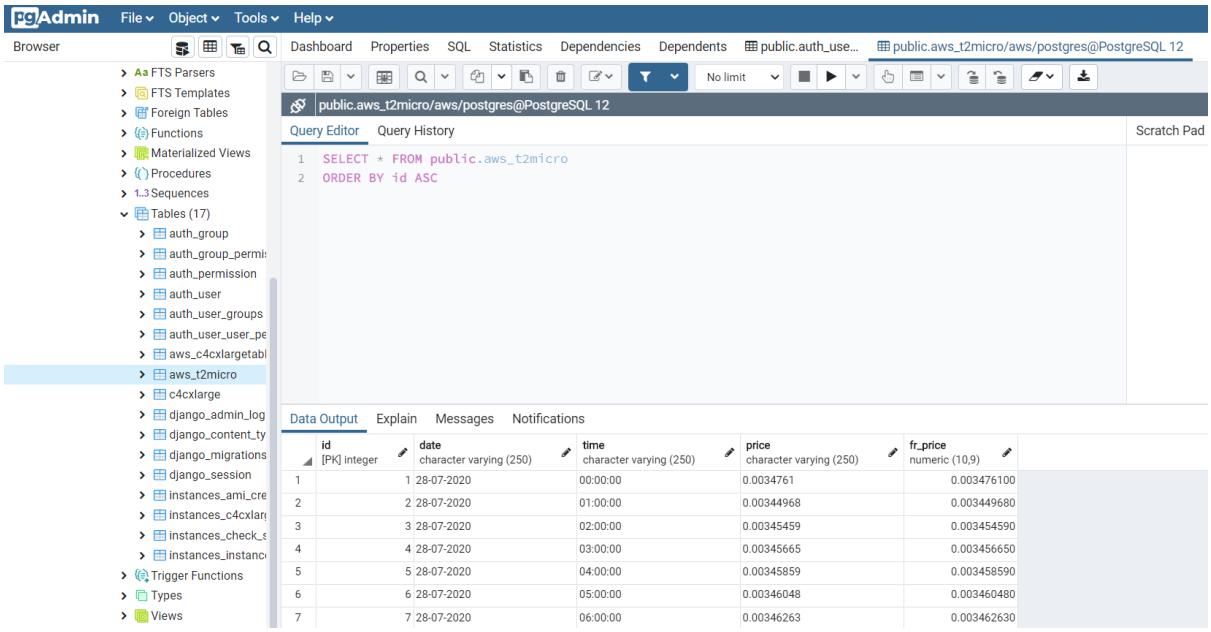

Figure 5: Postgres and PgAdmin

2.2.3 Implementation of Failure Recovery Module

Spot instance failure recovery module is implemented using various phases namely spot instance AMI creation, on-demand instance creation, on-demand AMI creation, spot instance creation. These phases would form a spot instance life cycle as proposed in the research document. Here the implementation of each phase is shown in the following figures.

Ami Creation The following figures would provide the implementation of AMI creation of the running instance in the proposed application. The logic behind the AMI creation of spot instance and on-demand instance could be observed from the below figures. As per the proposal, an AMI would be created for the running spot instance if the predicted spot price for that instance type is greater than the user bid price. An AMI would be created for the on-demand instance if the user bid price is equal to the current spot price of that instance type. The definition of AMI creation is also presented. It would create an AMI for that particular instance with the help of the instance id.

```
def ami_creation_init(in_id, session):

    image = session.create_image(
        BlockDeviceMappings=[
            {
                'DeviceName': '/dev/xvda',
                'VirtualName': 'string',
                'Ebs': {
                    'DeleteOnTermination': True,
                },
            },
        ],
        Description=in_id,
        DryRun=False,
        InstanceId=in_id,
        Name=in_id,
        NoReboot=False
    )
    # image.wait_until_exists()
    print("created ami")
    print(image['ImageId'])
    return image['ImageId']
```

```
if((sp_in == 1) & (bid_price < fr_pr) & (prs == 'running')):
    ami_id = ami_creation_spot(each, session)
    ami_obj = ami_creation(instance_id=each, ami_id=ami_id)
    ami_obj.save()

if ((sp_in == 0) & (float(bid_price) == float(cr_sp)) & (prs == 'running')):
    ami_id = ami_creation_spot(each,session)
    ami_obj = ami_creation(instance_id=each, ami_id=ami_id)
    ami_obj.save()
```

Figure 6: AMI Creation Implementation

On-Demand Instance Creation The logic behind the on-demand instance could be observed from the following figures. If the spot instance is terminated by the cloud service provider then the web application should automatically initiate the on-demand instance creation process as using the AMI of the terminated spot instance. Logic and the definition of the on-demand instance creation process is presented below:

```
def ondemandinstancecreate(session,ins_type,ter_id,img):

    created_ins = session.run_instances(ImageId=img,
                                InstanceType=ins_type,
                                MinCount=1,
                                MaxCount=1,KeyName='sarath_aws_kp',
                                SecurityGroupIds=[
                                'sg-0e5e525ee4d336a9d',
                                ],)
    for instance in created_ins['Instances']:
        cr_id =instance['InstanceId']
        print(cr_id)
        ins_cr = instance_creation(terminated_id=ter_id,creation_id=cr_id)
        ins_cr.save()
```

```
if ((prs == "terminated") & (spot_in == 1)):
    if (instance_creation.objects.filter(terminated_id=each).exists()) == 0:
        new_img = ami_creation.objects.filter(instance_id=each)
        for img_prop in new_img:
            print(img_prop.ami_id)
            new_img = img_prop.ami_id
        new_ami_det = get_platform_details( new_img,session)
        if((ami_creation.objects.filter(instance_id=each).exists() == 1) & (new_ami_det['state'] == 'available')):
            ondemandinstancecreate(session, ins_type, each, new_img)
        else:
            img = get_instance_img( each,session)
            ondemandinstancecreate(session, ins_type, each, img)
    else:
        print("already instance is created")
```

Figure 7: OnDemand Instance Creation Implementation

From the above figures, we could understand how the OnDemand instance is created in the proposed web application. Various parameters are passed as part of the creation process namely session, instance type, termination id, and img. The session attribute would hold the AWS secret key and access key. Instance Type would hold the type of the instance that is terminated. Termination id would hold the id of the terminated instance and img would hold the AMI of the terminated instance. Based on the logic that is shown corresponding action would be taken every ten seconds in order to maintain the spot instance life cycle.

Spot Instance Creation The logic behind the spot instance could be observed from the following figures. If the on-demand instance is terminated by the web application then the application should automatically initiate the spot instance creation process as using the AMI of the terminated on-demand instance. Logic and the definition of the spot instance creation process is presented below:

```
def spot_ins_creation(ami,_ins_type,az, session):
    response = session.request_spot_instances(
        DryRun=False,
        SpotPrice='0.10',
        InstanceCount=1,
        Type='one-time',
        LaunchSpecification={
            'ImageId': ami,
            'KeyName': 'sarath_aws_kp',
            'InstanceType': _ins_type,
            'Placement': {
                'AvailabilityZone': az,
            },
            'BlockDeviceMappings': [
                {

                },
            ],

            'EbsOptimized': False,
            'Monitoring': {
                'Enabled': False
            },
            'SecurityGroupIds': [
                'sg-0e5e525ee4d336a9d'
            ]
        }
    )
    for instance in response['SpotInstanceRequests']:
        req_id =instance['SpotInstanceRequestId']
    return req_id
```

Figure 8: Spot Instance Creation Method

From the above figure, it could be clearly seen that the spot instance creation method takes multiple parameters as input namely AMI, instance type, availability zone, and session. AMI would hold the AMI id of the terminated on-demand instance. Instance type would hold the type of the on-demand instance that is terminated. Availability Zone would have the region in which the spot instance should be created. The session would hold the AWS parameter of the user.

```
if ( (spot_in == 0) & (prs == "running") & (float(bid_price) == float(cr_sp))  ):
    new_img = ami_creation.objects.filter(instance_id=each)
    for person in new_img:
        new_img = person.ami_id
    new_ami_det = get_platform_details( new_img,session)
    if((new_ami_det['state'] == 'available')):
        instance_termination(each,ec2_con_re)

if ((spot_in == 0) & (prs == "terminated")):
    if (instance_creation.objects.filter(terminated_id=each).exists()) == 0:
        new_img = ami_creation.objects.filter(instance_id=each)
        for img_detail in new_img:
            print(img_detail.ami_id)
            new_img = img_detail.ami_id
        new_ami_det = get_platform_details( new_img,session)
        if ((ami_creation.objects.filter(instance_id=each).exists() == 1) & (new_ami_det['state'] == 'available')):
            req_id = spot_ins_creation(new_img, ins_type, az, session)
            ins_cr = instance_creation(terminated_id=each, creation_id=req_id)
            ins_cr.save()
        else:
            img = get_instance_img( each,session)
            req_id = spot_ins_creation(img, ins_type, az, session)
            ins_cr = instance_creation(terminated_id=each, creation_id=req_id)
            ins_cr.save()
```

Figure 9: Spot Instance Creation Logic

From the above image, it could be clearly seen that the logic for the spot instance creation that is used in the web application.

3 User Requirements

To make use of the AWS Instance Management application the user must have the following pieces of information. The user must have an AWS account and they should have these parameters namely aws_access_key, aws_secret_key, and region from that account.

3.1 AWS Account Creation

If the user does not have an AWS account then they could go the following link https://portal.aws.amazon.com/billing/signup to create an account. The user should provide the following parameters to create an account with the AWS.

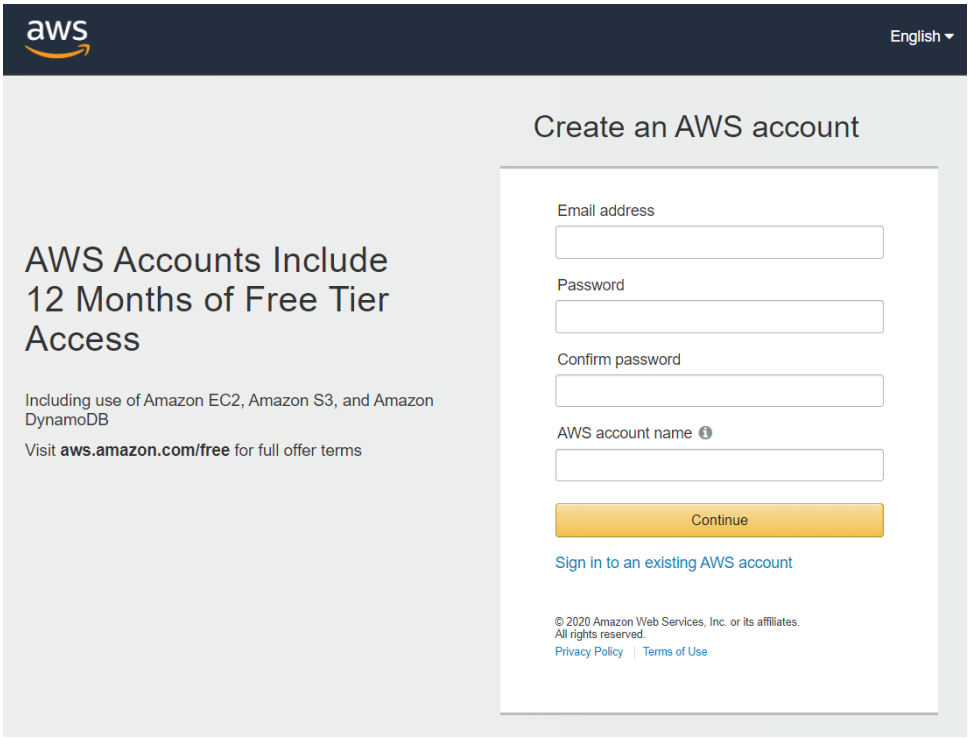

Figure 10: AWS Account Creation

3.2 Required AWS Parameters

If the user have an exsisting account with AWS then the user user could get the aws_access_key, aws_secret_key, and region from that account as shown below:

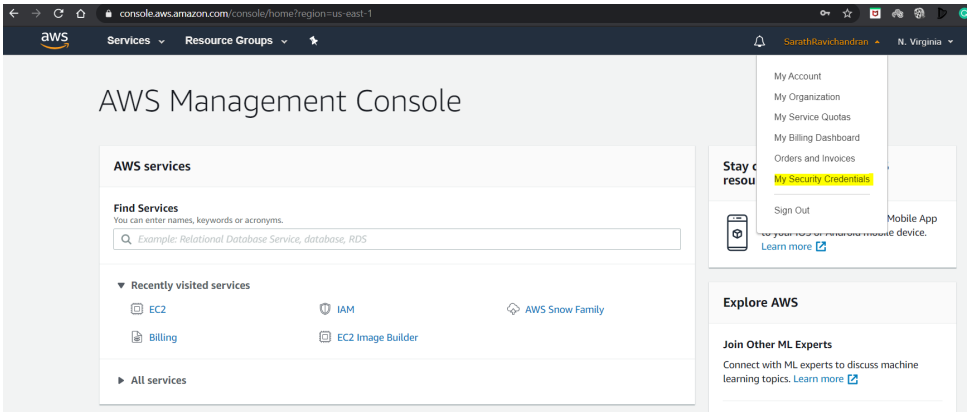

Figure 11: AWS Account Security

Once the user logged into the AWS account the user could see my security credential option as shown in the above diagram. After clicking on that option then the user should see the below figure. Through which the user could get the required parameters.

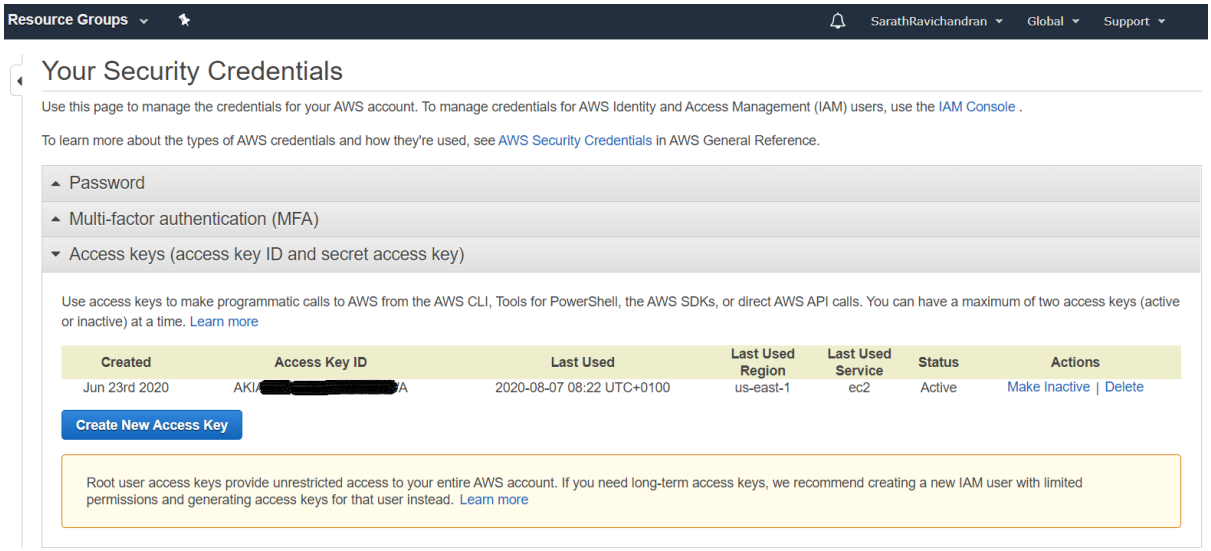

Figure 12: AWS Access Key and AWS Secret Key

4 AWS Instance Management Modules

Once the user has all the required parameter then the user could use our application through this link:

AWS Instance Management: https://awsinstancemanagement.herokuapp.com/

4.1 Search Form

Once the user registers with the application then they should log in to this application to make use of the failure recovery module. After the login to the application, the user could see a screen like below where they could provide the required parameters in the search form to make use of this proposal.

Figure 13: AWS Instance Management Application Search Form

4.2 Home Screen

Once the user hits the above URL they would be able to see the following home screen. Through which they could see the forecaster spot price for the instances. It helps them to register and login to our application

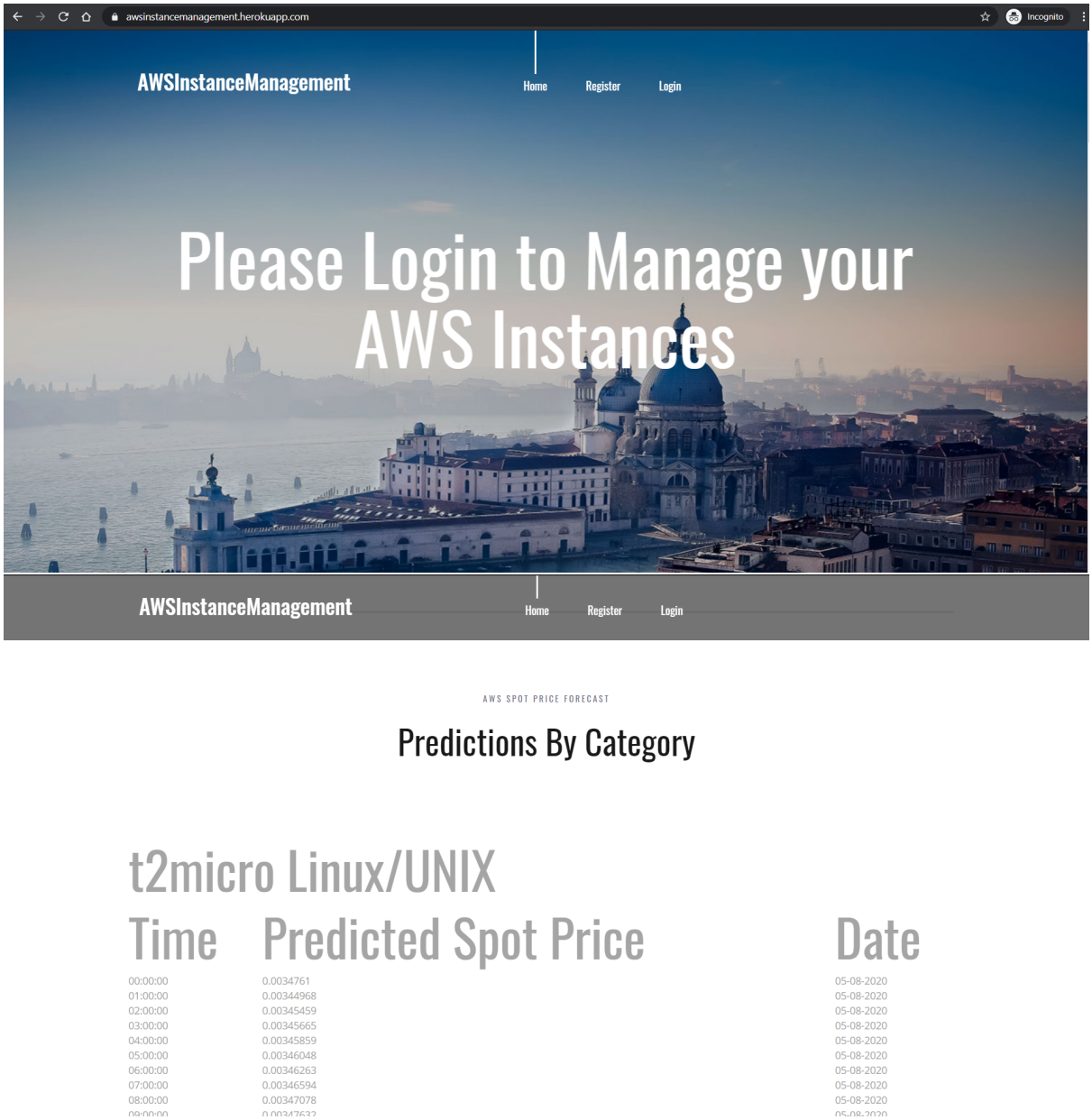

Figure 14: AWS Instance Management Application Home Screen

4.3 Registration Form

Once the user clicks on the registration tab of the home screen the user could see the following form that would allow them to register to the AWS Instance Management application.

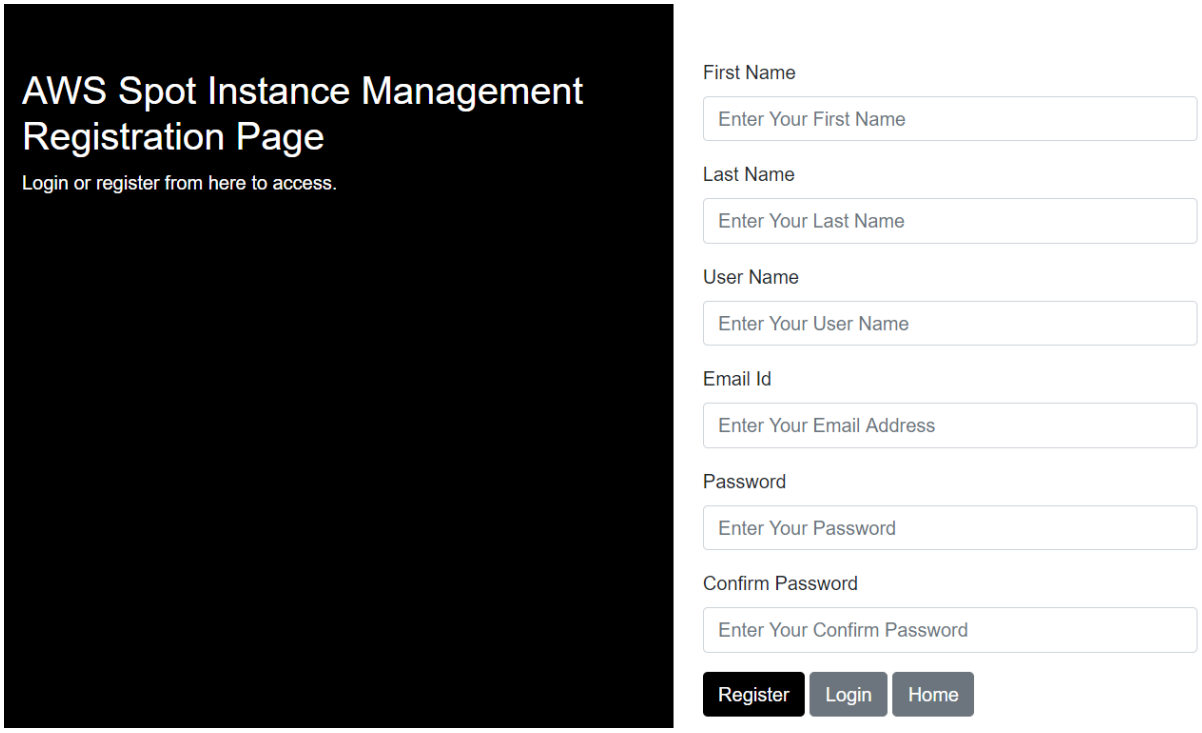

Figure 15: AWS Instance Management Application Registration Screen

4.4 Login Form

Once the user clicks on the login tab of the home screen the user could see the following form that would allow them to log in to the AWS Instance Management application.

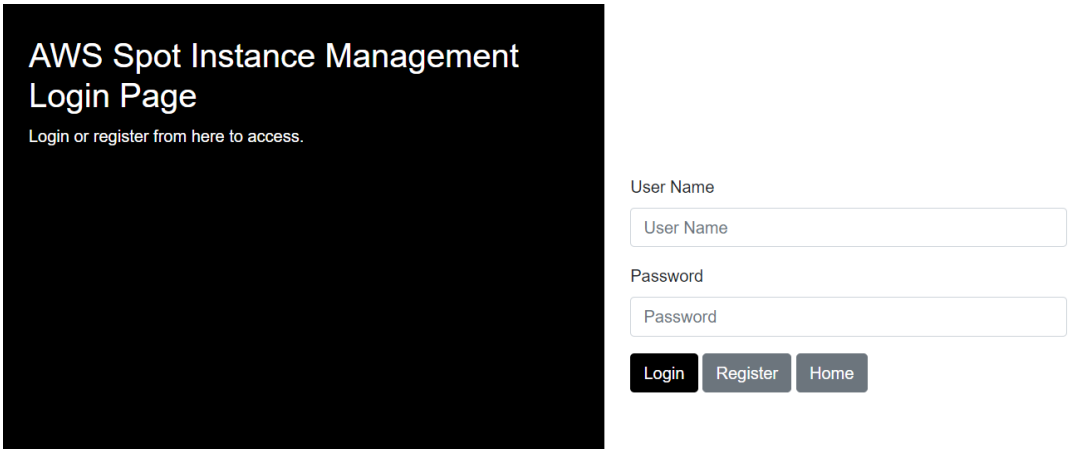

Figure 16: AWS Instance Management Application Login Screen

4.5 Admin Screen

If the user is already logged in to AWS Instance Application and if he is admin then they could see the admin tab in the home screen as shown in the below figure. Through which the user could interact with the tables that are present in the database.

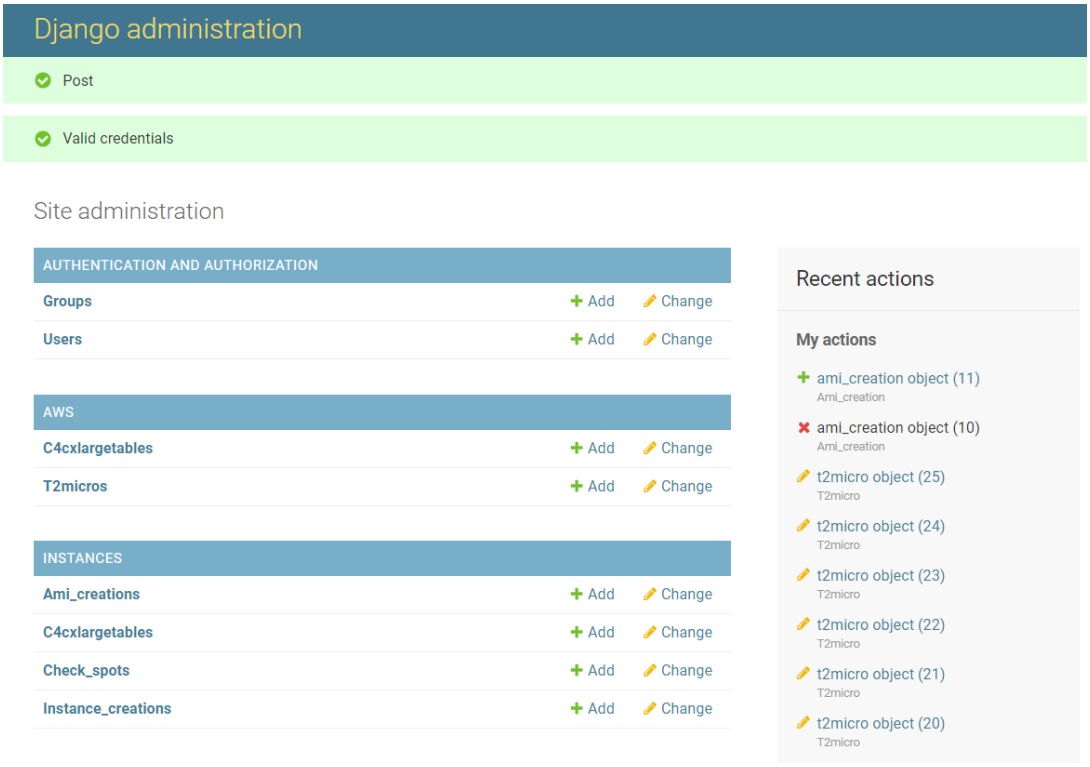

Figure 17: AWS Instance Management Application Admin Screen

4.6 Search Result Screen

Once the user makes a valid search operation after the login to the application. They would be able to see a search result as shown below with all the instances that are associated with the searched AWS parameters.

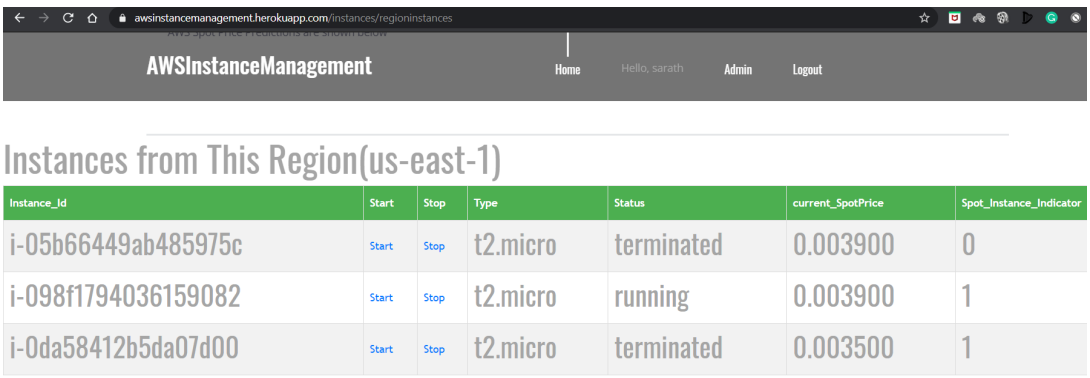

Figure 18: AWS Instance Management Application Admin Screen

References

[1] "Composite learning index: CLI," Amazon, 2006. [Online]. Available: https://docs.aws.amazon.com/cli/latest/reference/ec2/.

[2] "EC2 — Boto3 Docs 1.14.39 documentation", Boto3.amazonaws.com, 2020. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ec2.html.

[3] Khandelwal, V., Chaturvedi, A. and Gupta, C. P. (2017). Amazon ec2 spot price predic-tion using regression random forests,IEEE Transactions on Cloud Computingpp. 1–1.Khatua, S. and Mukherjee, N. (2013).

[4] A novel checkpointing scheme for amazon ec2spot instances,2013 13th IEEE/ACM International Symposium on Cluster, Cloud,and Grid Computing, pp. 180–181.Pham, T., Ristov, S. and Fahringer, T. (2018).

[5] Aresilient auction framework for deadline-aware jobs in cloud spot market,2017 IEEE36th Symposium on Reliable Distributed Systems (SRDS), pp. 247–249.Song, Y., Zafer, M. and Lee, K. (2012).

[6] Monetary cost-aware checkpointing andmigration on amazon cloud spot instances,IEEE Transactions on Services Computing5(4): 512–524.Zheng, L., Joe-Wong, C., Tan, C. W., Chiang, M. and Wang, X. (2015).