Assignment 2-ECE1779 - Documentation

Group Member: 1
Name: Hanyu Xi
Student Number: 1007489950

How to start the project

1. Extract the ECE1779A2.pem
   tar -xvf a1.tar
2. Make sure the EC2 instance (i-0fa1c6c098f541952) is running. See it in figure 1
3. Make sure RDS database assignment1-ece1779 is in available status
4. Put ECE1779A1.pem under your ~/.ssh directory. If "~/.ssh" doesn't exist, mkdir ~/.ssh.
5. Once pem file is in your .ssh directory, Check the public IPv4 address of worker instance (i-0fa1c6c098f541952) in your aws console (for example, ip address:3.84.146.98)
6. Run the following command to ssh into EC2, figure 3
   ssh -i ~/.ssh/ECE1779A2.pem ubuntu@3.84.146.98
7. Once you ssh into the EC2 instance. We need to cd into code repository directory: cd ~/Desktop/Assignment2/manager_app
8. Make sure to update the AWS credentials. Find the AWS credentials (IN Figure 4), open vi ~/.aws/credentials and delete expired tokens and keys and updated with the latest ones.
9. Before execute the start.sh script: ./start.sh
10. The website should be running like figure 5. The website is running on port 5000 so you can access the website like this: 3.84.146.98(your Public IPv4 address):5000

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
–	i-0fa1c6c098f541952	Running	t2.medium	–	No alarms	us-east-1b	ec2-3-84-146-98.com...	3.84.146.98	–
–	i-034d576ebac0b171e	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-0c356294220362301	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-098aa80dd5b47a2c1	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-0e44b864d57a31050	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-0a667b8e0fbc141a8	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-025e933b3fec6ede3	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-043e32e29c5d513c1	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–
–	i-0bf82b83427b6eadf	Terminated	t2.small	–	No alarms	us-east-1b	–	–	–

Figure 1

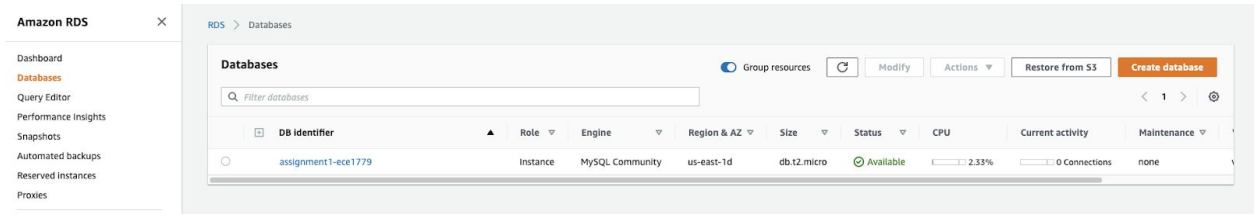


Figure 2

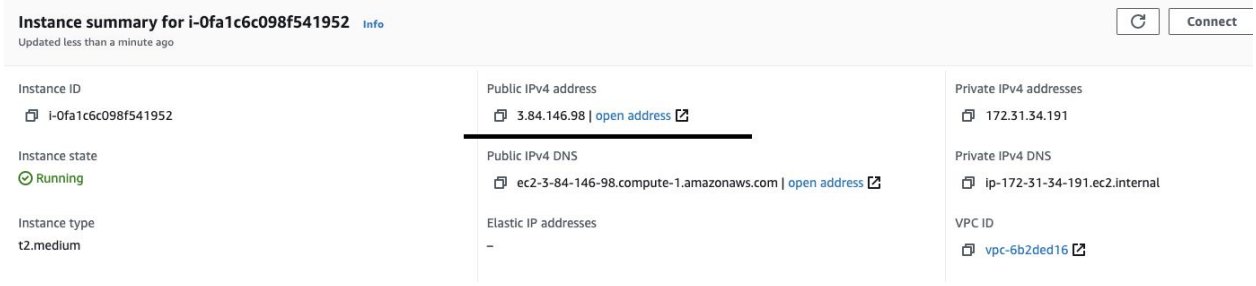


Figure 3

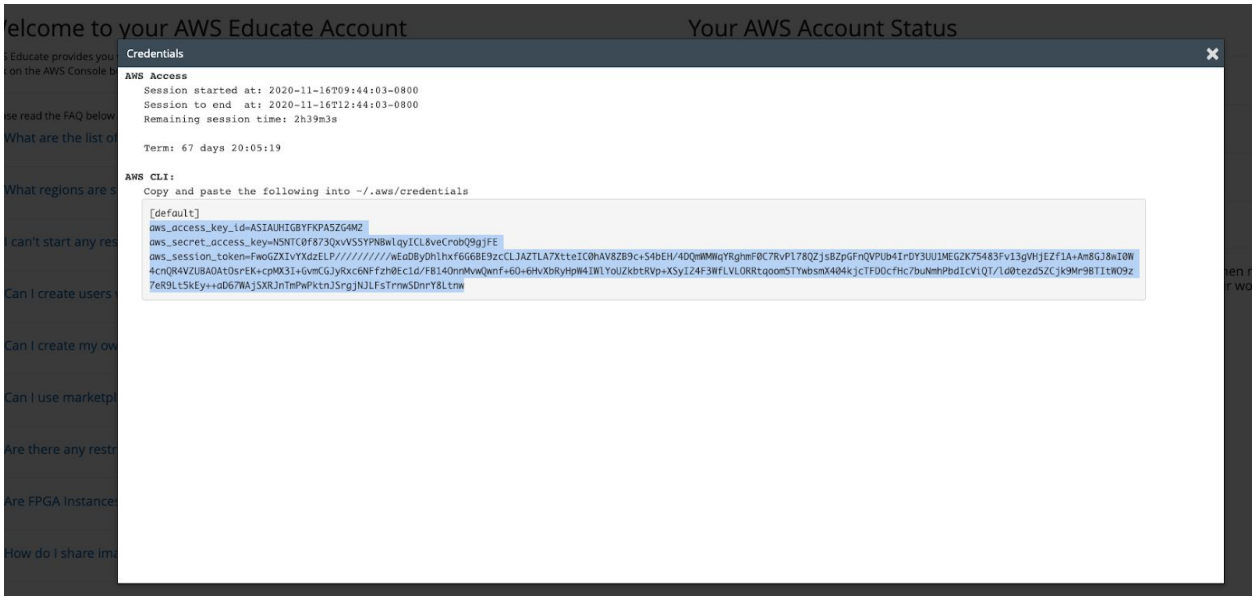


Figure 4

Assignment 2

Load Balanced Entry URL

The number of workers for the past 30 minutes: 9

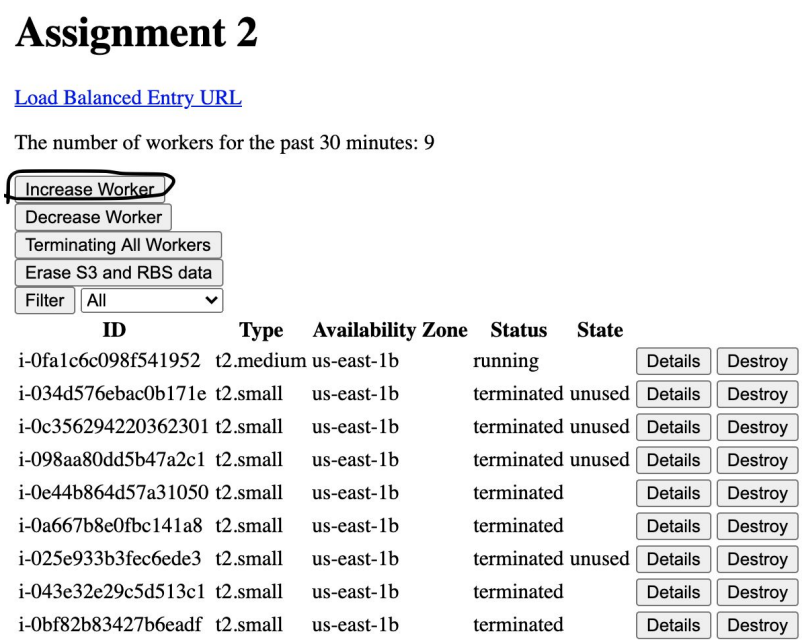
Increase Worker
Decrease Worker
Terminating All Workers
Erase S3 and RBS data
Filter All

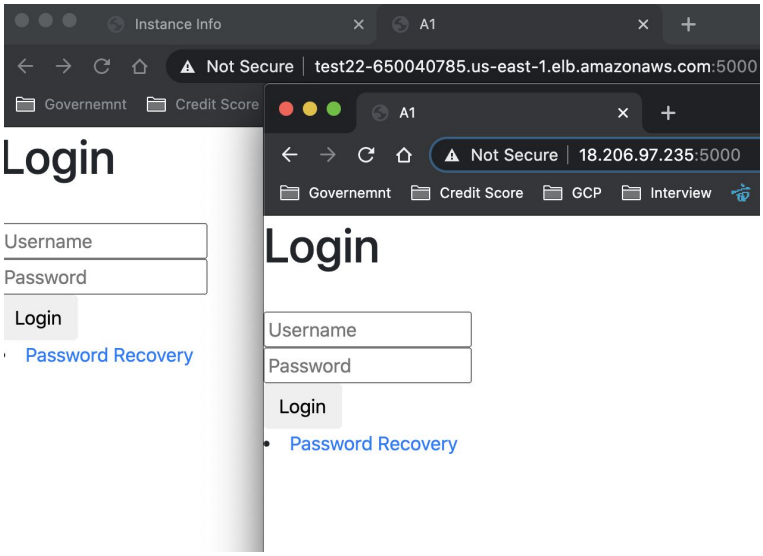
ID	Type	Availability Zone	Status	State		
i-0fa1c6c098f541952	t2.medium	us-east-1b	running		Details	Destroy
i-034d576ebac0b171e	t2.small	us-east-1b	terminated	unused	Details	Destroy
i-0c356294220362301	t2.small	us-east-1b	terminated	unused	Details	Destroy
i-098aa80dd5b47a2c1	t2.small	us-east-1b	terminated	unused	Details	Destroy
i-0e44b864d57a31050	t2.small	us-east-1b	terminated		Details	Destroy
i-0a667b8e0fbc141a8	t2.small	us-east-1b	terminated		Details	Destroy
i-025e933b3fec6ede3	t2.small	us-east-1b	terminated	unused	Details	Destroy
i-043e32e29c5d513c1	t2.small	us-east-1b	terminated		Details	Destroy
i-0bf82b83427b6eadf	t2.small	us-east-1b	terminated		Details	Destroy

Figure 5

How to use the application?

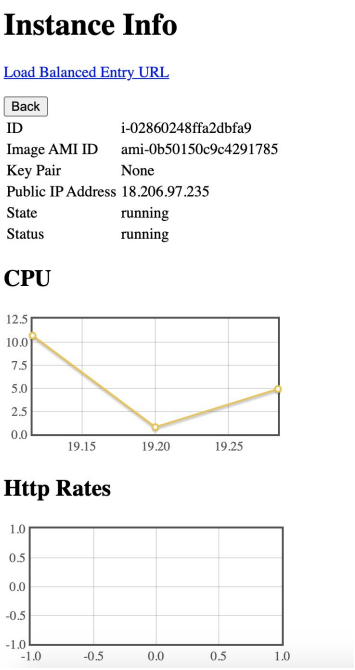
Main Page





1. Click on “Increase Worker”, it will bring up a new worker instance.The state of the new worker instance will change from “initial” to “healthy/unhealthy”. After the state changes to “healthy/unhealthy”, you can check the “Load Balanced Entry URL” or public address to view the work application. For the testing on work application, please refer to first assignment documentation.

2. The line under “Increase Worker” button is the chart showing the number of workers for the past 30 minutes
3. “Decrease Button”, it will terminate the workers with the “non draining” state. For instance, if the state is in “healthy, unhealthy or initial”, it will be terminated.
4. “Terminate All Workers” button will terminate all the workers
5. “Erase S3” button will erase all the pictures under the prefix '/home/ubuntu/Desktop/Assignment1/MaskImageDetectionWebApp/app/static/images/test/work/’ in bucket “ece1779a2hanyu”



Worker Info Page

1. The first graph shows the total CPU utilization of the worker for the past 30 minutes with the resolution of 1 minute
2. The second chart shows the rate of HTTP requests received by this worker.

Useful link

S3:https://s3.console.aws.amazon.com/s3/buckets/ece1779a2hanyu?region=us-east-1&tab=objects
RDS: https://console.aws.amazon.com/rds/home?region=us-east-1#databases:Instances

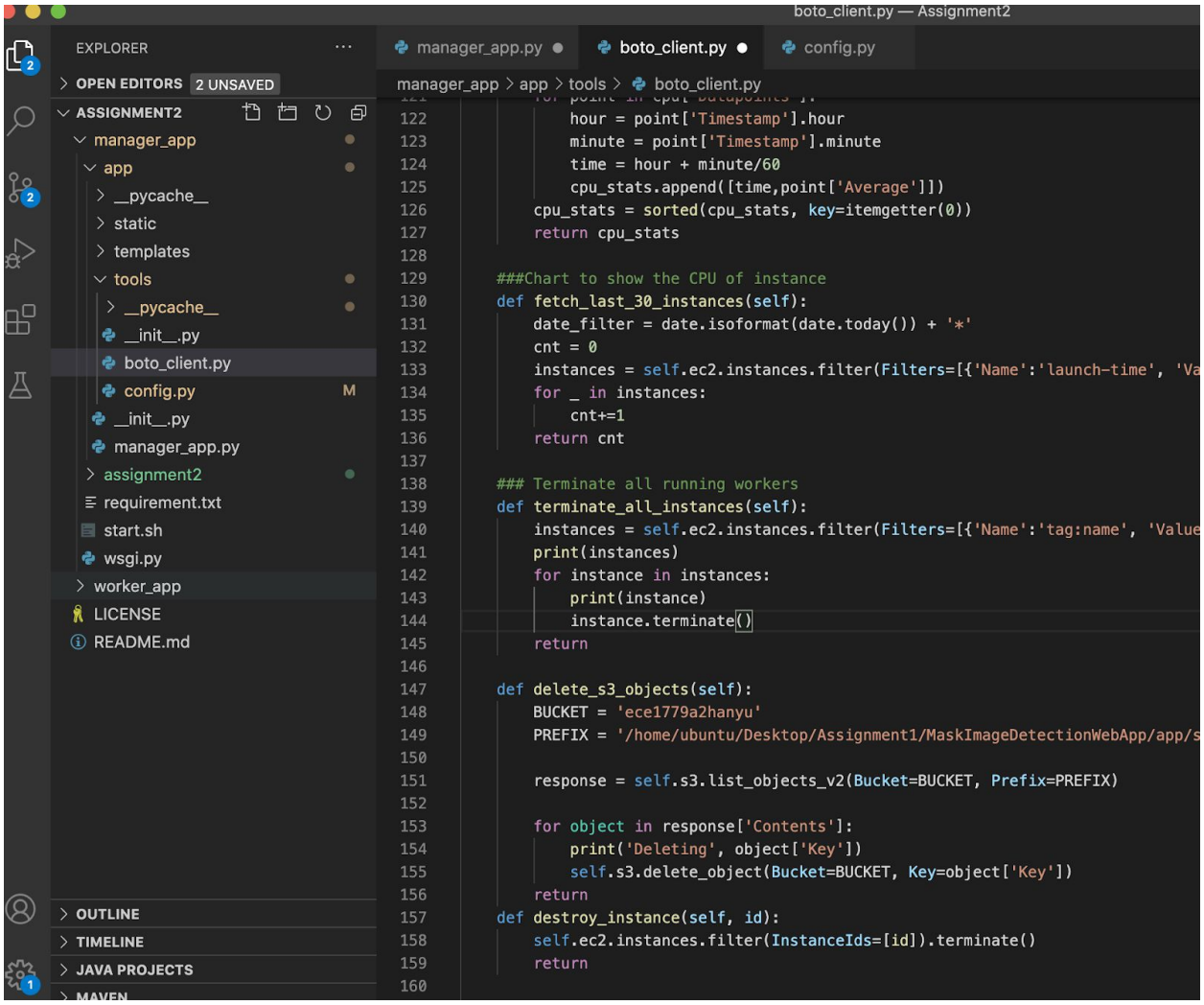
Database Schema

In order to migrate from MySQL Assignment 1 database schema to AWS RDS, the new script has been created as rds_db.py . It's under Assignment2/worker_app/db/rds_db.py. Pymysql library was used to connect to the RDS database and insert the default values (user: admin, password: admin). Overall the schema remains the same, compared to assignment 1.

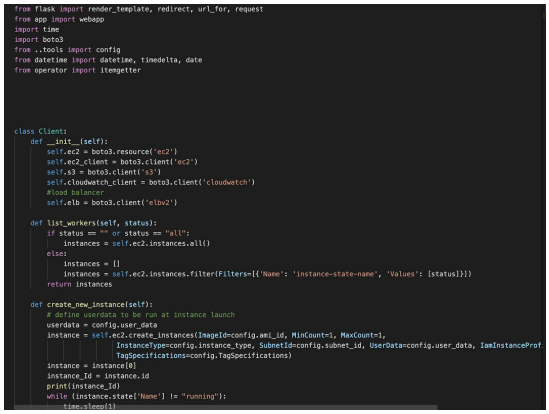
```
import pymysql
#import aws_credentials as rds
conn = pymysql.connect(
       host= "assignment1-ece1779.cgm5nkly1umo.us-east-1.rds.amazonaws.com", #endpoint
link
       port = 3306, # 3306
       user = "admin", # admin
       password = "adminadmin", #adminadmin
       db="assignment1-ece1779"
       )
##Update here from SQL file
#Table Creation
#cursor=conn.cursor()
#create_table="""
#INSERT INTO `users` (`id`, `username`, `password`, `email`, `admin`,
`api_registered`) VALUES (1, 'test',
'pbkdf2:sha256:150000$LcMw2gg3$c9dedb1034a04d74ce9f9f0cf5ab64ceefa04a8665e2d20183756e2
89e10aaeb', 'laoxi02@gmail.com', true, false);
#"""
#cursor.execute(create_table)
#conn.commit()

def get_details():
   cur=conn.cursor()
   cur.execute("SELECT *  FROM users")
   details = cur.fetchall()
   print(details)
   return details
get_details()
```

Architecture of Code

Manager App



The manager application uses a Singleton design pattern. The boto_client.py under tools is a Client class that provides all the necessary tools, like connection to different aws clients, getting metrics information from CloudWatch, update/delete on s3, etc.



```python
from flask import render_template, redirect, url_for, request
from app import webapp
import time
import boto3
from ..tools import config
from datetime import datetime, timedelta, date
from operator import itemgetter



class Client:
    def __init__(self):
        self.ec2 = boto3.resource('ec2')
        self.ec2_client = boto3.client('ec2')
        self.s3 = boto3.client('s3')
        self.cloudwatch_client = boto3.client('cloudwatch')
        #load balancer
        self.elb = boto3.client('elbv2')

    def list_workers(self, status):
        if status == "" or status == "all":
            instances = self.ec2.instances.all()
        else:
            instances = []
            instances = self.ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values': [status]}])
        return instances

    def create_new_instance(self):
        # define userdata to be run at instance launch
        userdata = config.user_data
        instance = self.ec2.create_instances(ImageId=config.ami_id, MinCount=1, MaxCount=1,
                                             InstanceType=config.instance_type, SubnetId=config.subnet_id, UserData=config.user_data, IamInstanceProf
                                             TagSpecifications=config.TagSpecifications)
        instance = instance[0]
        instance_Id = instance.id
        print(instance_Id)
        while (instance.state['Name'] != "running"):
            time.sleep(1)
```

The config.py is where I defined the AMI Id and worker instance information.
“RUN CMD” is used to spin up the flask application right after the instance is initialized. Since there is a delay until the instance enters the “running state”. I have used a while loop to constantly check if the new created instance is in “RUNNING”.

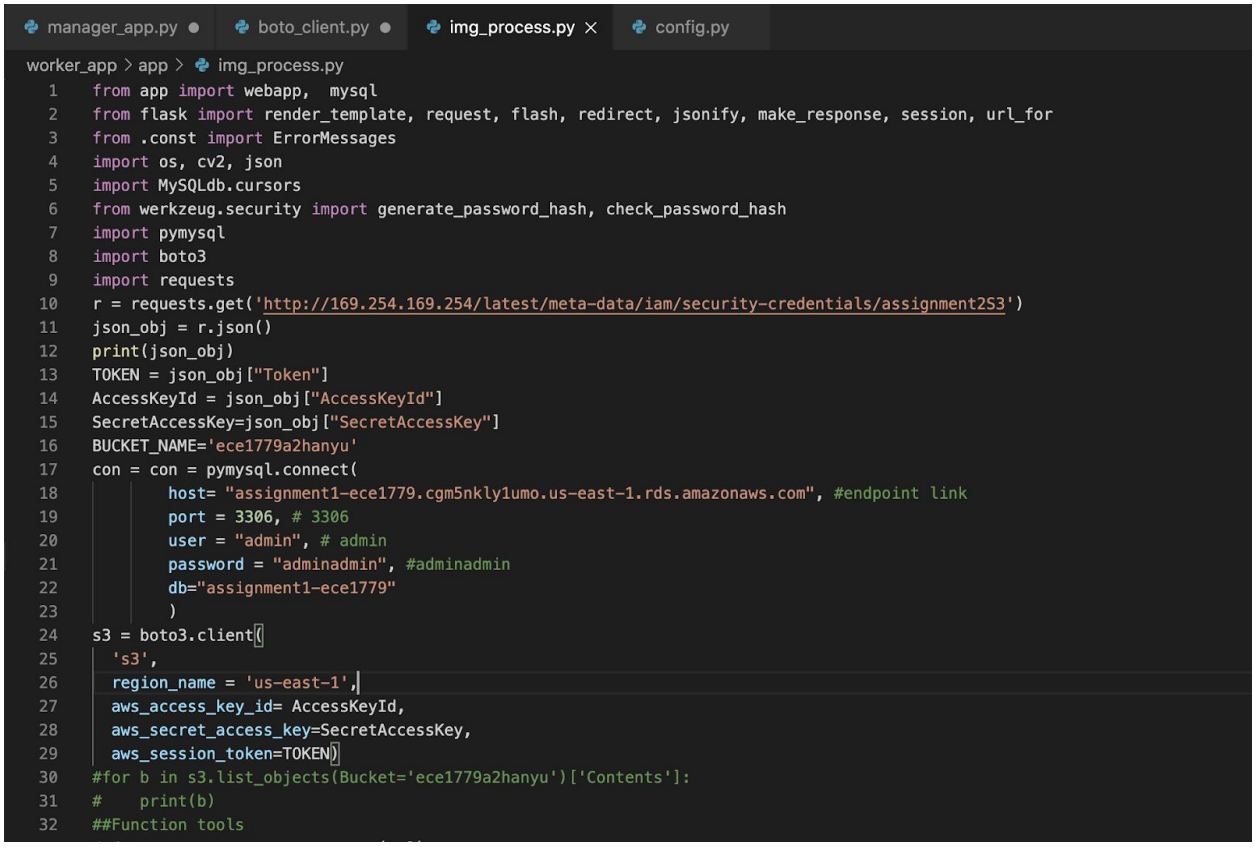
manager_app.py ● boto_client.py ● img_process.py config.py ×

manager_app > app > tools > config.py

```python
#ami_id = 'ami-0bf618774e7879c6a'
ami_id = 'ami-0b50150c9c4291785'
subnet_id = 'subnet-04ed5d5b'
target_group_arn = 'arn:aws:elasticloadbalancing:us-east-1:290459861332:targetgroup/a2/cf3d370e735dd362'
load_balancer_arn ='arn:aws:elasticloadbalancing:us-east-1:290459861332:loadbalancer/app/test22/0a81dea572c4ab23'
security_group = ['sg-07ab8eeb13e883e5a']
monitoring_status = True
instance_type = 't2.small'
user_data = """#cloud-config
runcmd:
- cd /home/ubuntu/Desktop/Assignment1/MaskImageDetectionWebApp
- cp -r /home/ubuntu/Desktop/Assignment1/MaskImageDetectionWebApp/* ~
- . config/ProjectEnv.sh
- cd ~
- python3 -m pip install torch torchvision
- python3 -m pip install opencv-python
- python3 -m pip install -r requirement.txt
- python3 wsgi.py
"""
TagSpecifications= [
    {
      'ResourceType': 'instance',
      'Tags': [
        {
          'Key': 'name',
          'Value': 'ece1779'
        },
      ]
    }]
IamInstanceProfile={'Name': 'assignment2S3'}

```

Worker App

Worker app basically remains the same from assignment 1. To be able to connect to S3, IAM role “assignment2S3” has been set up to fetch temporary TOKEN, AccessKeyId and SecretAccessKey information. Pymysql is used to connect to RDS database.
The image of AMI was created based the instance.

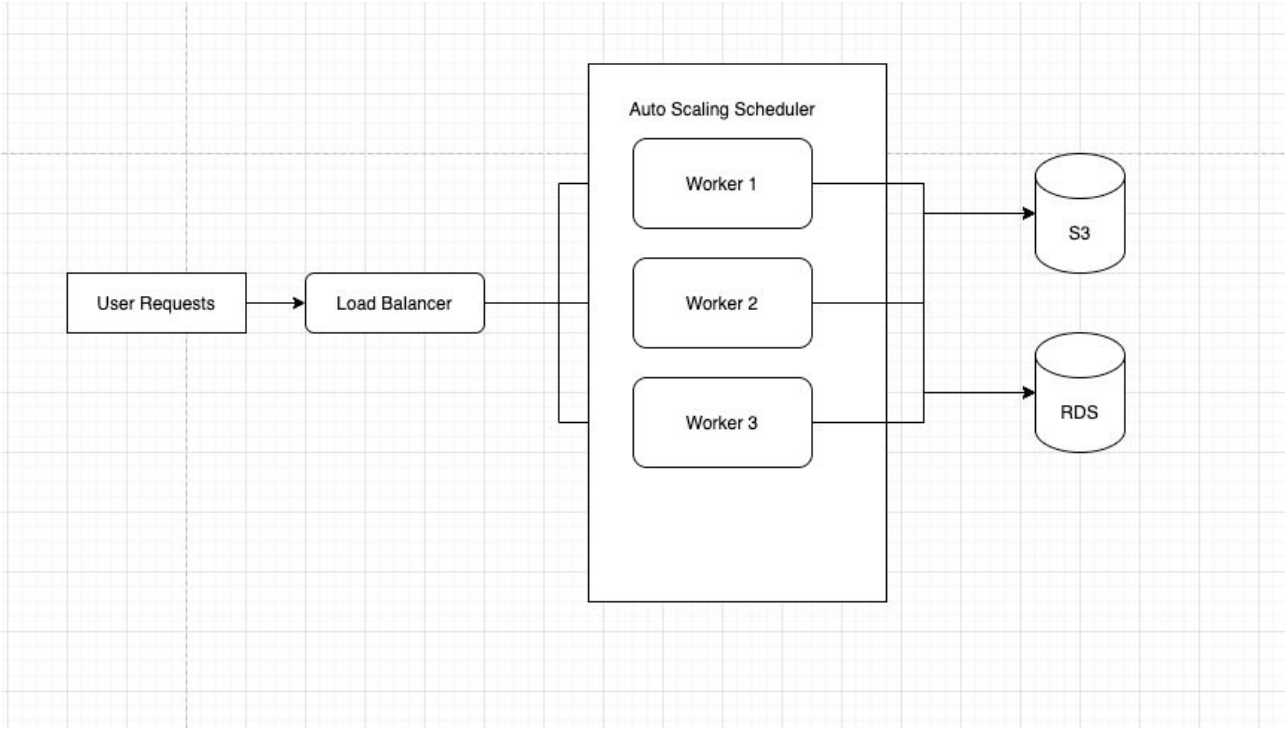


Results

Due to the amount of workload, I didn't have the chance to work on the EC2 auto-scaling feature, but here is what I would do to implement the customized auto scaler feature on AWS.

The overall workflow would be like the diagram below. The scheduler runs in a while loop, checking the CPU of each instance every 1 minute. If the average of overall CPU usage is higher than a threshold, the schedule will create one new instance to bring down the high CPU cost. In contrast, if the CPU usage is below the threshold and the number of worker instances is higher than threshold, it will terminate the "unused" worker instance to save the memory cost.

The Elastic load balancer sitting in front of the auto-scaler app uses Round-Robin technique, so when the number of worker instances goes up, the ELB will distribute the request into each worker instance based on 1,2,3,...N order.


Auto Scaling Scheduler
Worker 1
Worker 2
Worker 3
User Requests
Load Balancer
S3
RDS