Islamic University of Gaza

Faculty of Engineering

Computer Engineering Department

Parallel and Distributed System

Project " 02 "

Project Name:

Dynamic Resource Management

Prepared by:

Waseem Alfarram

ID Number:

120180182

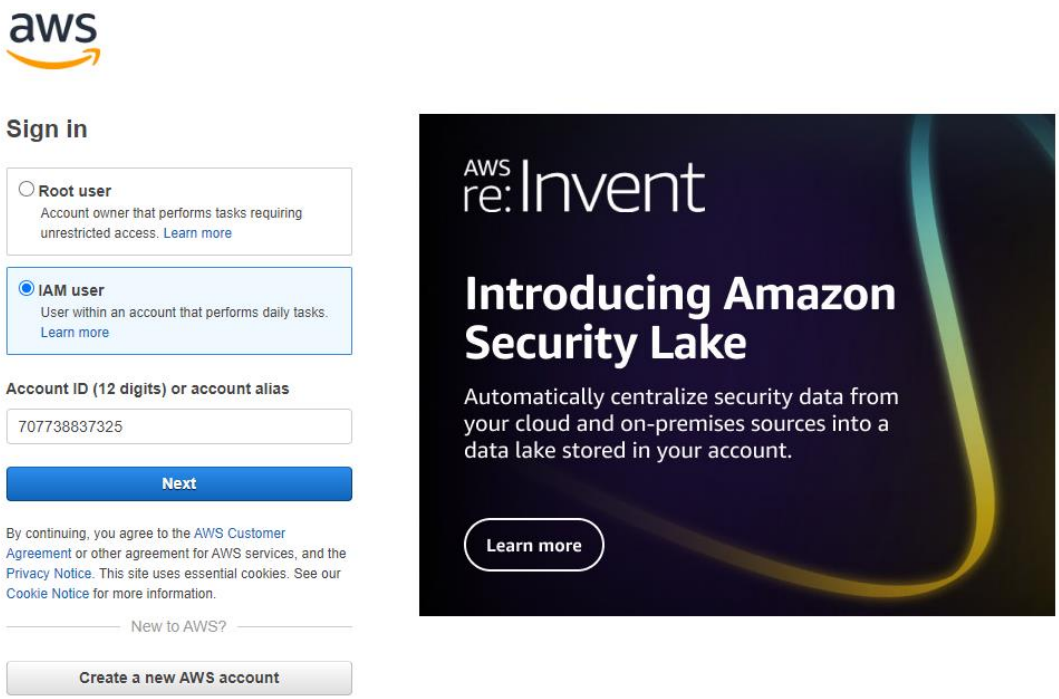
Supervised by:

Prof. Hatem M. Hamad

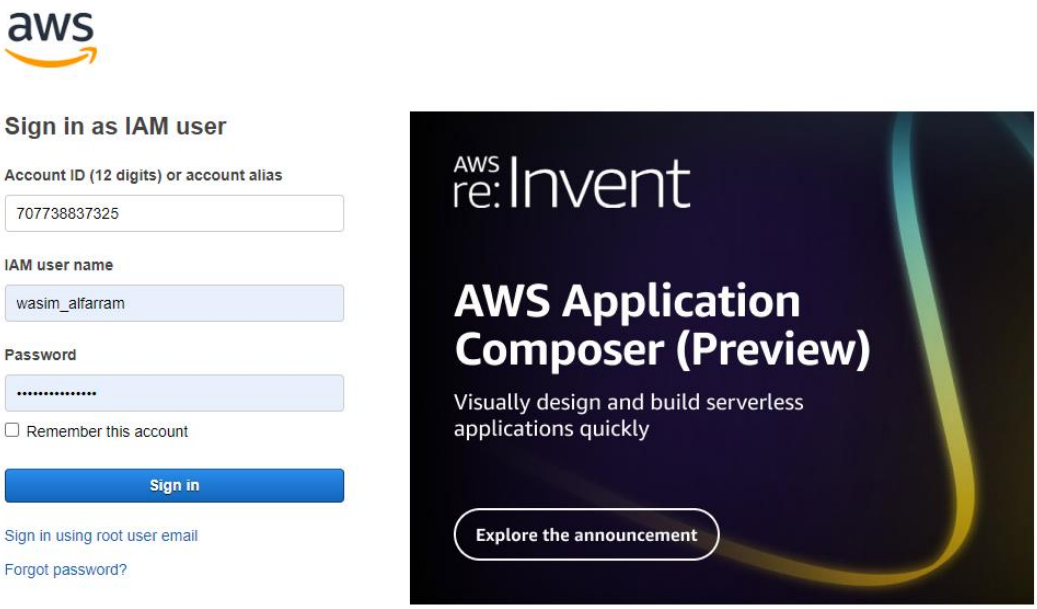
List of Content

How to login in to the Amazon account, start the instance and run the codes of the project:

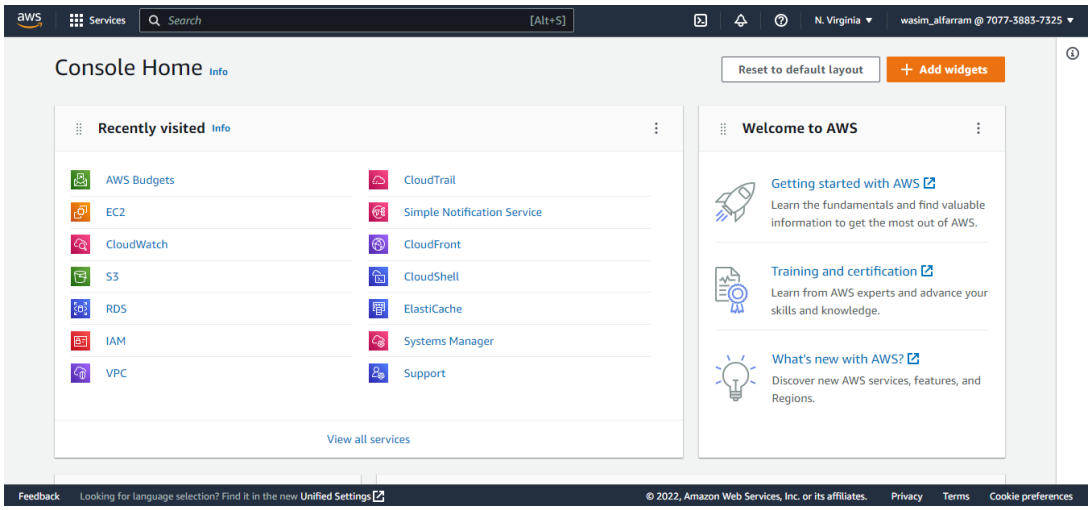
1. Go to aws.amazon.com, choose the IAM user and enter the account ID.



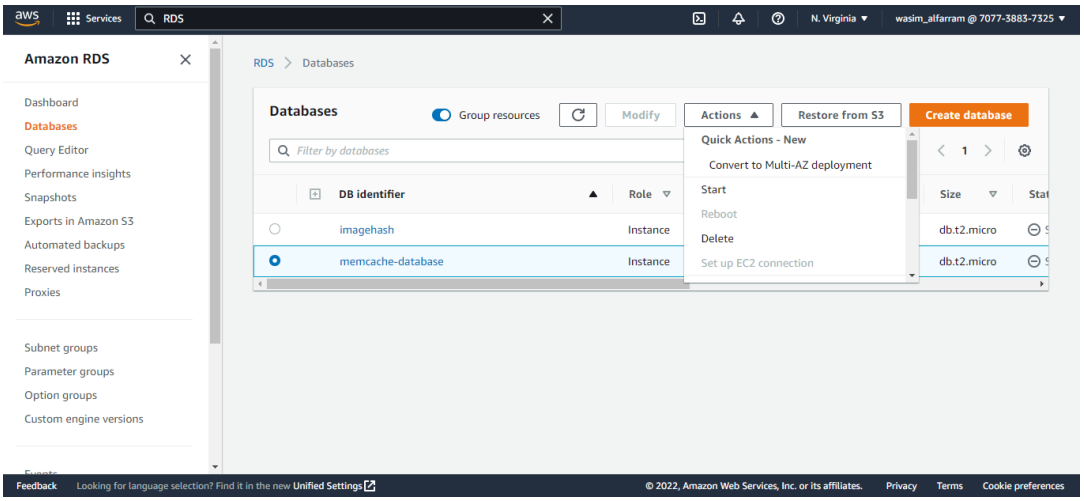
2. Enter the IAM user name and the password, the Sign in.



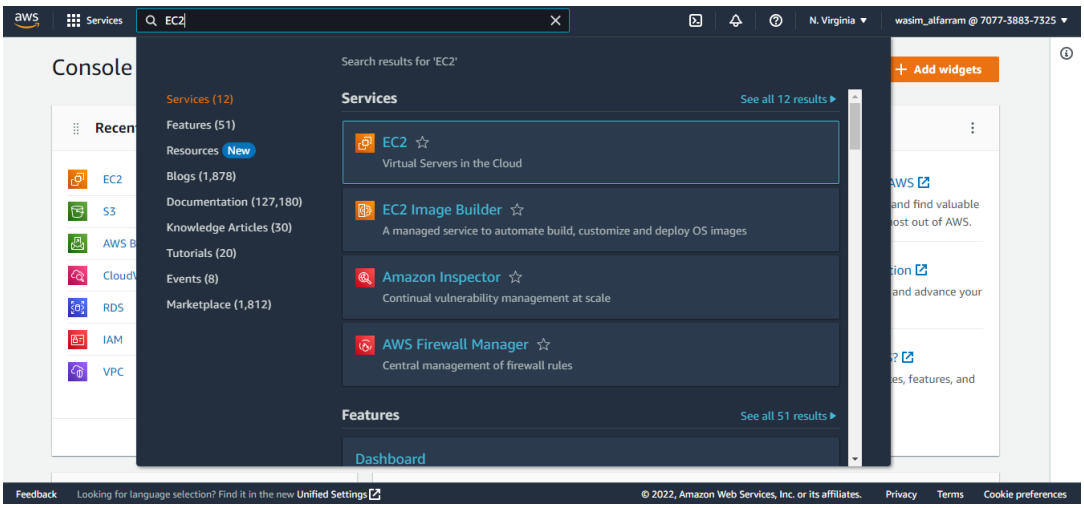
3. You will be redirected to a page like this.



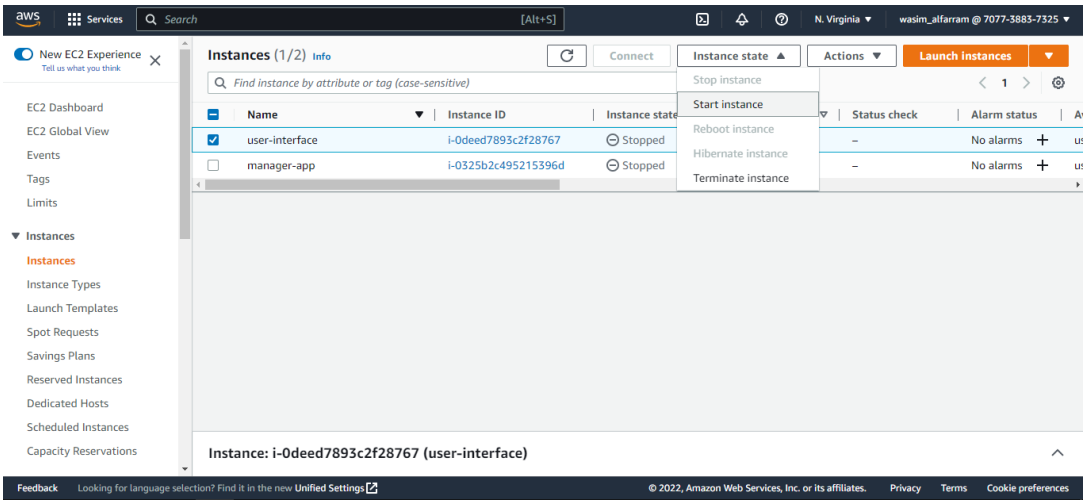
4. From the search bar above, enter **RDS** and **Start** your RDS from **Actions**.



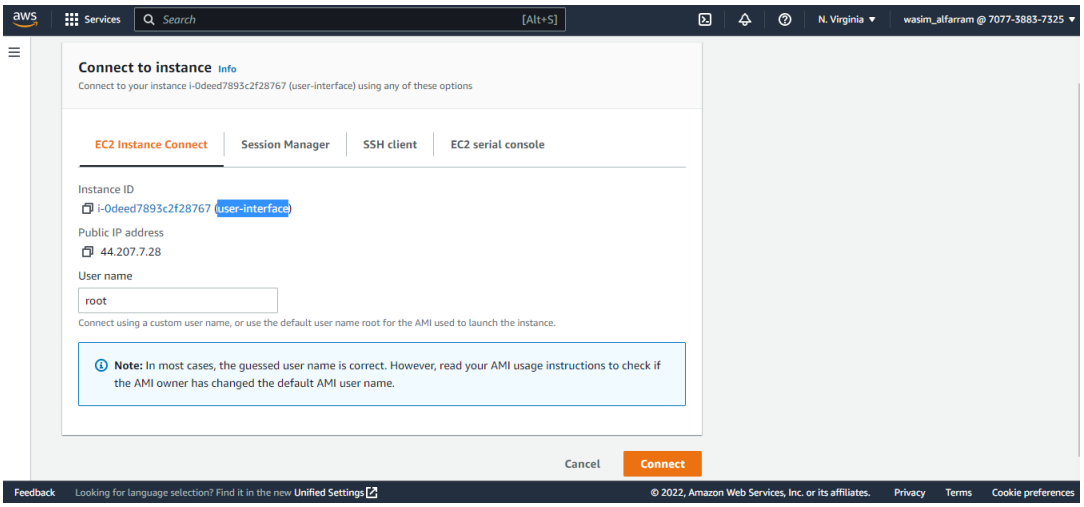
5. From the search bar above, enter **EC2** to go to your instances.



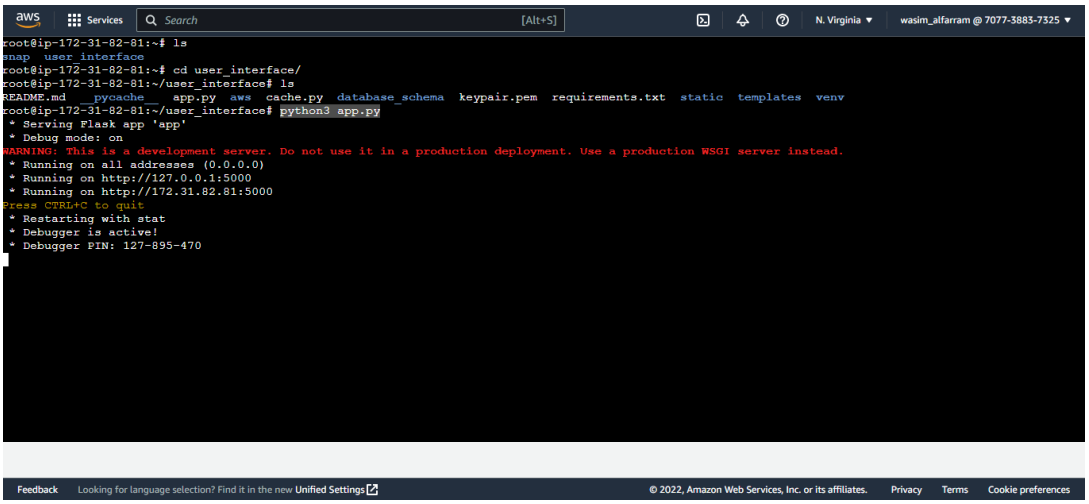
6. You will be redirected to this page, choose your EC2 instance and **Start** it.



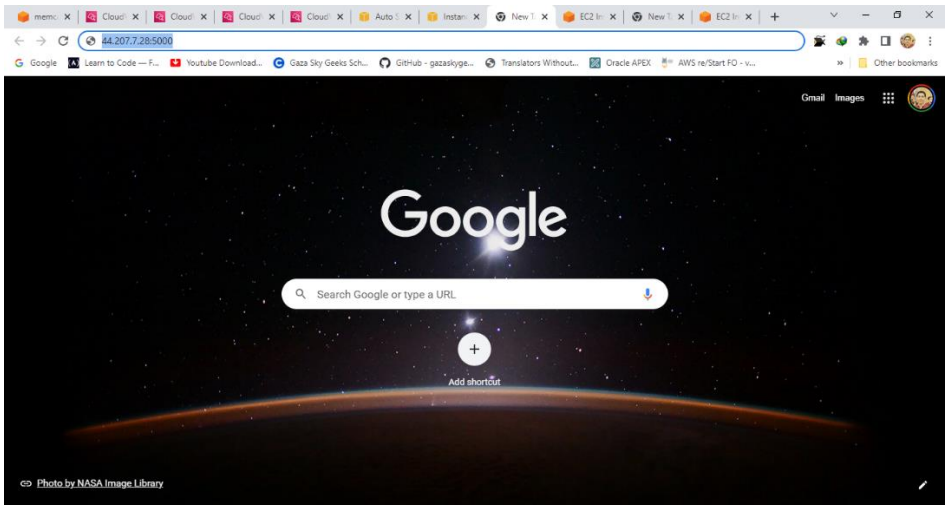
7. After it is running, **Connect** to it and copy the **Public IP address** of it.



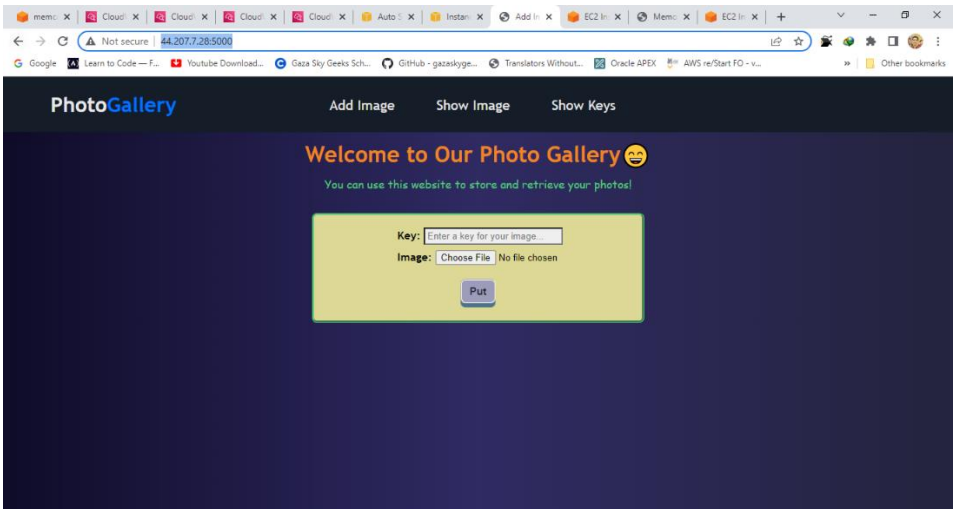
8. After connecting, you will have such a terminal, put the following commands to run the application.



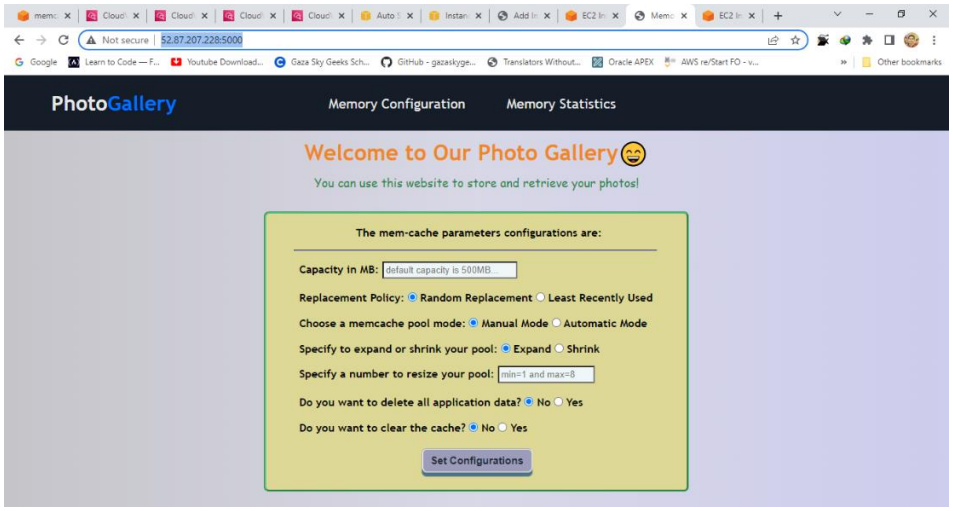
9. Paste the **Public IP** you copied before to the browser followed by **:5000**



10. You will be redirected to the **user-interface** application as shown below.

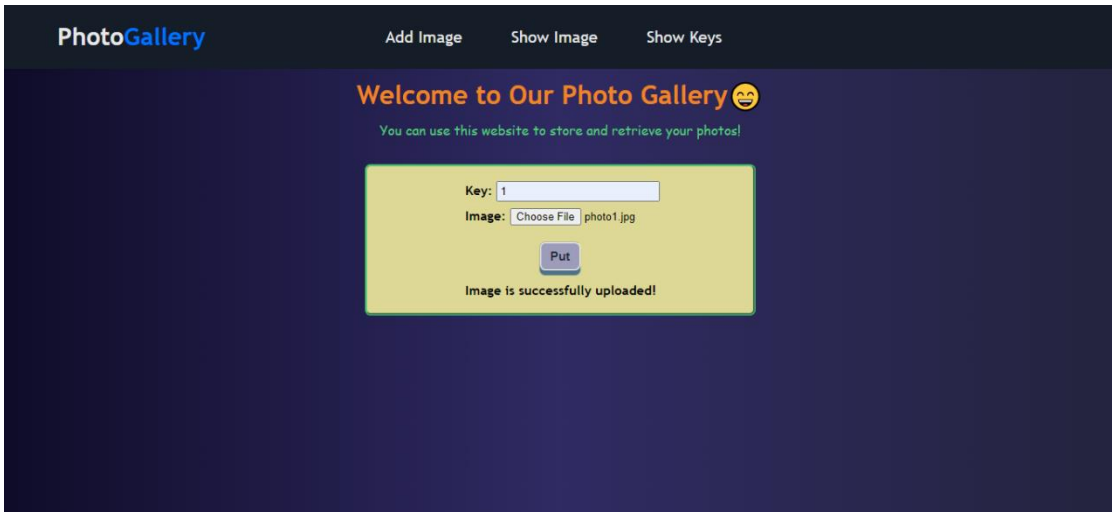


11. When running the **manager-app** application, you will have this page.

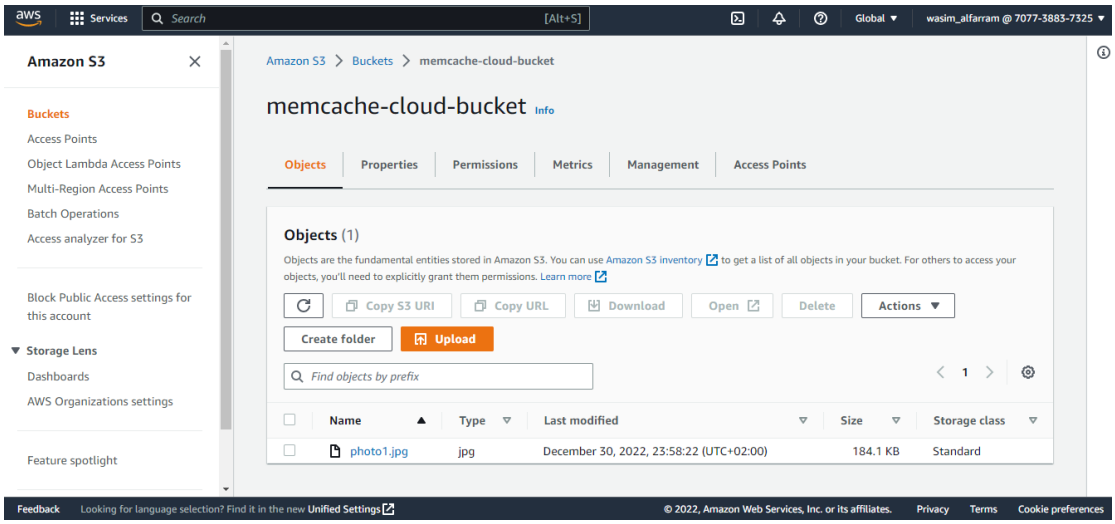


Workflow of the whole project:

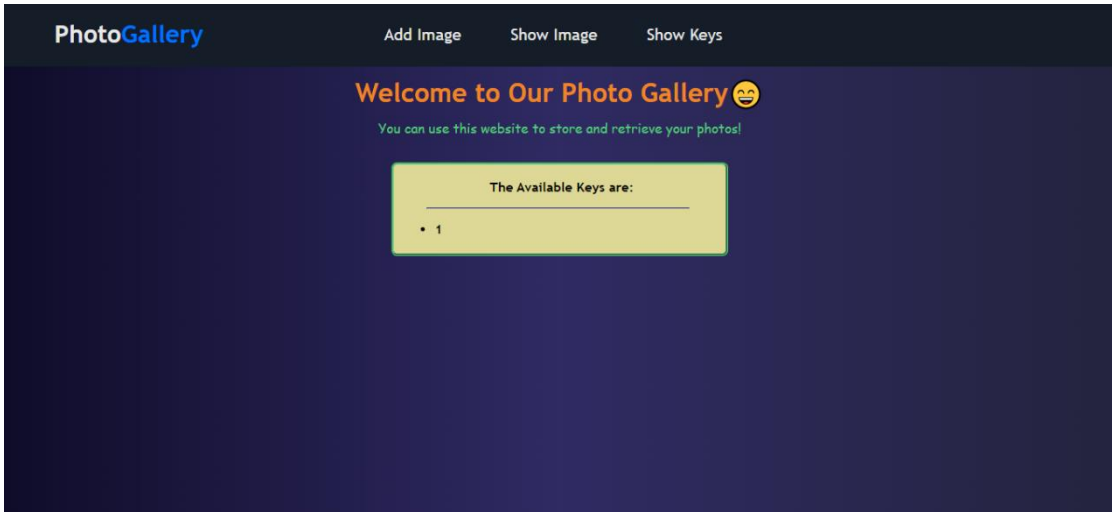
1. We are trying here to put a photo with key = **1** and name = **photo1.jpg**



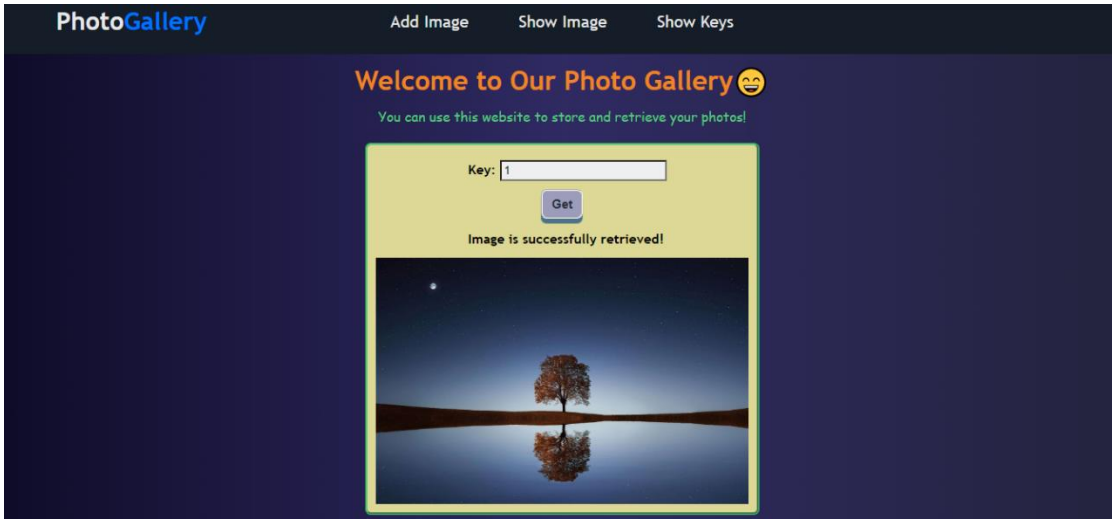
2. We can notice below that **photo1.jpg** is now in our S3 bucket.



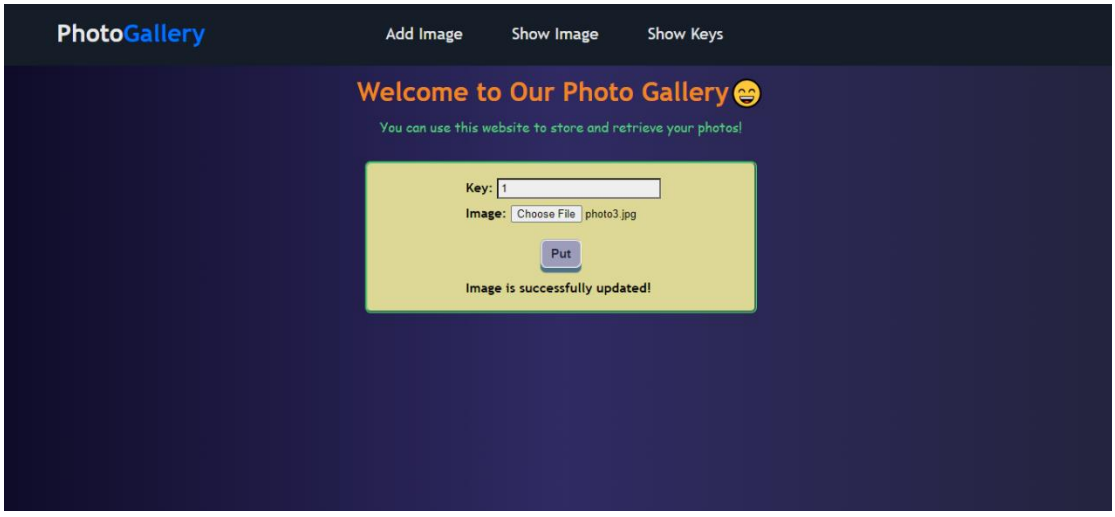
3. Also, the key **1** and **photo1.jpg** are in the database now.



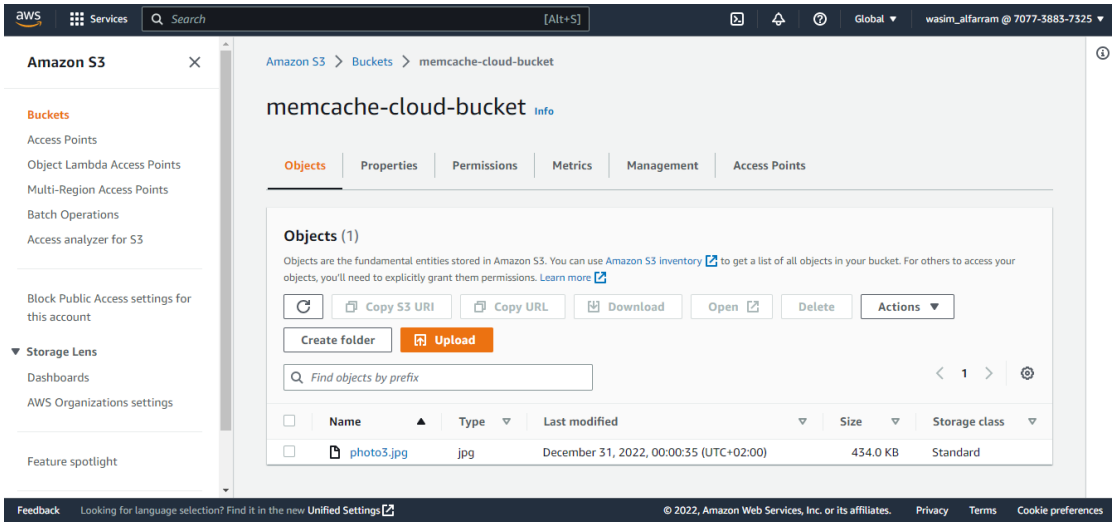
4. Here we are showing the content of key **1** which is **photo1.jpg**



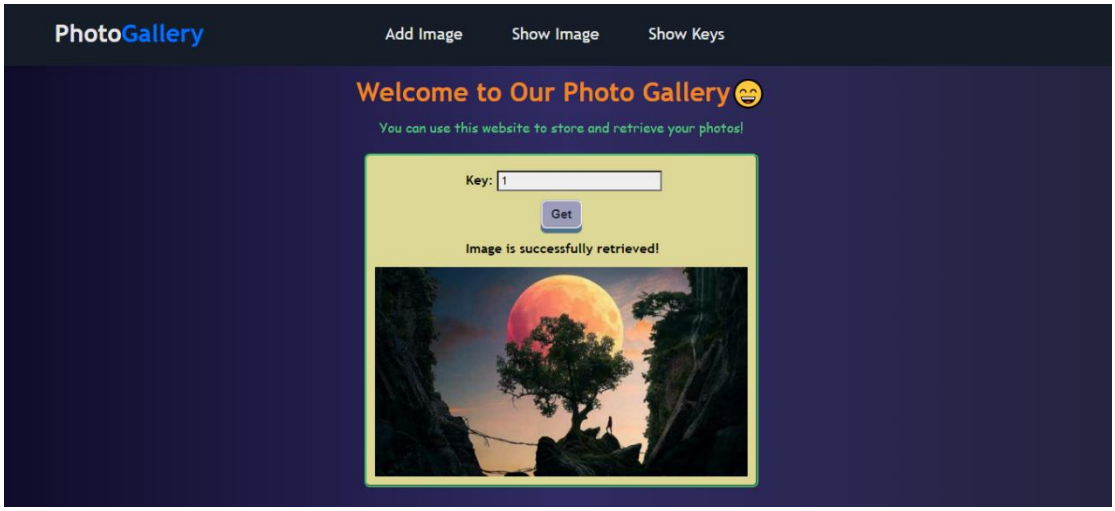
5. Now, we are updating the content of key **1** from **photo1.jpg** to **photo3.jpg**



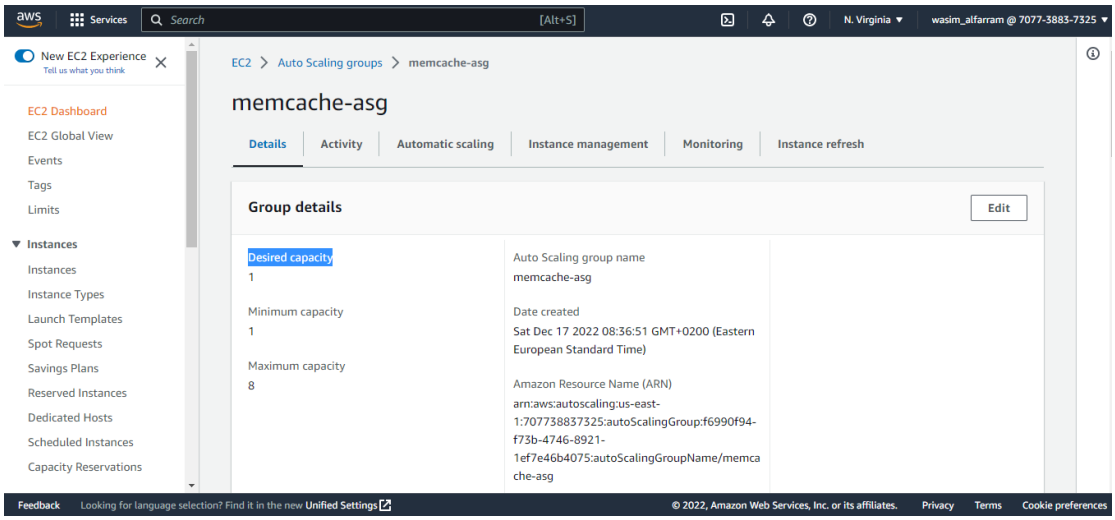
6. We can notice that **photo1.jpg** is replaced by **photo3.jpg** in our S3 bucket.



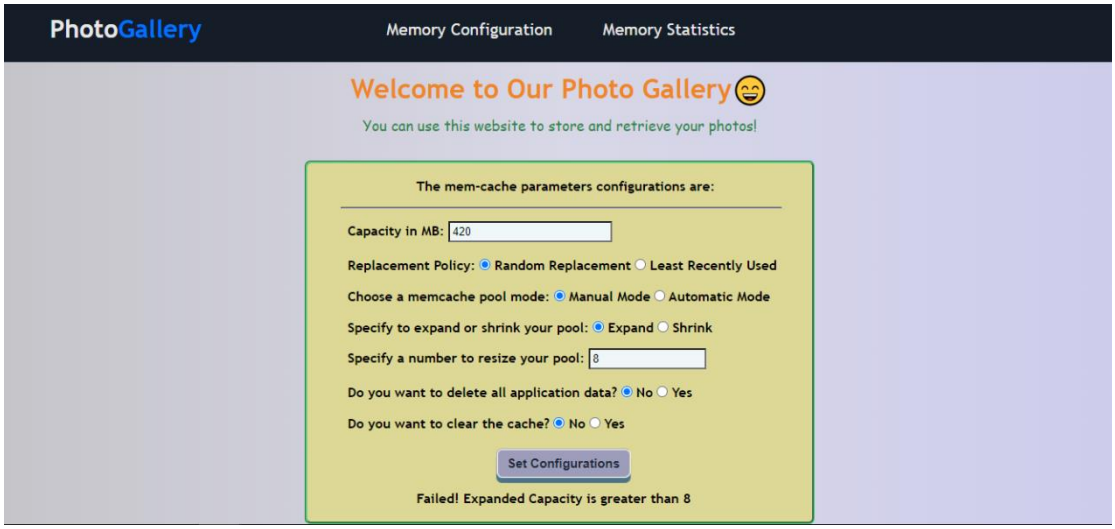
7. When showing the content of key **1** now, the photo is **photo3.jpg**



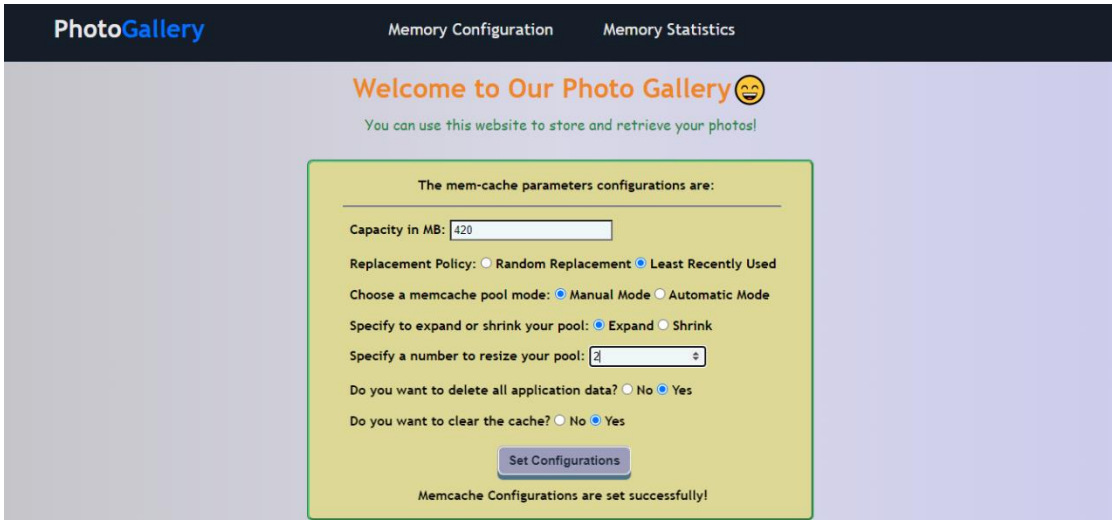
8. Now, for our pool, we currently have **1** node running.



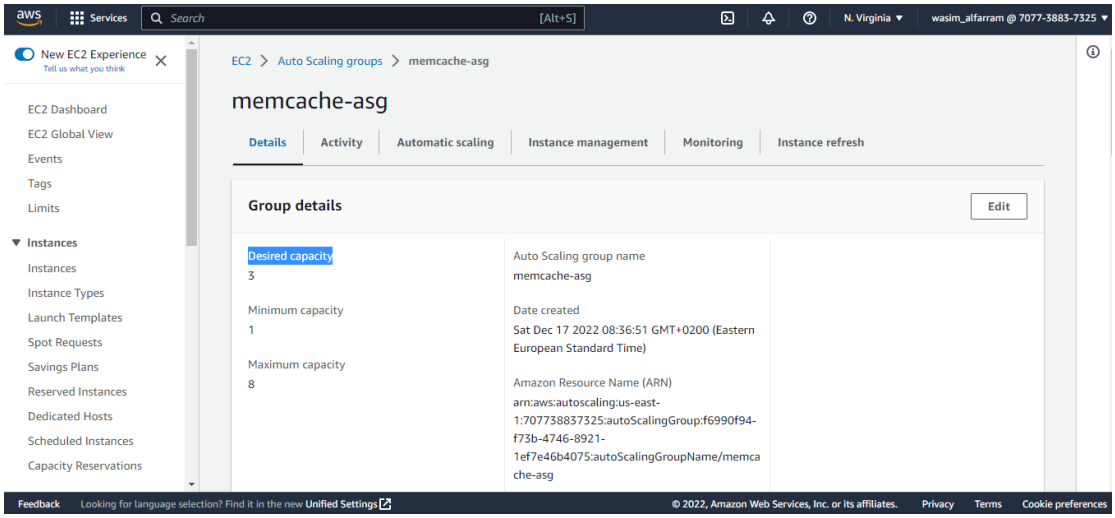
9. In the **Memory Configuration** page in our **manager-app**, we are trying to **expand** our pool by **8** so the total in the pool will be **9** which is not allowed to exceed **8 nodes** in the pool, so we got a caution messge as shown.



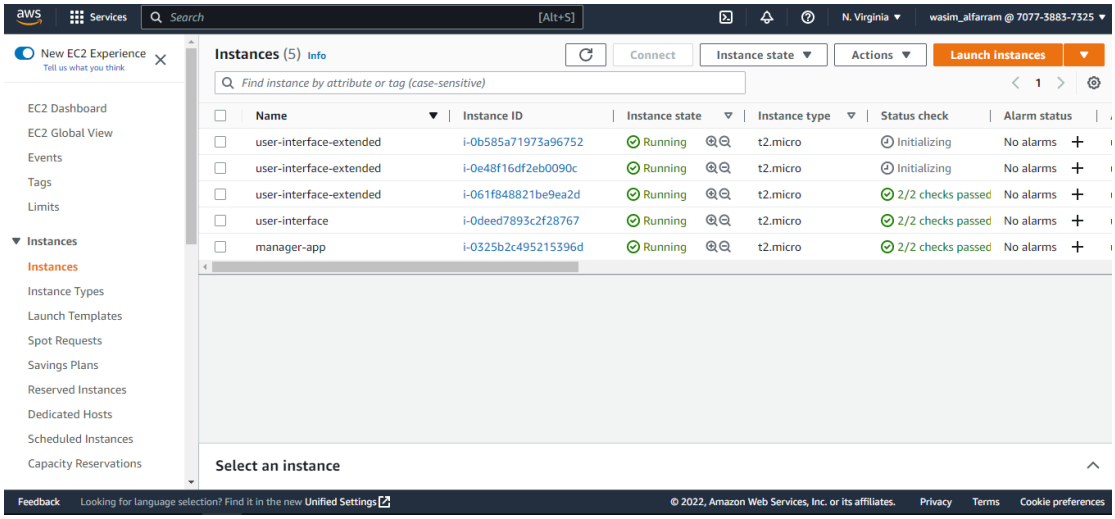
10. But when are trying to **expand** the pool by **2**, it will be expanded successfully so that we now have **3 running nodes** in our pool.



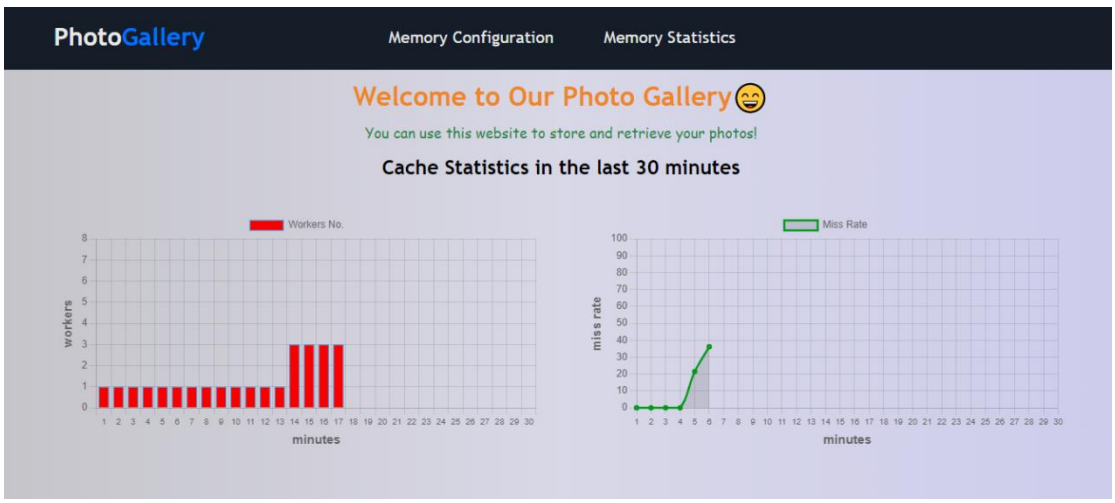
11. If we check the nodes in our pool, they will be **3 running nodes** exactly.



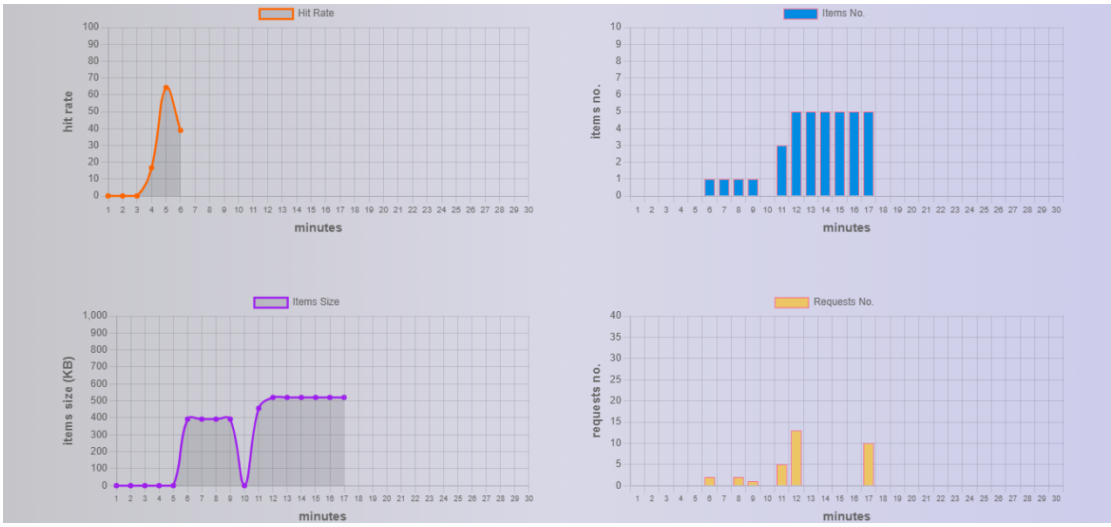
12. And here are **3 running nodes** in our pool called **user-inteface-extended**.



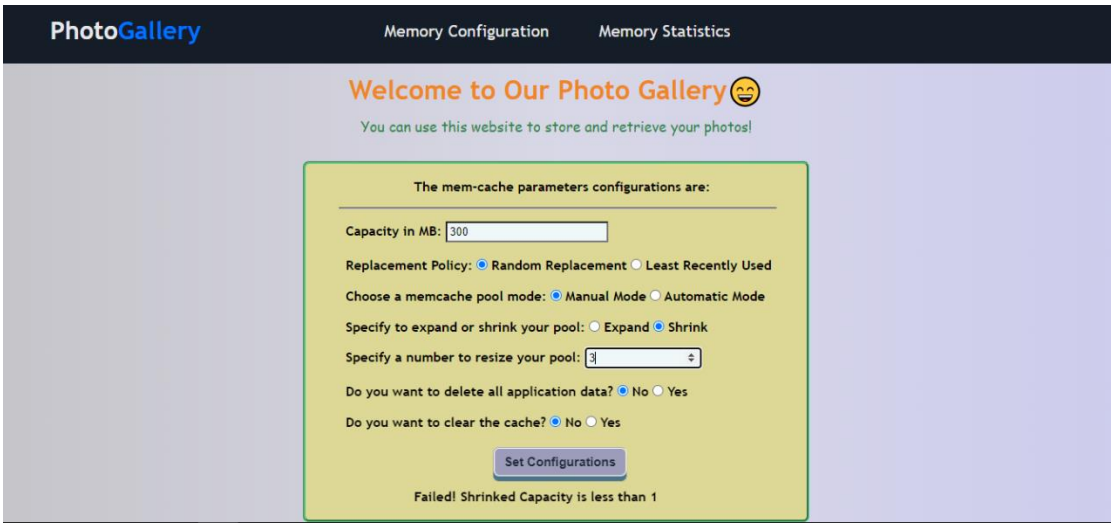
13. In the Memory Statistics page in our manger-app, we can see that the number of **running workers** in the pool is **3** now.



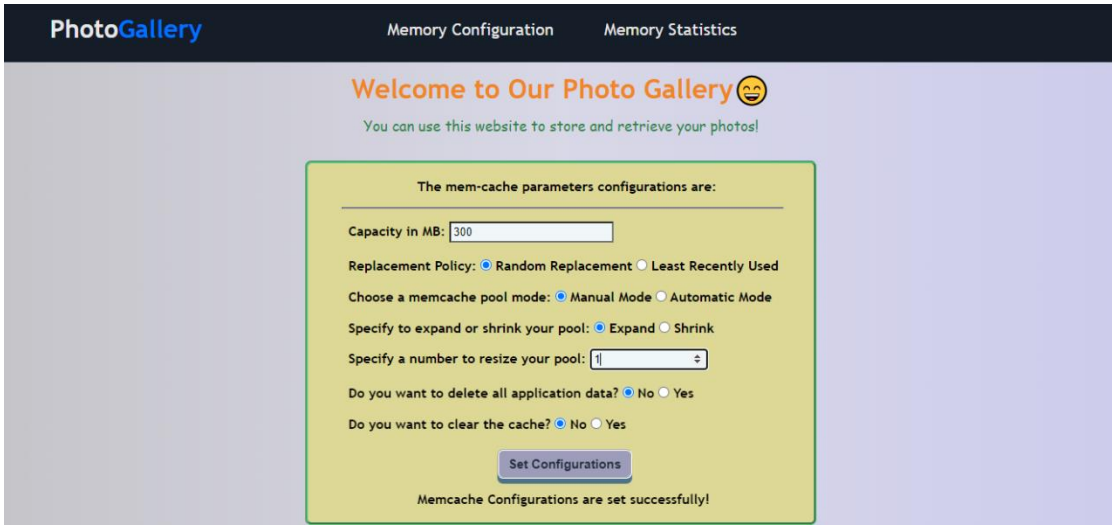
14. And here are the rest of the graphs for some photos I put into the gallery.



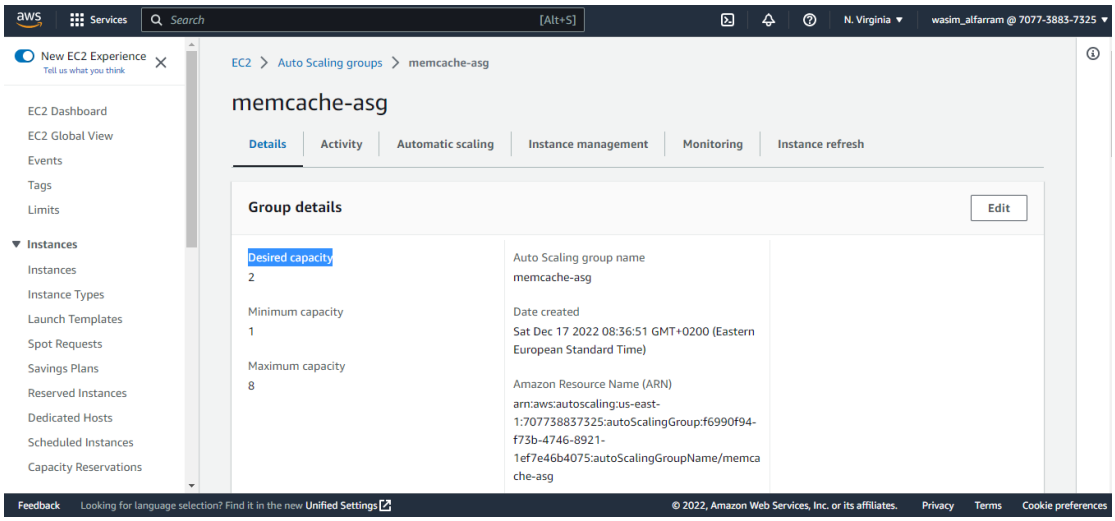
15. Now, I am trying to **shrink** the pool by **3** which will let our pool have a total of **0 nodes running** which is not allowed to have less than 1 running node in our pool. So, we got another caution message as shown.



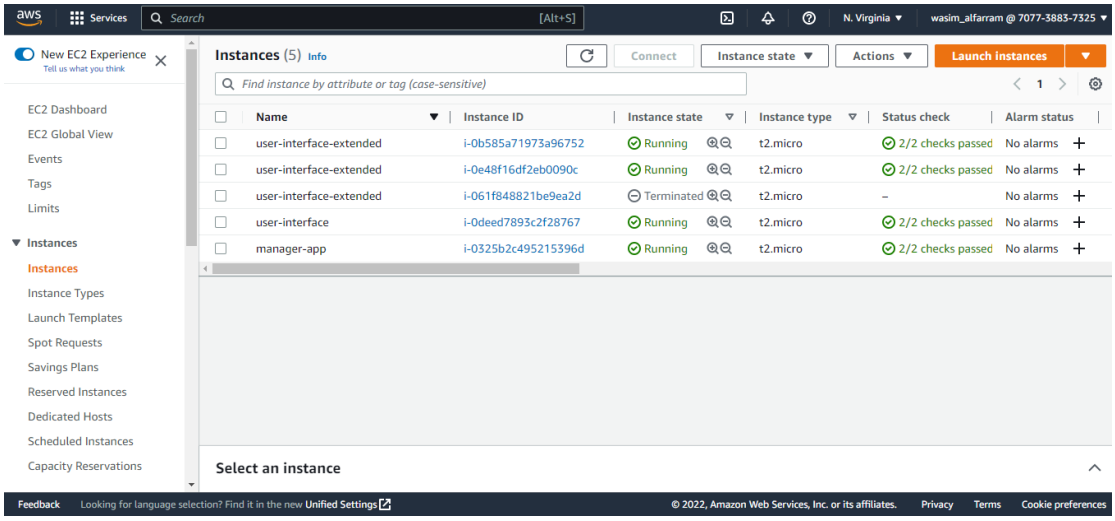
16. Now, I tried to **shrink** my pool by **1** and it is successfully shrinked from **3** to **2 running nodes** in the pool.



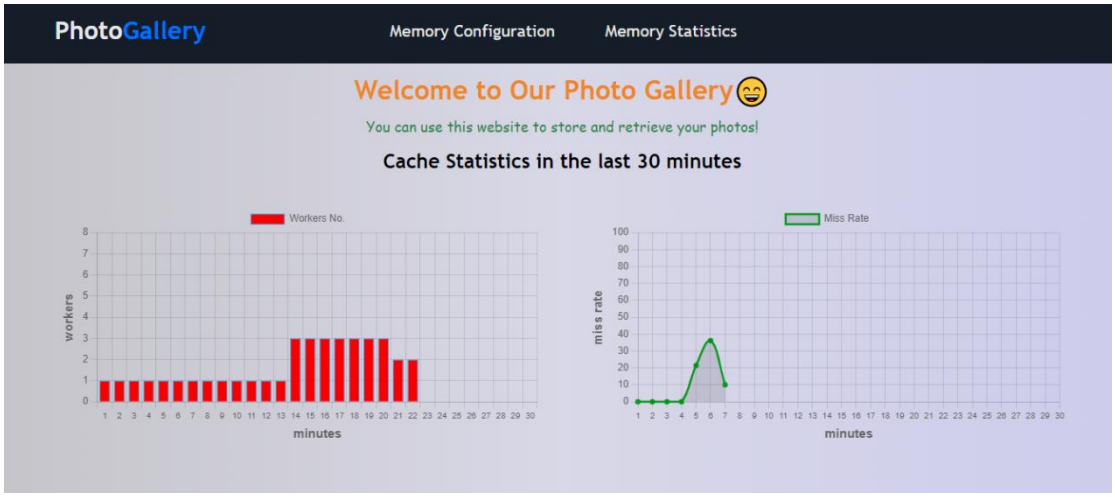
17. I am here checking that the current running nodes in my pool are **2 nodes**.



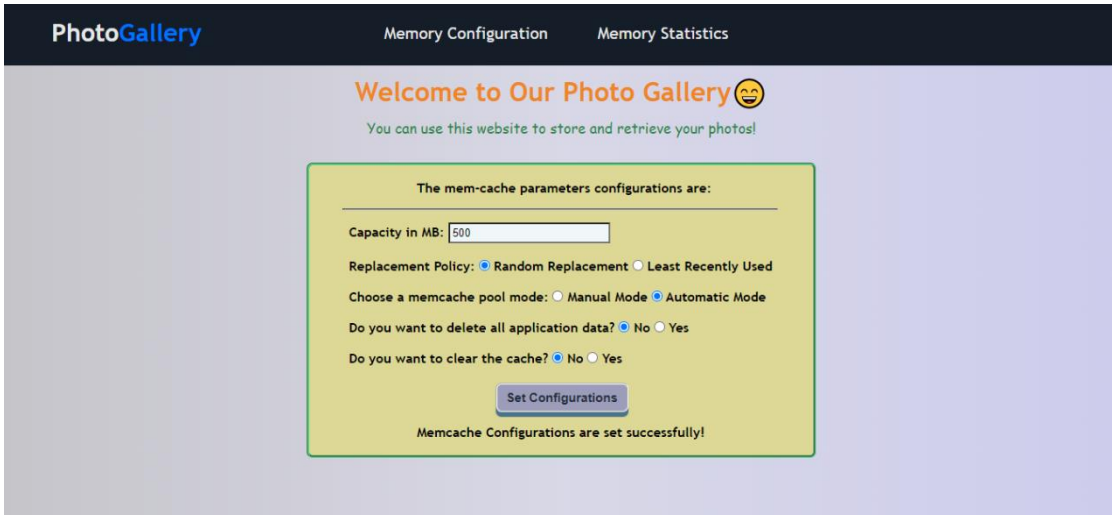
18. We can see here that the node we shrinked is terminated now.



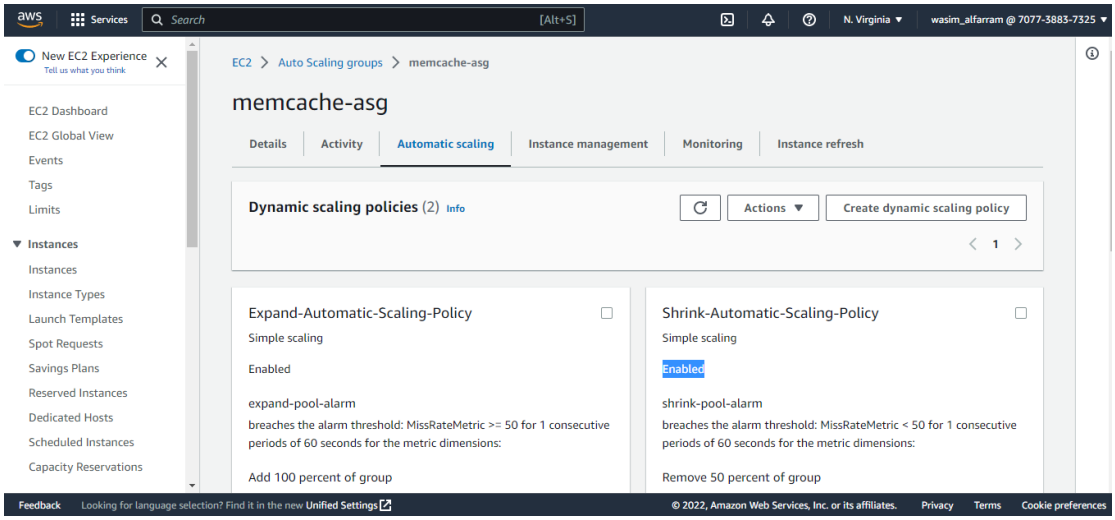
19. In the **Memory Statistics** page, we can see that the current running workers are **2 workers**.



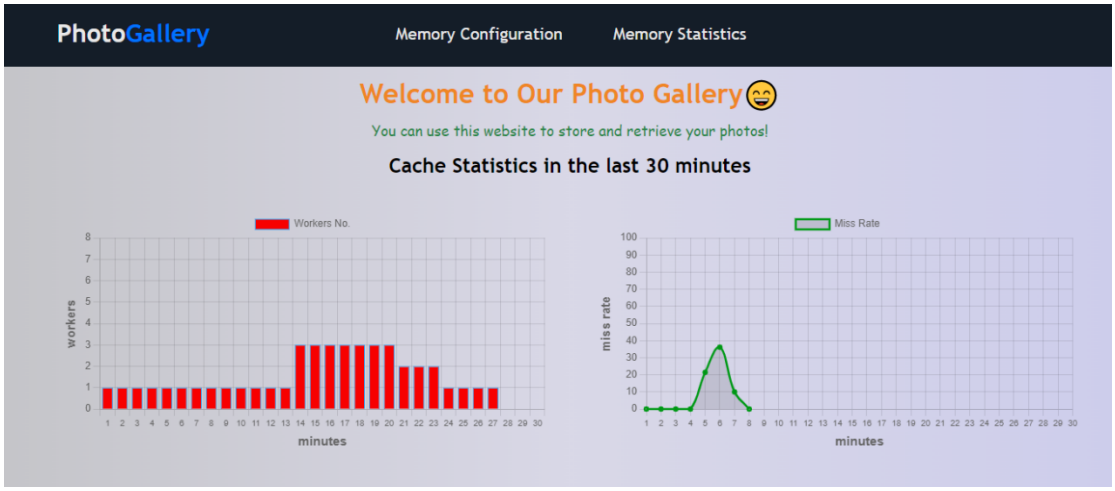
20. Now, we will check the Automatic Mode.



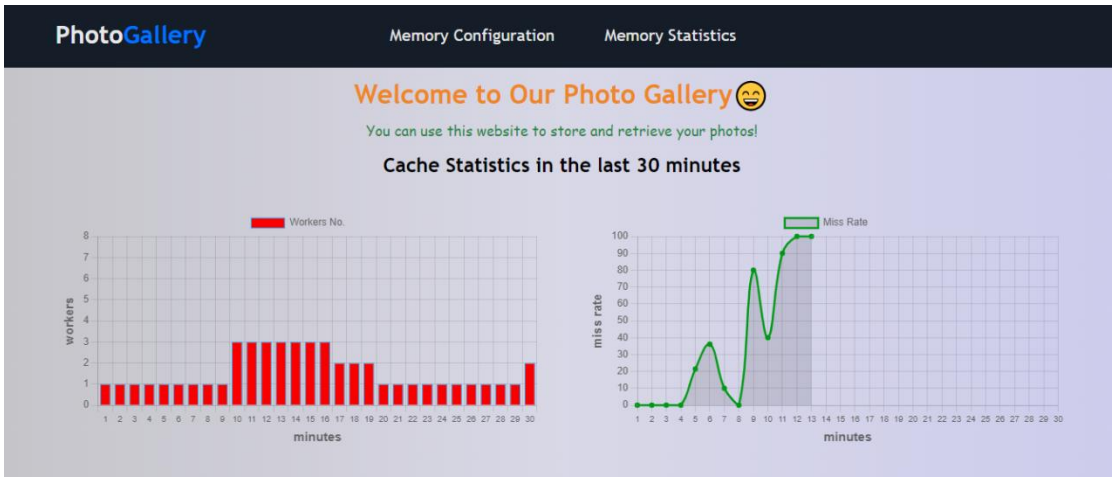
21. We can see here that after enabling the Automatic mode, **2 policies** for **Expanding** and **Shrinking** the pool are enabled.



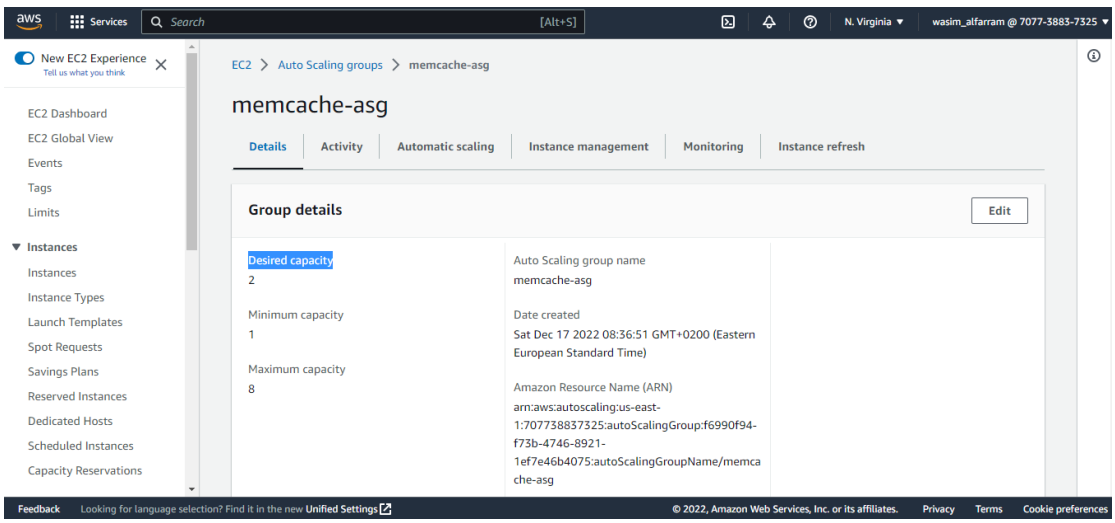
22. Since the miss rate is below 50%, the pool will be **automatically shrinked** from **2 workers** to **1 worker**.



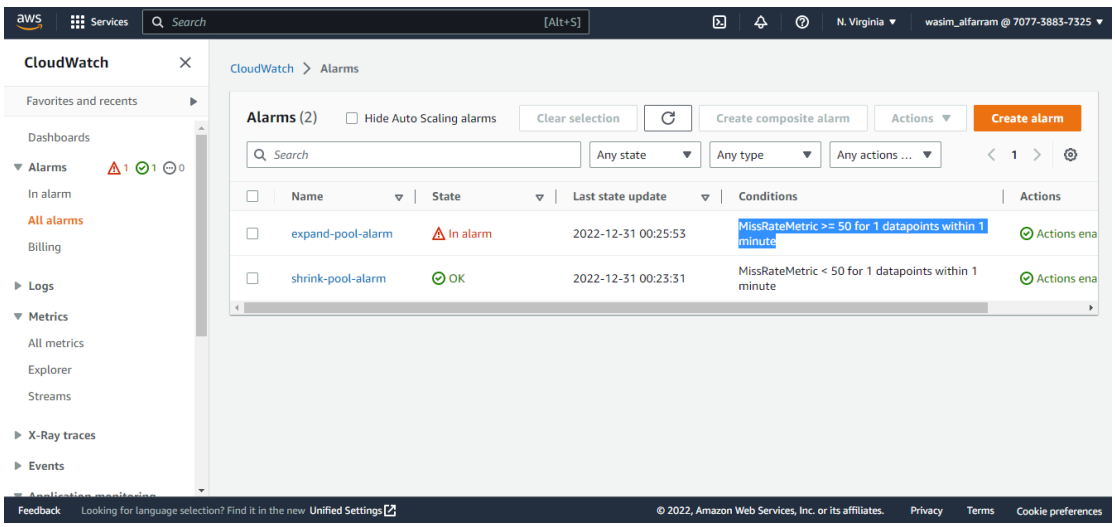
23. After having some action with too many miss rate, we can see the miss rate is now above 50% and the pool is **automatically expanded** from **1 worker** to **2 workers**.



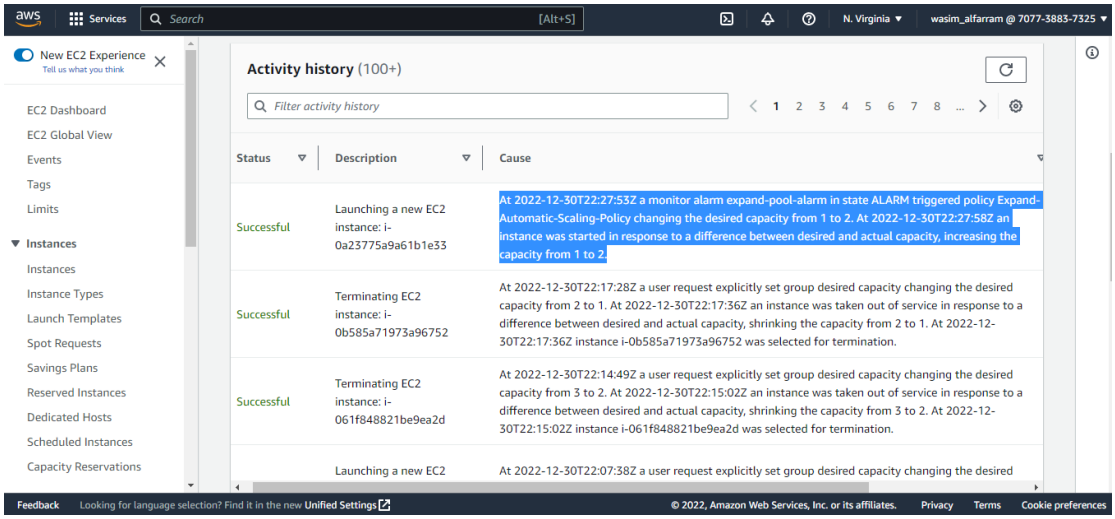
24. And here we are checking that the pool really has **2 nodes running**.



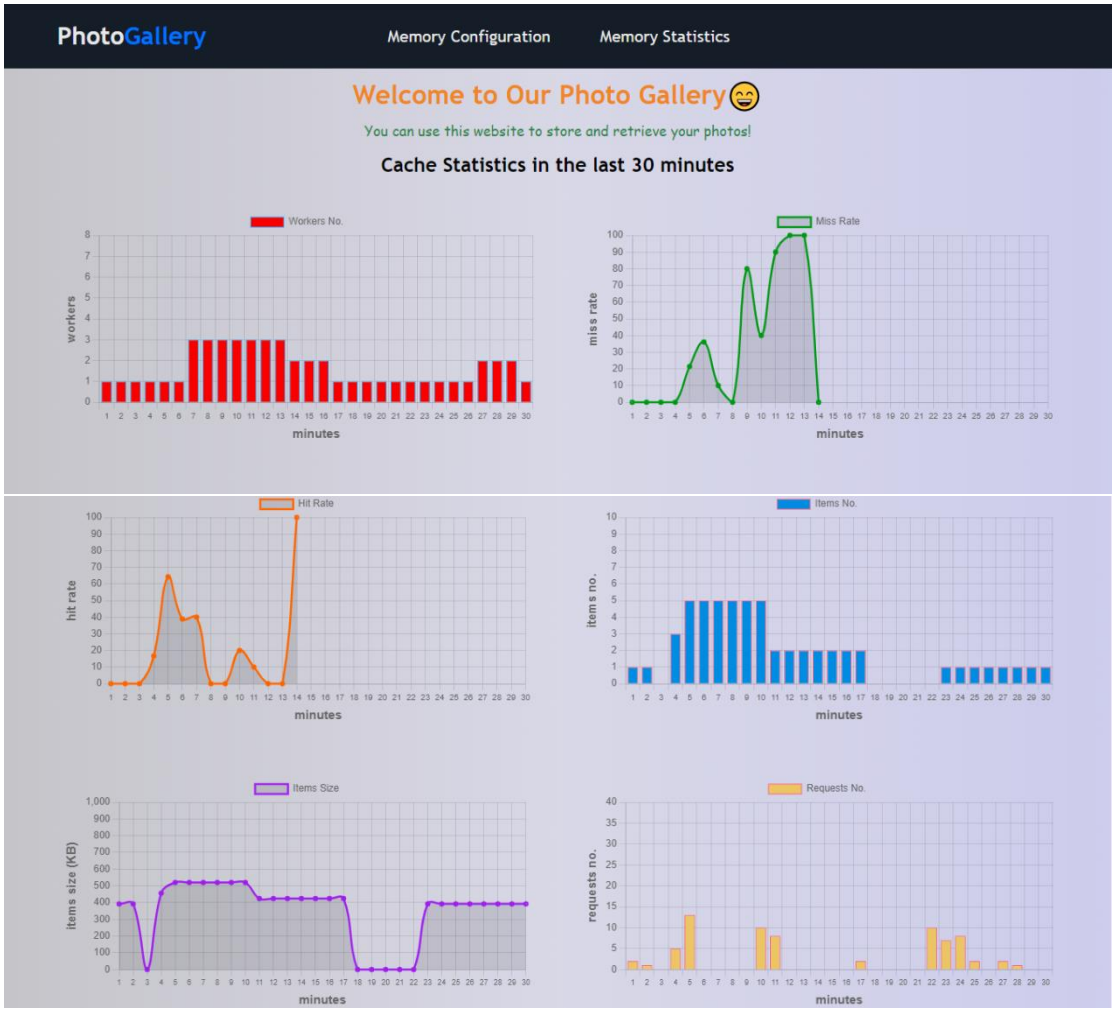
25. This is the **alarm** that notify me on email that my pool is **expanded to 2**



26. From the **Activity history** of my pool, I can see that the expansion has been done due to the **Expansion Policy** enabled before.

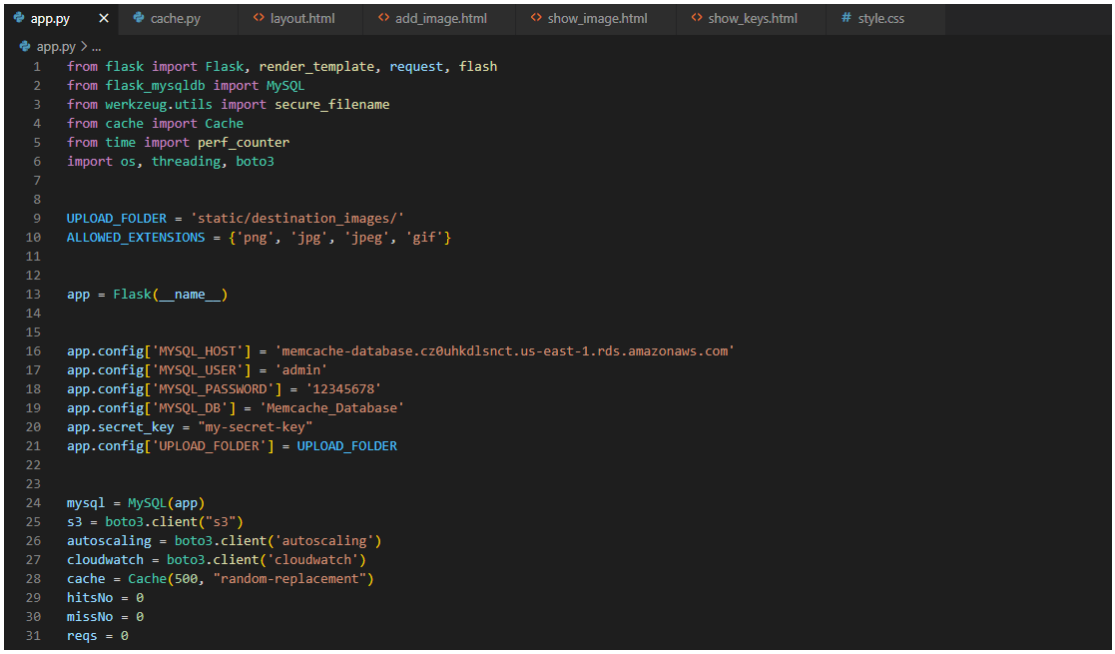


27. Now, we the miss rate is below 50%, the pool is **automatically shrinked** from **2 workers** to **1 worker**.



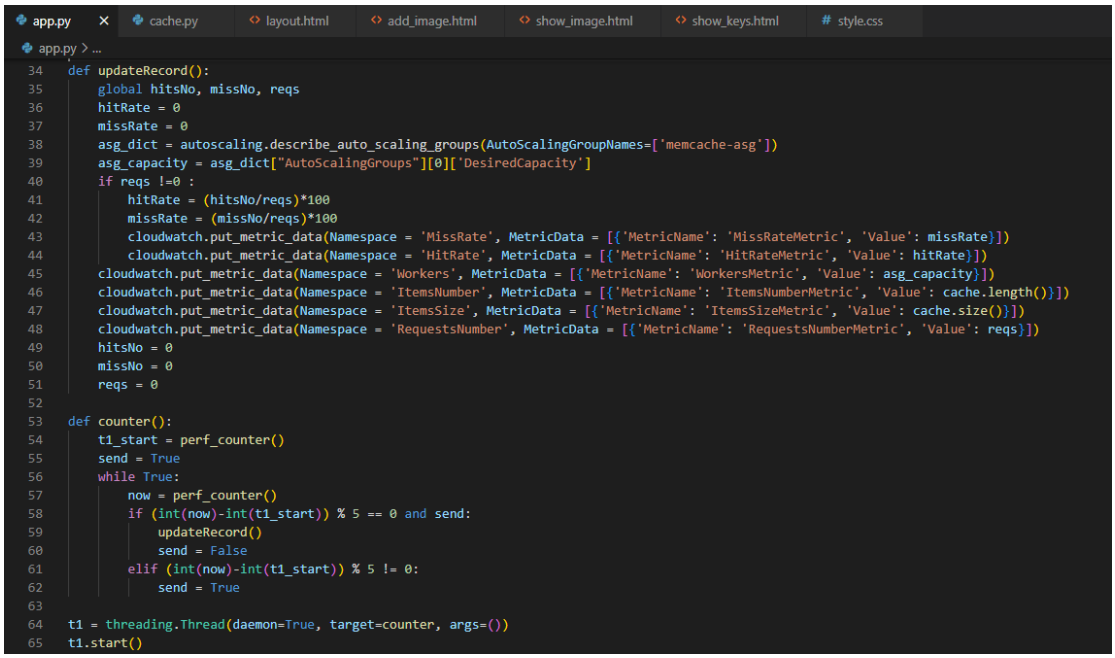
Codes explanation of the user-interface application:

1. This part of code imports the used libraries in the project, defining the allowed photos extensions, configuring the connection to the RDS database and making clients from the S3, Autoscaling Group, CloudWatch and the Cache.



```python
from flask import Flask, render_template, request, flash
from flask_mysqldb import MySQL
from werkzeug.utils import secure_filename
from cache import Cache
from time import perf_counter
import os, threading, boto3


UPLOAD_FOLDER = 'static/destination_images/'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}


app = Flask(__name__)


app.config['MYSQL_HOST'] = 'memcache-database.cz0uhkdlsnct.us-east-1.rds.amazonaws.com'
app.config['MYSQL_USER'] = 'admin'
app.config['MYSQL_PASSWORD'] = '12345678'
app.config['MYSQL_DB'] = 'Memcache_Database'
app.secret_key = "my-secret-key"
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


mysql = MySQL(app)
s3 = boto3.client("s3")
autoscaling = boto3.client('autoscaling')
cloudwatch = boto3.client('cloudwatch')
cache = Cache(500, "random-replacement")
hitsNo = 0
missNo = 0
reqs = 0
```

2. This part of code updates the statistics data of miss rate, hit rate, number of workers, items number in cache, items size in cache and requests number in their corresponding CloudWatch custom metrics every 5 seconds using threading.



```python
def updateRecord():
    global hitsNo, missNo, reqs
    hitRate = 0
    missRate = 0
    asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
    asg_capacity = asg_dict["AutoScalingGroups"][0]['DesiredCapacity']
    if reqs !=0 :
        hitRate = (hitsNo/reqs)*100
        missRate = (missNo/reqs)*100
        cloudwatch.put_metric_data(Namespace = 'MissRate', MetricData = [{'MetricName': 'MissRateMetric', 'Value': missRate}])
        cloudwatch.put_metric_data(Namespace = 'HitRate', MetricData = [{'MetricName': 'HitRateMetric', 'Value': hitRate}])
    cloudwatch.put_metric_data(Namespace = 'Workers', MetricData = [{'MetricName': 'WorkersMetric', 'Value': asg_capacity}])
    cloudwatch.put_metric_data(Namespace = 'ItemsNumber', MetricData = [{'MetricName': 'ItemsNumberMetric', 'Value': cache.length()}])
    cloudwatch.put_metric_data(Namespace = 'ItemsSize', MetricData = [{'MetricName': 'ItemsSizeMetric', 'Value': cache.size()}])
    cloudwatch.put_metric_data(Namespace = 'RequestsNumber', MetricData = [{'MetricName': 'RequestsNumberMetric', 'Value': reqs}])
    hitsNo = 0
    missNo = 0
    reqs = 0

def counter():
    t1_start = perf_counter()
    send = True
    while True:
        now = perf_counter()
        if (int(now)-int(t1_start)) % 5 == 0 and send:
            updateRecord()
            send = False
        elif (int(now)-int(t1_start)) % 5 != 0:
            send = True

t1 = threading.Thread(daemon=True, target=counter, args=())
t1.start()
```

3. This part of code defines a function that checks if the photo extension uploaded is in the extensions allowed or not.

```
70  def allowed_file(filename):
71      return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

4. This part of code is for the **Add Image** page. First, we establish a connection to the RDS then we check if the Delete all Application Data option is chosen or not by checking the number of keys in the database. After that, we save the key and the photo entered and manipulate them by making some checks if the key is previously entered so that it just needs to be updated, else add the new key and image to the database and upload them to the S3 bucket. Last, commit and close the RDS connection.

```
74  @app.route("/", methods=['POST', 'GET'])
75  @app.route("/add_image/", methods=['POST', 'GET'])
76  def add_image():
77      global cache
78      cursor = mysql.connection.cursor()
79      cursor.execute("SELECT image_key FROM image")
80      keys = cursor.fetchall()
81      if len(keys) == 0:
82          list = os.listdir('static/destination_images/')
83          for file in list:
84              if os.path.exists('static/destination_images/'):
85                  os.remove('static/destination_images/' + file)
86      cursor.execute("SELECT capacity FROM memory_configuration WHERE seq = 1")
87      capacity = int(cursor.fetchone()[0])
88      cursor.execute("SELECT replacement_policy FROM memory_configuration WHERE seq = 1")
89      replacement_policy = cursor.fetchone()
90      cache = Cache(capacity, replacement_policy)
91      if request.method == 'POST':
92          my_key = request.form['key']
93          name = request.files['name']
94          if name and allowed_file(name.filename):
95              filename = secure_filename(name.filename)
96              cursor.execute("SELECT image_key FROM image")
97              keys = cursor.fetchall()
98              key_exist = 'false'
99              for key in keys:
100                 if int(my_key) == int(key[0]):
101                     key_exist = 'true'
102                     break
103             if key_exist == 'false':
104                 name.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
105                 img_size = os.path.getsize(os.path.join(app.config['UPLOAD_FOLDER'], filename))
106                 s3.upload_file(Filename="static/destination_images/"+filename, Bucket="memcache-cloud-bucket", Key=filename)
107                 cursor.execute("INSERT INTO image(image_key, image_name, size) VALUES(%s, %s, %s)", (my_key, name.filename, img_size))
108                 flash('Image is successfully uploaded!')
109             else:
110                 if my_key in cache.data:
111                     cache.invalidateKey(my_key)
112                 cursor.execute("SELECT image_name FROM image WHERE image_key=%s", [my_key])
113                 filename2 = cursor.fetchone()
114                 s3.delete_object(Bucket="memcache-cloud-bucket", Key=filename2[0])
115                 os.unlink(os.path.join(app.config['UPLOAD_FOLDER'], filename2[0]))
116                 cursor.execute("DELETE FROM image WHERE image_key=%s", [my_key])
117                 name.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
118                 img_size = os.path.getsize(os.path.join(app.config['UPLOAD_FOLDER'], filename))
119                 cursor.execute("INSERT INTO image(image_key, image_name, size) VALUES(%s, %s, %s)", (my_key, name.filename, img_size))
120                 s3.upload_file(Filename="static/destination_images/"+filename, Bucket="memcache-cloud-bucket", Key=filename)
121                 flash('Image is successfully updated!')
122         else:
123             flash('(png, jpg, jpeg, gif) files only!')
124     mysql.connection.commit()
125     cursor.close()
126     return render_template("add_image.html")
```

5. This part of code is for the **Show Image** page. First, we establish an RDS connection and check if the Clear Cache option is enabled or not. If enabled, we clear the cache data. Then, we add the key and the photo entered and its encrypted path to the cache, increment the hit rate by 1 if the key and photo are existed previously, else increment the miss rate by one. Finally, we commit and close the RDS connection.

```python
@app.route("/show_image/", methods=['POST', 'GET'])
def show_image():
    global hitsNo, missNo, reqs
    reqs += 1
    img_src = '/static/temp.jpg'
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT clear_cache FROM memory_configuration WHERE seq = 1")
    clear_cache = cursor.fetchone()
    if clear_cache[0] == 'yes':
        cache.clear()
    cursor.execute("SELECT image_key FROM image")
    keys = cursor.fetchall()
    if len(keys) == 0:
        list = os.listdir('static/destination_images/')
        for file in list:
            if os.path.exists('static/destination_images/'):
                os.remove('static/destination_images/' + file)
    if request.method == 'POST':
        key = int(request.form['key'])
        cursor.execute("SELECT image_name FROM image WHERE image_key = %s", [key])
        file = cursor.fetchone()
        if key in cache.data:
            hitsNo += 1
            [src, ext] = cache.get(key)
            img_src = "" + src.decode()
        else:
            if file:
                filename = secure_filename(file[0])
                s3.download_file(Bucket="memcache-cloud-bucket", Key=filename, Filename="static/destination_images/"+filename)
                cache.put(int(key), filename)
                img_src = '/' + os.path.join(app.config['UPLOAD_FOLDER'], filename)
                flash('Image is successfully retrieved!')
            else:
                missNo += 1
    mysql.connection.commit()
    cursor.close()
    return render_template("show_image.html", image_src = img_src)
```

6. This part of code is for the **Show Keys** page. We establish an RDS connection, checking if the Delete all Application Data option is enabled or not. If not, we show all the keys stored in the database.

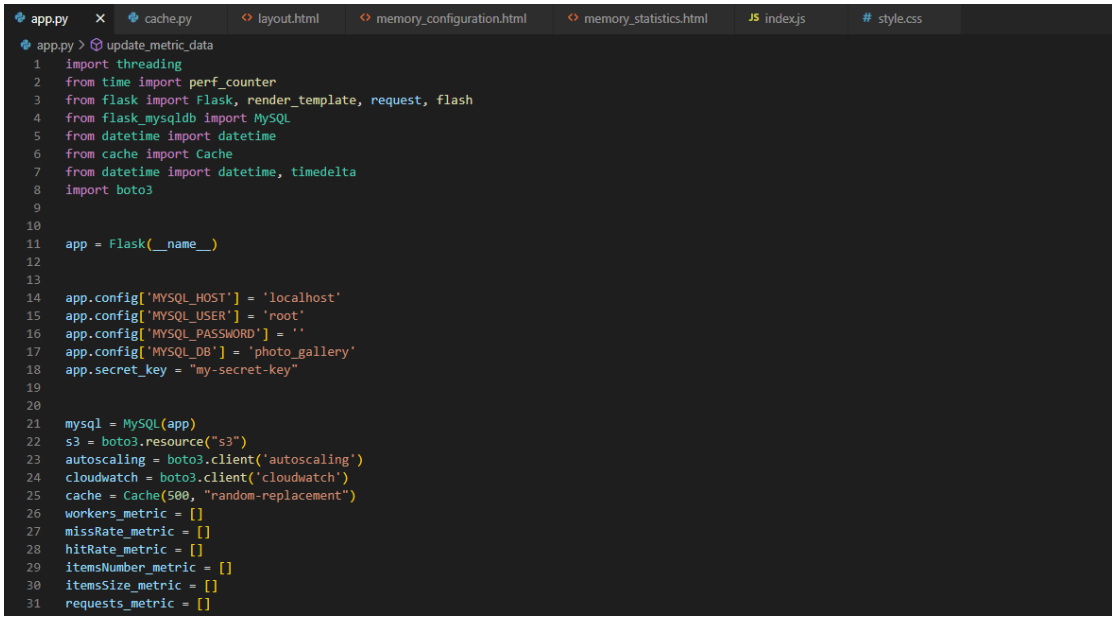
app.py | cache.py | layout.html | add_image.html | show_image.html | show_keys.html | style.css

app.py > ...

```python
@app.route("/show_keys/", methods=['POST', 'GET'])
def show_keys():
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT image_key FROM image")
    keys = cursor.fetchall()
    if len(keys) == 0:
        list = os.listdir('static/destination_images/')
        for file in list:
            if os.path.exists('static/destination_images/'):
                os.remove('static/destination_images/' + file)
    if request.method == 'GET':
        cursor.execute("SELECT image_key FROM image ORDER BY image_key ASC")
        keys = cursor.fetchall()
        for key in keys:
            flash(key[0])
    mysql.connection.commit()
    cursor.close()
    return render_template("show_keys.html")


app.run(host='localhost', port=5000, debug=True)
```

Codes explanation of the manager-app application:

1. This part of code imports the used libraries in the project, configuring the connection to the RDS database and making clients from the S3, Autoscaling Group, CloudWatch and the Cache. Finally, defining empty lists to store the retrieved statistics data in.



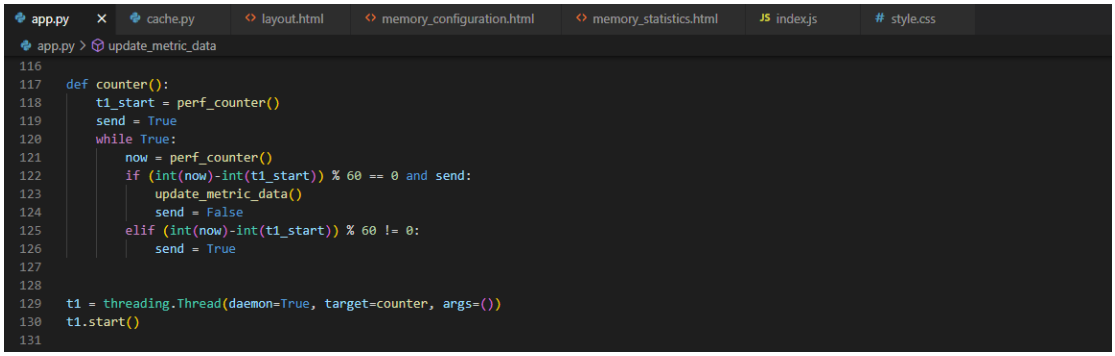
2. This part of code gets the statistics data from the CloudWatch custom metrices every 1 minute and adds them to their corresponding lists defined in the code above and make sure that the length of the lists does not exceeds 30 data points, if so, then eliminate the first data point in the list and add the new data point to the end of the list.

app.py | cache.py | layout.html | memory_configuration.html | memory_statistics.html | index.js | style.css

app.py > ...

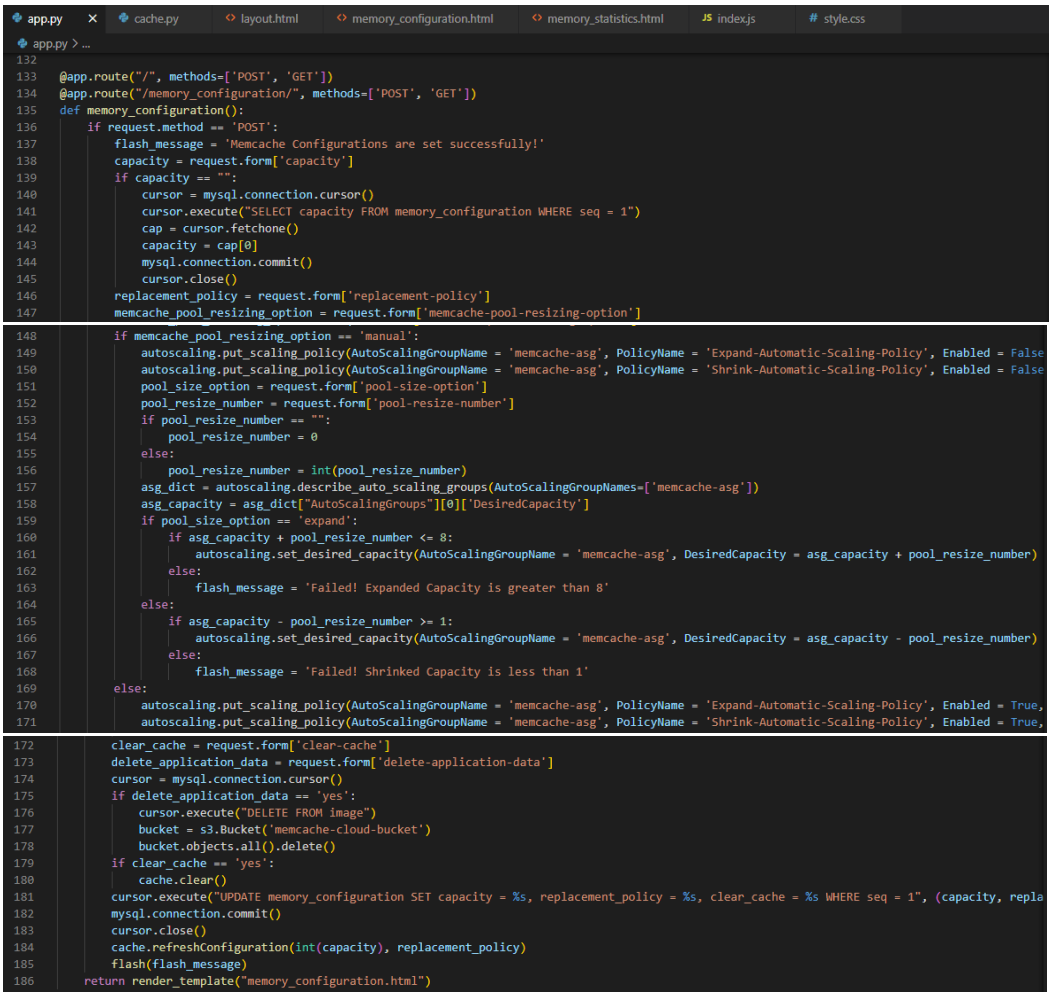
```python
def update_metric_data():
    start_time = datetime.utcnow() - timedelta(seconds=60)
    end_time = datetime.utcnow()
    missRate_params = {
        'Namespace': 'MissRate',
        'MetricName': 'MissRateMetric',
        'StartTime': start_time,
        'EndTime': end_time,
        'Period': 60,
        'Statistics': ['Average']
    }
    hitRate_params = {
        'Namespace': 'HitRate',
        'MetricName': 'HitRateMetric',
        'StartTime': start_time,
        'EndTime': end_time,
        'Period': 60,
        'Statistics': ['Average']
    }
    itemsNumber_params = {
        'Namespace': 'ItemsNumber',
        'MetricName': 'ItemsNumberMetric',
        'StartTime': start_time,
        'EndTime': end_time,
        'Period': 60,
        'Statistics': ['Maximum']
    }
    itemsSize_params = {
        'Namespace': 'ItemsSize',
        'MetricName': 'ItemsSizeMetric',
        'StartTime': start_time,
        'EndTime': end_time,
        'Period': 60,
        'Statistics': ['Maximum']
    }
    requests_params = {
        'Namespace': 'RequestsNumber',
        'MetricName': 'RequestsNumberMetric',
        'StartTime': start_time,
        'EndTime': end_time,
        'Period': 60,
        'Statistics': ['Sum']
    }

    missRate_response = cloudwatch.get_metric_statistics(**missRate_params)
    hitRate_response = cloudwatch.get_metric_statistics(**hitRate_params)
    itemsNumber_response = cloudwatch.get_metric_statistics(**itemsNumber_params)
    itemsSize_response = cloudwatch.get_metric_statistics(**itemsSize_params)
    requests_response = cloudwatch.get_metric_statistics(**requests_params)

    if len(workers_metric) == 30:
        workers_metric.pop(0)
    if len(missRate_metric) == 30:
        missRate_metric.pop(0)
    if len(hitRate_metric) == 30:
        hitRate_metric.pop(0)
    if len(itemsNumber_metric) == 30:
        itemsNumber_metric.pop(0)
    if len(itemsSize_metric) == 30:
        itemsSize_metric.pop(0)
    if len(requests_metric) == 30:
        requests_metric.pop(0)

    asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
    workers_data_point = asg_dict["AutoScalingGroups"][0]['DesiredCapacity']
    workers_metric.append(workers_data_point)
    if len(missRate_response['Datapoints']) != 0:
        missRate_data_point = missRate_response['Datapoints'][0]['Average']
        missRate_metric.append(missRate_data_point)
    if len(hitRate_response['Datapoints']) != 0:
        hitRate_data_point = hitRate_response['Datapoints'][0]['Average']
        hitRate_metric.append(hitRate_data_point)
    if len(itemsNumber_response['Datapoints']) != 0:
        itemsNumber_data_point = itemsNumber_response['Datapoints'][0]['Maximum']
        itemsNumber_metric.append(itemsNumber_data_point)
    if len(itemsSize_response['Datapoints']) != 0:
        itemsSize_data_point = itemsSize_response['Datapoints'][0]['Maximum']
        itemsSize_metric.append(itemsSize_data_point)
    if len(requests_response['Datapoints']) != 0:
        requests_data_point = requests_response['Datapoints'][0]['Sum']
        requests_metric.append(requests_data_point)
```

3. This part of code repeats the previous step in the code every 1 minute using threading.



```python
def counter():
    t1_start = perf_counter()
    send = True
    while True:
        now = perf_counter()
        if (int(now)-int(t1_start)) % 60 == 0 and send:
            update_metric_data()
            send = False
        elif (int(now)-int(t1_start)) % 60 != 0:
            send = True


t1 = threading.Thread(daemon=True, target=counter, args=())
t1.start()
```

4. This part of code is for the **Memory Configuration** page. It allows the manger to control the capacity of the cache in MB, the policy used in the cache when it is full, Random Replacement policy or Least Recently Used policy. Also, it allows the manager to manually or automatically expand or shrink the nodes in the pool within the allowed range (1 minimum and 8 maximum). Moreover, it allows the manager to delete all application data by deleting all photos and data stored in the database and S3. Finally, the manager also can delete the cache data in all nodes.



```python
@app.route("/", methods=['POST', 'GET'])
@app.route("/memory_configuration/", methods=['POST', 'GET'])
def memory_configuration():
    if request.method == 'POST':
        flash_message = 'Memcache Configurations are set successfully!'
        capacity = request.form['capacity']
        if capacity == "":
            cursor = mysql.connection.cursor()
            cursor.execute("SELECT capacity FROM memory_configuration WHERE seq = 1")
            cap = cursor.fetchone()
            capacity = cap[0]
            mysql.connection.commit()
            cursor.close()
        replacement_policy = request.form['replacement-policy']
        memcache_pool_resizing_option = request.form['memcache-pool-resizing-option']
        if memcache_pool_resizing_option == 'manual':
            autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Expand-Automatic-Scaling-Policy', Enabled = False
            autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Shrink-Automatic-Scaling-Policy', Enabled = False
            pool_size_option = request.form['pool-size-option']
            pool_resize_number = request.form['pool-resize-number']
            if pool_resize_number == "":
                pool_resize_number = 0
            else:
                pool_resize_number = int(pool_resize_number)
            asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
            asg_capacity = asg_dict["AutoScalingGroups"][0]['DesiredCapacity']
            if pool_size_option == 'expand':
                if asg_capacity + pool_resize_number <= 8:
                    autoscaling.set_desired_capacity(AutoScalingGroupName = 'memcache-asg', DesiredCapacity = asg_capacity + pool_resize_number)
                else:
                    flash_message = 'Failed! Expanded Capacity is greater than 8'
            else:
                if asg_capacity - pool_resize_number >= 1:
                    autoscaling.set_desired_capacity(AutoScalingGroupName = 'memcache-asg', DesiredCapacity = asg_capacity - pool_resize_number)
                else:
                    flash_message = 'Failed! Shrinked Capacity is less than 1'
        else:
            autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Expand-Automatic-Scaling-Policy', Enabled = True,
            autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Shrink-Automatic-Scaling-Policy', Enabled = True,
        clear_cache = request.form['clear-cache']
        delete_application_data = request.form['delete-application-data']
        cursor = mysql.connection.cursor()
        if delete_application_data == 'yes':
            cursor.execute("DELETE FROM image")
            bucket = s3.Bucket('memcache-cloud-bucket')
            bucket.objects.all().delete()
        if clear_cache == 'yes':
            cache.clear()
        cursor.execute("UPDATE memory_configuration SET capacity = %s, replacement_policy = %s, clear_cache = %s WHERE seq = 1", (capacity, repla
        mysql.connection.commit()
        cursor.close()
        cache.refreshConfiguration(int(capacity), replacement_policy)
        flash(flash_message)
    return render_template("memory_configuration.html")
```

5. This part of code is for the **Memory Statistics** page. It sends the lists of the data points retrieved from the custom metrices of the CloudWatch to the front-end code to show the data points on the graphs.

```
@app.route("/memory_statistics/")
def memory_statistics():
    global workers_metric, missRate_metric, hitRate_metric, itemsNumber_metric, itemsSize_metric, requests_metric
    return render_template("memory_statistics.html", workers = workers_metric, miss_rate = missRate_metric, hit_rate = hitRate_metric, items_numb


app.run(host='localhost', port=5000, debug=True)

```