



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

No.	Items	Page
1.	How to log in to the Amazon account, start the instance and run the codes of the project	3
2.	Workflow of the whole project	7
3.	Codes explanation of the user-interface application	18
4.	Codes explanation of the manager-app application	21

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

1. Go to [aws.amazon.com](https://aws.amazon.com), choose the IAM user and enter the account ID.



### Sign in

☐ Root user

Account owner that performs tasks requiring unrestricted access. [Learn more](#)

☒ IAM user

User within an account that performs daily tasks. [Learn more](#)

Account ID (12 digits) or account alias

707738837325

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

New to AWS?

Create a new AWS account



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



### Sign in as IAM user

Account ID (12 digits) or account alias

707738837325

IAM user name

wasim\_alfarram

Password

\*\*\*\*\*

☐ Remember this account

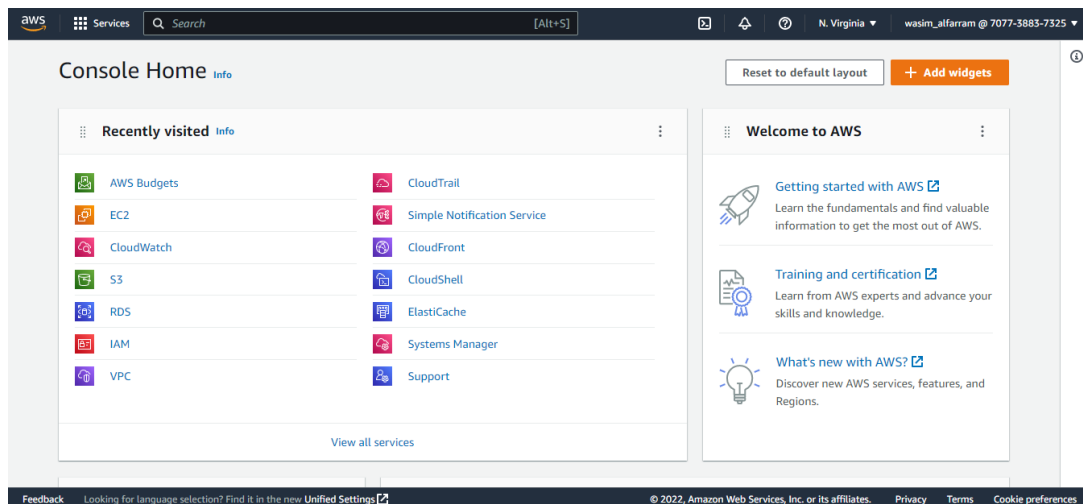
Sign in

[Sign in using root user email](#)

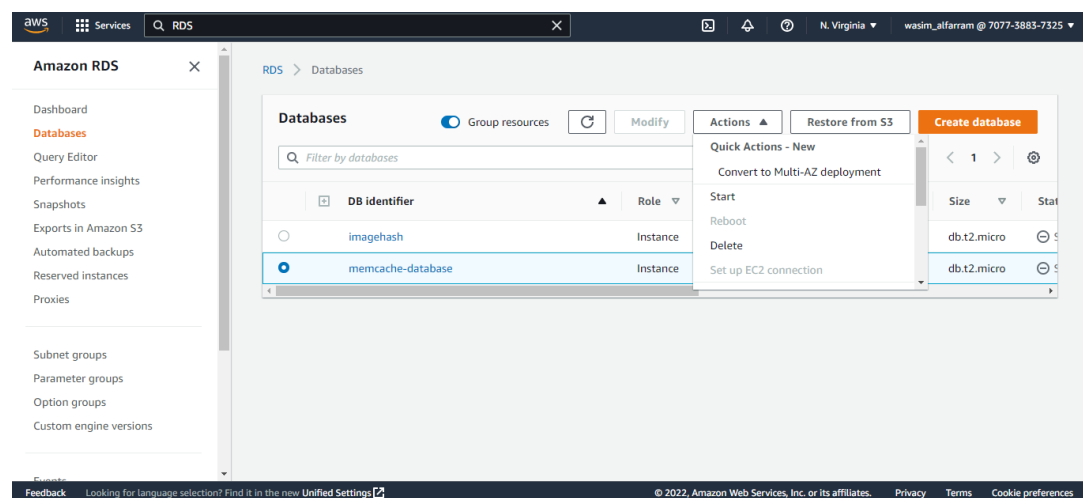
[Forgot password?](#)



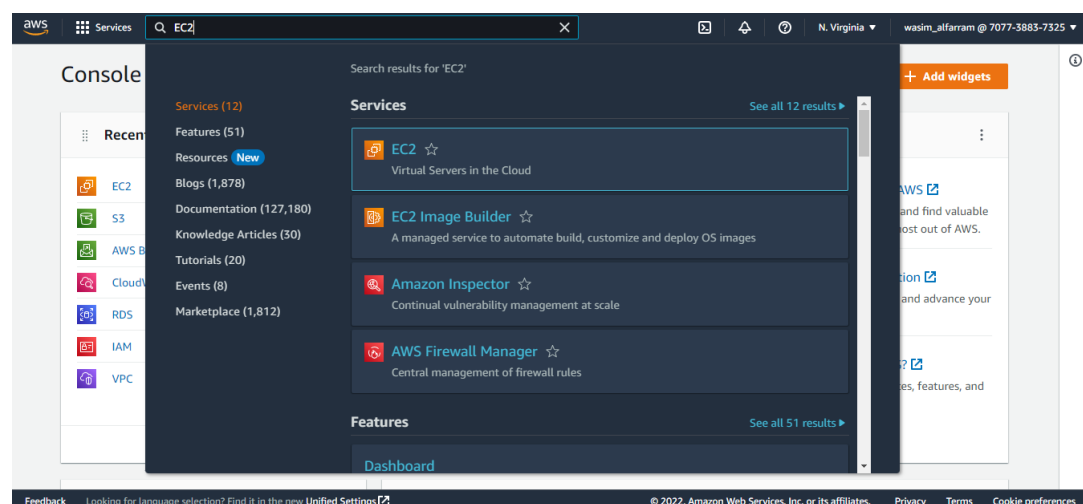
3. You will be redirected to a page like this.



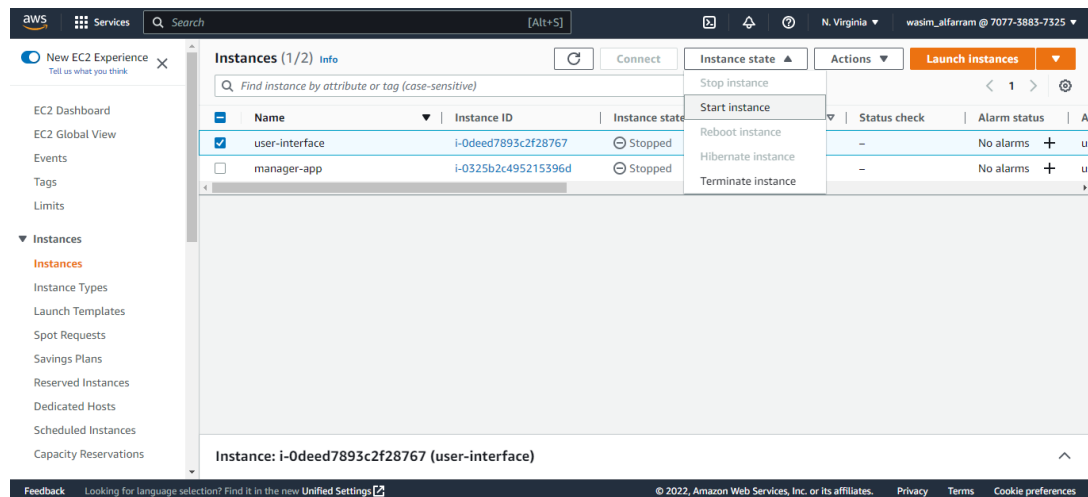
4. From the search bar above, enter **RDS** and **Start** your RDS from **Actions**.



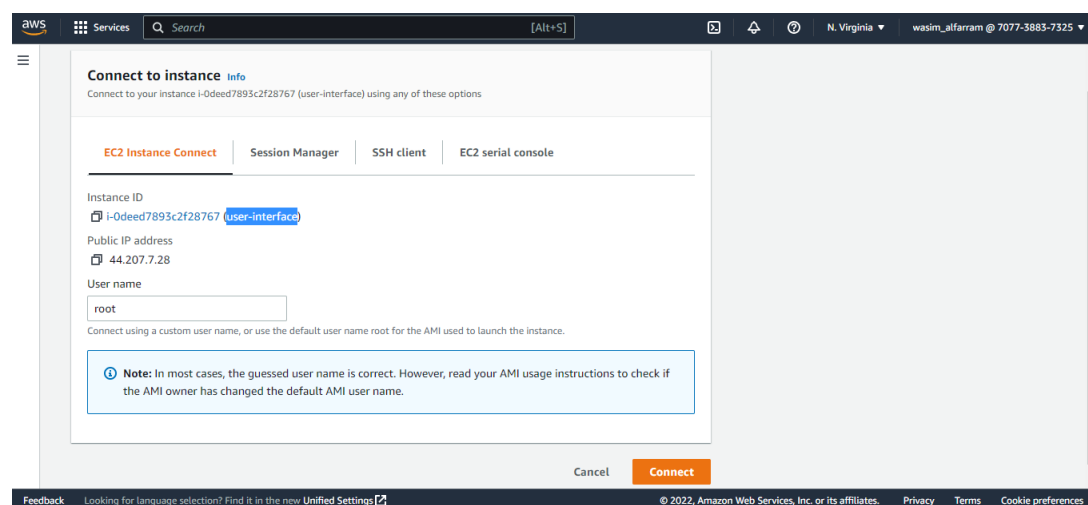
5. From the search bar above, enter **EC2** to go to your instances.



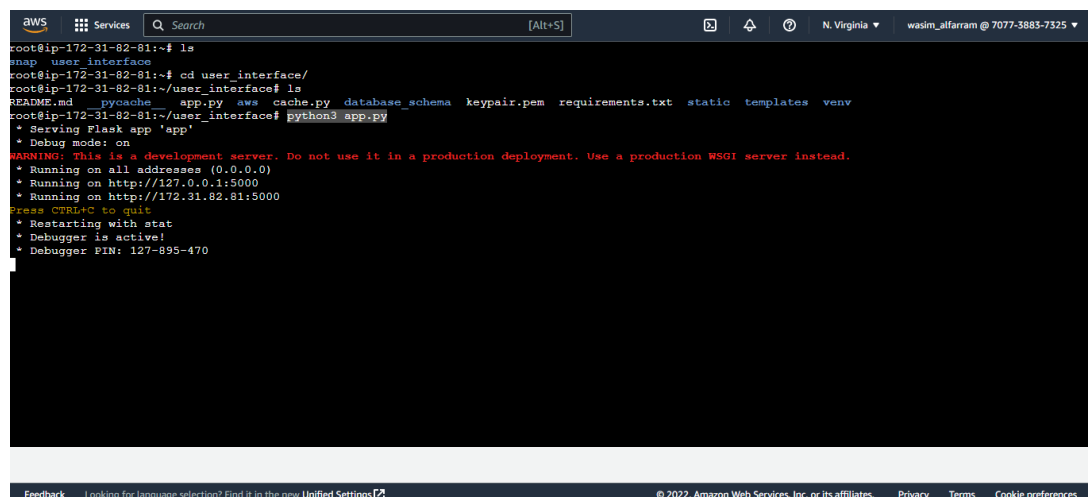
6. You will be redirected to this page, choose your EC2 instance and **Start** it.



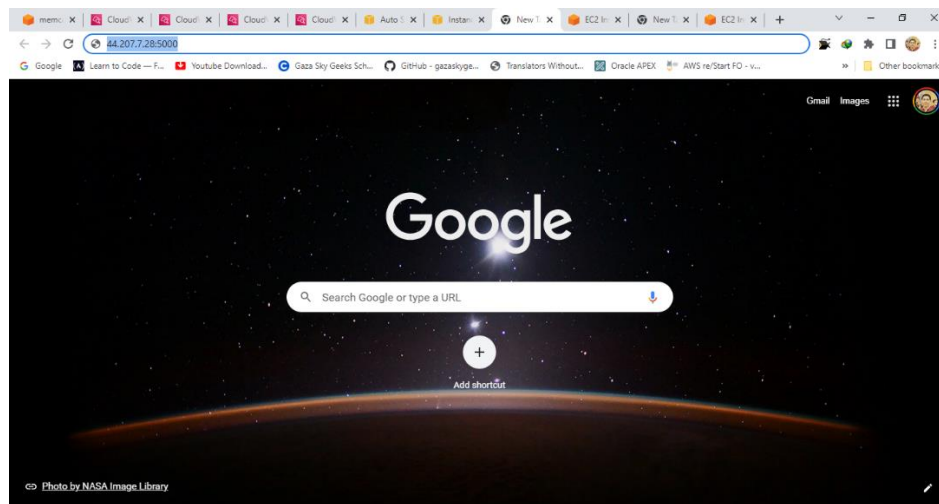
7. After it is running, **Connect** to it and copy the **Public IP** address of it.



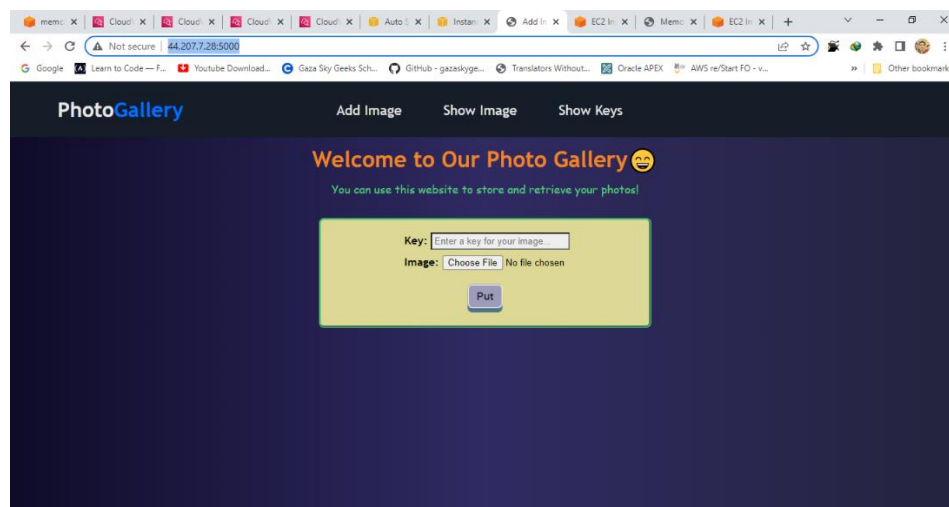
8. After connecting, you will have such a terminal, put the following commands to run the application.



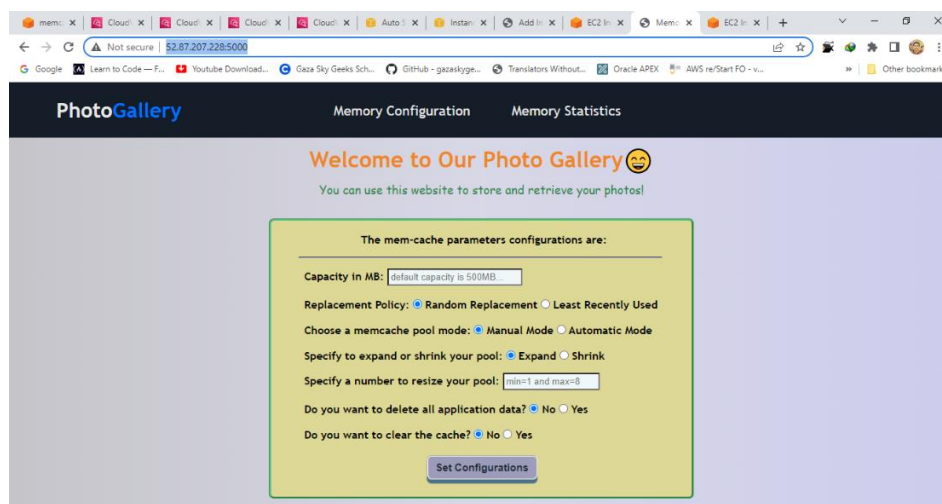
9. Paste the **Public IP** you copied before to the browser followed by **:5000**



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

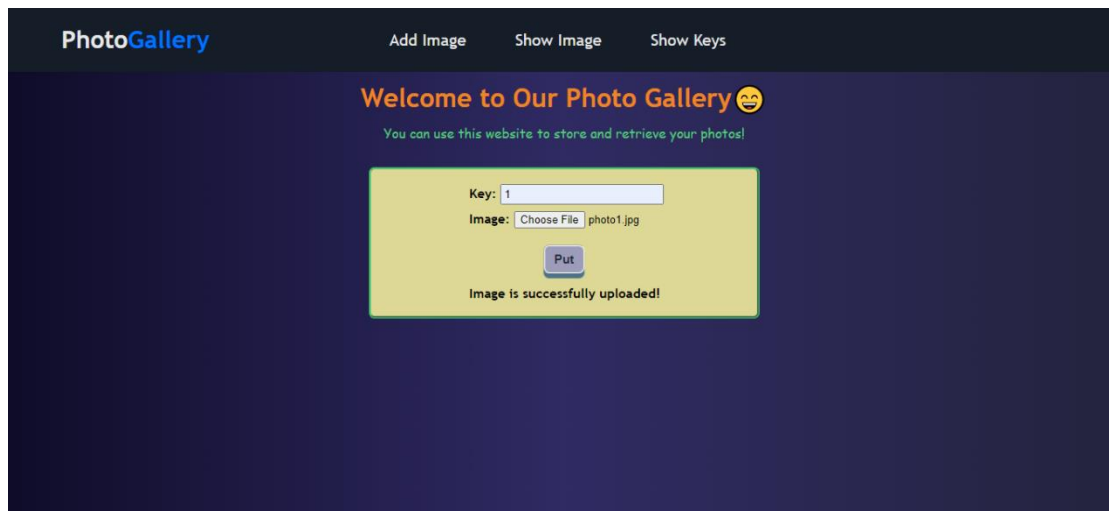


11. When running the **manager-app** application, you will have this page.

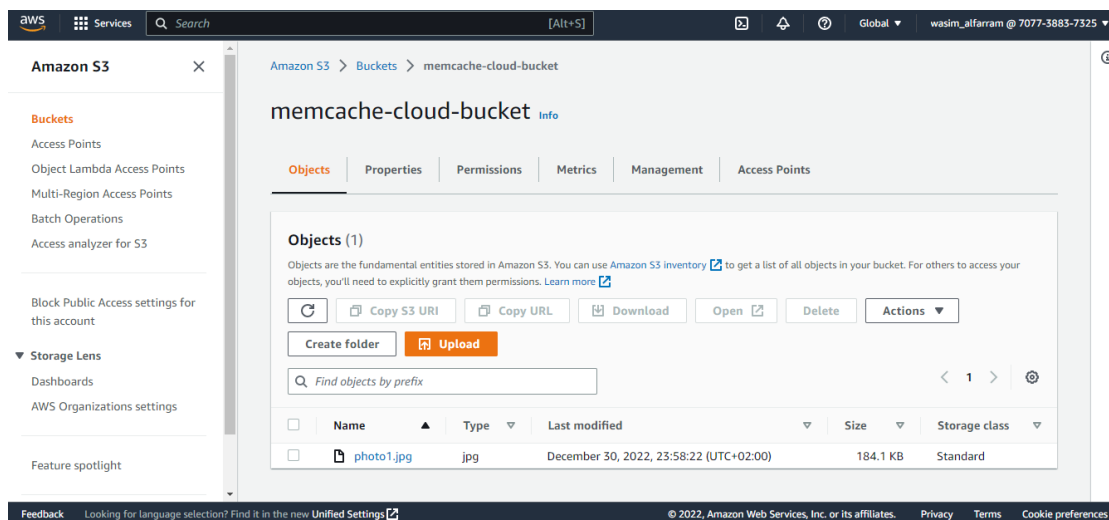


## Workflow of the whole project:

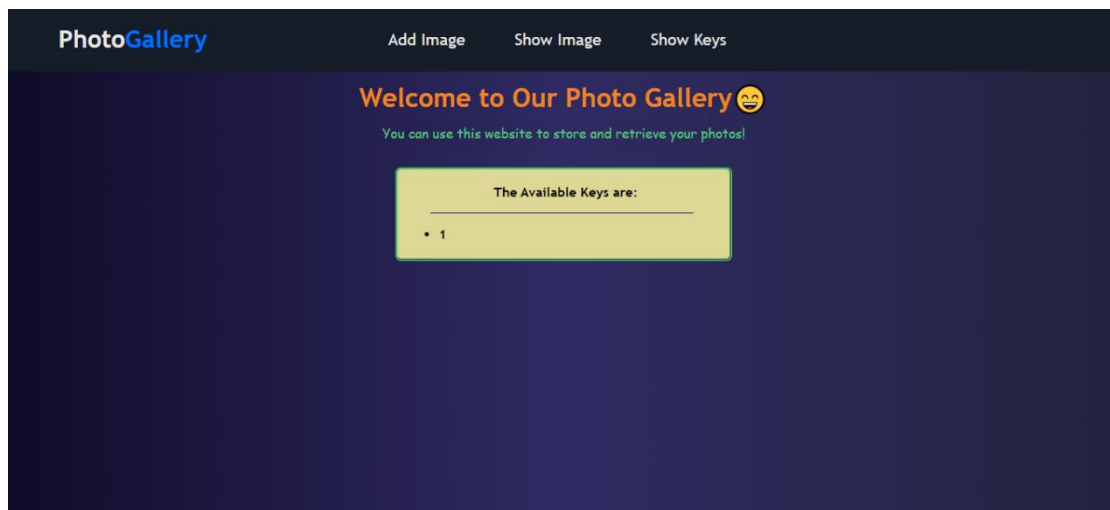
1. We are trying here to put a photo with key = **1** and name = **photo1.jpg**



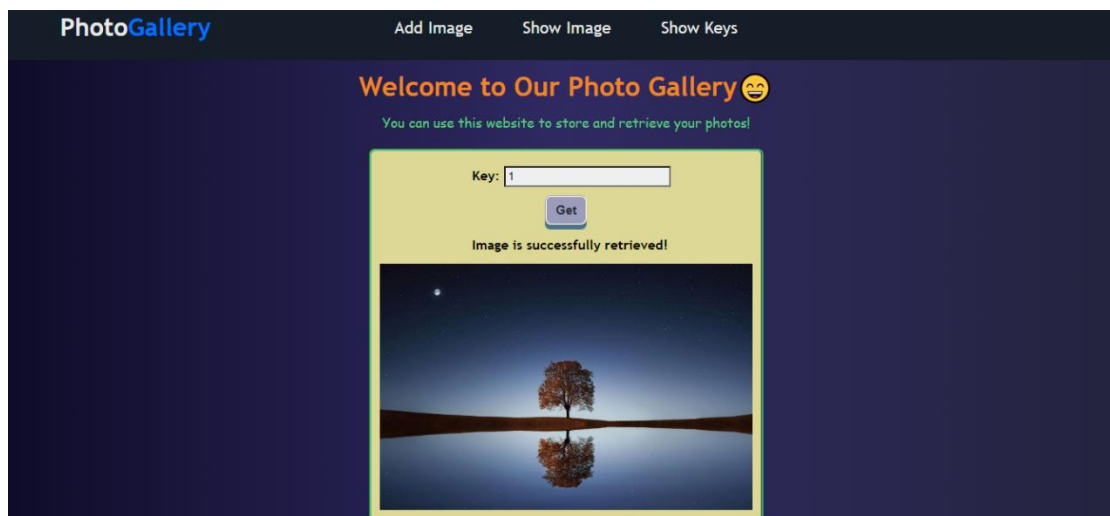
2. We can notice below that **photo1.jpg** is now in our S3 bucket.



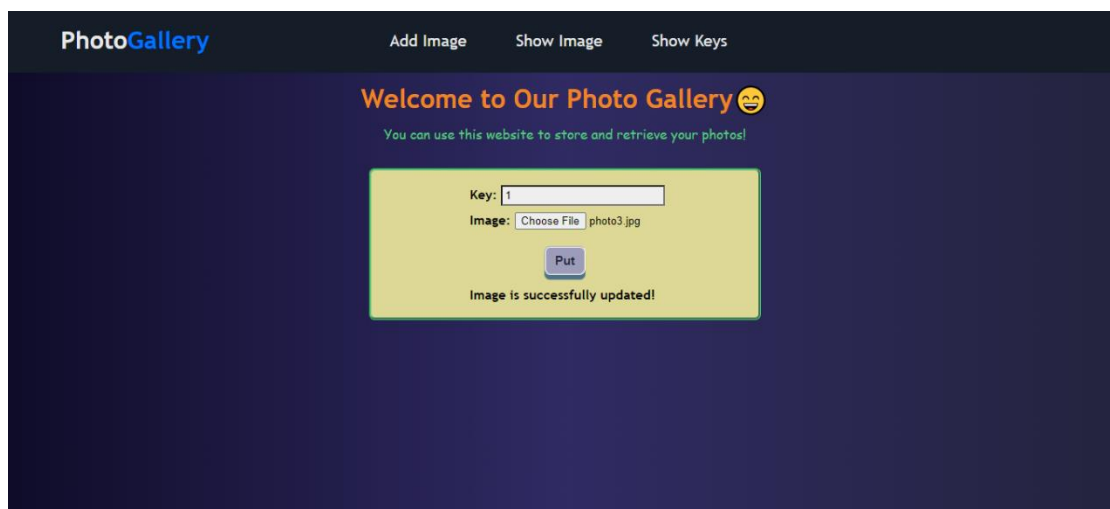
3. Also, the key **1** and **photo1.jpg** are in the database now.



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

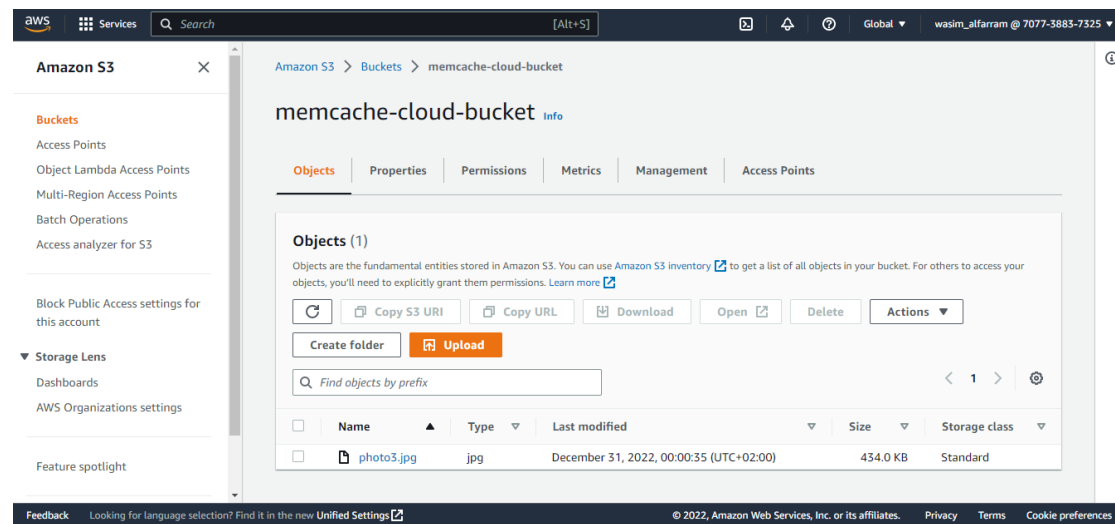


5. Now, we are updating the content of key **1** from **photo1.jpg** to **photo3.jpg**

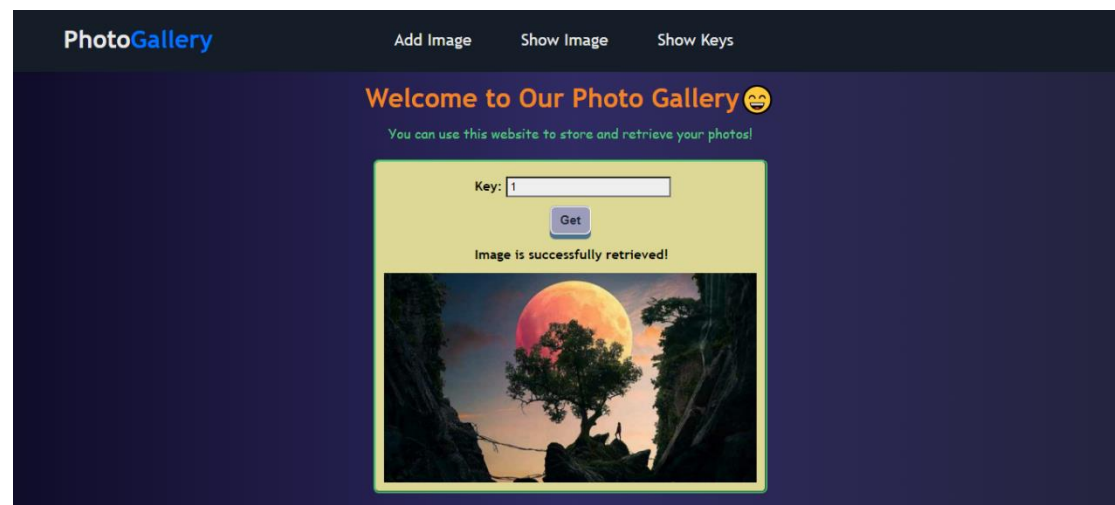




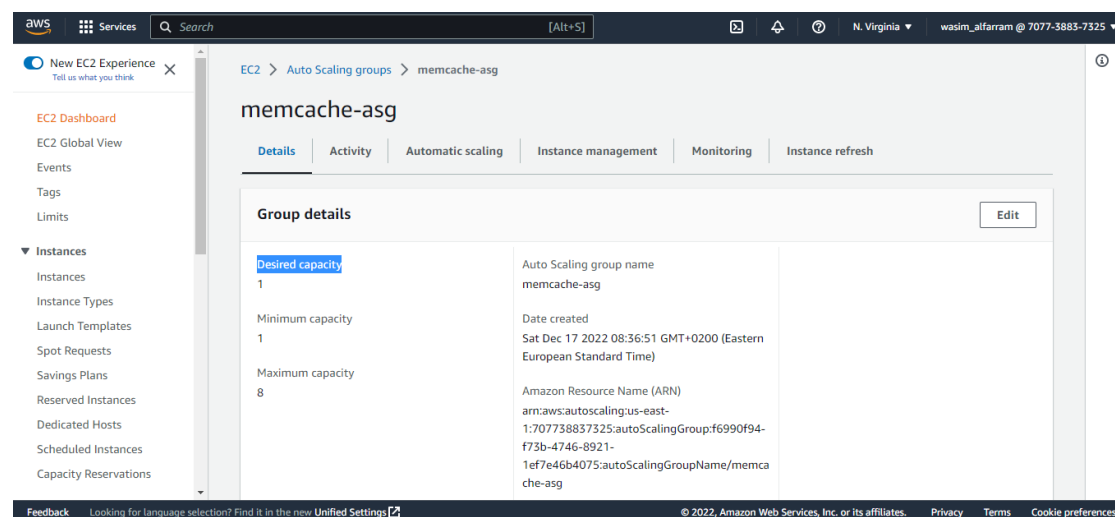
6. We can notice that **photo1.jpg** is replaced by **photo3.jpg** in our S3 bucket.



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.



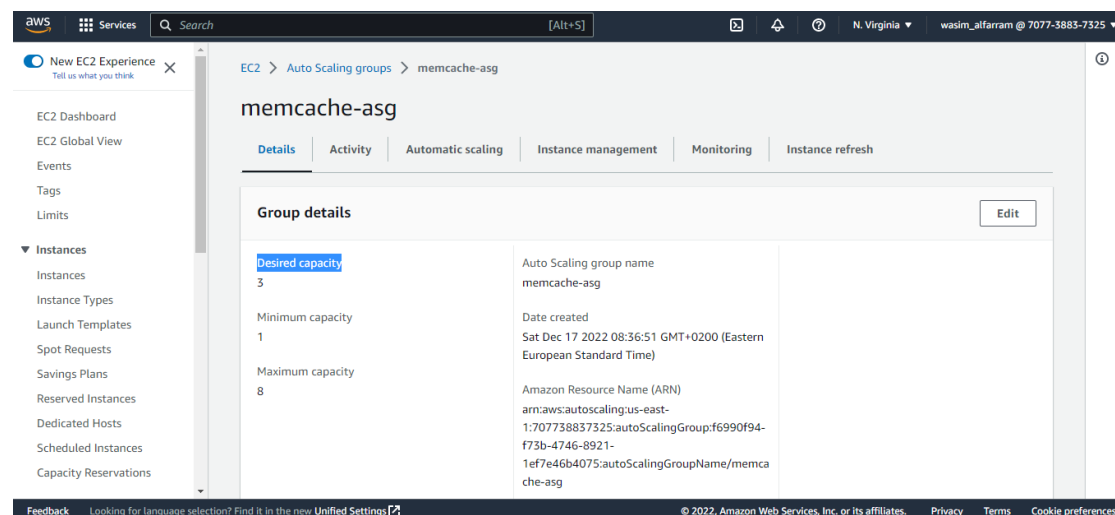
9. In the **Memory Configuration** page in our **manager-app**, we are trying to **expand** our pool by **8** so the total in the pool will be **9** which is not allowed to exceed **8 nodes** in the pool, so we got a caution message as shown.

The screenshot shows the 'PhotoGallery' application interface. At the top, there are tabs for 'Memory Configuration' and 'Memory Statistics'. Below the tabs, a welcome message reads 'Welcome to Our Photo Gallery' with a smiley face icon and a sub-message 'You can use this website to store and retrieve your photos!'. The main content area is a yellow box titled 'The mem-cache parameters configurations are:'. It contains several configuration options: 'Capacity in MB' set to 420, 'Replacement Policy' with 'Random Replacement' selected, 'Choose a memcache pool mode' with 'Manual Mode' selected, 'Specify to expand or shrink your pool' with 'Expand' selected, and 'Specify a number to resize your pool' set to 8. There are also two checkboxes: 'Do you want to delete all application data?' (No selected) and 'Do you want to clear the cache?' (No selected). A 'Set Configurations' button is at the bottom. Below the button, a red error message states 'Failed! Expanded Capacity is greater than 8'.

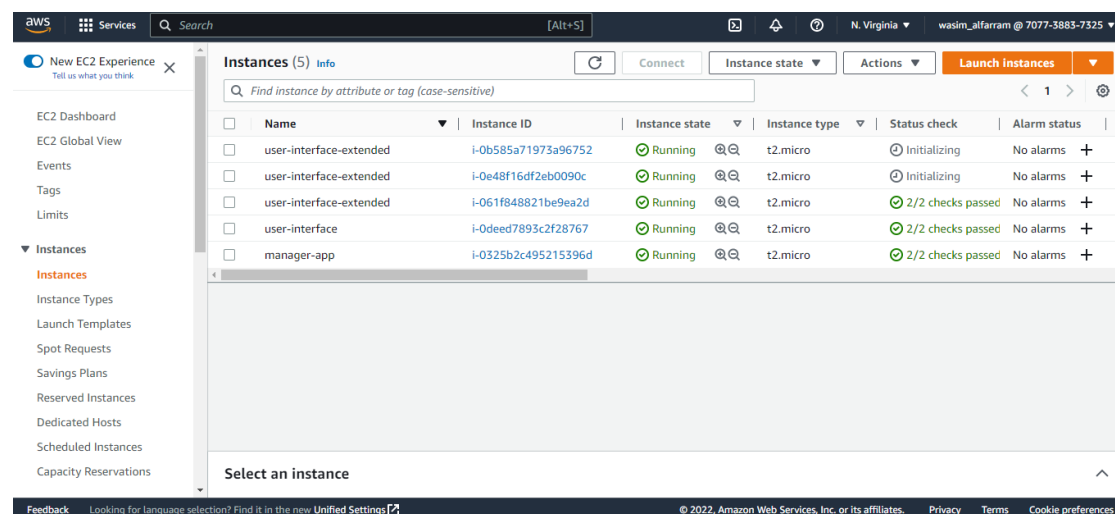
10. But when are trying to **expand** the pool by **2**, it will be expanded successfully so that we now have **3 running nodes** in our pool.

The screenshot shows the 'PhotoGallery' application interface, similar to the previous one. The 'Specify a number to resize your pool' field is now set to 2. The 'Set Configurations' button is at the bottom. Below the button, a green success message states 'Memcache Configurations are set successfully!'.

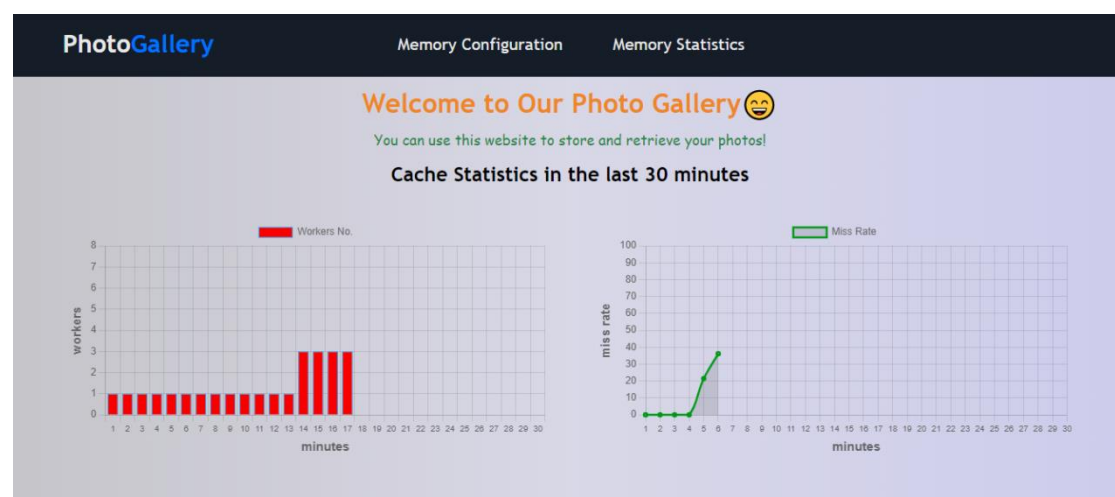
11. If we check the nodes in our pool, they will be **3 running nodes** exactly.



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

The screenshot shows the 'PhotoGallery' Memory Configuration page. The page has a header with 'PhotoGallery', 'Memory Configuration', and 'Memory Statistics' tabs. The main content area has a welcome message and a configuration panel for the mem-cache.

The configuration panel is titled 'The mem-cache parameters configurations are:' and contains the following settings:

- Capacity in MB: 300
- Replacement Policy: ☒ Random Replacement ☐ Least Recently Used
- Choose a memcache pool mode: ☒ Manual Mode ☐ Automatic Mode
- Specify to expand or shrink your pool: ☐ Expand ☒ Shrink
- Specify a number to resize your pool: 3
- Do you want to delete all application data? ☒ No ☐ Yes
- Do you want to clear the cache? ☒ No ☐ Yes

At the bottom of the panel is a 'Set Configurations' button. Below the panel, a red error message states: 'Failed! Shrunk Capacity is less than 1'.

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

The screenshot shows the 'PhotoGallery' website with a dark blue header containing 'PhotoGallery', 'Memory Configuration', and 'Memory Statistics'. Below the header is a light purple banner with the text 'Welcome to Our Photo Gallery 😊' and 'You can use this website to store and retrieve your photos!'. In the center, there is a yellow box titled 'The mem-cache parameters configurations are:'. Inside this box, there are several configuration options: 'Capacity in MB:' with a text input field containing '300'; 'Replacement Policy:' with radio buttons for 'Random Replacement' (selected) and 'Least Recently Used'; 'Choose a memcache pool mode:' with radio buttons for 'Manual Mode' (selected) and 'Automatic Mode'; 'Specify to expand or shrink your pool:' with radio buttons for 'Expand' (selected) and 'Shrink'; 'Specify a number to resize your pool:' with a text input field containing '1'; 'Do you want to delete all application data?' with radio buttons for 'No' (selected) and 'Yes'; and 'Do you want to clear the cache?' with radio buttons for 'No' (selected) and 'Yes'. At the bottom of the yellow box is a 'Set Configurations' button. Below the button, it says 'Memcache Configurations are set successfully!'.

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

The screenshot shows the AWS Management Console interface. The top navigation bar includes the AWS logo, 'Services', a search bar, and user information. The left sidebar shows the 'EC2 Dashboard' and various navigation links. The main content area displays the 'memcache-asg' Auto Scaling Group details. The 'Details' tab is selected, showing a table with the following information:

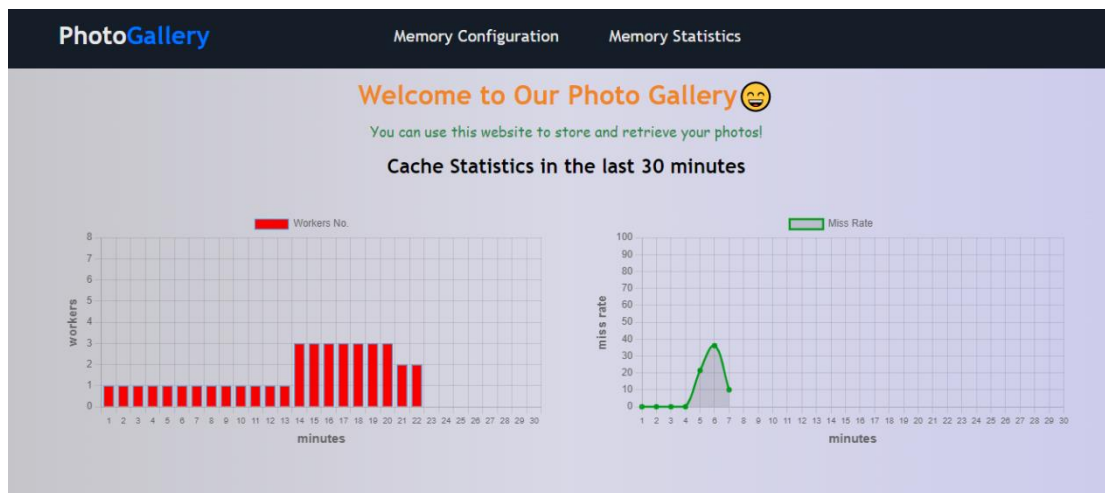
Group details	
Desired capacity	2
Minimum capacity	1
Maximum capacity	8
Auto Scaling group name	memcache-asg
Date created	Sat Dec 17 2022 08:36:51 GMT+0200 (Eastern European Standard Time)
Amazon Resource Name (ARN)	arn:aws:autoscaling:us-east-1:707738837325:autoScalingGroup:f6990f94-f73b-4746-8921-1ef7e46b4075:autoScalingGroupName/memcache-asg

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

The screenshot shows the AWS Management Console interface, specifically the 'Instances (5)' page. The top navigation bar and left sidebar are the same as in the previous screenshot. The main content area displays a table of instances. The table has columns for 'Name', 'Instance ID', 'Instance state', 'Instance type', 'Status check', and 'Alarm status'. The instances are listed as follows:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
user-interface-extended	i-0b585a71973a96752	Running	t2.micro	2/2 checks passed	No alarms
user-interface-extended	i-0e48f16df2eb0090c	Running	t2.micro	2/2 checks passed	No alarms
user-interface-extended	i-061f848821be9ea2d	Terminated	t2.micro	-	No alarms
user-interface	i-0deed7893c2f28767	Running	t2.micro	2/2 checks passed	No alarms
manager-app	i-0325b2c495215396d	Running	t2.micro	2/2 checks passed	No alarms

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

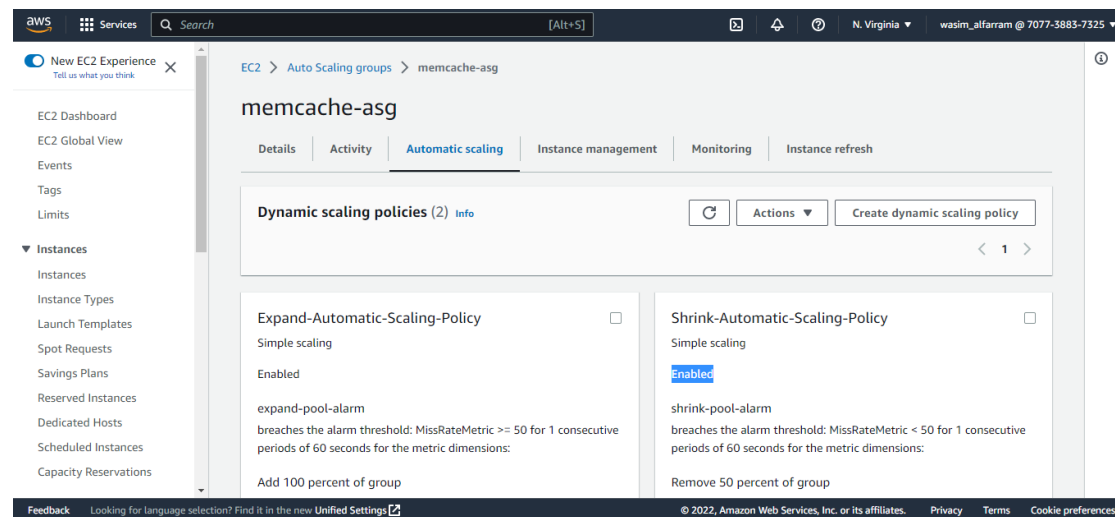


20. Now, we will check the Automatic Mode.

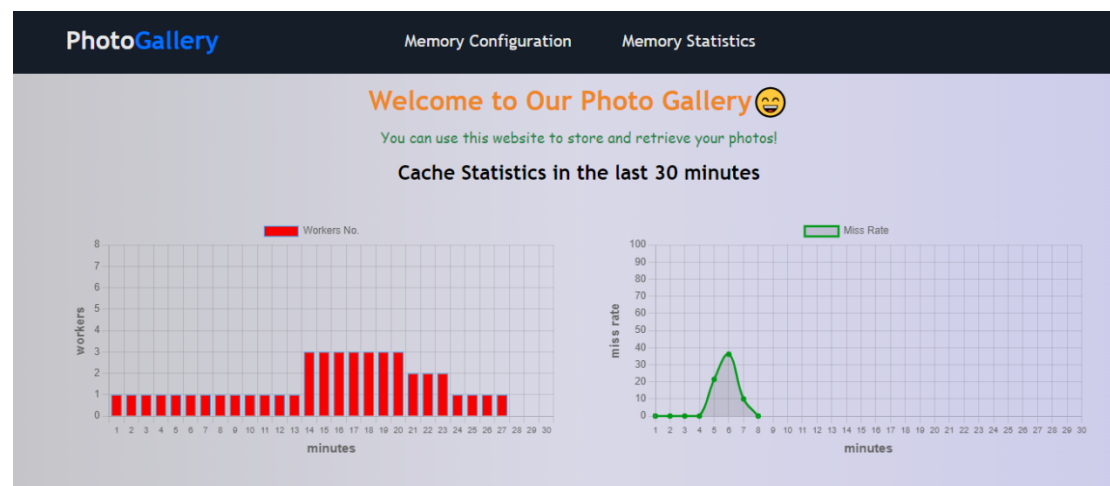
The screenshot shows the 'Memory Configuration' page of the 'PhotoGallery' application. The page has a dark blue header with the application name and two navigation links: 'Memory Configuration' and 'Memory Statistics'. Below the header, there is a welcome message and a subtitle 'The mem-cache parameters configurations are:'. A form is displayed with the following fields and options: 'Capacity in MB: 500', 'Replacement Policy: ☒ Random Replacement ☐ Least Recently Used', 'Choose a memcache pool mode: ☐ Manual Mode ☒ Automatic Mode', 'Do you want to delete all application data? ☒ No ☐ Yes', and 'Do you want to clear the cache? ☒ No ☐ Yes'. A 'Set Configurations' button is located below the form. At the bottom of the form, a message states 'Memcache Configurations are set successfully!'.

Parameter	Value
Capacity in MB	500
Replacement Policy	Random Replacement
Choose a memcache pool mode	Automatic Mode
Do you want to delete all application data?	No
Do you want to clear the cache?	No

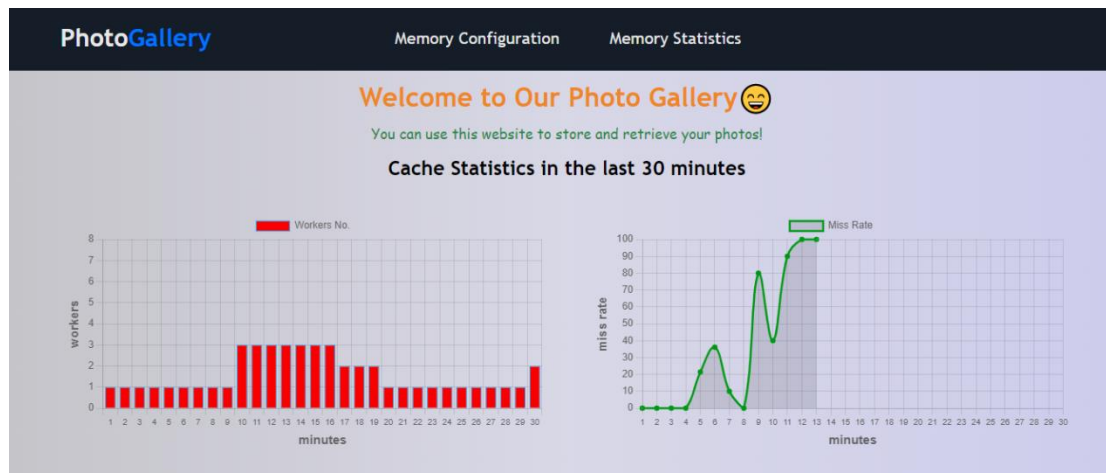
21. We can see here that after enabling the Automatic mode, **2 policies for Expanding and Shrinking the pool** are enabled.



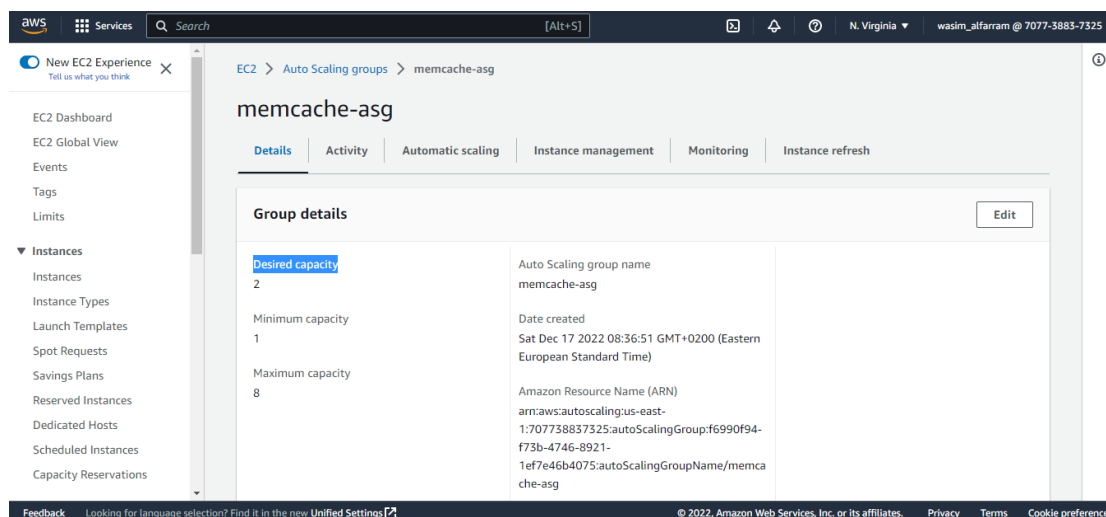
22. Since the miss rate is below 50%, the pool will be **automatically shrunk** from **2 workers** to **1 worker**.



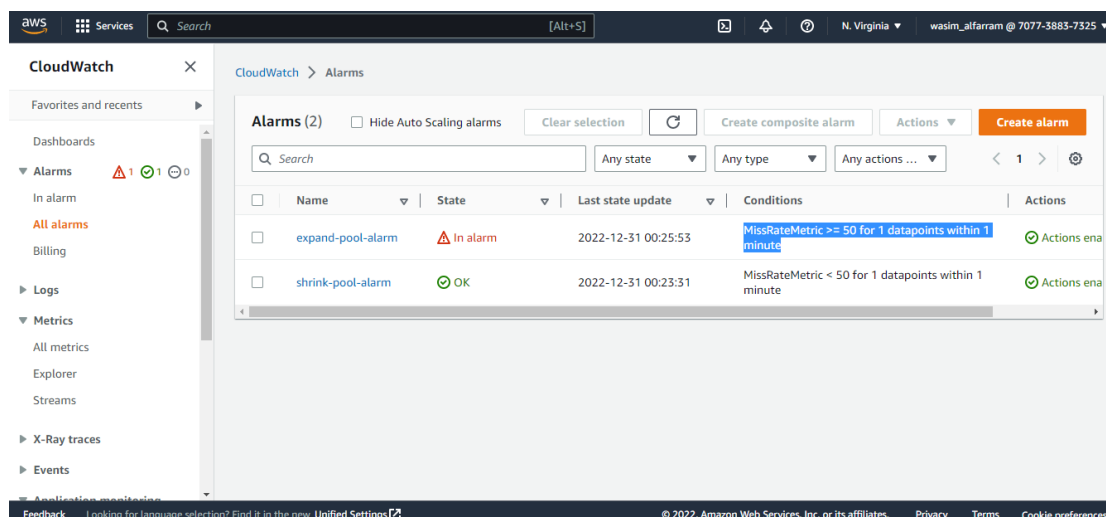
23. After having some action with too many miss rate, we can see the miss rate is now above 50% and the pool is **automatically expanded from 1 worker to 2 workers**.



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

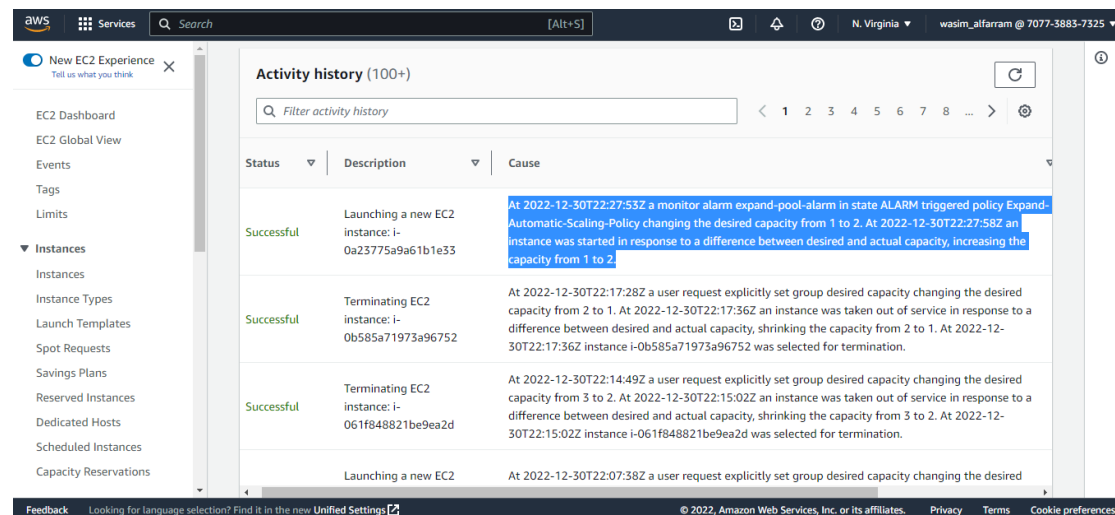


25. This is the **alarm** that notify me on email that my pool is **expanded to 2**





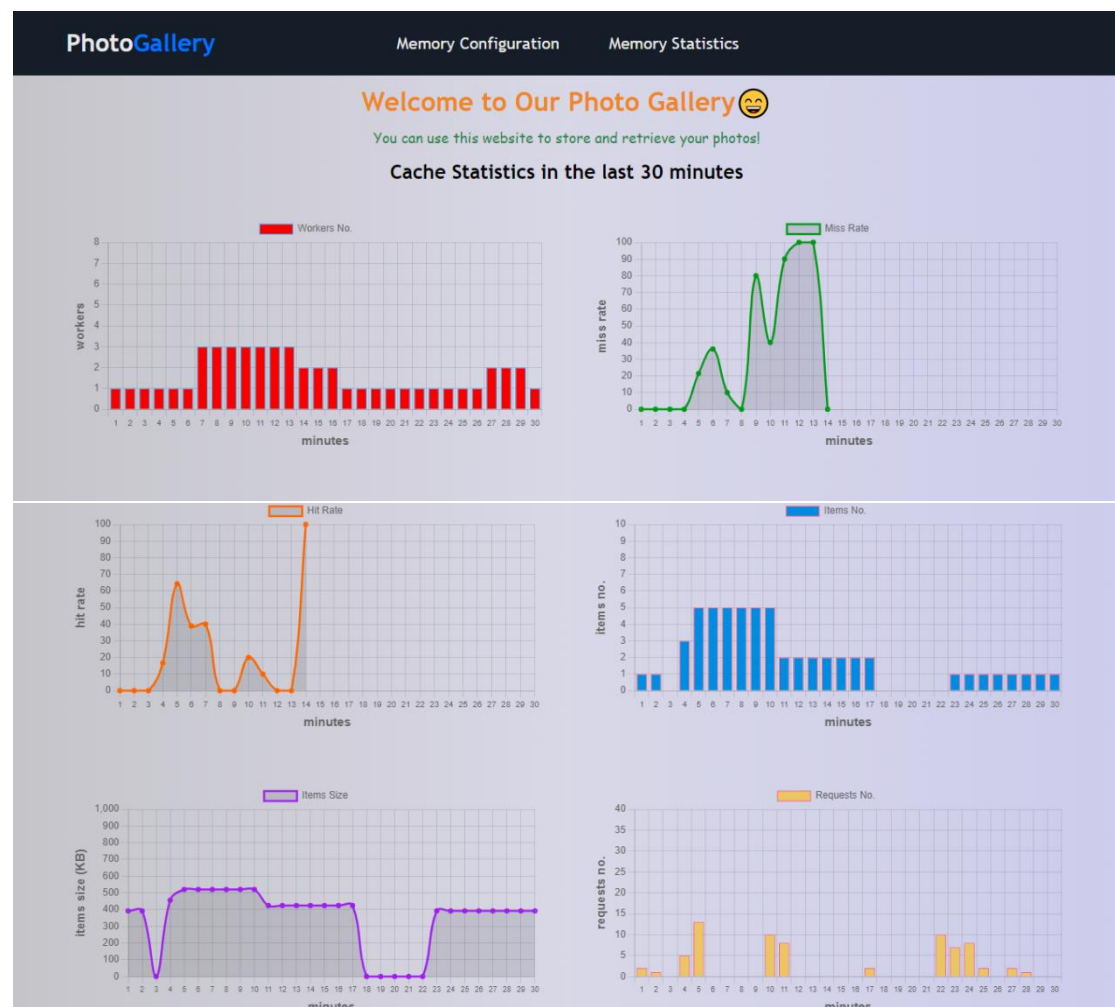
26. From the **Activity history** of my pool, I can see that the expansion has been done due to the **Expansion Policy** enabled before.



The screenshot shows the AWS Activity History console. The left sidebar contains navigation links for EC2 Dashboard, EC2 Global View, Events, Tags, Limits, and Instances. The main area displays the 'Activity history (100+)' table. The table has columns for Status, Description, and Cause. The first row shows a 'Successful' status for 'Launching a new EC2 instance: i-0a23775a9a61b1e33'. The cause for this event is highlighted in blue: 'At 2022-12-30T22:27:53Z a monitor alarm expand-pool-alarm in state ALARM triggered policy Expand-Automatic-Scaling-Policy changing the desired capacity from 1 to 2. At 2022-12-30T22:27:58Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.' The second row shows a 'Successful' status for 'Terminating EC2 instance: i-0b585a71973a96752'. The cause is: 'At 2022-12-30T22:17:28Z a user request explicitly set group desired capacity changing the desired capacity from 2 to 1. At 2022-12-30T22:17:36Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2022-12-30T22:17:36Z instance i-0b585a71973a96752 was selected for termination.' The third row shows a 'Successful' status for 'Terminating EC2 instance: i-061f848821be9ea2d'. The cause is: 'At 2022-12-30T22:14:49Z a user request explicitly set group desired capacity changing the desired capacity from 3 to 2. At 2022-12-30T22:15:02Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2022-12-30T22:15:02Z instance i-061f848821be9ea2d was selected for termination.' The fourth row shows a 'Successful' status for 'Launching a new EC2 instance: i-0a23775a9a61b1e33'. The cause is: 'At 2022-12-30T22:07:38Z a user request explicitly set group desired capacity changing the desired capacity from 1 to 2. At 2022-12-30T22:07:43Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.'

Status	Description	Cause
Successful	Launching a new EC2 instance: i-0a23775a9a61b1e33	At 2022-12-30T22:27:53Z a monitor alarm expand-pool-alarm in state ALARM triggered policy Expand-Automatic-Scaling-Policy changing the desired capacity from 1 to 2. At 2022-12-30T22:27:58Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.
Successful	Terminating EC2 instance: i-0b585a71973a96752	At 2022-12-30T22:17:28Z a user request explicitly set group desired capacity changing the desired capacity from 2 to 1. At 2022-12-30T22:17:36Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2022-12-30T22:17:36Z instance i-0b585a71973a96752 was selected for termination.
Successful	Terminating EC2 instance: i-061f848821be9ea2d	At 2022-12-30T22:14:49Z a user request explicitly set group desired capacity changing the desired capacity from 3 to 2. At 2022-12-30T22:15:02Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2022-12-30T22:15:02Z instance i-061f848821be9ea2d was selected for termination.
Successful	Launching a new EC2 instance: i-0a23775a9a61b1e33	At 2022-12-30T22:07:38Z a user request explicitly set group desired capacity changing the desired capacity from 1 to 2. At 2022-12-30T22:07:43Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.

27. Now, we the miss rate is below 50%, the pool is **automatically shrunk** 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.

```
app.py x cache.py layout.html add_image.html show_image.html show_keys.html # style.css
app.py > ...
1 from flask import Flask, render_template, request, flash
2 from flask_mysql import MySQL
3 from werkzeug.utils import secure_filename
4 from cache import Cache
5 from time import perf_counter
6 import os, threading, boto3
7
8
9 UPLOAD_FOLDER = 'static/destination_images/'
10 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
11
12
13 app = Flask(__name__)
14
15
16 app.config['MYSQL_HOST'] = 'memcache-database.cz0uhkd1snct.us-east-1.rds.amazonaws.com'
17 app.config['MYSQL_USER'] = 'admin'
18 app.config['MYSQL_PASSWORD'] = '12345678'
19 app.config['MYSQL_DB'] = 'Memcache_Database'
20 app.secret_key = "my-secret-key"
21 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
22
23
24 mysql = MySQL(app)
25 s3 = boto3.client("s3")
26 autoscaling = boto3.client("autoscaling")
27 cloudwatch = boto3.client("cloudwatch")
28 cache = Cache(500, "random-replacement")
29 hitsNo = 0
30 missNo = 0
31 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.

```
app.py x cache.py layout.html add_image.html show_image.html show_keys.html # style.css
app.py > ...
34 def updateRecord():
35     global hitsNo, missNo, reqs
36     hitRate = 0
37     missRate = 0
38     asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
39     asg_capacity = asg_dict["AutoScalingGroups"][0]["DesiredCapacity"]
40     if reqs != 0:
41         hitRate = (hitsNo/reqs)*100
42         missRate = (missNo/reqs)*100
43         cloudwatch.put_metric_data(Namespace = 'MissRate', MetricData = [{'MetricName': 'MissRateMetric', 'Value': missRate}])
44         cloudwatch.put_metric_data(Namespace = 'HitRate', MetricData = [{'MetricName': 'HitRateMetric', 'Value': hitRate}])
45         cloudwatch.put_metric_data(Namespace = 'Workers', MetricData = [{'MetricName': 'WorkersMetric', 'Value': asg_capacity}])
46         cloudwatch.put_metric_data(Namespace = 'ItemsNumber', MetricData = [{'MetricName': 'ItemsNumberMetric', 'Value': cache.length()}])
47         cloudwatch.put_metric_data(Namespace = 'ItemsSize', MetricData = [{'MetricName': 'ItemsSizeMetric', 'Value': cache.size()}])
48         cloudwatch.put_metric_data(Namespace = 'RequestsNumber', MetricData = [{'MetricName': 'RequestsNumberMetric', 'Value': reqs}])
49     hitsNo = 0
50     missNo = 0
51     reqs = 0
52
53 def counter():
54     t1_start = perf_counter()
55     send = True
56     while True:
57         now = perf_counter()
58         if (int(now)-int(t1_start)) % 5 == 0 and send:
59             updateRecord()
60             send = False
61         elif (int(now)-int(t1_start)) % 5 != 0:
62             send = True
63
64     t1 = threading.Thread(daemon=True, target=counter, args=())
65     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.

```
129 @app.route("/show_image/", methods=['POST', 'GET'])
130 def show_image():
131     global hitsNo, missNo, reqs
132     reqs += 1
133     img_src = '/static/temp.jpg'
134     cursor = mysql.connection.cursor()
135     cursor.execute("SELECT clear_cache FROM memory_configuration WHERE seq = 1")
136     clear_cache = cursor.fetchone()
137     if clear_cache[0] == 'yes':
138         cache.clear()
139     cursor.execute("SELECT image_key FROM image")
140     keys = cursor.fetchall()
141     if len(keys) == 0:
142         list = os.listdir('static/destination_images/')
143         for file in list:
144             if os.path.exists('static/destination_images/'):
145                 os.remove('static/destination_images/' + file)
146     if request.method == 'POST':
147         key = int(request.form['key'])
148         cursor.execute("SELECT image_name FROM image WHERE image_key = %s", [key])
149         file = cursor.fetchone()
150         if key in cache.data:
151             hitsNo += 1
152             [src, ext] = cache.get(key)
153             img_src = "data:image/" + ext + ";base64," + src.decode()
154         else:
155             if file:
156                 filename = secure_filename(file[0])
157                 s3.download_file(Bucket="memcache-cloud-bucket", Key=filename, Filename="static/destination_images/"+filename)
158                 cache.put(int(key), filename)
159                 img_src = '/' + os.path.join(app.config['UPLOAD_FOLDER'], filename)
160                 flash('Image is successfully retrieved!')
161             else:
162                 missNo += 1
163     mysql.connection.commit()
164     cursor.close()
165     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 x cache.py layout.html add_image.html show_image.html show_keys.html # style.css
app.py > ...
167
168 @app.route("/show_keys/", methods=['POST', 'GET'])
169 def show_keys():
170     cursor = mysql.connection.cursor()
171     cursor.execute("SELECT image_key FROM image")
172     keys = cursor.fetchall()
173     if len(keys) == 0:
174         list = os.listdir('static/destination_images/')
175         for file in list:
176             if os.path.exists('static/destination_images/'):
177                 os.remove('static/destination_images/' + file)
178     if request.method == 'GET':
179         cursor.execute("SELECT image_key FROM image ORDER BY image_key ASC")
180         keys = cursor.fetchall()
181         for key in keys:
182             flash(key[0])
183     mysql.connection.commit()
184     cursor.close()
185     return render_template("show_keys.html")
186
187
188 app.run(host='localhost', port=5000, debug=True)
189
```

## 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.

```
app.py x cache.py layout.html memory_configuration.html memory_statistics.html JS indexjs # style.css
app.py > update_metric_data
1 import threading
2 from time import perf_counter
3 from flask import Flask, render_template, request, flash
4 from flask_mysqldb import MySQL
5 from datetime import datetime
6 from cache import Cache
7 from datetime import datetime, timedelta
8 import boto3
9
10
11 app = Flask(__name__)
12
13
14 app.config['MYSQL_HOST'] = 'localhost'
15 app.config['MYSQL_USER'] = 'root'
16 app.config['MYSQL_PASSWORD'] = ''
17 app.config['MYSQL_DB'] = 'photo_gallery'
18 app.secret_key = "my-secret-key"
19
20
21 mysql = MySQL(app)
22 s3 = boto3.resource("s3")
23 autoscaling = boto3.client('autoscaling')
24 cloudwatch = boto3.Client('cloudwatch')
25 cache = Cache(500, "random-replacement")
26 workers_metric = []
27 missRate_metric = []
28 hitRate_metric = []
29 itemsNumber_metric = []
30 itemsSize_metric = []
31 requests_metric = []
```

- This part of code gets the statistics data from the CloudWatch custom metrics 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 x cache.py layout.html memory_configuration.html memory_statistics.html JS index.js # style.css
app.py > ...
34 def update_metric_data():
35     start_time = datetime.utcnow() - timedelta(seconds=60)
36     end_time = datetime.utcnow()
37     missRate_params = {
38         'Namespace': 'MissRate',
39         'MetricName': 'MissRateMetric',
40         'StartTime': start_time,
41         'EndTime': end_time,
42         'Period': 60,
43         'Statistics': ['Average']
44     }
45     hitRate_params = {
46         'Namespace': 'HitRate',
47         'MetricName': 'HitRateMetric',
48         'StartTime': start_time,
49         'EndTime': end_time,
50         'Period': 60,
51         'Statistics': ['Average']
52     }
53     itemsNumber_params = {
54         'Namespace': 'ItemsNumber',
55         'MetricName': 'ItemsNumberMetric',
56         'StartTime': start_time,
57         'EndTime': end_time,
58         'Period': 60,
59         'Statistics': ['Maximum']
60     }
61     itemsSize_params = {
62         'Namespace': 'ItemsSize',
63         'MetricName': 'ItemsSizeMetric',
64         'StartTime': start_time,
65         'EndTime': end_time,
66         'Period': 60,
67         'Statistics': ['Maximum']
68     }
69     requests_params = {
70         'Namespace': 'RequestsNumber',
71         'MetricName': 'RequestsNumberMetric',
72         'StartTime': start_time,
73         'EndTime': end_time,
74         'Period': 60,
75         'Statistics': ['Sum']
76     }
77
78     missRate_response = cloudwatch.get_metric_statistics(**missRate_params)
79     hitRate_response = cloudwatch.get_metric_statistics(**hitRate_params)
80     itemsNumber_response = cloudwatch.get_metric_statistics(**itemsNumber_params)
81     itemsSize_response = cloudwatch.get_metric_statistics(**itemsSize_params)
82     requests_response = cloudwatch.get_metric_statistics(**requests_params)
83
84     if len(workers_metric) == 30:
85         workers_metric.pop(0)
86     if len(missRate_metric) == 30:
87         missRate_metric.pop(0)
88     if len(hitRate_metric) == 30:
89         hitRate_metric.pop(0)
90     if len(itemsNumber_metric) == 30:
91         itemsNumber_metric.pop(0)
92     if len(itemsSize_metric) == 30:
93         itemsSize_metric.pop(0)
94     if len(requests_metric) == 30:
95         requests_metric.pop(0)
96
97     asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
98     workers_data_point = asg_dict['AutoScalingGroups'][0]['DesiredCapacity']
99     workers_metric.append(workers_data_point)
100     if len(missRate_response['Datapoints']) != 0:
101         missRate_data_point = missRate_response['Datapoints'][0]['Average']
102         missRate_metric.append(missRate_data_point)
103     if len(hitRate_response['Datapoints']) != 0:
104         hitRate_data_point = hitRate_response['Datapoints'][0]['Average']
105         hitRate_metric.append(hitRate_data_point)
106     if len(itemsNumber_response['Datapoints']) != 0:
107         itemsNumber_data_point = itemsNumber_response['Datapoints'][0]['Maximum']
108         itemsNumber_metric.append(itemsNumber_data_point)
109     if len(itemsSize_response['Datapoints']) != 0:
110         itemsSize_data_point = itemsSize_response['Datapoints'][0]['Maximum']
111         itemsSize_metric.append(itemsSize_data_point)
112     if len(requests_response['Datapoints']) != 0:
113         requests_data_point = requests_response['Datapoints'][0]['Sum']
114         requests_metric.append(requests_data_point)
```

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

```
app.py x cache.py layout.html memory_configuration.html memory_statistics.html JS indexjs # style.css
app.py > update_metric_data
116
117 def counter():
118     t1_start = perf_counter()
119     send = True
120     while True:
121         now = perf_counter()
122         if (int(now)-int(t1_start)) % 60 == 0 and send:
123             update_metric_data()
124             send = False
125         elif (int(now)-int(t1_start)) % 60 != 0:
126             send = True
127
128
129 t1 = threading.Thread(daemon=True, target=counter, args=())
130 t1.start()
131
```

4. This part of code is for the **Memory Configuration** page. It allows the manager 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.

```
app.py x cache.py layout.html memory_configuration.html memory_statistics.html JS indexjs # style.css
app.py > ...
132
133 @app.route("/", methods=['POST', 'GET'])
134 @app.route("/memory_configuration/", methods=['POST', 'GET'])
135 def memory_configuration():
136     if request.method == 'POST':
137         flash_message = 'Memcache Configurations are set successfully!'
138         capacity = request.form['capacity']
139         if capacity == "":
140             cursor = mysql.connection.cursor()
141             cursor.execute("SELECT capacity FROM memory_configuration WHERE seq = 1")
142             cap = cursor.fetchone()
143             capacity = cap[0]
144             mysql.connection.commit()
145             cursor.close()
146         replacement_policy = request.form['replacement-policy']
147         memcache_pool_resizing_option = request.form['memcache-pool-resizing-option']
148
149     if memcache_pool_resizing_option == 'manual':
150         autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Expand-Automatic-Scaling-Policy', Enabled = False)
151         autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Shrink-Automatic-Scaling-Policy', Enabled = False)
152         pool_size_option = request.form['pool-size-option']
153         pool_resize_number = request.form['pool-resize-number']
154         if pool_resize_number == "":
155             pool_resize_number = 0
156         else:
157             pool_resize_number = int(pool_resize_number)
158         asg_dict = autoscaling.describe_auto_scaling_groups(AutoScalingGroupNames=['memcache-asg'])
159         asg_capacity = asg_dict["AutoScalingGroups"][0]["DesiredCapacity"]
160         if pool_size_option == 'expand':
161             if asg_capacity + pool_resize_number <= 8:
162                 autoscaling.set_desired_capacity(AutoScalingGroupName = 'memcache-asg', DesiredCapacity = asg_capacity + pool_resize_number)
163             else:
164                 flash_message = 'Failed! Expanded Capacity is greater than 8'
165         else:
166             if asg_capacity - pool_resize_number >= 1:
167                 autoscaling.set_desired_capacity(AutoScalingGroupName = 'memcache-asg', DesiredCapacity = asg_capacity - pool_resize_number)
168             else:
169                 flash_message = 'Failed! Shrunk Capacity is less than 1'
170     else:
171         autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Expand-Automatic-Scaling-Policy', Enabled = True)
172         autoscaling.put_scaling_policy(AutoScalingGroupName = 'memcache-asg', PolicyName = 'Shrink-Automatic-Scaling-Policy', Enabled = True)
173
174     clear_cache = request.form['clear-cache']
175     delete_application_data = request.form['delete-application-data']
176     cursor = mysql.connection.cursor()
177     if delete_application_data == 'yes':
178         cursor.execute("DELETE FROM image")
179         bucket = s3.Bucket('memcache-cloud-bucket')
180         bucket.objects.all().delete()
181     if clear_cache == 'yes':
182         cache.clear()
183         cursor.execute("UPDATE memory_configuration SET capacity = %s, replacement_policy = %s, clear_cache = %s WHERE seq = 1", (capacity, repla
184         mysql.connection.commit()
185         cursor.close()
186         cache.refreshConfiguration(int(capacity), replacement_policy)
187         flash(flash_message)
188         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 metrics of the CloudWatch to the front-end code to show the data points on the graphs.

```
189 @app.route("/memory_statistics/")
190 def memory_statistics():
191     global workers_metric, missRate_metric, hitRate_metric, itemsNumber_metric, itemsSize_metric, requests_metric
192     return render_template("memory_statistics.html", workers = workers_metric, miss_rate = missRate_metric, hit_rate = hitRate_metric, items_numb
193
194
195 app.run(host='localhost', port=5000, debug=True)
196
```