

CLOUD COMPUTING

DYNAMIC RESOURCE MANAGEMENT

By

Akshat Mathur

Table of Contents

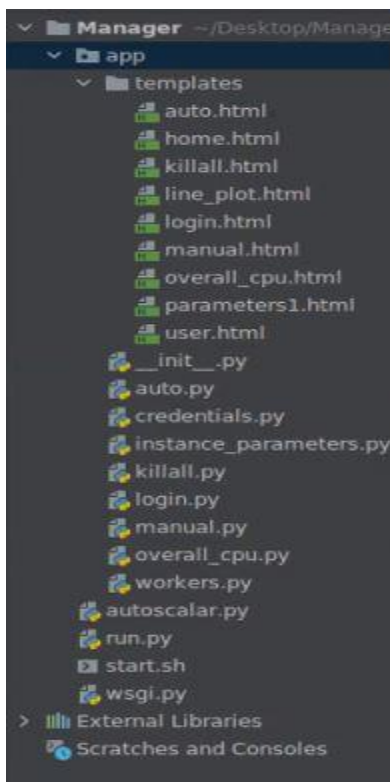
1.Introduction.....	3
2.Structure of the Program.....	3
3.Database Schema	4
4.Launching the Web Application	6
4.1 Running the EC2 Instance	6
4.2 Launching the Application.....	6
4.3 Testing the application with load generator:.....	8
4.4 HTTP request rate:	9
5. Using the Application	9
5.1 Logging In.....	9
5.2 Home page	10
5.3 Set manual mode.....	11
5.4 Set auto mode.....	12
5.5 EC2 parameters.....	13
5.6 Kill all	14
5.7 Overall CPU utilization.....	15
6.Auto scalar simulation:	15

1.Introduction

In this assignment the website developed for mask detection (user application) has been implemented using the S3 bucket and Relational database services of AWS. Apart from this we have developed a manager application to increase or decrease the number of user application EC2 instances (Auto Scaling) and to monitor their CPU usage and HTTP request rate. We also used EC2 elastic load balancer to distribute the requests among different EC2 user instances. In this report we will first go through the how the manager application is structured and the design of the different components of manager instances. Then we will look at the different webpages and their usage. This report does not provide explanations about user application (mask detection program) since it is the same which was developed in [Face_Mask_Detection_Webapp](#) with some minor changes (to make it work with AWS RDS).

2.Structure of the Program

In the manager app we have 2 separate python programs, one to implement the auto scalar function and the other to host the website. The file structure is shown below:



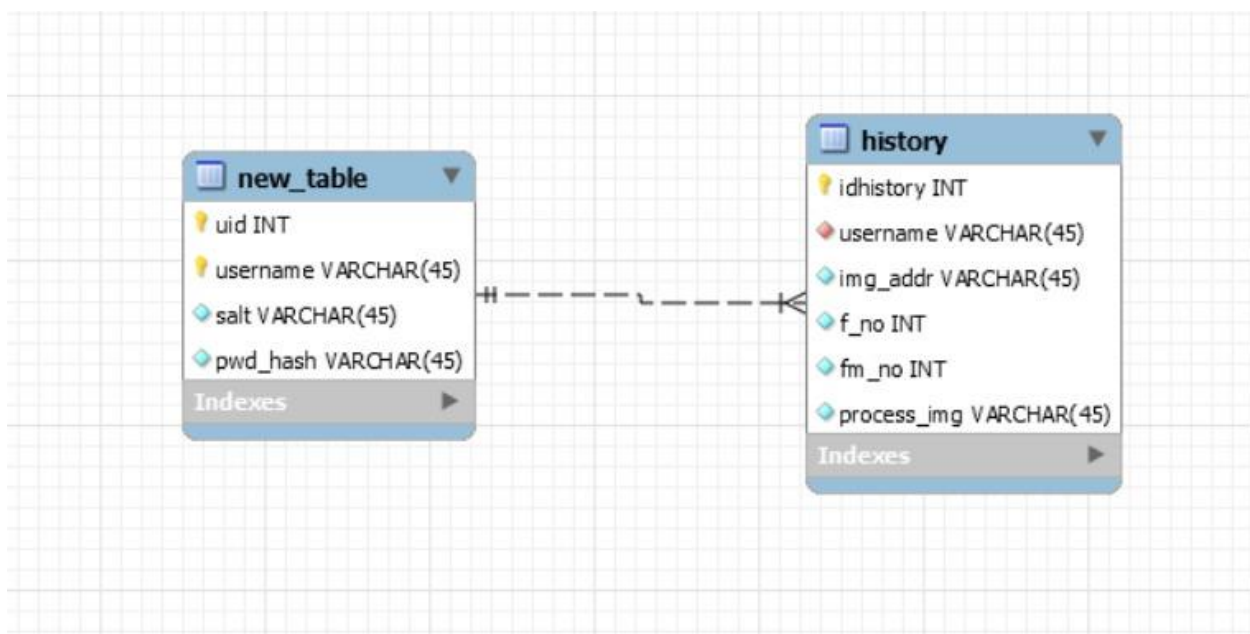
Inside the Manager folder we have autoscaler.py which implements the auto scaling of user application ec2 instances. We have generated an image of user application EC2 instance which automatically starts the website once it is launched. This image AMI is used to dynamically create new instances as and when needed.

Apart from this we have a run.py file which is used to host the manager website, all the files required for the working of this website are inside app folder. Inside this, we have templates folder which has the HTML files required for the website.

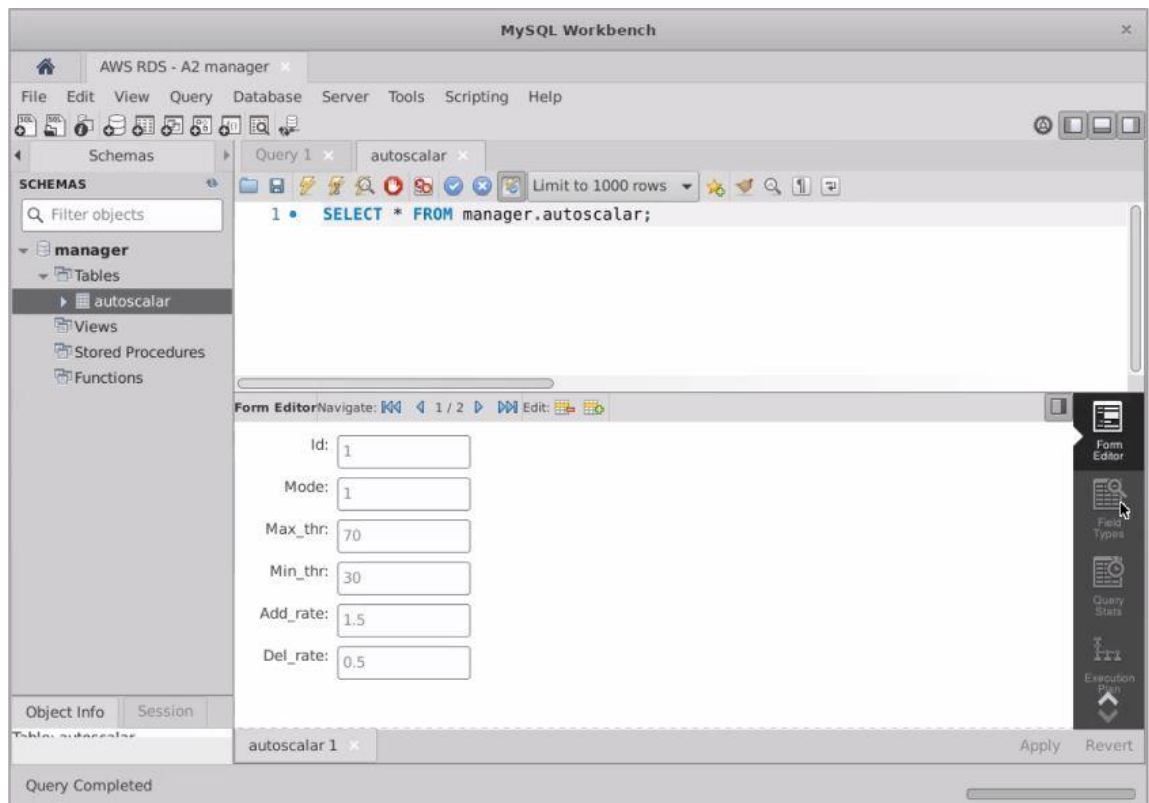
We will look at the individual webpages and their functions later in this report.

3.Database Schema

The database schema for the user application is same as the one used in [Face Mask Detection Webapp](#). The only difference is that the MySQL database is now created in Amazon RDS instead of the EC2 instance. This is required for this assignment because we will be dynamically creating instances based on load and having a common database which is independent of instances is required. The database schema of the RDS database is shown below and it is same as the one used in [Face Mask Detection Webapp](#).



Apart from this we also have a RDS database for manager application. The purpose of the database is to store the auto scalar parameters which can be updated from the website. The database has only one table with one row, the values in this row are changed as and when the parameters are updated in the manager website.



Here Id is the primary key, mode represents automatic or manual mode selection, and all other parameters are related to the auto scalar function.

A separate database for the auto scalar function was created as it is an independent program which runs continuously irrespective of the other operations happening in the manager app. Hence, this database serves as the link between the auto scaling operation and the manual input of the parameters if changes are to be made.

4.Launching the Web Application

4.1 Running the EC2 Instance

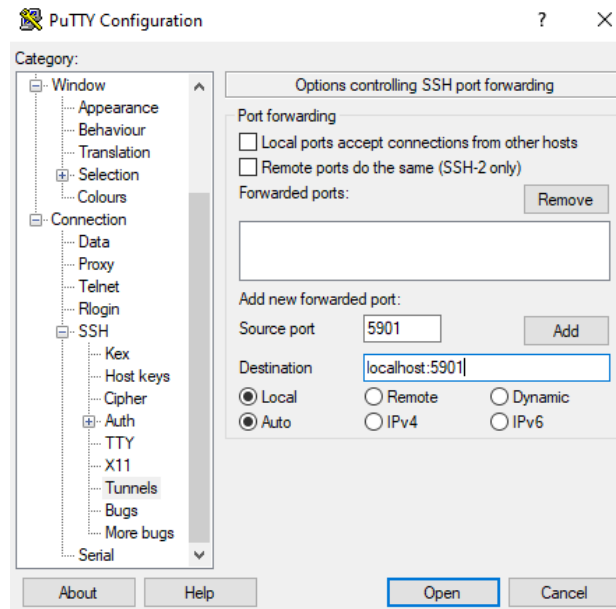
1. This account was created using AWS educate link provided to our class, so the TAs and professor will have access to the account.
2. Once you log in to the account, go to services and click on EC2.
3. Now select *Instance* on the left side of the page to view all available instances.
4. There will be a 'A2_manager' instance available in the account
5. Now click on the *Instance ID*, then go to *Action* and select *Start Instance*

Once the Instance has started, we will be able to get the “Public IPv4 address” of the instance which is needed to access the instance from our system.

4.2 Launching the Application

To launch the application from a remote PC we need the IP address of the EC2 instance (can be obtained after running the EC2 instance) and we need the key (.pem file) which has been attached with the submission. The following steps explain how to access the EC2 instance using putty and start the web application.

1. Install Putty and Puttygen in your PC.
2. Open Puttygen and load the key (.pem file) into it using Load option
3. Next click on “Generate” and then click on save private key (.ppk file)
4. Putty accepts the key in ppk format.
5. Open Putty, enter the hostname *ubuntu@<Instance IP Address>*
6. Click on Auth under Category->connection->SSH, Here browse and add the .ppk file
7. Next go to tunnels under Category->connection->SSH, Now enter the following details



8. Now click on *Add* to add the tunnel and then click on *Open*
9. Now a terminal window will open giving us access to our manager Ec2 instance and we also have created a tunnel to access the instance using VNC viewer if needed.
10. In the terminal run the following command to start the web application
 - `cd ~/Desktop`
 - `./start.sh`
11. Now open the web browser in your PC and type the following Ip to access our web application
 <Instance public IP address:5000>

The start.sh file automatically starts the manager application with one user EC2 instance. The contents of start.sh file is shown below

```

/home/ubuntu/Desktop/start.sh - Mousepad
File Edit Search View Document Help
#./bin/gunicorn --bind 0.0.0.0:5000 --workers=1 --access-logfile access.log --error-logfile error.log app:app
#start.sh
#source start.sh

cd
cd /home/ubuntu/Desktop/Manager
sudo ufw allow 5000
gunicorn --bind 0.0.0.0:5000 wsgi:app &
python3 autoscalar.py &

```

We start the manager web application and also run the auto scalar function concurrently in the background.

4.3 Testing the application with load generator:

The manager application remote desktop is shown below:



All programs are stored in the 'Manager' folder on the desktop. Apart from this we have gen.py to test the application. This file uploads the test images stored in 'load_images' folder using the "<LOAD Balancer DNS>/api/upload" URL. The following command is used in terminal to upload the images using gen.py:

```
python3 gen.py http://A2loadbalancer-1041861342.us-east-1.elb.amazonaws.com/api/upload  
admin ECE1779pass 1 ./load_images/ 100
```


It is important to note that some minor modifications were made in the gen.py file to use it with our application.

```
fd = FormData()
fd.add_field('file', file_contents, filename=path.basename(file_path))
fd.add_field('uname', username)
fd.add_field('pwd', password)
```

The field for username must be '**uname**' and the one for password must be '**pwd**'.

4.4 HTTP request rate:

The http request rate was calculated in user application ec2 instance and uploaded to cloud watch every minute. Later in the manager application we download the data from cloud watch to create the http request rate graph for each instance.

5. Using the Application

5.1 Logging In

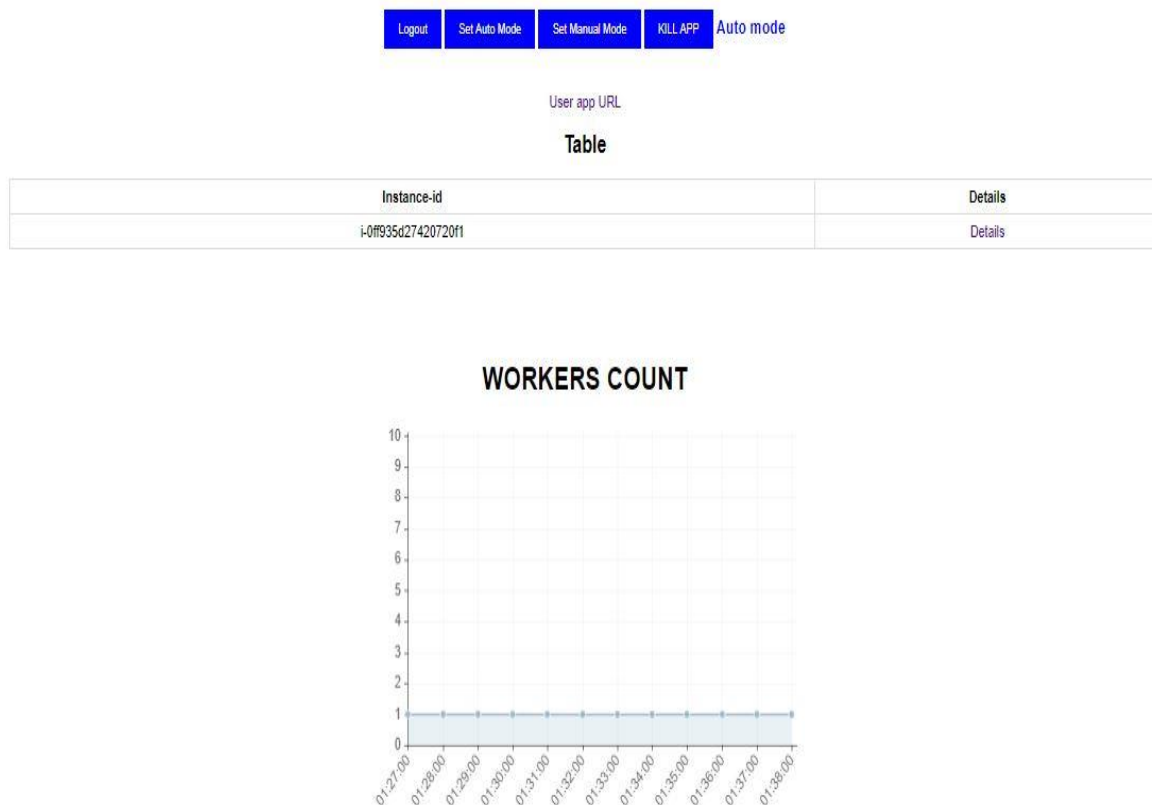
The manager app is not an open website. After you open the website, you will come across a login page. There is only one username and password to login to manager application. More usernames cannot be created. The username is **admin** and the password is also **admin**.



The login form consists of a title 'Login' in bold black font. Below it are two input fields: 'Username:' followed by a text box containing 'Enter username', and 'Password:' followed by a text box containing 'Enter password'. At the bottom center is a blue button with the text 'Login' in white.

Login with your username and password to access the homepage where you can access all your functions such as increasing or decreasing the number of instances, monitoring ec2 instances CPU utilization and the HTTP request rate. There is also an option to kill all running instances and to stop the manager app.

5.2 Home page



This page has the list of running instances and a plot of number of workers. This plot will show the values for the past 30 minutes, but this data is initialized when the manager app starts, so if we visit the home page as soon as the manager app is started, we will have just a single datapoint and only after 30 minutes of starting the manager application we will have the data for workers count plot for the past 30 minutes. Also, the time is in UTC and not the local time. All plots in the application are in UTC time.

Apart from this we also have an indicator on the top right corner of webpage to show the status of the manager application – automatic mode or manual mode.

The 'Kill App' function is to terminate all the ec2 instances, wipe the data from s3 bucket and RDS database and to stop the manager app.

We also have 'set auto mode' and 'set manual mode' buttons which take us to the respective pages. We have **user app URL** in the home page which redirects to link of Load balancer DNS to access the user app.

5.3 Set manual mode

[Home](#)

TOTAL INSTANCES RUNNING NOW: 1

Instances to add:

Instances to terminate:

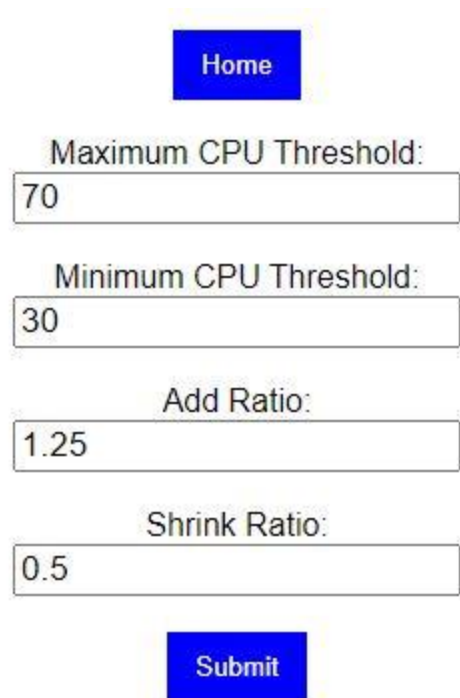
[Submit](#)

*** Pressing submit will set the mode to manual.**

Go to 'Set Auto' page to change it back to auto mode.

From this page we can manually add or delete instances. The max number of instances that can be created is limited to 8. Also, all instances cannot be terminated, at least one will be running. Once we submit the data, the status will change to manual mode and stays in manual until we go to the auto page and update the required parameters and submit them.

5.4 Set auto mode



Home

Maximum CPU Threshold:
70

Minimum CPU Threshold:
30

Add Ratio:
1.25

Shrink Ratio:
0.5

Submit

In auto mode page we can change the auto scaling parameters. Here the program will add instances when the add ratio is greater than 1 and it will shrink the instances only when the shrink ratio is less than 1. Apart from this we have thresholds to add and remove instances. Once we submit the details the manager app will be set to auto mode and will stay in auto mode unless changed in manual mode page.

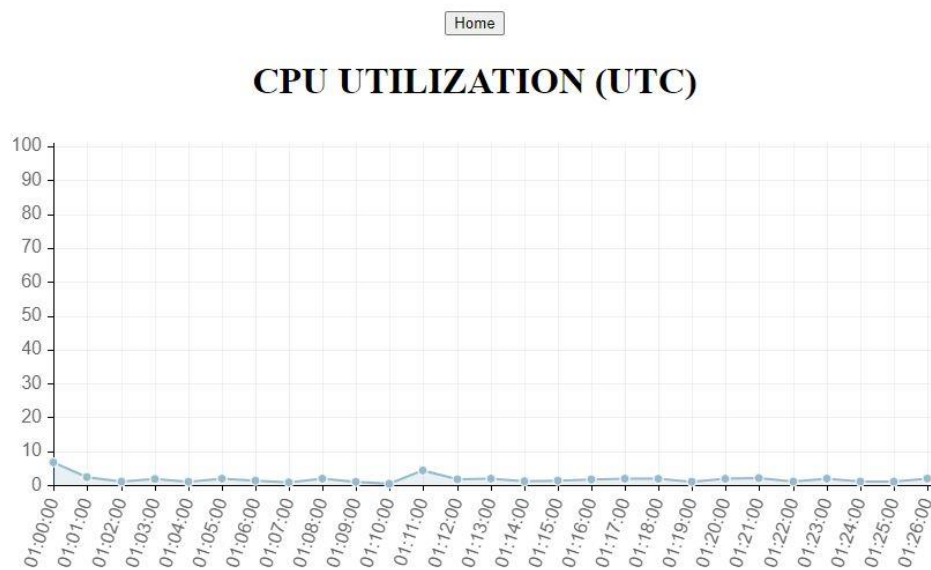
5.5 EC2 parameters

[Home](#)

Choose category: CPU PERFORMANCE ▼

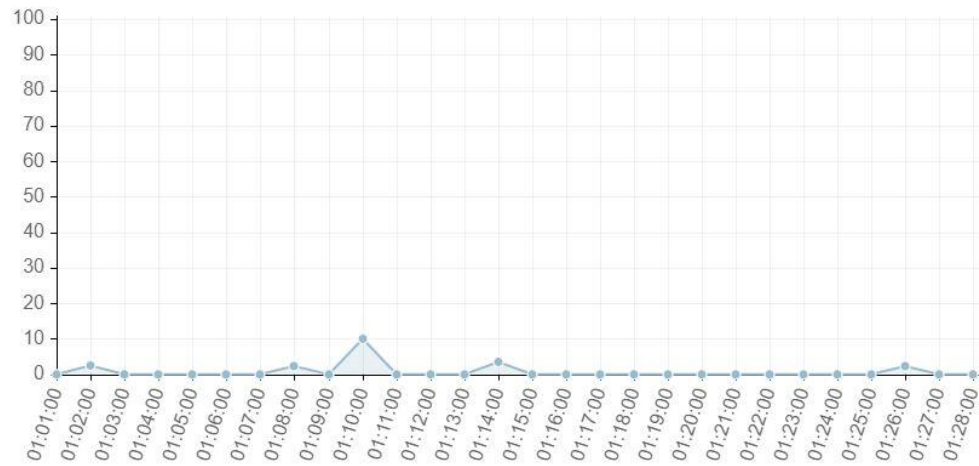
[Search](#)

In the home page we have a list of instances along with their details. When we click the details link it will take us to the above page where we have options to select CPU utilization or HTTP request rate. Once we submit, we will get the corresponding plot as shown below:



[Home](#)

HTTP REQUEST RATE



5.6 Kill all

[Home](#)

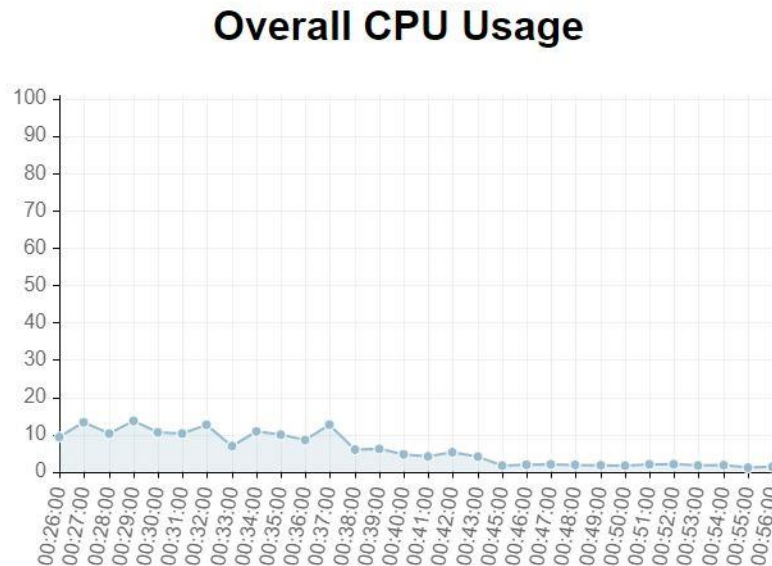
KILL APP

**ARE YOU SURE YOU WANT TO TERMINATE ALL THE
USER INSTANCES, DELETE ALL USER DATA AND STOP
THE MANAGER APP?**

CONFIRM KILL

In kill all page we can terminate all EC2 instances, delete all user data in S3 bucket, RDS values and stop the manager application.

5.7 Overall CPU utilization



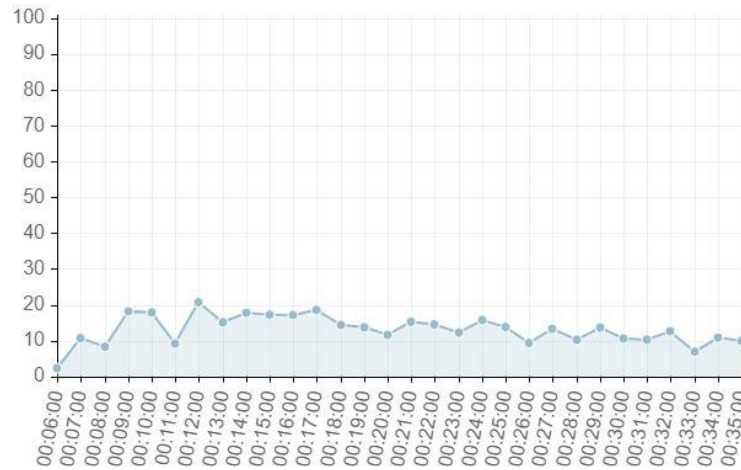
This page was created to help us in testing the manager application. This plot shows the average CPU utilization (Past 2 minutes average) of all the running instances. This plot along with the working instance plot will help check the working of auto scalar.

To access this information, one can simply go to <Instance public IP address>:5000/cpu

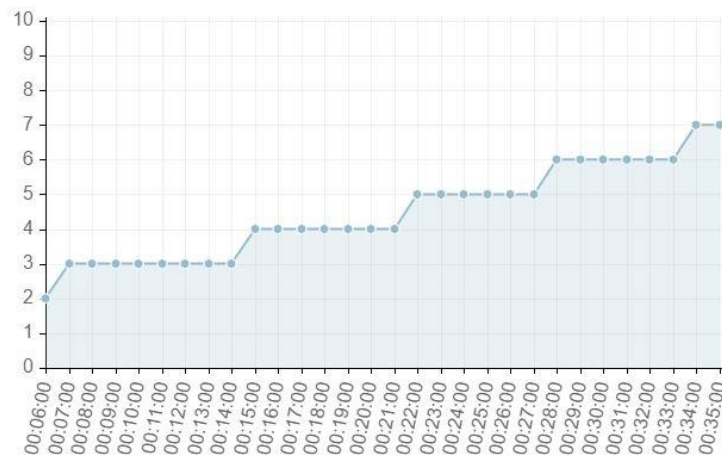
6.Auto scalar simulation:

The working of auto scalar function was simulated using the gen.py file. The following plots illustrate the working of auto scalar function. For this test, a t2.medium size Ec2 instance was used. The max threshold was set to 10 percent and min threshold was set to 2 percent. This might look small but t2.medium was able to handle the requests easily and there was not much increase in load, so these thresholds were set just to test the application and are not meant to be used in production environment.

Overall CPU Usage



WORKERS COUNT



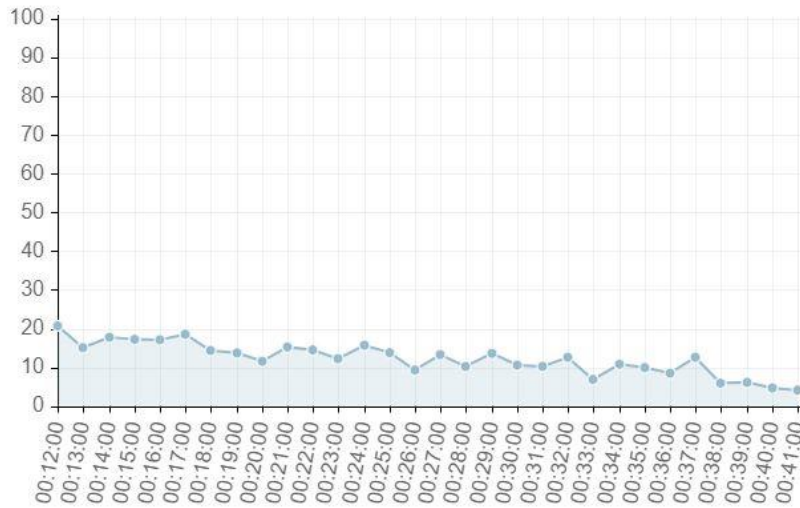
Once the load increased and stabilized, the 4th EC2 instance started at the time 00:15 and came into service at 00:19.

Even after the start of the 4th instance the load remained higher than the threshold and the 5th instance started at 00:22 and it came into service at 00:25.

But even then, the load remained higher than the threshold and the 6th instance was started at 00:28 and it came into service at 00:32.

We can see after few minutes 7th instance was also started to bring the load below threshold.

Overall CPU Usage



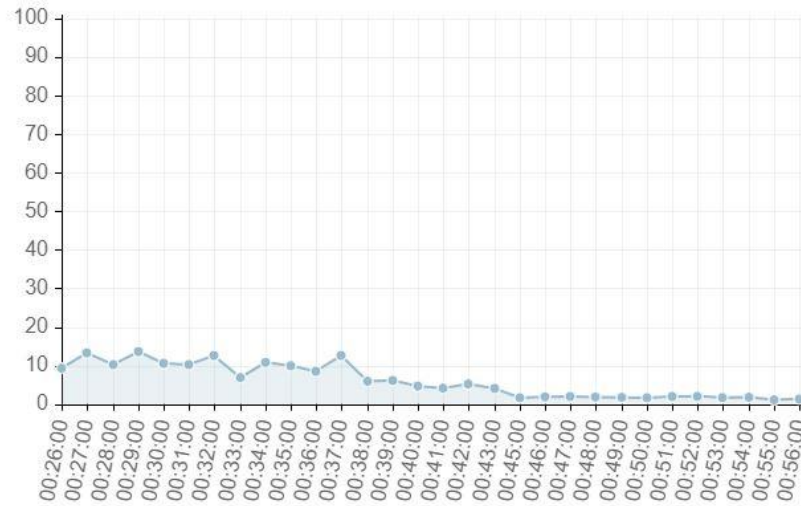
WORKERS COUNT



After the 7th instance started the CPU utilization stayed within the threshold for some time as it can be seen in the plot above.

Once the load remained within the threshold for some time, we reduced the load to test if the autoscaler function begins to terminate instances.

Overall CPU Usage



WORKERS COUNT



It can be seen from the above plot that once the CPU usage falls below minimum threshold the number of instances falls from 7 to 1.

This shows that the Auto scalar function converges.

For the auto scalar to converge and function properly, the program has to wait until the instance is create and is registered to the ELB. For this, we used boto3 built-in functions which will wait till the instance is ready. As any instance takes around 4-5 minutes to start listening to the world, a wait period is required or else the autoscalar will keep on adding new instances every minute as the previously launched instance is not ready yet.

*Sometimes, t2.small and t2.micro instances take longer to start and register with the ELB and causes a timeout for the waiter function. This is why we went with creating medium size instances.