ECE1779 Assignment 2
Dynamic Resource Management

Xiaoxi Yu	Jiani Jia	Shuya Wang
1005654748	1002226245	1006549503

Submitted on November 15, 2020

1 Login Info

AWS account: shuya.wang@mail.utoronto.ca
Instance Name: ece1779a2manager

2 Project Summary

In this project, we designed a manager web app to get control of the user web app. This website shows the total number of workers in the past 30 minutes to the manager as well as their instance details, which include the CPUUtilization chart and total HTTP request chart in the past 30 minutes. Also, this web app allows the manager to manually add one new instance or remove one old instance each time on the change worker pool size page. Sometimes when huge amounts of request flow into or flash out of the worker pool, managers won't be able to calculate the exact number of instances to resize, thus, we implement a load balancer which can partition the flow equally into all the workers, an auto-scaling part which can rationally resize the worker pool per minute automatically by comparing the CPU utilizations of each instance with given thresholds into this web app. We can also clear all the user data in the RDS database as well as S3 bucket and terminate all worker instances then stop the manager app. The below flow figure 1 and figure 2 summarize the structure of this web app.

3 Features

1. Save user information into the RDS database and save all the upload files into S3 bucket.
2. Resize worker pool size to one
3. Display number of workers of the past 30 minutes
4. Display an info list of all workers in the worker pool
5. Plot the CPU utilization chart and HTTP request of each instance
6. Display the load balancer DNS name and link it with the user app
7. Manually increase and decrease one instance each time
8. Customize the parameters of the auto scaler
9. Delete all user data and S3 data
10. Terminate all workers in the worker pool then stop manager instance

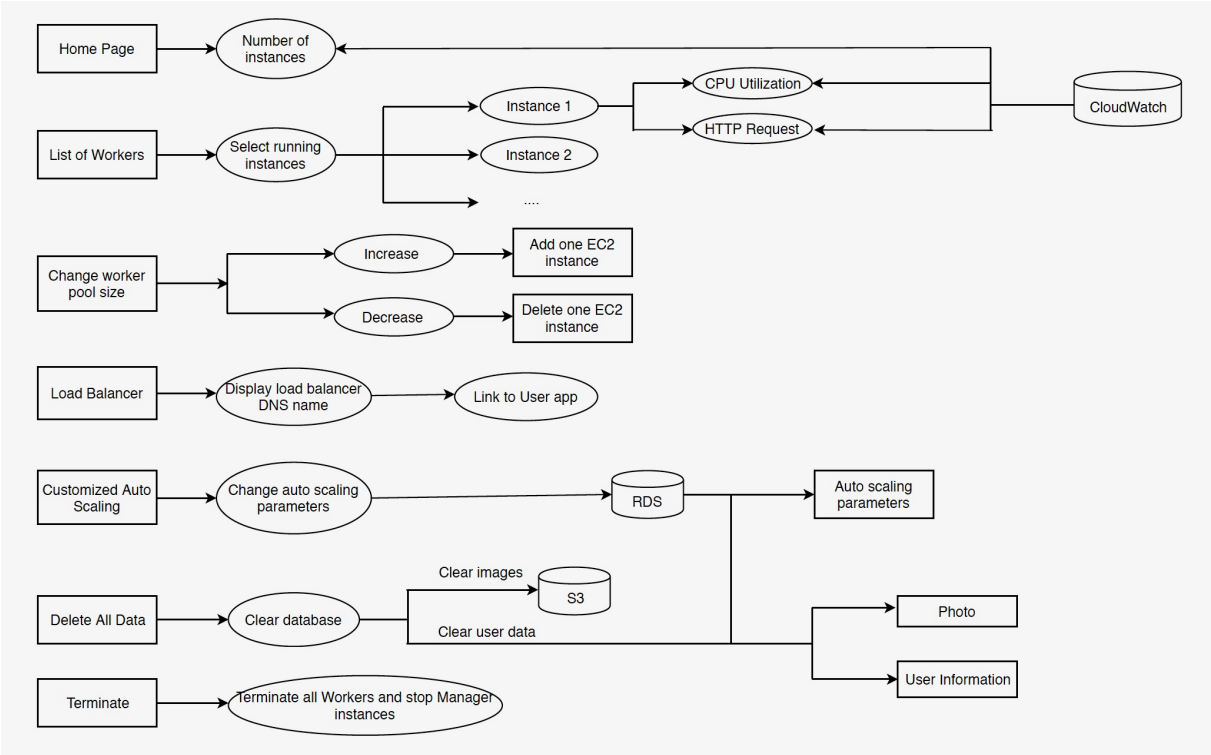

Figure 1: Flowchart for Manager App

4 Extension packages

1. Flask-SQLAlchemy: help manage a database
2. Matplotlib: plot the chart for the number of instances, CPU utilization, and HTTP request in the past 30 mins.
3. APScheduler: a Python library that schedules the Python code to be executed later, either just once or periodically. In our code, we put the auto-scaling check into schedules and execute once per minute.

5 Manager Instructions

1. After running start.sh on the Desktop folder in the virtual network computing (VNC), we can access the manager web app through localhost:5000 on VNC or through IPV4 address:5000 outside VNC.
2. Before first request to the manager web app, we resize the worker pool size to one. This means that once we enter the manager web app, it will start to check the length of the running instance, if the length exceeds one, then it will remove all the other instances except the oldest one. If there is no instance in the worker pool then add one new instance into the pool.
3. The auto scaler will be initialized immediately once the manager web app is requested. It will check the average CPU utilization of the worker pool, if the average value exceeds the given maximum threshold, it will expand the worker pool by the given expand ratio, if the average value is below the given minimum threshold, the auto scaler will shrink the worker pool by the given shrink ratio until the average CPU utilization lies in the range between maximum and minimum value.
4. Click on **'home'** to see the trend of number of workers during the past 30 minutes.
5. **'List of workers'** web page will show a table of all the running workers' info, including their instance ID, Keyname, image ID, Tag, instance type. Click on the **'details'** button, it will show one plot of CPU utilization and one plot of HTTP request in the past 30 minutes of each instance.

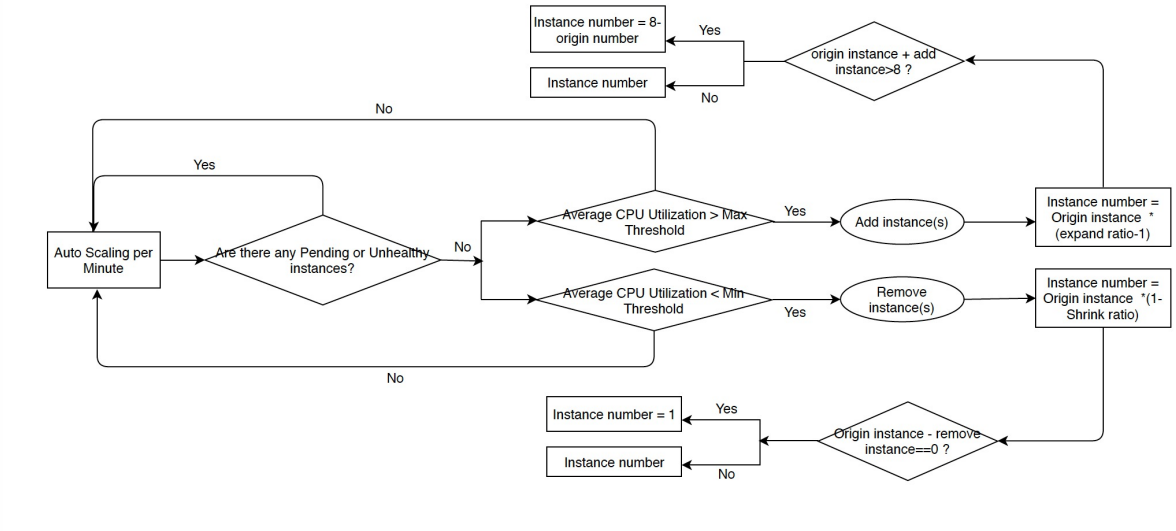

Figure 2: Flowchart for Auto-scaling Description

6. **'Change Worker Pool Size'** web page has one increase button and one decrease button, click on the increase button to manually add one new instance into the worker pool, click on the decrease button to manually remove one instance from the worker pool.

7. **'Load Balancer'** web page displays the user app URL which is the load balancer DNS name, clicking on the DNS name will lead to user app sign in page.

8. **'Customized Auto scale'** web page can customize the parameters of auto scaler, after completing four parameters chart and submit, the data will be transmitted to the AutoScale table in RDS database. Each time auto scaler is initialized, it will get the latest parameter data from the cloud database.

9. **'Delete All Data'** web page can clear all the data in user table and photo table in the RDS database and all images in S3 bucket. After all the data is cleared, manager will receive a message showing that it has finished the job.

10. **'Terminate'** web page can select all running workers in the pool, deregister them from the target group then terminate all of them. After terminating all the workers, the web app will stop the manager instance and flash the message on the page showing that it has completed the job.

6 System architecture

6.1 File description

1. __init__.py: initialization file for our program.

2. config.py: contains the essential information for the instances and load balancer.

3. auto_scaling.py: define 'auto_handler()' function:

   (a) Get the latest parameter data from the RDS database as auto scaler's parameters.

   (b) Get the length of the collection of all running and pending instances.

   (c) Get the length of the collection of all running instances.

   (d) Get the length of the collection of all healthy instances.

   (e) If all three length values are equal, then we will select all running instances and get the average CPU utilization. If the average CPU utilization is over the maximum threshold and the length is shorter than 8, then we expand our worker pool with the given expand ratio. However, if the number of new instances plus original ones is over 8, then we limit the total number of workers within 8. If the average CPU utilization is below the minimum threshold and the

length is longer than 1, then we shrink the worker pool size with the given shrink ratio. But if the number of original instances minus new instances is lower than 1, we will limit the number of instances to be removed and keep the number of running instances equals to 1.

(f) If all three length values are not equal, then we will print 'Stopped Auto Scaling!' message in terminal waiting for pending instances and unhealthy instances.

4. manager.py:

   (a) inst_CPU(inst_id): Using watch.get_metrics_statics(MetricName='CPUUtilization') function to get CPU utilization of the worker in the past 30 min from cloudwatch, time interval shifts by 1 min.

   (b) inst_HTTP(inst_id): Using watch.get_metrics_statics(MetricName='RequestCountPerTarget') function to get HTTP request rate of the worker in past 30 minutes, initialize HTTP request of each 1 min.

   (c) number_workers(): Using watch.get_metrics_statics(MetricName= 'HealthyHostCount') function to get the number of workers in the past 30 minutes from cloudwatch, time interval shifts by 1 min.

   (d) select_running_inst(): Using ec2.instances.filter() and image_id, instance_state_name, placement_group_name parameters to filter out a collection of all the running instances.

   (e) compare_inst(): Compare all three length values and return true or false.

   (f) average_CPU_uti(instances): Get the average CPU utilization in every two minutes and return the values for auto scaling.

   (g) full_load_check(instance_id): Check whether the number of workers is in the range (1-8).

   (h) inst_add(): add instance(s) for auto-scaling, when it checks the average CPU utilization is larger than the maximum threshold.

   (i) inst_remove(inst_id): remove instance(s) for auto-scaling, when it checks the average CPU utilization is smaller than the minimum threshold.

5. routes.py:

   (a) auto_check(): before the first request, we first check if any instance exists. If there is more than one instance in the worker pool, we resize to one by terminating other instances. If there is no running instance, we add one.

   (b) auto_scale(): auto_scaling is running simultaneously when the manager app begins to run.

   (c) home(): in route /home, get the number of running workers from number_workers(), and plot the chart for the number of workers in the past 30 mins.

   (d) worker_control(): in route /worker_control, redirect to worker_control.html

   (e) increase_workers(): increase an EC2 instance and register it into target group.

   (f) decrease_workers(): decrease an EC2 instance and deregister the instance from the target group.

   (g) view(): in route /worker_list/<instance_id>. For each instance ID, this function will return a CPU utilization and HTTP request for plotting.

   (h) worker_list(): in route /worker_list. The function select_running_inst() returns all running instances in the worker pool and displays their information on the page.

   (i) auto_modify(): in route /auto_modify. Get the auto-scaling parameter from the html form, and commit into the RDS database.

   (j) clear_all_data(): in route /clear_all_data. Clear all user data includes user information and upload photos that store in the S3 bucket and RDS database.

   (k) terminate_stop(): in route /terminate_stop. Terminate all workers and stop the manager instance.

6.2 Templates:

This folder contains all HTML templates referenced by each route.

1. base.html: it displays a top navigation bar for all pages.
2. home.html: it displays a figure to show the number of running instances in the past 30 minutes.
3. worker_control.html: it displays two buttons, one for adding a worker, one for deleting a worker.
4. workers.html: it displays a worker list table.
5. worker_list.html: it displays two figures, CPU utilization and HTTP request in past 30 minutes.
6. clear_all_data.html: it displays a success message for clearing all data.
7. terminate_stop.html: it displays a success message for terminating and stopping workers.
8. loadbalancerDNS.html: it displays a link accessing the user app.
9. auto_modify.html: it displays a table to get four inputs: threshold_max, threshold_min, ratio_expand and ratio_shrink.

7 Database Schema

This project connects the local mysql database from Assignment 1 with the RDS cloud database and creates an S3 bucket to store all the uploaded images in the input folder and output folder which reside in the static folder in S3. The screenshot figure 3 and figure 4 is shown as below.

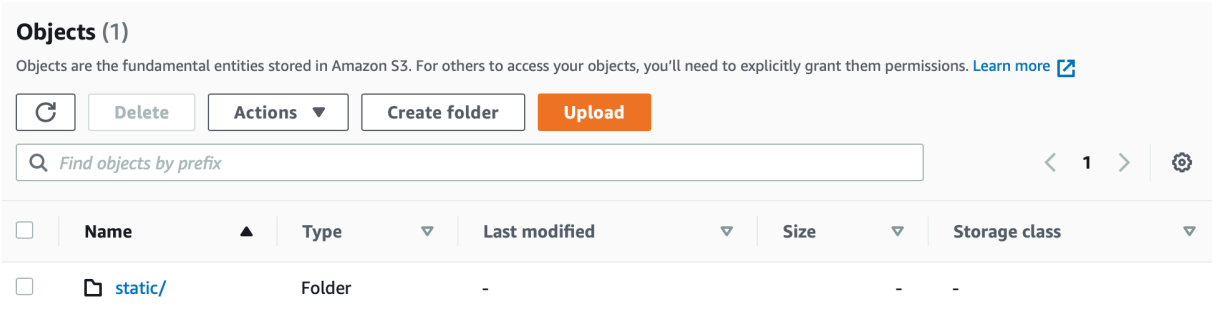

Figure 3: Flowchart for Auto-scaling Description

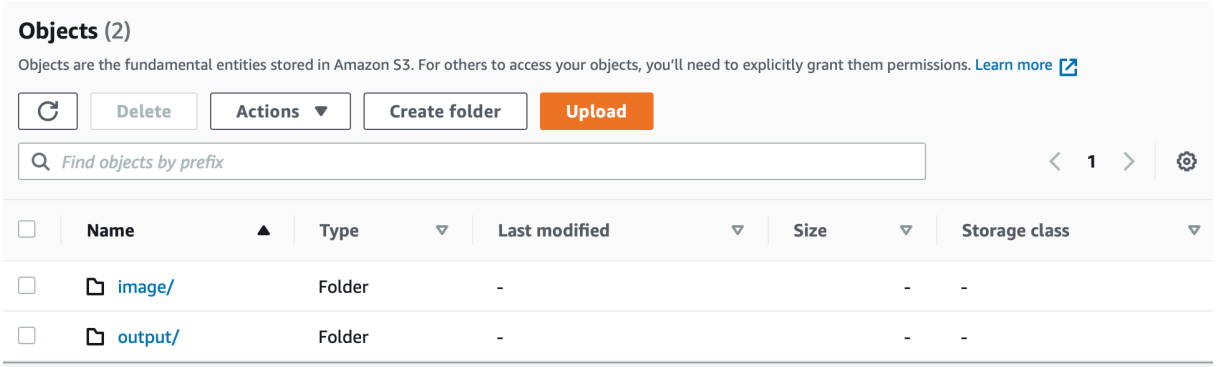

Figure 4: Flowchart for Auto-scaling Description

8 Result: Auto Scaling Algorithm Test

1. Do a stress test on one worker.
2. Inside the VNC terminal of the chosen EC2 instance, type the following codes and impose the CPU utilization to 100% for the next 15 minutes.
   $ sudo apt-get install stress
   $ sudo stress –cpu 2 –timeout 15m
3. Auto-scaling parameter: maximum threshold: 80, minimum threshold: 20, ratio expand: 2, ratio shrink: 0.5.
4. Observation:
   The table 5 below records the change of worker pool.

Time(min)	Running Instances	Healthy Instances	CPU Utilzation	Adding Instances	Terminating Instance	Auto Scaling Status
1	i-07e67035c4f6d30b8	i-07e67035c4f6d30b8	[13.83]			
2	i-07e67035c4f6d30b8	i-07e67035c4f6d30b8	[100]	i-0f3de1a9eb2efc720		
3	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-07e67035c4f6d30b8				Stop
4-8	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-07e67035c4f6d30b8				Stop
9-20	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	Around [100, 0.98]			
21	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	[86.67, 0.25]			
22	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	[0.33, 0.25]		i-07e67035c4f6d30b8	
23-27	i-07e67035c4f6d30b8 i-0f3de1a9eb2efc720	i-0f3de1a9eb2efc720				Stop
28	i-0f3de1a9eb2efc720	i-0f3de1a9eb2efc720	[0.25]			
29-40	i-0f3de1a9eb2efc720	i-0f3de1a9eb2efc720	Around[0.20]			

Figure 5: the Change of Worker Pool

   (a) In the first one minute, only one running instance is in the pool, then we impose its CPU utilization to 100%. At the second minute, the average CPU utilization is larger than the maximum threshold, 80. Based on the auto-scaling policy, the system automatically adds an instance into the pool. When a new instance first adds to the pool, its state is turning from pending to running, which will cost a few seconds. But the new instance will cost a long time to turn to a healthy state.

   (b) In the third second, there are two running instances and one healthy instance. Our policy will stop auto-scaling if the number of healthy instances is not equal to running instances. So that begins from 3 min to 8 min, our auto-scaling stops and waits for the instance to turn to health.

   (c) From 9 min to 20 min, there are two running and healthy instances, in each minute, the CPU utilization is continuously counting, which means the average CPU is continuously compared with the maximum threshold.

   (d) At 21 min, we stop the stress test and the CPU utilization begins to come down.

   (e) At 22 min, the average CPU threshold is smaller than the minimum threshold. So that auto-scaling removes an instance from the pool.

   (f) When an instance is removed, it will first turn unhealthy but it is still in the running state. So that the running instances are not equal to healthy instances, the auto-scaling stops to run.

   (g) From 28 min to 40 min, one instance keeps running, and auto-scaling continuously to count the average CPU utilization and do the comparison with the threshold.

5. Conclusion: From the test above, we deliberately make one of the running instances' CPU reaches 100% utilization, since our CPU maximum threshold is below 100, our auto scaler doubles the number of instances to balance the CPU utilization. As we can see in the table, after a period of full loaded time, the number of instances is still not changed, which means the CPU utilization for

that instance goes back to the normal range. What's more, when the CPU utilization drops below the minimum threshold, the auto scaler terminates half instances and always keeps the number of instances no less than one. This stress test shows that our policy can converge in a long time period. The number of instances in the pool will not explode or be eliminated, it will eventually reach a stationary state, which means our model is converged. The number of workers during the press test (figure 6) is shown below.

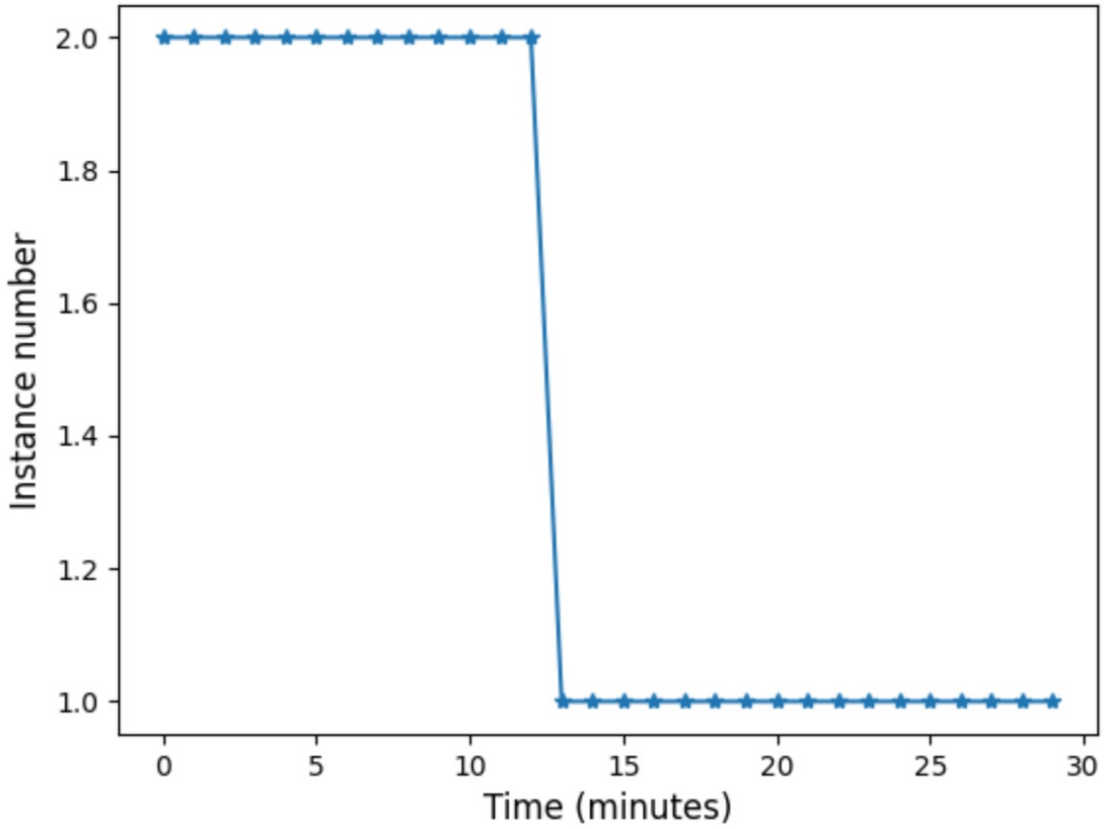

Figure 6: Number of Workers

9 Group work structure

Basically, we did all of this project together, the following structure is just a roughly divided version:

- Shuya Wang: EC2 instance, RDS database, S3 Creation, instance create and decrease.
- Jiani Jia: UI Design, Documentation, plot all the charts displayed on the web.
- Xiaoxi Yu: Documentation, auto scaling backend code, terminate and clear data.

10 Manager UI Design

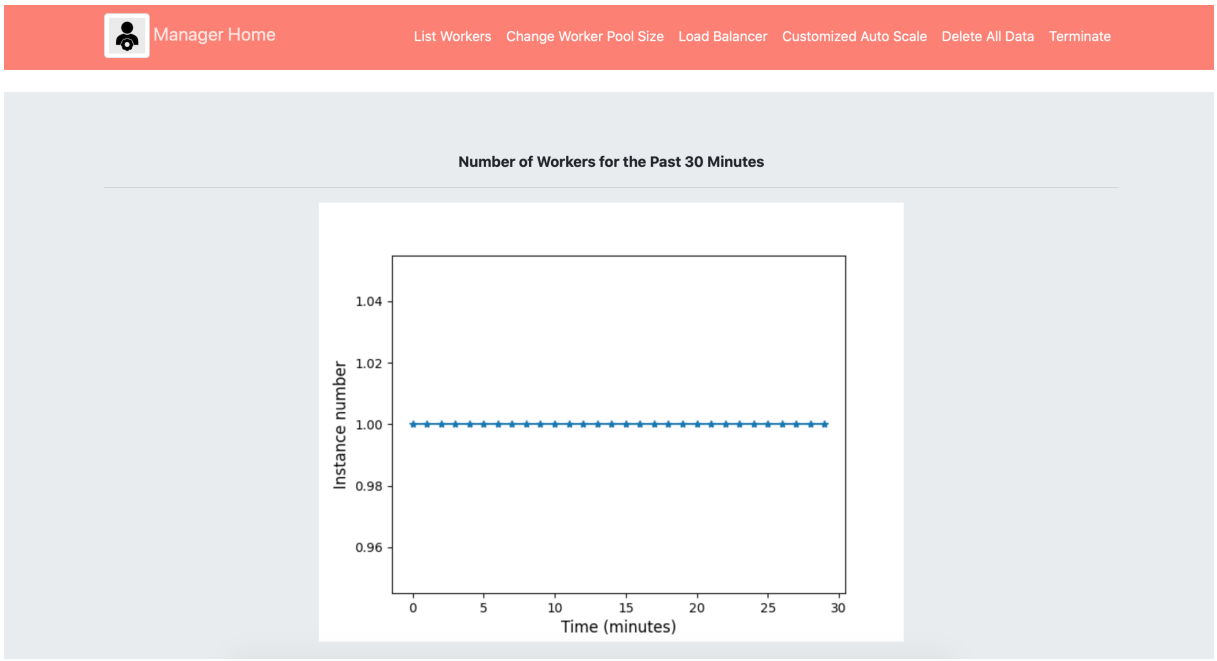

Figure 7: Home Page

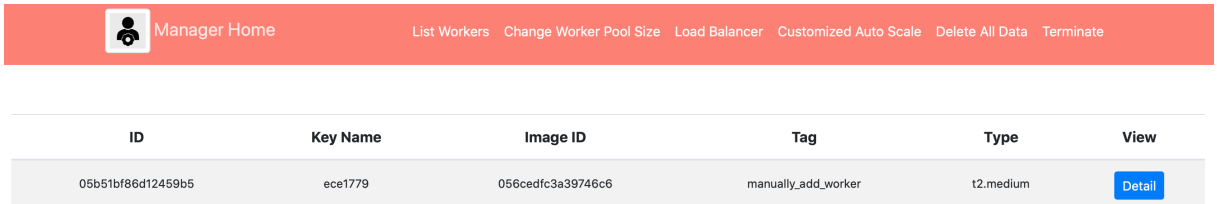

Figure 8: Worker List

Figure 9: CPU utilization figure and Http request figure

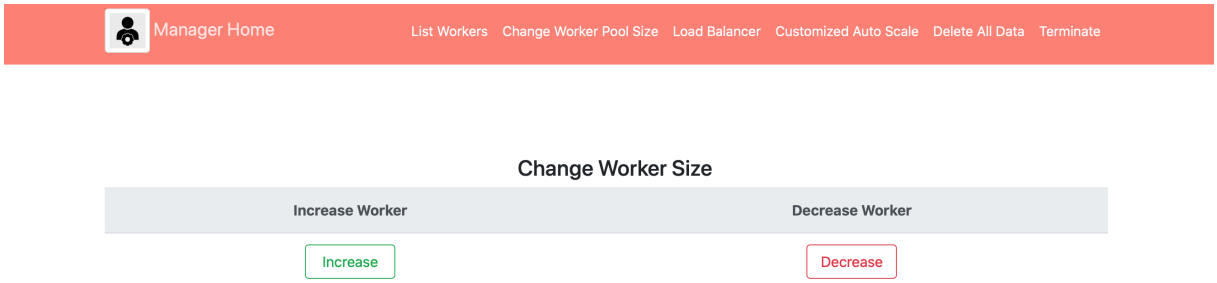

Figure 10: Change Work Pool Size

Figure 11: Load Balancer Page

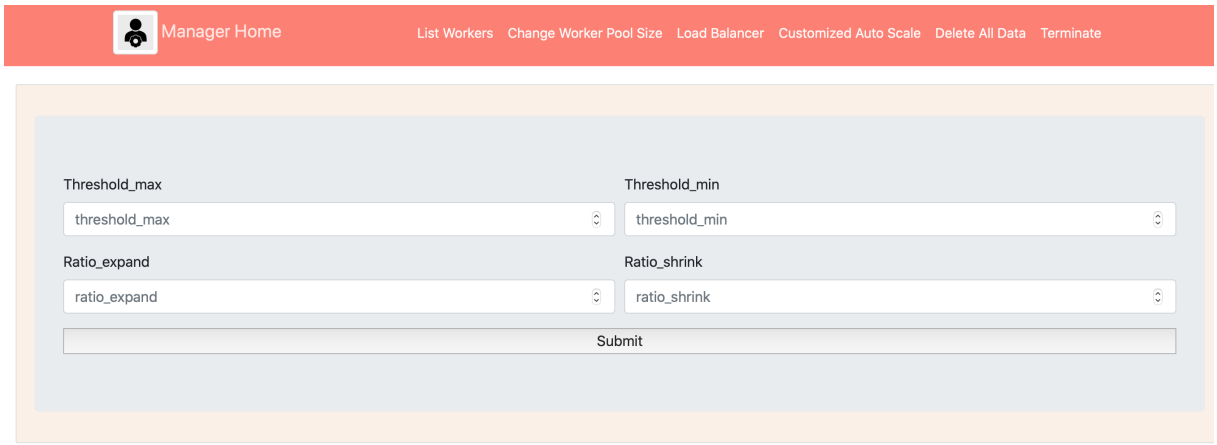

Figure 12: Customized Auto Scale Input Table

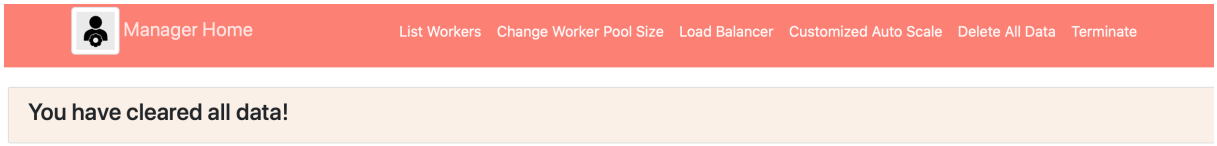

Figure 13: Clear Data Page

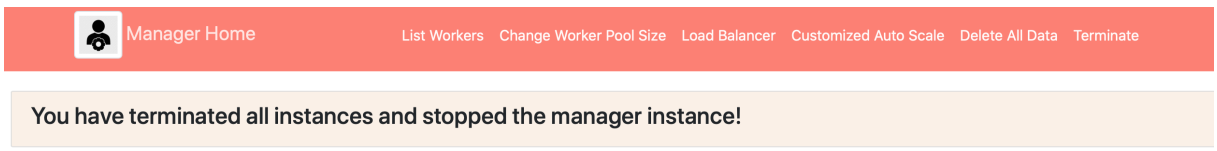

Figure 14: Stop and Determinate Instances Page