

# ECE1779 Assignment 2:

## Dynamic Resource Management

### 1. User Guide

#### 1.1 Menu Bar

The menu bar is the most commonly used navigation tool for users.



*Figure 1. Menu bar*

As figure 1, there are six different buttons in total. Each of them would perform certain functionalities or lead users to a web page. Details of different web pages would be described in the next section.

1. “Home” button will lead users to the “Home” page which provides a brief introduction to the functions we manage in this project. Users can also get access to these functions by clicking the name of the function in “Home” Page.
2. “Worker List” button takes users to the “Worker List” page which enable users to create or destroy instances. Users can also monitor some details of instances such as CPU utilization rate and HTTP request rate.
3. “ELB DNS” button takes users to return to user app.
4. “Auto Scale” button will users to auto scale page where they can tune the scaling policy.
5. “Delete” button can let users delete the application data stored on the database as well as all images stored on S3.
6. “Stopping” button can help users to terminates all the current working workers and then stops the manager itself.

## 1.2 Web Pages

The application contains a few different web pages allowing users to interact with the system. How each web page functions is detailed below.

1. “Home” page is the place that a user gets started. It contains a simple welcome message for everyone. As shown in Figure 2, there is also some brief introduction to the functions we managed in this app. Users will be directed to correspond function page once they click the name of the function in its introduction.



### Welcome! Manager

**This app is developed for the manager to monitor and control the worker pool.**

**To use, you can directly click on their function name in their descriptions or you can click them in the navigation bar.**

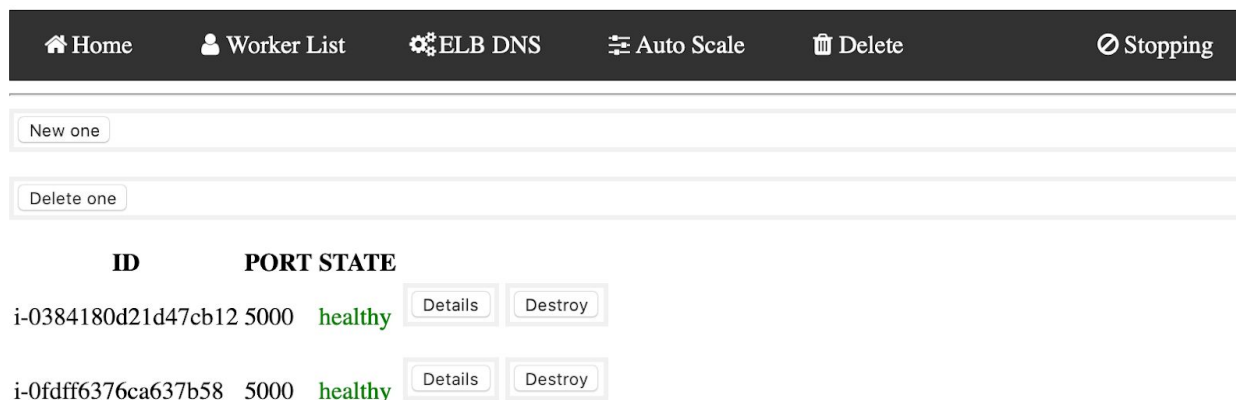
You can press [Worker List](#) to monitor and control current working workers

Or you can press [ELB DNS](#) to .....

You can press [Delete](#) to delete application data stored on the database as well as all images stored on S3.

You can press [Stopping](#) to terminates all the workers and then stops the manager itself.

*Figure 2. Home Page*



*Figure 3. Worker List Page*

- As figure 3, “Worker List” page lists the current existing workers. Users can also create and delete instances by clicking the “New one” button and “Delete one” button. Users can also specify which worker they want to delete by clicking the “Destroy” button after the worker. As figure 4, after clicking “Detailed” button, some detailed information such as of that worker CPU utilization rate and HTTP request rate will be displayed on the screen.



Figure 4. More detailed information of a worker

- “ELB DNS” can take users back to user app.



Figure 5. ELB DNS button

4. As figure 6, “Auto Scale” Page will first display the current autoscaling policy. More specifically, it will show users the value of CPU grow threshold, CPU shrink threshold, grow ratio and shrink ratio. In the next window, users can tune their desired scaling policy by adjusting the value of CPU grow threshold, CPU shrink threshold, grow ratio and shrink ratio. Then users need to click “update” button to actually apply the policy.

[Home](#) [Worker List](#) [ELB DNS](#) [Auto Scale](#) [Delete](#) [Stopping](#)

- Policy successfully updated

**Current Autoscaling Policy:**

cpu grow threshold: **10.4**  
cpu shrink threshold: **2.3**  
grow ratio: **3.4**  
shrink ratio: **5.5**

**You can change autoscaling policy here**

CPU Grow Threshold  (0.1 ~ 99.9)  
CPU Shrink Threshold  (0.1 ~ 99.9)  
Grow Ratio  (input must > 1)  
Shrink Ratio  (input must > 1)

Update

Figure 6. Auto Scale Page

5. “Delete” button will call clear function and delete application data stored on the database. as well as all images stored on S3.



Figure 7. Delete Button

6. “Stopping” button will call shutdown function and terminates all the workers and then stops the manager itself.



*Figure 8. Stopping Button*

## 2. Developer Guide

### 2.1 Architecture

There are mainly five routes in this manager web applications: Home, ELB DNS, Autoscale, Delete and Stopping.

#### 2.1.1 Navigation

This application does not require users to log in. Therefore, all functions will be available for users at the beginning. There are mainly 3 ways to navigate around, you can either:

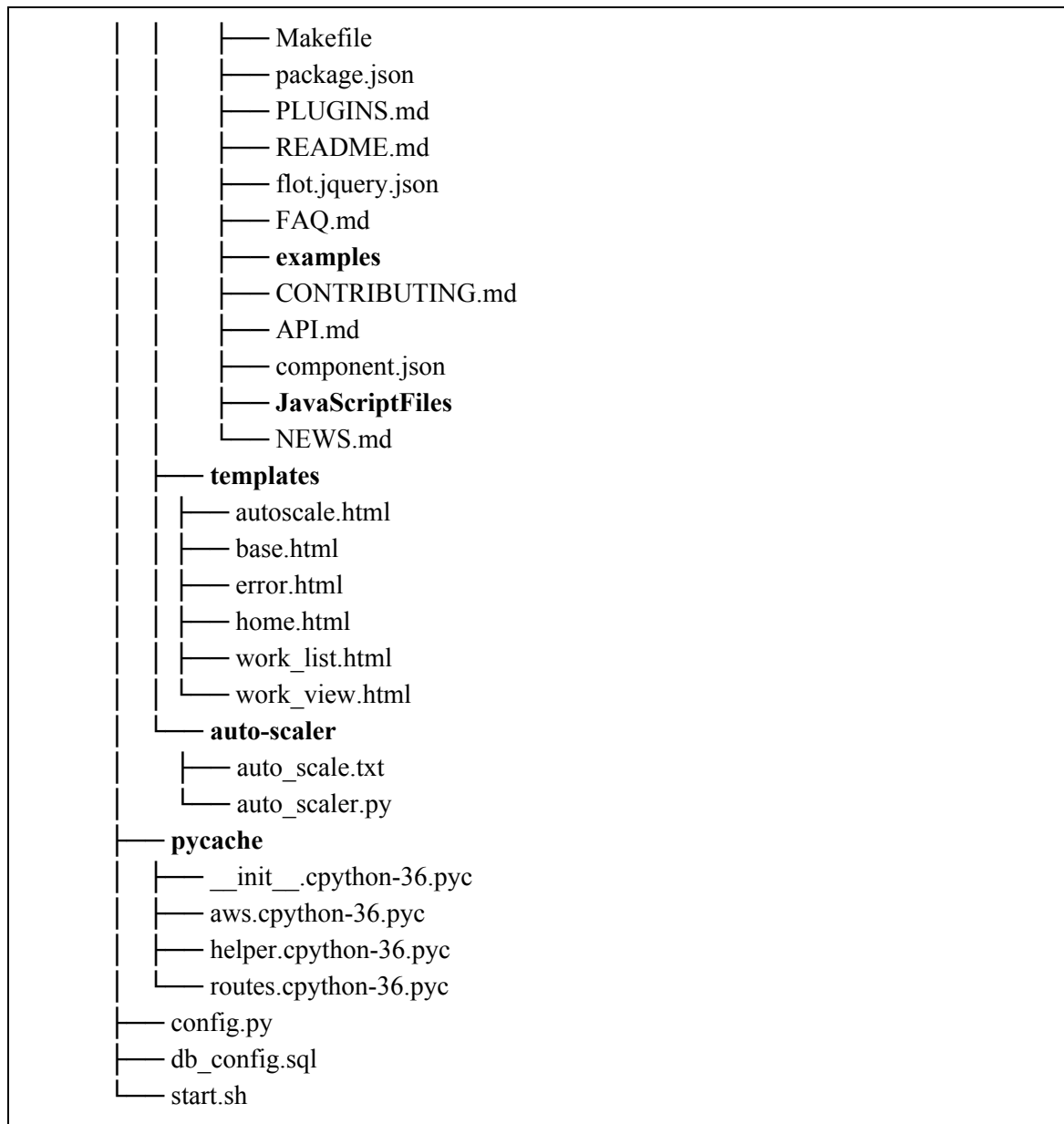
- 1) Use the navigation bar containing various web links at the top of each web page.
- 2) Click the name of the function that you want to call before its description in “Home” page.
- 3) Type in URL directly.

#### 2.1.2 Code Structure

The structure of the code of this web application is as shown below:

```
A2 (top level folder)
  -README.md
  -user_app
    |— A1.py
    |— README.md
    |— app
      |— __init__.py
      |— helper.py
      |— models.py
      |— routes.py
      |— utils.py
```





As the structure in the user app didn't change as in A1. We would only briefly describe the structure in the manager app.

#### Top level directory

**run.py** and **start.sh** can be used to run the application. Command to use is `"/start.sh"` (also starts the auto-scaler) or `"python3 run.py"` (without auto-scaler).

**config.py** defines environment variables used by Flask, including AWS credentials, ELB target group arn, template ID used to create new workers, etc.

### “app” folder

**\_\_init\_\_.py** contains the initialization of the Flask instance, the database and the AWS client used for our web application, which are configured using environment variables defined in **config.py**. It also adjusts the worker pool size to 1 and initializes **auto\_scale.txt** described below.

**aws.py** contains **AwsClient** class, which contains EC2, S3, ELB and CloudWatch clients and handles most of the AWS API calls. These includes add workers, terminate workers, grab metrics, list current targets, etc.

**routes.py** contains routing and action functions that tells Flask what URL will trigger what to happen. This file heavily depends on **aws.py**. They are the core of the functionality of the manager application, which will be detailed in the **Developer Guide - Function and Implementation Considerations** section below.

### “static” folder

A folder contains some JavaScript dependencies. It is mainly used to support the plotting of CPU utilization and HTTP request rate charts.

### “template” folder

This folder is where all html files being stored.

### “auto-scaler” folder

**auto\_scale.txt** is a CSV file stores auto-scaler configurations. It is managed by the manager app and stores the thresholds and ratios. It also stores another bit to indicate whether the auto-scaler is enabled, so that we could ensure that the auto-scaler won't randomly starts to do things based on nonsense initial values, such as zeros for all parameters.

**auto\_scaler.py** is the auto-scaler itself. It reads data from **auto\_scale.txt** and decides whether or not to add or cut some workers. It also depends on **aws.py** and has some core functionalities, which will be detailed in the **Developer Guide - Function and Implementation Considerations** section below.

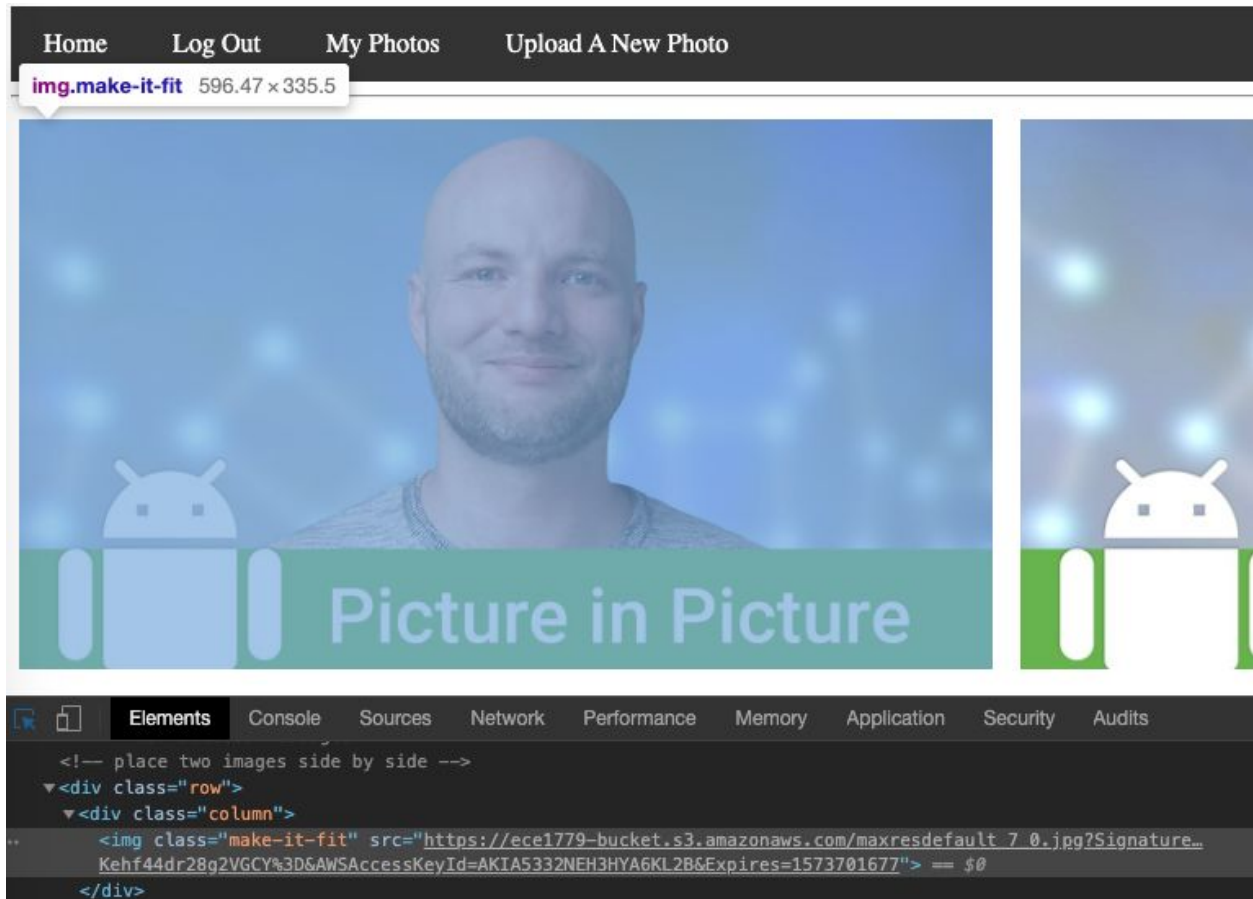
## **2.2 Function and Implementation Considerations**

### **2.2.1 User App Modification**

Basic functionalities of the user app are unchanged. It still performs exactly the same as it did in A1. Some modifications and updates are introduced to make the application compatible with AWS RDS and S3. We created a database in RDS using the same configuration as in A1. The database structure is discussed in the next section. We also created a bucket in AWS S3 to store

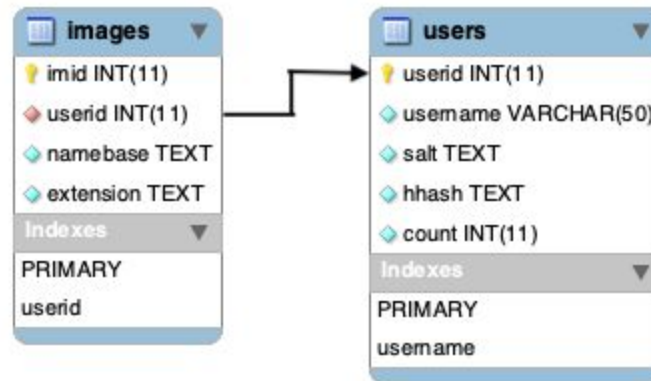


any images uploaded by users and things like thumbnails generated by the system. One major update here is that we use Boto3 to generate presigned URLs in order to access those images in S3, as shown in Figure 9 below. This could also help us prevent images from being accessed by users other than their owners. Another major update is that we add an HTTP request metric publisher in every endpoint. Thus, we could publish each request as a data point to CloudWatch for the manager app to query and use later.



*Figure 9. An image from a presigned URL*

### 2.2.3 Database



*Figure 10. Database structure*

The database contains two tables: users and images. The schema is shown in Figure 10 above. Details of the two tables are described below.

Table “users” stores user information. Column “userid” is the primary key. Each user also has a unique “username”. A per-user salt value would be stored in column “salt”. Column “hhash” stores hash value of (salt + password). Therefore, the system could easily verify a user’s password without storing it in clear text. Column “count” tracks how many photos a user have uploaded so far. By appending a user’s “userid” and “count” to each of his/her images, the system could ensure that no duplicate names are shared by two images.

Table “images” stores information of every image. Column “imid” is the primary key. Column “namebase” stores names, excluding extensions, of every image. As described above, a “namebase” is a combination of an image file name, its owner’s “userid” and “count”. Therefore, the system can ensure that no duplicate “namebase” across the whole system. Column “extension” stores file extensions. Column “userid” is a foreign key referencing “userid” in table “users”. It can be used to retrieve an image's owner’s information, ensure an image can only be viewed by its owner and ensure that only existing users can upload images.

Since in both tables, every field depends only on the primary keys and there is no transitive dependencies, the database is 3NF normalized.

### 2.2.4 Manager Application Functions

List Workers: With the ELB client included in the AwsClient, the application could directly query ELB for currently registered workers and their status.

View Worker Details: With a worker instance ID and the CloudWatch client included in the AwsClient, the application could directly query CloudWatch for CPU utilization and HTTP request rate metrics for the past 30 minutes. CPU utilization is statistically computed using “Average” and HTTP request is statistically computed using “Sum”. Both in a resolution of 1 minute. Then, the plotting work is done by JavaScript.

Grow Worker(s): With the EC2 and ELB clients included in the AwsClient, the application could create new instance(s) and register them on ELB. A launch template is ready in the AWS console for the application to use, which includes all parameters needed like instance type, AMI to use, etc. It also includes a user data section, which acts as a bash script so that a newly created worker instance would automatically run the user app. Therefore, to create a new worker instance, the application first ask EC2 to create one and get the instance ID of it. Then, wait for some time until the instance is properly running, which can be done by continuously querying EC2 for instance status. Then, ask ELB to register the new worker. To create multiple workers, just repeat the process above for multiple times.

Terminate and Deregister Workers: With the EC2 and ELB clients included in the AwsClient, the application could terminate any worker needed. Once a worker is terminated, ELB would automatically deregister it. To terminate a random worker, the application asks ELB for the list of current target instances, gets the ID of the first one and asks EC2 to terminate it. To terminate a specific one, the application asks EC2 to terminate it directly by providing the instance ID. To terminate multiple instances, the application first asks ELB for the list of current target instances, gets as many as IDs needed, then asks EC2 to terminate them at once rather than terminating them one by one. Please be noted that, before any termination, the application would wait for a few seconds, provided that an ongoing request won't take any longer than that, to terminate any worker gracefully.

Change Auto-Scaling Policy: The application provides users a place to change the auto-scaling policy. It also checks if the input is valid. For example, a threshold to shrink greater than the threshold to grow would not be allowed. Once a user submits the form, the application would write all parameters into the configuration CSV file mentioned above. Also, it changes the enable bit to 1 to enable the auto-scaler. Each time a user changes the policy, the application would overwrites the existing configuration file. Once the application is shutdown or restarted, it overwrites the configuration file to all zeros to indicate the policy is reset and the auto-scaler is disabled.

Shutdown: The application would issue a system call to kill the process on the listening port, which is 5000 in this case. Before this happens, it would terminate all workers and overwrite the auto-scaler configuration file to all zeros to reset and disable the auto-scaler.

Clear: With S3 client included in the AwsClient and RDS database client, the application could clear all data stored in S3 and RDS. To clean up everything in S3, the application first asks S3 for the list of all objects, gets objects' keys and ask S3 to delete them one by one. To empty the RDS database, the application first tell RDS to turn off the safe update mode, otherwise it would be very inconvenient. Then, ask RDS to delete all rows in both tables and turn the safe update mode back on.

Auto Scale: The auto-scaler runs as a separate part of the manager application. It reads parameters from the configuration file every minute and decides whether to change the worker pool size. Unlike functions in the manager app, which don't have any restrictions on the worker pool size, the auto-scaler is restricted to scale the number of workers between 1 and 10. It first reads the enable bit to know whether it should work. If enabled, it uses CloudWatch client in AwsClient to read CPU utilization data of all workers for the past two minutes. Then, it computes the average and compares it to the threshold parameters. If it needs to change the worker pool size, it computes how many workers need to be added/cut according to ratio parameters. Then, it adds workers or terminates them similarly as the manager application does. If it successfully grows/shrinks the worker pool size, it would wait 5 minutes instead of 1 minute before the next round. Since the CPU utilization is always very high for the first two minutes after a new instance is created and the auto-scaler reads CPU utilization of the past two minutes, it is reasonable to wait for 5 minutes to make sure new instances are stable. This also helps to prevent oscillation. Please be noted that the auto-scaler is disabled does not mean it is not running. You can always enable it by changing the auto-scaling policy. The auto-scaler is disabled at the time the manager app starts or shuts down. This design is to ensure that the auto-scaler won't behave abnormally if the manager app is not running. Also, to completely turn off the auto-scaler, you may need to find out its pid and kill it directly.

## **2.3 Testing**

### **2.3.1 Unit Test**

As the test we did in A1, "Postman" is still used to send different requests to each endpoint so that we can test them separately. "Postman" enable us to test only the backend logic without frontend and we could bypass frontend to test server side checks for corner cases.

### **2.3.2 Test Locally and on Cloud Environment**

After the unit test, we next test the actual functionality of the application and the usability of the UI. We first let the application running on "localhost:5000" and test every function we managed. We then migrate the whole application to our EC2 instance. Figure 11 below shows an example

of successfully changing autoscaling policy. In the end, we test the functionality of delete and stopping to make sure they work correctly.

[Home](#) [Worker List](#) [ELB DNS](#) [Auto Scale](#) [Delete](#) [Stopping](#)

- Policy successfully updated

### Current Autoscaling Policy:

cpu grow threshold: **9.9**  
cpu shrink threshold: **5.8**  
grow ratio: **2.0**  
shrink ratio: **3.0**

### You can change autoscaling policy here

CPU Grow Threshold  (0.1 ~ 99.9)

CPU Shrink Threshold  (0.1 ~ 99.9)

Grow Ratio  (input must > 1)

Shrink Ratio  (input must > 1)

Update

Figure 11. Successfully change the autoscaling policy

## 2.4 Results

We would mainly discuss the test of our auto-scaler and its performance in this section. In this test case, the policy is 20 for the CPU threshold to grow, 10 to shrink and both ratios to grow and shrink are 2. We started with 1 worker. And then, we used a script to continuously curling the ELB endpoint in a frequency of 2 times per seconds. After a few minutes, we doubled the frequency. Then, after a few minutes, we stopped curling. The output of the auto-scaler is shown in Figure 12. Although the log says it waited for 3 minutes after every success, it is a typo here. It actually waited for 5 minutes. Figure 13 shows the plot of time versus worker pool size. We could see from the output that starting from 1, the auto-scaler first doubled the worker pool size to 2. After 5 minutes, it was already stable. After a few minutes, we doubled the HTTP request rate. The auto-scaler again doubled the worker pool size and it was stable for a few minutes. Then, as the HTTP request rate drastically dropped, the auto-scaler halved the worker pool size.

```

((cloud_computing) bash-3.2$ python3 auto_scaler.py
INFO:root:Avg CPU util is 27.33333333333333
INFO:root:Current worker pool size is 1
WARNING:root:Grow 1 instances
WARNING:root:Status are [200]
INFO:root:Grow or shrink successfully, wait for 3 min
INFO:root:Avg CPU util is 13.74936324905067
INFO:root:Current worker pool size is 2
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 17.043785310734464
INFO:root:Current worker pool size is 2
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 22.289640640918762
INFO:root:Current worker pool size is 2
WARNING:root:Grow 2 instances
WARNING:root:Status are [200, 200]
INFO:root:Grow or shrink successfully, wait for 3 min
INFO:root:Avg CPU util is 14.481681419441886
INFO:root:Current worker pool size is 4
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 14.626126304925963
INFO:root:Current worker pool size is 4
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 14.734086187962253
INFO:root:Current worker pool size is 4
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 10.950912290451086
INFO:root:Current worker pool size is 4
WARNING:root:Nothing changes
INFO:root:Avg CPU util is 4.671919448524071
INFO:root:Current worker pool size is 4
WARNING:root:Shrink 2 instances
WARNING:root:Status are [None, None]
INFO:root:Grow or shrink successfully, wait for 3 min

```

Figure 12. Auto-scaler output

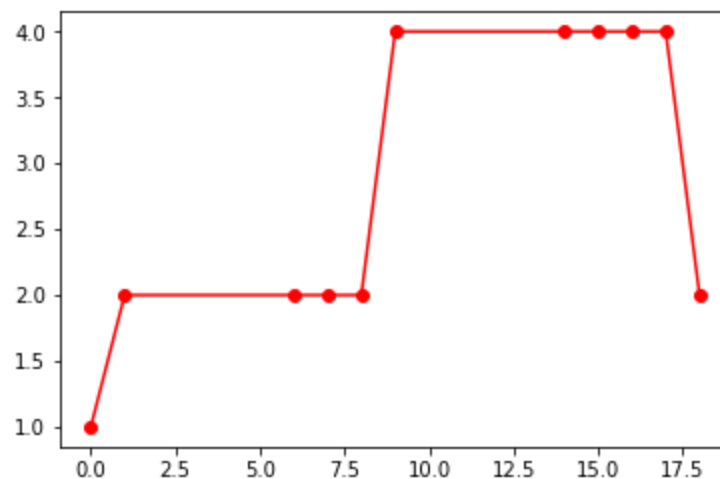
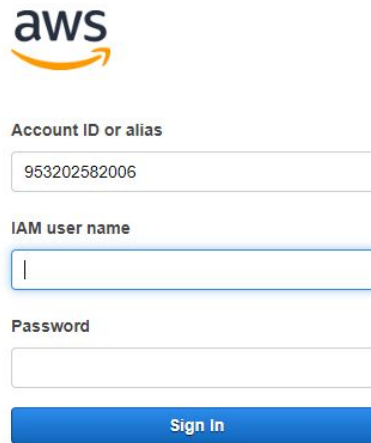


Figure 13. Auto-scaler plot

## 2.4 How-Tos

### 2.4.1 AWS Console Login

Go to <https://953202582006.signin.aws.amazon.com/console> (as shown in Figure 14 below) and use the IAM credentials provided in “credentials.txt” to sign in.

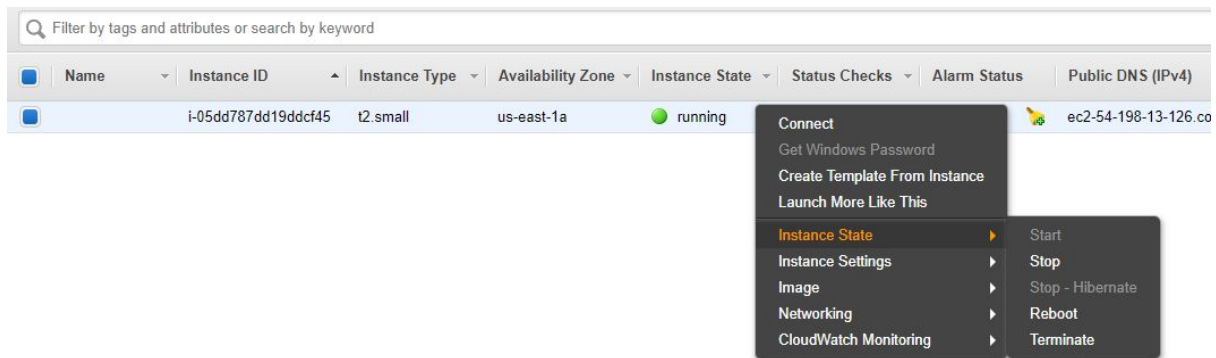


The image shows the AWS IAM login panel. At the top is the AWS logo. Below it are three input fields: "Account ID or alias" with the value "953202582006", "IAM user name" with a cursor, and "Password". At the bottom is a blue "Sign In" button.

*Figure 14. AWS IAM login panel*

### 2.4.2 Start/Stop the Manager Instance

After logged in, navigate to “Services -> EC2 -> Instances”. Please make sure the location is “US East (N. Virginia)”. You should see the manager instance with the ID “i-0da352aa0116b76df”. Then, you could right click it to start/stop the instance, as shown in Figure 15 below.



*Figure 15. Start/Stop the instance*

### 2.4.3 Connect to the Instance

You could SSH into the instance (`ssh -i keypair.pem ubuntu@Public IP`). The public IP of the instance is shown in the “Public DNS” field in Figure 15 above. A private key file “keypair.pem” is provided. You may need to give correct permissions to the file in order to use it if you are on Linux or Mac OS. Use command “`chmod 400 keypair.pem`” to do so.

#### **2.4.4 Run the Manager Application**

Navigate to “/home/ubuntu/Desktop/”, which should contain a shell script “start.sh”. Open terminal and simply type “./start.sh”. It starts the manager application on “0.0.0.0:5000” as well as the auto-scaler. Now, the application can be accessed anywhere in the world at “Public IP:5000”. Please be noted that the manager application is running in background. Shutting down the manager application doesn’t necessarily mean the auto-scaler would be shutdown too. Therefore, you may still need to manually shutdown the auto-scaler by finding its “pid” using “top” command.

#### **Team Member**

Shi, Hu (1001416498, rock.hu@mail.utoronto.ca)

Jian, Shi (1001841536, jianisme.shi@mail.utoronto.ca)