

ECE 1779: Project 2

Group Members:

Corey Kirschbaum, #: 997384165

Mohammed Baquir, #:1007489816

Date: November 15, 2020

Contents

Table of Figures	4
Glossary	6
1. Introduction	7
2. Face Mask Detect Web Application	8
2.1. File Directory Layout Changes	8
2.2. Web Application Log-in	9
2.3. EC2 AMI and Start-Up Script	9
2.3.1. Start-up Script	9
2.3.2. EC2 Worker Details	10
2.4. Amazon S3	11
2.5. Amazon RDS and Database Design	12
2.5.1. Amazon RDS	12
2.6. Testing Endpoints	14
2.6.1. Register - /api/register	14
2.6.2. Upload - api/upload	14
2.6.3. Register - /api/cpuTest	14
3. Load Balancer	15
4. Manager Web Application	18
4.1. File Directory Layout	18
4.2. Flask Web Application	20
4.3. Manager Home Page	20
4.4. Manager Worker Page	21
4.4.1. Worker List	21
4.4.2. Worker Details	21
4.5. Manager Scaler Metrics Page	22
4.6. Database Design	24
4.7. How to run the app	24
5. Auto-Scaler	25
5.1. Design	26
5.2. Results	27
5.2.1. No CPU Load	28

5.2.2.	Upload Rate of 8	29
5.2.3.	Upload Rate Increase to 10	30
5.2.4.	Upload Rate Increase to 100	31
5.2.5.	Upload Rate Decrease to 10	33
5.2.6.	Upload Rate Decrease to 8	34
5.2.7.	Back to no CPU Load	35
6.	AWS Information	36
7.	IAM ROLES	37
Future Improvements		38
Group Member Efforts		38
Corey Kirschbaum		38
Mohammed Baquir		39
References:		39

Table of Figures

Figure 1: Overview of the face mask application.....	7
Figure 2: AMI-ID of workers.....	10
Figure 3: S3 Permissions	11
Figure 4: S3 Bucket Policy	11
Figure 5: S3 Access Control List.....	12
Figure 6: RDS Face Mask Detect Database Info - Part 1	13
Figure 7: RDS Face Mask Detect Database Info - Part 2	13
Figure 8: Face Mask Detect Database Schema	14
Figure 9: Load Balancer Security Group.....	16
Figure 10: Load Balancer Target Port.....	17
Figure 11: Load Balancer Target Setting 1	17
Figure 12: Load Balancer Target Setting 2 - Current	18
Figure 13: HealthyHostCount Cloud Metric.....	20
Figure 14: Manager App Main Page.....	20
Figure 15: Manager App Worker List	21
Figure 16: Manager App Worker Details	22
Figure 17: Manager App Auto-Scaler Metrics Policy Form Page	23
Figure 18: Manager App Auto-Scaler Policy Form Validation.....	23
Figure 19: Manager App Database Schema.....	24
Figure 20: Manager App RDS Database Info.....	24
Figure 21: Manager App start.sh Script Running Processes	25
Figure 22: Auto-Scaler Grow Logic.....	26
Figure 23: Auto-Scaler Shrink Logic	26
Figure 24: Example of a Control System Response Plot [1]	28
Figure 25: Result Plots for No CPU Load	29
Figure 26: Auto-Scaler Results Log for No CPU Load	29
Figure 27: Result Plots for Upload Rate 8 - Increasing.....	29
Figure 28: Auto-Scaler Results Log for Upload Rate 8 - Increasing.....	30
Figure 29: Result Plots for Upload Rate 10 - Increasing.....	30
Figure 30: Auto-Scaler Results Log for Upload Rate 10 - Increasing.....	31
Figure 31: Result Plots for Upload Rate 100 - Increasing.....	32
Figure 32: Workers Status at Upload Rate 100 - Part 1	32
Figure 33: Workers Status at Upload Rate 100 - Part 2	32
Figure 34: Workers Status at Upload Rate 100 - Part 3	33
Figure 35: Auto-Scaler Results Log for Upload Rate 100 - Increasing.....	33
Figure 36: Auto-Scaler Results Log for Upload Rate 10 - Increasing - Worker Averages.....	33
Figure 37: Result Plots for Upload Rate 10 - Decreasing	34
Figure 38: Auto-Scaler Results Log for Upload Rate 10 - Decreasing	34
Figure 39: Result Plots for Upload Rate 8 - Decreasing	34

Figure 40: Auto-Scaler Results Log for Upload Rate 8 - Decreasing	35
Figure 41: Result Plots for No CPU Load - Decreasing	35
Figure 42: Auto-Scaler Results Log for No CPU Load - Decreasing	36
Figure 43: Manager App AWS Info /app/config.py.....	37
Figure 44: Manager App And Workers IAM Roles	37

Glossary

S3	Amazon Simple Storage Service
EC2	Amazon Elastic Compute Cloud
AMI	Amazon Machine Image
ECE	Electrical and Computer Engineering
AWS	Amazon Web Services
App	Application
ELB	Amazon Elastic Load Balancer

1. Introduction

The following report is for ECE1779 Cloud Computing Project 2. The project builds off project 1, a single EC2 web server running a web application which allows users to login and upload images for which to apply a face mask detection algorithm on. The face mask detection page on the web application will process a user's specified image and display that image plus the processed image containing a green box around faces wearing masks and a red box around faces not wearing masks. Additionally, the metrics numbers used to generate these coloured boxes are presented. The system tracks each user's uploads and allows users to view their upload history.

The pieces added for project 2:

A load balancer has now been added to handle user requests, forwarding requests to multiple EC2's running the face mask detect web application. Additionally, the face mask detect web application is now storing image files to an S3 to ensure all EC2's can access all the images and to ensure that the EC2's is stateless. The same idea is also applied to the database, using AWS RDS to store our MySQL database.

A manager app has been developed to run on a single EC2 instance which will show information and statistics related to the load balancer and the running EC2 instances for the face mask detection web application. The manager application web page will allow for manual addition/removal of EC2 instances running the face mask detection web app and will take in policy preference for auto-scaling the EC2's and have a background task perform the policy.

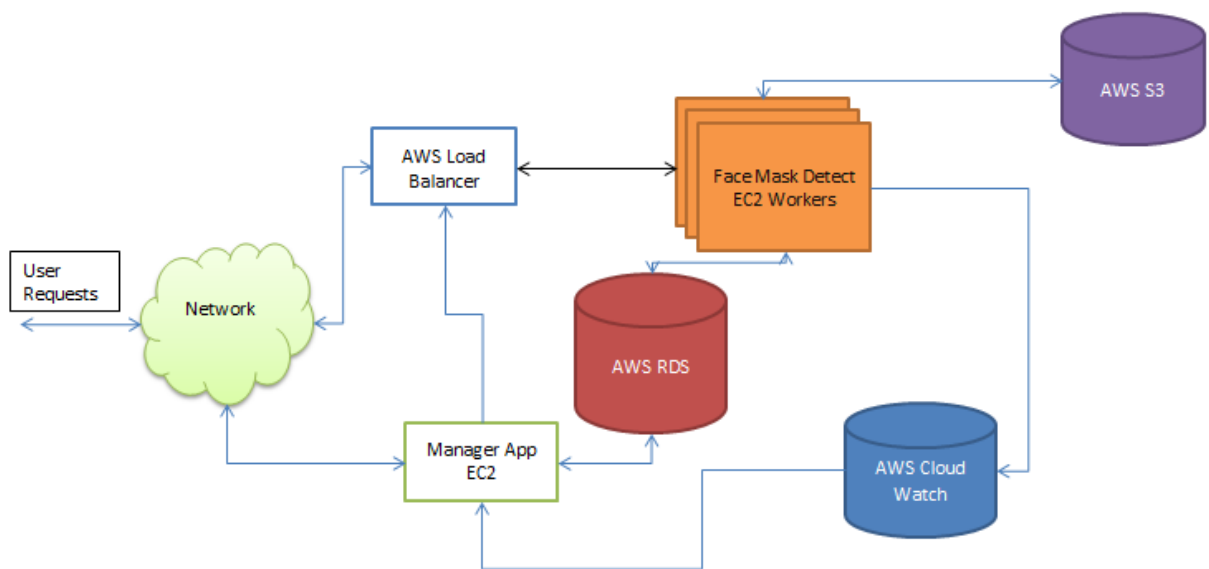


Figure 1: Overview of the face mask application

2. Face Mask Detect Web Application

Please refer to our group's ECE 1779 Project 1 report in relation to details regarding the Face Mask Detect web application (as specified by course instructor). The alterations in the original face mask detect web application to meet the requirements for ECE 1779 Project 2 will be documented below in this report.

2.1.File Directory Layout Changes

The below files are changes from the original Project 1 Face Mask Detect.

<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ FaceMaskDetection<ul style="list-style-type: none">└─ pytorch_loader.py	Added an if condition check that the processed file path is valid. For added test endpoint /api/cpuTest.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ templates<ul style="list-style-type: none">└─ base.html└─ add_user.py└─ change_password.py└─ facemask_detect.py└─ login.py└─ recovery.py└─ upload_history.py	Added IP address of EC2 for debugging load balancer.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ templates<ul style="list-style-type: none">└─ facemask_detect.html└─ upload_history.html	Modified image path to be S3 URL.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ __init__.py	1) Added function and code to get current EC2 instance id, used in publishing metric to cloud watch for number of HTTP requests. 2) Change secret key from being a random function to just the static output from calling a random function once. This fixed issues with bad cookies between multiple EC2 instance workers.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ config.py└─ db_utils.py	Added AWS RDS credentials and information.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ endpoints.py└─ facemask_detect.py└─ cpu_test.py└─ cpu_test_param.txt	1) Modified endpoints to use AWS S3 for image storage. 2) Added /api/cpuTest endpoint that only performs face mask detection algorithm on the provided image with check user credentials or image upload. For testing auto-scalar without having to worry about using the RDS or S3 and its related financial or memory cost.
<ul style="list-style-type: none">└─ app<ul style="list-style-type: none">└─ metric_utils.py	This is a new file that publishes a +1 count to cloud watch for our defined metric for rate of HTTP requests received by worker "HttpRequestCount". This is performed on request teardown callback.

└─ app └─ s3_utils.py	This is a new file to store AWS S3 bucket information.
└─ startup_script └─ README.md └─ ec2_startup.sh └─ facemask_app.init.d └─ facemask_app.systemd └─ restartService.sh	Newly created files for the EC2 worker systemd startup script. This systemd service just calls the init.d script which then call the start.sh script. Reason for the two call is that initially tried to use init.d service but did not work and was recommend online to use system.d.
└─ ece1779_demo_coreyK.pem	New ssh key to use for AWS account.

2.2.Web Application Log-in

We have already created an admin account in the webapp. To view pages that only admin can see for example “Creating and deleting users” use the following admin credentials:

username: mbaquir

Password: password

This will allow you to see the admin privileged pages.

2.3.EC2 AMI and Start-Up Script

In order to allow the Manager Application and auto-scale load balancer policy to create a new EC2 instance with an AMI that already contains our Face Mask Detect code up-and-running it was decided to export our EC2 instance to an AMI. Hence, we updated our code from project 1 to handle the new requirements for project 2 and loaded a EC2 instance with the final code. Then using the Amazon web services EC2 dashboard (website) we manually exported the EC2 instance to an AMI. We made sure that the Load Balancer, S3, RDS, security roles, and start-up scripts were all setup first before exporting.

A systemd start-up script was developed to allow the Face Mask Detect web application to start-up on EC2 boot. There was the option of using AWS –user-data option when starting an instance, however it was found to be easier to develop a stat-up service and provided greater flexibility when debugging.

AMI-ID = ami-03df5d00505b72a20

Please refer to Manager Application section below for details.

2.3.1. Start-up Script

The EC2 start-up script details are in the /startup_script folder. It contains a ec2_startup.sh bash script which copies over systemd service files and then enables/registers the facemask_app service which calls our start.sh script from project 1 that uses gunicorn to start our Face Mask Detect web application. The systemd service is enabled so that our exported AMI will boot with the Face Mask Detect web application running. See README.md for more details.

The script restartService.sh can be used to restart the service during manual ssh debugging on a EC2 instance.

To enable service:

- sudo systemctl enable facemask_app

To start service:

- `sudo systemctl start facemask_app.service`

To stop service:

- `sudo systemctl stop facemask_app.service`

See `/var/log/syslog` for details on service started or stopped. See `error.log` and `access.log` for gunicorn details.

Note: Had to add the following in `run.py`.

```
# Had to add this gevent thing cause of following error cause by gunicorn
# botocore.exceptions.HTTPClientError: An HTTP Client raised an unhandled
exception: maximum recursion depth exceeded while calling a Python object
import gevent.monkey
gevent.monkey.patch_all()
```

Related files:

```
├─ startup_script
│   ├── README.md
│   ├── ec2_startup.sh
│   ├── facemask_app.init.d
│   └── facemask_app.systemd
```

2.3.2. EC2 Worker Details

Refer to next section on the Manager App and Auto-Scalar to see the AWS configuration details used to create a worker from the AMI id previously specified above.

A summary:

We are using the AWS educate account for running our EC2 instance.

The educate account we are using is: `corey.kirschbaum@mail.utoronto.ca`

The ec2 AMI-id we are using is: `ami-03df5d00505b72a20`

AMI ID	ami-03df5d00505b72a20	AMI Name	ECE1779-A2-Worker-Final
Owner	475051144224	Source	475051144224/ECE1779-A2-Worker-Final
Status	available	State Reason	-
Creation date	November 14, 2020 at 2:46:03 AM UTC-5	Platform details	Linux/UNIX
Architecture	x86_64	Usage operation	RunInstances
Image Type	machine	Virtualization type	hvm
Description	-	Root Device Name	/dev/sda1
Root Device Type	ebs	RAM disk ID	-
Kernel ID	-	Product Codes	-
Block Devices	/dev/sda1=snap-0cbd427ada26e60b1:40:true:gp2		

Figure 2: AMI-ID of workers

We have included the ssh keys in the tar file project called : 'ece1779_demo_coreyK.pem'. Also created a copy of the pem file called 'keypair.pem'

Once the instance is started, copy the public IPV4 DNS or IP address to ssh into the server. For example:

```
`ssh -i keypair.pem ubuntu@ec2-3-235-239-51.compute-1.amazonaws.com`
```

Once sshed in, cd into the Desktop directory.

Under the Desktop directory we have ece1779_a1 which is our git repo with the project.

2.4.Amazon S3

Previously images were stored on the EC2 file system running the face mask detect web application, however now that multiple EC2's will be running a centralized file storage system is required. A single amazon S3 will be used as the centralized file system for which images will be stored in.

The S3 Bucket region is us-east-1 so that our EC2 workers can access it. Bucket versioning is disabled, and static website hosting was enabled as type redirect request. For our Face Mask Detect EC2 workers and Manager app EC2 to access the bucket and allow for clients to display image URLs in their browsers the bucket's public access was enabled and a bucket policy was added (see below).

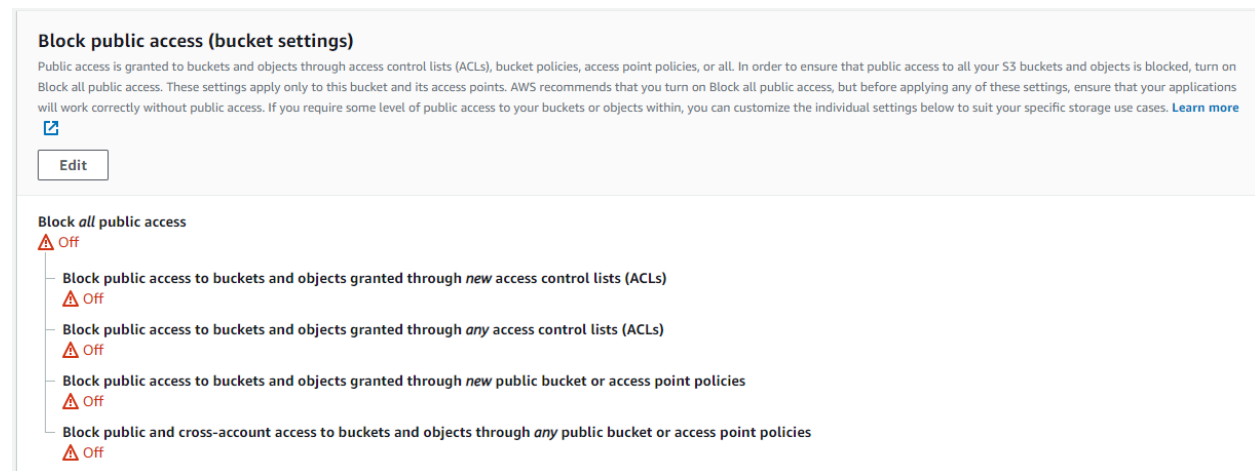


Figure 3: S3 Permissions

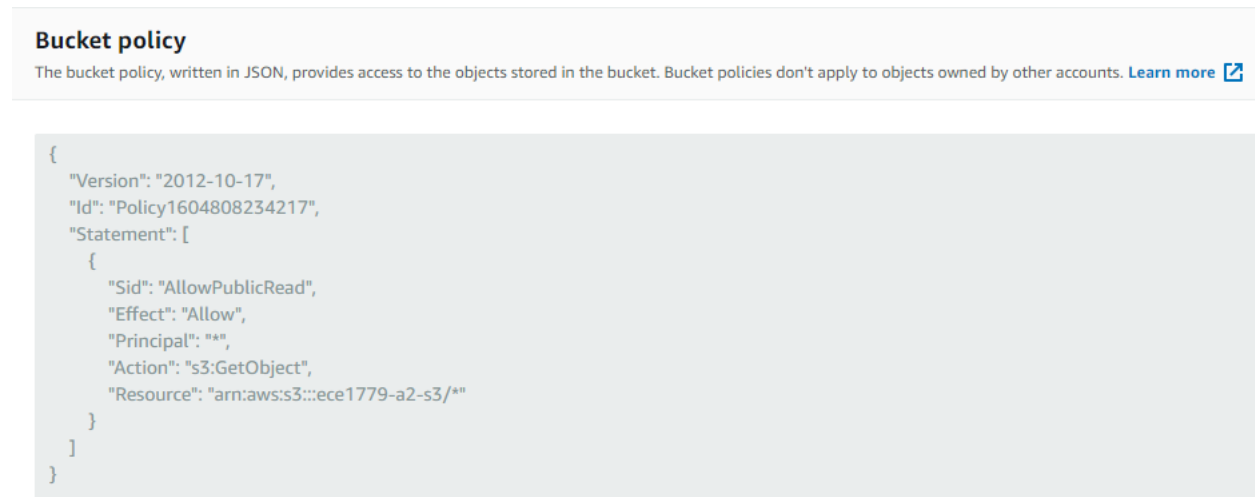


Figure 4: S3 Bucket Policy







Access control list (ACL)		
Grant basic read/write permissions to other AWS accounts. Learn more		
<div>  AWS doesn't recommend granting access to the Everyone grantee Anyone in the world can access the objects in this bucket. Learn more </div>		
Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID:  2d6f3eb4018666017e3657c5b1a879ba1bbee751b04ddeb681f283e39f651dc	List, Write	Read, Write
Everyone (public access) Group:  http://acs.amazonaws.com/groups/global/AllUsers	-	 Read
Authenticated users group (anyone with an AWS account) Group:  http://acs.amazonaws.com/groups/global/AuthenticatedUsers	-	-
S3 log delivery group Group:  http://acs.amazonaws.com/groups/s3/LogDelivery	-	-

Figure 5: S3 Access Control List

S3 website endpoint: <http://ece1779-a2-s3.s3-website-us-east-1.amazonaws.com>

S3 Bucket = ece1779-a2-s3

S3 Bucket Location = <https://ece1779-a2-s3.s3.amazonaws.com/>

See

All EC2 workers running the Face Mask Detect web application are using the IAM role `arn:aws:iam::475051144224:instance-profile/ECE1779-A2-S3-FullAccess` we created. Refer to the IAM Role details section below for further information.

2.5.Amazon RDS and Database Design

2.5.1. Amazon RDS

Previously in project 1 MySQL database was used on the local EC2 instance running the Face Mask Detect application. However, as multiple EC2's will now be running, a centralized database is required.

The RDS was created manually using the Amazon Web Services website. The RDS exists on the same availability zone subnet as our EC2's running the Face Mask Detect (us-east-1c). A db.t2.micro is used as it appears to be the only available size using the free tier.

The only difference now using RDS is that instead of connecting to MySQL on localhost, we are now connecting to the RDS endpoint.

Database credentials => config.py

```
# Details for when MySQL
class DBConfig():
    def __init__(self, user, password, host):
        self.user = user
        self.password = password
        self.host = host
        self.database = 'facemask_app'
        self.auth_plugin = 'mysql_native_password'
```

Figure 6: RDS Face Mask Detect Database Info - Part 1

```
db_config_corey = DBConfig(
    user = 'ece1779',
    password = 'secret20',
    host = 'ece1779-a2-db.cgqfpukixcde.us-east-1.rds.amazonaws.com')
```

Figure 7: RDS Face Mask Detect Database Info - Part 2

The database schemas are in /app/database_schema/facemask_app.sql

Note: to generate database and tables run in MySQL:

mysql --host=ece1779-a2-db.cgqfpukixcde.us-east-1.rds.amazonaws.com -u ece1779 -p

mysql> source app/database_schema/facemask_app.sql

The Face Mask Detect web application uses one MySQL database: "facemask_app"

This database contains only two tables: "users", "images"

Please refer to our group's Project 1 report for the Face Mask Detect web application database design details as no changes were made. The database schema was already normalized and did not require denormalization to improve performance as only one join is made between two tables using the primary keys when display a user's upload history. If required denormalization could be performed to include username in the image table so that no joins are required.

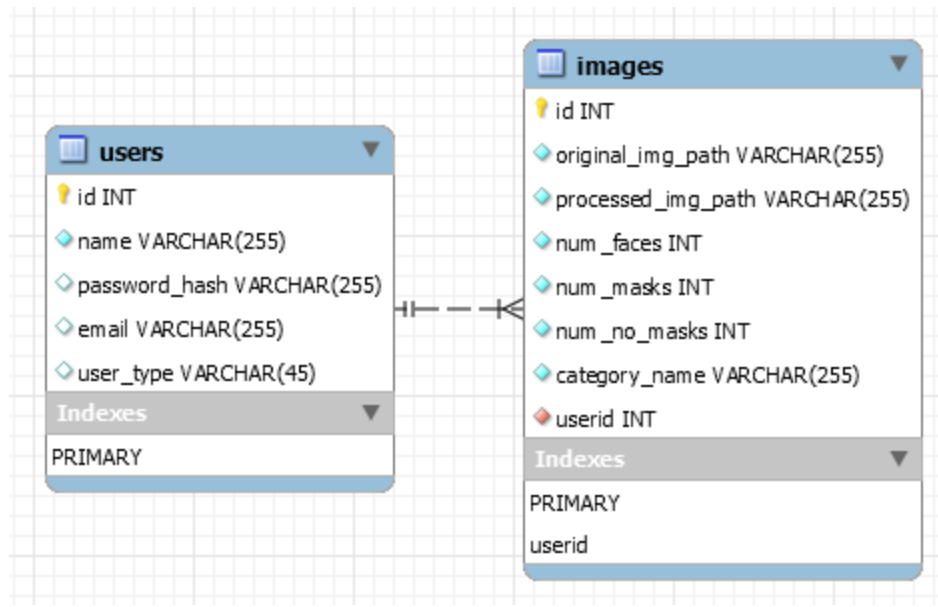


Figure 8: Face Mask Detect Database Schema

2.6. Testing Endpoints

To allow for automated testing, three URL endpoints have been added. The register and upload endpoints are the same from our Project 1.

2.6.1. Register - /api/register

This conforms to the post form specified in Project 1 description to allow new users to be register with the Face Mask Detect web application. Please refer to our group's Project 1 report for further details.

2.6.2. Upload - api/upload

This conforms to the post form specified in Project 1 description to upload images to be processed by the face mask detection algorithm. Information relating to user, original image, processed image, and output from face mask detect algorithm are stored in the RDS MySQL database and S3 file system. Please refer to our group's Project 1 report for further details.

2.6.3. Register - /api/cpuTest

This is a new endpoint created to test the auto-scalar by uploading images to the Face Mask Detect web application, without storing the image or information in the RDS MySQL database and S3 (to try an reduce costs while testing).

Hence, this endpoint is similar to the /api/upload endpoint (same POST form) but only the image file is used.

If file is not provided as an argument => error code 400 with a "Bad Request - No file part" is sent back in json.

If the file argument is empty => error code 400 with a "Bad Request - No image selected for detection" is sent back in json.

If the file type is not valid => error code 400 with a "Bad Request -filename:<filename> - Allowed image types are -> png, jpg, jpeg, gif" is sent back in json.

If could not save file in local file system => error code 500 with a "Bad image_path - Could not save the image" is sent back in json.

If could not parse image (bad image file) => error code 500 with a "Bad Image - Could not parse the image" is sent back in json.

```
{
  "success": true,
  "payload": {
    "num_faces": number,
    "num_masked": number,
    "num_unmasked": number
  }
}
```

Figure 21: cpuTest Json Success Message

```
{
  "success": false,
  "error": {
    "code": servererrorcode,
    "message": "Error message!"
  }
}
```

Figure 22: cpuTest Json Error Message

3. Load Balancer

The load balancer was created manually using the AWS website and was given the name "ECE1779-A2-LoadBalancer".

Name: ECE1779-A2-LoadBalancer

ARN: arn:aws:elasticloadbalancing:us-east-1:475051144224:loadbalancer/app/ECE1779-A2-LoadBalancer/86347ba9d0649e8f

DNS Name: http://ECE1779-A2-LoadBalancer-778242510.us-east-1.elb.amazonaws.com

The load balancer has been setup to using two availability zones:

- us-east-1c

- us-east-1d

The load balancer takes HTTP requests from port 80 for anywhere and uses the following security groups.






Security group name: ECE1779-A2-LoadBalancer-SecurityGroup

Security group ID: sg-0cee92133c88678c4

EC2 > Security Groups > sg-0cee92133c88678c4 - ECE1779-A2-LoadBalancer-SecurityGroup

sg-0cee92133c88678c4 - ECE1779-A2-LoadBalancer-SecurityGroup Actions ▾

Details

Security group name  ECE1779-A2-LoadBalancer-SecurityGroup	Security group ID  sg-0cee92133c88678c4	Description  ECE 1779 A2 Load Balancer Security Group	VPC ID  vpc-8bcf09f6 🔗
Owner  475051144224	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules

Outbound rules

Tags

Inbound rules Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	174.93.115.20/32	-

Figure 9: Load Balancer Security Group

The load balancer target group is called "ECE1779-A2-Targets" and receives traffic on HTTP port 5000 (same as Face Mask Detect application is listening on).

Name: ECE1779-A2-Targets

ARN: arn:aws:elasticloadbalancing:us-east-1:475051144224:targetgroup/ECE1779-A2-Targets/29001bcde241d570

The Face Mask Detect login page (app/login.py) was modified to return a 200 status code to be using by the load balancers health check. The health check is performed every 50 seconds and requires two consecutive successful health checks.

Round-robin is used with a slow start of 120 seconds in order to provide newly created EC2 instances running the Face Mask Detect application more http requests. The reason to use the slow start feature is that it was found while testing the auto-scaler that it takes time for new instances to get traffic levels equal to that of already existing EC2 instances. Least-outstanding-requests could also be used, however did not see any improvements with this option when testing.

A deregistration time of 30 seconds was set to allow exists requests time to complete. 30 seconds is long enough for any single request to complete and short enough that were waiting long periods of time.

ECE1779-A2-Targets			
arn:aws:elasticloadbalancing:us-east-1:475051144224:targetgroup/ECE1779-A2-Targets/29001bcde241d570			
Basic configuration			
Target type instance	Protocol : Port HTTP : 5000	VPC vpc-8bcf09f6	Load balancer ECE1779-A2-LoadBalancer
	Protocol version HTTP1		

Figure 10: Load Balancer Target Port

Health check settings	
Protocol HTTP	Path /login
Port traffic-port	Healthy threshold 2 consecutive health check successes
Unhealthy threshold 2 consecutive health check failures	Timeout 30 seconds
Interval 50 seconds	Success codes 200
Attributes	
Stickiness Disabled	Deregistration delay 30 seconds
Slow start duration 120 seconds	Load balancing algorithm Round robin

Figure 11: Load Balancer Target Setting 1

Also tried (and currently latest version).

Health check settings

Edit

Protocol	Path
HTTP	/login
Port	Healthy threshold
traffic-port	2 consecutive health check successes
Unhealthy threshold	Timeout
2 consecutive health check failures	119 seconds
Interval	Success codes
120 seconds	200

Attributes

Edit

Stickiness	Deregistration delay
Disabled	30 seconds
Slow start duration	Load balancing algorithm
0 seconds	Least outstanding requests

Figure 12Load Balaner Target Setting 2 - Current

4. Manager Web Application

4.1.File Directory Layout

```

├── README.md
├── app
│   ├── __init__.py
│   ├── config.py
│   ├── database_schema
│   │   └── manager_facemask.sql
│   ├── db_utils.py
│   ├── forms.py
│   ├── main.py
│   ├── metric_utils.py
│   ├── s3_utils.py
│   ├── scaler_metrics.py
│   ├── static
│   │   └── flot
│   │       └── <flot packages>
│   ├── templates
│   │   ├── base.html
│   │   ├── main.html
│   │   ├── scaler_metrics.html
│   │   └── workers
│   │       ├── list.html
│   │       └── view.html
│   └── workers.py
├── auto_scaler_out.txt
├── auto_scaler_std.py
└── cpu_test.py

```

```
|— cpu_test_param.txt
|— ece1779_demo_coreyK.pem
|— keypair.pem
|— gen.py
|— gitPull.sh
|— imagesTest
|   |— <images>.jpeg
|— requirements.txt
|— run.py
|— start.sh
|— startup_script
|   |— README.md
|   |— ec2_startup.sh
|   |— manager_app.init.d
|   |— manager_app.systemd
|   |— restartService.sh
```

The AWS ssh keys are contains in ece1779_demo_coreyK.pem (and keypair.pem is a backup copy).
The Manager web application code is contained the /app folder.

The database schema for the Manager app is in the /app/database_schema/manager_facemask.sql, which is used to create our database, it's tables, and initial values.

The /app/metric_utils.py provides helper functions to collect cloud watch metrics data for plots by sorting and organizing the time and value data return from the cloud watch metrics. Data for plots are organized with time being the negative difference from current time, to show past 30 minutes or so (based on arguments to function GetMetricData()), as can be seen in the figures in section 4.3 Manager Home Page below.

As we found that cloud watch sometime returned skips in the data (time wise) when a worker or manager went down and did not return the complete past 30 minutes' worth of data, a min heap was used in GetMetricData() to perform auto-insert of time data points with metric value of zero so that plots and data always contained the same number of datapoints and showed continuous data points.

The flot library was using, similar to in-class tutorials, for plotting data.

/app/config.py contains the database and load balancer/aws details.

/app/s3_utils.py contains the S3 bucket details.

/app/scaler_metrics.py is the form handler for updating the auto-scaler policy.

/app/main.py is the form handler for the main page of the Manager app, showing the number of healthy workers for the past 30 minutes, using the AWS/ApplicationELB -> HealthyHostCount, and the average CPU utilization across all workers, using the AMI-ID:

```

num_workers = GetMetricData(
    Namespace='AWS/ApplicationELB',
    MetricName='HealthyHostCount',
    DurationMinute=30,
    Statistic='Maximum',
    Dimension=[
        {"Name": "LoadBalancer", "Value": aws_config.LoadBalancer},
        #{'Name': 'AvailabilityZone', 'Value': aws_config['AvailabilityZone']},
        {'Name': 'TargetGroup', 'Value': aws_config.TargetGroup}
    ]
)

```

Figure 13: HealthyHostCount Cloud Metric

/app/main.py also contains the two buttons to delete all application data from S3 and RDS database, and a button to terminate all workers and stop the Manager App EC2.

4.2.Flask Web Application

The flask web application is contained in the run.py. This will start the flask application in a single instance; we will be using gunicorn to execute our flask application as a threaded web server in order to provide higher performance at scale. See start.sh at top level for script which starts-up our web application using gunicorn. The same gunicorn settings applied from Project 1 for the Face Mask Detect web application will be using for the Manager web application.

Flask application name = webapp

4.3.Manager Home Page

The manager home page allows the manager to visualize the total number of workers registered to the load balancer. The home page also gives the information about the average cpu utilization of all the workers. These two plots are crucial to know the health and status of the worker pool. These graphs can also help visualize the auto scaler and the trend. With the change in average cpu utilization across all workers, the number of workers would change as well. This would help to know whether the auto scaler is performing well.

[Home](#) | [Workers](#) | [Scaler Metrics](#) |

Manager app

Number of Workers for past 30 Minutes



Load Balancer DNS: [Link](#)

Stop All workers and manager

Delete All application Data

Average CPU Utilization per 1 minute for all Workers in past 30 Minutes



Figure 14: Manager App Main Page

With 3 workers currently registered and a rate of 6 upload/second we can see that the average cpu utilization is around 30 %. This should also reflect the same way to the individual ec2 instances aswell.

The home page also has a link to the load balancer.

The home page has a button to delete all application data. This button would delete the images table and delete all the pictures from the S3 bucket.

The home page also has a stop all workers and manager button. This button would terminate all the workers and stop the auto scaler and stop the manager ec2 instance.

4.4.Manager Worker Page

4.4.1. Worker List

The worker list page displays the list of workers that are registered as targets to our load balancer. ID, name and other information is also displayed about each ec2 instance. 2 additional buttons allow to either delete the ec2 instance or to get more details about that instance. The delete button deregisters the ec2 from the load balancer and then terminates it. Threading is added to that the application is not waiting for the deregistration to happen before terminating the instance. This allows for a richer user experience.

The worker list also has a button to add a new ec2 instance and register it to our load balancer. In order to allow smooth user experience, once a worker is created the registration to the load balancer happens in a separate thread. This allows the user to not wait while we wait for the ec2 to get ready. The GUI is not blocked and the user can navigate to other pages.

Filter option is also allowed for the user to filter the ec2 instance based on its state.

[| Home](#) | [Workers](#) | [Scaler Metrics](#) |

Manager app

New Worker

Filter	All						
ID	Name	Type	Availability Zone	EC2 Status	Load Balancer Status		
i-01f3713820e5fbe49	worker4	t2.medium	us-east-1d	running	healthy	Details	Delete
i-094c34ad75186a7cc	worker6	t2.medium	us-east-1d	running	healthy	Details	Delete
i-0c5f5373d85409be0	worker5	t2.medium	us-east-1d	running	healthy	Details	Delete

Figure 15: Manager App Worker List

4.4.2. Worker Details

For each worker, there is a worker's detail page. This page shows more information about the ec2 worker in details.

4 plots are shown about the workers. Average CPU utilization, Number of HTTP requests/min, Network IN traffic/min and Network OUT traffic/min.

The #of http request/min is a custom metric that we added for our ec2. The metric is added in our user_app flask application under metric_utils.py. ("ece1779_a1/app/metric_utils.py").

On every `@webapp.teardown_request` we add a value of '1' and the unit 'count' to our cloud watch metric. Since this is added to the `teardown_request` call, we essentially are able to count the http request that the app fulfilled. We read `"/var/lib/cloud/data/instance-id"` to figure out what the instance id is where the flask app is running.

The diagram below shows the worker details page.

Manager app

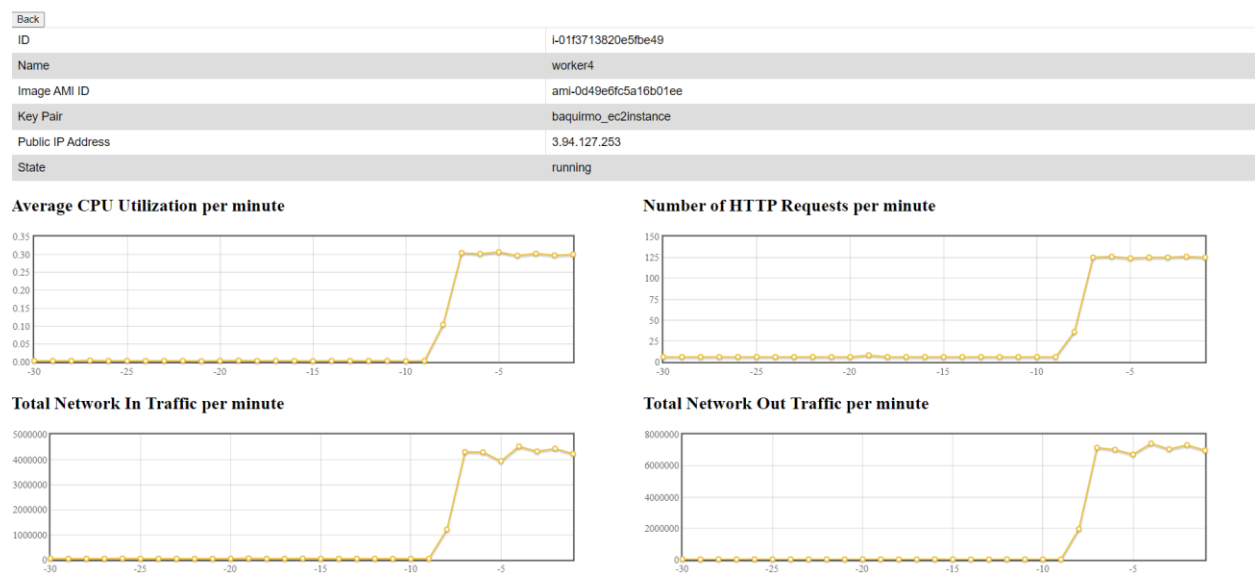


Figure 16: Manager App Worker Details

As you can see with a 2 picture upload/s rate the number of http request is coming out to around 120. We can also see that as the number of HTTP requests are increasing, the cpu utilization is also increasing. With a constant load of http requests coming in, the utilization is also stable. The same trend can also be seen in the network in and out plots.

4.5. Manager Scaler Metrics Page

The scaler metric page allows the manager to tune the auto scaler. It provides an interface for the manager to change the parameters that the auto scaler reads. The information is stored in the database in the metrics table. The auto scaler then reads the data from the database in every iteration.

Manager app

Current Metrics:

CPU Threshold for growing worker pool:	CPU Threshold for shrinking worker pool:	Ratio by which to expand the worker pool:	Ratio by which to shrink the worker pool:
60	10	1.1	0.5

New Metrics for the auto scaler to use:

CPU Threshold (average for all workers over past 2 minutes) for growing worker pool (as a percent between 0 and 100):

CPU Threshold (average for all workers over past 2 minutes) for shrinking worker pool (as a percent between 0 and 100):

Ratio by which to expand the worker pool (float value above 1.0):

Ratio by which to shrink the worker pool (float value between [0 and 1.0)):

Figure 17: Manager App Auto-Scaler Metrics Policy Form Page

Input validation is done in the backend to ensure that the values are within the specified range. Additional input checking is done to ensure that the cpu threshold for growing is larger then the cpu threshold for sinking for example.

```
# Validate Values
if cpu_grow <= 0.0:
    flash("CPU grow threshold should be greater than 0.0.")
    return redirect(url_for('scaler_metrics'))
if cpu_shrink <= 0.0:
    flash("CPU shrink threshold should be greater than 0.0.")
    return redirect(url_for('scaler_metrics'))
if cpu_grow < cpu_shrink:
    flash("CPU grow threshold should be greater than shrink threshold.")
    return redirect(url_for('scaler_metrics'))
if worker_grow <= 1.0:
    flash("The expand ratio should be greater than 1.0.")
    return redirect(url_for('scaler_metrics'))
if worker_shrink < 0.0 or worker_shrink >= 1.0:
    flash("The shrink ratio should be between 0.0 and 1.0, excluding end values.")
    return redirect(url_for('scaler_metrics'))
if worker_grow < worker_shrink:
    flash("The grow ratio should be greater than shrink ratio.")
    return redirect(url_for('scaler_metrics'))
```

Figure 18: Manager App Auto-Scaler Policy Form Validation

4.6.Database Design

See below diagram for database design for the Manager App, called 'manager_facemask' database containing the one table for the auto-scaler policy.

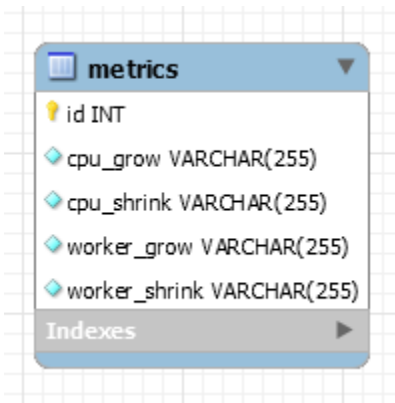


Figure 19: Manager App Database Schema

The database for the manager application is fairly simple and straight forward. The database contains 1 table called metrics to store the scaler tuning policy. The main purpose of the table is to send data to the auto scaler. The manager flask application writes data to the table and the auto scaler code reads data from the table.

See app/database_schema/manager_facemask.sql for database schema.

See app/config.py for database credential information for localhost and RDS.

```
# Corey's RDS
# Manager App Database
class db_config_corey_manager(db_config_local):
    def __init__(self):
        super().__init__()
        self.host = 'ece1779-a2-db.cgqfpukixcde.us-east-1.rds.amazonaws.com'
        self.password = 'secret20'
        self.db_id = 'ece1779-a2-db'
```

Figure 20: Manager App RDS Database Info

4.7.How to run the app

To run the Manager Application the EC2 instance will need to be manually starts, in addition to the RDS.

1. Start RDS MySQL. Name: "ece1779-a2-db"
2. Start Manager App EC2.

Name: ECE1779-A2-Manager

Instance ID: i-0a19049fead5c4c9b

3. Get Public IPv4 or DNS address

4. ssh into Manager App EC2 machine using `ece1779_demo_coreyK.pem` key

```
ssh -i ece1779_demo_coreyK.pem ubuntu@<enter IP>
```

5. Go into the Manager application folder containing the code from the git project.

```
cd ece1779_a2_manager/
```

6. To start the Manager App and Auto-scaler background process run the `./start.sh` script.

```
./start.sh &
```

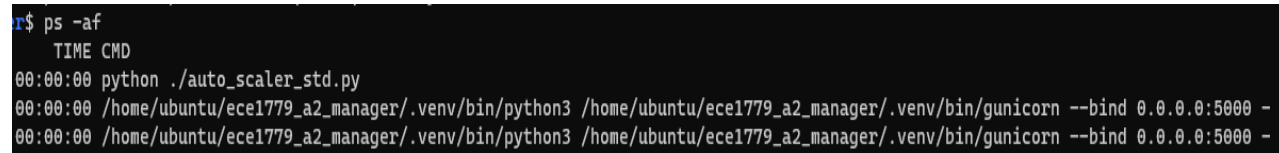
7. Once started it will start the auto-scaler background process `auto_scaler_std.py`, and then start up a worker EC2 instance if one does not already exist, before starting the Manager App flask web application using gunicorn.

See `error.log` and `access.log` for logs related to gunicorn

See `auto_scaler_std.out` for output results of the auto scaler, updates every minute.

See `/var/log/syslog` for any additional details that might be missed on error.

8. Once you see in the ssh terminal the following message “Creating worker on startup of manager app” and running “`ps -af`” command shows that the auto scaler and gunicorn are running, you can now open up the Manager App in your browser using the public IP or DNS using for sshing with port 5000.



```
r$ ps -af
TIME CMD
00:00:00 python ./auto_scaler_std.py
00:00:00 /home/ubuntu/ece1779_a2_manager/.venv/bin/python3 /home/ubuntu/ece1779_a2_manager/.venv/bin/gunicorn --bind 0.0.0.0:5000 -
00:00:00 /home/ubuntu/ece1779_a2_manager/.venv/bin/python3 /home/ubuntu/ece1779_a2_manager/.venv/bin/gunicorn --bind 0.0.0.0:5000 -
```

Figure 21: Manager App `start.sh` Script Running Processes

9. You should be able to see one work in the Manager web application.

When finished press the Manager app main pages delete user application button and then the stop manager button. This will shut everything down except the RDS, so please remember to put stop the RDS database. Reason did not add RDS start/stop into the Manager app code is due to the fact that it takes considerable amount of time to wait for the RDS to start and stop, which during debugging is a waste of time.

5. Auto-Scaler

The auto-scaler is a component that automatically resizes the worker pool based on the load by monitoring the Average CPU Utilization of the workers using cloud watch metrics.

The auto-scaler will use the policy as specified in the Project 2 description and as described above in the Scaler Metrics Page. The CPU threshold will be a percent value from 0 to 100, the grow ratio to expand workers will be a float greater than 1.0, and the shrink ratio will be a float between 0.0 and 1.0.

```
num_workers_to_grow = math.ceil(len(instance_ids) * (worker_grow - 1.0)) # 1.0 is min makes a ratio
```

Figure 22: Auto-Scaler Grow Logic

```
num_workers_to_shrink = math.ceil(len(instance_ids) * worker_shrink)
```

Figure 23: Auto-Scaler Shrink Logic

The auto-scaler code is contained in auto_scaler_std.py file.

It will ensure the number of workers will only be between 1 and 8.

The auto-scaler policy details are stored in the database and are updatable using the Manager web app's Scaler Metrics Page.

5.1.Design

The group came up with a few ideas:

- Current average CPU utilization for past 2 minutes collected every minute
- Time window average over past 3-5 minutes using the average CPU utilization for the past 2 minutes collected every minute
- Weighted time window average over past 3-5 minutes using the average CPU utilization for the past 2 minutes collected every minute
- Using squared difference in the average CPU utilization for past 2 minutes, since it will cause small values to get even smaller and large values to get even larger
- Using standard deviation to see the spread in the Average CPU utilization over the past 2 minutes across all workers

After trying a few of these out, we landed on the last solution plus some additional condition checks. The reason the idea of using the standard deviation is good and quite clever is that it will prevent any possible double add or removal of workers and will wait for the balance of the load to become evenly distributed across all workers before deciding to add more or remove any.

Algorithm:

1. Loop
 - 1.1. Get auto-scaler policy from database
 - 1.1.1. Get average CPU utilization over past 2 minutes for all workers
 - 1.2. Setup a hash map/dictionary that counts the number of workers within each policy threshold

- Below
 - Between
 - Above
- 1.3. Check: If all workers are between threshold do nothing and wait a minute before trying again
 - 1.4. Check: If any workers contain a zero average CPU utilization then that means it was just added and has not gotten any traffic yet. So, wait for worker to get traffic before making any more decisions about adding more workers.
 - 1.5. Check: If number of workers > 1
 - 1.5.1. Check if distributing workload across all workers evenly will result in all workers average CPU utilization being greater or less than threshold
 - 1.5.1.1. If so, then add or remove workers by appropriate ratio
 - 1.5.2. If not all workers are below or above threshold, then check the standard deviation. If not less than or equal to 5.0 (chose by inspection), than continue and do nothing. This means wait until workload is balanced across all workers.
 - 1.6. Else/End: Get the zone/region where majority of workers lie (ie. Below, Between, Above) and add or remove or do nothing as appropriate.

Note: When removing workers an additional minute sleep is used as through testing it was found that it took time for remaining workers to receive the requests which were being serviced by the removed workers. This additional delay removes the throttling of workers, not fluctuating number of workers and a double delete can occur if requests are not yet being passed to remaining running workers. This can be a result of the load balancer settings for deregistering.

A lot of these checks are similar in a manner, trying to ensure decisions are made based on assuming current workload will continue, and predicting if workload is evenly distributed across all workers will it be below, between, or above the policy thresholds.

In part 1.5 of algorithm above the reason for the two check is that 1.5.1 will check to see if current workload continues will all workers be below or above the policy threshold, whereas 1.5.2 standard deviation check does not check for even distribution in relation to the thresholds, but just makes sure that it is even.

5.2.Results

This section will demonstrate that the auto-scaler component works.

To generate CPU traffic to test the auto-scaler the gen.py was used with the /api/upload and /api/cpuTest. A python script was made call cpu_test.py that transmits images from a specified folder to the /api/cpuTest.

During this specific test scenario, a single auto-scaling policy was used.

CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1

The expected output can be thought of as a control system response plot where the margins are the threshold, and eventually the number of workers will settle based on constant workload and policy. See below figure as an example [1].

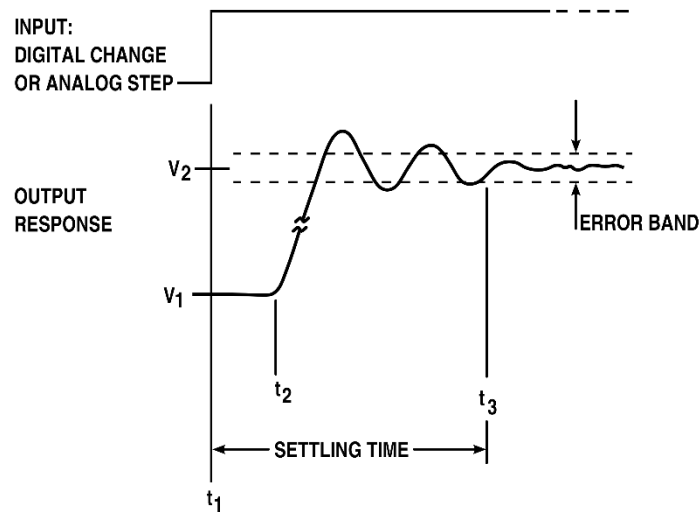


Figure 24: Example of a Control System Response Plot [1]

Hence, the performance of the auto-scaler depends on the workload and the policy values selected.

Example:

If the workload is such that it heavy and the policy thresholds margin is small, it is possible that new workers will get created and then removed in a cycle, even without have a grow rate of double.

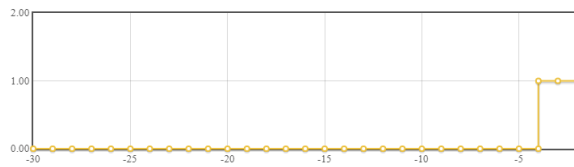
Below results were generated using the `cpu_test.py` script which call `gen.py` where the URL is the <load balancer DNS>/api/cpuTest for an image folder containing 103 images.

It is important to note that having constant workload was somewhat hard to maintain as it was found when using a small number of images or using the RDS large dips were seen between reruns of the `gen.py` script or from database connection which caused add/remove cycles. Hence, it's important to use a large enough workload and local MySQL database that the dip will not affect the threshold checks or make the difference between the policy threshold large enough.

5.2.1. No CPU Load

On boot of the Manager app, it creates 1 worker. It was found that when a worker is created it will have a CPU spike and then a drop.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

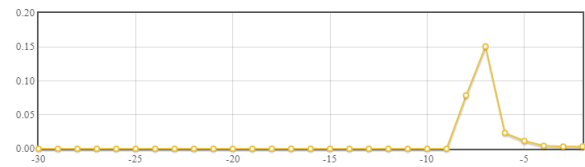


Figure 25: Result Plots for No CPU Load

The auto-scaler will not add or remove workers when no load exists. The CPU utilization will be low, however as a minimum of 1 worker is required, nothing will happen.

```
-----  
Time: 2020-11-15 05:07:29.523640, minute:7  
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1  
Workers Averages: [0.338983050847456]  
zones: {'below': 1, 'between': 0, 'above': 0}  
remove_worker  
deleting 1 workers  
not deleting worker due to min 1 worker threshold
```

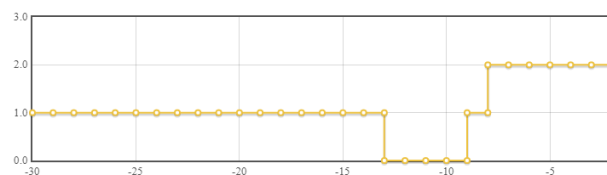
Figure 26: Auto-Scaler Results Log for No CPU Load

5.2.2. Upload Rate of 8

Using an update rate of 8 images and 1000 number of uploads.

The results of the auto-scaler appear to have converged in around 12 minutes to use 2 workers. The beginning change in the number of workers was just cause by us manually stopping/starting during debugging. The converged Average CPU Utilization for workers appears to be around 25% which is within the thresholds.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

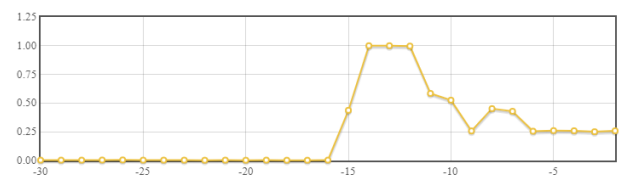


Figure 27: Result Plots for Upload Rate 8 - Increasing

```

-----
Time: 2020-11-15 06:47:37.908072, minute:47
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [25.655737704918, 26.07252014448455]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----

Time: 2020-11-15 06:48:38.078017, minute:48
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [25.4237288135593, 26.75048624617945]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----

Time: 2020-11-15 06:49:38.241302, minute:49
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [25.08333333333334, 25.9731638418079]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----

Time: 2020-11-15 06:50:38.391571, minute:50
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [25.0, 25.875]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----

```

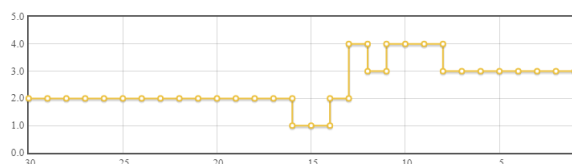
Figure 28: Auto-Scaler Results Log for Upload Rate 8 - Increasing

5.2.3. Upload Rate Increase to 10

Using an update rate of 10 images and 1000 number of uploads.

The results of the auto-scaler appear to have converged in around 12 minutes to use 3 workers. The converged Average CPU Utilization for workers appears to be around 28% which is within the thresholds.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

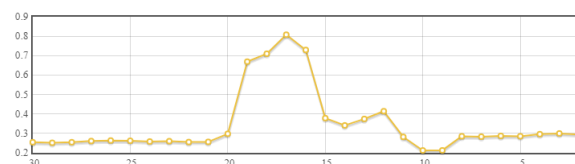


Figure 29: Result Plots for Upload Rate 10 - Increasing

```

Time: 2020-11-15 07:15:32.758134, minute:15
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [26.75, 27.1186440677966, 29.7224576271187]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 07:16:32.908451, minute:16
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [28.2786885245902, 28.375, 28.9166666666667]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 07:17:33.085944, minute:17
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [28.5, 28.6065573770492, 28.6803278688525]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 07:18:33.277470, minute:18
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [28.5593220338983, 28.6805555555555, 29.476938038344]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do

```

Figure 30: Auto-Scaler Results Log for Upload Rate 10 - Increasing

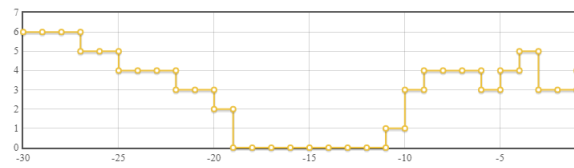
5.2.4. Upload Rate Increase to 100

Using an update rate of 100 images and 1000 number of uploads.

When running at this high rate the plot for number of workers for past 30 minutes is unable to capture the correct number of workers on load balancer as the load balancer or workers are overloaded with requests and so is unable to handle the load balancers health pings in time. As the plot seen below, taken from the Manager app's main page for Number of Workers uses the HealthyHostCount cloud metric, so since the health check pings timeout, they are considered unhealthy. To fix this the load balancer settings might need to adjust to increase the timeout for health check pings.

The auto-scaler does not converge as the workload is too high, however from the CPU Utilization results it seems to settle around 75% to 80% per worker. Though, in the auto-scaler log seen in figures below it seems that 8 workers is not enough for this workload as majority of workers have a CPU utilization for 90% to 100% which way beyond the policy threshold.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

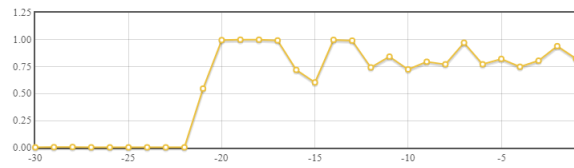


Figure 31: Result Plots for Upload Rate 100 - Increasing

As can be seen in below figure taken from the Manager app the workers are considered unhealthy due to larger work load/requests.

Filter	All				
ID	Name	Type	Availability Zone	EC2 Status	Load Balancer Status
i-0bf39cc48902476ef	worker7	t2.medium	us-east-1c	running	unhealthy
i-0d76a0c1c4b7f6721	worker8	t2.medium	us-east-1c	running	unhealthy
i-0b8894a76252ff6c0	worker15	t2.medium	us-east-1c	running	unhealthy
i-076eb5613a18867c8	worker19	t2.medium	us-east-1c	running	unhealthy
i-04e74af5c04c56460	worker17	t2.medium	us-east-1c	running	unhealthy
i-0b618ac22f5054526	worker20	t2.medium	us-east-1c	running	unhealthy
i-08bf6ee8c705ad17c	worker16	t2.medium	us-east-1c	running	unhealthy
i-0c7d44a8725ff08ec	worker18	t2.medium	us-east-1c	running	unhealthy

Figure 32: Workers Status at Upload Rate 100 - Part 1

Below, it was seen that at time when the load balancer's health check passes we see the workers are healthy.

Filter	All				
ID	Name	Type	Availability Zone	EC2 Status	Load Balancer Status
i-0bf39cc48902476ef	worker7	t2.medium	us-east-1c	running	healthy
i-0d76a0c1c4b7f6721	worker8	t2.medium	us-east-1c	running	healthy
i-0b8894a76252ff6c0	worker15	t2.medium	us-east-1c	running	healthy
i-076eb5613a18867c8	worker19	t2.medium	us-east-1c	running	healthy
i-04e74af5c04c56460	worker17	t2.medium	us-east-1c	running	healthy
i-0b618ac22f5054526	worker20	t2.medium	us-east-1c	running	healthy
i-08bf6ee8c705ad17c	worker16	t2.medium	us-east-1c	running	healthy
i-0c7d44a8725ff08ec	worker18	t2.medium	us-east-1c	running	healthy

Figure 33: Workers Status at Upload Rate 100 - Part 2

Registered targets (8)


[Deregister](#)
[Register t](#)

[1](#)

<input type="checkbox"/>	Instance ID	Name	Port	Zone	Status	Status details
<input type="checkbox"/>	i-0b8894a76252ff6c0	worker15	5000	us-east-1c	healthy	
<input type="checkbox"/>	i-0bf39cc48902476ef	worker7	5000	us-east-1c	unhealthy	Request timed out
<input type="checkbox"/>	i-0b618ac22f5054526	worker20	5000	us-east-1c	healthy	
<input type="checkbox"/>	i-08bf6ee8c705ad17c	worker16	5000	us-east-1c	healthy	
<input type="checkbox"/>	i-076eb5613a18867c8	worker19	5000	us-east-1c	unhealthy	Request timed out
<input type="checkbox"/>	i-04e74af5c04c56460	worker17	5000	us-east-1c	unhealthy	Request timed out
<input type="checkbox"/>	i-0c7d44a8725ff08ec	worker18	5000	us-east-1c	unhealthy	Request timed out
<input type="checkbox"/>	i-0d76a0c1c4b7f6721	worker8	5000	us-east-1c	unhealthy	Request timed out

Figure 34: Workers Status at Upload Rate 100 - Part 3

```

Time: 2020-11-15 08:32:30.002085, minute:32
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [0.423728813559322, 45.96721311475405, 61.36158192090395, 98.8524590163934, 99.66666666666666, 99.74999999999999, 99.75409836065575, 99.7540983606558]
zones: {'below': 1, 'between': 2, 'above': 5}
expecting the current work load, if continues, to go above threshold, so adding worker
creating 8 workers
not creating worker due to max 8 worker threshold
Time: 2020-11-15 08:33:30.298082, minute:33
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [50.9322033898308, 56.9672131147541, 81.33333333333333, 90.77868852459015, 99.25, 99.37978142076506, 99.8305084745763, 100.0]
zones: {'below': 0, 'between': 2, 'above': 6}
expecting the current work load, if continues, to go above threshold, so adding worker
creating 8 workers
not creating worker due to max 8 worker threshold
Time: 2020-11-15 08:34:30.566058, minute:34
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [64.5498749652681, 84.8360655737705, 87.83333333333333, 99.16666666666667, 99.5081967213115, 99.71311475409846, 100.0, 100.0]
zones: {'below': 0, 'between': 1, 'above': 7}
expecting the current work load, if continues, to go above threshold, so adding worker
creating 8 workers
not creating worker due to max 8 worker threshold
Time: 2020-11-15 08:35:30.838068, minute:35
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [1.39344262295076, 45.1639344262294, 48.3064516129033, 73.77118644067795, 85.50847457627114, 99.66666666666662, 100.0, 100.0]
zones: {'below': 1, 'between': 2, 'above': 5}
expecting the current work load, if continues, to go above threshold, so adding worker
creating 8 workers
not creating worker due to max 8 worker threshold
Time: 2020-11-15 08:36:31.130062, minute:36
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [41.41666666666667, 51.2871631008613, 95.25, 99.2622950819673, 99.3333333333334, 99.915254237288, 100.0, 100.0]
zones: {'below': 0, 'between': 2, 'above': 6}
expecting the current work load, if continues, to go above threshold, so adding worker
creating 8 workers
not creating worker due to max 8 worker threshold

```

Figure 35: Auto-Scaler Results Log for Upload Rate 100 - Increasing

```

Time: 2020-11-15 08:53:36.054092, minute:53
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [42.8884180790959, 99.0833333333334, 99.2213114754098, 99.6666666666668, 100.0, 100.0, 100.0, 100.0]

```

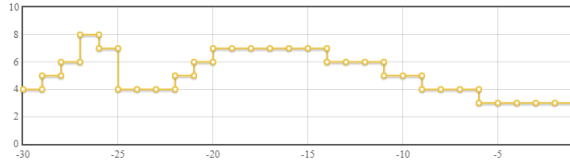
Figure 36: Auto-Scaler Results Log for Upload Rate 10 - Increasing - Worker Averages

5.2.5. Upload Rate Decrease to 10

Using an update rate of 10 images and 1000 number of uploads.

The number of workers converges to the same number as when previously increasing to a rate of 10, that is 3 workers, with around 30%. This took 16 minutes to converge due to the extra wait after removal so that traffic from deleted workers can be passed to remaining workers and the shrink rate is 0.1.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

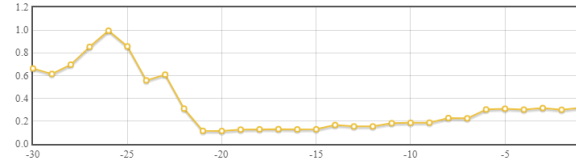


Figure 37: Result Plots for Upload Rate 10 - Decreasing

```
Time: 2020-11-15 17:28:27.523126, minute:28
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [29.70833333333334, 30.33333333333333, 30.7627118644067]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 17:29:27.699032, minute:29
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [29.333333333333332, 30.666666666666668, 32.2370218579235]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 17:30:27.961516, minute:30
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [30.3813559322033, 30.791666666666667, 31.4142817449291]
zones: {'below': 0, 'between': 3, 'above': 0}
All workers in between threshold - nothing to do
```

Figure 38: Auto-Scaler Results Log for Upload Rate 10 - Decreasing

5.2.6. Upload Rate Decrease to 8

Using an update rate of 8 images and 1000 number of uploads.

The number of workers converges to the same number as when previously increasing to a rate of 8, that is 2 workers, with around 37%. This took only around 5 minutes to converge since was previously at 3 workers, so only needed to remove one worker to converge.

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes



Figure 39: Result Plots for Upload Rate 8 - Decreasing

```

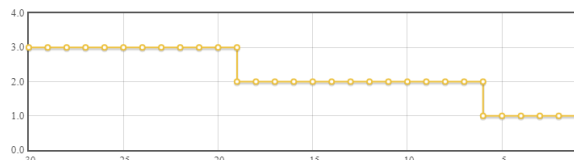
Time: 2020-11-15 17:43:00.481062, minute:43
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [37.7049180327869, 38.33333333333333]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 17:44:00.652377, minute:44
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [37.66666666666667, 38.49999999999999]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 17:45:00.787683, minute:45
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [37.5, 37.844827586207]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do
-----
Time: 2020-11-15 17:46:00.961397, minute:46
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [37.41666666666667, 39.2622950819671]
zones: {'below': 0, 'between': 2, 'above': 0}
All workers in between threshold - nothing to do

```

Figure 40: Auto-Scaler Results Log for Upload Rate 8 - Decreasing

5.2.7. Back to no CPU Load

Number of Workers for past 30 Minutes



Average CPU Utilization per 1 minute for all Workers in past 30 Minutes

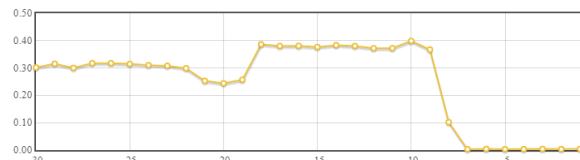


Figure 41: Result Plots for No CPU Load - Decreasing

```

Time: 2020-11-15 17:55:33.403330, minute:55
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [0.333425951653313]
zones: {'below': 1, 'between': 0, 'above': 0}
remove_worker
deleting 1 workers
not deleting worker due to min 1 worker threshold
-----
Time: 2020-11-15 17:56:33.637504, minute:56
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [0.37782485875710103]
zones: {'below': 1, 'between': 0, 'above': 0}
remove_worker
deleting 1 workers
not deleting worker due to min 1 worker threshold
-----
Time: 2020-11-15 17:57:33.846703, minute:57
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [0.375000000000003797]
zones: {'below': 1, 'between': 0, 'above': 0}
remove_worker
deleting 1 workers
not deleting worker due to min 1 worker threshold
-----
Time: 2020-11-15 17:58:34.050744, minute:58
CPU Threshold Grow:65.0%, Shrink:25.0% | Worker Ratio Grow:2.0, Shrink:0.1
Workers Averages: [0.37158469945358996]
zones: {'below': 1, 'between': 0, 'above': 0}
remove_worker
deleting 1 workers
not deleting worker due to min 1 worker threshold

```

Figure 42: Auto-Scaler Results Log for No CPU Load - Decreasing

6. AWS Information

See app/config.py for AWS credentials and information used.

```
# AWS details for Corey
class aws_config_corey(aws_config):
    def __init__(self):
        super().__init__()
        self.AvailabilityZone = 'us-east-1c'
        self.LoadBalancer = 'app/ECE1779-A2-LoadBalancer/86347ba9d0649e8f'
        self.TargetGroupArn = 'arn:aws:elasticloadbalancing:us-east-1:475051144224:targetgroup/ECE1779-A2-Targets/29001bcde241d570'
        self.TargetGroup = 'targetgroup/ECE1779-A2-Targets/29001bcde241d570'
        self.lb_link = 'http://ECE1779-A2-LoadBalancer-778242510.us-east-1.elb.amazonaws.com'
        self.IAM_Role_Arn = 'arn:aws:iam::475051144224:instance-profile/ECE1779-A2-S3-FullAccess'
        self.ami_id = 'ami-03df5d00505b72a20' # 'ami-0207da643e954able'
        self.subnet_id = 'subnet-4de59f00'
        self.security_group_id_lb = 'sg-0cee92133c88678c4'
        self.security_group_id_5000 = 'sg-0ddf9ee037f762e27'
        self.keypair_name = 'ece1779_demo_coreyK'
```

Figure 43: Manager App AWS Info /app/config.py

7. IAM ROLES

A custom role was created to be applied to the Manager app and its workers (all of which are EC2 instances) to allow permissions to access S3, RDS, EC2, Load Balancer and cloud watch.

The created role is called “ECE1779-A2-S3-FullAccess”.

Role ARN: arn:aws:iam::475051144224:role/ECE1779-A2-S3-FullAccess









Policy name ▾
▶  IAMFullAccess
▶  CloudWatchAgentServerPolicy
▶  CloudWatchActionsEC2Access
▶  AmazonSSMManagedInstanceCore
▶  AmazonSSMFullAccess
▶  AmazonS3FullAccess
▶  AmazonEC2RoleforSSM
▶  AmazonEC2FullAccess
▶ ECE1779-A2-UpdateInstanceInformation

Figure 44: Manager App And Workers IAM Roles

It should be noted that there is some overlap in these policies, and in future better optimizing the policies can be performed.

Refer to /app/config.py for full AWS instance details (such as security group for EC2 and load balancer, availability zone, IAM, ...)

Future Improvements

- Denormalize facemask_app database so that username is stored in image table to remove all joins from application. Faster performance possibly.
- Could try to add a cache to speed up database access performance.
- Perform more rigorous testing to handle edge cases
- Go through all security roles and permissions, and reduce to minimum
- Add RDS start and stop in Manager app
- Upgrade RDS database to either larger size like t2.small or a newer version t3, or a production quality.

Issues:

- Tried upload large exe caused something to block my localhost
- Large amount of traffic or workload on workers seems to cause a 504-http status error, is either issue with number of database connections or number of aiohttp open request in gen.py provided by Professor.
- Notice once got a boto3 crash on instance.wait_until_running(). So will probably need to find alternative and contact boto3 developers about this issue.

Unexpected error: (<class 'botocore.exceptions.WaiterError'>, WaiterError('Waiter InstanceRunning failed: Waiter encountered a terminal failure state'), <traceback object at 0x7fc02aaa75c0>)

Group Member Efforts

Group member worked together very evenly on this project. Each member work on all aspects in some manner. Each member created their own S3, RDS and Load Balancer to do local testing. And both members came up with an auto-scaler design before final design chosen.

Corey Kirschbaum

- Documentation
- Update Face Mask Detect web app to use S3, RDS, and have EC2 AMI start-up running flask web app using gunicorn (Update A1 code)
- Create cloud watch metric for rate of HTTP requests by adding the publishing/putting of the metric on request teardown in the Face Mask Detect web app code (Update A1 code)
- Add cpuTest api endpoint for testing file upload without dealing with actual uploading to S3 and RDS to save on costs
- Fixed issue with multiple workers running flask login causing session cookies to be invalid by making sure secret-key is a static random value.
- Created systemd start-up scripts to start the Face Mask Detect web application when a new instance is created using the AMI
- Create IAM roles, security groups, and permissions to ensure everything works
- Create load balancer and configured
- Added plots and worked on ensuring plots always show last 30 minutes even if cloud watch did not return full metric. Used a min heap to sort and insert zeros where appropriate.
- Designed and tested auto-scaler
- Tested and fixed up where needed the creation and deletion of workers

- Added buttons with backend code to delete RDS and S3 user application data
- Add buttons with backend code to stop auto-scaler, terminate all workers, and stop manager app. Code reuse from functions Mohammed made for creating and removing workers
- Add EC2 status with load balancer to instance details list page

Mohammed Baquir

- Documentation
- Update face mask detect web app to use S3, RDS
- Created a first draft of auto scaler to use queue to keep track of previous 3 runs of average CPU utilization.
- Created function to create and delete workers.
- Created page to display list of workers and display details.
- Created the manager database to keep track of scaler metrics.
- Created a page to read data from metrics table and allow ability to write to the table for auto scaler to read.
- Create IAM roles, security groups, and permissions to ensure everything works
- Create load balancer and configured
- Designed and tested first draft of auto-scaler
- Tested and fixed up where needed for both worker and manager app
- Added ability to name EC2 workers to help with debug
- Added display of IP address for debug purposes with Load Balancer

References:

[1] https://en.wikipedia.org/wiki/Step_response

[2] <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>