UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Cloud Computing for Big Data Analytics

Course Project

Chen, Yu-Hsuan (yu-hsuan.chen@est.fib.upc.edu)

Jin, Haonan (haonan.jin@est.fib.upc.edu)

Li, Yalei (yalei.li@est.fib.upc.edu)

Lim, Ariston Harianto (ariston.harianto.lim@est.fib.upc.edu)

Nguyen, Manh Hung (manh.hung.nguyen@est.fib.upc.edu)

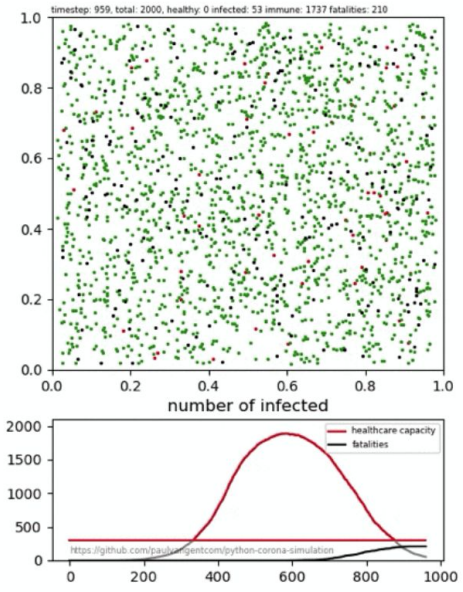
Project Description & Scope

1. Functionalities

As required for the course project, we create a simplified model of people behaviour and infection spread based on academic infectious disease simulation model[1]. Besides, a prototype of contact tracing system proposed by Google and Apple[2] is implemented to evaluate its efficiency. Clients can access our deployed website[3] to set simulation parameters and receive an email detailing the result with almost no time intervening. There are two kinds of simulation: basic pandemic infection simulation and contact tracing simulation.

Simulation of people's behaviour and infection spread

Basically, people in a particular area are represented by dots with different colours which indicate their health status. The coordinates of these dots depend on the locations of people and how they are moving based on individual daily agenda. These following factors can be specified by users via parameters.



- **Population size**: determine how many people participate in the simulation. Larger number of the population infects the simulation runtime, but better reflects the reality.
- **Symptomatic stage duration**: how many days it takes to show symptoms from infected individuals. Larger numbers affect the infected number in the simulation.
- **Incubation stage duration:** how many days are necessary for quarantine.
- **Fighting duration:** how many days it takes for an infected person to recover or die from the infection.
- **Mortality probability:** default percentage for infected ones to die from the infection.
- **Mean number of transmission events per hour**

[1] Mei, S., Chen, B., Zhu, Y., Lees, M. H., Boukhanovsky, A. V., & Sloot, P. M. (2015). *Simulating city-level airborne infectious diseases. Computers, Environment and Urban Systems, 51,* 97-105.
[2] https://www.theverge.com/2020/4/10/21216484/google-apple-coronavirus-contract-tracing-bluetooth-location-tracking-data-app
[3] http://eb-django-express-signup-dev.eu-west-1.elasticbeanstalk.com/

Simulation with Contact Tracing System

We implement a simplified Contact Tracing System to keep track of people who have been recently in contact with an infected person. We then take some measures, for example, self-quarantined and specify it in the simulation to see the changes in infection rate whether it goes down or not. To better reflect the efficiency of this measure, we invite 2 more parameters:

- **App installed probability:** the percentage of the population who has installed this app. Larger numbers indicate wider usage and may correlate to application efficiency.
- **Contact tracing compliance:** percentage of the app holders who will comply with the self-quarantine alarm. Larger number indicates higher compliance.

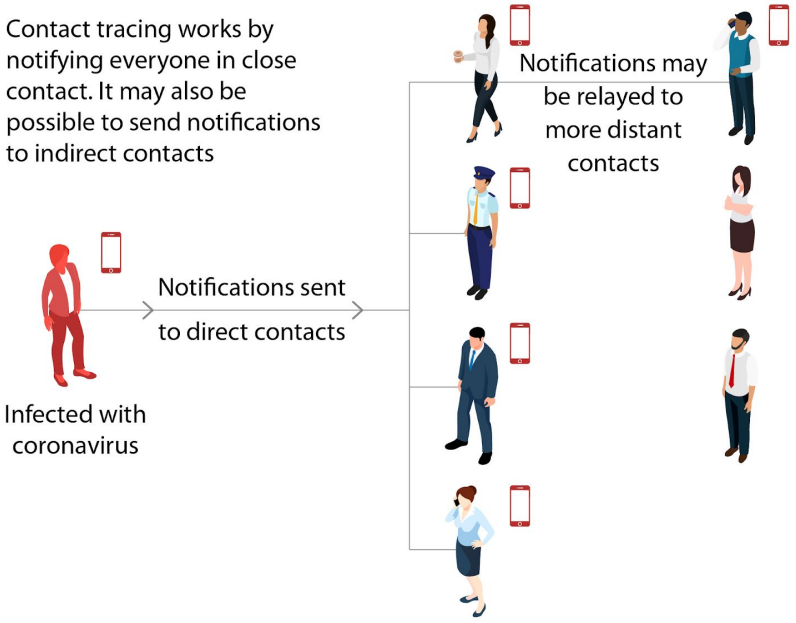


Figure 2. Contact Tracing Implication[4]

2. Scope

This project aims at developing COVID19 infecting simulation with and without Contact Tracing System to evaluate and decide which measures should be taken to stop people from getting infected with coronavirus.

- **Deliverables**: Simulating Services with/without Contact Tracing System
- **Customers & Stakeholders**: Government, Health Agency, Citizens,...
- **Duration**: 5 weeks
- **Development methods**: Agile methodology

4 https://edps.europa.eu/data-protection/our-work/publications/techdispatch/techdispatch-12020-contact-tracing-mobile_en

Team organization

Given the unexpected and uneasy quarantine time, we have to make full use of online platforms and form new norms to organize our team. Fortunately, with the joint efforts and strong commitment, we are able to carry on our work and overcome all the difficulties in coping with different time zones. Below presents our teamwork schedule and distribution.

1. Communication

- Weekly meetings: Having video calls twice a week
- Tasks assignment: Trello
- Version control tool: Git

2. Development Timetable & Hours Invested

TIME TABLE		
Sprint	**Works**	**Hours**
1	- Research on different tools and services - Build draft ideas - Generate data samples	20
2	- Deploy simple simulation to the cloud using Lambda function - Use SQS queue for distributing jobs	40
3	- Move simulation to EC2 instances - Implement auto-scaling	30
4	- Implement Contact Tracing System - Deploy simulation with Contact Tracing System to the cloud - Build a website to send simulation requests	60
5	- Send results to the clients via email using AWS SES - Complete connecting all modules together - Work on the final report	50

* Detailed sprint reports can be found in our group repository.

WORK DISTRIBUTION	
Yu-Hsuan Chen	Data generation, plots generation and send result to clients (SES and S3 configuration)
Haonan Jin	Auto Scaling and Cloud watch, Data generation, SES and S3 configuration
Yalei Li	EB Django website development, simulation model development
Ariston Harianto Lim	Tutorial creation, simulation model development
Manh Hung Nguyen	Build draft cloud architecture, develop the simulation and write scripts for EC2 instances

Twelve-factor methodology application[5]

The twelve-factor is a methodology for building software-as-a-service apps that minimize the effort for new developers joining the project and offering maximum portability and agility. This methodology is especially suitable for developing scalable, maintainable, and portable cloud-native applications. Following is how we use the twelve-factor in our project:

Codebase

Github is chosen as our version control system simply because it's the most popular one and every team member has experience with it. We create a git repository for deploying our project and tracking the version of the code. In the repo, it contains everything that we need to make our application work. The code base is also updated every time EC2 instances start.

Dependencies

As our project is programmed with Python, we manage our dependencies by defining a requirements.txt file. In the file, not only the dependencies but also versions are included. By using this, others can run `pip install -r requirements.txt` to install every library needed to run this project.

```
boto3==1.13.6
botocore==1.16.6
certifi==2020.4.5.1
cycler==0.10.0
docutils==0.15.2
jmespath==0.9.5
kiwisolver==1.2.0
matplotlib==3.2.1
numpy==1.18.4
pandas==1.0.3
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2020.1
s3transfer==0.3.3
six==1.14.0
urllib3==1.25.9
```

[5] "Applying the Twelve-Factor App Methodology to Serverless ...."https://aws.amazon.com/blogs/compute/applying-the-twelve-factor-app-methodology-to-serverless-applications/

Configuration

We use the environmental variables when the information is confidential such as the AWS access key and those variables that can be differ from the environment.

Backing services

We use SQS to store incoming requests and distribute them to EC2 instances. We also use S3 to keep the result of each simulation before sending them to our clients.

Build, release, run

Every time new code is released, it is pulled to EC2 instances from the repository and run automatically using scheduled scripts which run at boot time.

Processes

Our simulation currently runs as a whole big process, breaking it into smaller stateless processes would be better for maintaining and debugging. Due to the limitation of time, we could not decouple the processes.

Port binding

Our approach uses many services which connect to each other via service URLs

Concurrency

We are using SQS queue to receive requests from users and send it to EC2 workers that are auto-scaled. This workflow can handle concurrency without any problems so far.

Disposability

Disposability can be achieved by auto-scaling configuration of EC2, and it allows us to adjust the application's performance by dynamically changing the computing resources along with the workload of component changes. Once the resource is no longer necessary, it can be shut down.

Dev/prod parity

The environments are the same for the developing stage and production stage. As this is only a prototype, we did not split the stages. Again, also due to time limitation.

Logs

Our logs are currently available through the console and we monitor the usage of the resource from AWS CloudWatch.

Admin processes

According to the project scopes, we do not think that we need any admin processes so far.

Simulation

1. Basic Simulation Model

With reference to the agent-based simulation of COVID19 from Paul van Gent[6], we modify the infection model based on our project functionalities. Full programming exposure can be found in our project repository[7]. The following will discuss main code classes in detail.

Population: `population.py`

The simulator will first initiate the population, which is implemented by the population matrix. It stores individual parameters and will be used later in determining simulation results. For example, columns 1-5 indicate the individual mobility. If setting the mortality rate to age-correlated, column 7 will affect the health status. The population matrix for this simulation has the following columns:

```
    0 : unique ID
    1 : current x coordinate
    2 : current y coordinate
    3 : current heading in x direction
    4 : current heading in y direction
    5 : current speed
    6 : current state (0=healthy, 1=sick, 2=immune, 3=dead, 4=immune but infectious)
    7 : age
    8 : infected_since (frame the person got infected)
    9 : recovery vector (used in determining when someone recovers or dies)
    10 : in treatment
    11 : active destination (0 = random wander, 1, .. = destination matrix index)
    12 : at destination: whether arrived at destination (0=traveling, 1=arrived)
    13 : wander_range_x : wander ranges on x axis for those who are confined to a
location
    14 : wander_range_y : wander ranges on y axis for those who are confined to a
location
```

Individual mobility: `motion.py`, `path_planning.py`

To imitate the daily agenda of an individual's behavior and social character, we simplify the complex agenda into the number of destinations a person will visit per day and the moving speed (i.e. a delivery man will visit more places and has higher moving speed per day). If a person is in treatment or performing self isolation, the speed will be reduced, and the wonder range will be shortened.

[6] https://github.com/paulvangentcom/python_corona_simulation
[7] https://github.com/CCBDA-UPC/2020_Project_Fri_9_30/tree/master/simulation

Infection: `infection.py`

The virus is theoreticized into location-based infection. For each iteration, it will look for all the healthy agents within the infection zone of an infected case. For each healthy individual, the chance to get infected needs to be tested. To speed up computation, if more than half of the population are infected, it will look for all healthy agents, and check if they have been in contact with an infected agent. It is defaultly set that the infection starts in time 50, and 0.05 default mortality rate is applied.

Simulation: `simulation.py`

To run the simulation, it will take each hour as an iteration step and run the program until no one is infected. Timestamp is marked for each iteration, and a population tracker is set to monitor the infection situation (e.g. infected number, death number etc.).

The console will express the logs containing detailed simulation information, and an optional visualizer is also set to better test the program locally. At the end, the summary will be printed revealing the whole simulation process. Below demonstrate the sample result:

```
/opt/anaconda3/envs/CCBDA_UPC/bin/python3.7 "/Users/testing/Desktop/CC
Project/python_corona_simulation-master/simulation.py"
50: healthy: 1000, infected: 0, immune: 0, in treatment: 0, dead: 0, of total: 1000,
lost production: 0
infecting person
...
...
...
1513: healthy: 755, infected: 0, immune: 222, in treatment: 0, dead: 23, of total:
1000, lost production: 0

-----stopping-----

total timesteps taken: 1514
total dead: 23
total recovered: 222
total infected: 0
total infectious: 0
total unaffected: 755
```

2. Contact Tracing Model

Based on the basic simulation model, we further introduce the contact tracing logic. First, we include 2 more parameters within the population matrix to store the quarantine indicators (shown below). Boolean marks will be put in column 15 if an individual obeys the contact tracing suggestion and performs social distancing. Column 16 will mark the start time of the quarantine.

```
    15 : quarantined after receiving app alarm (contact tracing)
    16 : quarantine since
```

For `infection.py`, we regularly check the whole population for sick and healthy, which are location-based infection (i.e. check the individual physical indicator for infected probability, based on those who have been within the infection area of a confirmed case). During the simulation iteration, if one receives the app alarms, we will reduce his/her speed, and imitate home isolation by setting the destination as an isolated range:

```
if len(population[population[:, 10] == 1]) <= Config.healthcare_capacity:
   patient[10] = 1
   if send_to_location | Config.contact_tracing:
       # send to location if die roll is positive
       if np.random.uniform() <= location_odds:
           if population[np.int32(patient[0])][15] == 0:
               if np.random.random() < Config.contact_tracing_compliance *
Config.app_installed_probability:
                   population[np.int32(patient[0])][15] = 1
                   population[np.int32(patient[0])][5] = 0
                   population[np.int32(patient[0])][16] = frame
                   population[np.int32(patient[0])], \
                   destinations[np.int32(patient[0])] =
go_to_location(population[np.int32(patient[0])],
destinations[np.int32(patient[0])], Location_bounds, dest_no=location_no)
```

We further assume that people will be self-quarantined for 14 days (i.e. `incubation_stage_duration = 336`) after their contacts are confirmed infected. Here, we assume that defaultly it will take 2 days for patients to show symptoms (i.e. `symptomatic_stage_duration = 48`), which will delay the alarm sent to the nearby contact. After the quarantine period, the individual will be tested by his/her default chance of getting infected. If healthy, his/her mobility will be restored. For those confirmed cases, if there is healthcare capacity (i.e. `healthcare_capacity = 300`), they will be marked as 'in treatment', and further isolated in the hospital area.

Besides, in order to test the necessity of the app, we introduce 2 more parameters: `app_installed_probability` and `contact_tracing_compliance` to set the amount of people who actually comply with self-quarantine.

Furthermore, in order to track the effect on the economy, we monitor the total production time lost due to the quarantine order (code shown below). For each iteration, those who are in quarantine will be viewed as not working, and when the infected amount reaches more than 70% of the population, system alarms of “collapsing economy” will be sent to the client.

```
production_time_lost += np.sum(frame - population[population[:, 6] == 1][:, 8])
production_time_lost += np.sum(frame - population[population[:, 15] == 1][:, 16])
```

As a result, the contact tracing mode shows significant improvement in defeating the COVID19 pandemic. The testing results are displayed below:

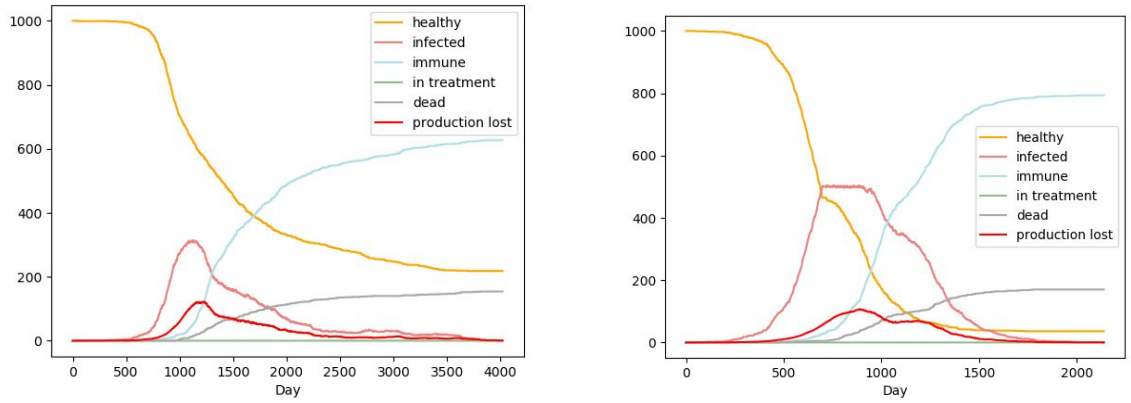


Figure 3. Simulation Result with (left) & without (right) Contact Tracing App

Statistically, the results are as below:

```
[Population = 1000]

     Contact Tracing                          No Contact Tracing
 (app_installed_probability=1,
 contact_tracing_compliance=1)

 total timesteps taken: 3842                 total timesteps taken: 2438
 total dead: 147                             total dead: 169
 total recovered: 630                        total recovered: 784
 total infected: 0                           total infected: 0
 total infectious: 0                         total infectious: 0
 total unaffected: 223                       total unaffected: 47
```

Generally, the death rate shows a slight decrease between the two scenarios (speculation may be that we set the mortality rate to be only correlated with individual age and whether to get treatment). However, the infection curve is greatly flattened with the contact tracing app while a large number of people are not affected at all. Worth mentioning is that the widely applied self-quarantine may have a negative effect on the economy, given that the lost production time peaked higher than normal.

Cloud Architecture

After successfully testing the simulation functionality locally, we now deploy the whole project on AWS cloud. The diagram below illustrates most of the elements in this cloud-based project and how they relate to each other.

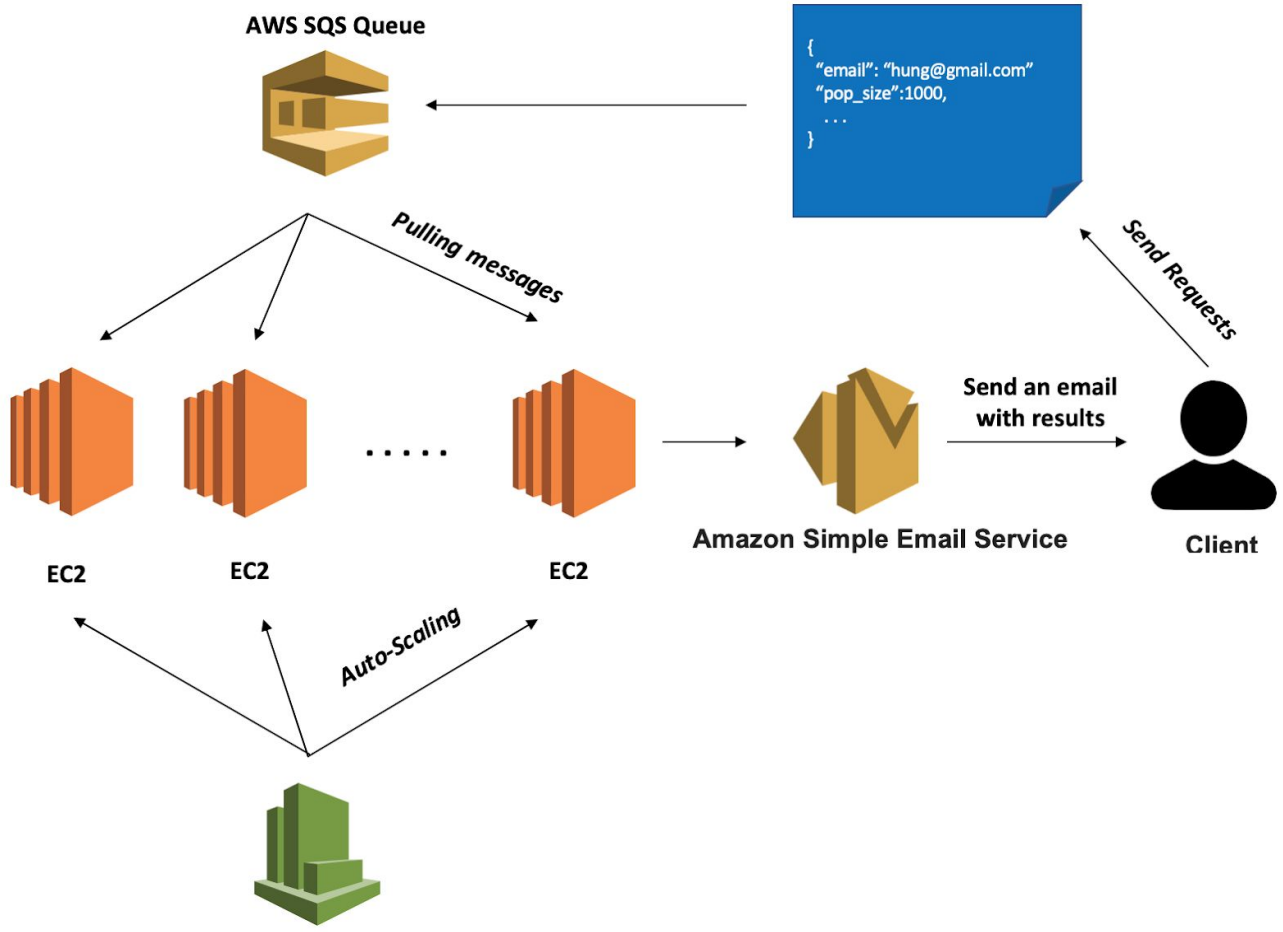


Figure 4. Project Cloud Architecture

Our clients use our website to send simulation requests to the SQS queue which manages messages in First In First Out (FIFO) manner. Multiple EC2 instances keep getting jobs from the queue and initializing new simulations based on parameters that they got in parallel. These workers are auto-scaled in and out by AWS Auto-scaling service which takes into account their CPU utilization. Once they finish the simulation, the results are stored in a S3 bucket and sent to our clients by AWS SES service. After that, they start a new cycle with getting the next request from SQS.

1. Elastic Beanstalk Website

Elastic Beanstalk (EB) can offer plenty of integrity platforms or services, which can improve the efficiency of our work. We make use of Elastic Beanstalk throughout the whole tasks including simulation and prototype of the Google Apple project because of its universality, convenience and scalability.

Advantage

Compared to local API, the PaaS service offered by Amazon Elastic Beanstalk can greatly reduce our management complexity without sacrificing our choices or control. We can focus on the main functionality development, leaving out the environment deployment and maintenance to EB. All we need to do is upload the application and Elastic Beanstalk will automatically handle deployment details about capacity provisioning, load balancing, scaling, and application health monitoring. It supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. When we deploy an application, Elastic Beanstalk offers concurrently-supported versions for the selected supported platform and provisions suitable for AWS resources (e.g. Amazon S3, EC2 instances) to run the application.

Application

In order to better connect our client with our simulation application, we deploy a Django website with Elastic Beanstalk. Here, we refer to the SQS tutorial[8] proposed by our colleagues to integrate the website with SQS service. In the case of this step, Python (boto3) has the necessary libraries to operate SQS services without any further management.

To be more specific, we deploy an AWS Beanstalk environment to host a web application which will ask for demanded simulation parameters to run the simulation in backend EC2. The information will be sent to the SQS queue which will launch a new instance in the auto scaling group in order to process each request.

[8] https://github.com/CCBDA-UPC/Research-projects-2020/tree/master/05_SQS

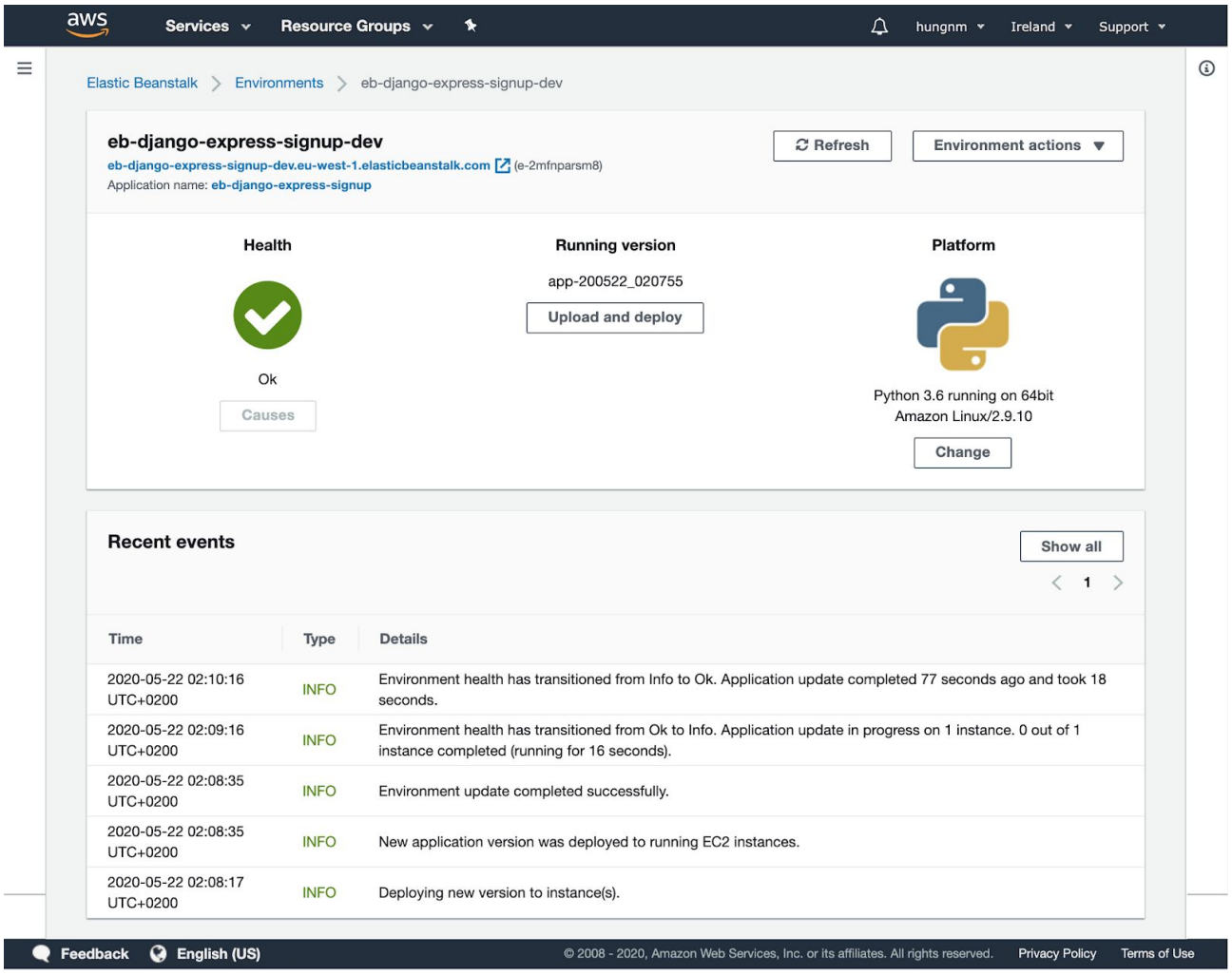


Figure 5. Established Elastic Beanstalk Environment

The Django website[9] is established as below to collect simulation parameters from clients.

[9] http://eb-django-express-signup-dev.eu-west-1.elasticbeanstalk.com/

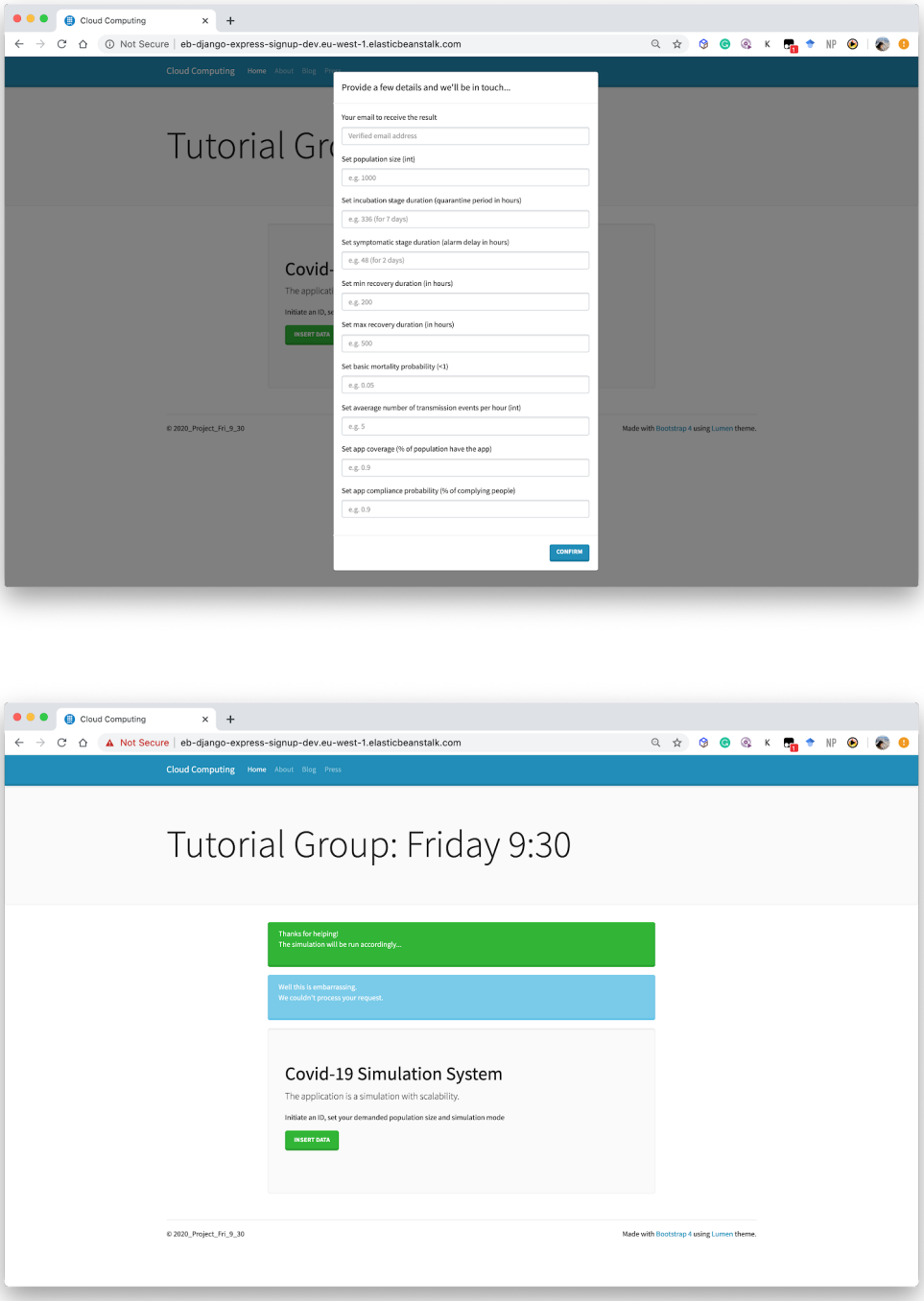


Figure 6. Django Data Insertion Page & Mainpage

We are considerably benefitting from employing the Elastic Beanstalk, where we can focus on working with the detailed simulation functions instead of the operating environment. With the development of the project, new ideas kept appearing, and we had to change the functional codes frequently. Taking the advantage of EB, we do not need to set up the virtual environment for different function requirements, but simply deploy the changes to the cloud (i.e. with simple command `eb deploy`). Besides, we can provide our client with 24/7 available service through the website. The cloud-based Django webpage depends on AWS server, which offers great availability to customers. We do not need to manage the server locally, but be able to enjoy the appreciable flexibility and availability.

2. Amazon Simple Queue Service (SQS)

Amazon Simple Queue Service (SQS) is a secure and available queue message service which provides integration and scalability of “serverless” microservices, distributed systems and applications. SQS gets in charge of the complexities in the management and functionalities of the middleware so that developers can develop applications more efficiently.[10] Two kinds of queues are offered by SQS to send, store and receive messages. The standard queues provide rapid processing; while the First-In-First-Out (FIFO) queues ensure an orderly execution of the messages efficiently. Noteworthy, the API service implemented by SQS presents substantial flexibility which can be accessed from any programming language (which supports the SDK of AWS). In the case of our project, Python (boto3) has the necessary libraries to enjoy SQS services seamlessly.

In the following architecture, the autoscaling group is used to manage the different EC2 instances according to the requests from the queue. In Amazon CloudWatch, we are going to measure the number of requests from the queue so that the AutoScaling group can launch or terminate instances based on demand and also set parameters of configuration as time of execution, delay and others.

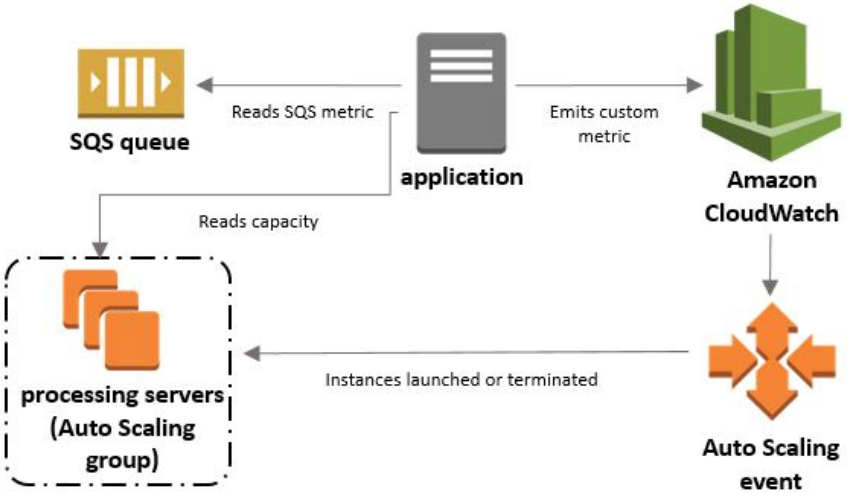


Figure 7. SQS Usage

With the help of AWS SQS, the website traffic can be automatically managed with FIFO setting and auto scaling operation in the later phase. Details of Django application settings can be found in the [eb-django-express-signup-base-master] folder of our group repository[11]

After the client fills in simulation parameters of the online file,SQS will receive the message containing all the required attributes (shown below). The EC2 (details in the latter section) will be able to constantly scan the SQS and pull the newest message for computation. The

[10] https://aws.amazon.com/sqs/

[11] https://github.com/CCBDA-UPC/2020_Project_Fri_9_30/blob/master/eb-django-express-signup-base-master

FIFO queue can help us better manage the simulation orders, and avoid duplications (which further ensures customer satisfaction).

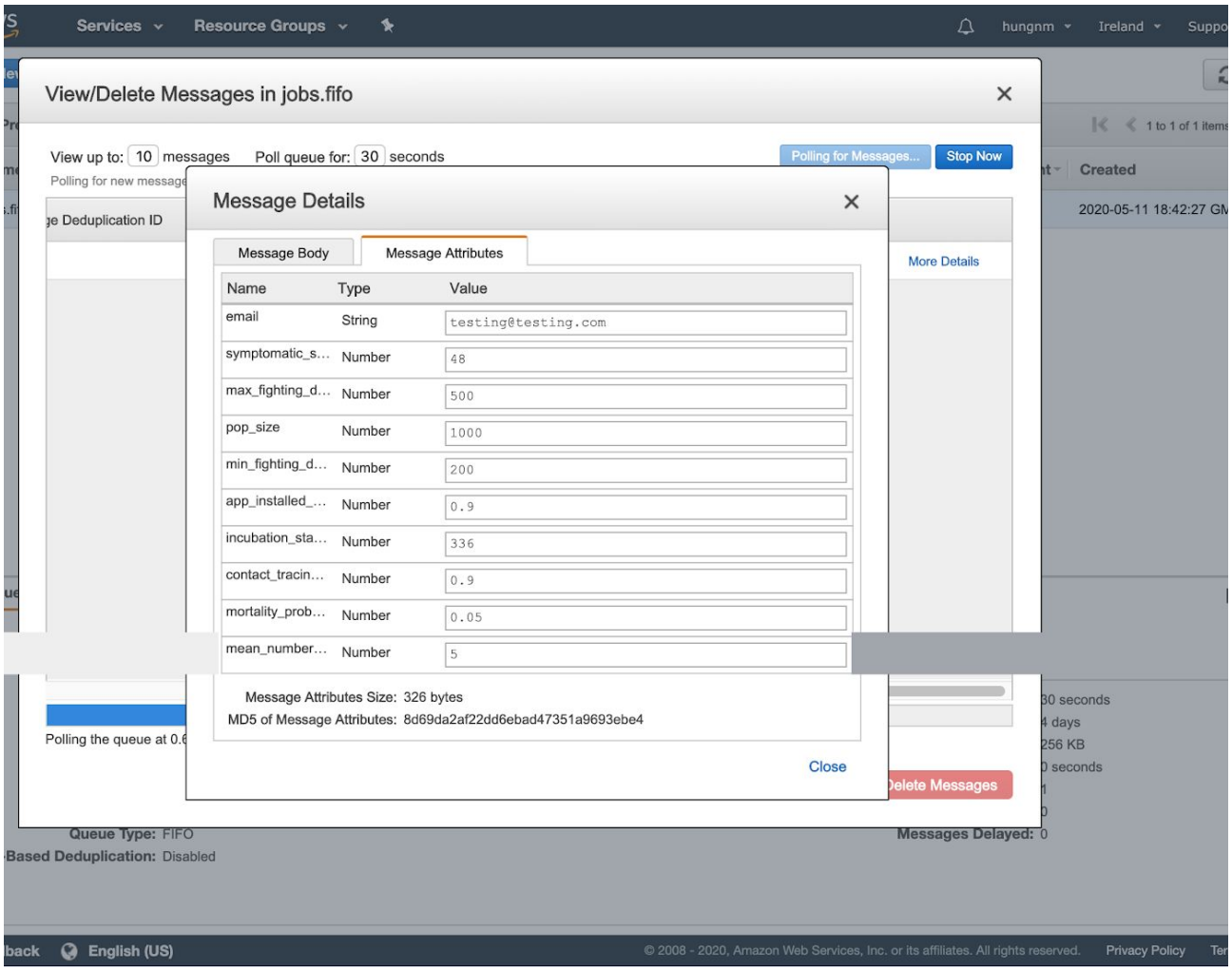


Figure 8. SQS Message Sample

3. EC2

AWS EC2 instances are used to run the simulations since they are suitable for computational intensive jobs. Each EC2 is launched from a customized AMI which contains a script running every time the instance starts. The latest logic code and configuration are pulled from github to the EC2 instance automatically.

```
#!/bin/bash
cd /home/ubuntu/2020_Project_Fri_9_30/
git pull
/usr/bin/python3.6 /home/ubuntu/2020_Project_Fri_9_30/simulation/simulation.py
```

At the end of the script, the python code runs and it starts pulling a new message (job) from the given SQS url to get simulation parameters and running 2 simulations with and without the Contact Tracing System (CTS) to evaluate the efficiency of CTS. After successfully processing a message, we deleted it from the SQS queue. The code used to do this is shown as below.

```
def pull_jobs():
    sqs = boto3.client('sqs', region_name='eu-west-1')
    queue_url = 'https://sqs.eu-west-1.amazonaws.com/355914966584/jobs.fifo'
    while True:
        try:
            # Get 1 message from SQS
            response = sqs.receive_message(
                QueueUrl=queue_url,
                AttributeNames=['SentTimestamp'],
                MaxNumberOfMessages=1,
                MessageAttributeNames=['All'],
                VisibilityTimeout=30,
                WaitTimeSeconds=0
            )

            # Delete the received message
            message = response['Messages'][0]
            receipt_handle = message['ReceiptHandle']
            sqs.delete_message(QueueUrl=queue_url, ReceiptHandle=receipt_handle)

            # Parse the message to get parameters
            parameters = message['MessageAttributes']
            print(parameters)

            # Config
            config1 = Configuration()
            config1.contact_tracing = False
            config1.pop_size = int(parameters['pop_size']['StringValue'])  # Set
population size
            config1.email = parameters['email']['StringValue']  # Set email
           config1.symptomatic_stage_duration =
int(parameters['symptomatic_stage_duration']['StringValue'])

config1.incubation_stage_duration=int(parameters['incubation_stage_duration']['StringValue'])

config1.min_fighting_duration=int(parameters['min_fighting_duration']['StringValue'])

config1.max_fighting_duration=int(parameters['max_fighting_duration']['StringValue'])

config1.mortality_probability=float(parameters['mortality_probability']['StringValue'])

config1.mean_number_of_transmission_events_per_hour=int(parameters['mean_number_of_transmission
_events_per_hour']['StringValue'])


            config2 = Configuration()
            config2.contact_tracing = True

config2.set_contact_tracing(app_installed_probability=float(parameters['app_installed_probabili
ty']['StringValue']),
```

```
contact_tracing_compliance=float(parameters['contact_tracing_compliance']['StringValue']),

symptomatic_stage_duration=int(parameters['symptomatic_stage_duration']['StringValue']),

incubation_stage_duration=int(parameters['incubation_stage_duration']['StringValue']),

min_fighting_duration=int(parameters['min_fighting_duration']['StringValue']),

max_fighting_duration=int(parameters['max_fighting_duration']['StringValue']),

mortality_probability=float(parameters['mortality_probability']['StringValue']),

mean_number_of_transmission_events_per_hour=int(parameters['mean_number_of_transmission_events_
per_hour']['StringValue']))
            config2.pop_size = int(parameters['pop_size']['StringValue'])  # Set
population size
            config2.email = parameters['email']['StringValue']  # Set email

            # initialize
            sim1 = Simulation(config1)
            sim2 = Simulation(config2)

            # Run
            sim2.run()
            sim1.run()

            # Send emails
            send_results(sim1, sim2)
```

4. S3

Amazon Simple Storage Service (Amazon S3) provides highly-scalable, secure, durable, object storage service. The pay-as-you-go feature eliminates the effort to the capacity plan.[12] Another benefit of S3 is not only can it be used alone, but it also provides a high level of integration with other AWS cloud services. It also offers a wide range of settings to the permissions and access control.

The plots are generated after finishing the simulation. From the website, each client can send several simulation requests with different parameters, so the results will also be stored respectively. Every file that uploads to S3 here is named by a string and a timestamp to make sure it's unique.

As for the cost, the price is $0.023 per GB for the first 50 TB; 1 GB allows us to perform 17476 simulations as each of them generates approximately 60 KB results.

[12] "Amazon Simple Storage Service (S3) — Cloud Storage — AWS." https://aws.amazon.com/s3/faqs/.

contact-tracing-1590188240.55133.png	May 23, 2020 6:57:22 AM GMT+0800	37.0 KB	Standard
contact-tracing-1590188911.752069.png	May 23, 2020 7:08:33 AM GMT+0800	41.6 KB	Standard
contact-tracing-1590189000.212228.png	May 23, 2020 7:10:01 AM GMT+0800	33.9 KB	Standard
contact-tracing-1590189124.675056.png	May 23, 2020 7:12:06 AM GMT+0800	34.4 KB	Standard
no-contact-tracing-1590078700.547033.png	May 22, 2020 12:31:42 AM GMT+0800	37.5 KB	Standard
no-contact-tracing-1590078818.865043.png	May 22, 2020 12:33:40 AM GMT+0800	38.0 KB	Standard
no-contact-tracing-1590078924.520598.png	May 22, 2020 12:35:25 AM GMT+0800	37.2 KB	Standard

Figure 9. The objects inside the S3 bucket.

The results stored in S3 bucket are going to be presented in the email that is sent to the client. By modifying the bucket policy, the client can download the resulting pictures without access to the s3 bucket directly.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::simulationresult2/*"
        }
    ]
}
```

5. Simple Email Service (SES)

Amazon Simple Email Service (Amazon SES) is a cloud-based email sending service. It is a reliable, cost-effective service for businesses of all sizes that use email to keep in contact with the client.[13]

In SES, there are two formats to present the mail content to the client, firstly is the text-only form and second is an HTML form. We adopt the second method due to the flexibility, plus we also want to present our client with plots that illustrate the simulation result. However, most web email browsers do not support showing images by using the inline image in HTML even with base 64 encoded. That's also the reason that we implement S3 and upload the result to the bucket at the previous step. Then, we use A href attribute to link to the object in the bucket. In this case, the client can not only see the picture presenting, but the download choice is also available for them.

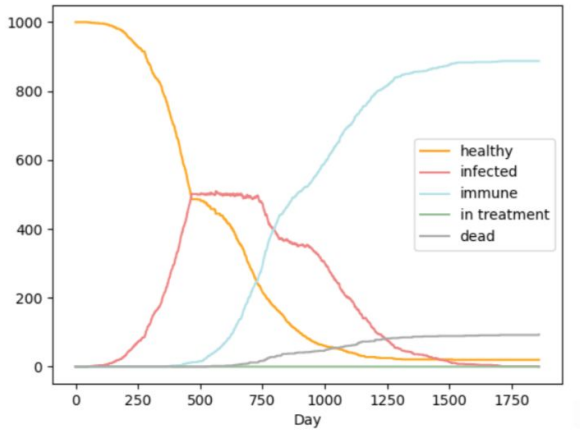
[13] "Amazon Simple Email Service (Amazon SES)." https://aws.amazon.com/ses/.

Dear user,

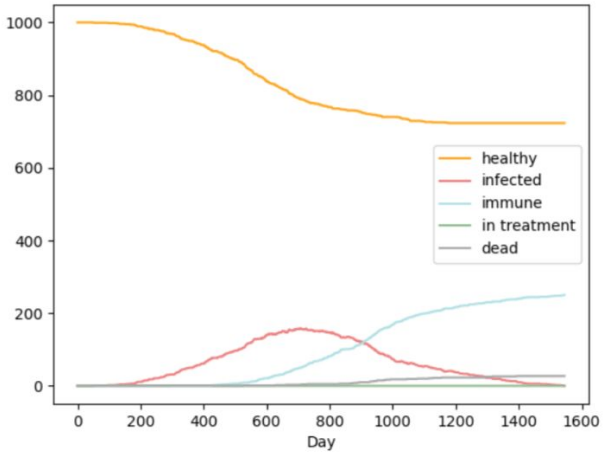
Here is the simulation report based on the parameter that you provide.
Simulated population:1000
Contact Tracing System:True
The number of deaths decreases by: 70.21%
The number of infected people decreases by: 71.85%

Report

Simulation without Contact Tracing System



Simulation with Contact Tracing System



Best regards

Figure 10. The email that client received

The code used for sending a result email to our clients is shown below.

```
def sent(file, file2, population, contact_tracing, email, ...message1, message2):
SENDER = "Administrator <mail address>"
RECIPIENT = email
AWS_REGION = "eu-west-1"
SUBJECT = "Your simulation result."
BODY_HTML = """<html>
 <head></head><body></body>
 </html>
 """.format(statistic=file, contact_tracing=file2....)
```

6. AWS Auto-scaling

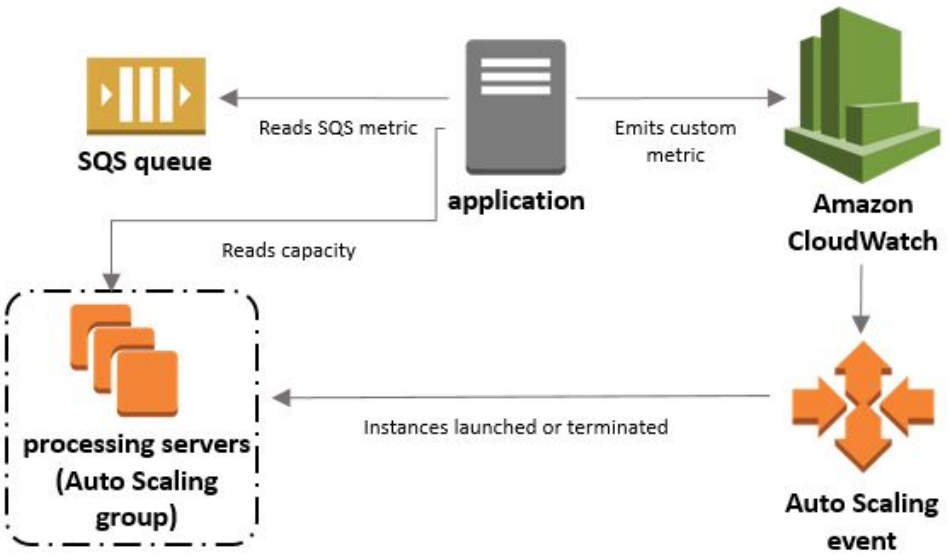


Figure 11. Structure of Auto Scaling

We implement auto scaling by managing the auto scale group of EC2 instances in response to the changes of the workload from Amazon SQS queue. As the number of requests are unpredictable, so the dynamically scale up capacity is needed.

We design two kinds of auto scaling policies so as to increase or decrease the running EC2. For one thing, auto scaling policies are decided by CPU utilization. Based on the minimum and maximum running EC2 between 1 and 4 instances as well as the one initial running EC2 configuration, we decrease 2 EC2 if CPU utilization is less than 25% while increase 2 EC2 if CPU utilization is over 70%.

Figure 12. Scale in and scale out the EC2 capacity

Also, another auto scaling policy is decided by the size of SQS. Firstly we need to count the number of messages waiting in the queue and the running capacity of the auto-scaling group. Then we calculate the backlog per instance by dividing the number of messages available for retrieval in the queue by the fleet's running capacity; in our case, the number would be 4000, which is calculated from our approximate number of messages divided by the number of Inservice instances. After defining the capacity 4000, we make use of it to deploy the auto scaling action and cloud watch alarm by AWS CLI.

```
$ cat ~/config.json
{
   "TargetValue":4000,
   "CustomizedMetricSpecification":{
      "MetricName":"MyBacklogPerInstance",
      "Namespace":"MyNamespace",
      "Dimensions":[
         {
            "Name":"MyOptionalMetricDimensionName",
            "Value":"MyOptionalMetricDimensionValue"
         }
      ],
      "Statistic":"Average",
      "Unit":"None"
   }
}
```

7. CloudWatch

Based on the actions on EC2 from auto scaling service, it is necessary for us to deploy Cloud Watch service by AWS CLI. Three unique kinds of alarms in views of network out, CPU utilization and consumed write and read capacity units have been deployed in order to monitor the state of our EC2. For network out, if network out is over 6000000 or less than 2000000 for 1 data point within 5 minutes, alarms will be triggered automatically. Also, if CPU utilization is over 70% or less than 25% for 1 data point within 5 minutes , and

consumed write or read capacity units is more than 240 for 5 data points within 5 minutes, alarms can also be triggered automatically.

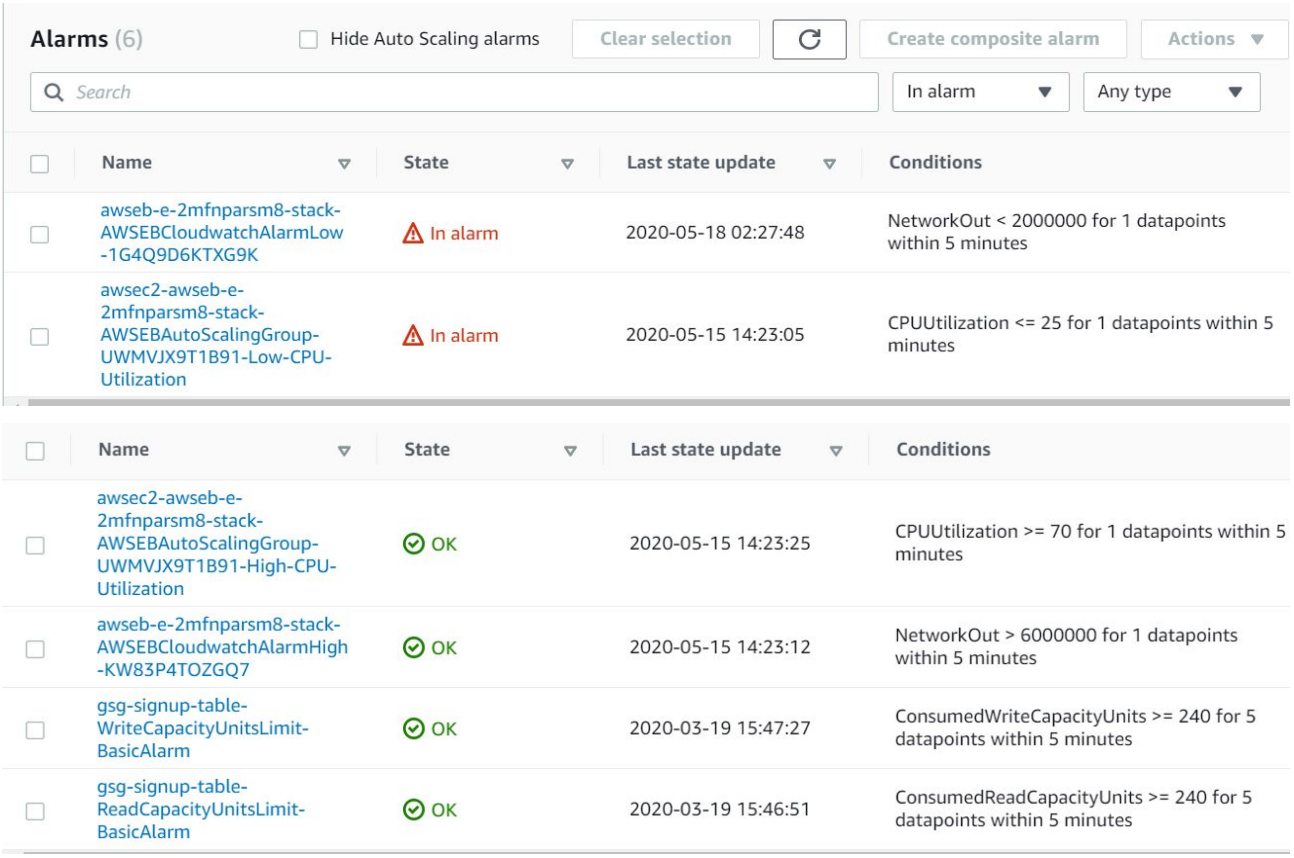


Figure 13. Alarm state on inservic EC2

Pricing

Another advantage of deploying applications in the cloud would be its cost-efficiency. Below demonstrates the main resources we have used and their costs. Compared to local deployment and maintenance, it verifies its strong competitiveness in today's market.

AWS Service	Free tier	Cost
Amazon Simple Queue Service (SQS)	1,000,000 Requests	$0.50 for 1 million requests
EC2	750 hours per month (t2.micro)	$0.0114 per Hour (t2.micro)
S3	5 GB	$0.023 per GB
Simple Email Service (SES)	62,000 Messages per month	$0.10 for every 1,000 emails
AWS Auto-scaling	Free	Free
CloudWatch	10 Custom Metrics and 10 Alarms 1,000,000 API Requests	$0.01 per 1,000 requests
Elastic Beanstalk	Free	Free

Alternative Solutions

Throughout the development of this project, the final product deviated from the very first draft to some extent. The adjustments are made due to either better practice consideration, or simply practicability. Below lists several deviations worth mentioning.

Simulation: simplified model design

The first draft of the simulation would be to include more detailed individual information including profession, daily agenda, transportation means etc. Along with the development process, we realize these parameters were less useful and took the instance longer time to process. Instead, we simplified the individual indicators with simple elements directly related to mortality rate (i.e. age) and their mobility situation (i.e. speed, destinations).

User interface: node.js vs Django

At first, we developed a simple HTML static webpage to interact with Lambda functions and collect the clients' simulation requests (shown below). With reference to AWS official document, we used AWS JavaScript SDK to deploy and call AWS Lambda functions, where the `simulation.py` is run in the cloud environment. We installed Node.js locally to run various scripts that help set up the Amazon S3 bucket, and created and configured the Lambda function. The Lambda function itself runs in the AWS Lambda Node.js environment. (node is pre-installed in the testing machine).

However, during the stage, the prerequisites were onerous: the browser script needed to be manually updated to the designated S3 bucket in order to be accessible. Everytime we change functionality within Lambda, the REST API embedded within the static webpage requires re-configuration. Besides, the static website was built through JavaScript, which technically demanded more knowledge and effort from us.

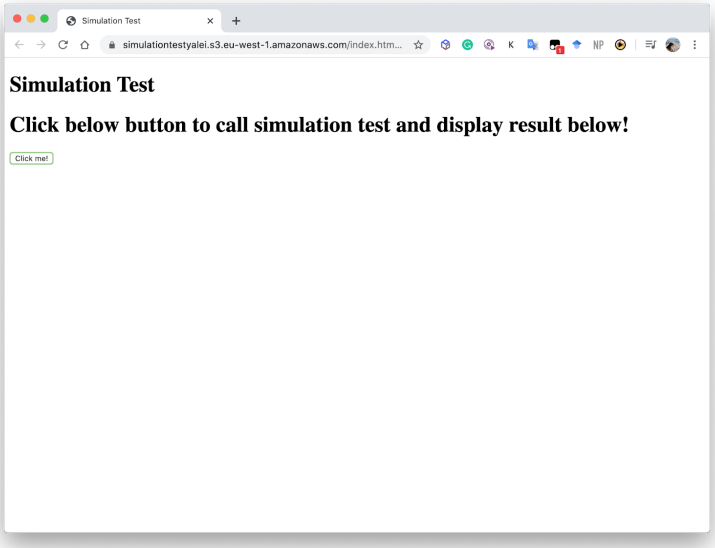


Figure 14. Initial Interface Page

With the EB-embedded Django website, the Python environment mitigated our development tasks, where we can focus on the cloud architecture instead of modifying the less valued element. Given the former practice, we considerably took the advantage and established a website with more flexibility and availability.

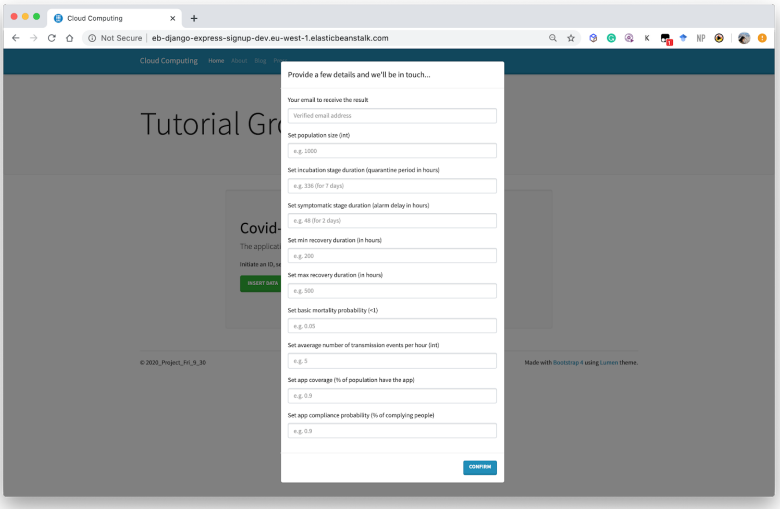


Figure 15. Final Interface Page

Program computation: Lambda vs EC2

The initial idea is to deploy the program to AWS Lambda function. We locally tested the possibility of deploying AWS Lambda functions and calling through REST API. (Detailed documents can be found in our sprint report 02). The deployment was troublesome: multiple prerequisites (Docker, AWS SAM CLI, PyCharm AWS Toolkit), multiple programming languages (Python for the main functionality, and AWS SDK for JavaScript to call Lambda functions), and the dreadful runtime restrictions of the module (maximum of 15 minutes). When we made the Lambda functions exposed to other services through APIs we also needed some workarounds to handle the API timeout (29 seconds). At the end, we decided to move the simulation program to EC2 instances, which is more suitable for computationally intensive jobs.

Result delivery: SES vs Kibana

Our initial idea is that we can make use of Kibana to have better visualization results. However, we realized that we can implement SES to simplify the process and still achieve the desired result. Aesthetically it was not as good as Kibana but we still managed to convey our intended message to the clients. The decision to choose SES is also based on the least-cost rule and better interaction between services. We created the visualization, saved the results into S3, and sent the email to clients automatically in `simulation.py`.The result is shown below

Dear user,

Here is the simulation report based on the parameter that you provide.
Simulated population:1000
Contact Tracing System:True
The number of deaths decreases by: 70.21%
The number of infected people decreases by: 71.85%

Report

Simulation without Contact Tracing System

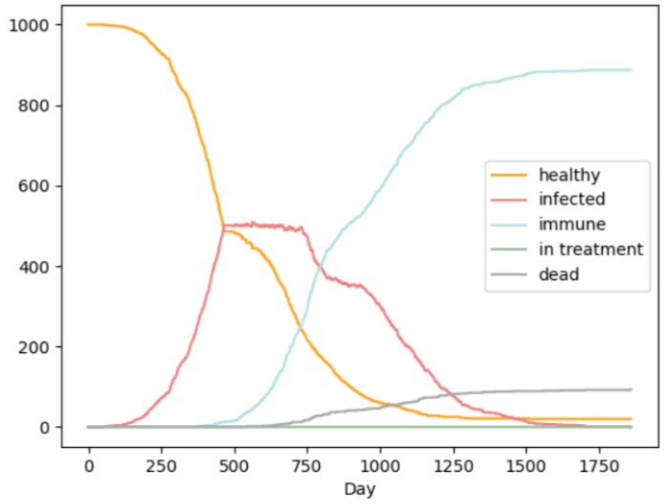


Figure 16. Emails content of simulation results

Challenges

Throughout the development of the project, several problems were encountered but solved eventually. Through the problem solving experience, we deepened our understanding of the cloud service infrastructure, team organization, and technical abilities. Major challenges are demonstrated as follows.

Coding

The most challenging part of the project would be to reenact the COVID19 infection model based on a large population. We referred to open source code for the basic structure and added our own thoughts. The majority of the efforts at the beginning were devoted to understanding different sources regarding this infectious disease. Our group meeting proposed multiple plans for embedding the contact tracing system, and eventually theoreticized the idea to resemble self-isolation mode.

Next thing is how to bring the simulation to the cloud using cloud services. We decided to use Python to connect AWS services together. Different tools and libraries are considered to interact with those services. We have some difficulties at the beginning with programming on the cloud since it is different from programming locally. Debugging on the cloud is not easy but at the end we managed to do it.

Team organization

When we get to know this project for the first time, we have no idea about how to organize our team tasks and finish our projects. Some people have less programming background and most team members are not familiar with AWS. Facing these troubles, we try our best to assign tasks based on personal wishes and programming background. Thankfully, the weekly sprint meetings with the Professor were really helpful in clearing doubts about tool selection and cloud implementation. Eventually, we managed to find a clear direction and be more efficient in splitting tasks and finishing the project.

Specifically, we organized a group discussion once or twice a week to present our tasks results respectively and solve some problems that we encountered. Also, everyone needs to finish Github reports once a week for their part. People who do well in programming can help others about coding problems during discussion while others who are not good at programming can pay more attention to AWS configuration and learn programming knowledge gradually. Therefore, everyone can learn from each other thus improving the efficiency and productivity.

Services

When it comes to making a decision of which services to use, we have to take into account multiple criteria at the same time such as cost, time limits, scalability, concurrency, etc. It takes a large amount of time to try out some services and evaluate them. For example, we spent one sprint to deploy our simulation to a Lambda function and it turned out not to be an efficient solution due to its time limits. Therefore, we moved the simulation to EC2 instances which are more suitable for our needs.

Resources used

Another major challenge is to determine the available but practical cloud resources to establish our product. After our initial research, we proposed several tempting solutions which involve multiple ecosystems and decrease the overall usability. Proposals include Amazon EMR (to process massive amounts of data), Apache ecosystem (distribution storage and analysis), Amazon Redshift (data warehouse) and so on. However, those complicated tools diverted our focus in delivering a simple but usable product. After a couple of extensive group discussions, we realized that most of the proposals were out of the project scope and abandoned the impractical structure.

Conclusion

Thus far, this report documents the whole process of the project development in detail. We built the structure of this COVID-19 infection simulation application, and deployed it with full AWS cloud services, which enables the application with considerable flexibility, availability and usability. The simulation models reflect our understanding of the disease model and embedded with engineering thinking. We implemented the complex agent model into practical programming logic, and eventually established the functional prototype for our future clients.

In addition, the intact cloud-based infrastructure not only offers 24/7 stable service for our clients, but also eases the entire development process. We deployed the application on the Django website supported by Elastic Beanstalk, where our consumers can access our service and send simulation requests to the designated SQS queue, which handles requests through a First In First Out (FIFO) queue. Multiple EC2 instances are set up to constantly retrieve jobs from the queue and initialize new requested simulation jobs in parallel. These workers are auto-scaled in and out by AWS Auto-scaling service which takes into account their CPU utilization. Once the simulation is completed, the results are stored in a S3 bucket and sent to our clients in an email by AWS SES service. After that, they start a new cycle by getting the next request from SQS.

Notwithstanding the functionalities and benefits, we are accurately aware of the insufficiency of our project. Due to the lack of time and knowledge, our final product deviated from the original plan to some degree: the contact tracing model could be more complete and realistic by introducing more parameters, and more cloud resource can be included to improve the website stability and usability (e.g. Amazon CloudFront to offer global accessibility, Apache tools to increase the scalability of simulation processing). Due to the time limit, we did not implement the real-time visualization of the final result through a dynamic website, given the consideration of the subject scope. In another scenario, Elastic stack can be used to present more appealing graphic results.

From a learning point of view, we believe that we have dramatically improved our knowledge about various AWS services (and their advantages) as well as implementing a project using a cloud architecture. As mentioned above, initially we were confused about the scope and how to implement the project but gradually we managed to create our version of the project architecture and understood how to leverage each AWS product to help us in the implementation.

Overall, this product satisfies our preliminary proposal of tackling the project requirements. The cloud-empowered web application is simple yet practical to provide adequate information for our potential clients to assist their understanding in the disease infection. The whole development journey is demanding but inspirational to every one of us. This project offers a significant chance for us to actually work with cloud infrastructures and get to know how different services can be interlinked to each other. We enriched our

knowledge of cloud computing and gained deeper insights in this area. This project will be very beneficial as part of our knowledge tool-box later on, which expands our career choices without doubt.