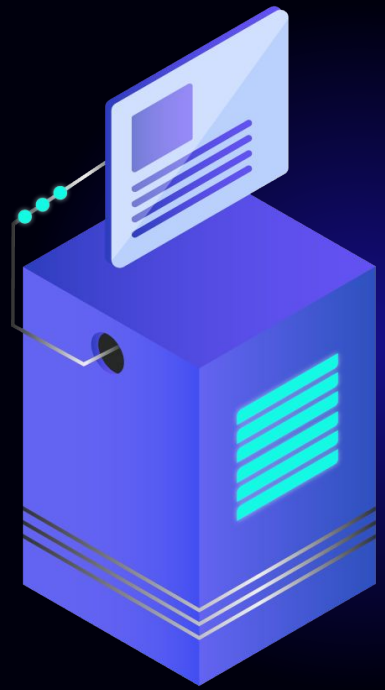
Deploying a flask app on an AWS EC2 instance

Temperature Convertor Application



INTRODUCTION

In this project a temperature conversion application was deployed on an AWS EC2 instance. The application is where you can enter a temperature in either celsius or fahrenheit and get its converted value in fahrenheit or celsius, respectively. Also the Functionality and auto scalability were evaluated.

TABLE OF CONTENTS

01

Flask App

In this section, we will explain our application, and how we code it.

Flask App

- A Flask application is an instance for the **Flask** class.
- Everything about the application, such as **configuration** and **URLs**, will be registered with this class.

Flask App - step 1

- First, we imported Flask and request from the flask.
- Next, we created a Flask instance using this command: `app = Flask(__name__)`.
- Then, using command `@app.route('/')` a simple route was created.
    - It creates a connection between the **URL** and a **function that returns a response**.

Flask App - step 2

HTML was used as a markup language to build our application's frontend.

```
6   def homepage():
7       website = '''
8   <!DOCTYPE html>
9   <html>
10  <body>
11      <center><img src="https://www.pngkey.com/png/full/297-2976588_temperature-conversion-convert-temperature-icon.png"
        width="640" height="250">
12      </center>
13      <h1><center>Welcome to our Temperature Convertor application</center></h1>
14      <br>
15      <h2><center>Enter a value in Celsius or Farenheit, and it will be converted</center></h2>
16      <center><form action="tempConversion" method="POST">
17      <label for="temp-type-select">Choose a temperature type:</label>
18      <select name="types" id="type-select">
19      <option value="">--Please choose an option--</option>
20      <option value=0>Celcius</option>
21      <option value=1>Farenheit</option>
22      </select>
23          <input type="text" name="tempInCelsius">
24          <input type="submit" value="Convert">
25      </form></center>
26
27  </body>
28  </html>'''
29      return website
```

Flask App - step 3

Finally, we defined another function which allows the user to select one option between celsius and fahrenheit and enter a value. Then, the converted temperature will be returned as a result.

temp-convertor M

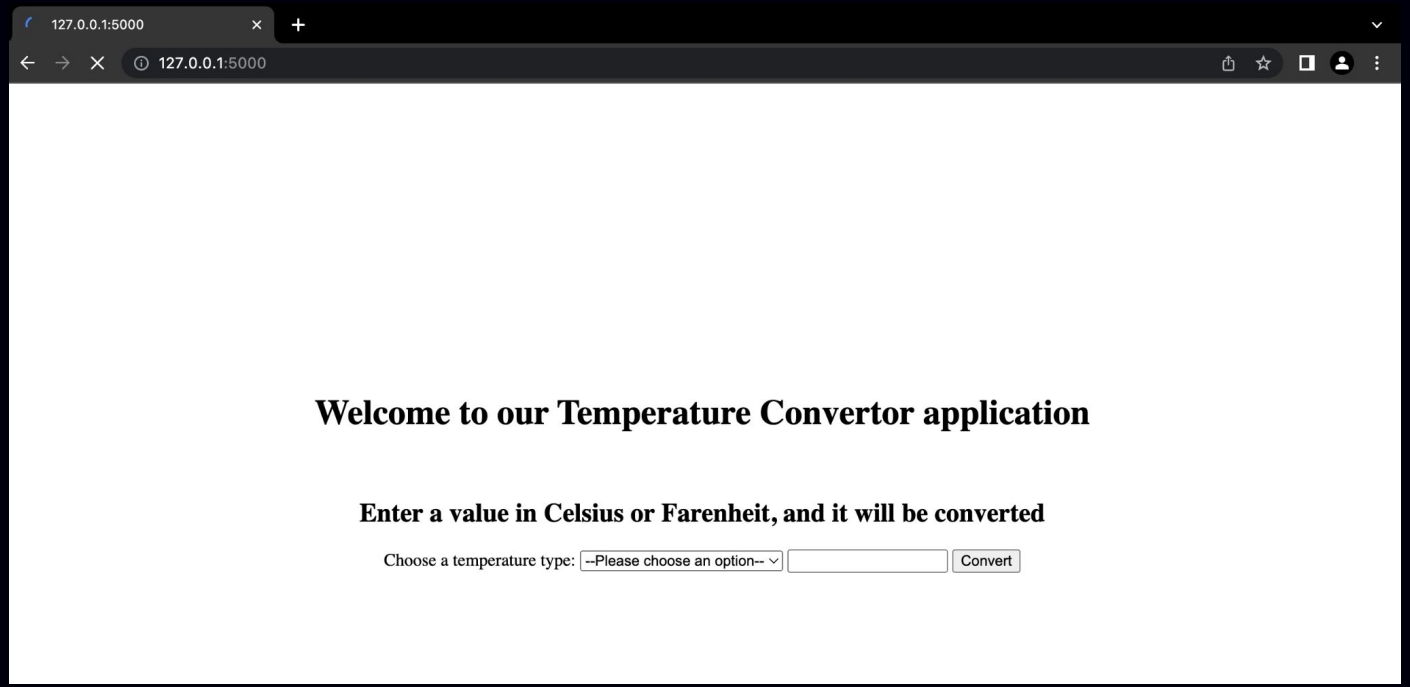
```
@app.route('/tempConversion', methods=['POST'])
def my_form_post():
    temp = request.form['tempInCelsius']
    option = request.form['types']
    try:
        if option == '0':
            celsiusTemp = float(temp)
            fahrenheitTemp = (celsiusTemp * 9.0/5) + 32
            return str(fahrenheitTemp)
        elif option=='1':
            farenheitTemp = float(temp)
            celsiusTemp = (farenheitTemp - 32) * 5.0/9
            return str(celsiusTemp)
        else:
            return "You should first choose a temperature type. Please go back and try again."
    except:
        return "That's not a number!"



if __name__ == '__main__':
    app.run()
```

Flask App - Result

By visiting http://127.0.0.1:5000 in a browser we were running our Flask web application on a local host.

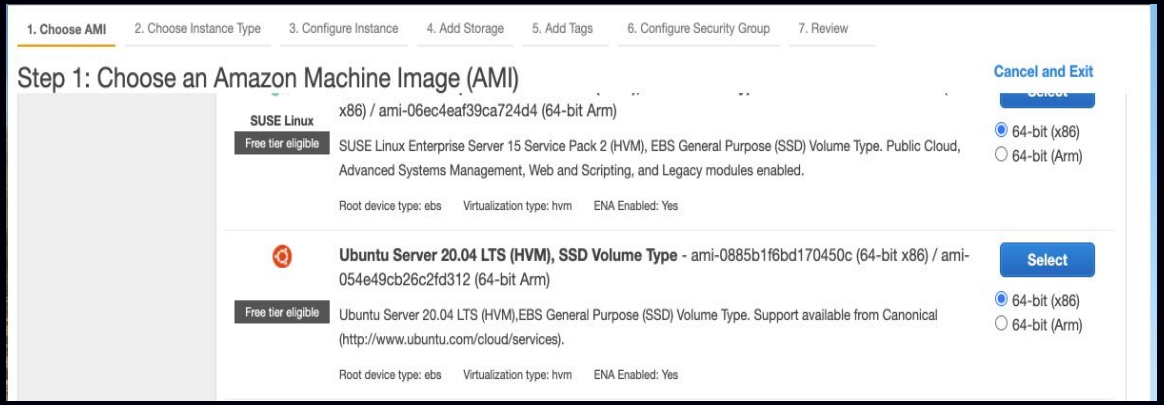


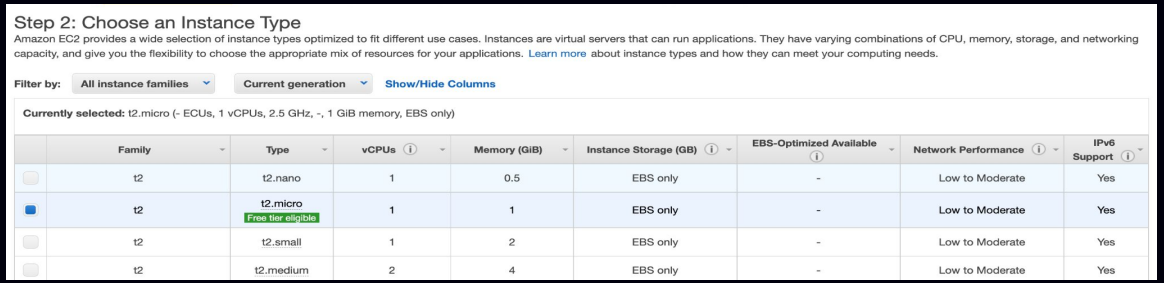
02

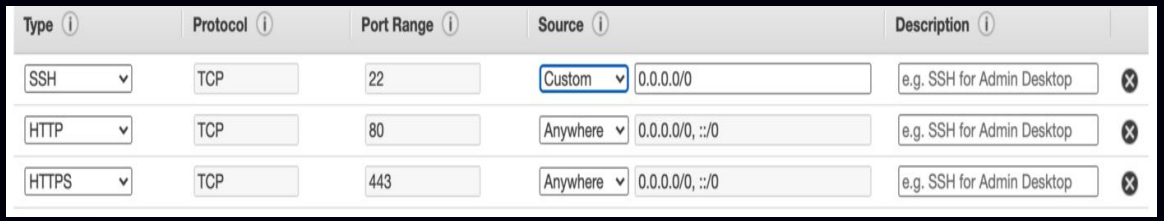
Deployment on an AWS EC2 instance

Step1 : Launch a new EC2 instance

- Opening EC2 console
- Select Ubuntu server 20.04
- Instance type: t2.micro
- Create a security group with following inbound rules:
  - SSH
  - HTTP
  - HTTPS
- **SSH** setting allows ssh access from any IP address
- **HTTP** allows to access to the new instance from the browser
- **HTTPS** is for running secure connections

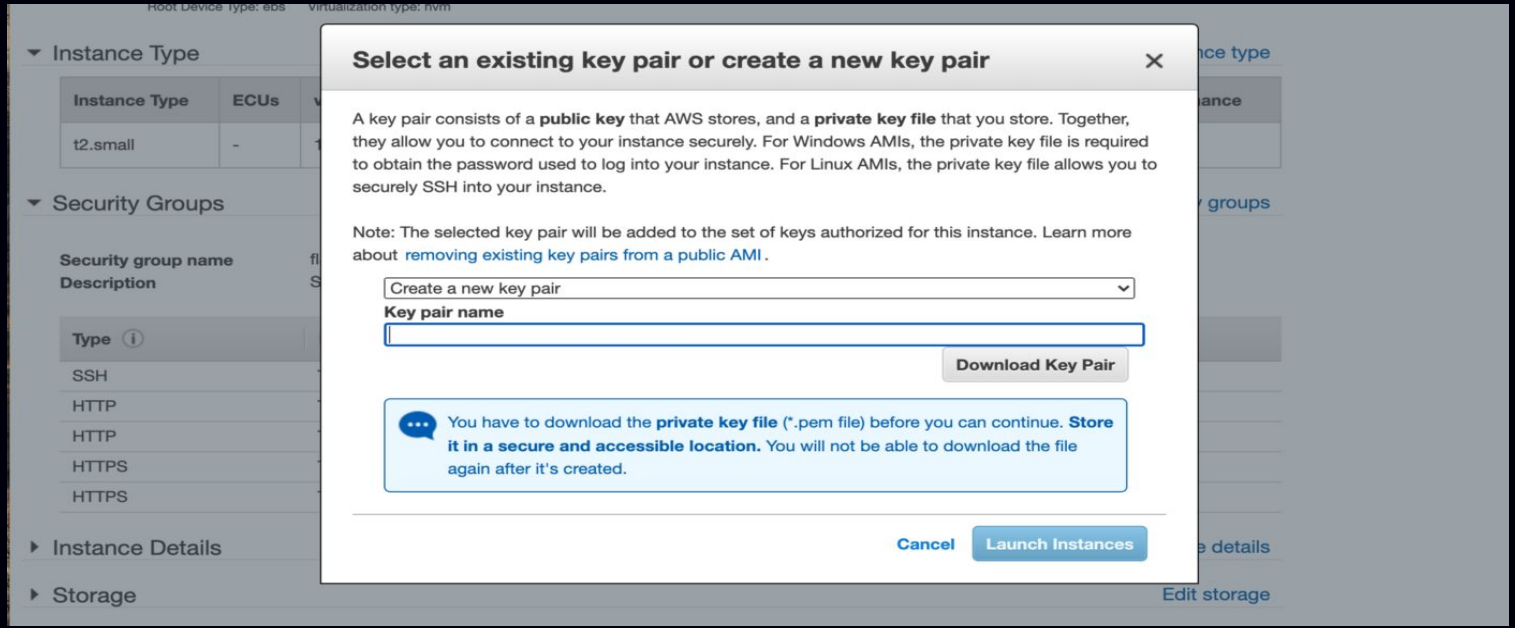




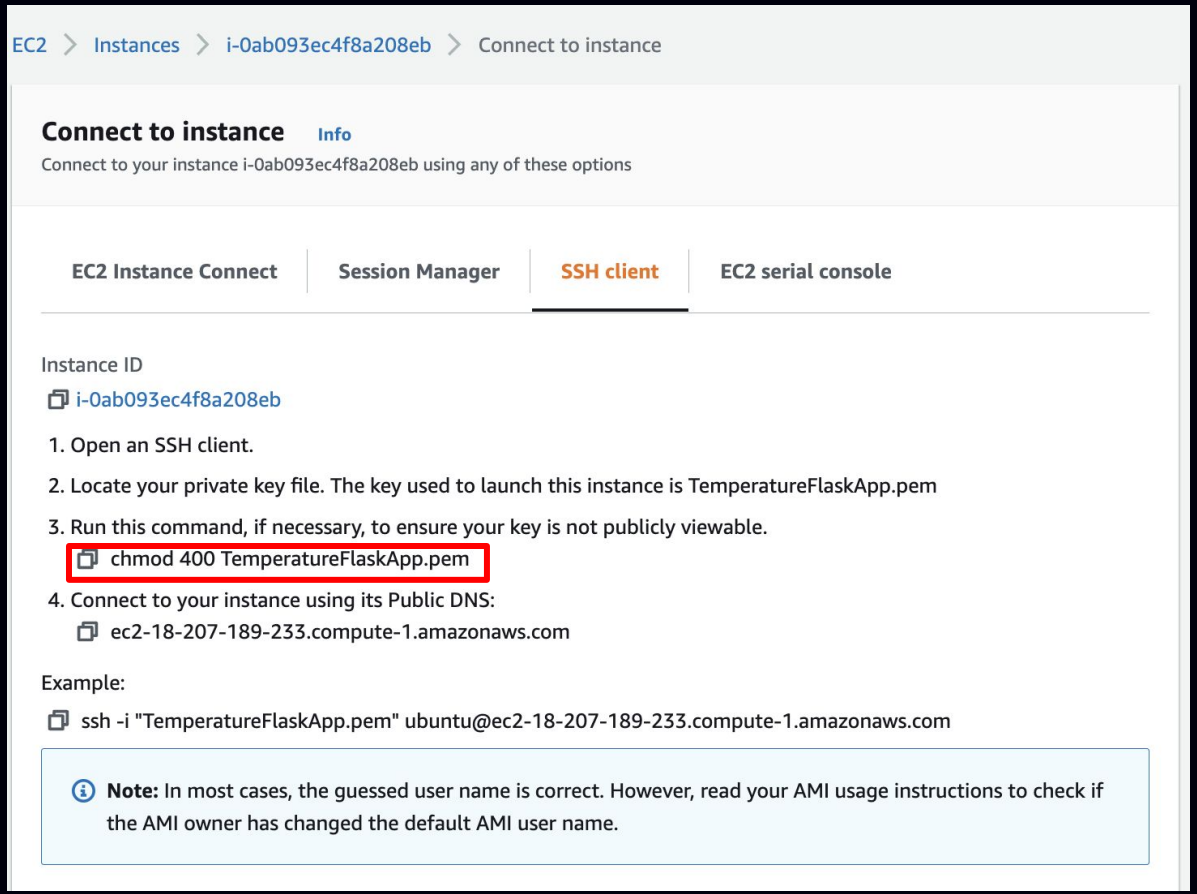


Step1 : Launch a new EC2 instance

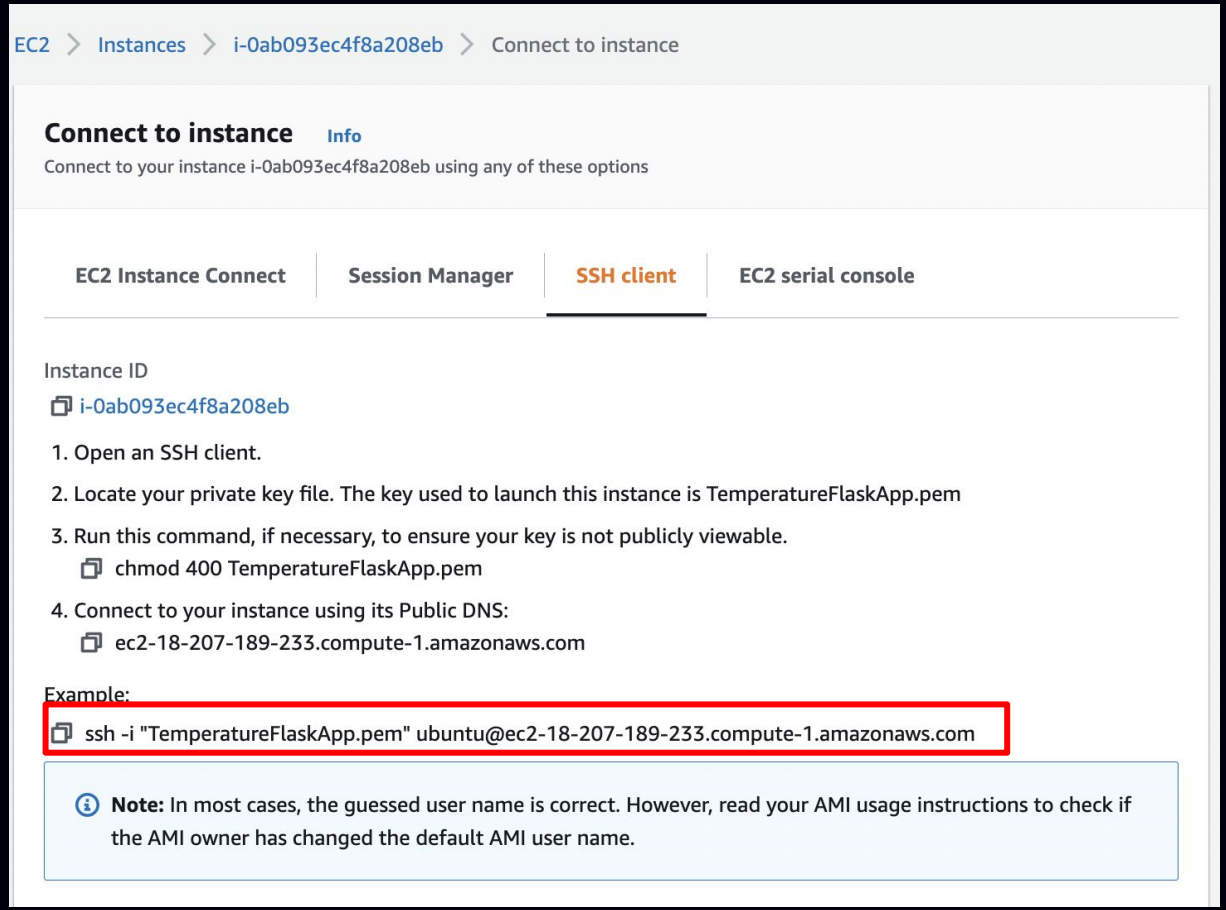
- Create a key pair and save it



Step 2: Update the permission of private key



Step 3: Connect to our new instance via SSH



Step4: Update , upgrade and set up the environment

- Update the local package index and upgrade the system using following commands:

```
$ sudo apt-get update
$ sudo apt-get -y upgrade
```

  - This makes sure everything is up to date and prevents any errors due to deprecations

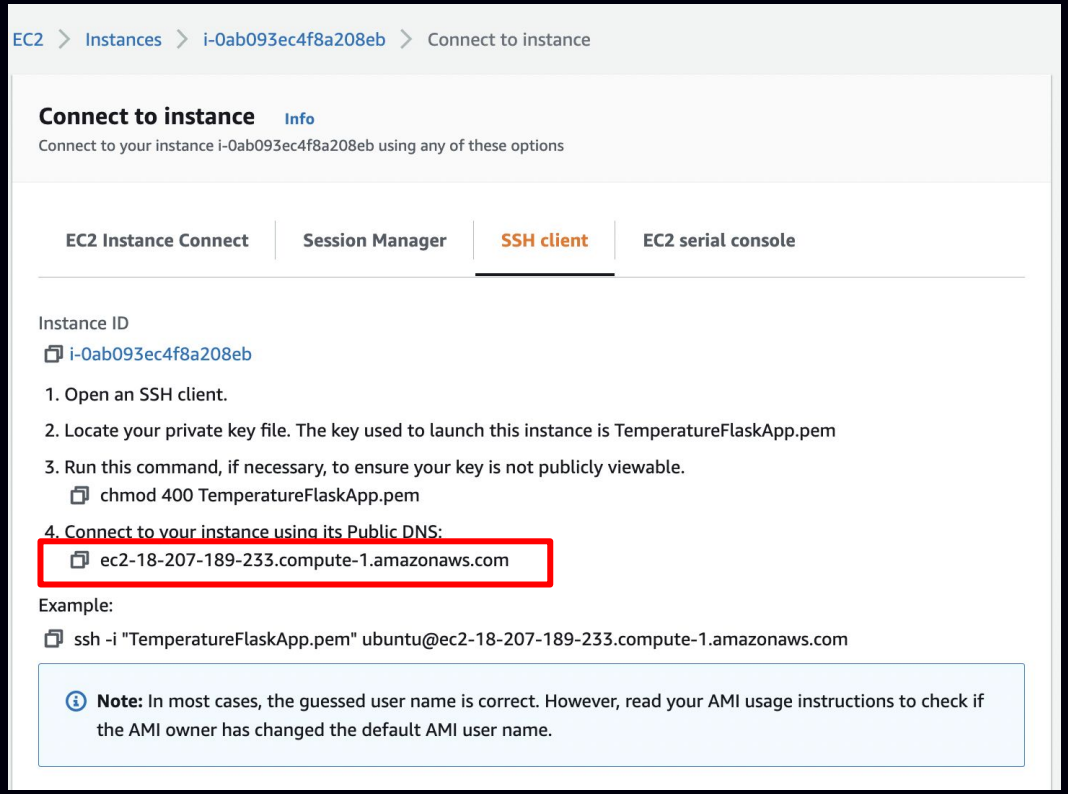
- Install Python
- Install PIP
  - It is needed to manage software packages for python

Step 5: Install and configure Apache and WSGI

- Install the Apache webserver with the **mod_wsgi** module using the command below to interface with our *Flask* app.

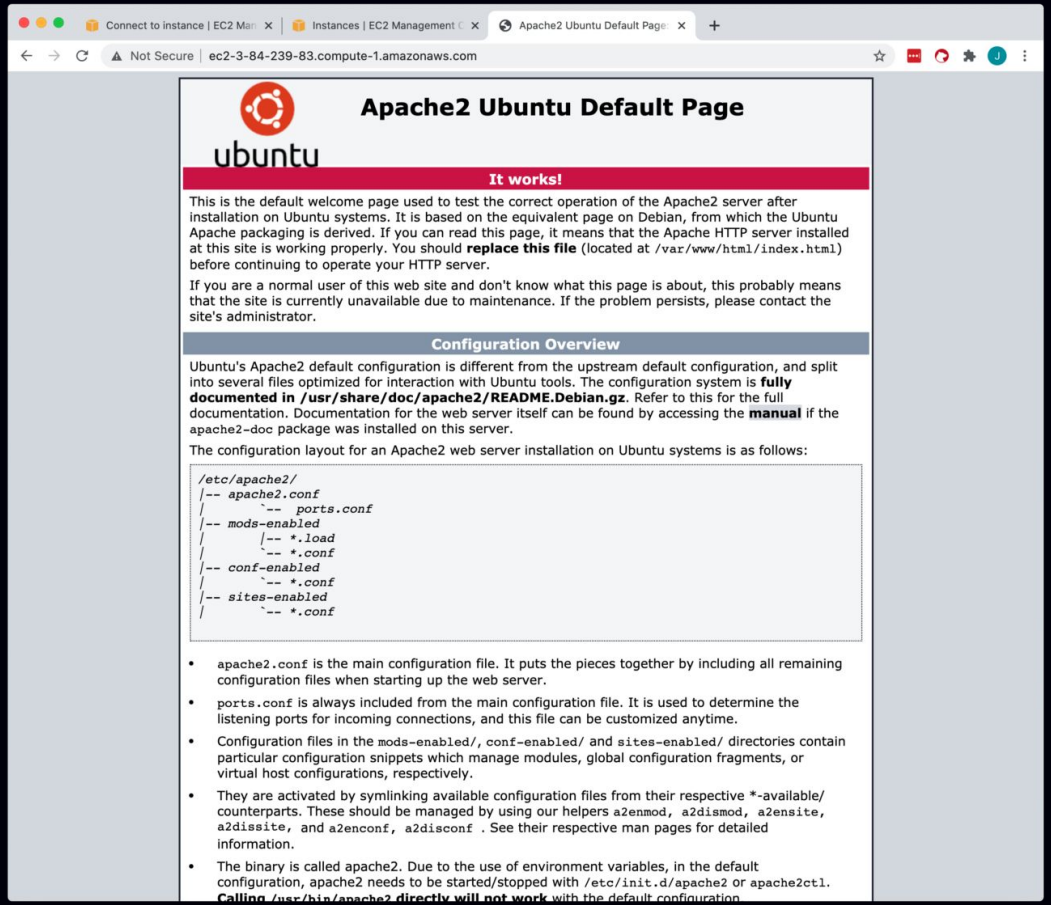
  ```
  $ sudo apt-get install -y apache2
  ```

  - It is required to interface with our flask app because web servers do not understand python and WSGI makes the communication happen.
- By opening our instance's public DNS in a browser we saw Apache's default page, indicating the installation is working correctly.



Step 5: Install and configure Apache and WSGI

Apache's default page is indicating that the installation is working correctly.



Step 6: Import flask app in python file .(run.py)

temp-convertor

```python
from flask import Flask
from flask import request
#from flask import render_template

app = Flask(__name__)

@app.route('/')
def homepage():
    website = '''
```

```python
@app.route('/tempConversion', methods=['POST'])
def my_form_post():
    temp = request.form['tempInCelsius']
    option = request.form['types']
    try:
        if option == '0':
            celsiusTemp = float(temp)
            fahrenheitTemp = (celsiusTemp * 9.0/5) + 32
            return str(fahrenheitTemp)
        elif option=='1':
            farenheitTemp = float(temp)
            celsiusTemp = (farenheitTemp - 32) * 5.0/9
            return str(celsiusTemp)
        else:
            return "You should first choose a temperature type. Please go back and try again."
    except:
        return "That's not a number!"



if __name__ == '__main__':
    app.run()
```

Step 7: Configure virtual host

- Apache displays HTML pages by default but to serve dynamic content from Flask we made the following changes.
- The default Apache configuration file is located at etc /apache2 / sites-available/000-default.conf. Instead of overriding that file we created a new one.
- $ sudo vi / etc / apache2 / sites -available/microservices.com.conf
- Add the following to the microservices.com.conf

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/html/microservices.com
    WSGIDaemonProcess microservices.com threads=5
    WSGIScriptAlias / /var/www/html/microservices.com/production.wsgi
     <Directory microservices.com>
         WSGIProcessGroup microservices.com
         WSGIApplicationGroup %{GLOBAL}
         Order deny,allow
         Allow from all
     </Directory>
     ErrorLog ${APACHE_LOG_DIR}/error.log
     CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Step 8:Reload Apache , and get updated public DNS

- Disable the default Apache config by running the following command.

  $ sudo a2dissite 000-default.conf

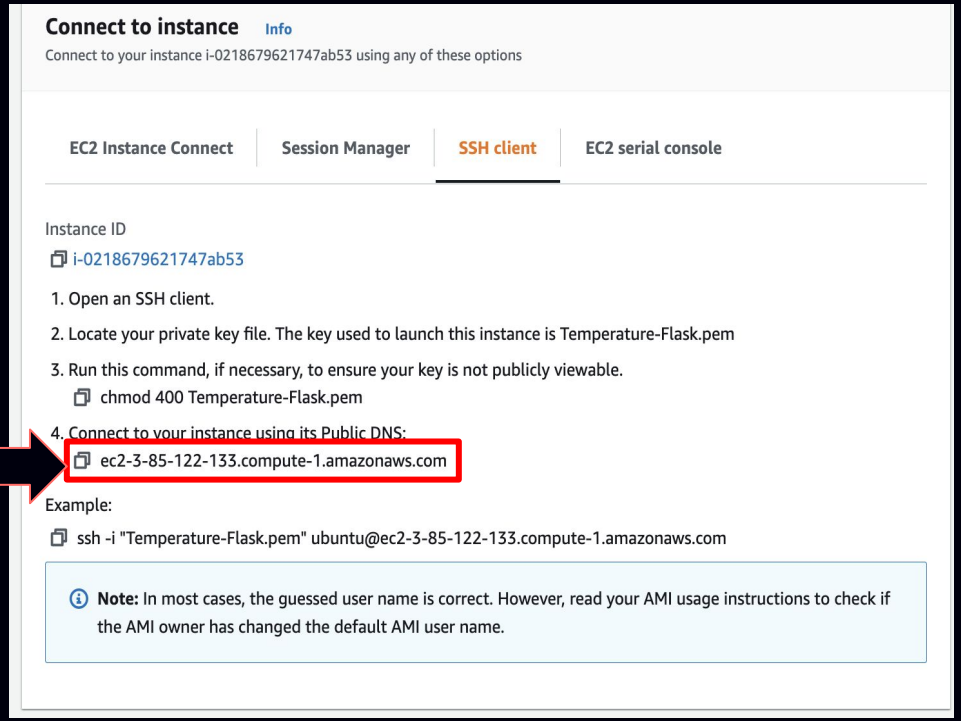
- Enable our new conf

  $ sudo a2ensite microservices.com.conf

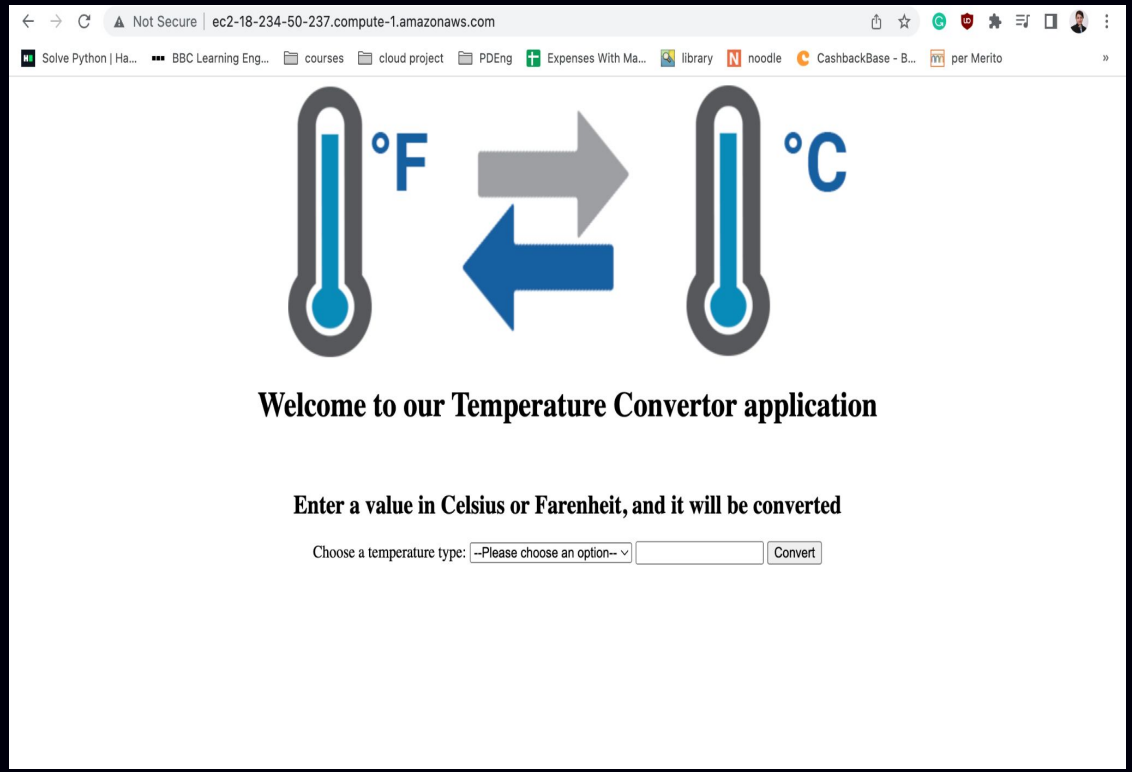
- Reload apache to implement the changes made.

  $ sudo systemctl reload apache2

- **Public DNS link:**



Step 9: Test and run the deployed app



03

Evaluation

Scale and load balance of the Architecture

- **Elastic Load Balancing :**

  Automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables us to achieve fault tolerance in our applications by seamlessly providing the required amount of load balancing capacity needed to route application traffic.

- **Auto Scaling :**

  - It helps us to maintain application availability and allows us to scale our Amazon EC2 capacity out or in automatically according to conditions we defined. You can use Auto Scaling to help ensure that you are running your desired number of Amazon EC2 instances.

  - Auto Scaling can automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

Objective

- **Create an Amazon Machine Image (AMI) from a running instance.**
- **Create a load balancer.**
- **Create a launch configuration and an Auto Scaling group.**
- **Automatically scale new instances within specific subnets.**
- **Create Amazon CloudWatch alarms and monitor performance of our infrastructure.**

Step 1 : Creating AMI for Autoscaling

- First, confirm that the instance is running and Web Server checks passed displayed.
- Create an AMI based upon this instance.
- Select Web Server >click Image and templates > Create image, then configure: (Image name and Image description)
- Click Create image and use this AMI when launching the Auto Scaling group later.

Step 2 :Create a load balancer

- we started with:

  - choosing the Target Groups>Choose a target type: Instances>Target group name and VPC> create target group.

  - Create Load Balancer and enter our Load balancer name.

  - Network mapping section, we selected:
  VPC and specified which subnets the Load Balancer should use. The subnets Us-east-1a Us-east-1b

- Next Security groups section selection:

  We created a security group named Tempreature-Group with HTTP , HTTPS and SSH

- We created a load balancer successfully created and created an AMI based upon this instance

- Finally click to create image and use this AMI when launching the Auto Scaling group later

Step 3 :Create a launch Configuration

- First in the left navigation pane, click Launch Configurations>Create launch configuration

- Configure these settings:

  Launch configuration name>Amazon Machine Image (AMI) >Instance type

  Monitoring: Select Enable EC2 instance detailed monitoring within CloudWatch to allows Auto Scaling to react quickly to changing utilization.

- Security groups configuration: Choose Select an existing security group

- Key pair configuration: Key pair options >Existing key pair

- Finally ,create launch configuration

Step 4 :Create an auto scaling group

- **We created an Auto Scaling group that uses our previous Launch Configuration.**

  Enter Auto Scaling group name > Next > Network page configures>VPC Network > Subnet > Next

  Attach to an existing load balancing
  CloudWatch monitoring under Group size configure:

  Desired capacity: 2 Minimum capacity: 2 Maximum capacity: 5
  Under Scaling policies, choose Target tracking scaling policy and configure:

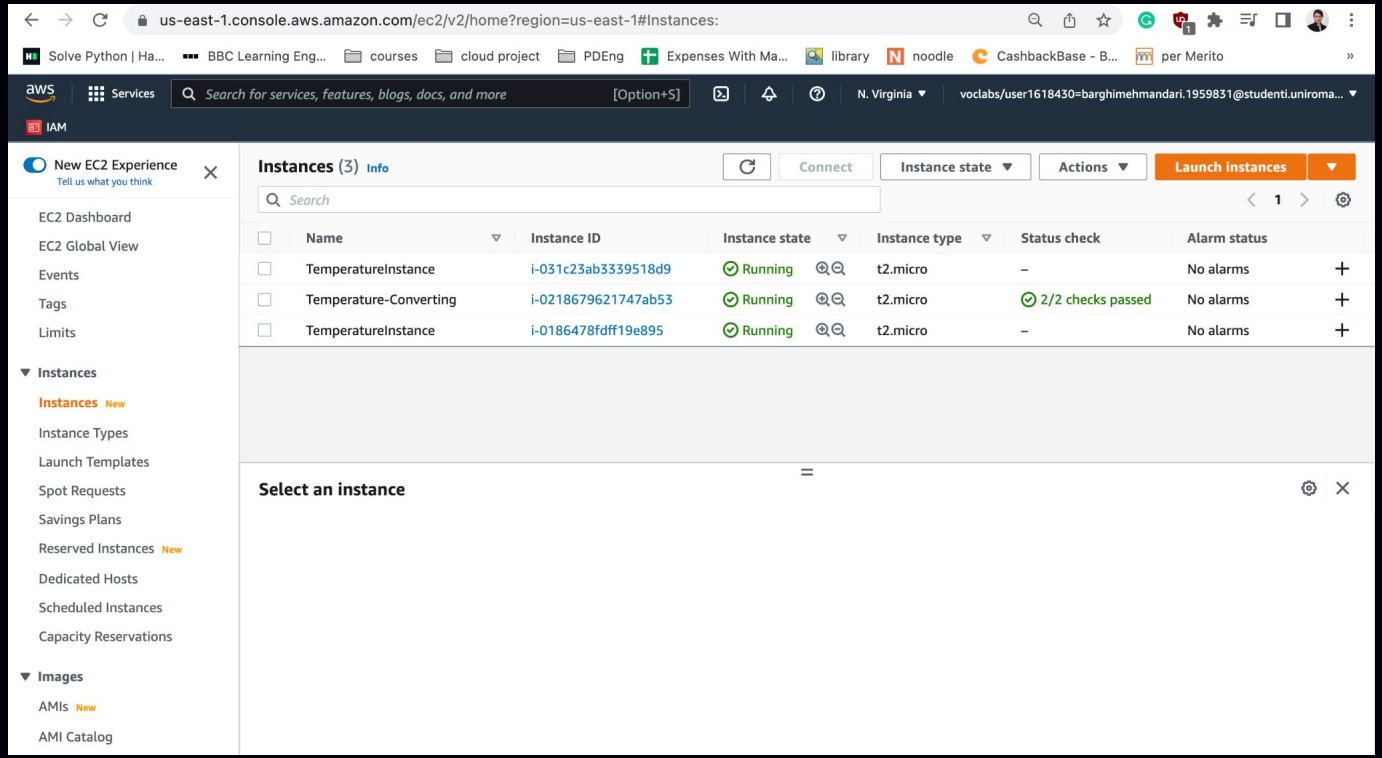
  - **policy name>Metric type>Target value> Next**
    Choose Add tag and Configure the following:
  - **Key and Value>Next**
    Add notification: We added an email to receive an email when the alarm is triggered.
- **Our Auto Scaling group will initially show an instance count of zero, but new instances will be launched to reach the Desired count of 2 instances.**

04

Results

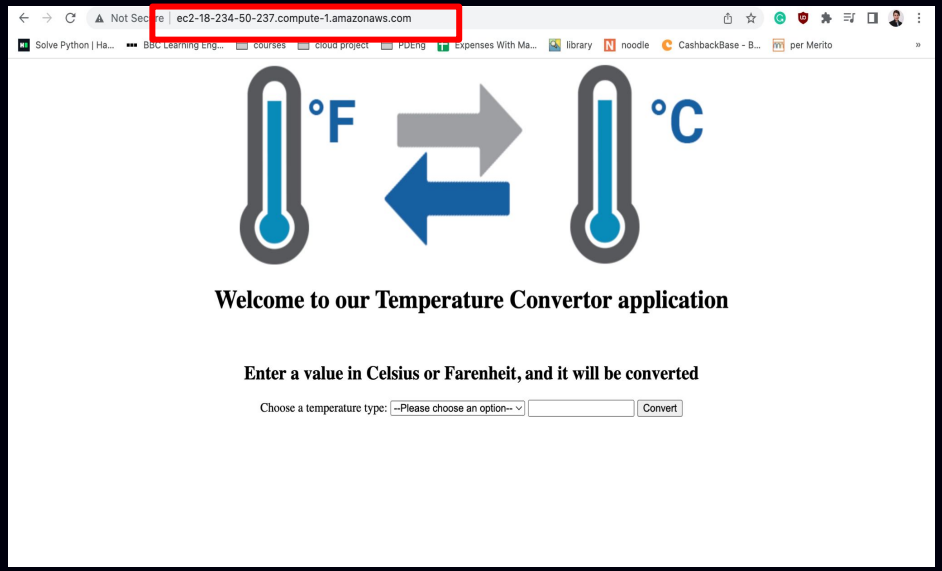
1. Verify that load balancing is working

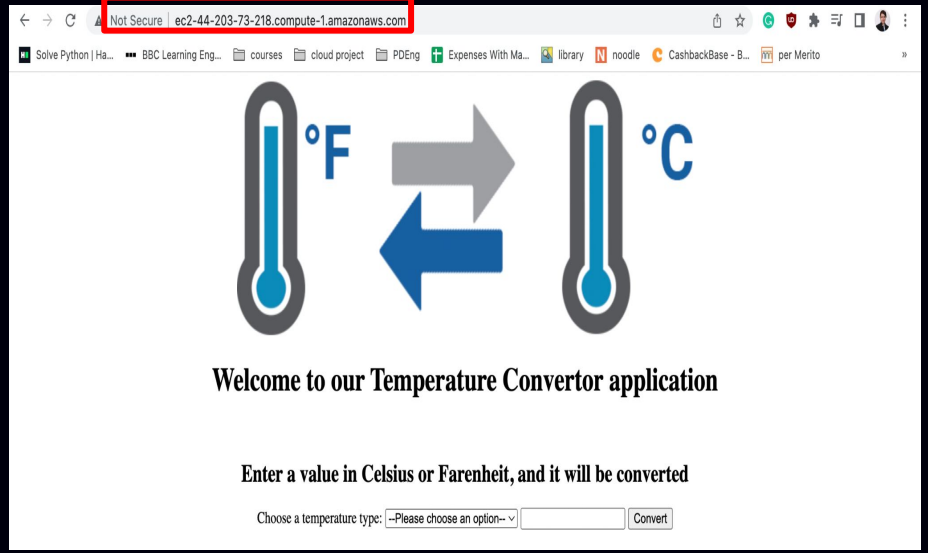
- We created an Auto Scaling group that uses Launch Configuration. The two new instances were created because we set the minimum number of instances to 2.



1. Verify that load balancing is working

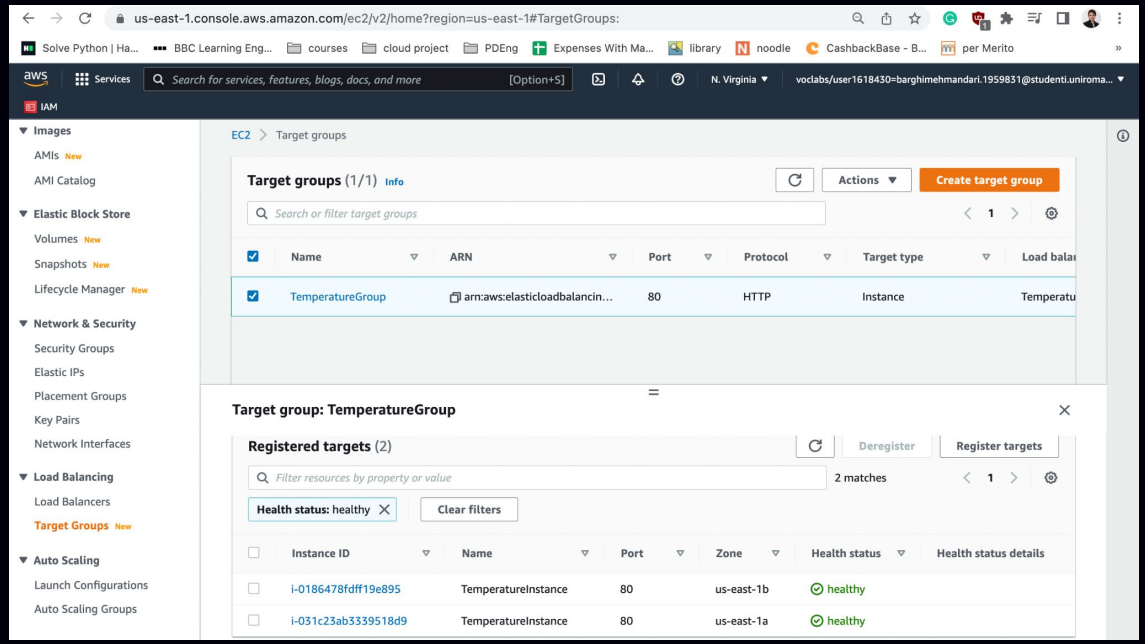
- Now we can access the Auto Scaling group via the Load Balancer. By copy and paste the DNS after connecting of new instances that were created.
  The application appeared in our browser. This indicates that the Load Balancer received the request, sent it to one of the EC2 instances, then passed back the result.





1. Verify that load balancing is working

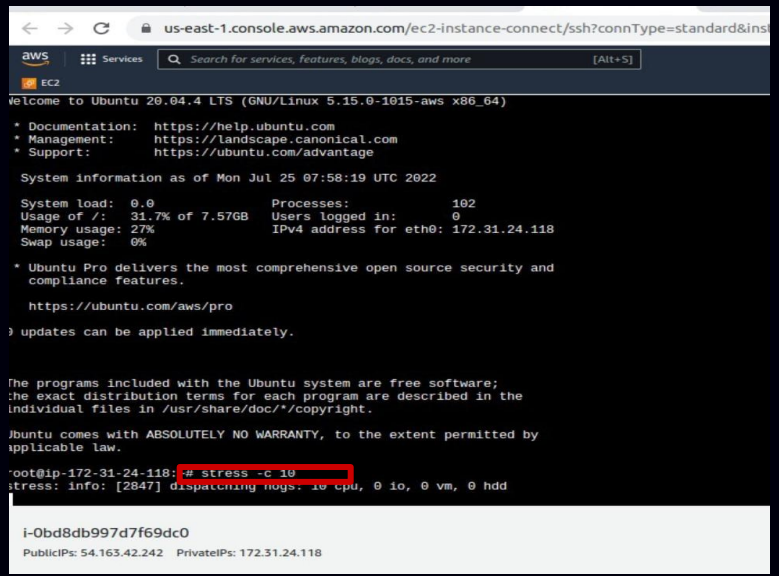
- In the Target Groups (in the Load Balancing section) two “TemperatureInstance”targets were listed for this target group. Healthy signs indicates that an instance has passed the Load Balancer's health check. This means that the Load Balancer will send traffic to the instance.



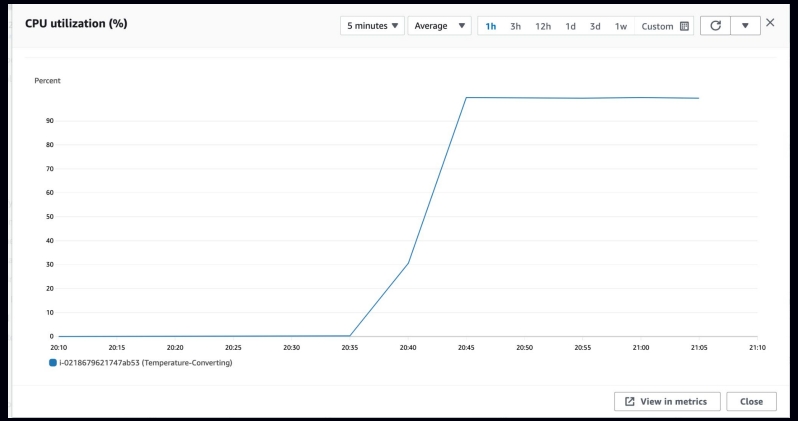
2. Stress Test

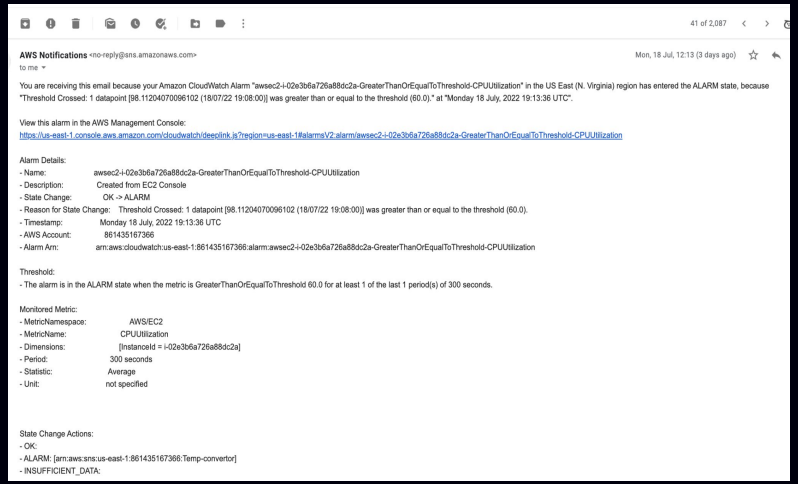
- We increased the stress and create an alarm for the instance that we have. We set it to be triggered when the cpu utilization is higher than 50% and also we will receive an email when the alarm be activated.

```
ubuntu@ip-172-31-87-196:~$ sudo apt install stress
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  stress
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 18.4 kB of archives.
After this operation, 55.3 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 stress amd64 1.0.4-6 [18.4 kB]
Fetched 18.4 kB in 0s (409 kB/s)
Selecting previously unselected package stress.
(Reading database ... 97048 files and directories currently installed.)
Preparing to unpack .../stress_1.0.4-6_amd64.deb ...
Unpacking stress (1.0.4-6) ...
Setting up stress (1.0.4-6) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
Processing triggers for man-db (2.9.1-1) ...
ubuntu@ip-172-31-87-196:~$ stress --cpu 90
stress: info: [1392] dispatching hogs: 90 cpu, 0 io, 0 vm, 0 hdd
```



2. Stress Test





41 of 2,087

AWS Notifications <no-reply@sns.amazonaws.com>
to me

Mon, 18 Jul, 12:13 (3 days ago)

You are receiving this email because your Amazon CloudWatch Alarm "awsec2-i-02e3b6a726a88dc2a-GreaterThanOrEqualToThreshold-CPUUtilization" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 datapoint [98.11204070096102 (18/07/22 19:08:00)] was greater than or equal to the threshold (60.0)." at "Monday 18 July, 2022 19:13:36 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/awsec2-i-02e3b6a726a88dc2a-GreaterThanOrEqualToThreshold-CPUUtilization

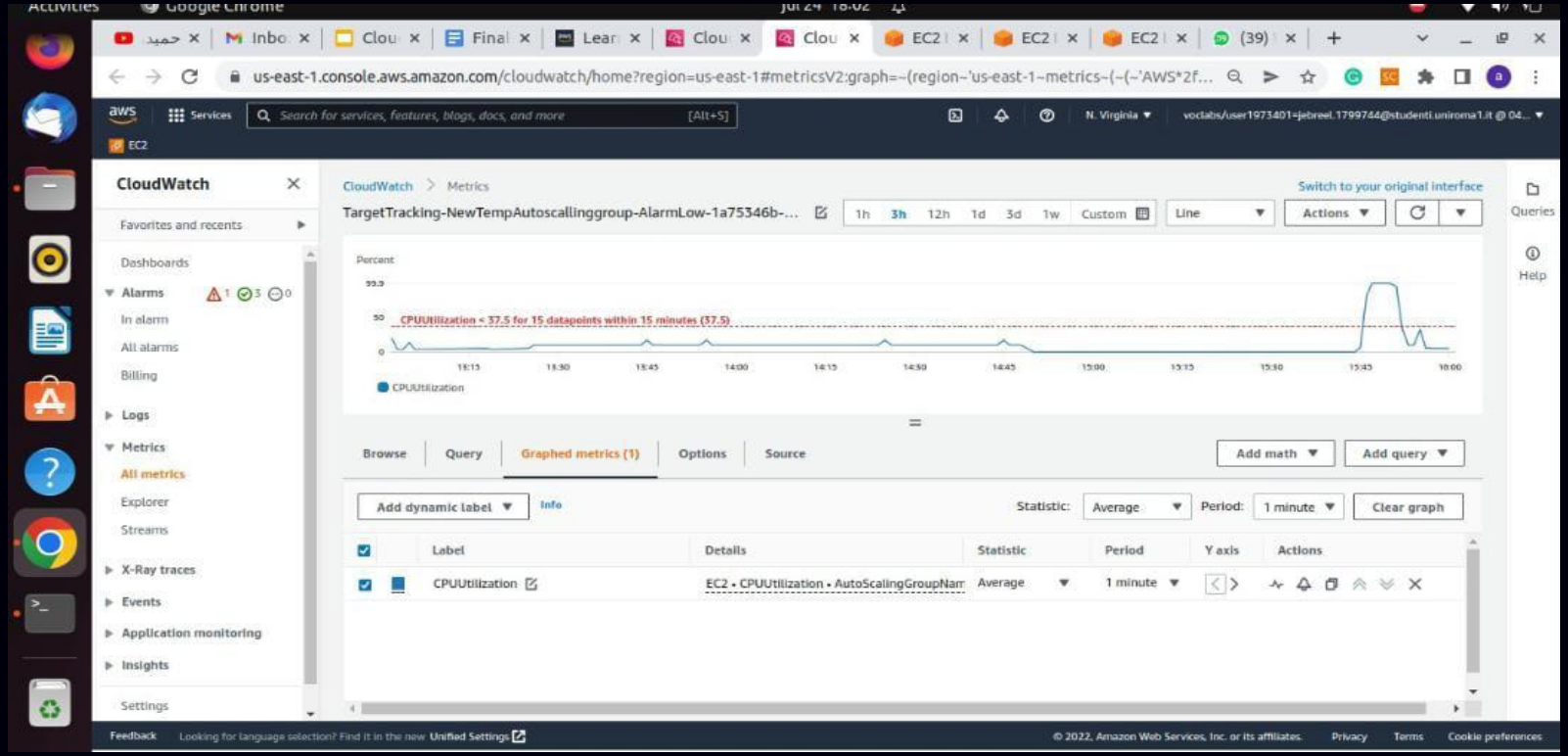
Alarm Details:
- Name: awsec2-i-02e3b6a726a88dc2a-GreaterThanOrEqualToThreshold-CPUUtilization
- Description: Created from EC2 Console
- State Change: OK -> ALARM
- Reason for State Change: Threshold Crossed: 1 datapoint [98.11204070096102 (18/07/22 19:08:00)] was greater than or equal to the threshold (60.0).
- Timestamp: Monday 18 July, 2022 19:13:36 UTC
- AWS Account: 861435167366
- Alarm Arn: arn:aws:cloudwatch:us-east-1:861435167366:alarm:awsec2-i-02e3b6a726a88dc2a-GreaterThanOrEqualToThreshold-CPUUtilization

Threshold:
- The alarm is in the ALARM state when the metric is GreaterThanOrEqualToThreshold 60.0 for at least 1 of the last 1 period(s) of 300 seconds.

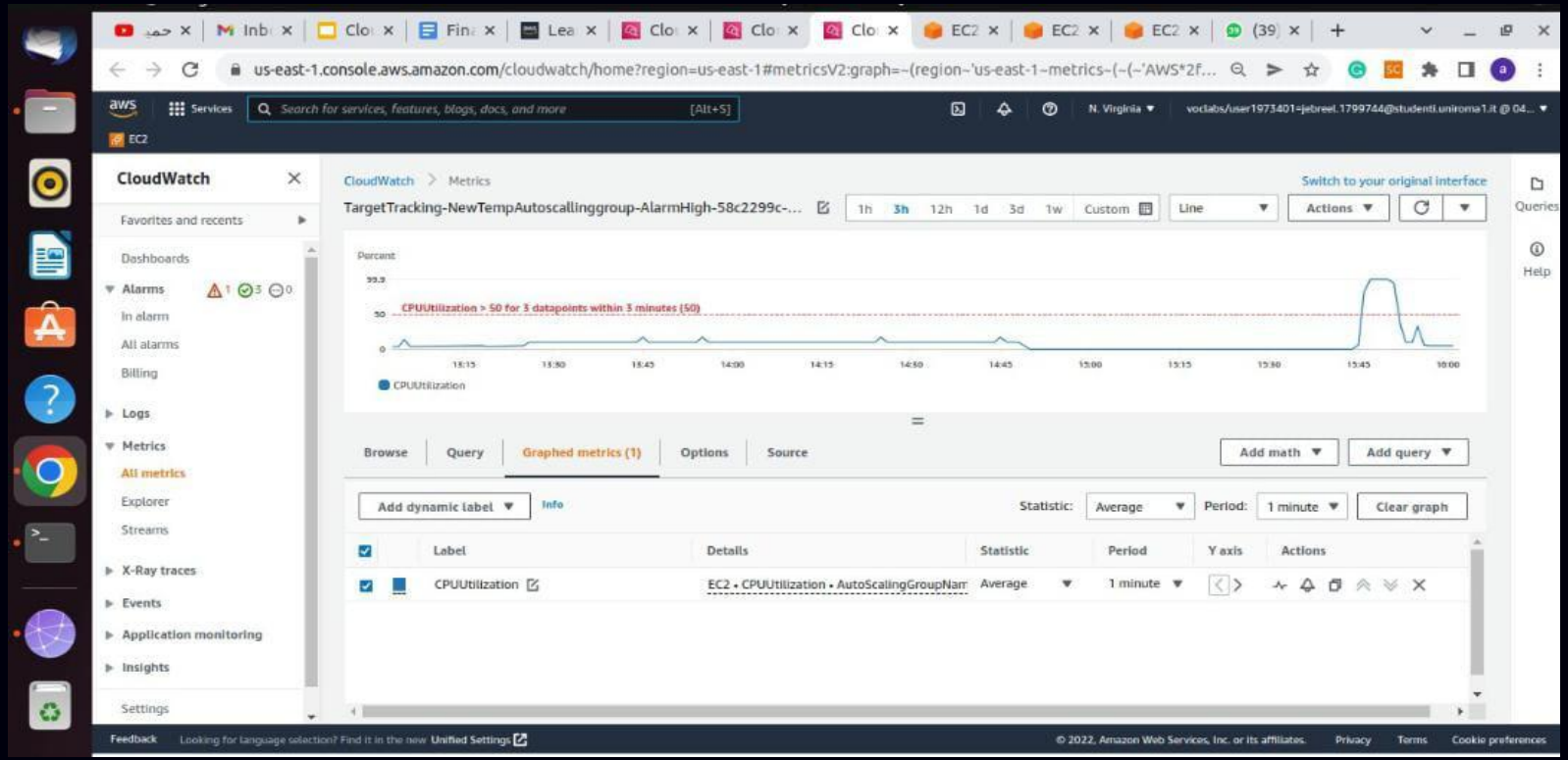
Monitored Metric:
- MetricNamespace: AWS/EC2
- MetricName: CPUUtilization
- Dimensions: [InstanceId = i-02e3b6a726a88dc2a]
- Period: 300 seconds
- Statistic: Average
- Unit: not specified

State Change Actions:
- OK:
- ALARM: [arn:aws:sns:us-east-1:861435167366:Temp-convertor]
- INSUFFICIENT_DATA:

2. Stress Test - Alarm-low

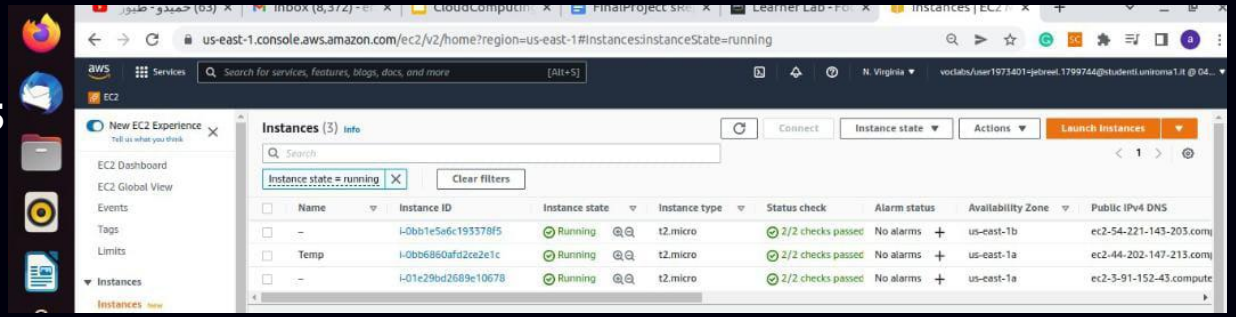


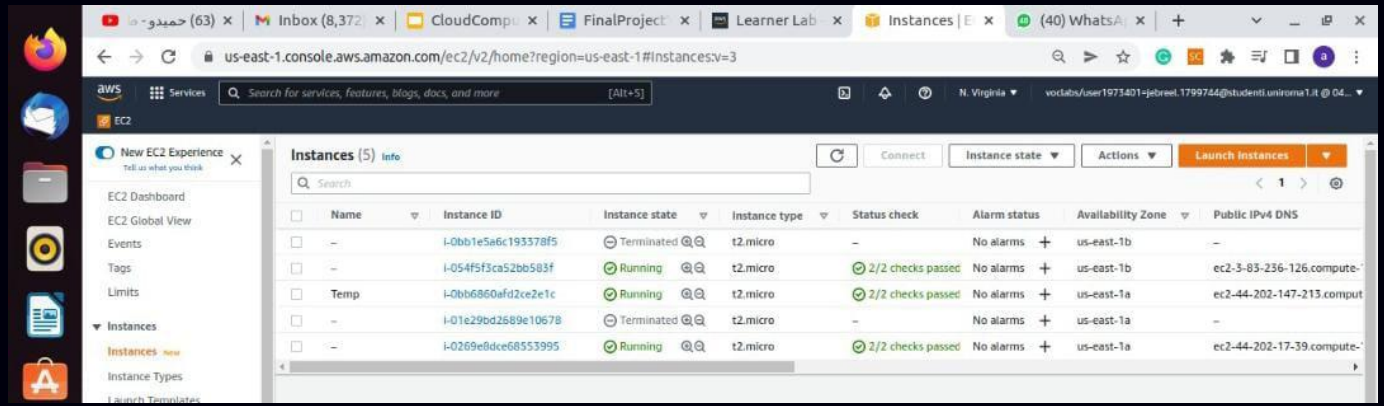
2. Stress Test - Alarm-high



3. Test Auto Scaling

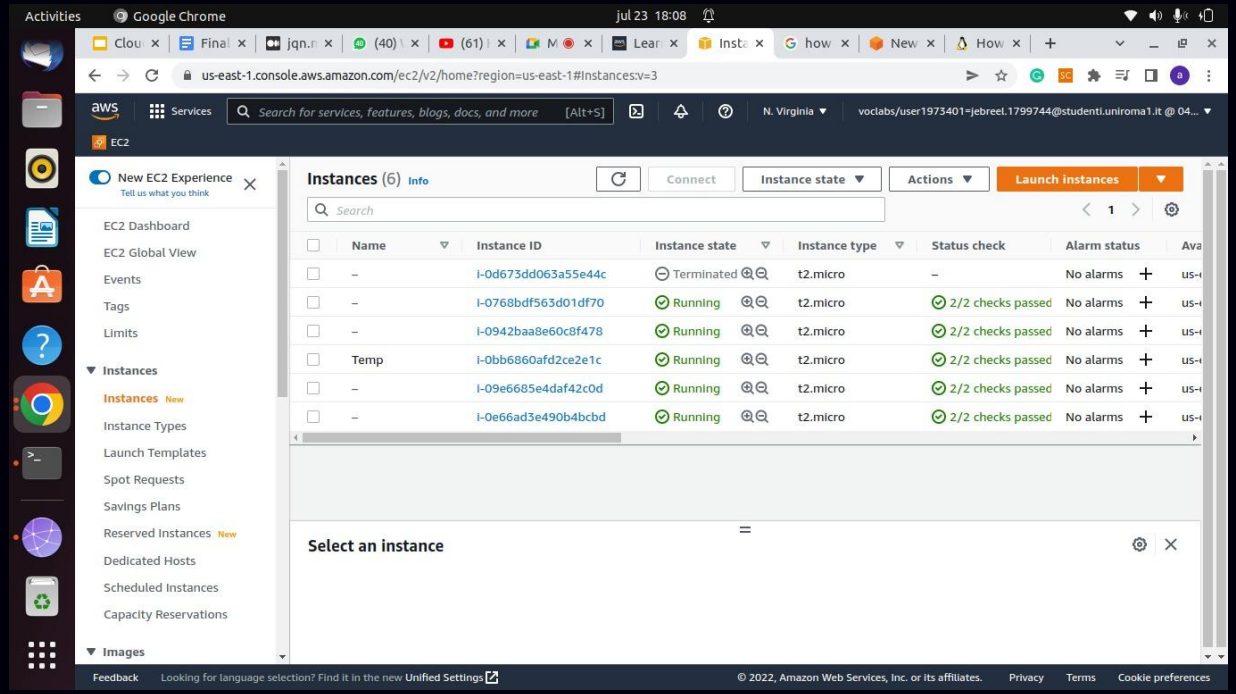
- For checking Auto Scaling, we terminated two of the instances manually and, as you can see in the image below, another instance is created.
- Desired capacity = 2
- Minimum capacity = 2
- Maximum capacity = 5



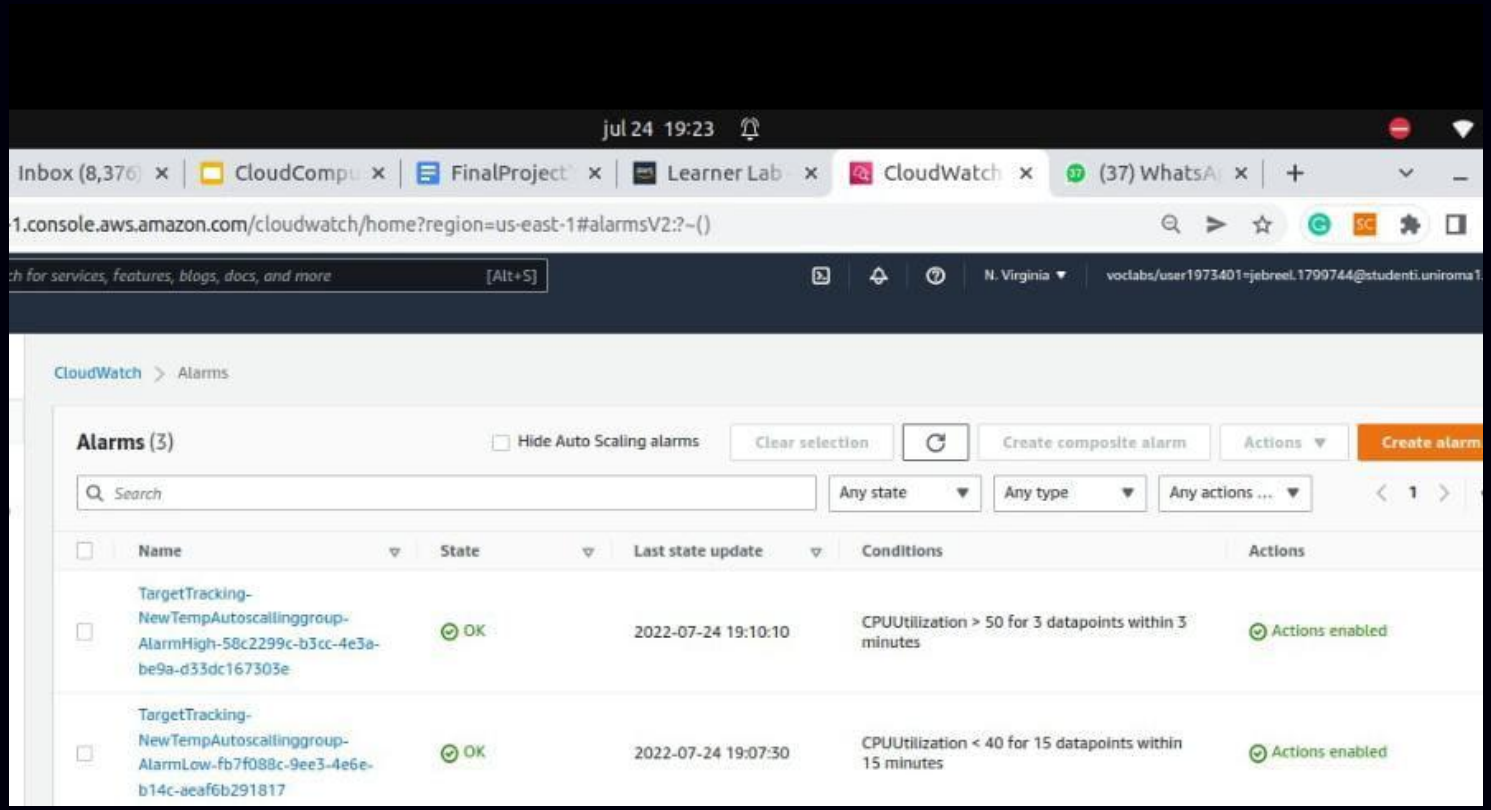


3. Test Auto Scaling with stress applied

- We increased the load on all 3 instances that we had. Since we set an alarm to be triggered when the average cpu utilization is higher than 50% we had an alarm triggered and also three new instances were created as we put that the maximum instance is five

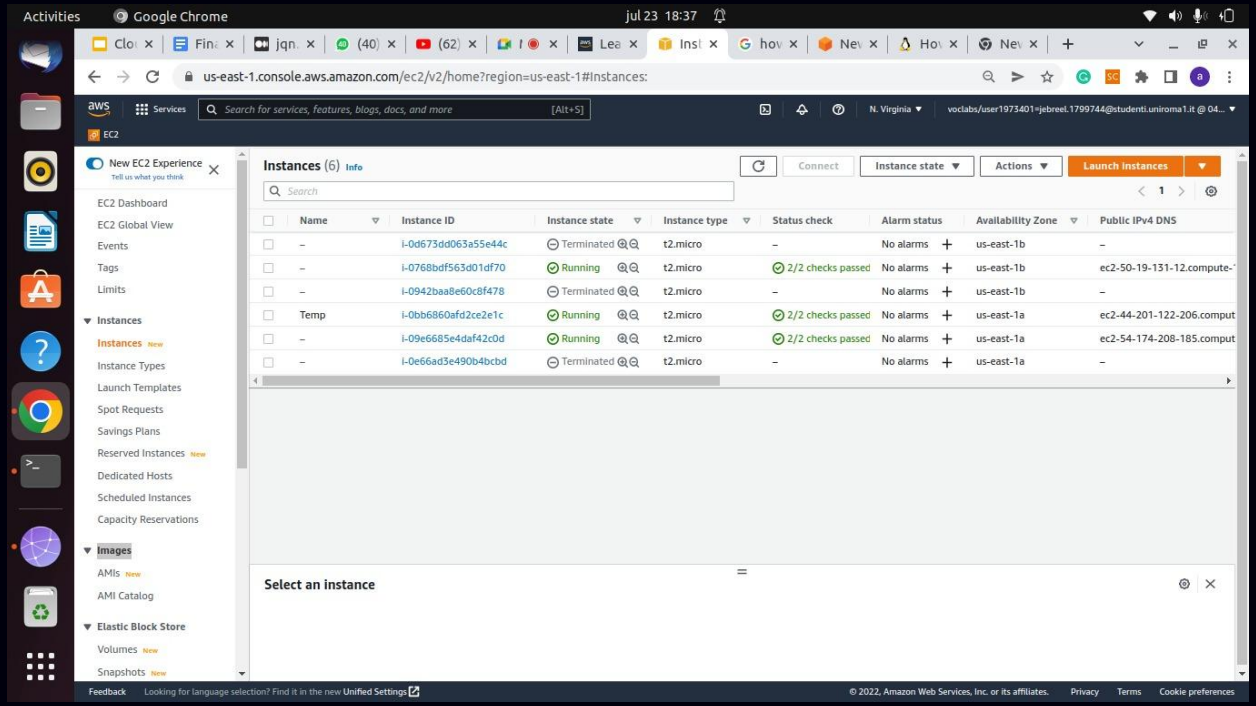


Alarms' status are ok again

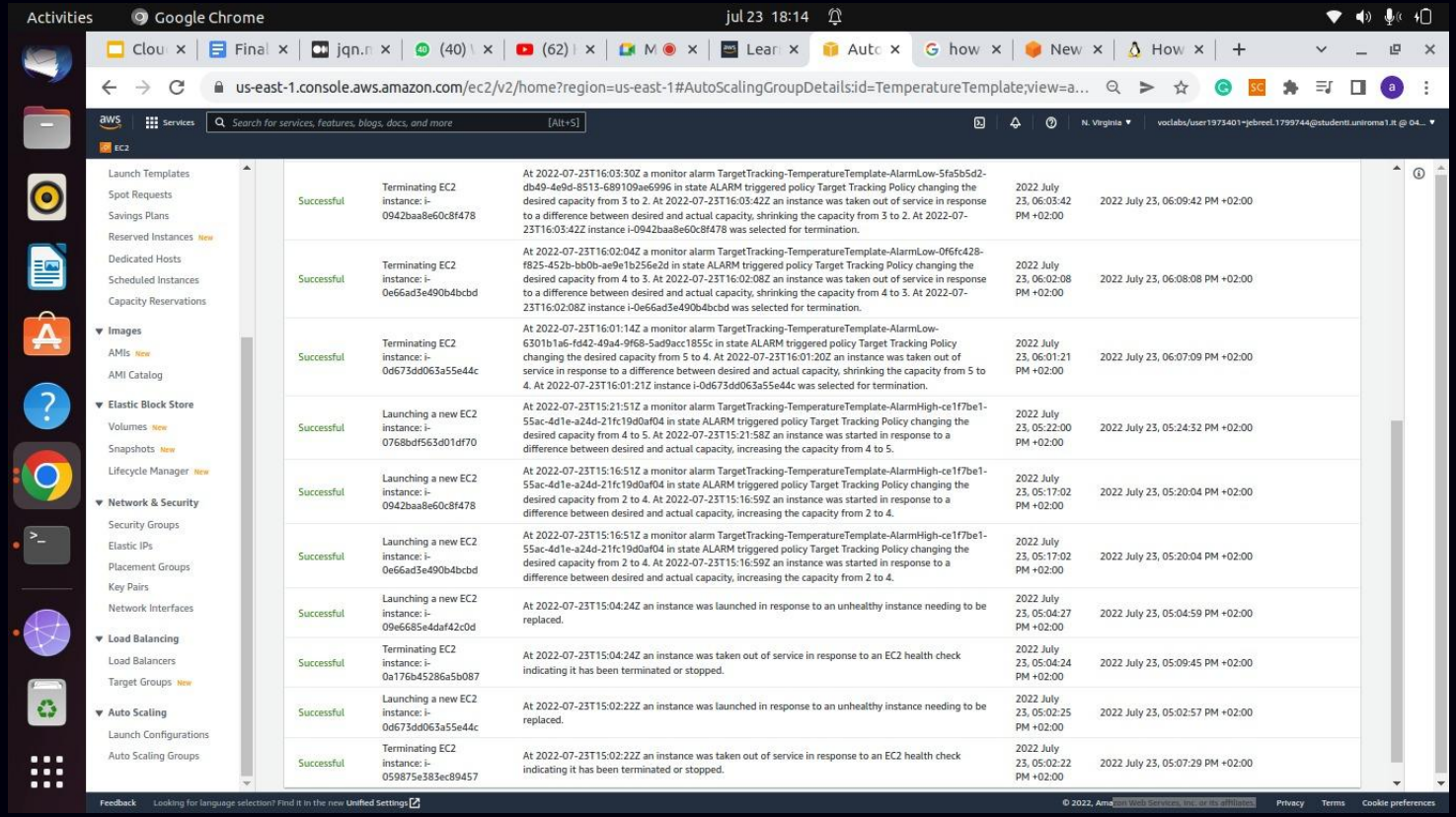


3. Test Auto Scaling after removing the stresses

- When we decreased the cpu utilization again, the instances were terminated since the cpu stress was removed



Summary report



Successful	Terminating EC2 instance: i-0942baa8e60c8f478	At 2022-07-23T16:03:30Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmLow-5fa5b5d2-db49-4e9d-8513-689109ae6996 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 2. At 2022-07-23T16:03:42Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2022-07-23T16:03:42Z instance i-0942baa8e60c8f478 was selected for termination.	2022 July 23, 06:03:42 PM +02:00	2022 July 23, 06:09:42 PM +02:00
Successful	Terminating EC2 instance: i-0e66ad3e490b4bcbd	At 2022-07-23T16:02:04Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmLow-0f6fc428-f825-452b-bb0b-ae9e1b256e2d in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 4 to 3. At 2022-07-23T16:02:08Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 4 to 3. At 2022-07-23T16:02:08Z instance i-0e66ad3e490b4bcbd was selected for termination.	2022 July 23, 06:02:08 PM +02:00	2022 July 23, 06:08:08 PM +02:00
Successful	Terminating EC2 instance: i-0d673dd063a55e44c	At 2022-07-23T16:01:14Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmLow-6301b1a6-fd42-49a4-9f68-5ad9acc1855c in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 5 to 4. At 2022-07-23T16:01:20Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 5 to 4. At 2022-07-23T16:01:21Z instance i-0d673dd063a55e44c was selected for termination.	2022 July 23, 06:01:21 PM +02:00	2022 July 23, 06:07:09 PM +02:00
Successful	Launching a new EC2 instance: i-0768bdf563d01df70	At 2022-07-23T15:21:51Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmHigh-ce1f7be1-55ac-4d1e-a24d-21fc19d0af04 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 4 to 5. At 2022-07-23T15:21:58Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 4 to 5.	2022 July 23, 05:22:00 PM +02:00	2022 July 23, 05:24:32 PM +02:00
Successful	Launching a new EC2 instance: i-0942baa8e60c8f478	At 2022-07-23T15:16:51Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmHigh-ce1f7be1-55ac-4d1e-a24d-21fc19d0af04 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 4. At 2022-07-23T15:16:59Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 4.	2022 July 23, 05:17:02 PM +02:00	2022 July 23, 05:20:04 PM +02:00
Successful	Launching a new EC2 instance: i-0e66ad3e490b4bcbd	At 2022-07-23T15:16:51Z a monitor alarm TargetTracking-TemperatureTemplate-AlarmHigh-ce1f7be1-55ac-4d1e-a24d-21fc19d0af04 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 4. At 2022-07-23T15:16:59Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 4.	2022 July 23, 05:17:02 PM +02:00	2022 July 23, 05:20:04 PM +02:00
Successful	Launching a new EC2 instance: i-09e6685e4daf42c0d	At 2022-07-23T15:04:24Z an instance was launched in response to an unhealthy instance needing to be replaced.	2022 July 23, 05:04:27 PM +02:00	2022 July 23, 05:04:59 PM +02:00
Successful	Terminating EC2 instance: i-0a176b45286a5b087	At 2022-07-23T15:04:24Z an instance was taken out of service in response to an EC2 health check indicating it has been terminated or stopped.	2022 July 23, 05:04:24 PM +02:00	2022 July 23, 05:09:45 PM +02:00
Successful	Launching a new EC2 instance: i-0d673dd063a55e44c	At 2022-07-23T15:02:22Z an instance was launched in response to an unhealthy instance needing to be replaced.	2022 July 23, 05:02:25 PM +02:00	2022 July 23, 05:02:57 PM +02:00
Successful	Terminating EC2 instance: i-059875e383ec89457	At 2022-07-23T15:02:22Z an instance was taken out of service in response to an EC2 health check indicating it has been terminated or stopped.	2022 July 23, 05:02:22 PM +02:00	2022 July 23, 05:07:29 PM +02:00

Thank You