IT00CE11-3005

Cloud Computing

Assignment # 3

September 30, 2024

Resources Autoscaling | STUDENT NO-2402262

Saad Abdullah

saad.abdullah@abo.fi

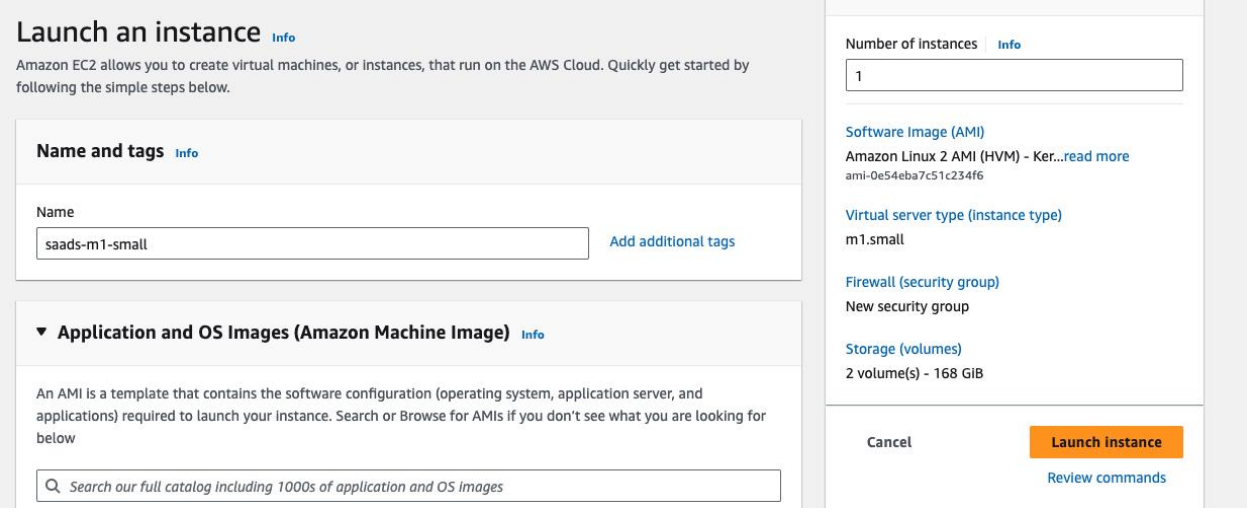
Table of Contents

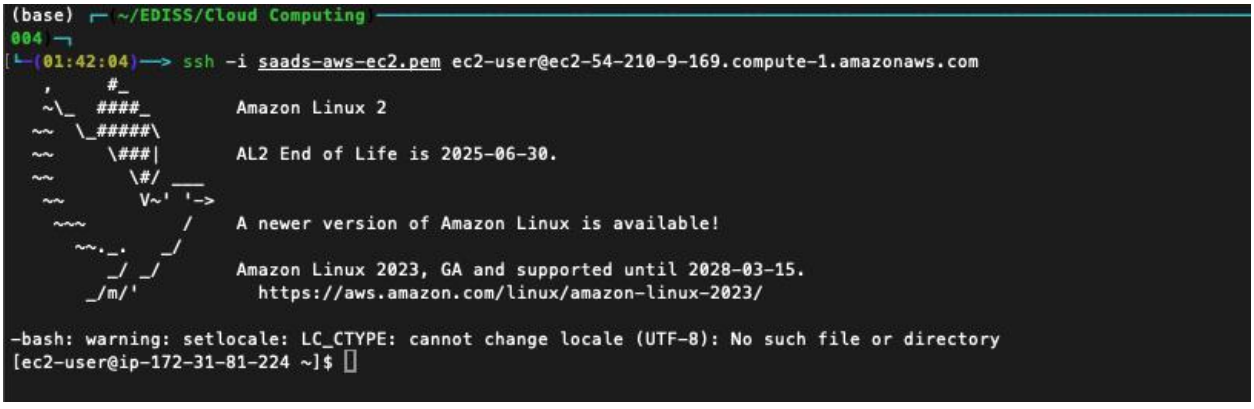
Saad Abdullah
2402262

Elastic Web Services: Resource Auto-scaling

Task 1: Steps for implementation of elastic web services.

To implement my elastic web service, I began by choosing the M1-small instance type. The reason for selecting this instance was its non-burstable nature, which would allow me to observe autoscaling more frequently under load without the burstable performance spikes of other instance types like T2 or T3.



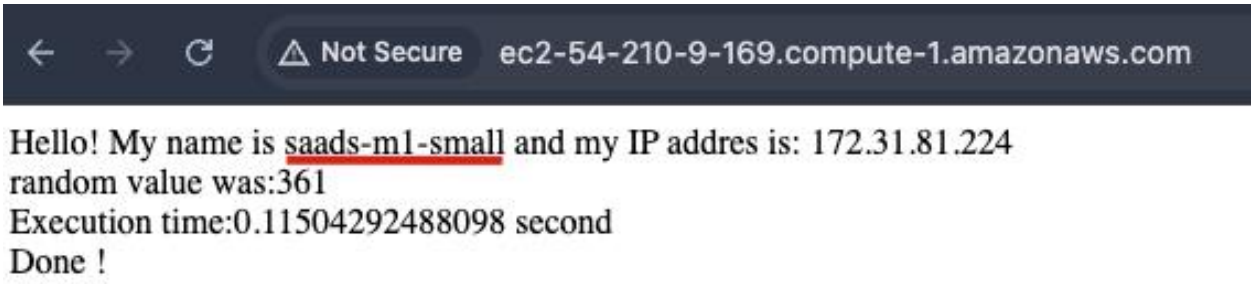


After setting up the instance, I installed Apache and PHP, following the instructions provided in the assignment manual to deploy the `index.php` file.

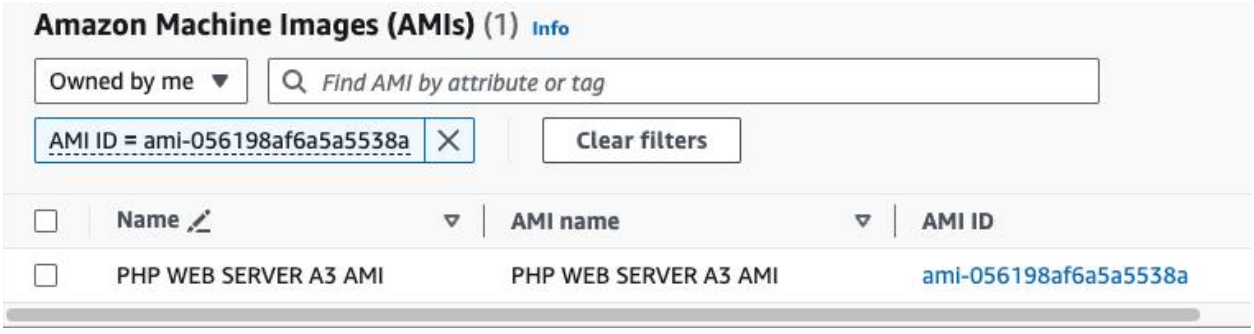
```
[ec2-user@ip-172-31-81-224 ~]$ sudo systemctl enable httpd
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-172-31-81-224 ~]$ sudo systemctl start httpd
[ec2-user@ip-172-31-81-224 ~]$ sudo systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2024-09-29 22:46:25 UTC; 6s ago
     Docs: man:httpd.service(8)
 Main PID: 3642 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─3642 /usr/sbin/httpd -DFOREGROUND
           ├─3643 /usr/sbin/httpd -DFOREGROUND
           ├─3644 /usr/sbin/httpd -DFOREGROUND
           ├─3645 /usr/sbin/httpd -DFOREGROUND
           ├─3656 /usr/sbin/httpd -DFOREGROUND
           └─3657 /usr/sbin/httpd -DFOREGROUND

Sep 29 22:46:25 ip-172-31-81-224.ec2.internal systemd[1]: Starting The Apache HTTP Server...
Sep 29 22:46:25 ip-172-31-81-224.ec2.internal systemd[1]: Started The Apache HTTP Server.
```

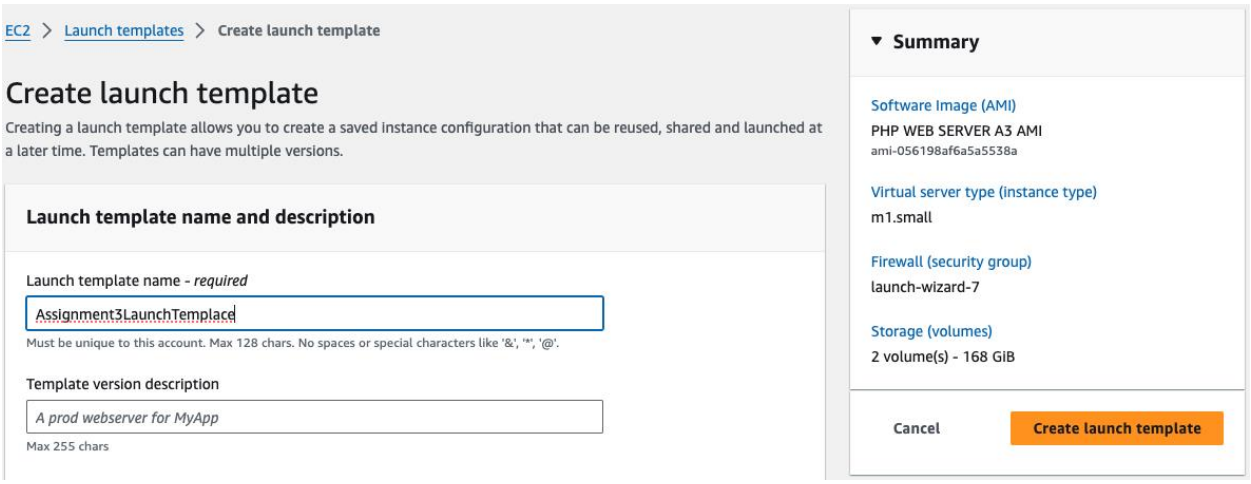
Then I used the public IP of the instance to check if the web server was running.



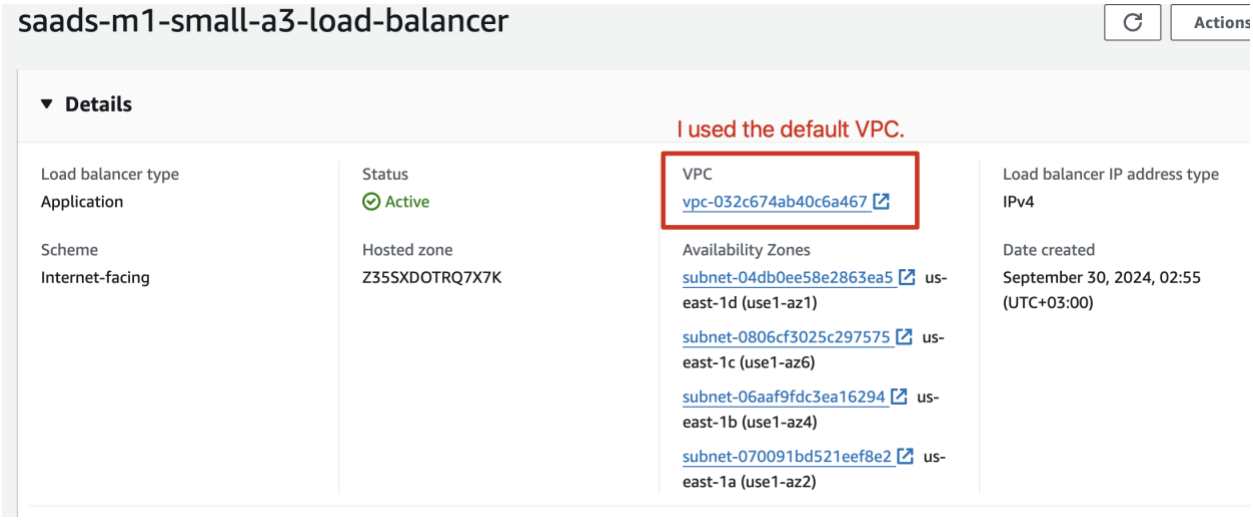
Once the instance was configured and working, I created an Amazon Machine Image (AMI) from the instance. This AMI would allow me to maintain consistent application configuration across all new instances launched by the Auto Scaling Group.



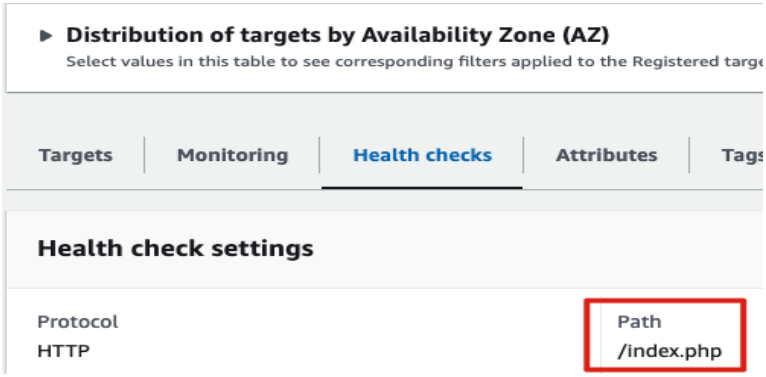
I then created a launch template (instead of a launch configuration) using the AMI. The launch template was needed to define specific instance settings such as the instance type (M1-small), security groups, key pairs for SSH access, and network configurations.



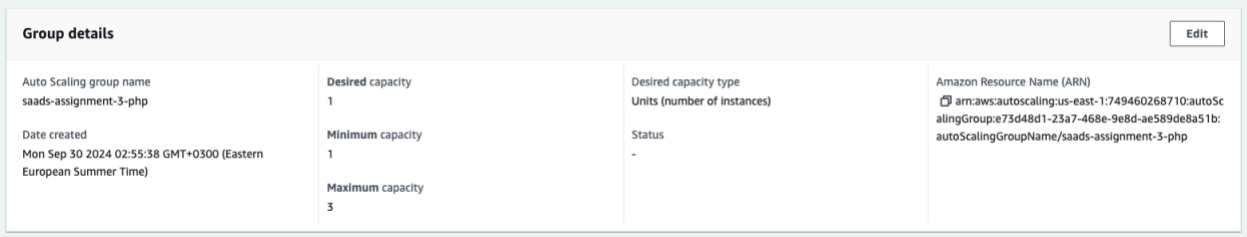
Then I begin by creating an autoscaling group, before finalizing the autoscaling group I need to create the load balancer. I set up an Application Load Balancer (ALB) to distribute the incoming traffic across the instances. The ALB is crucial for ensuring that traffic is balanced evenly across multiple instances, especially when the number of instances scales up or down. I created the ALB by defining the listener on port 80 (HTTP) and configuring health checks to monitor the instances' availability.

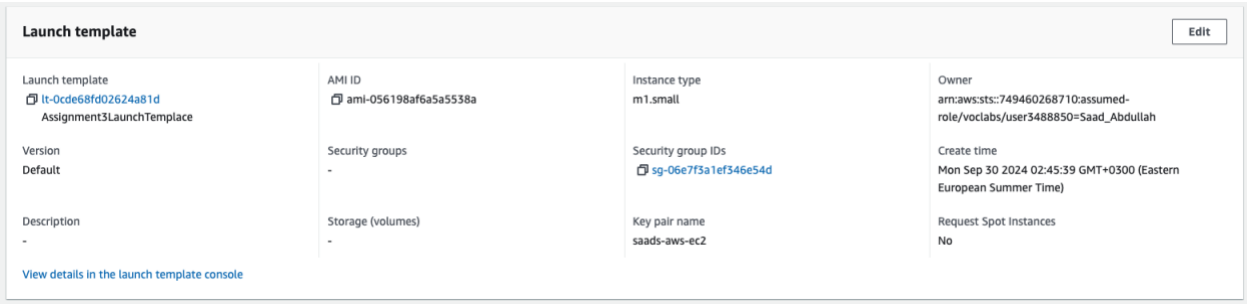


I turned on the health check in load balancer and pointed it to index.php

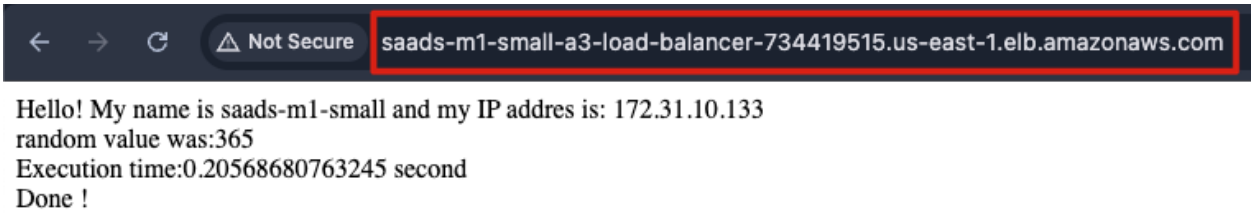


Once the load balancer was set up and ready, I proceeded to create the Auto Scaling Group (ASG). In this step, I attached the ALB to the ASG, ensuring that any instances created by the scaling group would automatically be added to the load balancer's target group.





So, now everything was set up to perform the experiments. For further checking, I entered the public IP of the load balancer to check if the PHP app is running.



Task 2: Used httperf commands to create load on the instances.

Below are the commands I used to create the load on the instances, I will add a summary after each command to show what results I got from the command.

```
1) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --uri=/index.php --wsess=600,5,2 --rate 1 --timeout 5
```

Generated a total of 600 connections with a request rate of 1.8 req/s. The average connection time was 12857.5 ms, and the system recorded 534 client-timeout errors. The reply rate averaged 0.9 replies/s.

```
2) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --uri=/index.php --wsess=800,5,2 --rate 2 --timeout 5
```

This command resulted in a segmentation fault. After researching, the error was identified to be related to the file descriptor limits of httperf. Tried multiple times, but -- rate 2 was causing the trouble.

Then I Executed 3 commands in parallel from my local console and the following were the results:

```
3a) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --
uri=/index.php --wsess=800,5,2 --rate 1 --timeout 5
```

Generated a total of 800 connections with a request rate of 3.5 req/s. The average connection time was 14064.5 ms, with 408 client-timeout errors. The reply rate averaged 3.0 replies/s.

```
3b) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --
uri=/index.php --wsess=1500,5,2 --rate 1 --timeout 5
```

Generated a total of 1500 connections with a request rate of 4.4 req/s. The average connection time was 12709.6 ms, and the system recorded 312 client-timeout errors. The reply rate averaged 4.2 replies/s.

```
3c) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --
uri=/index.php --wsess=200,5,2 --rate 2 --timeout 5
```

Generated 200 connections at a rate of 2.8 req/s. The test duration was 107.407 s with an average connection time of 13188.6 ms and 200 client-timeout errors.

After these, I ran a final test with a lot of requests and below are the results:

```
4) httperf --server saads-m1-small-a3-load-balancer-734419515.us-east-1.elb.amazonaws.com --
uri=/index.php --wsess=2500,5,2 --rate 1 --timeout 5
```

This command created 2500 connections with a request rate of 4.6 req/s. The test recorded 338 client-timeout errors, with an average connection time of 12202.7 ms and an average reply rate of 4.5 replies/s.

Task 3: Screenshots and Video Recordings for Auto-Scaling.

Following are the descriptions with the corresponding URLs of some videos:

1) Real-Time Scaling Up of ASG:

This video shows a real-time demonstration of the auto-scaling group scaling up as the CPU utilization increases.

[OneDrive Link: Real-Time Scaling Up]

2) Upscaling During Old Experiment:

This video captures the scaling up of instances during a previous experiment where CPU utilization increased, triggering instance creation.

[OneDrive Link: Upscaling in Old Experiment]

3) Downscaling During Old Experiment:

This video shows the downscaling of instances as the CPU utilization decreases in old experiment, causing instance termination.

[OneDrive Link: Downscaling in Old Experiment]

4) Snapshot of an activity of scaling up:

Below is the snapshot of the activity of scaling up I performed on the 30th of September 2024.

Successful	Launching a new EC2 instance: i-0f480676575ce8e90	At 2024-09-30T08:50:15Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmHigh-f37d9fda-ee09-485d-991a-7030f57cd928 in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 2 to 3. At 2024-09-30T08:50:22Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2024 September 30, 11:50:24 AM +03:00	2024 September 30, 11:55:30 AM +03:00
Successful	Launching a new EC2 instance: i-01f9b6e27dc968644	At 2024-09-30T08:44:15Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmHigh-f37d9fda-ee09-485d-991a-7030f57cd928 in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 1 to 2. At 2024-09-30T08:44:21Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2024 September 30, 11:44:23 AM +03:00	2024 September 30, 11:49:29 AM +03:00

5) Snapshot of an activity of scaling down:

Below is the snapshot of the activity of scaling down I performed on the 30th of September 2024.

Successful	Terminating EC2 instance: i-01f9b6e27dc968644	At 2024-09-30T09:07:58Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmLow-953e55c7-bc9b-4556-bfcd-5edcf897e77b in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 2 to 1. At 2024-09-30T09:08:06Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2024-09-30T09:08:06Z instance i-01f9b6e27dc968644 was selected for termination.	2024 September 30, 12:08:06 PM +03:00	2024 September 30, 12:13:52 PM +03:00
Successful	Terminating EC2 instance: i-004d8133e39307cbf	At 2024-09-30T09:06:58Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmLow-953e55c7-bc9b-4556-bfcd-5edcf897e77b in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 3 to 2. At 2024-09-30T09:07:07Z 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 2024-09-30T09:07:07Z instance i-004d8133e39307cbf was selected for termination.	2024 September 30, 12:07:07 PM +03:00	2024 September 30, 12:13:39 PM +03:00

6) Snapshot of an activity of scaling up and down on a new experiment:

Below is the snapshot of the activity of scaling up and down I performed on the 30th of September 2024. This one is a different experiment from above.

Successful	Terminating EC2 instance: i-0a9dc836a43807ad3	At 2024-09-30T11:32:58Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmLow-953e55c7-bc9b-4556-bfcd-5edcf897e77b in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 2 to 1. At 2024-09-30T11:33:06Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2024-09-30T11:33:06Z instance i-0a9dc836a43807ad3 was selected for termination.	2024 September 30, 02:33:06 PM +03:00	2024 September 30, 02:39:18 PM +03:00
Successful	Terminating EC2 instance: i-0f480676575ce8e90	At 2024-09-30T11:31:58Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmLow-953e55c7-bc9b-4556-bfcd-5edcf897e77b in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 3 to 2. At 2024-09-30T11:32:07Z 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 2024-09-30T11:32:07Z instance i-0f480676575ce8e90 was selected for termination.	2024 September 30, 02:32:07 PM +03:00	2024 September 30, 02:38:11 PM +03:00
Successful	Launching a new EC2 instance: i-0448e41020ab66e9e	At 2024-09-30T11:15:15Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmHigh-f37d9fda-ee09-485d-991a-7030f57cd928 in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 2 to 3. At 2024-09-30T11:15:20Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2024 September 30, 02:15:22 PM +03:00	2024 September 30, 02:20:28 PM +03:00
Successful	Launching a new EC2 instance: i-0a9dc836a43807ad3	At 2024-09-30T11:09:15Z a monitor alarm TargetTracking-saads-assignment-3-php-AlarmHigh-f37d9fda-ee09-485d-991a-7030f57cd928 in state ALARM triggered policy saads-m1-small-a3-scaling-policy changing the desired capacity from 1 to 2. At 2024-09-30T11:09:18Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2024 September 30, 02:09:20 PM +03:00	2024 September 30, 02:14:26 PM +03:00

Task 4: Analysis of the obtained behavior of web services:

In this analysis, I will discuss the behavior and reactivity of the deployed PHP web service in response to load variations, as observed during the experiment with AWS EC2's auto-scaling mechanism. The M1.small instances used are not burstable, meaning their performance remains consistent without temporary boosts, which makes the scaling behavior more predictable.

Behavior During Load Increases:

When testing with httperf to simulate increasing user requests, the service showed a clear pattern of rising CPU utilization. The PHP web app, which performs random computations and matrix multiplications, generates significant CPU activity. As CPU utilization passed the 70% threshold, the auto-scaling group responded by launching additional EC2 instances.

Since the M1.small provides a fixed level of CPU capacity. When this capacity was exceeded due to the load, the scaling up ensured that more instances with the same configuration were created, distributing the workload and preventing any single instance from being overwhelmed. The response time for launching new instances varied slightly depending on the load conditions, but on average, new instances were added within a few minutes (around 5 approx) of the threshold being crossed, maintaining system performance despite the increased load.

The results from the load tests show the following:

- **Initial Load Test:** A command with 600 connections (1.8 req/s) saw an average connection time of 12857.5 ms, along with 534 client-timeout errors.
- **Higher Load:** A test with 1500 connections resulted in an average connection time of 12709.6 ms and 312 client timeouts, showing the system's strain under increasing requests.

This kind of behavior is ideal for services where consistent performance is critical. Applications such **as e-commerce platforms** (during seasonal traffic surges) and **streaming platforms** (during peak hours) would benefit from this predictable scaling mechanism, ensuring smooth performance without sudden performance degradation.

Behavior During Load Decreases:

Once the load started to decrease, the auto-scaling group reacted by terminating the now-redundant instances, ensuring that resources were only being used when needed. This controlled downscaling ensured that there were no disruptions to ongoing requests, as instances were released gradually. On average, it took around 5-10 minutes for ASG to terminate unused instances.

During one experiment, I sent around 2500 requests using httperf, which ended up overwhelming instances, likely because they ran out of memory. Since my AWS student account has limits, the auto-scaling group couldn't add more than three instances to manage the extra load. Despite that, the system handled the situation well by quickly replacing the unhealthy instances with new, healthy ones. However, even after stopping the load test, it took a bit of time for the system to fully recover, as several instances remained unhealthy for a while. This shows how important it is to have a robust scaling mechanism when dealing with high traffic. Applications that would benefit from such a scaling strategy may include:

- **Microservices architecture-based applications:** Auto-scaling helps microservices that experience varying loads to handle each service separately. For example, a product search service in an e-

commerce platform might experience more traffic during specific times, and scaling up just this service ensures better efficiency.

- **Data processing applications:** These services often have varying computational needs, they can rely on predictable scaling patterns like mine to increase and decrease processing power as data loads change.

In conclusion, the auto-scaling mechanism in this experiment responded effectively to load variations. The non-burstable nature of the M1.small instance allowed for predictable scaling behavior, ensuring consistent performance during increases in load and efficient cost management during decreases.

Reflection:

Have you learned anything completely new?

Yes, I never made any autoscaling group so it was new for me.

Did anything surprise you?

No.

Did you find anything challenging? Why?

The video-making took some time as results in AWS console didn't appear abruptly.

Did you find anything satisfying? Why?

Completing the assignment was satisfying ☺

---------- THE END ----------