Transform the Retail Industry by Leveraging Cloud Technology

1 Introduction

The retail industry is the most dynamic sector with high competition among the competitors. Cloud technology is the best for it as it offers many advantages through which this industry can enhance its operational efficiency, improve customer experience, and innovate at a quick speed. Despite its advantages, this technology has some disadvantages that need to be overcome. This essay explores the transformation that cloud technology brings to the retail industry and examines the key features and services on AWS that make this transformation possible. Additionally, the essay discusses some potential problems of cloud technology. In summary, cloud technology is the most beneficial technology in the modern marketplace, especially for the retail industry, but some challenges need to be resolved [1].

2 Advantages of Cloud Technology in Retail

2.1 Scalability and Elasticity

One of its major benefits – the thing that makes cloud computing appealing to retailers – is elasticity. Cloud computing resources can easily be scaled up or down. Retailers often face elastic demand where it changes in reaction to evolving consumer demand, such as during the holidays or when a retailer wants to drive website traffic for a big sale. Autoscaling with AWS Auto Scaling and Elastic Load Balancing mean that a retailer can scale up or down their computing resources to correspond to the demand on their site so that it can perform well without overprovisioning for hardware and software resources when it isn't needed. This not only builds a better customer experience because the website is fast and responsive, but it optimises costs by scaling down when there's less demand [2].

2.2 Data Analytics and Insights

By using Amazon Redshift and AWS Glue – Amazon Web Services that can extract, load, and transform data – retailers can gather and store big data to analyse transactions, manage inventory, and identify sales trends, for instance. Analysing customers' purchase histories can help in customising marketing campaigns, and analysing inventory data can allow for rationalising stock levels, minimising both overstocking and stockouts. With a massive amount of data, retailers can enhance all their operations [3].

2.3 Enhanced Security

With their customers' data often at the core of the business, security is of utmost importance for retailers. This is why it's great that AWS has a highly secure infrastructure with things like AWS Identity and Access Management (IAM), AWS Key Management Service (KMS), and AWS Shield: services that enable the retailer to secure its data against malicious external access, to be compliant against existing and upcoming data protection regulations, and to be protected against cyber attacks; enabling the retailer to implement granular access controls and encrypt data both at rest and in motion [4].

2.4 Cost Efficiency

Compared with traditional IT infrastructure that demands massive capital expenditure for hardware, software, and maintenance of those, the cloud military dock changes the model for computing into the operational expenditure model. Retailers only pay for what they consume. To meet the various demands from the retailers, AWS provides three kinds of pricing models: on-demand, reserved instances, and spot instances. Ambiguity Bot All these help the retailers find the right price for their needs. With the cloud military dock, small retailers can even afford a good architecture of computing which otherwise requires a big investment at the very first place [5].

2.5 Innovation and Agility

The cloud offers organisations room for fast innovation and development. Through AWS services such as AWS Lambda, Amazon S3, and Amazon EC2, retailers can rapidly develop, test, and deploy new applications, as well as innovate and quickly meet the evolving customer expectations. All this can be particularly useful in the fast retail environment where there is ever-increasing pressure to continuously innovate, scale, and bring new customer features and services to market faster than competitors. There are many ways cloud-based development environments and DevOps practices facilitate the dynamic nature of retail environments [6].

3 Challenges of Cloud Technology in Retail

3.1 Data Privacy and Compliance

Data privacy and compliance are often seen as the most challenging areas. AWS provides state-of-the-art security services, but ultimately each retailer has to make sure that its use of cloud services keeps it compliant with the Data Protection Act 2018 (DPA), or the GDPR, or the California Consumer Privacy Act (CCPA), or the PCI-DSS (Payment Card Industry Data Security Standard) requirements [7]. Ensuring compliance in their use of cloud services can be a real challenge where one needs extensive knowledge of the various requirements, regulations, and related standards and a good working relationship with the cloud providers to ensure they adhere to whatever regulatory framework the retailer is required to comply with.

3.2 Downtime and Reliability

Even though cloud services are highly reliable, outages do occur. If say AWS has a service outage, Amazon's retail applications could be unavailable for a period of time, and the retailer could potentially miss revenue and orders during that time. A strong disaster recovery and business continuity plan is critical, involving multi-region deployments and backups [8].

3.3 Integration with Legacy Systems

Many retailers also still operate on legacy systems for their internal operations. The challenge (and time and money) in integrating with new modern cloud-based services poses a huge obstacle. Indeed, the majority of adopting new technologies requires deep reengineering of systems and processes to migrate to the cloud. Retailers have to plan their migration strategy, such as by understanding how services can be used from, say, AWS Database Migration Service (DMS) to migrate legacy databases to the cloud all while maintaining legislative and statutory requirements and the ongoing operations [9].

4 Conclusion

First, it provides the power and potential for scalability, cost efficiency, enhanced security, and innovation. Second, it provides the ability for every retailer to make their business quicker, easier, and more automated which in turn provides their retail store customers with a better experience by giving their retail store staff the time they need to satisfy consumers. Among the various capabilities that AWS provides such as Auto Scaling, Elastic Load Balancing, Amazon Redshift, AWS IAM, and AWS Lambda, there are various challenges that are slowing down cloud adoption. The main challenges are regarding data privacy, downtime, and legacy system integration. By resolving the challenges related to cloud adoption and exploring the specific capabilities that AWS and cloud technology provide, retailers are positioned to best exploit the advantages associated with the tremendous scalability of the cloud.

References

[1] Technologies, A. (2020). How Cloud Computing is Transforming Retail. Retrieved from https://anywhere.tech/cloud-services/cloud-computing-in-retail/

[2] Services, A. W. (n.d.). AWS Academy Cloud Foundations. Retrieved from AWS Academy: https://awsacademy.instructure.com/courses/56499

[3] Unacast. (2024, May 23). Retail Business Intelligence: Transforming Data into Strategic Insights. Retrieved from https://www.unacast.com/post/retail-business-intelligence

[4] Wright, V. (2024, June 6). Automated Data Security and Compliance for the Retail Industry. Retrieved from BigID: https://bigid.com/blog/automated-data-security-and-compliance-for-the-retail-industry/

[5] Services, A. W. (n.d.). AWS Academy Cloud Foundations. Retrieved from AWS Academy: https://awsacademy.instructure.com/courses/56499

[6] Services, A. W. (n.d.). AWS Academy Cloud Foundations. Retrieved from AWS Academy: https://awsacademy.instructure.com/courses/56499

[7] AWS. (n.d.). General Data Protection Regulation (GDPR) Center. Retrieved from AWS Cloud Security: https://aws.amazon.com/compliance/gdpr-center/

[8] Tunggal, A. T. (2021, August 25). The Cost of Downtime At The World's Biggest Online Retailer. Retrieved from UpGuard: https://www.upguard.com/blog/the-cost-of-downtime-at-the-worlds-biggest-online-retailer

[9] Zahir Irani, R. M. (2024). The impact of legacy systems on digital transformation in European public administration: Lesson learned from a multi-case analysis. Government Information Quarterly.

Benefits and Pitfalls of the Initial WordPress Architecture (TASK: A)

The initial set-up of single EC2 instance with WordPress application provides a clear and inexpensive approach to get a WordPress site running, in a simple way. Nevertheless, this architecture has clear limitations that should be addressed in order to make it a resilient, scalable and high performing application which can be taken into a production environment. Below are the advantages and disadvantages of this initial architecture.

Benefits of the Initial WordPress Architecture

Simplicity and Ease of Setup:

It's easy to initially set up WordPress using a pre-built AWS community AMI (an Amazon Machine Image) as it takes advantage of the packaged WordPress stack provided by Bitnami. This stack combines all of the underlying components for hosting a WordPress site into one deployment, so it readily comes pre-packaged and configured.

The provided instructions are granular, ensuring that users who are relatively new to AWS services can still complete the deployment of the application.

Cost-Effective:

Running on one EC2 instance – ideally a t2.micro instance – will keep the cost of the WordPress site as low as possible, at least until the site is more firmly established, or at an early stage of discovery or testing when you're not willing or able to spend much.

AWS Free Tier eligible resources reduce costs in deployments (aws, 2024).

Immediate Access and Functionality:

In the first branch, you get immediate access to the WordPress admin area to begin creating content and customising the site.

A pre-set plugin for load testing (AF WP StressKit) makes performance testing of the plugin ready to go right 'out of the box'.

Pitfalls of the Initial WordPress Architecture

Lack of Scalability:

An EC2 instance is not suited to handling high loads or spikes in traffic. When there is too much traffic, performance of the WordPress site drops in terms of page load rates and could even go down (Amazon EC2 instance become slow everyday, 2024).

Horizontal scaling – ie, adding more instances – is not factored in from the outset, nor does the system automatically grow vertically to meet demand (ie, it doesn't automatically increase instance size).

Single Point of Failure:

Depending on WordPress running alone on a single EC2 instance for example raises the hazard of downtime if the instance itself fails; any overnight failure without a redundancy or failover solution will cause a site to go down.

For example, having to restart the server to apply updates or for maintenance will inevitably result in downtime (that is, the site won't be available).

Security Concerns:

Opening the EC2 instance to the internet (with ports containing HTTP and SSH) means that your system is at an elevated risk for being compromised. Granted, the setup notes tell us to limit SSH access to specific IP addresses – but the setup is still vulnerable (Is starting an AWS instance with only ssh to port 22 significantly insecure?, 2024).

Performance Limitations:

In other words, it is easy to start the life of your site's hosting by having all your data and everything work with one single t2.micro instance, but as the amount of data grows, or your traffic grows, it might become harder to keep your instance running at reasonable speed.

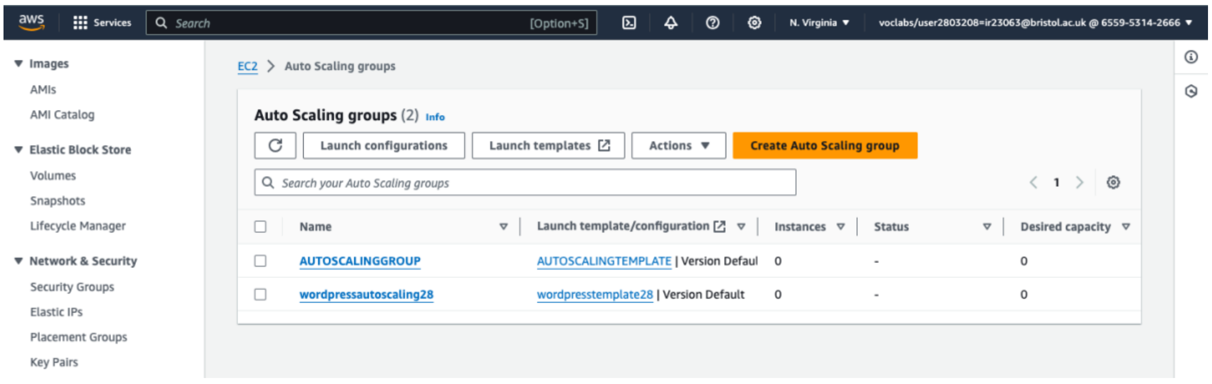
Conclusion

This simple, inexpensive, easy-to-implement architecture is appropriate for low-load or test environments where scalability and redundancy of the WordPress site are not critical, although security and performance may still be an issue for a production environment.

Design process to architect the scaling behaviours (Task B)

Autoscaling Groups:-

I made two autoscaling group. Capital letter autoscaling group **(Name:- AUTOSCALINGGROUP)** is made by using AMI as per the instructions of coursework. Second one(Name:- wordpressautoscaling28) is just my experiment. Both give very good results.



Auto Scaling groups (2) Info

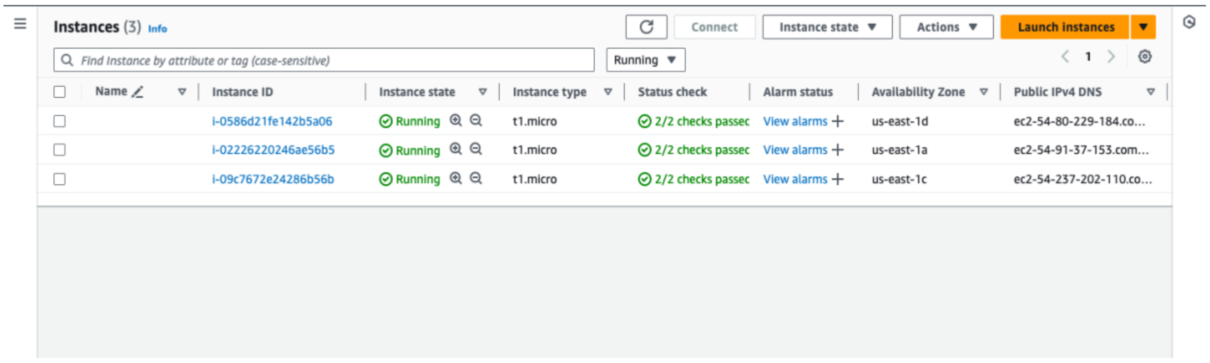
Name	Launch template/configuration	Instances	Status	Desired capacity
AUTOSCALINGGROUP	AUTOSCALINGTEMPLATE | Version Defaul	0	-	0
wordpressautoscaling28	wordpresstemplate28 | Version Default	0	-	0

Figure 1AutoScaling Groups

First autoscaling group Configuration:-

1. Name:- AUTOSCALINGGROUP
2. Launch template:- AUTOSCALINGTEMPLATE
3. AMI ID:- ami-0e802884e420243c7
4. Security group IDs:- sg-0b89387287a346005
5. Key pair name:- wordpress-keypair
6. Availability Zones:- us-east-1a, us-east-1b, us-east-1c, us-east-1d
7. Load balancer target groups:- wordpresstargetgroup28
8. Health check grace period:- 300
9. Termination policies:- NewestInstance (Help us to retain oldest instance and terminate newest instance due to which oldest instance always connect with terminal or SSH).
10. Suspended processes:- ReplaceUnhealthy(This replace the unhealthy instances).
11. Default cooldown:- 5 (Gap between scaling activity)
12. Default instance warmup:- 30 seconds

Running Instances of autoscaling groups:-

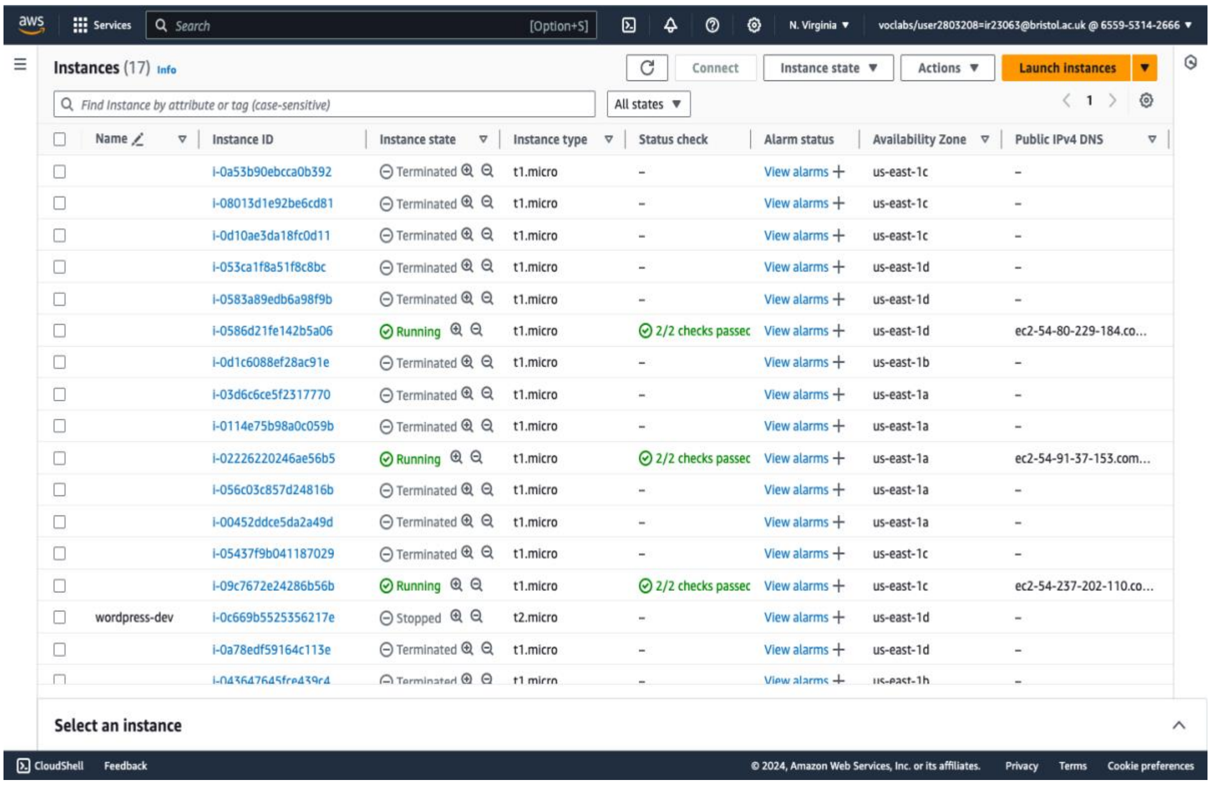

Instances (3) Info

Find Instance by attribute or tag (case-sensitive) | Running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
	i-0586d21fe142b5a06	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1d	ec2-54-80-229-184.co...
	i-02226220246ae56b5	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-54-91-37-153.com...
	i-09c7672e24286b56b	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1c	ec2-54-237-202-110.co...

Figure 2Autoscaling Running Instances

All instances of autoscaling group:-



Instances (17) Info

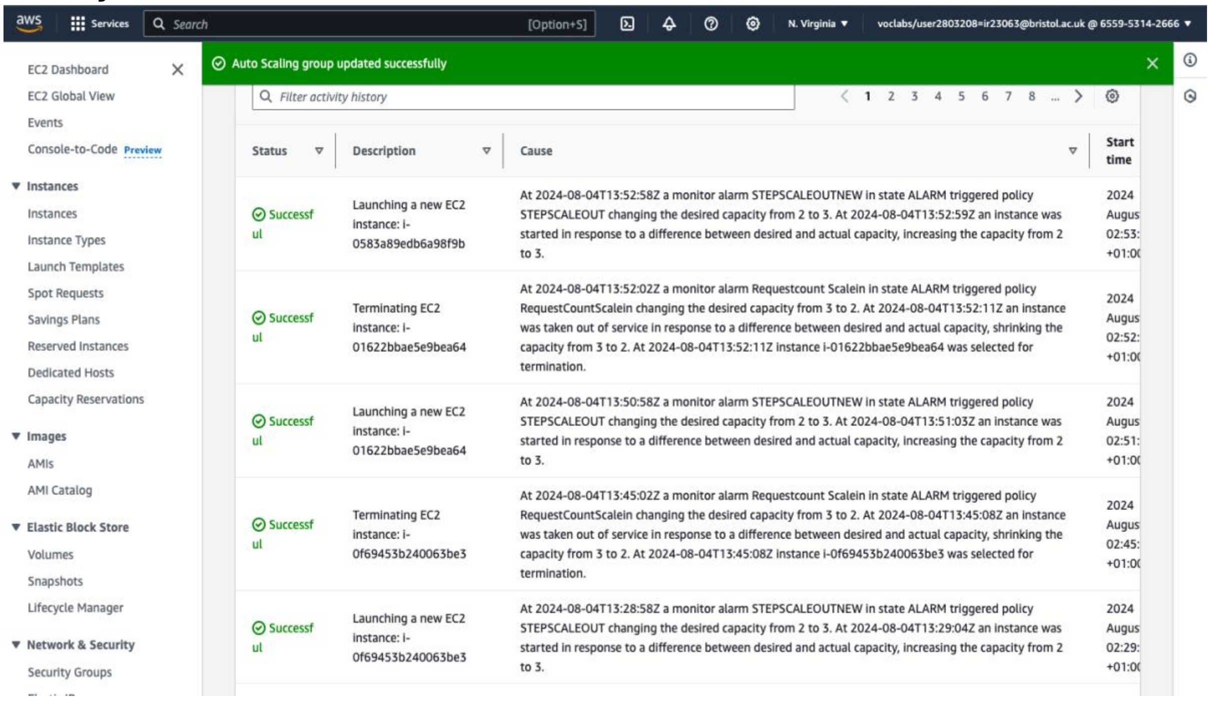
Find Instance by attribute or tag (case-sensitive) | All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
	i-0a53b90ebcca0b392	Terminated	t1.micro	–	View alarms +	us-east-1c	–
	i-08013d1e92be6cd81	Terminated	t1.micro	–	View alarms +	us-east-1c	–
	i-0d10ae3da18fc0d11	Terminated	t1.micro	–	View alarms +	us-east-1c	–
	i-053ca1f8a51f8c8bc	Terminated	t1.micro	–	View alarms +	us-east-1d	–
	i-0583a89edb6a98f9b	Terminated	t1.micro	–	View alarms +	us-east-1d	–
	i-0586d21fe142b5a06	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1d	ec2-54-80-229-184.co...
	i-0d1c6088ef28ac91e	Terminated	t1.micro	–	View alarms +	us-east-1b	–
	i-03d6c6ce5f2317770	Terminated	t1.micro	–	View alarms +	us-east-1a	–
	i-0114e75b98a0c059b	Terminated	t1.micro	–	View alarms +	us-east-1a	–
	i-02226220246ae56b5	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-54-91-37-153.com...
	i-056c03c857d24816b	Terminated	t1.micro	–	View alarms +	us-east-1a	–
	i-00452ddce5da2a49d	Terminated	t1.micro	–	View alarms +	us-east-1a	–
	i-05437f9b041187029	Terminated	t1.micro	–	View alarms +	us-east-1c	–
	i-09c7672e24286b56b	Running	t1.micro	2/2 checks passec	View alarms +	us-east-1c	ec2-54-237-202-110.co...
wordpress-dev	i-0c669b5525356217e	Stopped	t2.micro	–	View alarms +	us-east-1d	–
	i-0a78edf59164c113e	Terminated	t1.micro	–	View alarms +	us-east-1d	–
	i-043647645fce439c4	Terminated	t1.micro	–	View alarms +	us-east-1b	–

Select an instance

Figure 3 All Instances of autoscaling Group

Activity:-

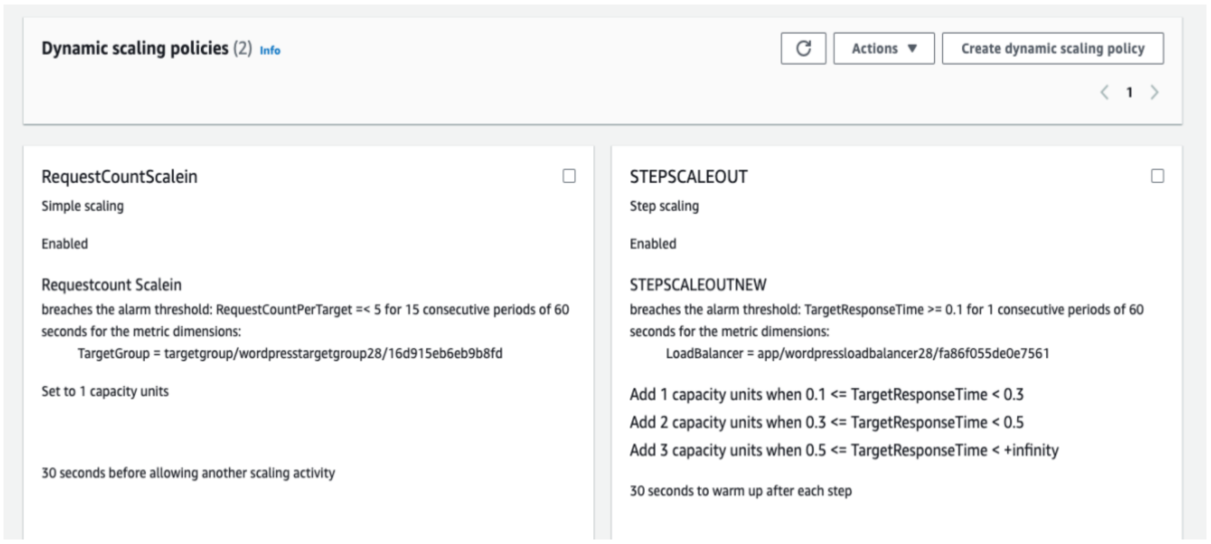


Auto Scaling group updated successfully

Status	Description	Cause	Start time
Successful	Launching a new EC2 instance: i-0583a89edb6a98f9b	At 2024-08-04T13:52:58Z a monitor alarm STEPSCALEOUTNEW in state ALARM triggered policy STEPSCALEOUT changing the desired capacity from 2 to 3. At 2024-08-04T13:52:59Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2024 Augus 02:53: +01:0
Successful	Terminating EC2 instance: i-01622bbae5e9bea64	At 2024-08-04T13:52:02Z a monitor alarm Requestcount Scalein in state ALARM triggered policy RequestCountScalein changing the desired capacity from 3 to 2. At 2024-08-04T13:52:11Z 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 2024-08-04T13:52:11Z instance i-01622bbae5e9bea64 was selected for termination.	2024 Augus 02:52: +01:0
Successful	Launching a new EC2 instance: i-01622bbae5e9bea64	At 2024-08-04T13:50:58Z a monitor alarm STEPSCALEOUTNEW in state ALARM triggered policy STEPSCALEOUT changing the desired capacity from 2 to 3. At 2024-08-04T13:51:03Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2024 Augus 02:51: +01:0
Successful	Terminating EC2 instance: i-0f69453b240063be3	At 2024-08-04T13:45:02Z a monitor alarm Requestcount Scalein in state ALARM triggered policy RequestCountScalein changing the desired capacity from 3 to 2. At 2024-08-04T13:45:08Z 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 2024-08-04T13:45:08Z instance i-0f69453b240063be3 was selected for termination.	2024 Augus 02:45: +01:0
Successful	Launching a new EC2 instance: i-0f69453b240063be3	At 2024-08-04T13:28:58Z a monitor alarm STEPSCALEOUTNEW in state ALARM triggered policy STEPSCALEOUT changing the desired capacity from 2 to 3. At 2024-08-04T13:29:04Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2024 Augus 02:29: +01:0

Figure 4Activity

Automatic Scaling:-



Dynamic scaling policies (2)

RequestCountScalein
Simple scaling
Enabled
Requestcount Scalein
breaches the alarm threshold: RequestCountPerTarget =< 5 for 15 consecutive periods of 60 seconds for the metric dimensions:
TargetGroup = targetgroup/wordpresstargetgroup28/16d915eb6eb9b8fd
Set to 1 capacity units
30 seconds before allowing another scaling activity

STEPSCALEOUT
Step scaling
Enabled
STEPSCALEOUTNEW
breaches the alarm threshold: TargetResponseTime >= 0.1 for 1 consecutive periods of 60 seconds for the metric dimensions:
LoadBalancer = app/wordpressloadbalancer28/fa86f055de0e7561
Add 1 capacity units when 0.1 <= TargetResponseTime < 0.3
Add 2 capacity units when 0.3 <= TargetResponseTime < 0.5
Add 3 capacity units when 0.5 <= TargetResponseTime < +infinity
30 seconds to warm up after each step

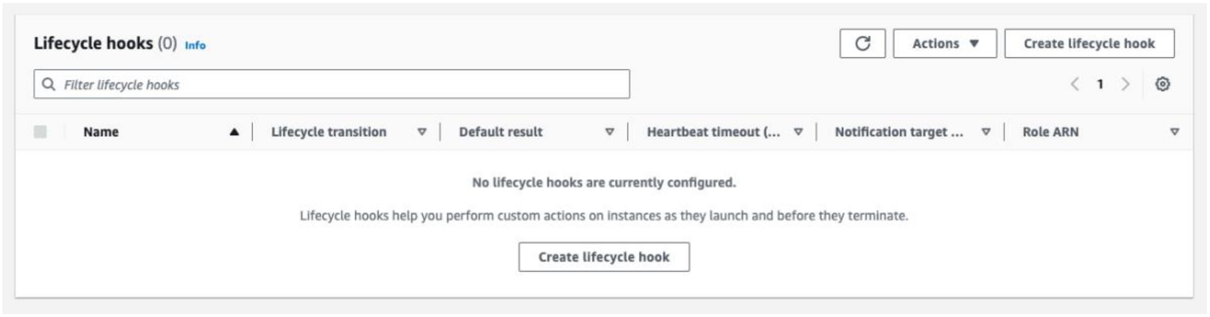
Figure 5 Dynamic Scaling Policies

Challenge:- I took the challenge from this coursework which excite me more. I took the t1 micro instance type as challenge because I can easily manage the response time from medium or large instance type. Due to t1 micro instance type, I used these thresholds value because t1 micro does not give any flexibility related to the values. You should be more precise to the values based on the data (graph of CloudWatch alarm) then after you can achieve the result in t1 micro and also I have a logic to take these metrices which I will explain in CloudWatch part. Also these values are based on CloudWatch alarm graph or you can say based on data.

Scale out:- I used step scaling for scale out with various threshold values like 100, 200 and 300ms.
When Target response time is greater than and equal to 100ms and lesser than 300ms, It will add one instance.
When Target response time is greater than and equal to 300 and lesser than 500ms, It will add two instance.
When Target response time is greater than and equal to 500, It will add three instance.

Scalein:- I used simple scaling for scalein with threshold value 5.
When Request count per target less than and equal to 5 then our instance value set to 1.
Instance Management:-

We can create lifecycle hook to pre warm our instances which help us to give better results but I did not apply because I did not know this is allowable for my coursework.


Lifecycle hooks (0) Info

Filter lifecycle hooks

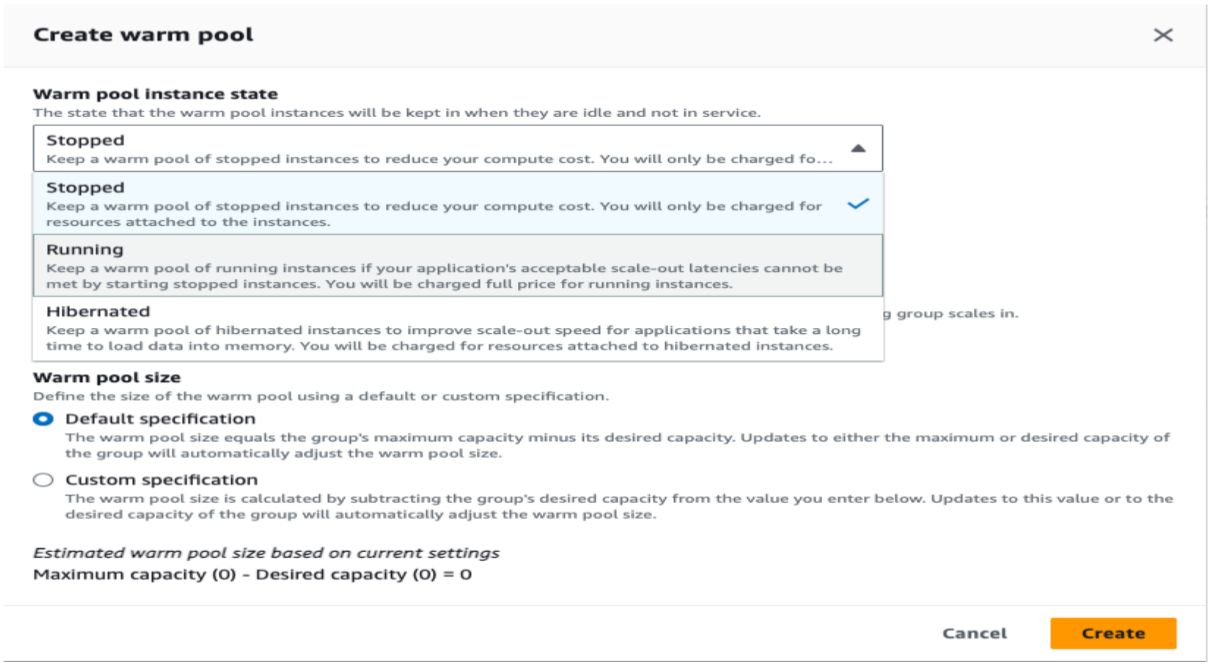
Name	Lifecycle transition	Default result	Heartbeat timeout (...	Notification target ...	Role ARN

No lifecycle hooks are currently configured.

Lifecycle hooks help you perform custom actions on instances as they launch and before they terminate.

Figure 6LifeCycle Hook

We can also create a warm pool for our hibernated instances which warm up our hibernated instances and then help out in the speed of scale out. But we did not authorise by this account so we cannot apply this service.


Create warm pool

Warm pool instance state
The state that the warm pool instances will be kept in when they are idle and not in service.

Stopped
Keep a warm pool of stopped instances to reduce your compute cost. You will only be charged fo...

Stopped ✓
Keep a warm pool of stopped instances to reduce your compute cost. You will only be charged for resources attached to the instances.

Running
Keep a warm pool of running instances if your application's acceptable scale-out latencies cannot be met by starting stopped instances. You will be charged full price for running instances.

Hibernated
Keep a warm pool of hibernated instances to improve scale-out speed for applications that take a long time to load data into memory. You will be charged for resources attached to hibernated instances.

g group scales in.

Warm pool size
Define the size of the warm pool using a default or custom specification.

● Default specification
The warm pool size equals the group's maximum capacity minus its desired capacity. Updates to either the maximum or desired capacity of the group will automatically adjust the warm pool size.

○ Custom specification
The warm pool size is calculated by subtracting the group's desired capacity from the value you enter below. Updates to this value or to the desired capacity of the group will automatically adjust the warm pool size.

Estimated warm pool size based on current settings
Maximum capacity (0) - Desired capacity (0) = 0

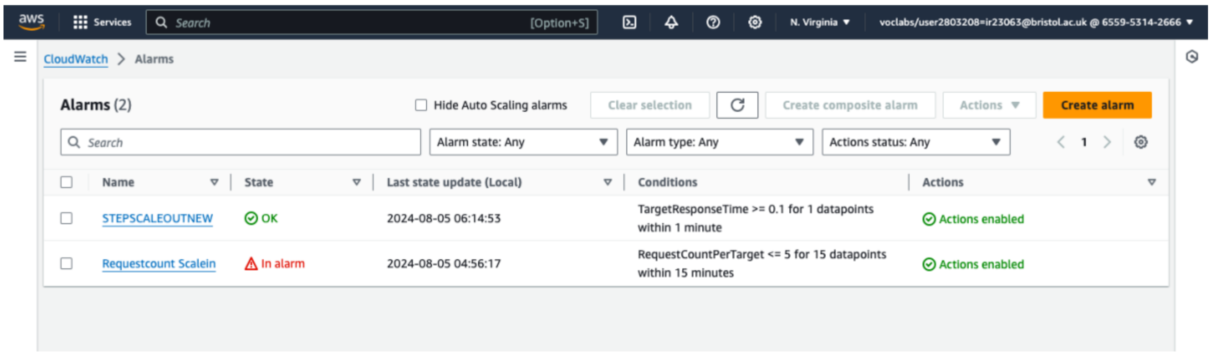
Figure 7Warm Pool

But we used on-demand instances service. On-demand instances have a fixed price and are always available which help us to optimise cost and availability.

Cloudwatch:- I tried various metrics for cloudwatch alarm like CPU Utilisation, NetworkIn, RequestCount,RequestCountperTarget, TargetResponseTime etc. I got best results from these two metrics. I used TargetResponseTime for SCALEOUT and RequestCountPerTarget for SCALEIN.

Scaleout Alarm will be triggered when TargetResponseTime is greater than or equal to 100ms for 1 datapoints within 1 minute.
ScaleIn Alarm will be triggered when RequestCountperTarget is less than or equal to 5 for 15 datapoints within 15 minutes.

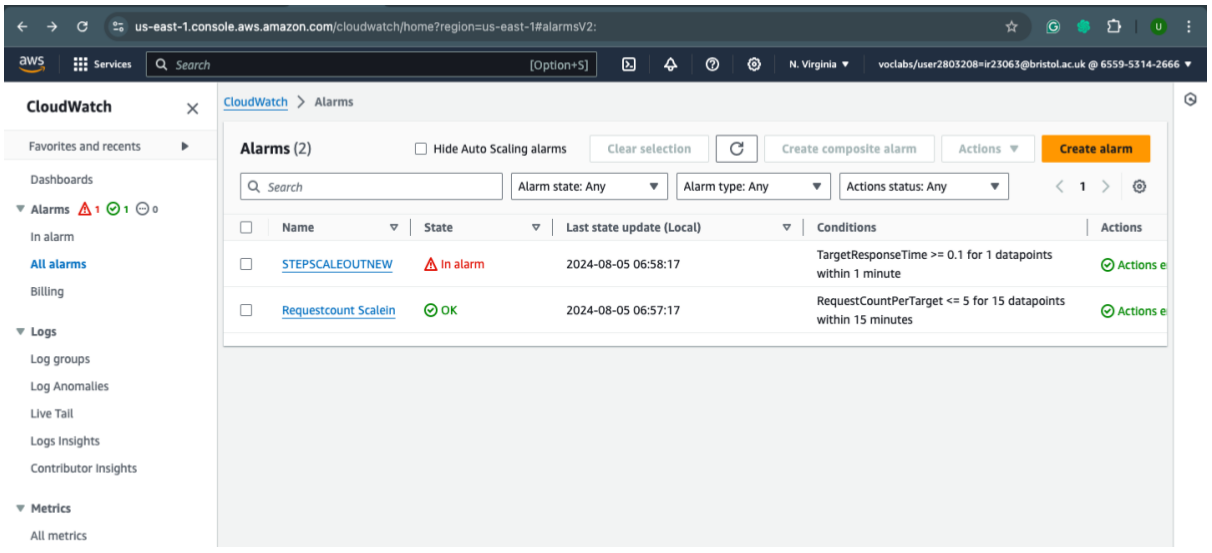
LOGIC:- I delayed the scalein alarm by 15 minutes. You can increase or decrease the delay. Actually I want to know the time when my website does not contain any clients or new clients that's why I created this scalein alarm which give us such type of information. This information can be very beneficial in future.



Alarms (2)

Name	State	Last state update (Local)	Conditions	Actions
STEPSCALEOUTNEW	OK	2024-08-05 06:14:53	TargetResponseTime >= 0.1 for 1 datapoints within 1 minute	Actions enabled
Requestcount Scalein	In alarm	2024-08-05 04:56:17	RequestCountPerTarget <= 5 for 15 datapoints within 15 minutes	Actions enabled

Figure 8CloudWatch Alarms



us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarmsV2:

Alarms (2)

Name	State	Last state update (Local)	Conditions	Actions
STEPSCALEOUTNEW	In alarm	2024-08-05 06:58:17	TargetResponseTime >= 0.1 for 1 datapoints within 1 minute	Actions e
Requestcount Scalein	OK	2024-08-05 06:57:17	RequestCountPerTarget <= 5 for 15 datapoints within 15 minutes	Actions e

Figure 9CloudWatch Alarms in different States

Task C - Perform Load Testing

Loadbalancer:-

Configuration:-

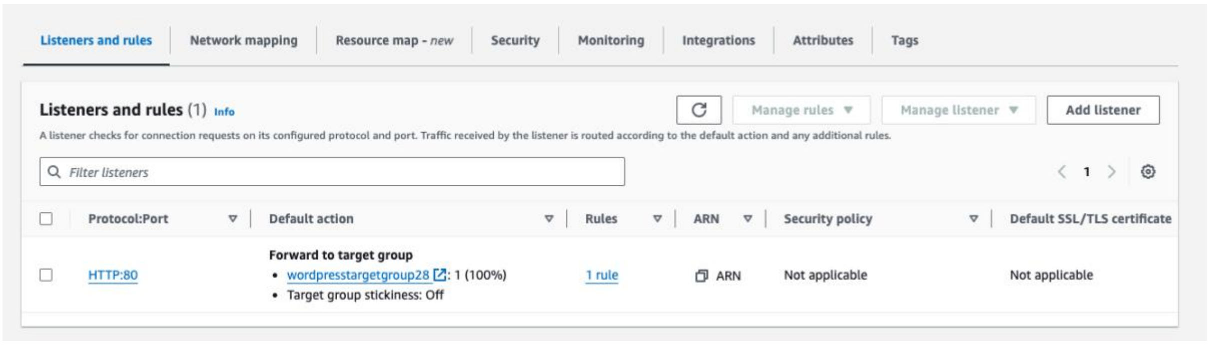
Name:- wordpressloadbalancer28

Load balancer type:- Application

Scheme:- Internet Facing

DNS Name:- wordpressloadbalancer28-188664688.us-east-1.elb.amazonaws.com

Listeners and rules:-



Listeners and rules | Network mapping | Resource map - new | Security | Monitoring | Integrations | Attributes | Tags

Listeners and rules (1) Info

Manage rules ▼ | Manage listener ▼ | Add listener

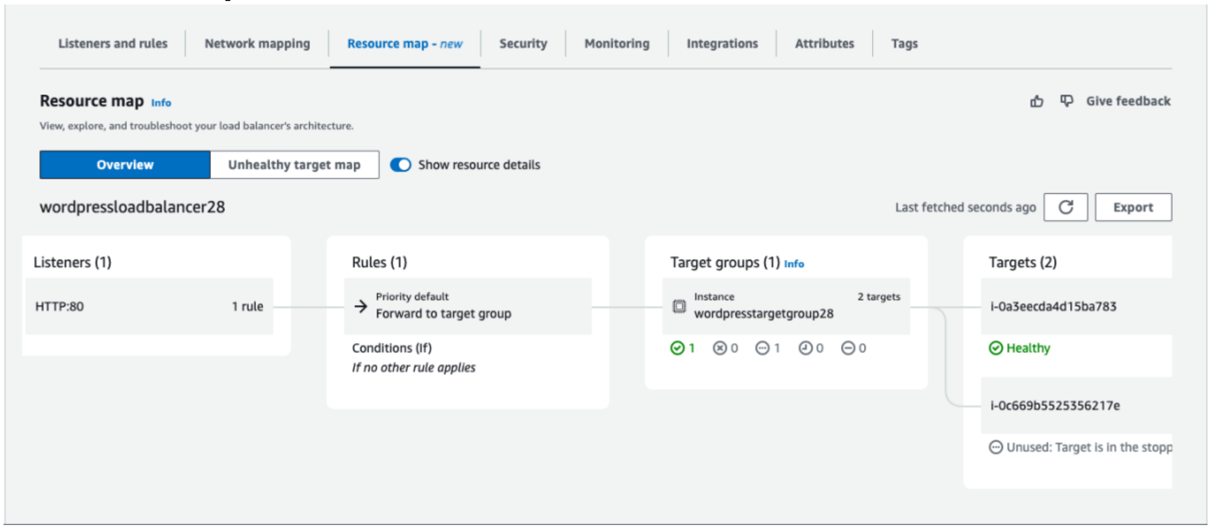
A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Filter listeners

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate
HTTP:80	Forward to target group • wordpresstargetgroup28: 1 (100%) • Target group stickiness: Off	1 rule	ARN	Not applicable	Not applicable

Figure 10Listeners and rules

Resource Map:-



Listeners and rules | Network mapping | Resource map - new | Security | Monitoring | Integrations | Attributes | Tags

Resource map Info

View, explore, and troubleshoot your load balancer's architecture.

Give feedback

Overview | Unhealthy target map | Show resource details

wordpressloadbalancer28

Last fetched seconds ago | Export

Listeners (1): HTTP:80 — 1 rule

Rules (1): Priority default — Forward to target group; Conditions (If) — *If no other rule applies*

Target groups (1) Info: Instance — wordpresstargetgroup28 — 2 targets; ✓1 ⊗0 ⊖1 ⊘0 ⊖0

Targets (2): i-0a3eecda4d15ba783 — Healthy; i-0c669b5525356217e — Unused: Target is in the stopp

Figure 11Resource Map

Targets Group:- Configuration

Target type:- Instance
Load Balancer:- wordpressloadbalancer28

Optimisation of Targets Group:-

Health Checks:- I optimise the health check of target group. This is very important for these types of task.

I change the healthy threshold (The number of consecutive health checks successes required before considering an unhealthy target healthy.) to 2 that means our targets used to be in healthy targets without any delay.

I change the Unhealthy threshold to 10 which means the number of consecutive health check failures require before considering a target unhealthy is 10. This delay the unhealthy circumstances.

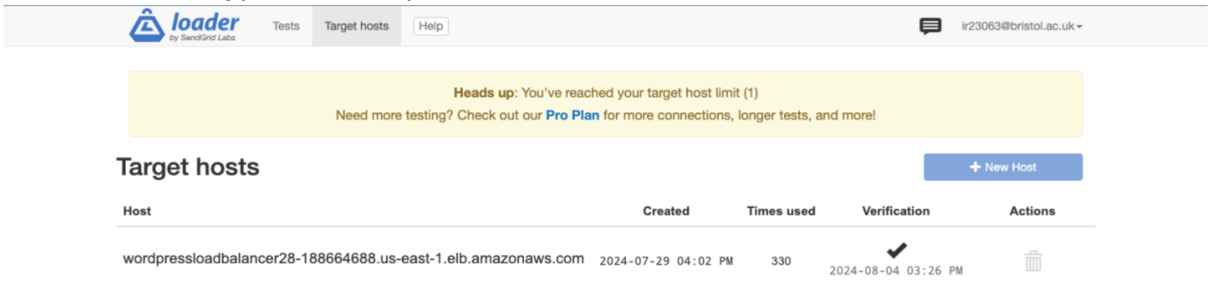
I change the value of timeout interval to 9. Timeout is the amount of time, in seconds, during which no response means a failed health check.

Total delay is unhealthy threshold multiply timeout interval that will be 90 seconds.

I took the Interval value to 10 seconds. Interval means the approximate amount of time between health checks of an individual target.

Attributes:- I also change the values of attributes like I change the value of Deregistration delay or you can say draining interval to zero second which help me to get better results.

Loader:- Test type:- Clients per Test. Times used:-330



Heads up: You've reached your target host limit (1)
Need more testing? Check out our **Pro Plan** for more connections, longer tests, and more!

Target hosts

Host	Created	Times used	Verification	Actions
wordpressloadbalancer28-188664688.us-east-1.elb.amazonaws.com	2024-07-29 04:02 PM	330	✓ 2024-08-04 03:26 PM	

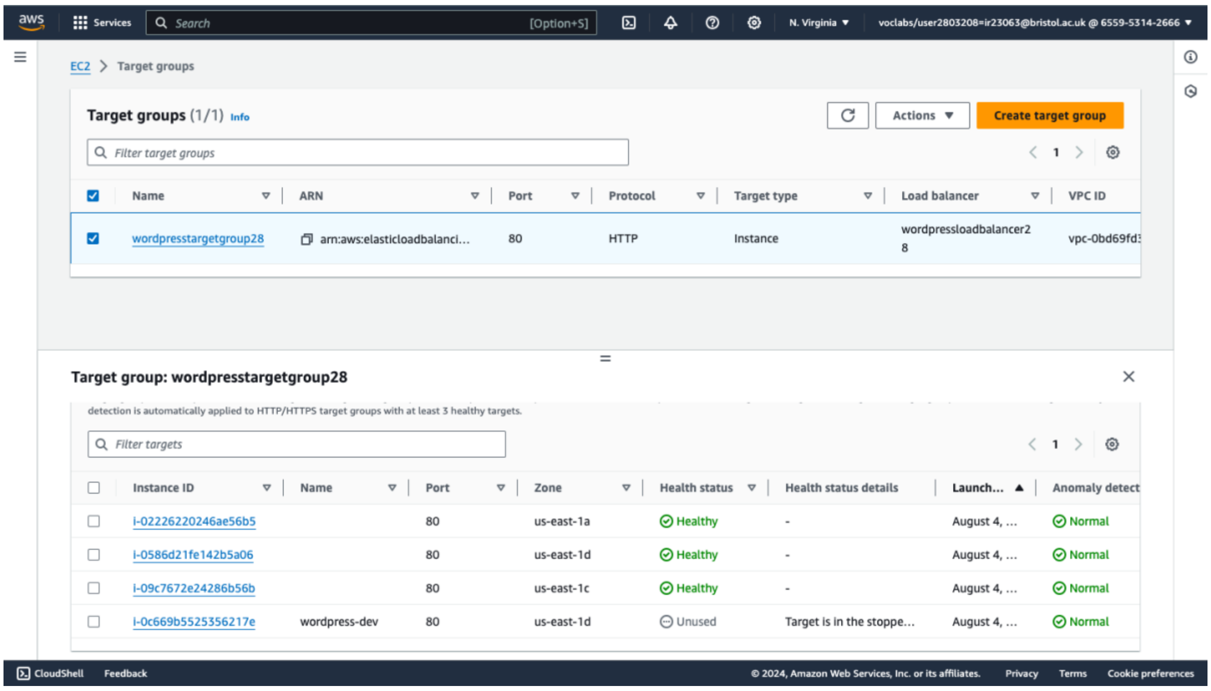
Figure 12Loader

Procedure to run the test:-

1. First we will stop the default wordpress instance.
2. Then give the values of minimum, maximum and desired capacity of autoscaling group. If you want more rapid result then you should take the minimum and desired capacity to 2 and maximum value to 3. You can also take the value to 1, 1 and 3 respectively.

3. You should wait some time to initialise or warm up the instances because we took the some delay in warmup.Most important part of the testing is that all the 3 instances should be healthy in both EC2 dashboard and load balancer.

4. Sometimes , you will not get the result in one go due to warmup, initialisations, healthchecks and instant load but after 2 or 3 times, you will get better and better results.



Target groups (1/1) Info

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
wordpresstargetgroup28	arn:aws:elasticloadbalanci...	80	HTTP	Instance	wordpressloadbalancer28	vpc-0bd69fd3

Target group: wordpresstargetgroup28

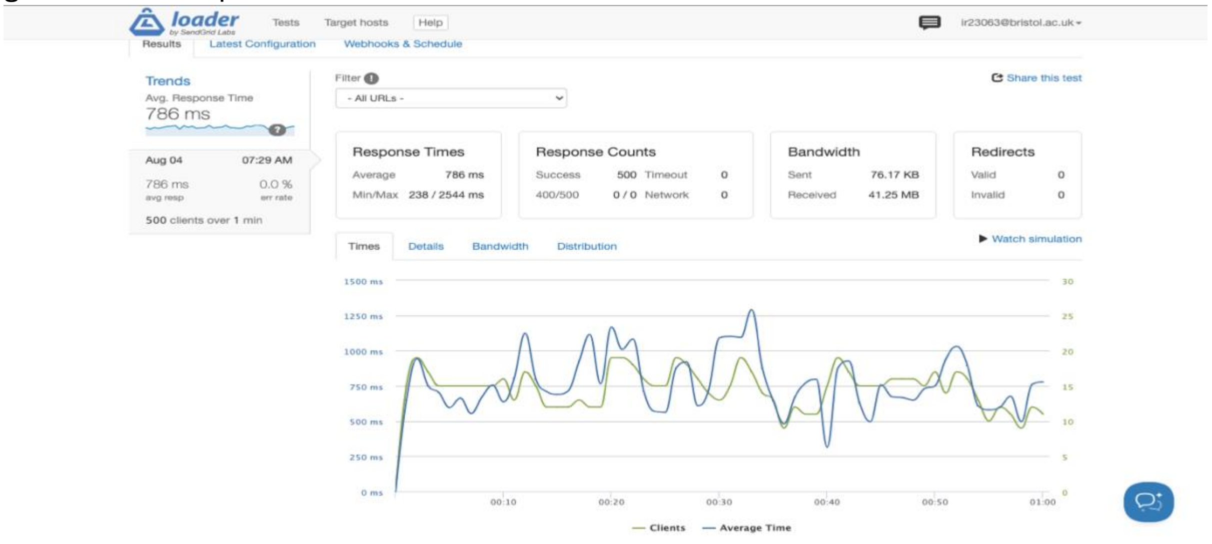
detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Instance ID	Name	Port	Zone	Health status	Health status details	Launch...	Anomaly detect
i-02226220246ae56b5		80	us-east-1a	✓ Healthy	-	August 4, ...	✓ Normal
i-0586d21fe142b5a06		80	us-east-1d	✓ Healthy	-	August 4, ...	✓ Normal
i-09c7672e24286b56b		80	us-east-1c	✓ Healthy	-	August 4, ...	✓ Normal
i-0c669b5525356217e	wordpress-dev	80	us-east-1d	Unused	Target is in the stoppe...	August 4, ...	✓ Normal

Figure 13Healthy Targets

Results:-

500 Clients:- My Average Response time is 786ms for 500 clients. I got 500 out of 500 success response counts and my graph is also overlapped each other very tightly which give the clear reponse of successful run test.



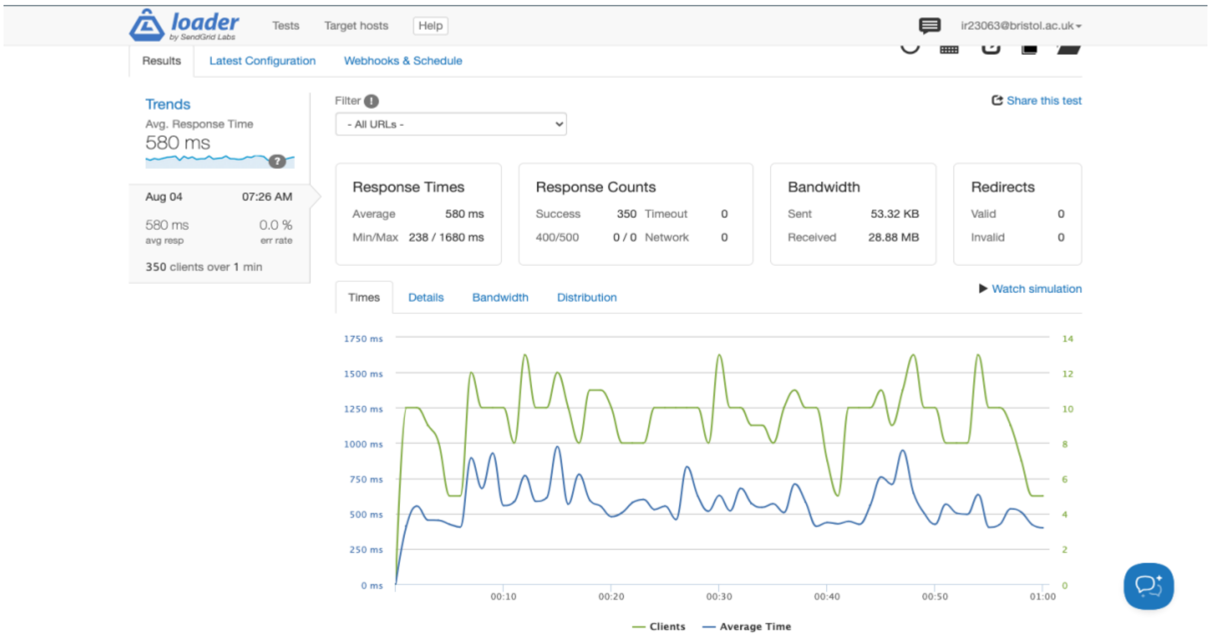
Trends
Avg. Response Time
786 ms

Aug 04 07:29 AM
786 ms avg resp 0.0 % err rate
500 clients over 1 min

Response Times		Response Counts				Bandwidth		Redirects	
Average	786 ms	Success	500	Timeout	0	Sent	76.17 KB	Valid	0
Min/Max	238 / 2544 ms	400/500	0 / 0	Network	0	Received	41.25 MB	Invalid	0

Figure 14 500clients Result

350 Clients:- My Average Response time is 580ms for 350 clients. I got 350 out of 350 success response counts and my graph is also resemblance to each other which give the clear reponse of successful run test.



loader by SendGrid Labs | Tests | Target hosts | Help | ir23063@bristol.ac.uk

Results | Latest Configuration | Webhooks & Schedule

Trends
Avg. Response Time
580 ms

Aug 04 07:26 AM
580 ms avg resp | 0.0 % err rate
350 clients over 1 min

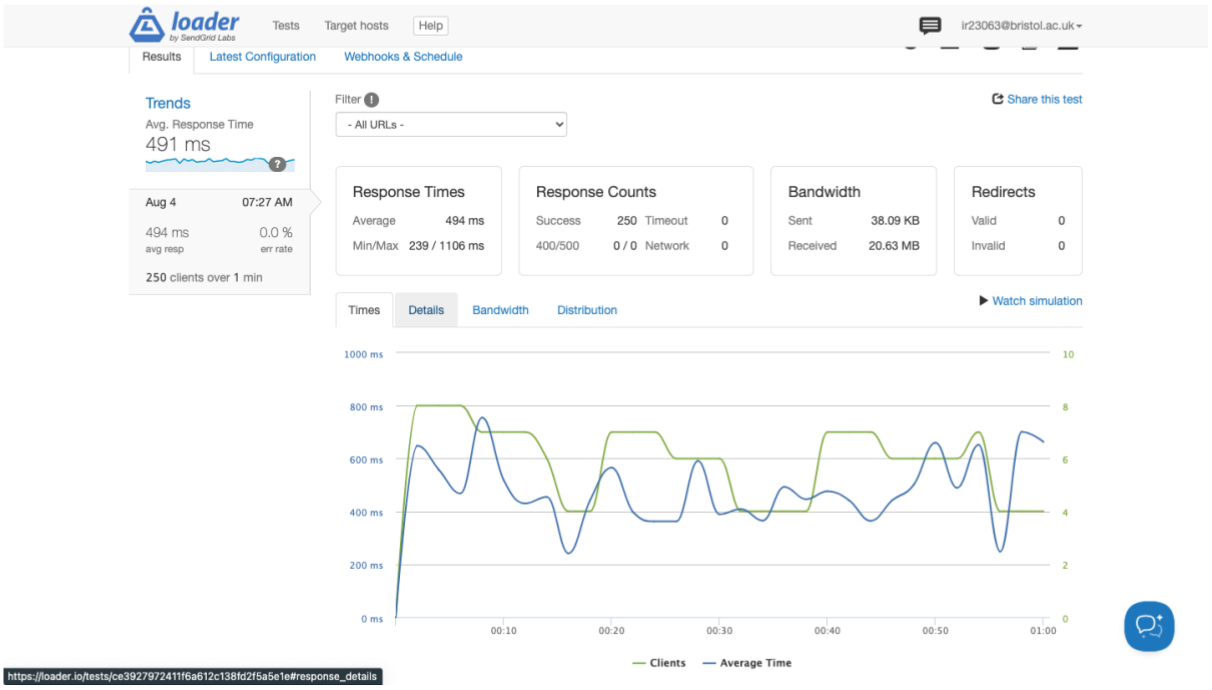
Filter: - All URLs -
Share this test

Response Times		Response Counts				Bandwidth		Redirects	
Average	580 ms	Success	350	Timeout	0	Sent	53.32 KB	Valid	0
Min/Max	238 / 1680 ms	400/500	0 / 0	Network	0	Received	28.88 MB	Invalid	0

Times | Details | Bandwidth | Distribution
Watch simulation

Figure 15 350clients Result

250 Clients:- My Average Response time is 494ms for 250 clients. I got 250 out of 250 success response counts and my graph is also resemblance to each other which give the clear reponse of successful run test.



loader by SendGrid Labs | Tests | Target hosts | Help | ir23063@bristol.ac.uk

Results | Latest Configuration | Webhooks & Schedule

Trends
Avg. Response Time
491 ms

Aug 4 07:27 AM
494 ms avg resp | 0.0 % err rate
250 clients over 1 min

Filter: - All URLs -
Share this test

Response Times		Response Counts				Bandwidth		Redirects	
Average	494 ms	Success	250	Timeout	0	Sent	38.09 KB	Valid	0
Min/Max	239 / 1106 ms	400/500	0 / 0	Network	0	Received	20.63 MB	Invalid	0

Times | Details | Bandwidth | Distribution
Watch simulation

https://loader.io/tests/ce3927972411f6a612c138fd2f5a5e1e#response_details

Figure 16 250Clients Result

Analysis on Different instance types:-

I used t1 micro in autoscaling groups which give me some challenging time but I enjoyed very much that's why I tried to optimise the response time on least instance type and I got the results but I also tried on other instance types like medium and large. One thing I observe if you try to optimise on least instance type after that if you increase the power then results get better and better.

Table 1.Result of Various Instance Types

	t1 micro	medium	large
250 Clients	494ms	389ms	324ms
350 Clients	580ms	457ms	409ms
500 Clients	786ms	667ms	602ms

Task D: Secure and Optimise the WordPress Architecture

To implement changes to WP architecture hosted in AWS EC2 instances, we will be focussing on three pillars; improving security, improving availability and reliability of the system, and reducing costs of infrastructure which enables users to use this WordPress.com website.

1. Security

a. IAM Policies and Roles:

Use IAM Roles: Attach IAM roles to the EC2 instances running WordPress to grant access privileges and permissions. Make sure they assign just the minimum set of permissions required to carry out their tasks.

IAM Policies: Create policies that ensure users and groups who use the AWS environment have strict permissions with Multi-Factor Authentication (MFA) enabled as a minimum.

IAM Users and Groups: Add an appropriate IAM user for each role and an IAM group for each user granting them the least privilege permissions (Services, 2024).

b. Security Groups and NACLs:

Security Groups: Specify security groups that allow traffic only from necessary sources (such as allowing only port 80 for HTTP or port 443 for HTTPS, from the internet, in the case that the server faces the internet; and restrict SSH [port 22] access to some known IP addresses).

Network ACLs (NACLs): Add an additional layer of protection by adding NACLs at the security group subnet level, to control both inbound and outbound traffic to the subnet where your EC2 instances are deployed (Services, Security group rules for different use cases, 2024).

c. Encryption:

Data Encryption: Use the AWS Key Management Service (KMS) to manage encryption keys used during the process of encrypting data at rest. This entails either EBS volumes attached to the EC2 instances or external data stored in S3. When applicable, enable SSL/TLS encryption for the data in transit (Codex, 2024).

Secret Management: Rotate your passwords using Amazon Web Services Secrets Manager to ensure any secrets such as database password and API keys are properly managed.

S3 Bucket Policies: Static content S3 buckets should have the appropriate bucket policy in place to enforce client-side encryption and prohibit public access.

d. Web Application Firewall (WAF):

AWS WAF: Configure AWS WAF to mitigate attacks against the WordPress application via common web vulnerabilities and exploits – for example, protect against two types of web attacks: SQL injection and cross-site scripting (XSS) attacks (Services, AWS WAF FAQs, 2024).

e. Monitoring and Logging:

Configure AWS CloudTrail to capture all API calls for your AWS account and write the logs to Amazon S3. This provides an audit trail for security analysis and compliance auditing.

Amazon CloudWatch: Access CloudWatch to view logs, metrics, and respond to unusual activity with alarms.

f. DDoS Protection:

AWS Shield: Protect yourself against common DDoS attacks with AWS Shield Standard, enabled by default at no additional cost, and against advanced DDoS attacks with AWS Shield Advanced (Services, AWS Shield Features, 2024).

2. Availability and Reliability

So as to enhance overall availability and reliability, the architecture should be fault-tolerant and load-tolerant. These are the tactics:

a. Auto Scaling:

Auto Scaling Group: Configure an Auto Scaling group to automatically add or remove EC2 instances depending on the demand of your application. Use CloudWatch metrics (CPU utilization, response time, etc) in the scaling policies.

Policy-based Scaling: Set up both real-time dynamic scaling policies and scheduled scaling (based on anticipated traffic flows at given times) to achieve both performance and cost efficiency.

b. Load Balancing:

Elastic Load Balancer (ELB): use an Application Load Balancer (ALB) to route traffic to a number of EC2 instances in order to not overload any single instance (fault tolerance).

Health Checks: Configure health checks on the ALB so that only healthy instances receive traffic. If an instance goes down, it is automatically removed from the list of rotating instances and a new instance is created.

Cross-Zone Load Balancing: Toggle this to On, so that any requests are sent to instances in other AZs if this AZ is depleted.

c. Multi-AZ Deployment:

Anything placed in multiple AZs is more fault-tolerant: you'd need failures in every AZ to bring down your whole system To create a multi-AZ deployment: 'Place instances in multiple AZs, like CloudFront's Edge Locations placed in multiple Availability Zones' So if you put something in multiple AZs, you get higher availability and better fault tolerance than that of any individual AZ.

Amazon RDS Multi-AZ: Those who use Amazon RDS for a WordPress database can enable Multi-AZ mode for high availability and failover.

Route 53 DNS Failover: route traffic to healthy instances or failover locations automatically in the event of an outage (Services, Amazon RDS Multi-AZ, 2024).

d. Backup and Recovery:

Automatic backups: Use AWS Backup to automate regular backups of EBS volumes and RDS databases. Use multiple AZs or regions to store backups to enable disaster recovery.

Keep the EC2 instances and databases backed up: regularly creating snapshots, so that if failures occur, you can simply recover from a previous snapshot with Lifecycle Manager.

e. Content Delivery Network (CDN):

Amazon CloudFront: Use CloudFront for a CDN to serve up static content to the region where they are delivered, capturing content as close to the delivery location as possible and thereby reducing latency times for location specific users (Services, Amazon CloudFront, 2024).

f. Elastic File System (EFS):

Amazon EFS: Shared storage between multi-EC2 instances Use EFS for clouds with multiple EC2 instances (eg, between multiple web nodes) where each EC2 is located in a different AWS availability zone (AZ) for high availability and data durability (Services, What is Amazon Elastic File System?, 2024).

3. Cost Optimization

Cost-saving means that no needless expenditures are made and that resources are used efficiently.

a. Right-sizing Instances:

Instance Type Optimisation: Examine utilisation trends and performance metrics to select the least expensive instance types. Take a look at AWS Compute Optimiser to get recommendations.

Reserved Instances and Savings Plans: Adjust the right amount of Reserved Instances or Savings Plans for predictable workloads considering your specific situation to pay less.

b. Storage Optimization:

S3 for Static Content: Move static content (images, videos) off the WordPress app entirely and serve them from Amazon S3. This reduces load on your EC2 instances and takes advantage of S3's lower per-gigabyte storage costs.

Lifecycle Policies: Use S3 lifecycle policies to migrate data to lower-cost storage classes (such as S3 Glacier) based on access patterns.

EBS Optimization: Monitor and likely scale back or adjust EBS volumne types and sizes based on usage; Use Elastic Volumes to dynamically resize and scale for storage.

EFS Infrequent Access: Use IA storage class for files that aren't accessed frequently to save on costs (Services, Amazon EBS features, 2024).

c. Cost Management Tools:

AWS Budgets: Create AWS Budgets to track and control spending. Get notifications when usage or costs surpass certain thresholds.

Cost Explorer: Use AWS Cost Explorer to analyze spending patterns and explore opportunities to spend less.

AWS Trusted Advisor: On a real-time basis, receive recommendations from Trusted Advisor around cost optimisation, performance, security and fault tolerance.

d. Serverless Architectures:

Wherever possible, use AWS Lambda functions to do specific types of work: these functions can reduce cost associated with always-on instances and scaling (Services, AWS Academy Cloud Foundations [56499], 2023).

Issues:- 1. Experiment Autoscaling group:- If we are creating a launch template directly from default wordpress instance and then if we use this template in autoscaling group then copy of default instances will be created by autoscaling group.I know this is not the right way but this is also give the very good results.

2. If you change the instance type of default instance like medium or large then you will get very less response time because default architecture is disturbed and then there is no means of autoscaling. When this type of problem occurred, you just have to create new default instance with default configuration.

3. Loader.io Verification Problem:- Solved by multiple using.

Issue with Consistent Blog Post Visibility in Scaled Architecture

In a scaled architecture with multiple instances, if you add a blog post to one instance, this post may not appear on other instances. This inconsistency occurs because each instance has its own local storage. When you add a blog post, it is stored locally on that specific instance, and the other instances do not have access to this new data. This issue arises because there is no shared storage or synchronization mechanism among the instances.

Solution: Using Amazon RDS and Amazon EFS for Consistent Data Storage

To fix this, all instances need to access the same data, for which we can use Amazon RDS for the database and Amazon EFS for shared file storage. Here's the high-level workflow to set it up:

Step-by-Step Workflow

Step 1: Setup Amazon RDS for WordPress Database

a. Create an RDS Instance:

b. Use Amazon RDS to create a MySQL or MariaDB database instance.

c. Ensure that the RDS instance is configured for Multi-AZ deployment for high availability.

d. Configure appropriate security groups to allow connections from your EC2 instances.

e. Migrate Existing Database:

f. Export your current WordPress database.

g. Import the database dump into the RDS instance.

h. Update WordPress Configuration:

i. Modify the wp-config.php file on all EC2 instances to point to the RDS database.

Step 2: Setup Amazon EFS for Shared File Storage

a. Create an EFS File System:

b. Create an Amazon EFS file system.

c. Make sure it is accessible from all your EC2 instances by creating appropriate security groups and mount targets in each Availability Zone.

d. Mount EFS on EC2 Instances:

e. Install NFS client on your EC2 instances.

f. Mount the EFS file system to a directory, such as /var/www/html/wp-content/uploads.

g. Update the WordPress configuration to use this directory for uploads.

h. Update WordPress Configuration:

i. Make sure that WordPress's wp-config.php file is updated to use the EFS-mounted directory for media uploads.

Step 3: Configure Auto Scaling and Load Balancing

a. Auto Scaling Group:

b. Make sure your Auto Scaling group is using the EC2 launch commands necessary to mount EFS.

c. Use a launch configuration or launch template that includes the setup for mounting EFS.

d. Application Load Balancer (ALB):

e. Use ALB to distribute traffic evenly across your EC2 instances.

f. Ensure that health checks are properly configured to monitor the instances.

Step 4: Ensure Consistency and Performance

a. Database Performance:

b. Use Amazon RDS read replicas when required to offload the read traffic from the primary database instance.

c. Monitor RDS performance and adjust instance types and storage as needed.

d. File System Performance:

e. Select the appropriate Amazon EFS performance mode (General Purpose or Max I/O) depending on your application's needs.

f. Configure EFS lifecycle management to move less accessed files to a lower-cost storage class to minimise cost.

Bibliography

Services, A. W. (2024, 07 10). *Using an IAM role to grant permissions to applications running on Amazon EC2 instances*. Retrieved from aws: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html

Services, A. W. (2024, 08 1). *Security group rules for different use cases*. Retrieved from aws: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules-reference.html

Codex, C. (2024, 02 17). *Encrypting Data on EC2 Instances using AWS Key Management Service (KMS)*. Retrieved from ReinTech Media: https://reintech.io/blog/encrypting-data-ec2-instances-aws-kms

Services, A. W. (2024). *AWS WAF FAQs*. Retrieved from aws: https://aws.amazon.com/waf/faqs/

Services, A. W. (2024). *AWS Shield Features*. Retrieved from aws: https://aws.amazon.com/shield/features/

Services, A. W. (2024). *Amazon RDS Multi-AZ*. Retrieved from aws: https://aws.amazon.com/rds/features/multi-az/

Services, A. W. (2024). *Amazon CloudFront*. Retrieved from aws: https://aws.amazon.com/cloudfront/

Services, A. W. (2024). *What is Amazon Elastic File System?* Retrieved from aws: https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html

Services, A. W. (2024). *Amazon EBS features*. Retrieved from aws: https://aws.amazon.com/ebs/features/

Services, A. W. (2023). *AWS Academy Cloud Foundations [56499]*. Retrieved from aws academy: https://awsacademy.instructure.com/courses/56499

aws. (2024). *Amazon EC2 T2 Instances*. Retrieved from aws: https://aws.amazon.com/ec2/instance-types/t2/

Amazon EC2 instance become slow everyday. (2024). Retrieved from stackoverflow: https://stackoverflow.com/questions/43670504/amazon-ec2-instance-become-slow-everyday

Is starting an AWS instance with only ssh to port 22 significantly insecure? (2024). Retrieved from StackExchange: https://security.stackexchange.com/questions/233785/is-starting-an-aws-instance-with-only-ssh-to-port-22-significantly-insecure

Phadke, R. (2020, 09 23). *Building Self-healing Load Balancing Services*. Retrieved from vmware: https://blogs.vmware.com/load-balancing/2020/09/23/self-healing-load-balancing-services/