EMATM0051 Large Scale Data Engineering

Task 1

Overview of the Key Responsibilities and Role of AWS Services in "MyTravel" Architecture

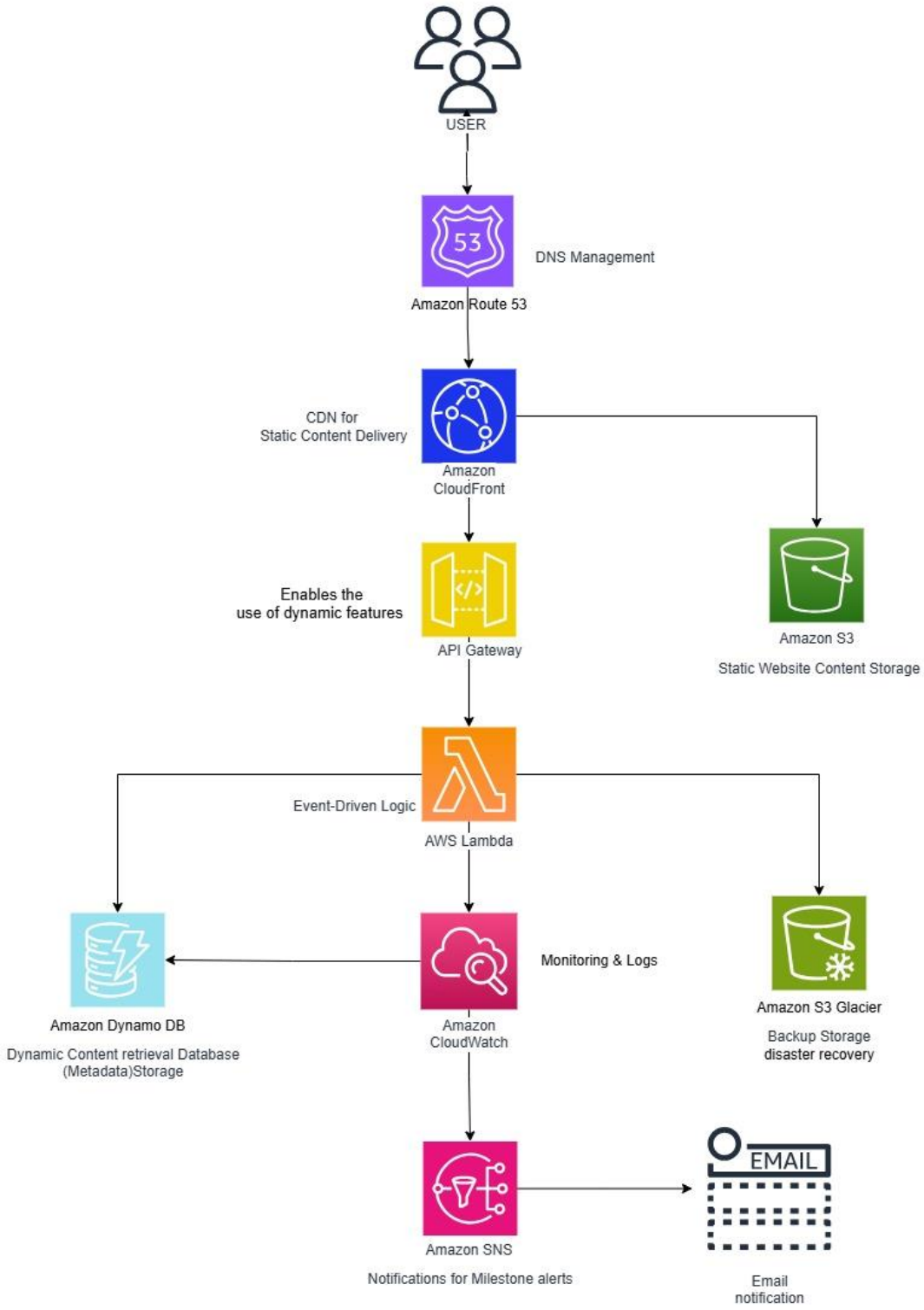
Key Importance and Responsibility	AWS Services Used	Uses and Benefits
Global Accessibility and High Availability	Amazon Route 53	Provides DNS routing to direct user requests to the nearest CloudFront edge location, ensuring high availability and low-latency content delivery.
	Amazon CloudFront	Caches static and dynamic content globally, improving load times and ensuring fast access for users worldwide.
	AWS Lambda & DynamoDB	Processes dynamic requests efficiently and stores metadata, supporting seamless website performance.
High Performance and Low Latency	Amazon CloudFront	Reduces latency by caching frequently accessed content at edge locations, improving website speed and user experience.
	AWS Lambda	Ensures fast processing of dynamic content requests through event-driven execution.
	Amazon DynamoDB	Provides low-latency access to metadata and other application data, maintaining quick response times.
Security	Amazon S3	Stores static website content securely with controlled access policies to protect data integrity.
	AWS Lambda	Implements secure access to backend services, managing sensitive operations and ensuring safe data handling.
	Amazon CloudFront	Protects content by integrating HTTPS and enforcing secure transport policies between users and the website.
Cost-Efficiency	AWS Free Tier	Enables cost savings by offering free usage limits for key services like Lambda, S3, and DynamoDB during the initial setup phase.
	Serverless Architecture	Minimizes costs by eliminating the need for server management and scaling resources automatically based on demand.
	S3 Glacier	Provides low-cost long-term storage for infrequently accessed backup data, optimizing costs for disaster recovery.
Congratulatory Email	AWS Lambda	Triggers email notifications when the website achieves significant milestones, like exceeding 1,000,000 visits.
Automatic Backup	AWS Lambda	Automates the backup process by triggering data archival to Amazon S3 Glacier, ensuring data recovery in case of a disaster.
	S3 Glacier	Stores backups with high durability and cost efficiency, supporting long-term data retention strategies.
Monitoring and Error Handling	Amazon CloudWatch	Monitors the health of AWS services, logs system events, and triggers alarms for performance or security anomalies.
	SNS	Sends alerts (e.g., email or SMS) when specific thresholds are exceeded, enabling prompt issue resolution and system optimization.

Backup and Storage functions in the design:

Automatic Archival by Amazon Lambda which triggers backups to Amazon S3 Glacier for disaster recovery. Low-Cost Storage: Reduces archival storage costs. Cost-effective long-term backup storage is given by Amazon S3 Glacier. Amazon S3 hosts the static assets of a website, such as pictures, CSS, and blog posts. Durability and scalability as it supports - 99.999999999% durability and scales by increased data upload. Backup and disaster recovery - defaults to backing up the content over different availability zones and will archive it at S3 Glacier and it reduces direct requests to S3 with CloudFront caching. Also DynamoDB is induced to respond for the requests by API Gateway which ensures additional security as Glacier is backup storage and is not used for processing requests and solely used for archival.

The architecture abides by the organizational pillars of Amazon Web Services (AWS) - the six pillars of the AWS Well-Architected Framework: **Operational Excellence**- AWS services like API Gateway, CloudWatch and SNS provide robust monitoring and alerting mechanisms, enabling the system to operate efficiently and handle events. Scalability by AWS API Gateway secure REST API accepting HTTPS requests. **Security** - API Gateway securely manages dynamic request routing to AWS Lambda, and Amazon S3 Glacier stores backups for disaster recovery, protecting sensitive metadata and user content. **Reliability** - Route 53 and CloudFront improve system reliability by ensuring high availability and low-latency content delivery through global edge locations. **Performance Efficiency**: CloudFront caches static content globally, reducing load times and improving user experience, while DynamoDB provides low-latency access to metadata. **Cost Optimization**: S3 Glacier offers a cost-effective solution for archiving infrequently accessed data. **Sustainability:** Serverless Architecture by using Lambda, the architecture minimizes the environmental impact through efficient resource utilization, as AWS optimizes underlying hardware and energy use.

AWS Services	Cost Efficiency Monthly using Free -Tier
Amazon Route 53	$0.50/month for hosted zones. First 1 million queries are free each month
Amazon CloudFront	1 TB of data transfer out. 10 million HTTP/HTTPS requests per month.
Amazon S3 (Storage for Static Website Content)	5 GB of standard storage, 20,000 GET requests, and 2,000 PUT requests per month.
API Gateway	1 million requests per month.
AWS Lambda	1 million requests and 400,000 GB-seconds per month.
Glacier	10 GB of retrieval data
DynamoDB	25 GB of storage, 25 write capacity units, and 25 read capacity units
CloudWatch	10 custom metrics, 5 GB of logs, and 3 dashboards
SNS	1 million publishes, 1,000 email notifications



Explanation of the workflow of the diagram:

"MyTravel" website page is accessed by the user request using their browser. Amazon Route 53 – Domain Name System (DNS) turns domain names into IP addresses, and makes the user request route to CloudFront distribution based on the nearest edge location and the CloudFront fetches and receives the static website content from S3 storage and for dynamic content retrieval requests the CloudFront forwards the request to the Amazon API Gateway. API Gateway invokes the Amazon Lambda, to process the request by an event-driven logic. The Amazon Lambda fetches the dynamic content such as from the Amazon Dynamo DB a NOSQL Metadata database storage. As, it is required to send a congratulatory email when the total number of website visits exceeds 1,000,000, Lambda function is connected to CloudWatch which is registering and logging the events that triggers the SNS to generates the email alert/notification when constraint is reached. CloudWatch events triggering the Lambda function for a scheduled event to check the website's health and each failed health is recorded at a Cloud watch metric and error handling in the DynamoDB Storage and Monitoring and logging the events.

TASK – 2:

TASK A: Installation and Configuration of the WordFreq Application

The WordFreq application is a cloud-based word frequency analyzer that is designed to process text files uploaded by users and return the top ten most frequent words from each file.

The WordFreq application using AWS services, focusing on key configurations for EC2, S3, SQS, and DynamoDB, as well as preparing the application environment on the EC2 instance.

Configuration and AWS Services used:

We began with the configuration of Launching an EC2 instance with the **Ubuntu Server 22.04 LTS** AMI, instance type **t2.micro**, named it **wordfreq-dev**, and using **learnerlab-keypair** key pair with extension .ppk for Windows which will **allow the SSH access** using PuTTY in the security group for connection with CLI, and assign the **EMR_EC2_DefaultRole** IAM instance profile under advanced details.

Created the S3 buckets, we created two buckets: an **uploading bucket** - tr-wordfreq-nov24-uploading for storing and loading text files and an **processing bucket** - tr-wordfreq-nov24-processing to hold files after transfer, with upload notifications enabled to trigger messages to the WordFreq SQS queue.

Created two SQS Queues -

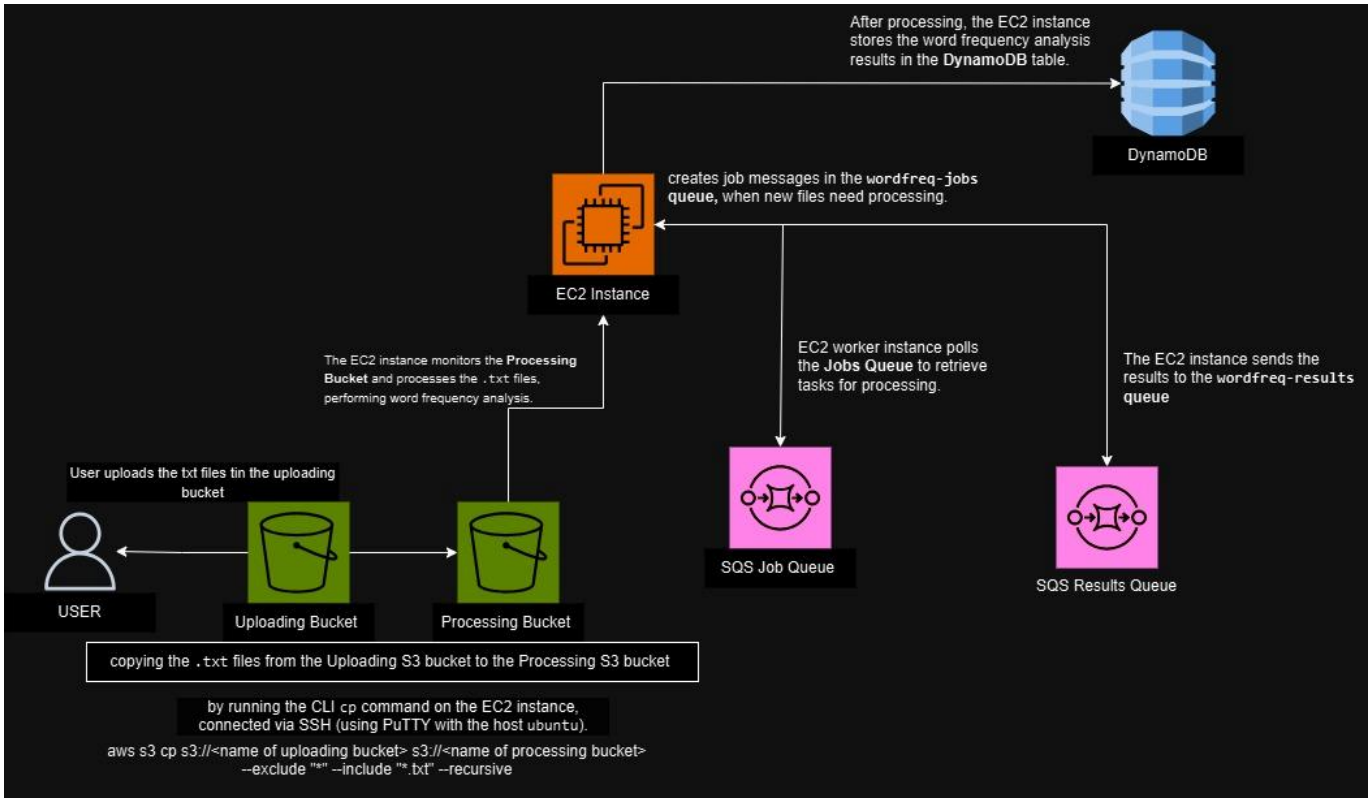
- **Job Queue -** Created a wordfreq-jobs SQS queue with a standard queue type and an access policy allowing any AWS entity to write to the queue.
- **Result Queue:** Created another queue named wordfreq-results with a similar access policy.

In the S3 Console, created an event notification for the processing bucket to trigger when a file is uploaded, sending the message to the wordfreq-jobs SQS queue.

Set up the connection to EC2 instance using SSH – PuTTY – Windows to upload the WordFreq Application Zip file to the uploading S3 bucket.

Application working:

The Word-Frequency Application performs word frequency analysis on the .txt files uploaded by the users and stores the results in a DynamoDB Table. Users upload .txt files into the Uploading Bucket (S3 bucket) and by running the CLI cp command on the EC2 instance, connected via SSH (using PuTTY with the host ubuntu) the Files from uploading bucket is copied to the processing bucket. The EC2 instance monitors the Processing Bucket and processes the .txt files, performing word frequency analysis and then creates job messages in the wordfreq-jobs queue, when new files need processing. EC2 worker instance polls the Jobs Queue to retrieve tasks for processing. After processing, the EC2 instance stores the word frequency analysis results in the DynamoDB table. The EC2 instance sends the results to the wordfreq-results queue. The processed word frequency data is available in DynamoDB, making it accessible for further use. the Word Frequency Application on the EC2 instance is the main processing. EC2 is used for file monitoring and management, Data processing and computation in S3, job queues management with SQS, and results management using DynamoDB and helps the application to perform its analysis using go language efficiently.



TASK B - Designing and Implementing EC2 Auto-scaling

After configuring AWS Services – S3 buckets, SQS Queues, DynamoDB, and EC2 set up for the word-freq application in each job process takes a random time to complete between 10-20 seconds to be able to process multiple load files we will be adjusting to add scaling in and scaling out time of the Application and use the EC2 instances AMI and stop the EC2 instance. Implementing the auto-scaling functionality for the WordFreq application to demonstrating how to configure the auto-scaling policies scale in and scale out.

First, we create a launch template using AMI image of the EC2 instance

Step-by-Step Instructions for Setting Up EC2 Auto Scaling Template

Access Auto Scaling Guidance:

- In the AWS Management Console, in **EC2 Auto Scaling** section.
- Select the option: **Provide guidance to help me set up a template that I can use with EC2 Auto Scaling**.

Choose Amazon Machine Image (AMI):

- In the **Application and OS Images (Amazon Machine Image)** section:
- Click **My AMIs**.
- Select the appropriate AMI for your application (e.g., **Web Server AMI**).

Select Instance Type:

- Under the **Instance type** section:
- Choose **t2.micro** as the instance type.

Choose Key Pair for SSH Access:

- In the **Key pair name** dropdown menu:
- Select **learnerlab-keypair** (or the appropriate key pair for your AWS account).

Configure Security Groups:

- In the **Firewall (security groups)** section:
- Select **Select existing security group**.
- Choose the security group associated with your application (e.g., **wordfreq-dev-sg sg-0e1fb658c06901d13**).

Select Virtual Private Cloud (VPC):

- Under the **VPC** section:
- Ensure the correct VPC is selected (e.g., **vpc-0ce59f96d08407c54**).

Expand Advanced Details:

- Scroll down to the **Advanced details** section.
- Click on the arrow to **expand** the advanced options.

Enable Detailed Monitoring:

- Locate the **Detailed CloudWatch monitoring** setting.
- Check the box to **Enable** detailed monitoring for your instances.

Metrics used – SQS - ApproximateNumberOfMessagesVisible

Create Cloud Watch Alarms:

- Scale-in Alarm- ApproximateNumberOfMessagesVisible >= 50 for 1 datapoints within 1 minute/ 60 secs and Add 1 and then wait – 150 seconds
- Scale-out Alarm - ApproximateNumberOfMessagesVisible =< 5 for 1 consecutive within 1 minute/ 60 secs and Remove 1 and then wait – 150 seconds

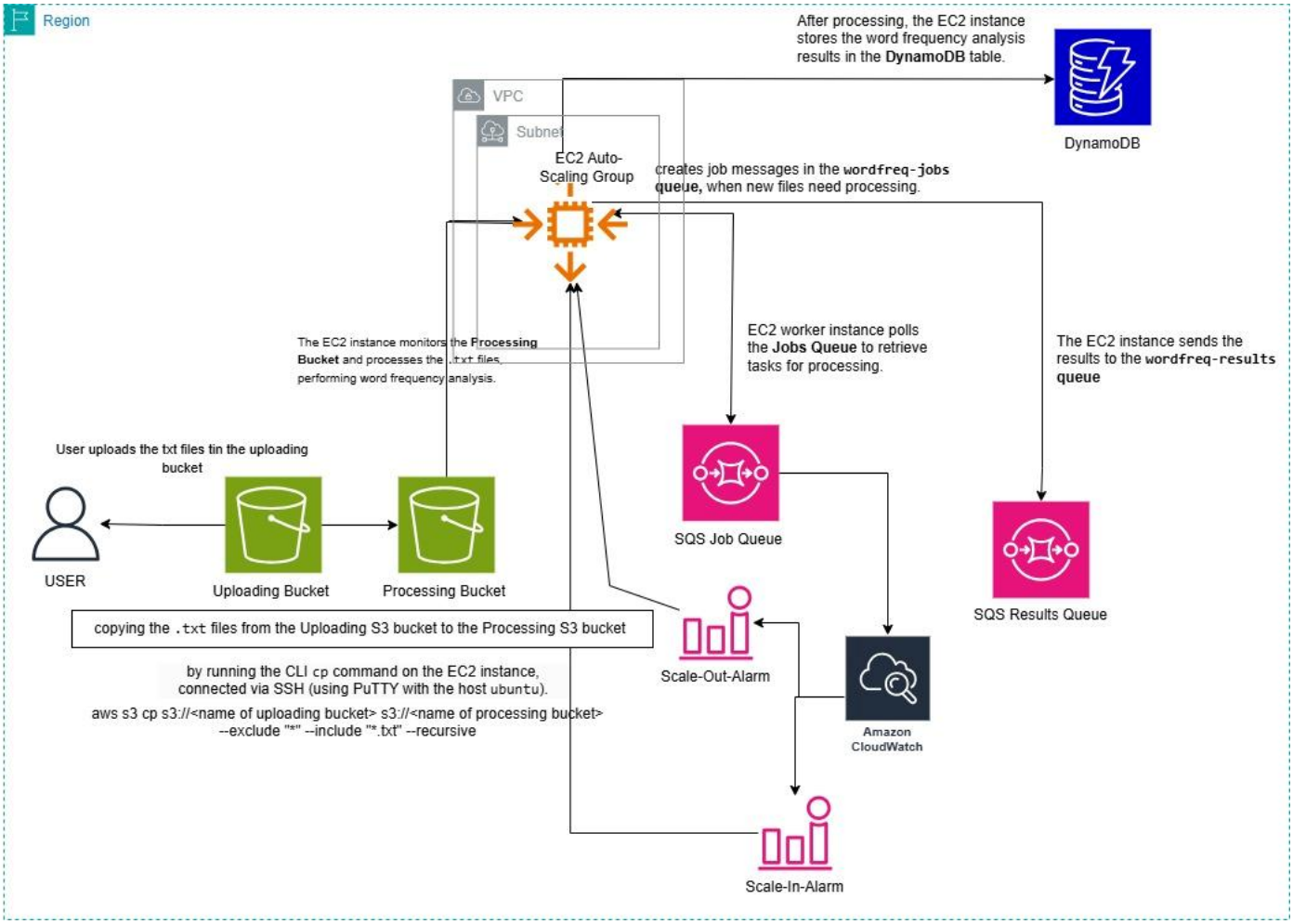
Create Auto Scaling Policies:

Scale-in- Policy – Simple Scaling with the Remove 1 instance

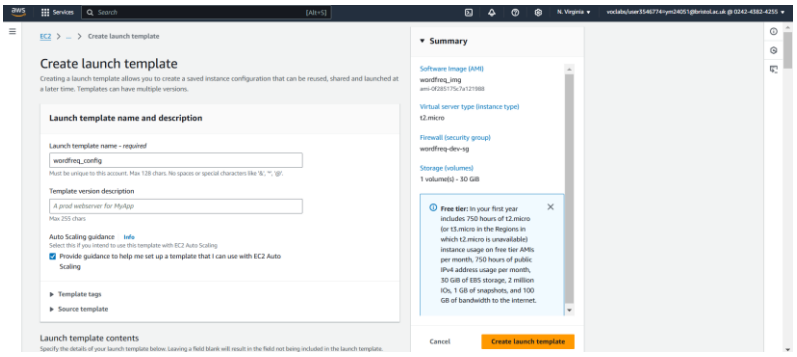
Scale-out-Policy - Simple Scaling with the Add 1 instance

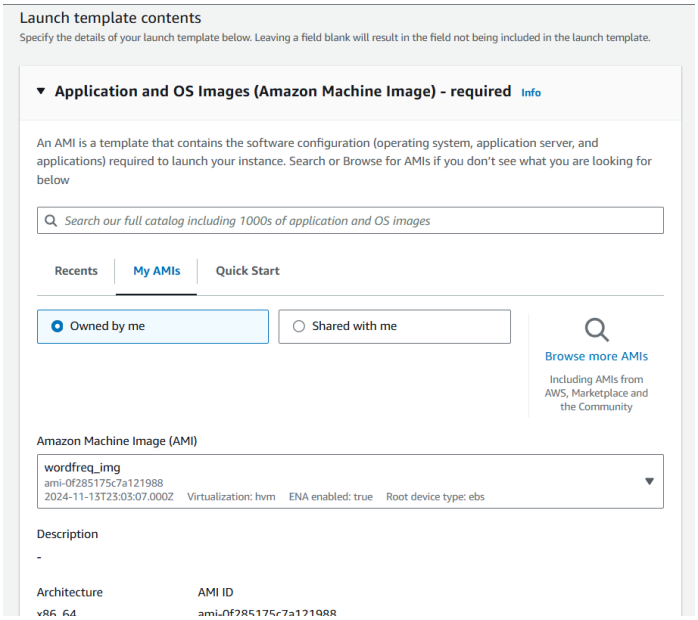
Summary of Configuration (Task B)

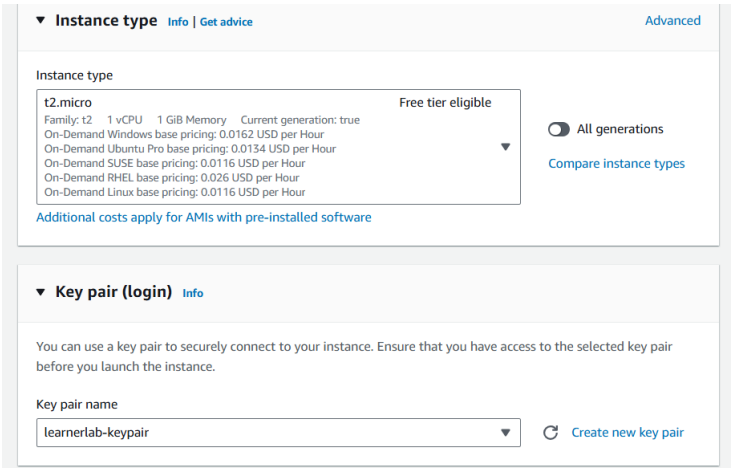
- **Launch Template**: Configured with t2.micro, key pair, security group, and detailed monitoring.
- **Scale-Out Policy**: Triggers when ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minutes.
- **Scale-In Policy**: Triggers when ApproximateNumberOfMessagesVisible >= 50 for 1 datapoints within 1 minutes.
- **Cooldown Period**: 150 seconds for both scale-out and scale-in actions.
- **Minimum Instances**: 1 (always active to handle the live workload).
- **Maximum Instances**: 8
- **Desired Instances**: 1
- **Metrics**: SQS queue length (ApproximateNumberOfMessagesVisible) is appropriate for scaling decisions.

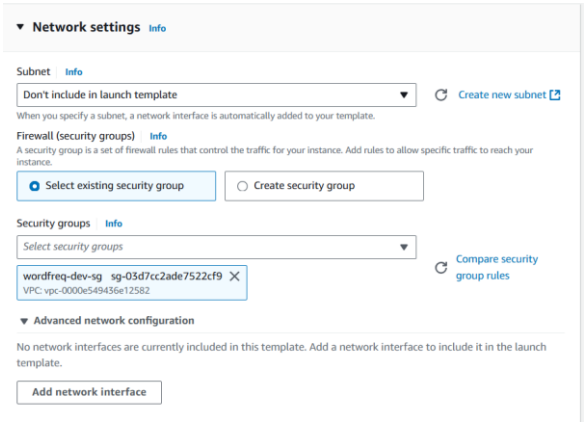


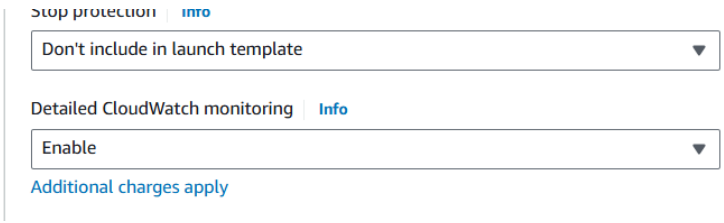
Creating Launch template for the Auto Scaling Group



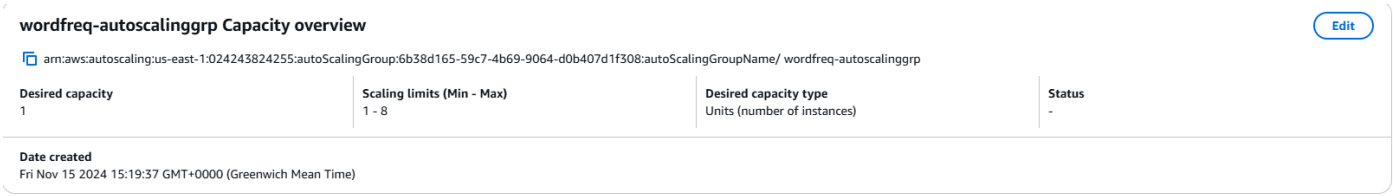




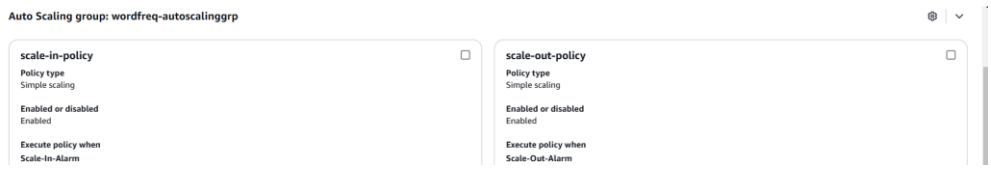




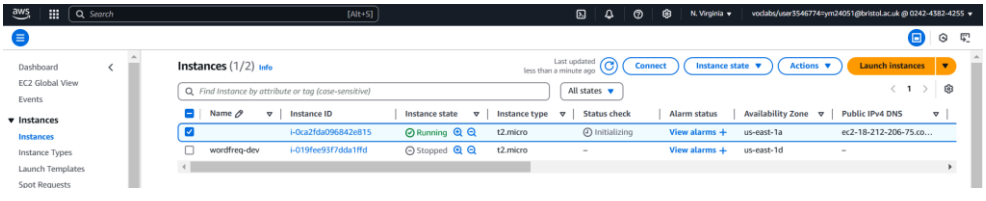
Created Auto-Scaling group configured :

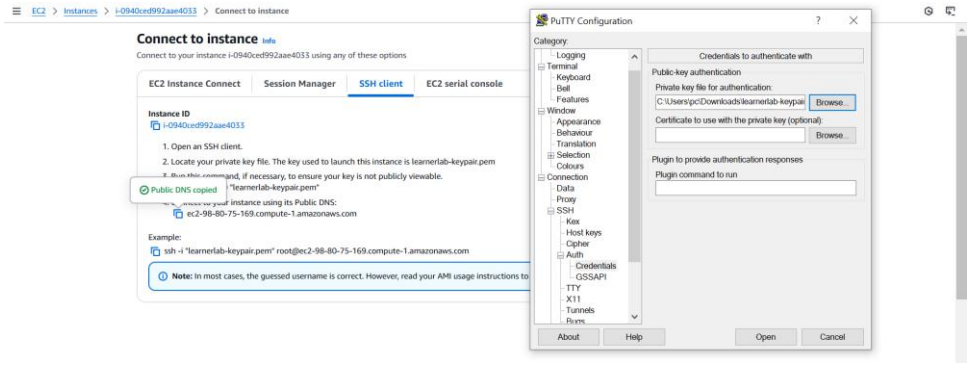


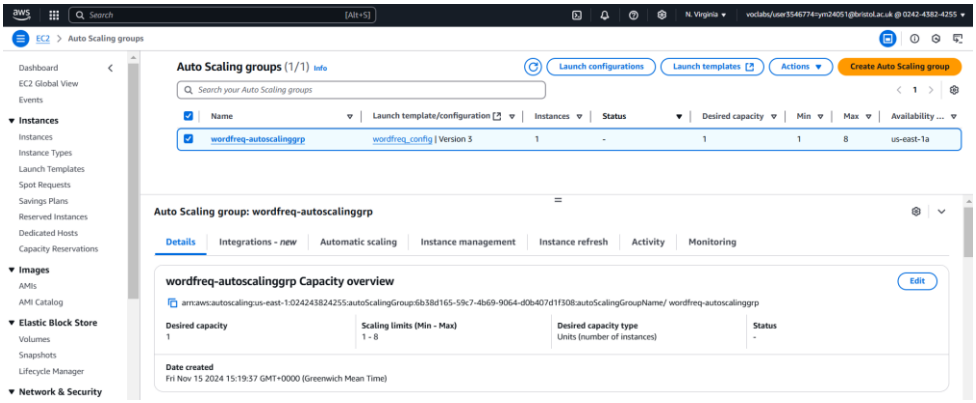
Auto-Scaling group policies set up:



Stopped the EC2 Instance and made the SSH connection using the Running instance of the Autoscaling group like below:



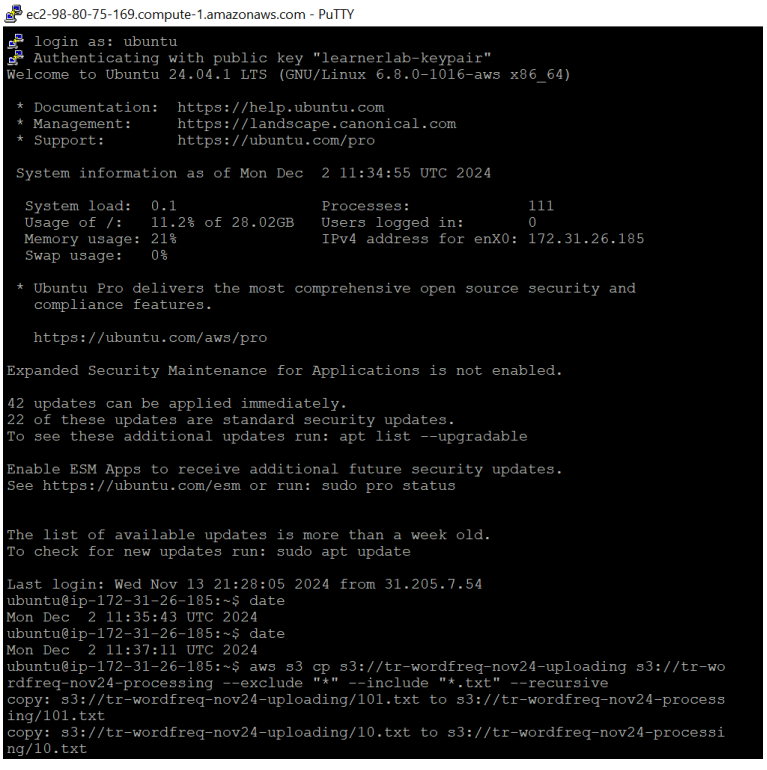




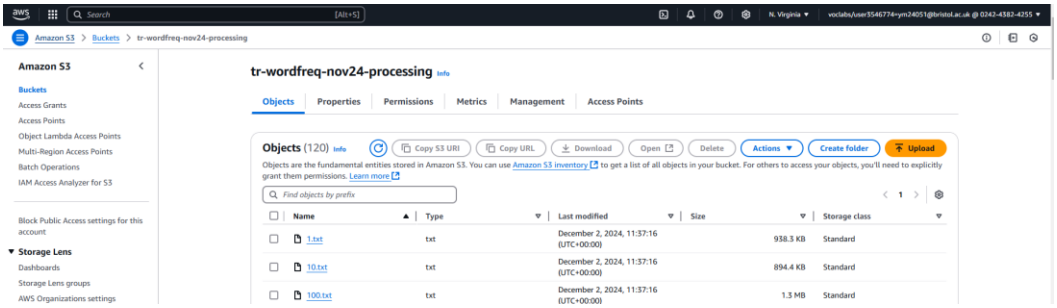
Connection Established:

SSH connection made with the running EC2 instance of ASG and then cp command to copy files from the uploading bucket to the Processing bucket using:

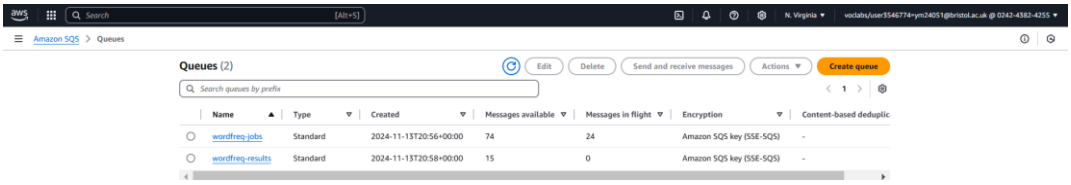
aws s3 cp s3://tr-wordfreq-nov24-uploading s3://tr-wordfreq-nov24-processing --exclude "*" --include "*.txt" –recursive

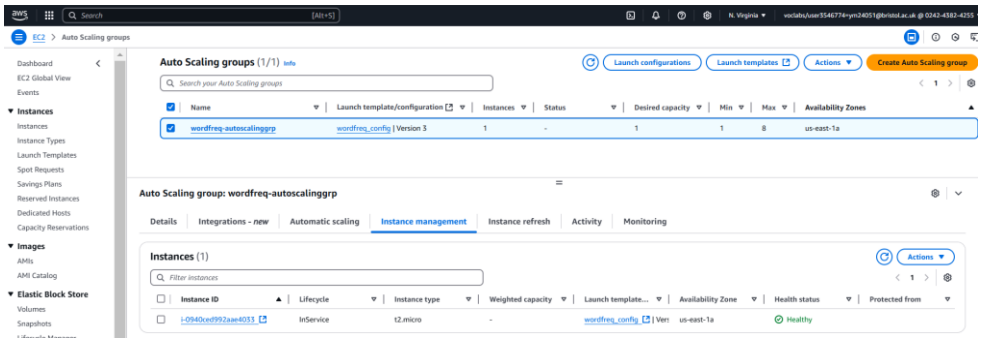


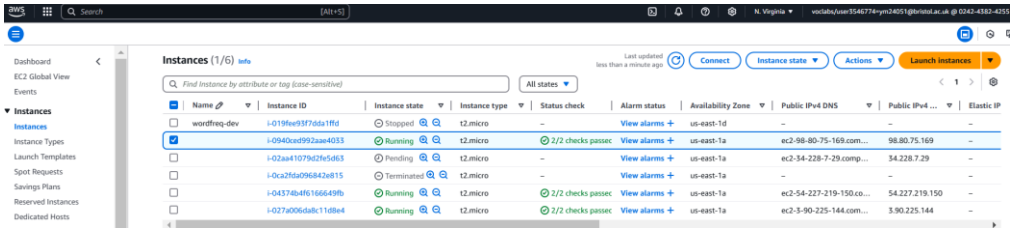
Processing bucket copies and holding the 120 Txt files:



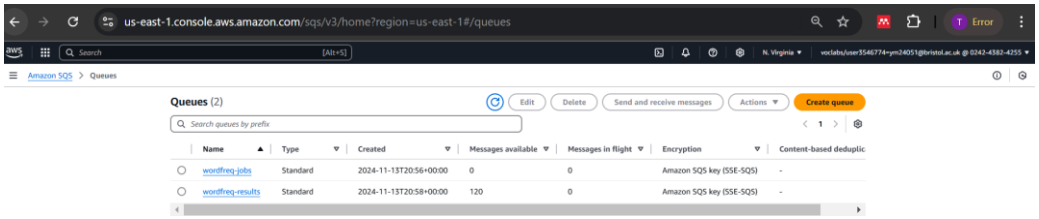
Queue starts to process

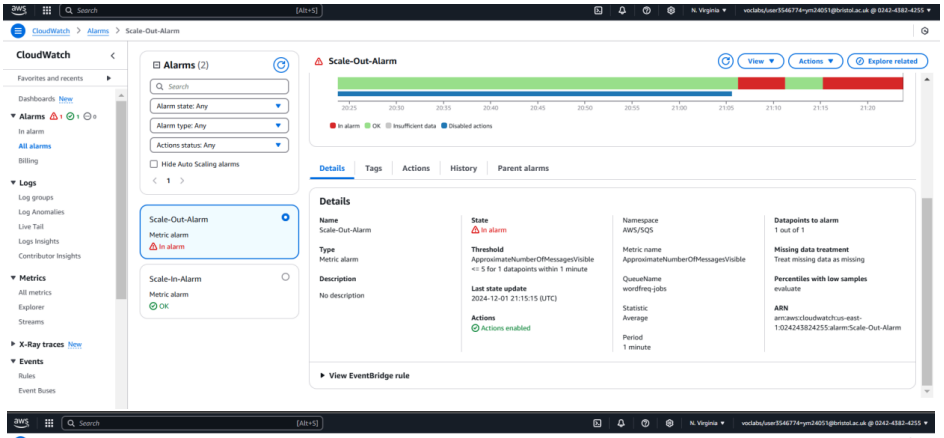


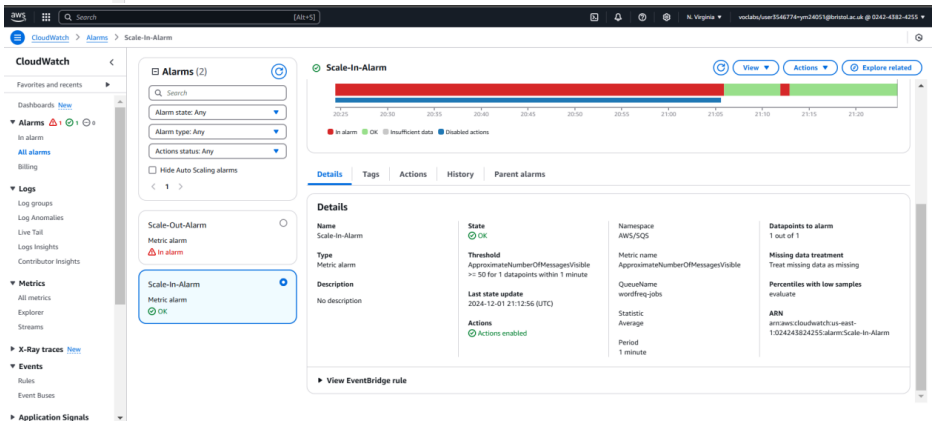


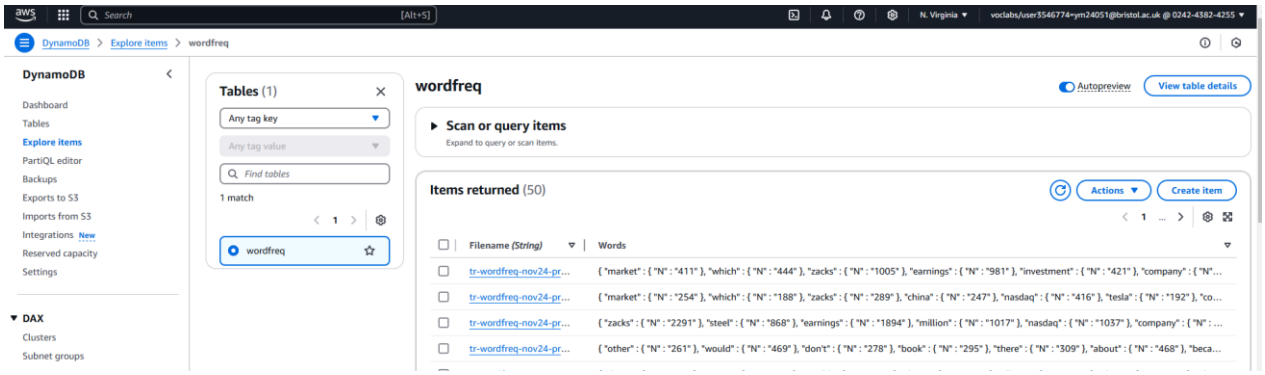


All txt files processed:









Start time: 11:37:11

End time: 11:50:01

Total Processing Time:
12 minutes and 50 seconds.

➔ using data command to check the time taken to process the .txt files

Task B		
EC2 Instance Type	t2.micro	
Simple Scaling Policy	Scale-Out-Policy	Add 1 capacity units
	Scale-In-Policy	Remove 1 capacity units
Scaling Metrics	SQS - ApproximateNumberOfMessagesVisible	
Desired Capacity	1	
Minimum Capacity	1	
Maximum Capacity	8	
Wait time	150 secs (2 mins)	
Cloudwatch Alarms Metrics	Scale-In-Alarm	Threshold: ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minute
	Scale-Out-Alarm	Threshold: ApproximateNumberOfMessagesVisible >= 50 for 1 datapoints within 1 minute
Default Cooldown	300 secs	
Scaling Policy	Simple Scaling Policy	
Start time	11:37:11	
End time	11:50:01	
Total Processing Time:	**12 minutes and 50 seconds**	

TASK C: Implementing and Testing Auto-Scaling Policies for Enhanced Processing Time

FIRST OPTIMIZATION ITERATION:

Desired capacity – 4

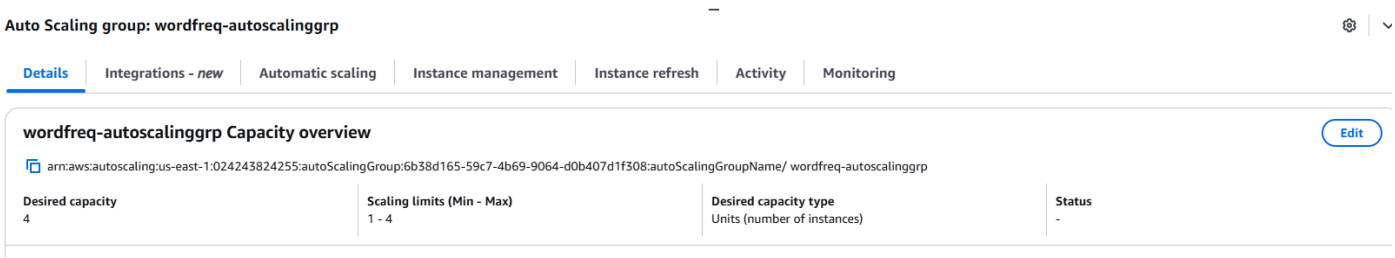
Minimum capacity – 1

Maximum – 4

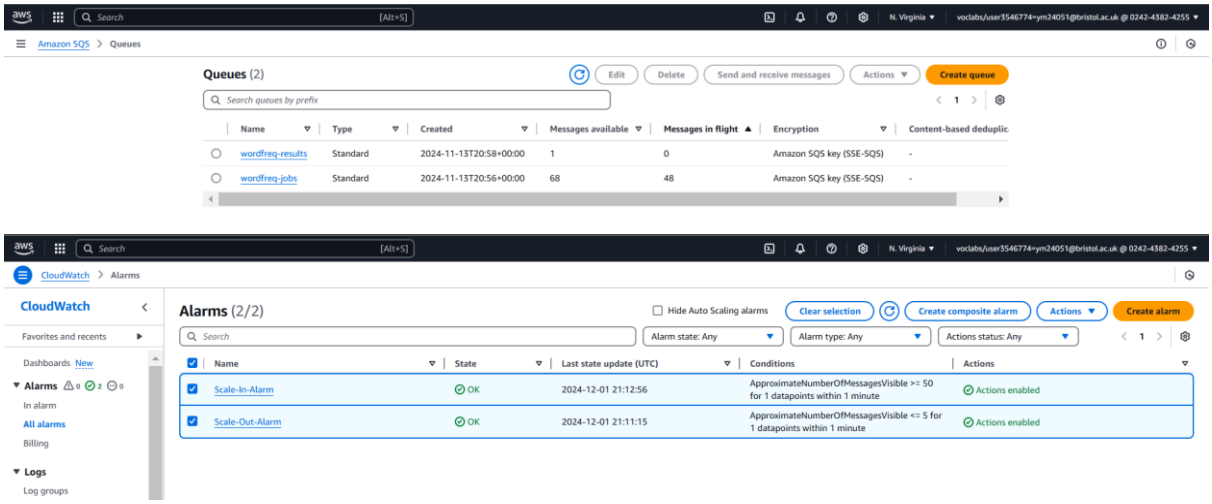
Below is task C's first optimization iteration, which focuses on balancing resource efficiency and processing speed. Initially, we designed the simple scaling policies to dynamically adjust the number of t2.micro instances based on the SQS queue's message load that ensures that the resources scale up during high demand and scale down during low demand, avoiding unnecessary costs.

By this optimization iteration, we reduced the processing time for the workload from *12 minutes 50 seconds to 8 minutes and 50 seconds* meeting the target. The dynamic scaling policies ensured faster task completion without over-provisioning or under-utilizing resources.

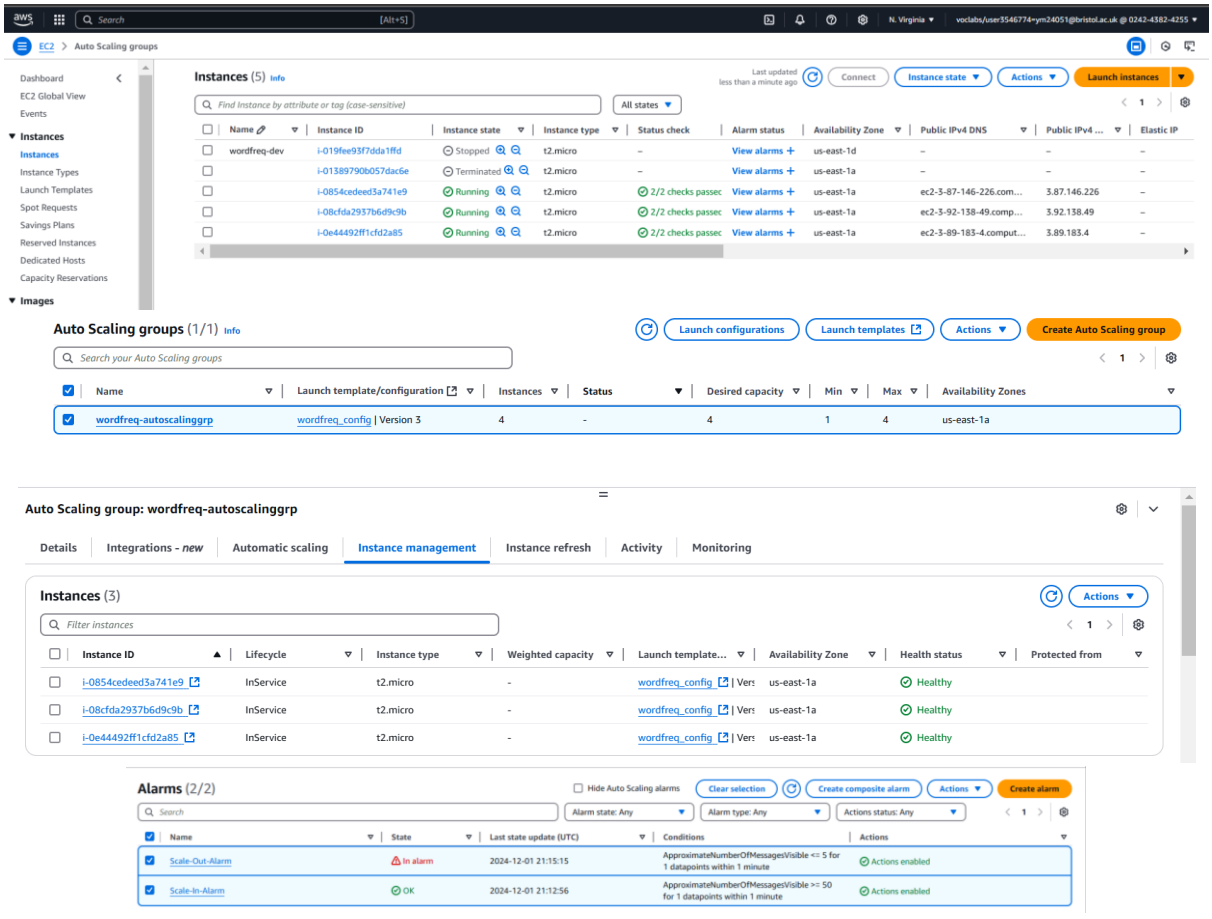
The configuration ensured a minimum of 1 instance for continuous processing, a maximum of 4 instances, and the desired capacity of 4 to handle peak workloads efficiently. A default cooldown of 300 seconds further stabilized scaling activities, preventing overly frequent adjustments.



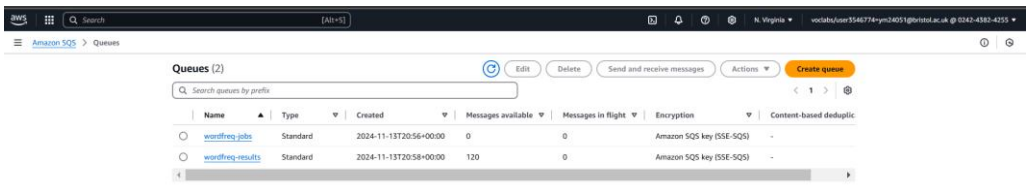
Below is the Processing Queues:



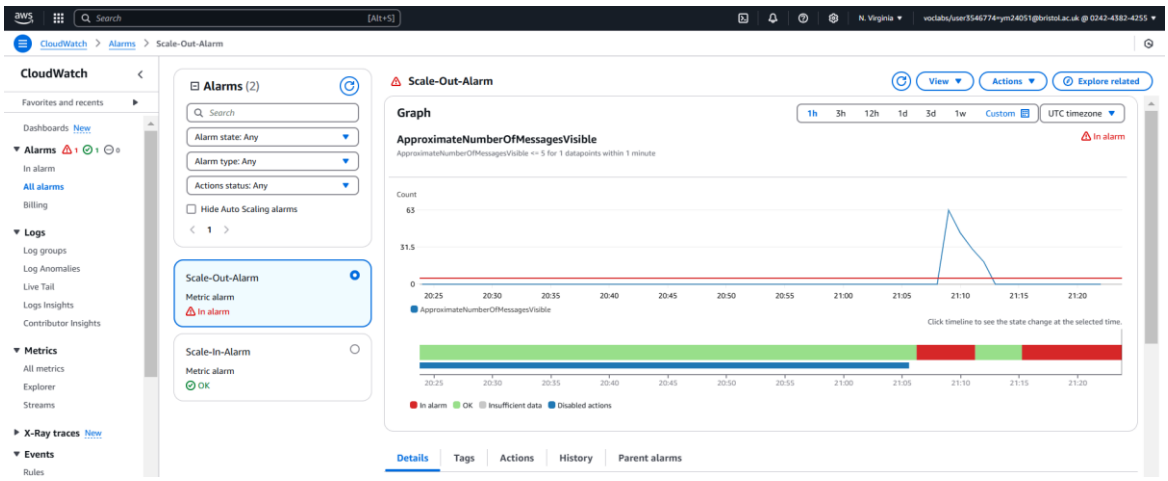
Instances of Auto Scaling group are running and established a connection using one of the running instances:

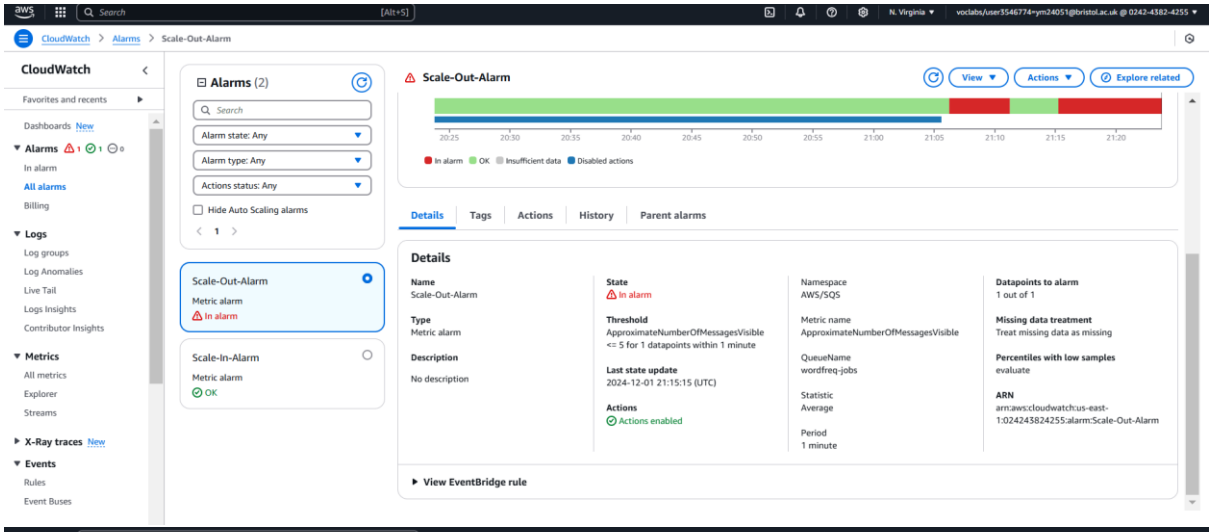


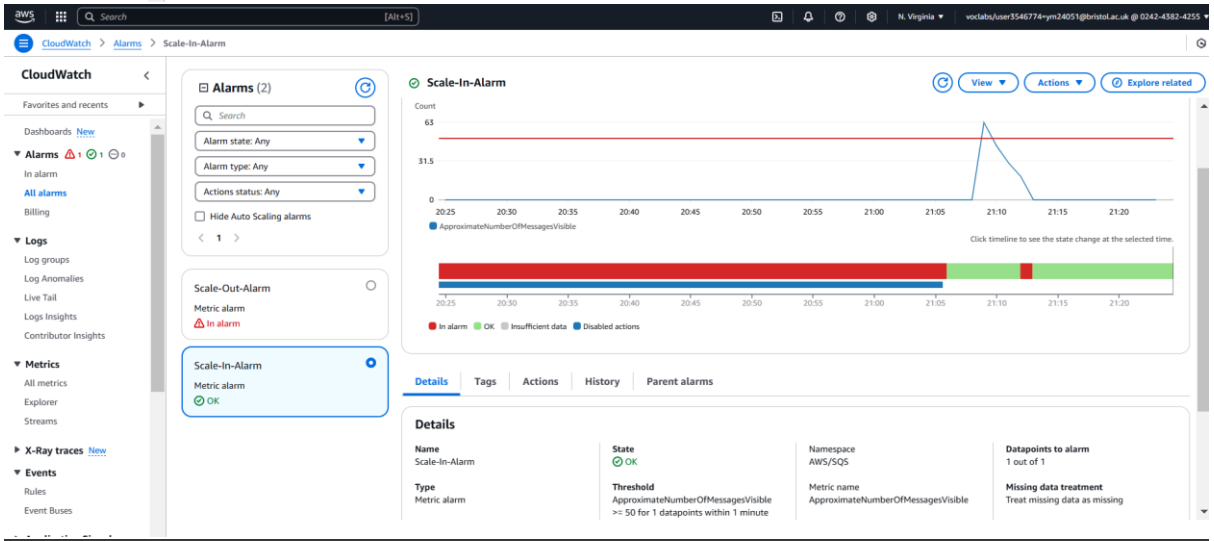
All the messages processed -

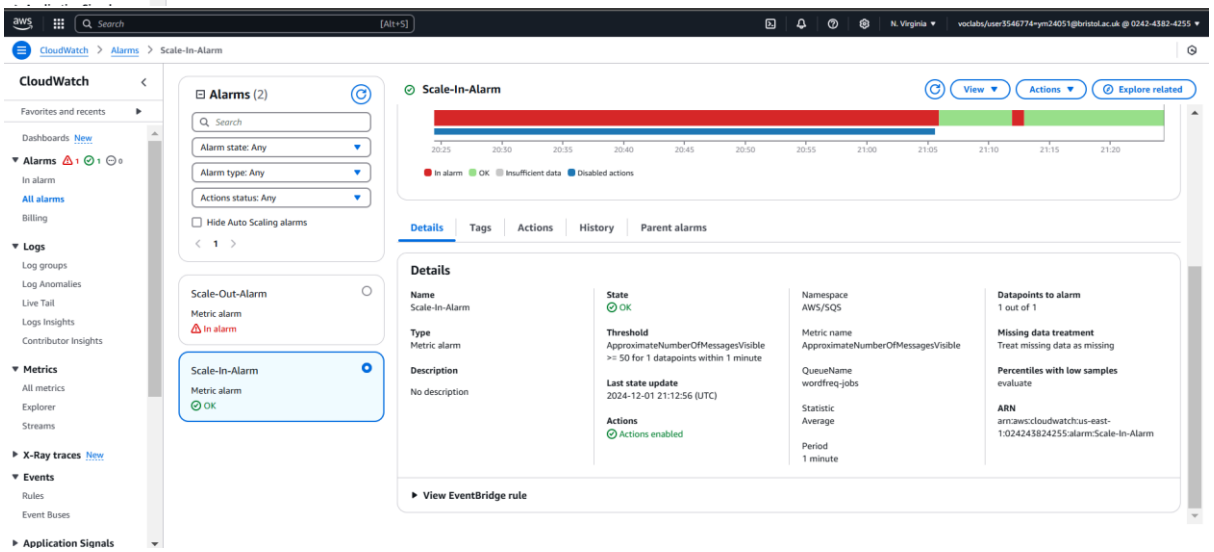


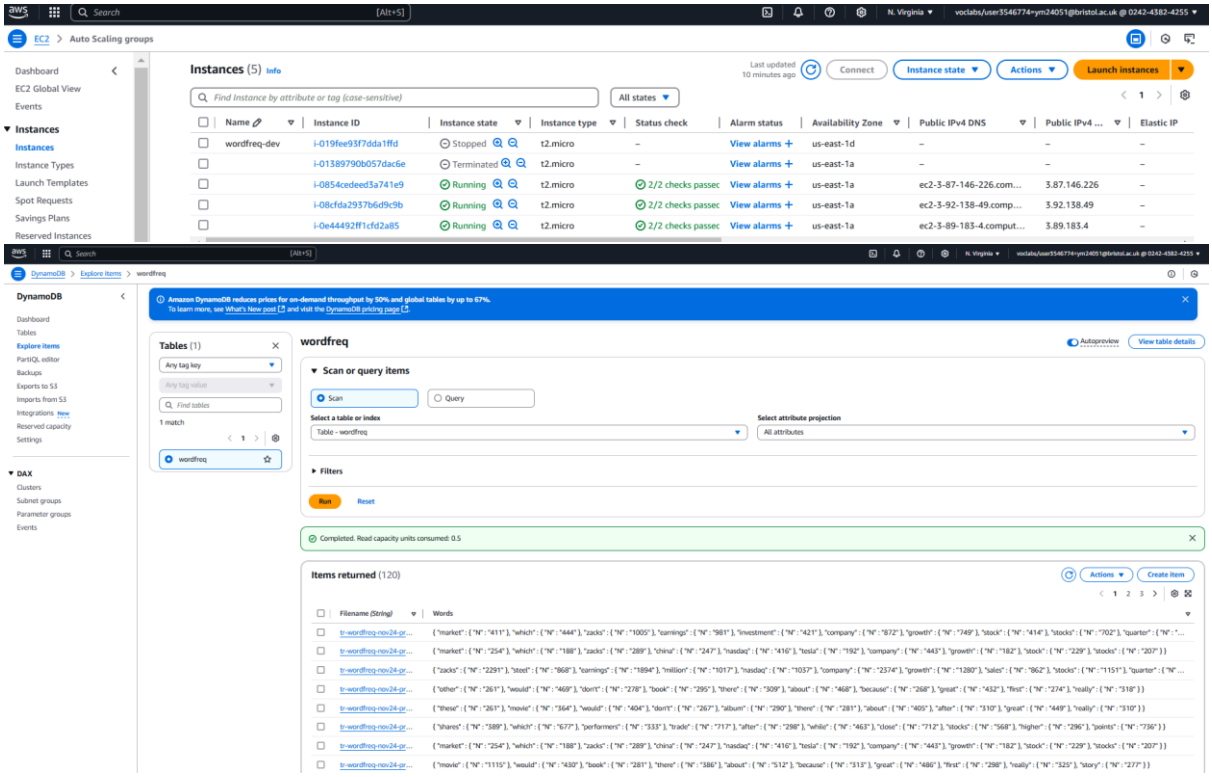
To optimize further we used CloudWatch alarms with clear thresholds using the metric - SQS - ApproximateNumberOfMessagesVisible: scales out (adds an instance) when the visible messages in the queue exceed 50 for 1 minute (Scale-Out-Alarm), ensuring prompt handling of high workloads. Similarly, it scales in (removes an instance) when visible messages drop below 5 for 1 minute (Scale-In-Alarm), preventing idle resources. The wait time between scaling actions was set to 150 seconds, reducing over-scaling or frequent scaling actions that might disrupt stability.




CloudWatch > Alarms > Scale-Out-Alarm
Alarms (2)
Scale-Out-Alarm
Metric alarm
In alarm
Scale-In-Alarm
Metric alarm
OK
Details
Tags
Actions
History
Parent alarms
Name
Scale-Out-Alarm
Type
Metric alarm
Description
No description
State
In alarm
Threshold
ApproximateNumberOfMessagesVisible
<= 5 for 1 datapoints within 1 minute
Last state update
2024-12-01 21:15:15 (UTC)
Actions
Actions enabled
Namespace
AWS/SQS
Metric name
ApproximateNumberOfMessagesVisible
QueueName
wordfreq-jobs
Statistic
Average
Period
1 minute
Datapoints to alarm
1 out of 1
Missing data treatment
Treat missing data as missing
Percentiles with low samples
evaluate
ARN
arn:aws:cloudwatch:us-east-1:024243824255:alarm:Scale-Out-Alarm
View EventBridge rule


CloudWatch > Alarms > Scale-In-Alarm
Alarms (2)
Scale-Out-Alarm
Metric alarm
In alarm
Scale-In-Alarm
Metric alarm
OK
Details
Tags
Actions
History
Parent alarms
Name
Scale-In-Alarm
Type
Metric alarm
State
OK
Threshold
ApproximateNumberOfMessagesVisible
>= 50 for 1 datapoints within 1 minute
Namespace
AWS/SQS
Metric name
ApproximateNumberOfMessagesVisible
Datapoints to alarm
1 out of 1
Missing data treatment
Treat missing data as missing


CloudWatch > Alarms > Scale-In-Alarm
Alarms (2)
Scale-Out-Alarm
Metric alarm
In alarm
Scale-In-Alarm
Metric alarm
OK
Details
Tags
Actions
History
Parent alarms
Name
Scale-In-Alarm
Type
Metric alarm
Description
No description
State
OK
Threshold
ApproximateNumberOfMessagesVisible
>= 50 for 1 datapoints within 1 minute
Last state update
2024-12-01 21:12:56 (UTC)
Actions
Actions enabled
Namespace
AWS/SQS
Metric name
ApproximateNumberOfMessagesVisible
QueueName
wordfreq-jobs
Statistic
Average
Period
1 minute
Datapoints to alarm
1 out of 1
Missing data treatment
Treat missing data as missing
Percentiles with low samples
evaluate
ARN
arn:aws:cloudwatch:us-east-1:024243824255:alarm:Scale-In-Alarm
View EventBridge rule



Below is the SSH Connection using PuTTY with the ASG's running instance and using "date" command to track the time:

```
ubuntu@ip-172-31-23-228:~$ date
Sun Dec  1 21:08:07 UTC 2024
ubuntu@ip-172-31-23-228:~$ aws s3 cp s3://tr-wordfreq-nov24-uploading s3://tr-wo
rdfreq-nov24-processing --exclude "*" --include "*.txt" --recursive
copy: s3://tr-wordfreq-nov24-uploading/1.txt to s3://tr-wordfreq-nov24-processin
g/1.txt
copy: s3://tr-wordfreq-nov24-uploading/104.txt to s3://tr-wordfreq-nov24-process
ing/104.txt
copy: s3://tr-wordfreq-nov24-uploading/10.txt to s3://tr-wordfreq-nov24-processi
ng/10.txt
copy: s3://tr-wordfreq-nov24-uploading/100.txt to s3://tr-wordfreq-nov24-process
ing/100.txt
copy: s3://tr-wordfreq-nov24-uploading/103.txt to s3://tr-wordfreq-nov24-process
ing/103.txt
copy: s3://tr-wordfreq-nov24-uploading/101.txt to s3://tr-wordfreq-nov24-process
ing/101.txt
copy: s3://tr-wordfreq-nov24-uploading/109.txt to s3://tr-wordfreq-nov24-process
ing/109.txt
copy: s3://tr-wordfreq-nov24-uploading/106.txt to s3://tr-wordfreq-nov24-process
ing/106.txt
copy: s3://tr-wordfreq-nov24-uploading/110.txt to s3://tr-wordfreq-nov24-process
ing/110.txt
copy: s3://tr-wordfreq-nov24-uploading/102.txt to s3://tr-wordfreq-nov24-process
ing/102.txt
copy: s3://tr-wordfreq-nov24-uploading/105.txt to s3://tr-wordfreq-nov24-process
ing/105.txt
copy: s3://tr-wordfreq-nov24-uploading/11.txt to s3://tr-wordfreq-nov24-processi
ng/11.txt
copy: s3://tr-wordfreq-nov24-uploading/93.txt to s3://tr-wordfreq-nov24-processi
ng/93.txt
copy: s3://tr-wordfreq-nov24-uploading/95.txt to s3://tr-wordfreq-nov24-processi
ng/95.txt
copy: s3://tr-wordfreq-nov24-uploading/97.txt to s3://tr-wordfreq-nov24-processi
ng/97.txt
copy: s3://tr-wordfreq-nov24-uploading/98.txt to s3://tr-wordfreq-nov24-processi
ng/98.txt
copy: s3://tr-wordfreq-nov24-uploading/99.txt to s3://tr-wordfreq-nov24-processi
ng/99.txt
```

- **Start Time:** Sun Dec 1 21:08:07 UTC 2024
- **End Time:** Sun Dec 1 21:16:57UTC 2024

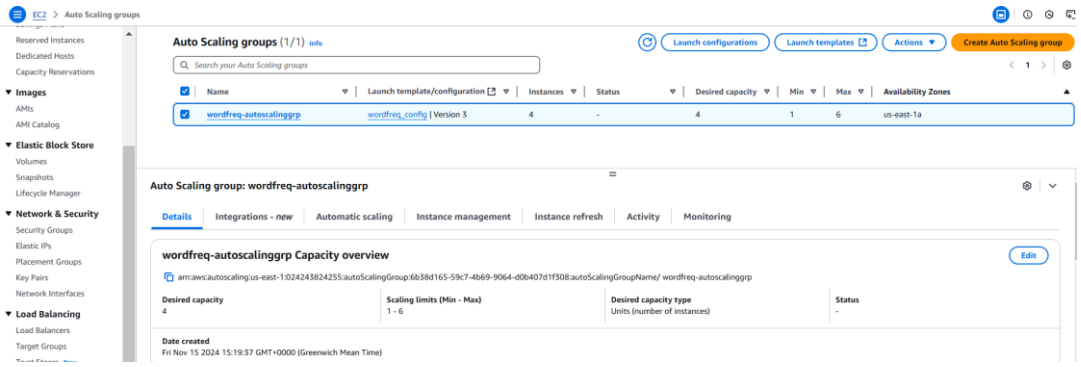
Total Processing Time:

- **8 minutes and 50 seconds.**

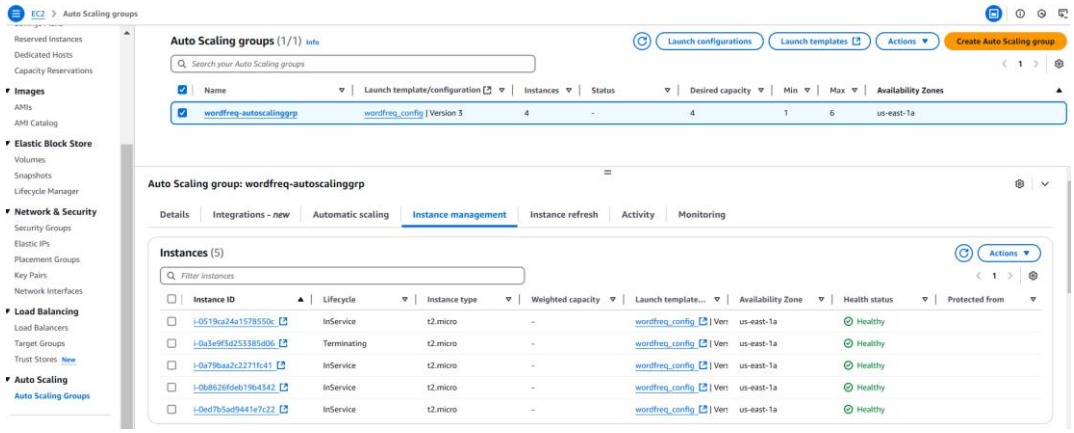
Task C - 1st Optimization		
EC2 Instance Type	t2.micro	
Simple Scaling Policy	Scale-Out-Policy	Add 1 capacity units
	Scale-In-Policy	Remove 1 capacity units
Scaling Metrics	SQS - ApproximateNumberOfMessagesVisible	
Desired Capacity	4	
Minimum Capacity	1	
Maximum Capacity	4	
Wait time	150 secs (2 mins 30 secs)	
Cloudwatch Metrics	Scale-In-Alarm	Threshold: ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minute
	Scale-Out-Alarm	Threshold: ApproximateNumberOfMessagesVisible >= 50 for 1 datapoints within 1 minute
Default Cooldown	300 secs	
Scaling Policy	Simple Scaling Policy	
Start time	21:08:07	
End time	21:16:57	
Total Processing Time:	**8 minutes and 50 seconds**	

SECOND OPTIMIZATION ITERATION:

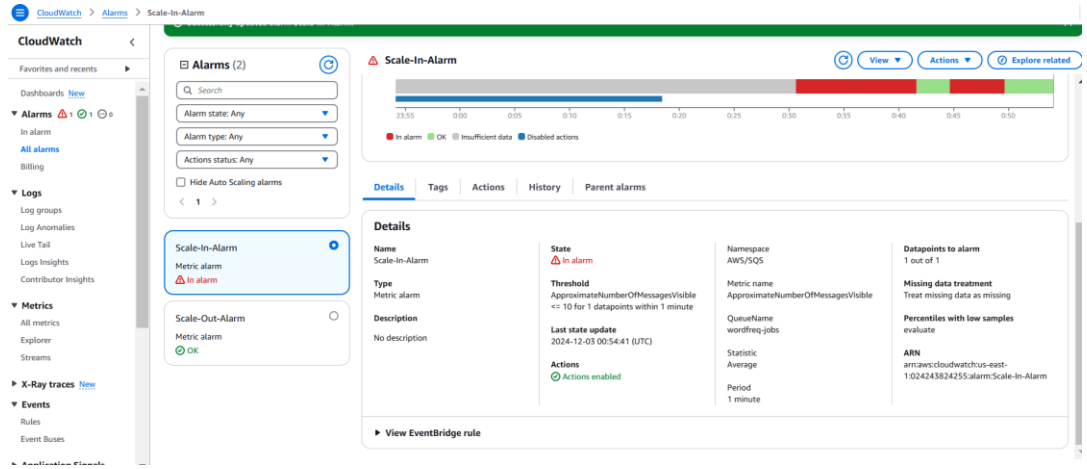
In this iteration, we will be using the t2.micro instance type and change the values of desired capacity to 4, minimum capacity to 1, and maximum to 6, and the threshold values ApproximateNumberOfMessagesVisible <= 10 for 1 datapoint within 1 minute for Scale-In Alarm and Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoint within 1 minute for Scale-Out Alarm used the same SQS – ApproximateNumberOfMessagesVisible and Simple Scaling policies.

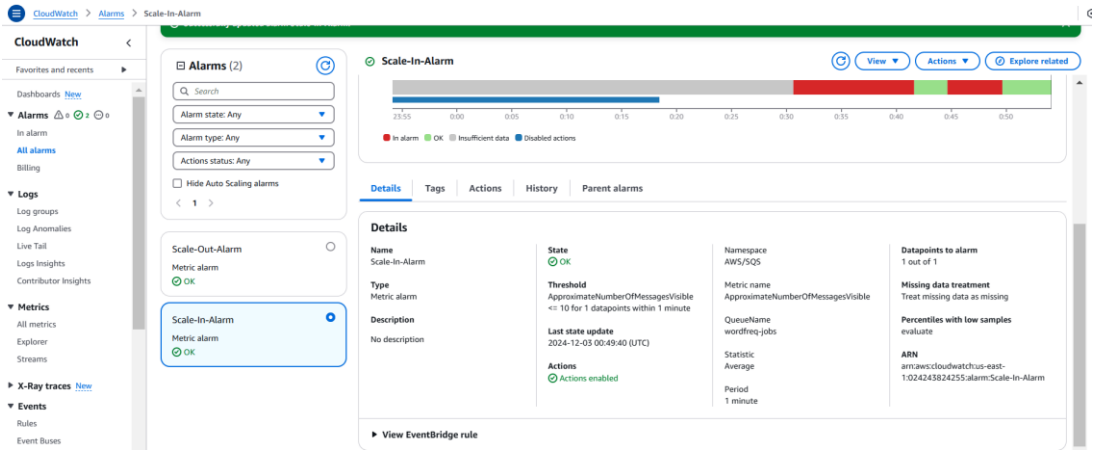


As we can see below the instances of the ASG group:



Cloudwatch Setting:





Using PuTTY for SSH connection with the running instance of the Automatic Scaling Groups

PuTTY (inactive)

```
login as: ubuntu
Authenticating with public key "learnerlab-keypair"
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1016-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Tue Dec  3 00:46:39 UTC 2024

  System load:  0.0               Processes:             107
  Usage of /:   12.5% of 28.02GB  Users logged in:       0
  Memory usage: 23%               IPv4 address for enX0: 172.31.16.243
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

70 updates can be applied immediately.
36 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Tue Dec  3 00:36:13 2024 from 31.205.7.54
ubuntu@ip-172-31-16-243:~$ date
Tue Dec  3 00:46:43 UTC 2024
ubuntu@ip-172-31-16-243:~$ date
Tue Dec  3 00:47:02 UTC 2024
ubuntu@ip-172-31-16-243:~$ aws s3 cp s3://tr-wordfreq-nov24-uploading s3://tr-wo
rdfreq-nov24-processing --exclude "*" --include "*.txt" --recursive
Completed 452.3 KiB/~91.7 MiB (1.8 MiB/s) with ~99 file(s) remaining (calculatin
copy: s3://tr-wordfreq-nov24-uploading/101.txt to s3://tr-wordfreq-nov24-process
ing/101.txt
Completed 452.3 KiB/~91.7 MiB (1.8 MiB/s) with ~98 file(s) remaining (calculatin
Completed 1.3 MiB/112.1 MiB (4.3 MiB/s) with 119 file(s) remaining
copy: s3://tr-wordfreq-nov24-uploading/10.txt to s3://tr-wordfreq-nov24-processi
ng/10.txt
```

- **Start Time: Tue Dec 3 00:47:02 UTC 2024**
- **End Time: Tue Dec 3 00:53:45 UTC 2024**

Total Processing Time:

6 minutes and 43 seconds

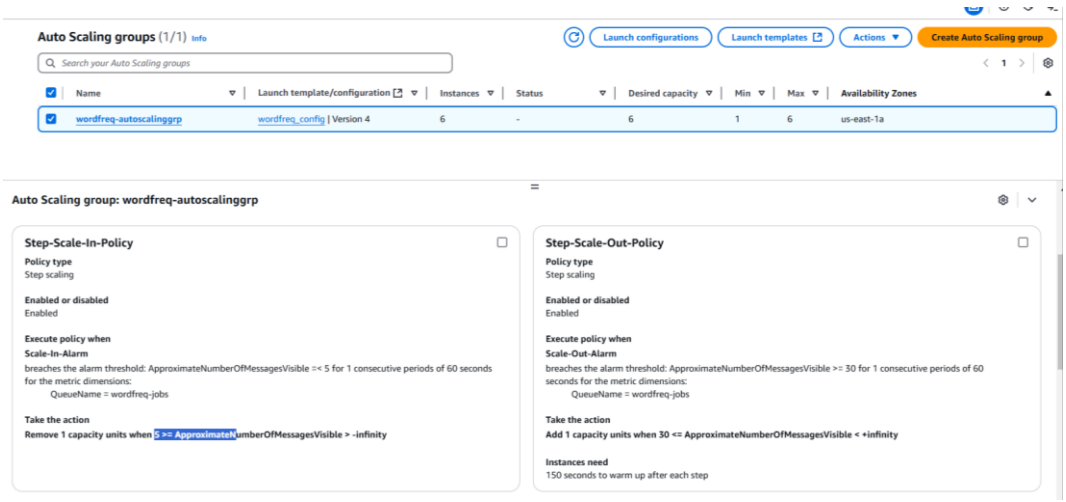
Task C - 2nd Optimization		
EC2 Instance Type	t2.micro	
Simple Scaling Policy	Scale-Out-Policy	Add 1 capacity units
	Scale-In-Policy	Remove 1 capacity units
Scaling Metrics	SQS - ApproximateNumberOfMessagesVisible	
Desired Capacity	4	
Minimum Capacity	1	
Maximum Capacity	6	
Wait time	150 secs (2 mins 30 secs)	
Cloudwatch Metrics	Scale-In-Alarm	Threshold: ApproximateNumberOfMessagesVisible <= 10 for 1 datapoints within 1 minute
	Scale-Out-Alarm	Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoints within 1 minute
Default Cooldown	300 secs	
Scaling Policy	Simple Scaling Policy	
Start time	00:47:02	
End time	00:53:45	
Total Processing Time:	**6 minutes and 43 seconds.**	

With the second iteration, we have successfully reduced the processing time by 2 minutes as the last optimization iteration took 8 minutes 50 seconds and the current iteration takes a total processing time of - 6 minutes and 43 seconds.

The reduction in the processing time of the second optimization is because of the more aggressive scaling policy and higher maximum capacity. Lowering the threshold for the Scale-Out-Alarm (from 50 to 30 visible messages) and raising the Maximum Capacity from 4 to 6 instances make the system respond faster to increased load by scaling out more instances earlier. Besides, Scale-In-Alarm is set higher, at 10 visible messages from 5, in order to keep more instances up and running until the queue size is radically reduced. This will continue to drive efficiency in processing. Such adjustments allow the system to process tasks faster, reducing the overall time it takes to do so.

THIRD OPTIMIZATION ITERATION:

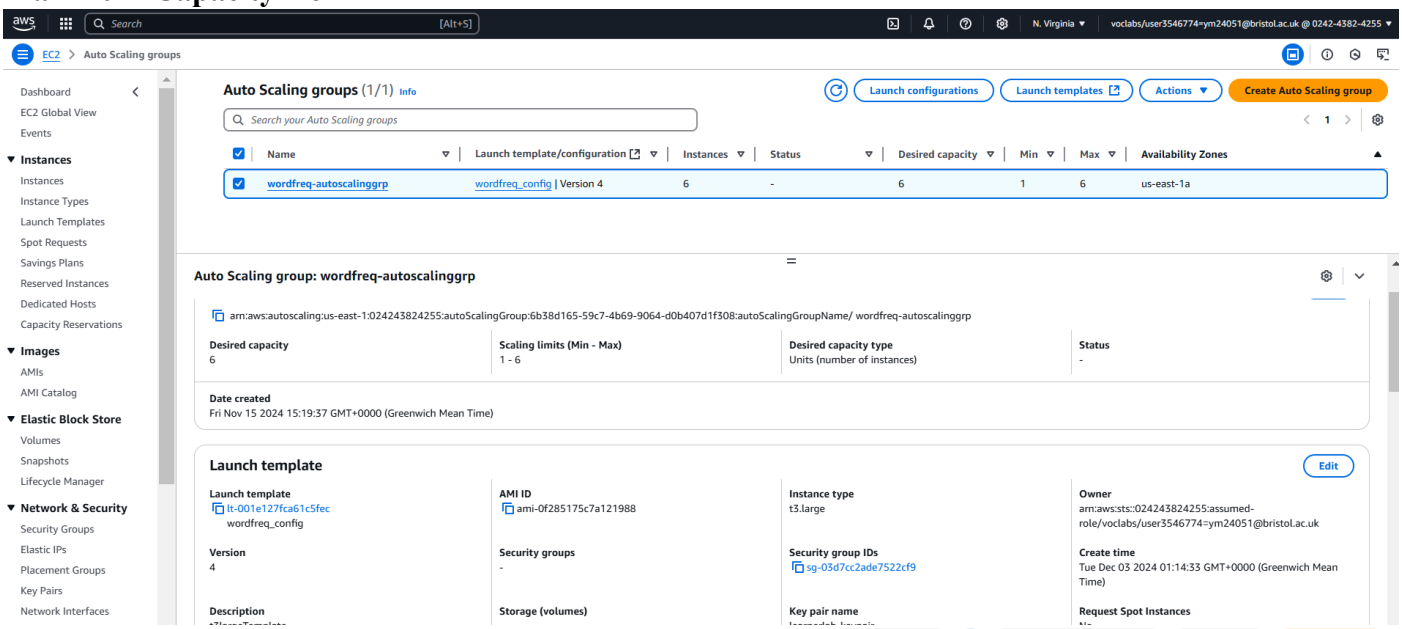
For the Third Optimization, we used a step scaling policy and launched a new autoscaling template for the t3.large instance type. We used SQS—ApproximateNumberOfMessagesVisible and Scale-In Alram Threshold: ApproximateNumberOfMessagesVisible = 5 for 1 datapoint within 1 minute and Scale-out Alarm Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoints within 1 minute.

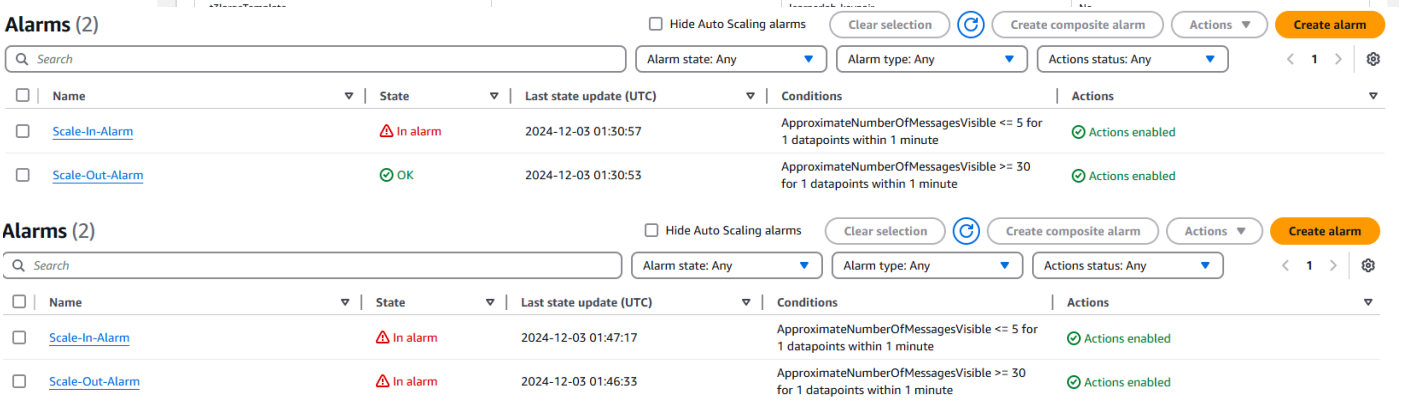


Desired Capacity – 6

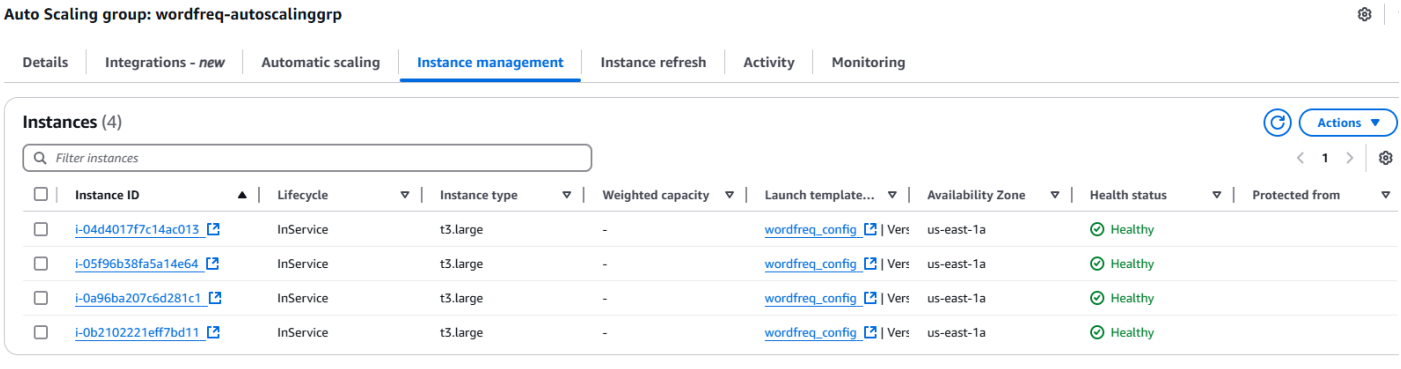
Minimum Capacity -1

Maximum Capacity – 6

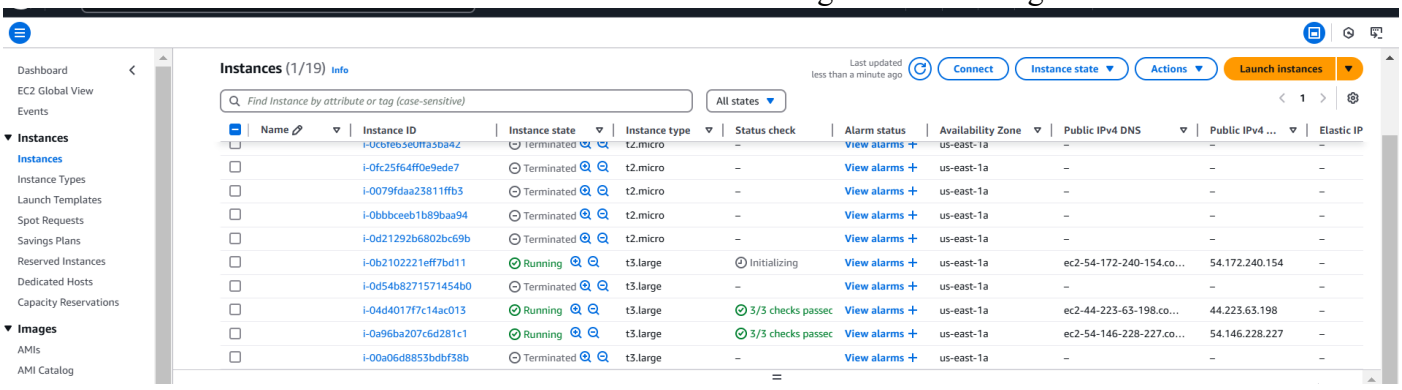




We can see the t3.large instances launched as below:



We can see the ASG instances running and terminating :



- **Start Time:** 01:43:00
- **End Time:** 01:47:34

Total Processing Time:

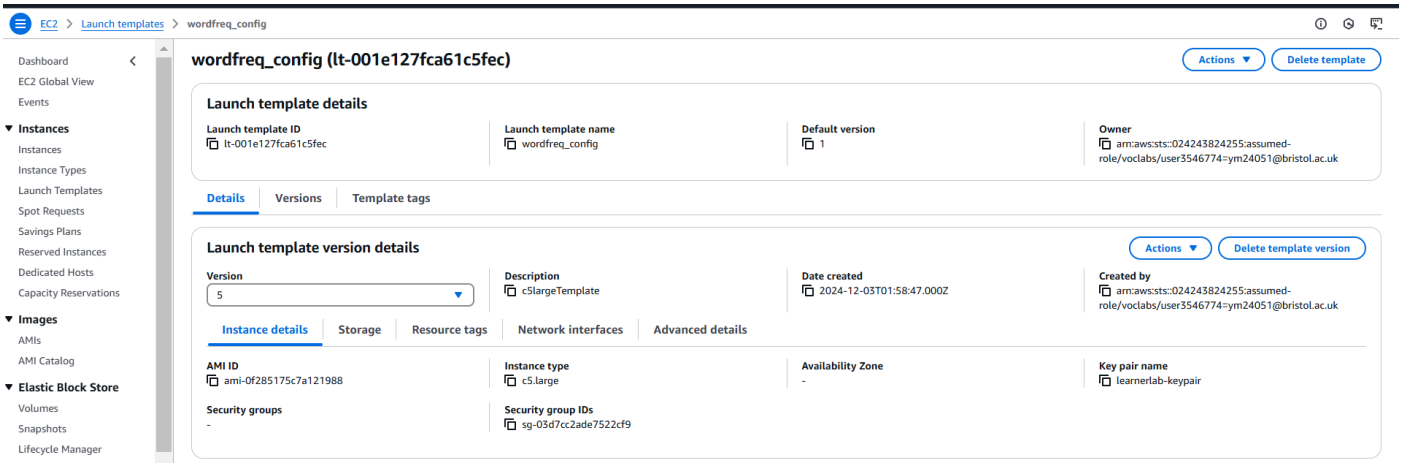
4 minutes and 34 seconds.

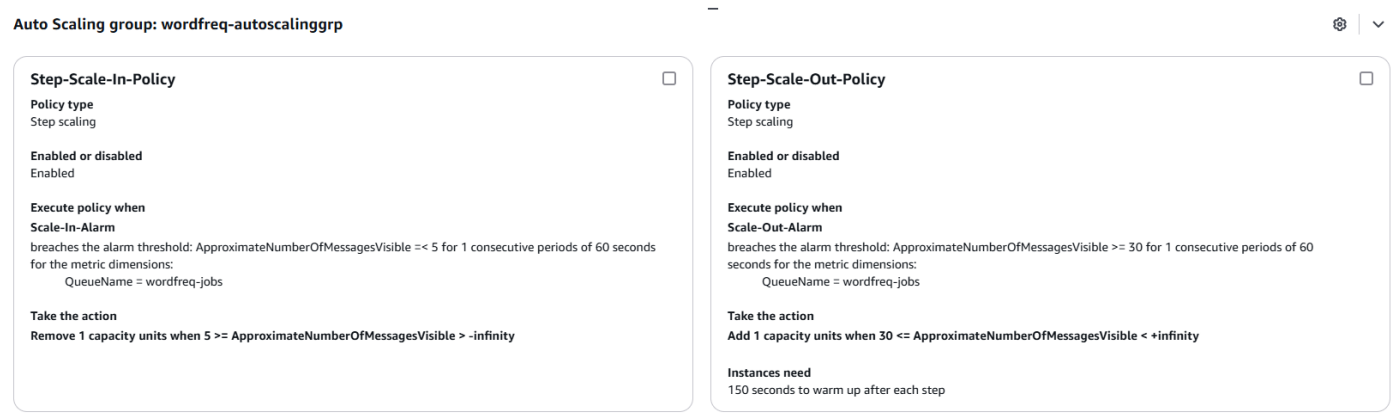
Task C - 3rd Optimization		
EC2 Instance Type	t3.large	
Step Scaling Policy	Scale-Out-Policy	Add 1 capacity units
	Scale-In-Policy	Remove 1 capacity units
Scaling Metrics	SQS - ApproximateNumberOfMessagesVisible	
Desired Capacity	6	
Minimum Capacity	1	
Maximum Capacity	6	
Wait time	150 secs (2 mins 30 secs)	
Cloudwatch Metrics	Scale-In-Alarm	Threshold: ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minute
	Scale-Out-Alarm	Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoints within 1 minute
Default Cooldown	300 secs	
Scaling Policy	Step Scaling Policy	
Start time	01:43:00	
End time	01:47:34	
Total Processing Time:	**4 minutes and 34 seconds.**	

The reduction in processing time in the third optimization is primarily due to the use of t3.large instances, which are much more powerful in both computation and networking compared with t2.micro instances. Also, the Step Scaling Policy will enable more granularity in scaling adjustments based on the size of the queue, enabling the system to handle the load more smoothly. With a higher Desired Capacity of 6 instances and maintaining scaling thresholds as per the second optimization, t3-large instances process messages faster, decreasing overall processing time. The result is due to a combination of stronger instance types and much finer scaling.

FOURTH OPTIMIZATION ITERATION:

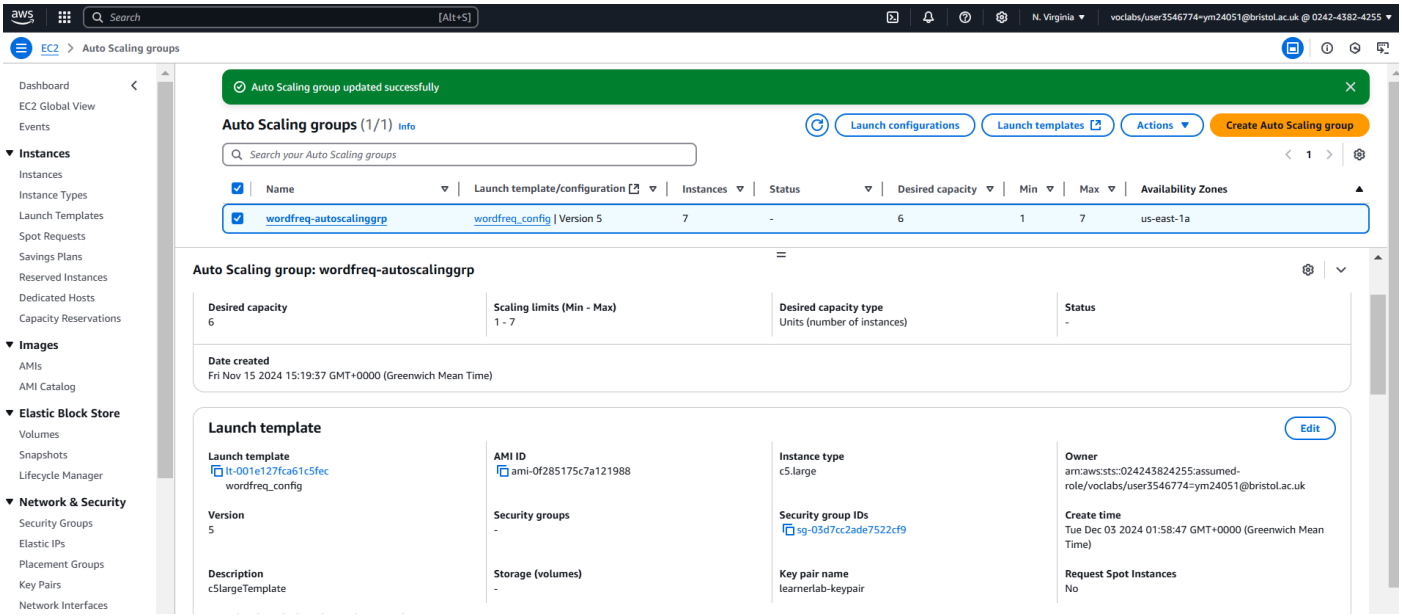
Created another Launch template for the Auto Scaling group for the c5.large instance Type as below:



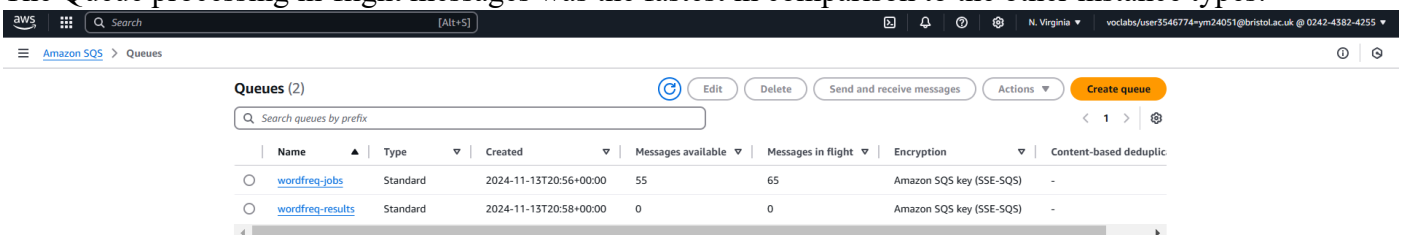


Used Step scaling policies - Step- Scale-Out Policy and Step-Scale -In Policy and Threshold: ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minute and Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoints within 1 minutes.

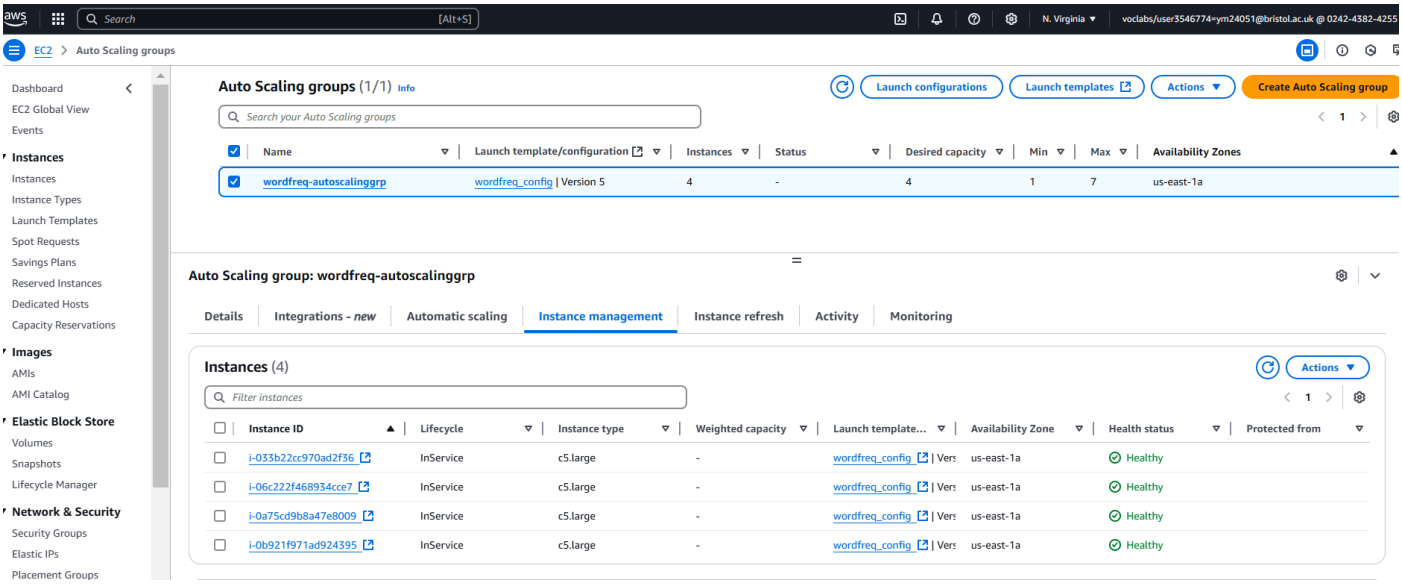
As we can see below:



The Queue processing in-flight messages was the fastest in comparison to the other instance types:



Below we can observe all the ASG instances running and healthy checks.



- **Start Time: 02:16:55**
- **End Time: 02:20:33**

Totat Processing time:

3 minutes and 38 seconds.

Task C - 4th Optimization		
EC2 Instance Type	c5.large	
Simple Scaling Policy	Scale-Out-Policy	Add 1 capacity units
	Scale-In-Policy	Remove 1 capacity units
Scaling Metrics	SQS - ApproximateNumberOfMessagesVisible	
Desired Capacity	7	
Minimum Capacity	1	
Maximum Capacity	7	
Wait time	150 secs (2 mins 30 secs)	
Cloudwatch Metrics	Scale-In-Alarm	Threshold: ApproximateNumberOfMessagesVisible <= 5 for 1 datapoints within 1 minute
	Scale-Out-Alarm	Threshold: ApproximateNumberOfMessagesVisible >= 30 for 1 datapoints within 1 minute
Default Cooldown	300 secs	
Scaling Policy	Step Scaling Policy	
Start time	02:16:55	
End time	02:20:33	
Total Processing Time:	**3 minutes and 38 seconds.**	

The fourth optimization has reduced the processing time because it uses c5.large instances, which are compute-optimized, and their CPU is far better compared to t3.large instances. In addition, Desired Capacity is increased to 7 instances, enabling the system to process more tasks concurrently. Although both optimizations use similar scaling policies and thresholds, the higher computational power and increased instance count that the fourth optimization allows for enable messages in the SQS queue to be processed sooner, hence the observed decrease in total processing time.

CONCLUSION:

Scaling Policy Optimization: Step scaling, besides being dynamically adjusted to capacity based on SQS metrics, can ensure efficient use of resources at each stage to make sure the task can be completed faster without over-provisioning.

Instance Type Selection: t2.micro to t3.large and finally c5.large played an important role in enhancing the processing times, especially for compute-intensive tasks.

Threshold Adjustments: Fine-tuning the thresholds for scaling, such as adjusting ApproximateNumberOfMessagesVisible for scaling out, making sure that the system scales appropriately based on the workload, reducing unnecessary scaling, and optimizing resource usage.

The optimizations progressively reduced the processing time by enhancing resource allocation, choosing the right instance types, and fine-tuning scaling policies, which resulted in the fastest processing time of 3 minutes and 38 seconds with the c5.large instance.

TASK D - Optimise the WordFreq Architecture

In the revised architecture, we have inculcated the AWS services from the storage, compute, databases, and security categories as per the LSDE course to optimize effectively and efficiently for Increasing resilience and availability of the application against component failure, Long-term backing up of valuable data, Cost-effective and efficient application for occasional use. Processing does not need to be immediate and to Prevent unauthorized access.

In the revised architecture of the WordFreq Application's cloud Architecture we have used the following AWS Services and compared the uses of the Architecture before and after the changes as below:

In the redesigned Architecture we have included the following modifications and changes:

Responsibilities of AWS Services	Previous Architecture	New Architecture (Revised)
Resilience and Availability	Limited and generally confined to one Availability Zone.	The instances of EC2 and SQS queues should be deployed across multiple Availability Zones for redundancy.
		Cross-Region Replication (CRR) for S3 buckets ensures that data will be available across regions.
Scaling	Limited auto-scaling of the EC2 instances; manual intervention is required.	Auto Scaling Group for multi-AZ-based EC2 instances scales dynamically depending on traffic and queue load.
Compute Service	Most of the file processing is done on the EC2 instances.	**AWS Lambda** for event-driven file processing.
		For heavier workloads, utilize the instances from the Auto Scaling of EC2.
Cost Efficiency	More expensive because On-Demand EC2 instances are used for processing.	EC2 Spot instances for cost savings on non-critical processing.
		AWS Lambda for occasional and event-driven workloads, being priced per invocation.
Backup and Long-Term Storage	Basic backup of S3 and EC2 instances; no automated long-term	**S3 Glacier** for long-term storage of processed files at a lower cost.
		AWS Backup for centralized, automated backups of **DynamoDB**, S3, and **EC2**.
Data Durability	Version-enabled S3 buckets; no cross-region replication.	S3 buckets that make use of **Cross-Region Replication** (**CRR**) for disaster recovery.
		Increased durability through the usage of DynamoDB Multi-AZ Replication.
Security	IAM roles and security groups for basic configuration of EC2 instances.	IAM fine-grained roles to facilitate least-privilege access.
		KMS for encryption at rest in S3 and DynamoDB.
		CloudTrail and **CloudWatch** for monitoring and auditing.
Unauthorized Access Prevention	Limited encryption for data at rest.	**KMS** for encryption at rest in S3 and **DynamoDB**.
		VPC Security Groups to provide an additional, more detailed control of network traffic.
		Fine-grained IAM access.
Networking	Simple networking: basic VPC configurations, for example.	**VPC** with **multiple subnets** across AZs for added fault tolerance.
		Internet Gateway for public access and restricted security settings.
Disaster Recovery	No automated cross-region backup strategy.	**Cross-Region Replication (CRR) in S3 for the purpose of regional failure of data replication**
Queue Management	Simple SQS setup; no DLQ.	Failed message handling using Dead Letter Queues (DLQs) for Upload Queue and Result Queue.
Event-Driven Processing	Manually process the uploaded file. No serverless design.	S3 Event Triggers: enabling automatic invocation of Lambda functions upon uploading new files.
Monitoring and Logging	Basic logging via CloudWatch	Improved logging using CloudTrail - Auditing and CloudWatch - Real-time Resource Usage and Performance Monitoring.

How This Structure Satisfies the Design Objectives:

1. Increase Resilience and Availability:

- Multi-AZ Deployment: EC2 instances in the Auto Scaling Group across multiple subnets in different Availability Zones (AZs) make sure the application can keep running even if one AZ or instance fails.

- The Auto Scaling feature keeps the number of EC2 instances within a certain range. It increases or decreases the number of instances based on the current demand for your application. Also, if there are messages waiting in the message queue, it makes sure they are processed right away.
- Making sure that data is copied across different AWS Regions means that data is safe and can be accessed from different places if a regional outage happens. Users in the affected Region won't be able to access data if the Region goes down, so making sure the data is available to them at other AWS facilities is really important.

2. Long-Term Backups of Valuable Data:

- AWS Backup: AWS Backup simplifies the backup process for your S3 and DynamoDB data. All your data is safely stored in a central place. You can automatically create backups and easily find them when you need to recover.
- S3 Glacier: We put files in S3 Glacier for the long-term backups and cheap storage it gives us. We have Lifecycle Policies that automatically move finished data to this cheaper place.
- When we use S3 Lifecycle Policies, moving data to S3 Glacier, which is a more affordable long-term storage option, happens automatically as cost-effectively as possible.

3. Cost-effective and Efficient Application for Occasional Use:

- Lambda: Lambda functions only runs the EC2 instances in the ASG when needed (driven by events), so it's really efficient and doesn't waste money or computer power.
- EC2 Spot Instances: You can use these for not very important, temporary computing tasks. Using them can save you money because you're using EC2 capacity that isn't being used at a lower price than other EC2 options.
- Auto Scaling: It only provides the resources when they are needed and reduces the resources when the demand is less. This helps control costs.

4. Prevent Unauthorized Access:

- IAM policies, specifically IAM roles, make sure that the principle of least privilege is used on all resources. We create EC2, Lambda, and SQS roles with tightly controlled permissions to minimize risk. We also create S3 bucket policies to further restrict access. Only a small number of authorized users can access a small number of resources.
- KMS encrypts data stored in S3 and DynamoDB. These services are part of AWS. This keeps but ensures that good stuff is safe. Whether it's just files (like Word documents or txt files) in S3, or big databases running on DynamoDB, KMS can protect them all.
- VPC Security Groups: Set up rules for allowed and blocked traffic. These rules control access to EC2 instances and other resources, stopping unwanted connections.
- Directs traffic to EC2 instances spread out in different availability zones.

5. Monitoring and Logging

- We use CloudWatch to watch things in real time and to set up alarms.
- Using CloudTrail to keep track of the API calls.
- The logs for EC2, Lambda, and SQS are available in detail.

The reasons for the choices and justification for not using some of the AWS services from the categories such as storage, compute, databases, and security from the AWS LSDE course are:

1. Storage Service Category
Chosen Services:

- Amazon S3 (Simple Storage Service):
  We are utilizing S3 to store the uploaded files and processed outputs since it provides highly scalable, durable, and available storage. Besides that, enabling versioning and Cross-Region Replication
  (CRR) makes data durable, and disaster-recovery-ready across AWS regions for resilience and availability.
- S3 Event Notifications trigger AWS Lambda for automatic processing of files, thus
  allowing further workflow automation without a need to continuously run the compute resources.
  Other AWS Storage Services which were Not Utilized in the new architecture:

- Amazon EBS:
  EBS is a block storage service that is used primarily to provide persistent storage for EC2 instances. While useful to run databases or applications requiring fast, low-latency storage, S3 is more appropriate for the WordFreq application because it provides object storage for files, offers higher durability, and better fits large file storage and backups.
- Amazon EFS (Elastic File System):
  EFS provides scalable, shared file storage and is mostly used for applications requiring a network file system with shared access to data. Since the WordFreq application does not need shared file access, S3 is a better choice for object storage.

2. Compute Service Category
Chosen Services:

- AWS Lambda:
  AWS Lambda is used because it's a serverless compute service that automatically scales with demand, only charges for actual execution time, and seamlessly integrates with S3 for file processing. With a workload as intermittent, constant compute resources are not necessary; hence, Lambda scales efficiently. This is also very cost-effective.
- Amazon EC2 Auto Scaling:
  EC2 instances are used for the intensive and long-running processes, such as large-scale batch processing. There, Lambda may not be cost-effective. Auto Scaling ensures that EC2 instances are spun up or terminated based on queue size or traffic demand to keep the system current and
  available without any human intervention. EC2 Auto Scaling ensures high availability across
  multiple AZs and can recover from instance failures, which is crucial for the resilience of the architecture.

  Other AWS Compute Services Not Used:
- Amazon ECS: Elastic Container Service; Amazon EKS: Elastic Kubernetes Service
  Both ECS and EKS are great solutions for containerized applications, but they are overkill for the WordFreq app because it doesn't require complex container orchestration or microservices. Given that the application is based on file processing, AWS Lambda or EC2 instances would be simpler and more cost-effective options.
- AWS Elastic Beanstalk:
  Elastic Beanstalk is also PaaS but facilitates the deployment of applications that require a more complicated environment or are based on some framework, whereas Lambda goes one step ahead in simplification by removing infrastructure management altogether and is recommended for a lightweight serverless architecture.
- AWS Fargate:
  Fargate enables containers to run without managing the infrastructure beneath but
  is best utilized by applications requiring container orchestration. Since WordFreq does not
  require running containers and/or complex orchestration, it is better suited to Lambda or EC2 Auto Scaling.

3. Database Service Category
Chosen Services:

- DynamoDB:
  DynamoDB is a NoSQL database that provides the ability to store the word frequency result in a highly available, low-latency manner. Since it supports multi-AZ replication, it is resistant to regional failures, which is important to keep application data alive and durable. DynamoDB is very cost-effective for workloads with unpredictable or low traffic since it leverages the pay-per-request model. Thus, DynamoDB is suitable for occasional uses.

  Other AWS Database Services Not Used:
- Amazon RDS and Amazon Aurora:
  Both RDS and Aurora are relational databases suited for applications requiring complex queries, transactions, or structured data. The data which WordFreq needs to store is unstructured-word frequency results. Hence, using a NoSQL database like DynamoDB will be more efficient and cost-effective.
- Amazon Redshift:
  The reason is that it's a data warehousing service applied for large analytics and reporting. For a word

frequency application, it doesn't need any complex analytics/reporting for the dataset, so using a heavyweight would be an overkill for simple lookups. Dynamodb therefore provides enough for word frequency result storage without the overhead of a data warehouse.

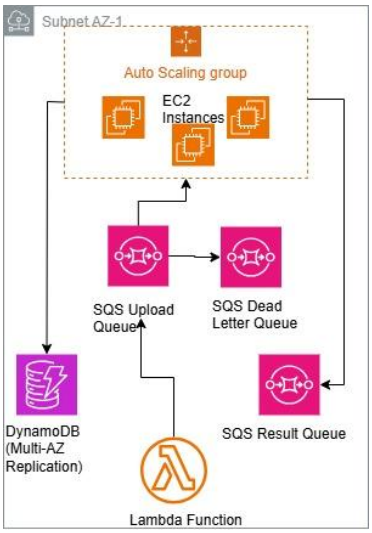
4. Security Service Category

Chosen Services:

- AWS IAM: The Identity and Access Management service is crucial in controlling the permissions of users against resources at AWS. These will grant access to various services like S3, Lambda, DynamoDB, etc., and ensure users have no more privileges on those sensitive data than is required to perform their specific function.
- The use of KMS encrypts information rested in S3 and dynamo databases, thus ensuring that everything is encrypted at rest; regular rotation of keys prevents sensitive data from being divulged to unauthorized users by ensuring better security.
- VPC Security Groups
  Security Groups control traffic flow to the ec2 instances and lambda functions hence allowing only the required traffic onto a particular network.
- AWS CloudTrail and CloudWatch :
- CloudTrail logs API calls and user activity, allowing an audit trail for security and compliance. CloudWatch is used to monitor, in real time, application performance, resource utilization, and alarms to help identify potential issues or security threats.

Following are the diagram of a subnet and the processing of Wordfreq Application txt files for the most frequent words:

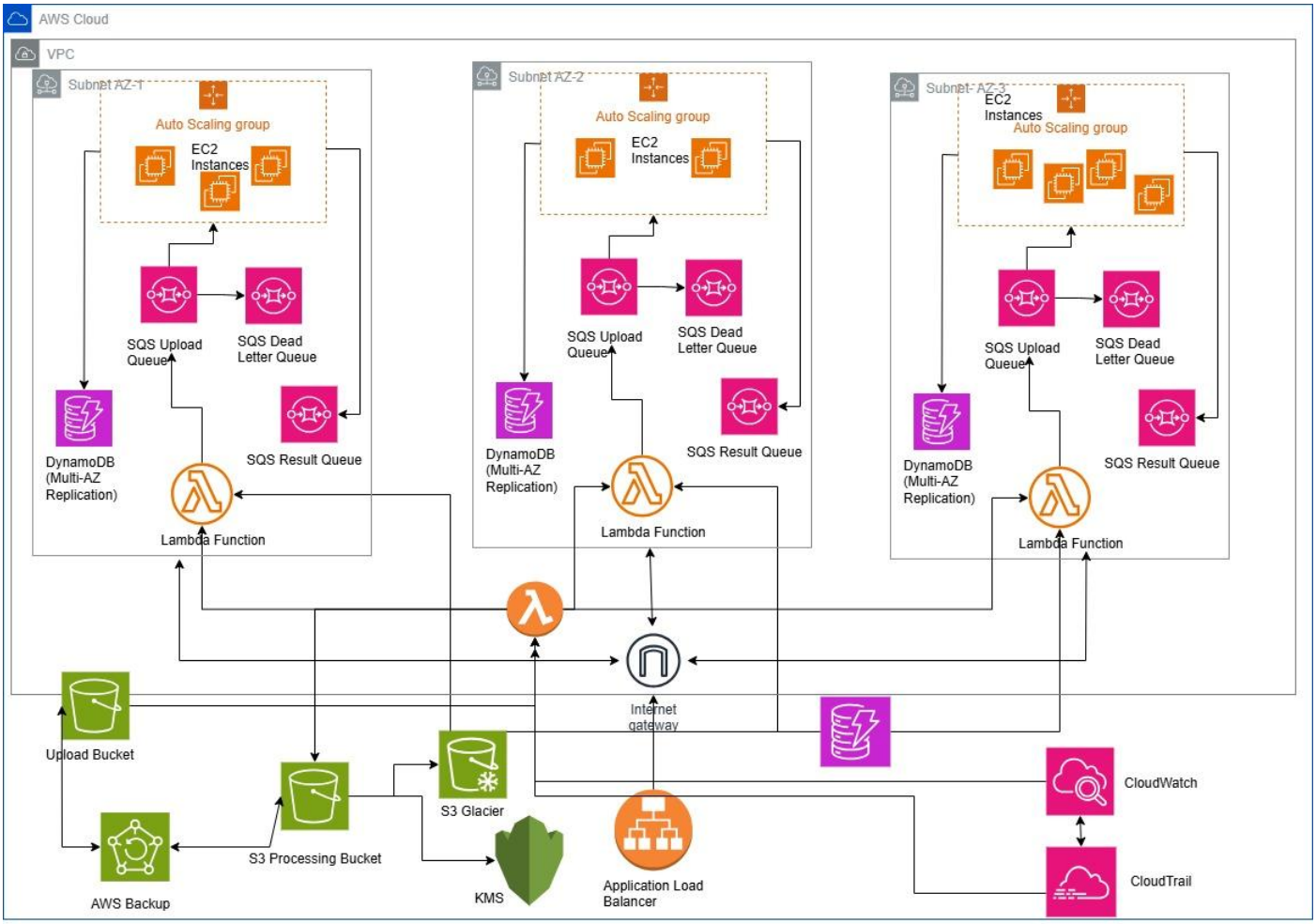
In this subnet, the process begins when a file is uploaded to the S3 Upload Bucket, triggering an S3 event notification that sends a message to the SQS Upload Queue. A Lambda function inside the subnet processes the message from the SQS Upload Queue and triggers EC2 instances in an Auto Scaling Group to perform the required file processing (e.g., word frequency analysis). After processing, the EC2 instances store the results in DynamoDB, which has multi-AZ replication for high availability. The processed results are then sent to the SQS Result Queue for further actions. If any task fails, the message is moved to the SQS Dead Letter Queue for debugging, ensuring reliable error handling and scalable processing within the subnet.



The Re-Designed Architecture for the Application:

This diagram depicts an AWS cloud architecture for the WordFreq application within a Virtual Private Cloud (VPC) spanning multiple subnets. Each subnet contains an Auto Scaling Group (ASG) with EC2 instances that process tasks from an SQS Upload Queue. Lambda functions, which process and interact with these SQS queues, are connected to

the application. The Lambda functions also interact with DynamoDB (with Multi-AZ replication for high availability) to store processed results, and the SQS Result Queue sends processed output. SQS Dead Letter Queues capture any unprocessed or failed messages for debugging. The system also integrates with various AWS services for resilience, backup, and security, including AWS Backup for S3 Buckets, KMS for encryption, S3 Glacier for long-term storage, and CloudWatch for monitoring. The Application Load Balancer routes traffic to the subnets through an Internet Gateway. CloudTrail tracks API activities for auditing. The architecture ensures availability, resilience, and cost-effectiveness by leveraging Auto Scaling, multi-AZ replication, and backup mechanisms.



TASK E - Alternative data processing AWS services

Two Alternative Data Processing Services that would be far more performant and robust for this WordFreq Application Processing task:

1. **AWS Lambda**

One of the Alternative Data Processing Services we can use for the WordFreq Processing Application which is used for its serverless computing architectural approach which helps to only pay as you use the the processing of the services. AWS Lambda is designed to be well-integrated with all parts of AWS Services, which makes it easy to automate operations tasks within your infrastructure. Lambda functions can be a great use to trigger the specific events in the application for Files uploading in S3, and storing the results in the dynamo DB only when triggered, etc. With Lambda we no longer have the need to install security updates or manage remote access such as SSH for administration, unlike the current Data Processing structure of WordFreq Application. Like the New Architecture we created for Task D as we can see the Lambda function acts as a Server and monitors and triggers functions to process the txt files or Store in the Dynamo DB or the SQS queues results Processing. The Lamda function easily handles multiple operations effectively and efficiently as it is highly scalable and can scale automatically according to the workload.

Advantages and Key Benefits of the AWS Lambda in WordFreq Applications:

- Lambda Function can easily replace EC2 instances in the current architecture, removing the need to manage Auto Scaling Groups and EC2 instances. Lambda being a serverless architecture helps drastically to simply the Architecture and make it more feasible and reduce operational overhead.
- It works well with other AWS services in WordFreq for SQS for queue-based handling of tasks, S3 for file uploads, and DynamoDB for storing the results of data processing. therefore, AWS Lambda helps would be far more performant and robust as a Data Processing Service.
- Cost Efficiency: Lambda's pay-per-use model ensures that you only pay for the time your code executes, making it very cost-effective for sporadic or burst processing tasks.
- Simplified Operations: Eliminates the need for provisioning or managing servers, reducing operational overhead.
- Scalability: Scales automatically with the number of events so that the application can handle increases in the number of processing tasks without manual intervention.

2. Amazon ECS (Elastic Container Service)

Amazon Elastic Container Service is a Platform as a Service - PaaS AWS Product. Similar to the Auto Scaling Group Instances the process of deploying to ECS is similar to deploying to EC2 instances, but instead of uploading application files to an instance, you deploy Docker images that run your application within containers. Elastic Container Service Clusters can be used to deploy the WordFreq Application to containers i.e. instead of EC2 instances and Auto Scaling Group the ECS Cluster layer can be used. ECS is an AWS service that we can use as it provides control over the environment and allows more effective scaling with demand. For the WordFreq Application.

For WordFreq, ECS can handle containerized applications by packaging the word frequency processing task in a Docker container. Such containers can be deployed on many EC2 instances, which enables high-performance parallel processing of huge datasets. The role of container orchestration will be done by ECS to make the application scale according to demand and optimize the resources. ECS also easily integrates into other AWS services such as Amazon S3 for file storage, Amazon DynamoDB to store results, and Amazon CloudWatch for monitoring.

Advantages and Key Benefits of the Amazon ECS – Elastic Container Service in WordFreq Applications:

- Scalability: ECS can automatically scale resources up or down based on demand, hence making sure that the application can handle volumes of big data with efficiency.
- Containerization: The WordFreq application will be packaged into containers for better control of the environment, dependencies, and execution.
- Cost Efficiency: You pay for the EC2 instances running the containers, and you can use AWS Fargate for serverless container management to lower your costs further.
- Fault tolerance: ECS automatically detects failed instances and replaces them maintaining high availability and resilience for the WordFreq Application.

REFERENCES:

1. Amazon Web Services, 2023. *AWS Certified Solutions Architect Study Guide.* 3rd ed. Indianapolis, IN: Wiley.
2. Barrett, J. and Stoughton, B., 2023. *AWS Certified Solutions Architect Official Study Guide: Associate Exam.* Indianapolis, IN: Sybex.
3. Barr, J. and Farley, J., 2023. *AWS Basics: Introduction to Cloud Computing with Amazon Web Services.* 2nd ed. Boston: Addison-Wesley Professional.
4. Berman, L., 2023. *AWS Certified Cloud Practitioner Study Guide.* Hoboken, NJ: Wiley.
5. Chauhan, A. and Anand, R., 2024. *AWS Simplified: The Complete Guide for Beginners.* San Francisco, CA: Packt Publishing.
6. Hedges, J., 2023. *AWS for Beginners: Learn Cloud Computing in an Easy Way.* New York: O'Reilly Media.
7. Ismail, A., 2023. *AWS Essentials: A Beginner's Guide to Cloud Services and Architecting Applications.* London: Packt Publishing.
8. McLaughlin, M., 2023. *Mastering AWS Cloud Practitioner: The Simplest Guide to AWS Fundamentals.* 3rd ed. New York: Sybex.
9. Amazon Web Services, 2024. *AWS Documentation: Core Services.* Available at: https://docs.aws.amazon.com/
10. Amazon Web Services, 2024. *AWS Academy Cloud Foundations Course Materials.* AWS Academy Learner Lab.
11. Amazon Web Services, 2024. *AWS Cloud Practitioner Essentials.* AWS Training and Certification. Available at: https://aws.amazon.com/training/
12. Amazon Web Services, 2024. *Getting Started with AWS: The Basics of Cloud Computing.* Available at: https://aws.amazon.com/getting-started/
13. Amazon Web Services (AWS) (2023). Amazon EC2 Auto Scaling User Guide. Available at: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html