Coursework - EMATM0051 Large Scale Data Engineering [Data Science cohort]

Nmae: Rahul Ray

Student id: fw24221

Introduction (Part-1)

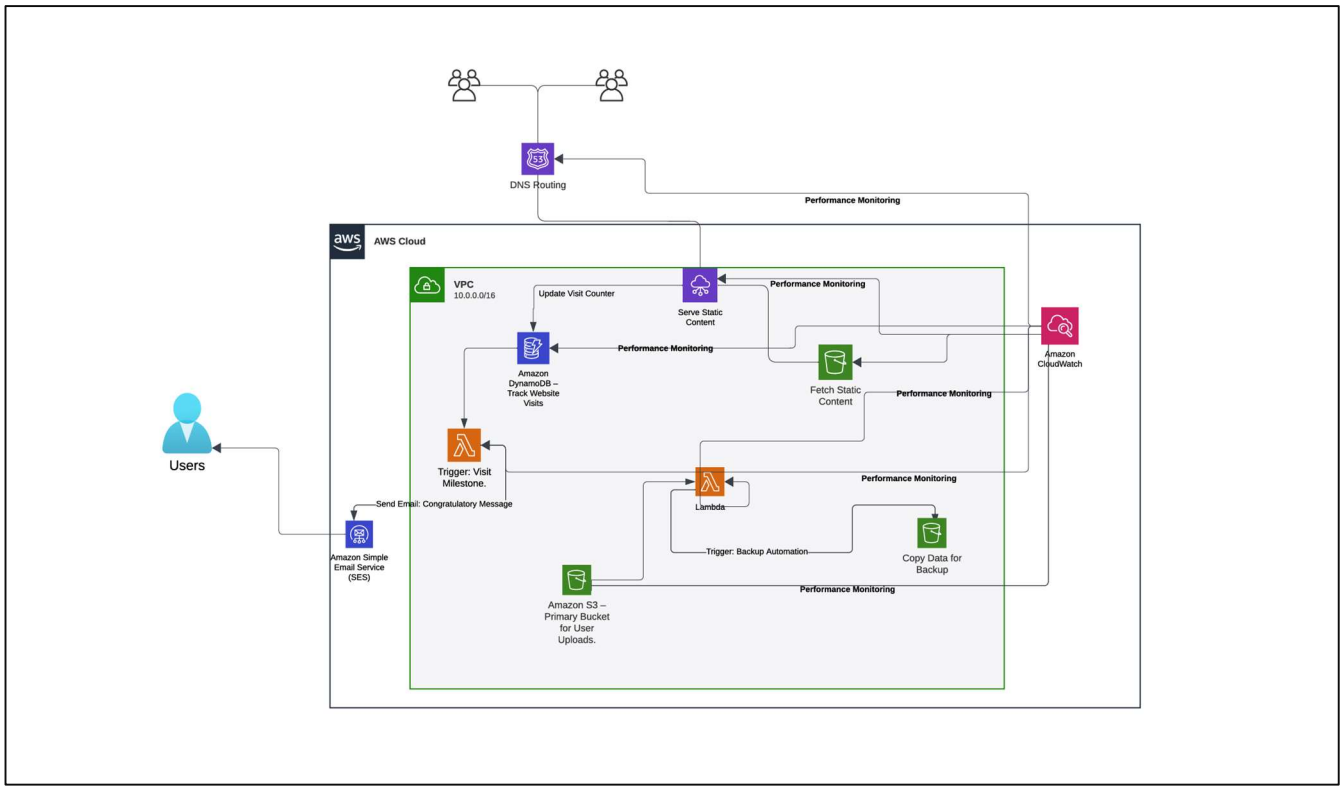
The “MyTravel” web site is an international travel forum in which users can post photos and text. The platform must deliver a truly outstanding user experience by being highly available, low latency, and globally reachable. The system should also automatically back-up the data uploaded to it, and send out an email of congratulations when total visits are over 1,000,000, etc. This architecture aims the fulfill those needs using the capabilities of AWS to support them in a scalable, reliable and cost effective way.

Services is used in this cloud architecture:

1. Amazon Route 53
2. Amazon CloudFront
3. Amazon S3 (Primary Bucket)
4. Amazon S3 (Backup Bucket)
5. AWS Lambda
6. Amazon DynamoDB
7. Amazon SES (Simple Email Service)
8. Amazon CloudWatch
9. Amazon S3 Glacier

Architecture diagram:

The following diagram illustrates the architecture of the "MyTravel" website:



Architecture Components and Workflow

The MyTravel website is a full-stack cloud-native website built on a combination of AWS services, with high availability, scalability, automatic maintenance, and cost-effectiveness. The architecture below gives a detailed look at MyTravel.

User Access and DNS Routing

Amazon Route 53 is crucial to the MyTravel website user's request for the CloudFront distribution. Although providing low-latency DNS routing and managing a custom domain (mytravel.com), Route 53 still ensures smooth user access. When users visit the site, Route 53 looks up the domain name and sends requests on to CloudFront for further processing.

Content Delivery:

Amazon CloudFront, designed to distribute static content with low latency throughout the world, fetches static contents from the primary S3 bucket and supports secure delivery through HTTPS. Content on CloudFront's edge nodes is cached. Consequently the user experience is significantly enhanced.

Static Content Storage:

The main mass-storage system for MyTravel comprising user-uploaded images and text, as well as the origin for static content delivery, Amazon S3 allows public access for content delivery. Users place their data upon this bucket, which also triggers automatic backup and monitoring processes.

Content Backup Automation:

To make certain Redundancy, the backup of user-uploaded content is automated by AWS Lambda. Each event of an upload to S3 triggers a lambda function that replicates contents in a backup bucket secondary to S3. Lifecycle policies move older data in the backup bucket into Amazon S3 Glacier and so save on long-term storage costs.

The section discusses Asynchronous Buffering, the benefits of DynamoDB Stream, and a way to keep track visits people make to dynamic pages on your Web site. Every time someone calls an active page, there will be an increase in the quantity on it for every visit. When the number of visits comes to exceed a preset maximum (1 million above is representative), DynamoDB Streams trigger event notifications and then messages of this kind will be sent by using Lambda. The congratulatory email, provided with the aid of Amazon SES and AWS Lambda function, can deliver the masseges in a timely manner with significant traffic on the road. 6 Monitor and Alarm Amazon CloudWatch monitors the application's performance,

Monitoring key metrics across all services: S3: Storage usage levels and how that storage gets accessed.

Lambda: It records how many invocation errors are committed according to time-frame and the time it takes for those executions themselves. DynamoDB: Read write throughput levels and latency.

High Availability and Low Latency: Route 53 and CloudFront provide global content distribution with low latency. Services like S3 and DynamoDB offer high availability,

long-term retention. Scalability: Serverless solutions like Lambda and managed services such as S3 and DynamoDB are employed to manage high traffic peaks

Automation: Done with automated backup procedures, notifications of milestones over email diminish reliance upon manual intervention.

Cost Effectiveness: S3 lifecycle policies meant that as data became old and no longer needed for everyday use, it would go into Glacier storage where it was much cheaper than its counterpart in regular servers; and serverless services with more scalable metering reduced operational costs.

Monitoring and Insights: CloudWatch yields actionable insights into resource use that streamline operating procedures and nip problems in the bud. Conclusion This architectural model shall guarantee the 'MyTravel' website is highly available, scalable and automated.

Scaling the WordFreq Application on AWS (part 2)

Introduction:

The WordFreq application is designed to analyze text files and identify the top ten most frequently used words. It uses several AWS services to manage its workflow: files are uploaded to an S3 bucket, jobs are queued in SQS, processed by worker instances running on EC2, and the results are stored in DynamoDB. This setup allows the application to handle multiple jobs efficiently.

For this project, the goal was to make the application smarter by adding auto-scaling. This means setting it up to automatically add or remove EC2 instances based on how many jobs are waiting in the queue. This ensures that the application can handle a sudden spike in workload while also keeping costs low during quiet periods.

Task A – Install the Application

The installation process is involved setting up essential AWS services to support the WordFreq application. To install the application, all the instructions has been followed mentioned in the readme.txt file in the application.

The setup of applications on EC2 instances.:

The WordFreq application was deployed to an EC2 instance which was we configured for job processing and interact with other Amazon Web Services (AWS) services such as S3, SQS, DynamoDB. It had the following specific attributes:

0 o instance type: t2.micro, for its economical cost and sufficient performance level 1 AMI Used: Ubuntu 22.04 LTS (HVM) In every respect efficient stable base operating systems C Static public IPv4: 4.165.17.36 by means of which instance is accessed from a distance and modified D IAM Role: EMR_EC2_DefaultRole, the role and rights necessary for carrying out required activities on AWS resources. E network configuration: launched in VPC ID as a security group rule. livingroom-restorbre The core compute resource where the WordFreq worker service executes, extracting jobs from the SQS queue for processing and allocating them in DynamoDB. The security group has SSH traffic on port 22 inbound only and free outbound traffic, without any AWS service restrictions for seamless communication.

The details of the EC2 instance are shown in the screenshot below:

Figure 1: EC2 Instance Summary for WordFreq Application

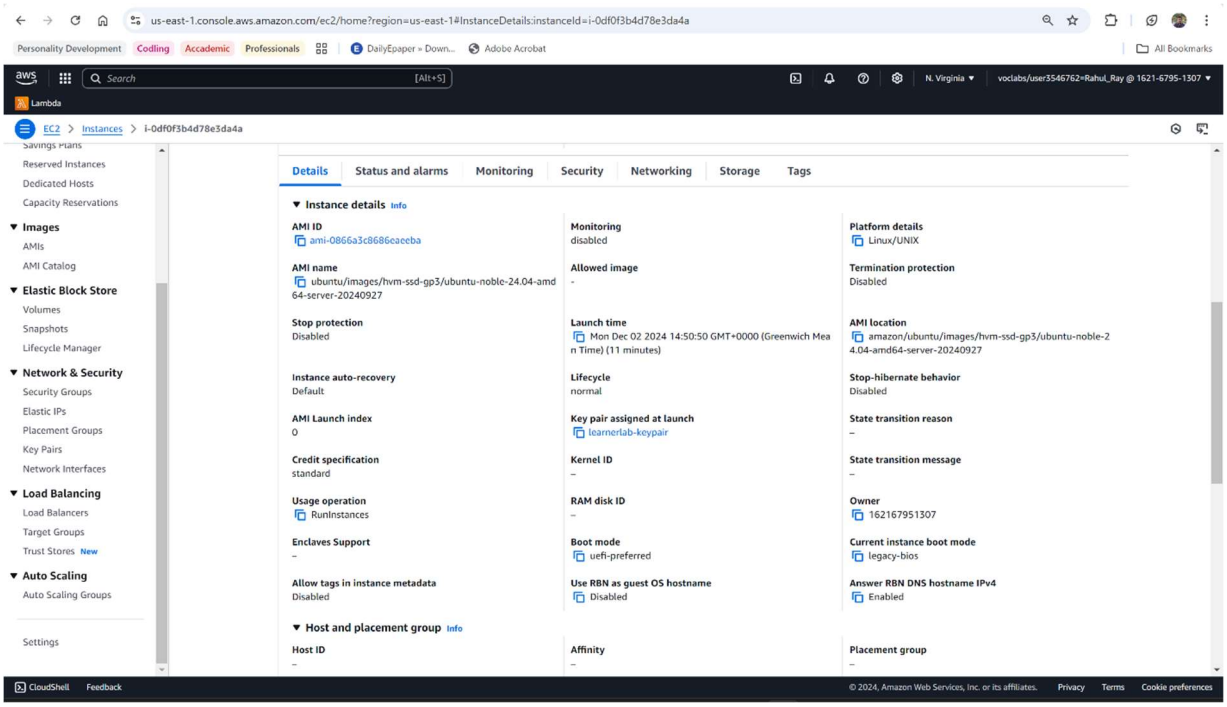
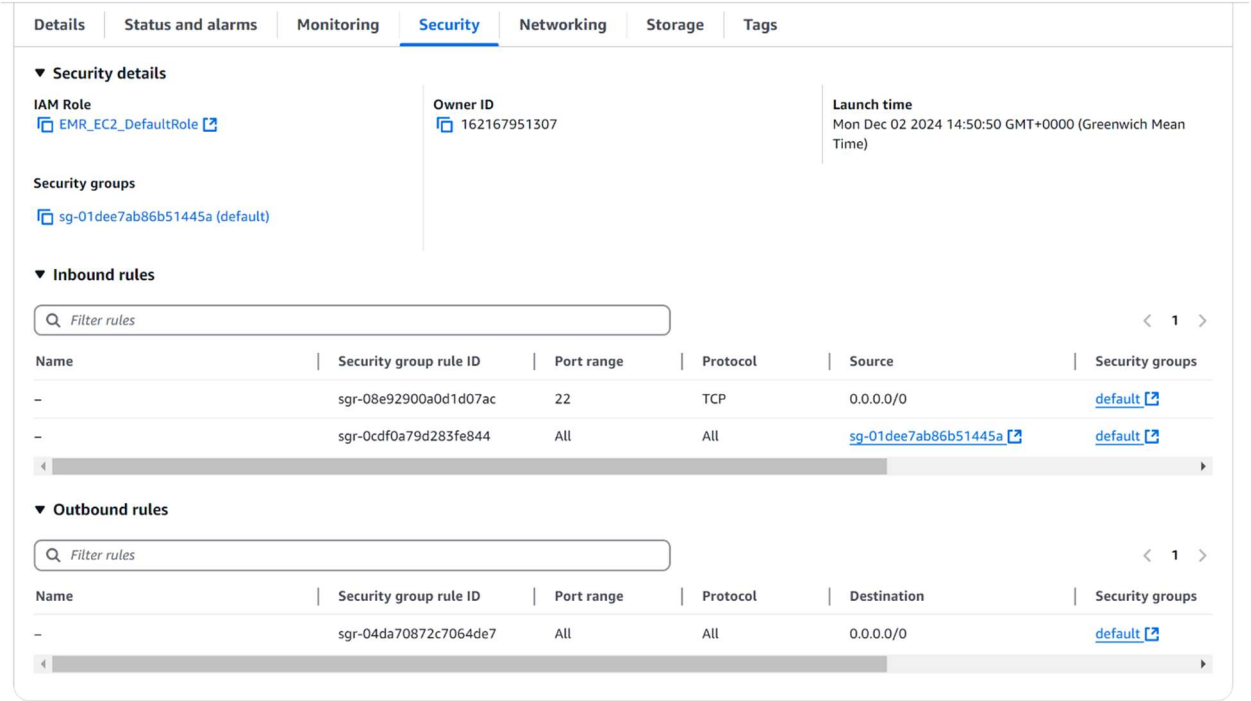


Figure 2: Security Configuration for the EC2 Instance.



S3 Bucket Setup:

I made two S3 buckets to handle file storage for the WordFreq program:

- **rahul-wordfreq-uploading:** Stores text files to be processed by our program. This bucket currently has 121.txt files.
- **rahul-wordfreq-processing:** Files that leave the uploading bucket head here next for WordFreq service consumption.

Both buckets were created in region US East (N. Virginia). For simplicity and monkey verification, standard settings have been applied - versioning (currently off) and encryption (also off) disabled.

Details of these S3 buckets are given in the screenshot:

Overview provides information about both Buckets:

The creation time can be seen here for the rahul-wordfreq-uploaded and rahul-wordfreq-processed (both created on 30 November 2024).

At the top of the uploaded file list is the uploading bucket itself.

When you click properties under a bucket's list view (Displaying the list of all objects inside an/each bucket) and start scrolling down to see details like ARNs (Amazon Resource Names), it's clear that's were copied directly out of various Amazon Web Services documents.

Figure 3: The rahul-wordfreq-uploading bucket properties page includes details like the Amazon Resource Names

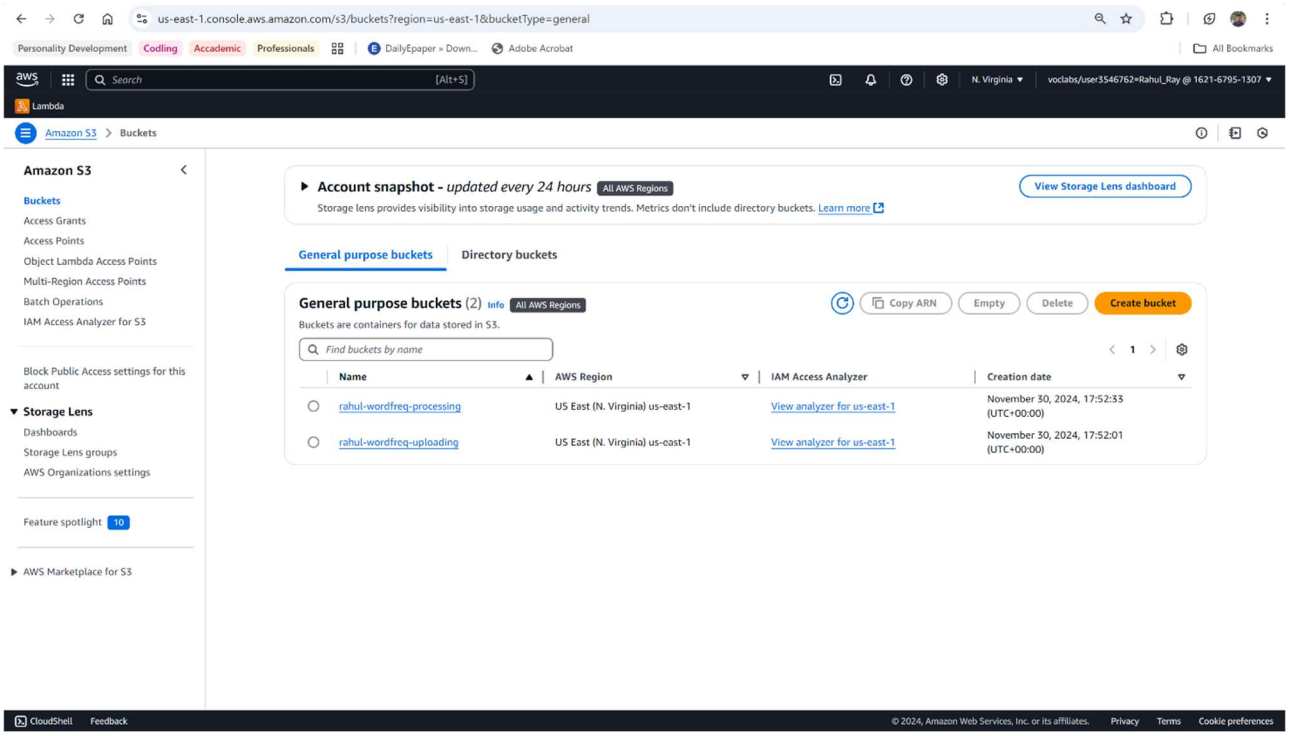


Figure 3: The rahul-wordfreq-processing bucket properties page includes details like the Amazon Resource Names

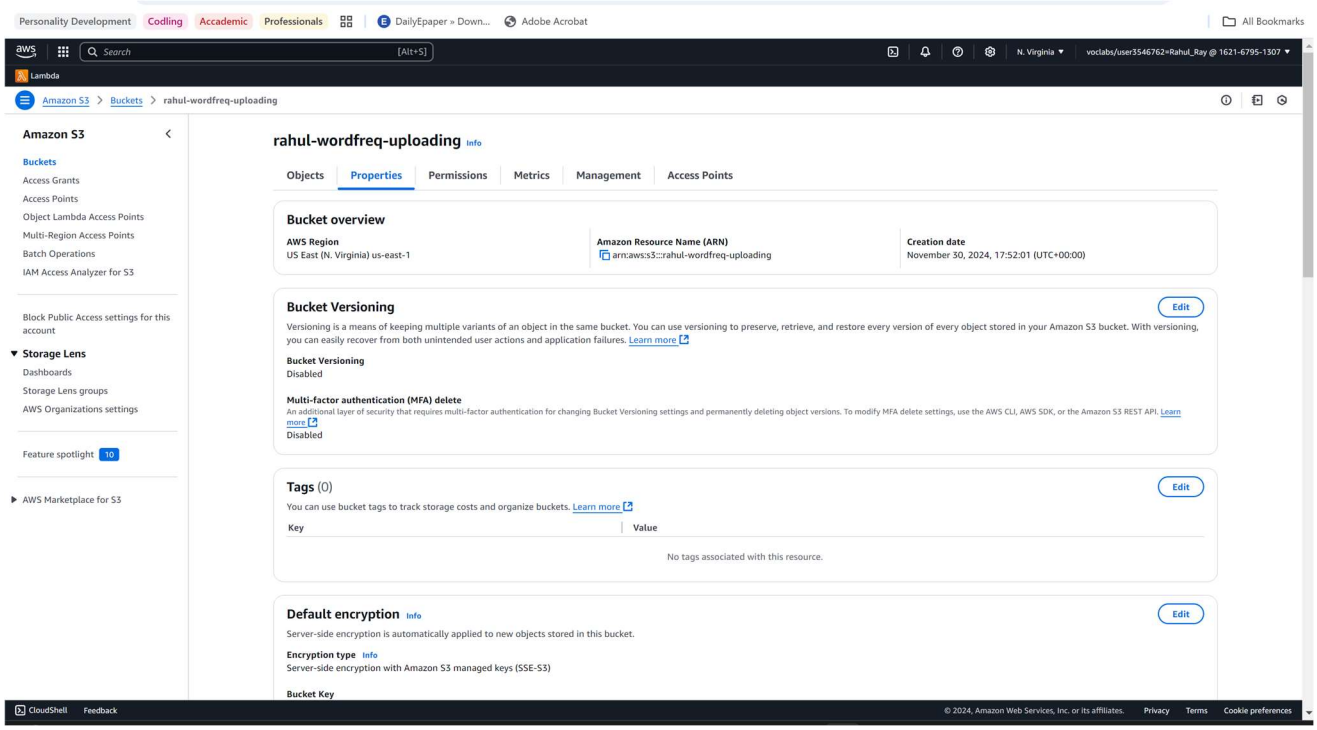
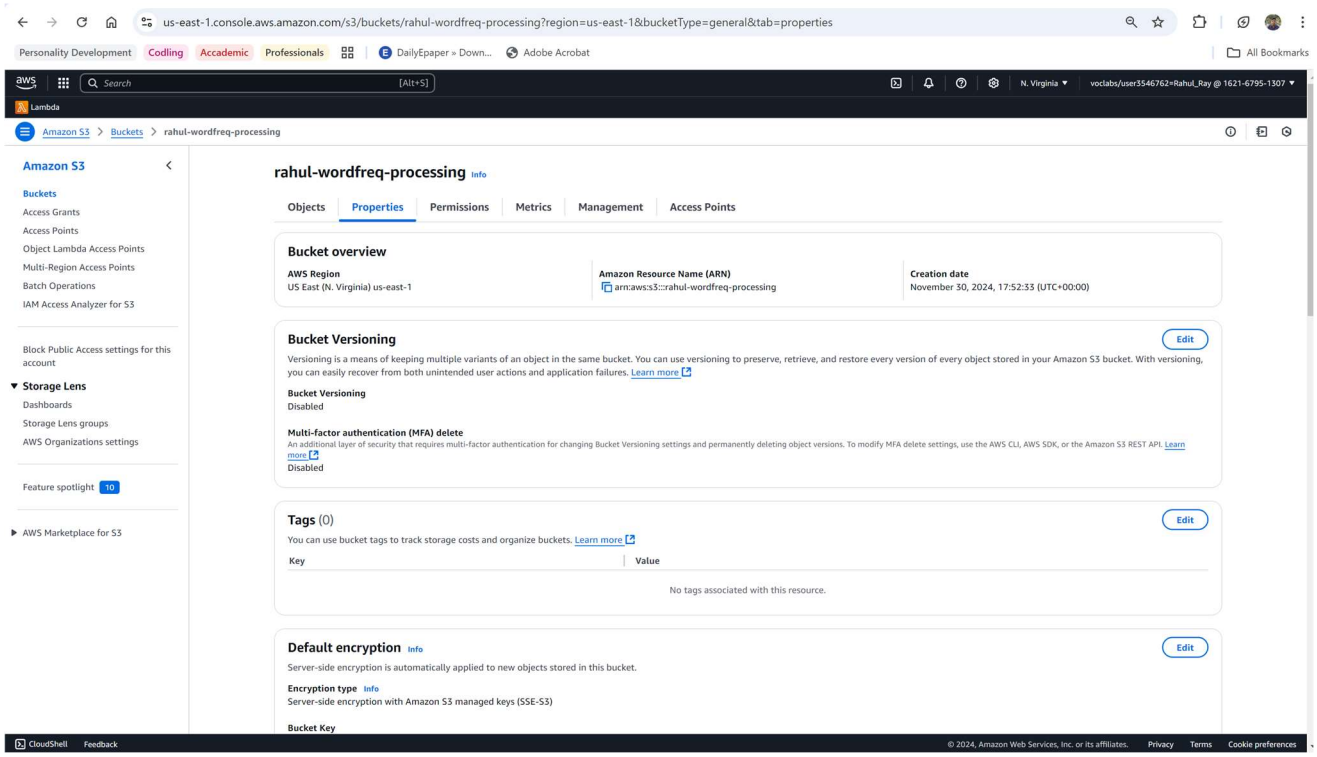


Figure 4: The overview of the buckets



SQS configuration:

For the WordFreq app, two Amazon SQS lists were created for you to manage message handling.

- **wordfreq-jobs:** This line is about the text files that need to be doctored stored by the queue. It will be an input point for your application where every message corresponds to a file in the S3 processing bucket all sent from clients around the world.
- **wordfreq-results:** After the worker service has gone through these text files, we can get results out the data for everybody--such as the most frequent words in files being processed by our system.

Important points:

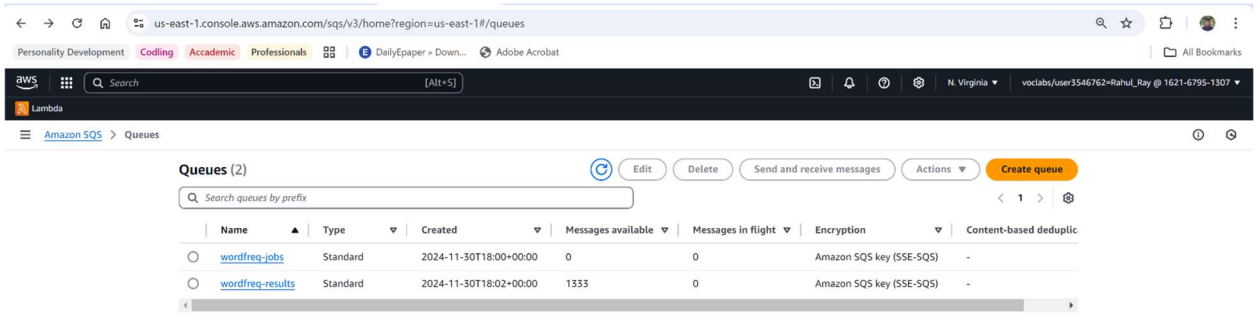
Both queues are of Standard type thus allowing high throughput and at-least-once delivery.

Encryption: Turned on using Amazon SQS-managed keys (SSE-SQS) assure safe handling of your messages.

Messages Available:

- wordfreq-jobs: 0 messages (not currently operating).
- wordfreq-results: 1,333 messages (results of processing now available).

The details of the S3 buckets are shown in the screenshot below:

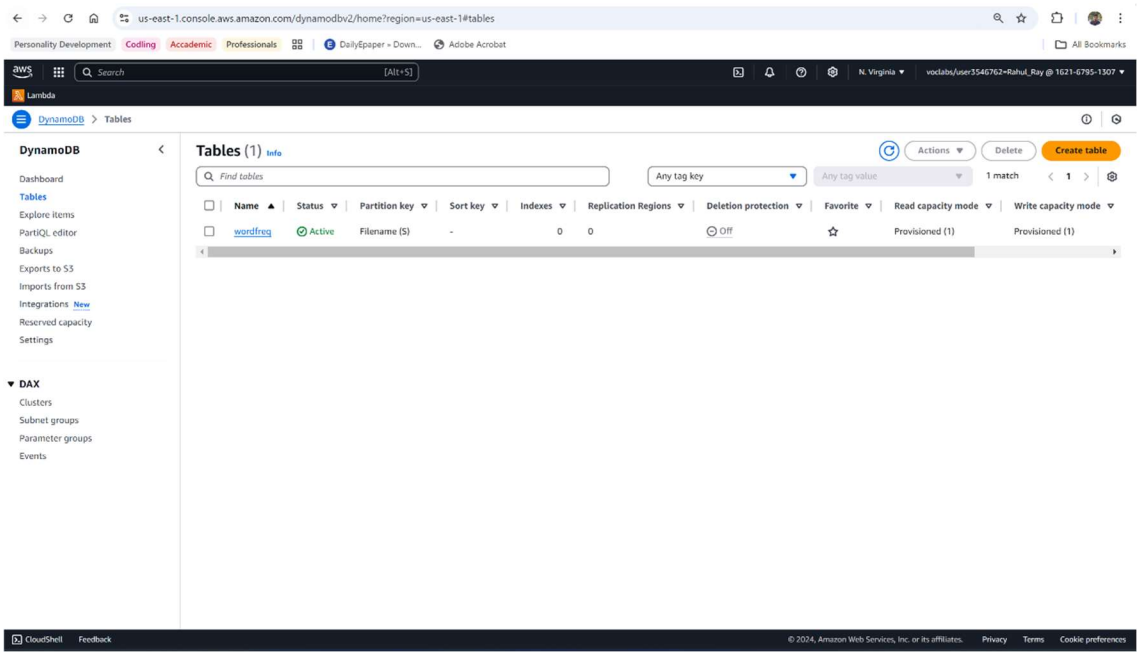


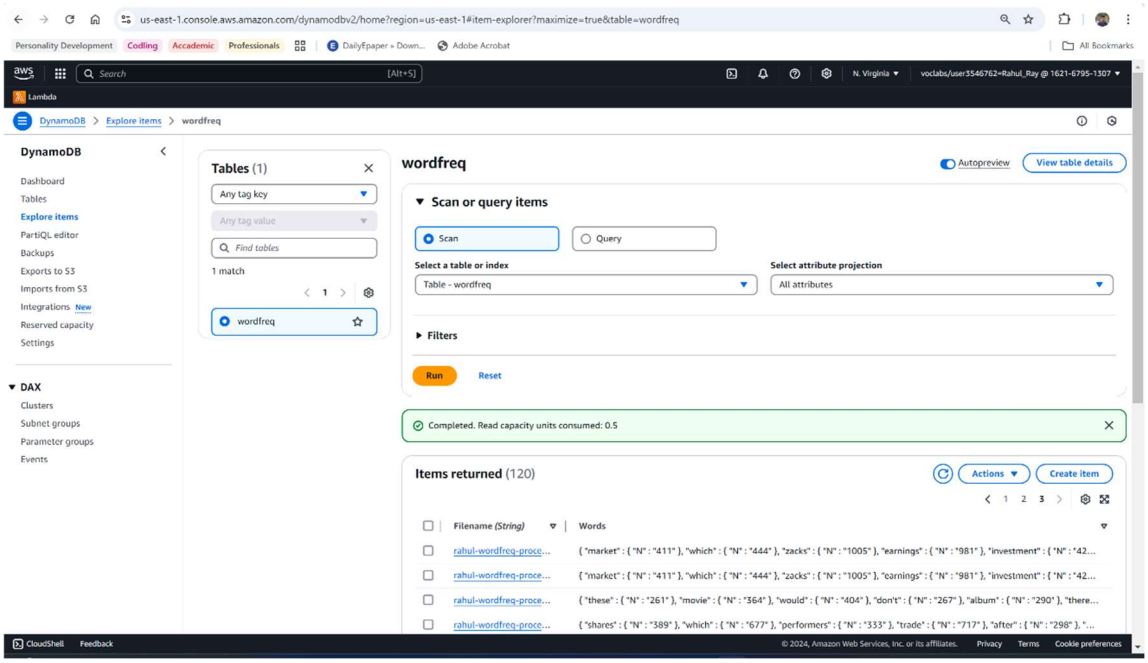
DynamoDB Setup:

The wordfreq DynamoDB table was created to store the results of the WordFreq application. Each entry contains the file name and the list of the most frequently used words along with their counts, making it easy to retrieve and review processed data.

The details of the S3 buckets are shown in the screenshot below:

Figure 4: The overview of DynamoDB table





Task B – Design and Implement Auto-scaling

In this step affected, is to WordFreq program to realize automatically scaling its own capacity according to changes in workload. Because of the integration of CloudWatch Alarms and an Auto Scaling (Scaling - changed from Group to show more scalability) Group, the application may scale effectively when more work comes in and after idle periods are over as well. By monitoring the SQS ApproximateNumberOfMessagesVisible metric, the auto-scaling setup can determine the number of unprocessed jobs waiting in line. Now when a user starts into any part of the pipeline that causes translation failing is no longer delayed as it used to be for a very long time. Thus we find that there are structural issues not provoked by accident, just as it would not happen by random chance if someone lived like this.

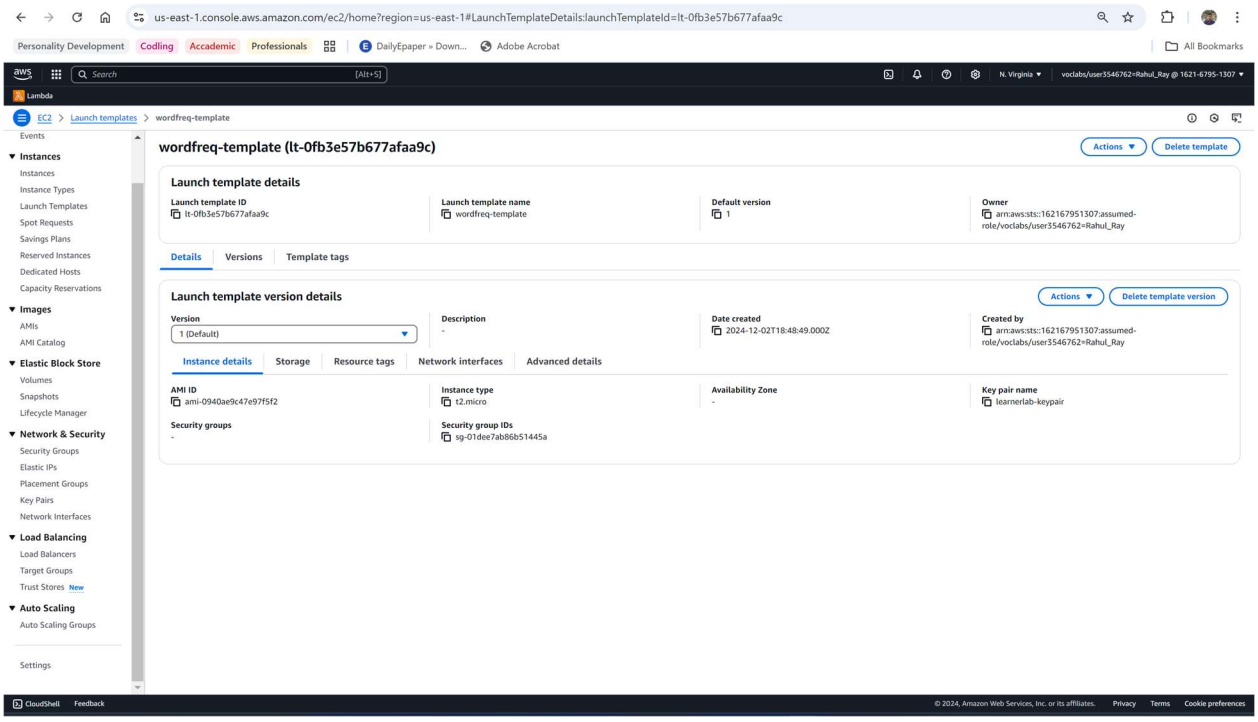
Launch Template

The **Launch Template** for the WordFreq Auto Scaling Group was carefully designed to ensure that new EC2 instances are consistently configured. This template includes all necessary details to streamline the provisioning of additional instances when scaling is triggered.

Key Details:

- **Version**: 1 (Default)
- **AMI ID**: ami-0940ae9c47e97f5f2
- **Instance Type**: t2.micro
- **Security Group**: sg-01dee7ab8651445a
- **Key Pair**: learnerlab-keypair

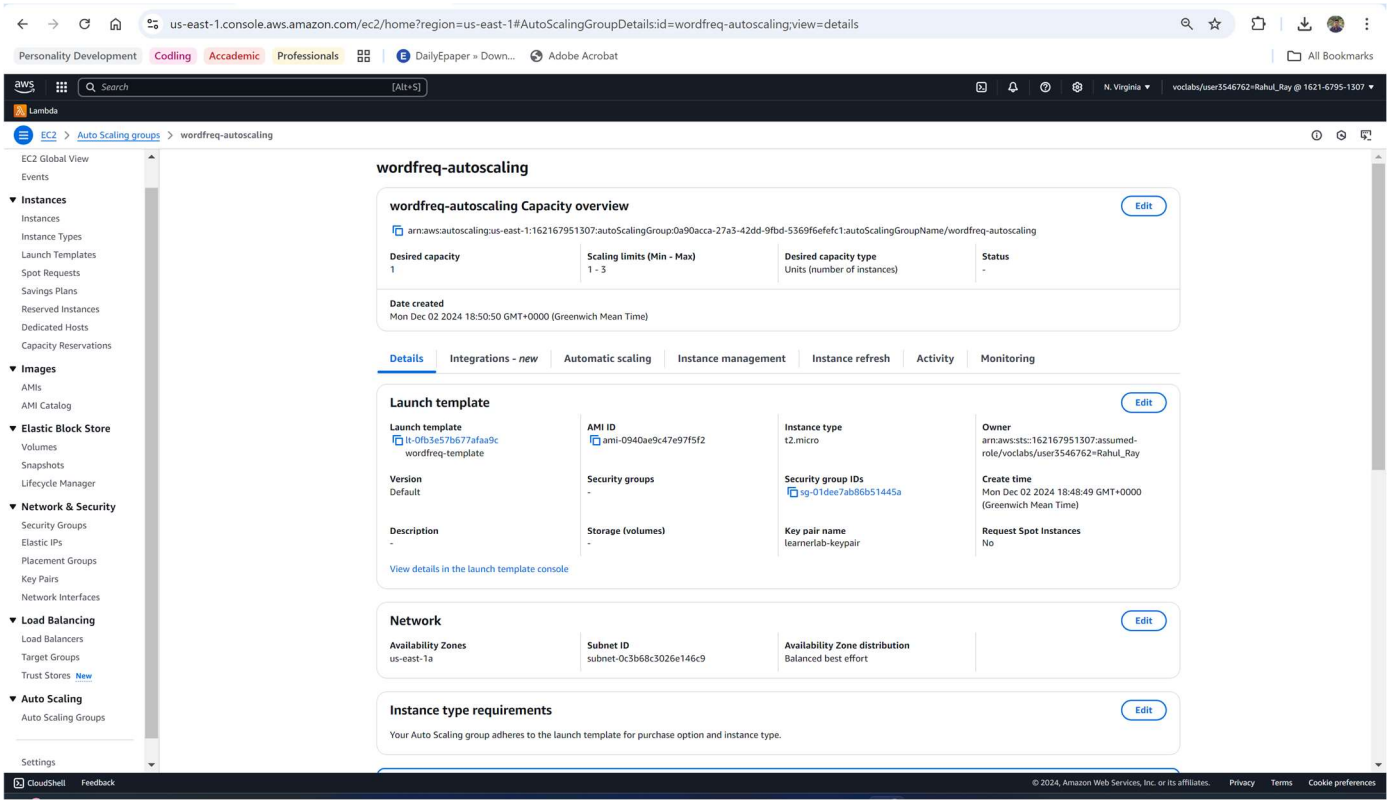
Figure 6: This screenshort shows EC2 launch tamplate configuration



An overview of Auto Scaling Groups setup:

The main design purpose of the Auto Scaling Group (ASG) is to dynamically manage EC2 instances running WordFreq application. Its main goal is to make sure that this single instance of the application scales up or down according to load, keeping a fine balance between performance and cost.Minimum, Maximum & Desired Capacity: Configured as Oldest First. Allows new instances to become fully operational or gives queues the chance to empty after scaling. A 300 second cooldown was set for both a scale-out and a scale-in event. This would enable the system enough time to stabilize after any of these before it begins to take further action.

Figure 4: The overview of autoscaling group



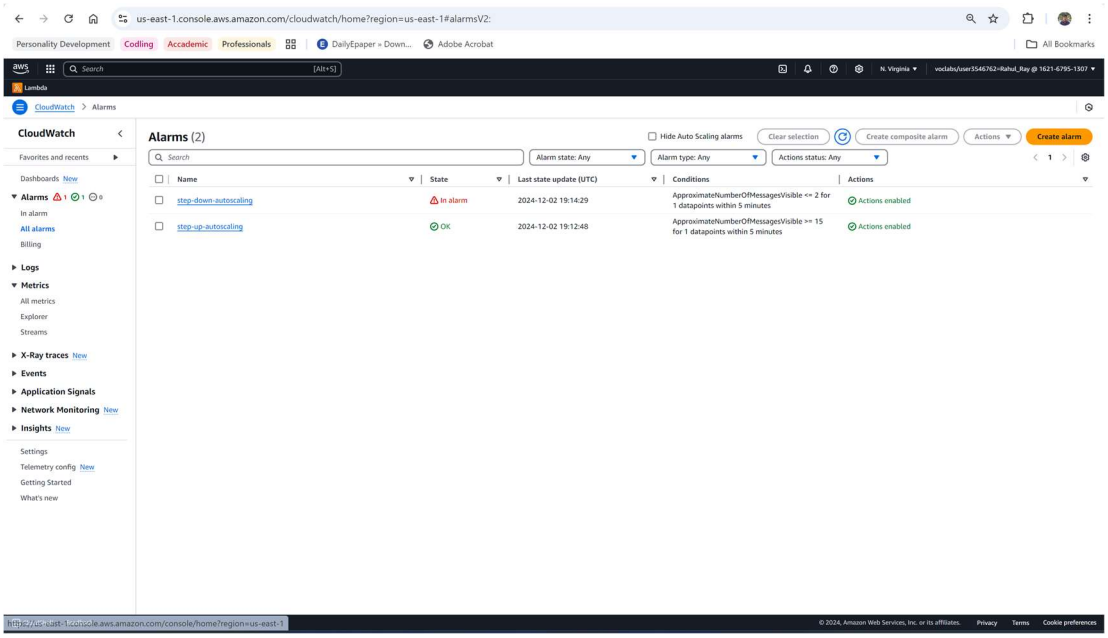
CloudWatch Alarms:

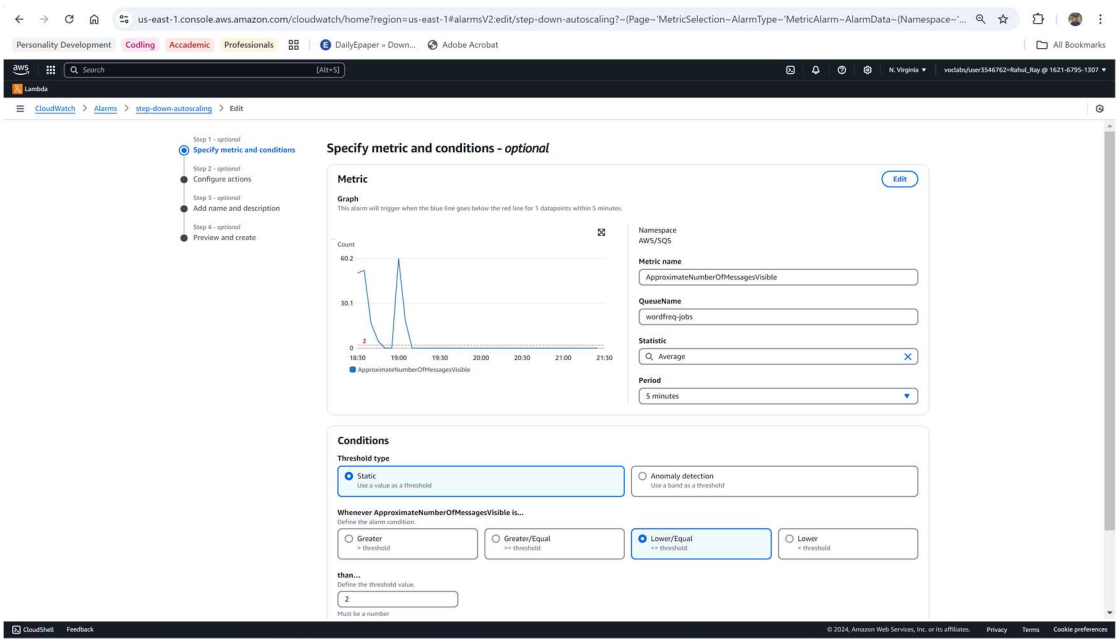
For the WordFreq application, two CloudWatch Alarms were set up to look after scaling expansion and contraction.

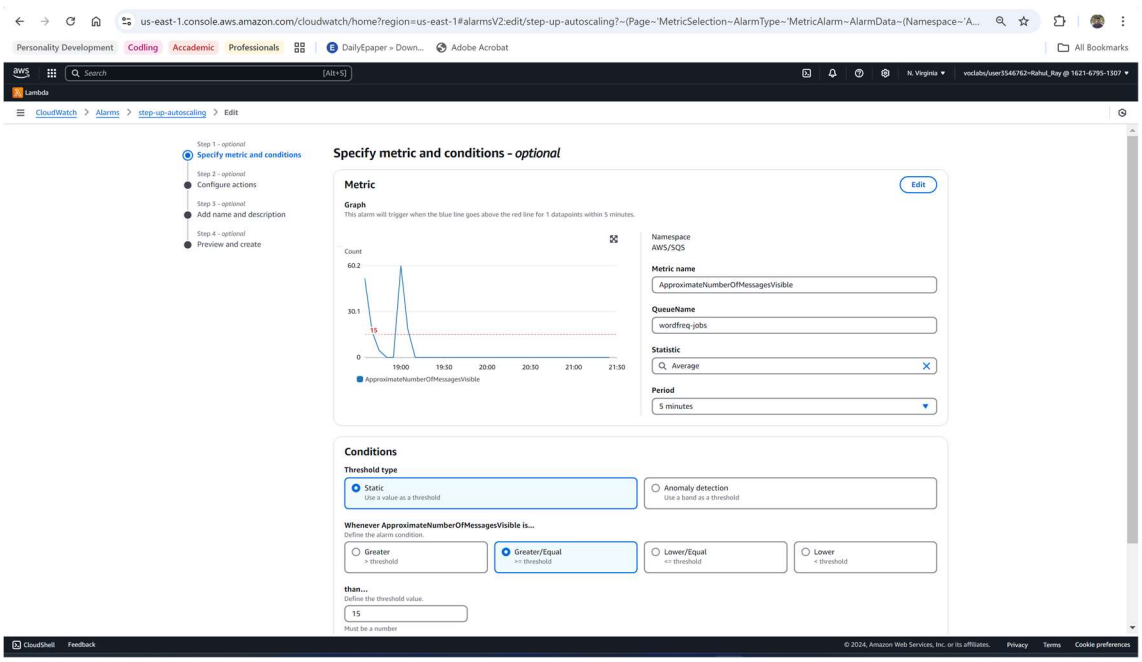
- Scaling Out Alarm: Explain that this is an alarm which triggers when the SQS queue length exceeds 15 messages-an indication that More instances are needed to deal with the workload.Note the assessment period (for example 5 minutes) and action taken in response to scaling out (add one instance).
- Scaling In Alarm:Note that this content alarm triggers when the number of messages on SQS queue drops to less than 2, which indicates that the system must scale back by turning off an instance. Note what action to take (stop 1 instance) and CoolDownPeriod.

These alarms allow the application to dynamically scale, ensuring performance and cost are in harmony.

Figure 1: Alarm configuration for scaling down and scaling up


us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarmsV2:
CloudWatch
Alarms (2)
Create alarm
step-down-autoscaling
In alarm
2024-12-02 19:14:29
ApproximateNumberOfMessagesVisible <= 2 for 1 datapoints within 5 minutes
Actions enabled
step-up-autoscaling
OK
2024-12-02 19:12:48
ApproximateNumberOfMessagesVisible >= 15 for 1 datapoints within 5 minutes
Actions enabled


us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarmsV2:edit/step-down-autoscaling
CloudWatch > Alarms > step-down-autoscaling > Edit
Specify metric and conditions - optional
Metric
Edit
Graph
This alarm will trigger when the blue line goes below the red line for 1 datapoints within 5 minutes.
Namespace
AWS/SQS
Metric name
ApproximateNumberOfMessagesVisible
QueueName
wordfreq-jobs
Statistic
Average
Period
5 minutes
Conditions
Threshold type
Static
Use a value as a threshold
Anomaly detection
Use a band as a threshold
Whenever ApproximateNumberOfMessagesVisible is...
Greater
Greater/Equal
Lower/Equal
Lower
than...
2


us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarmsV2:edit/step-up-autoscaling
CloudWatch > Alarms > step-up-autoscaling > Edit
Specify metric and conditions - optional
Metric
Edit
Graph
This alarm will trigger when the blue line goes above the red line for 1 datapoints within 5 minutes.
Namespace
AWS/SQS
Metric name
ApproximateNumberOfMessagesVisible
QueueName
wordfreq-jobs
Statistic
Average
Period
5 minutes
Conditions
Threshold type
Static
Use a value as a threshold
Anomaly detection
Use a band as a threshold
Whenever ApproximateNumberOfMessagesVisible is...
Greater
Greater/Equal
Lower/Equal
Lower
than...
15

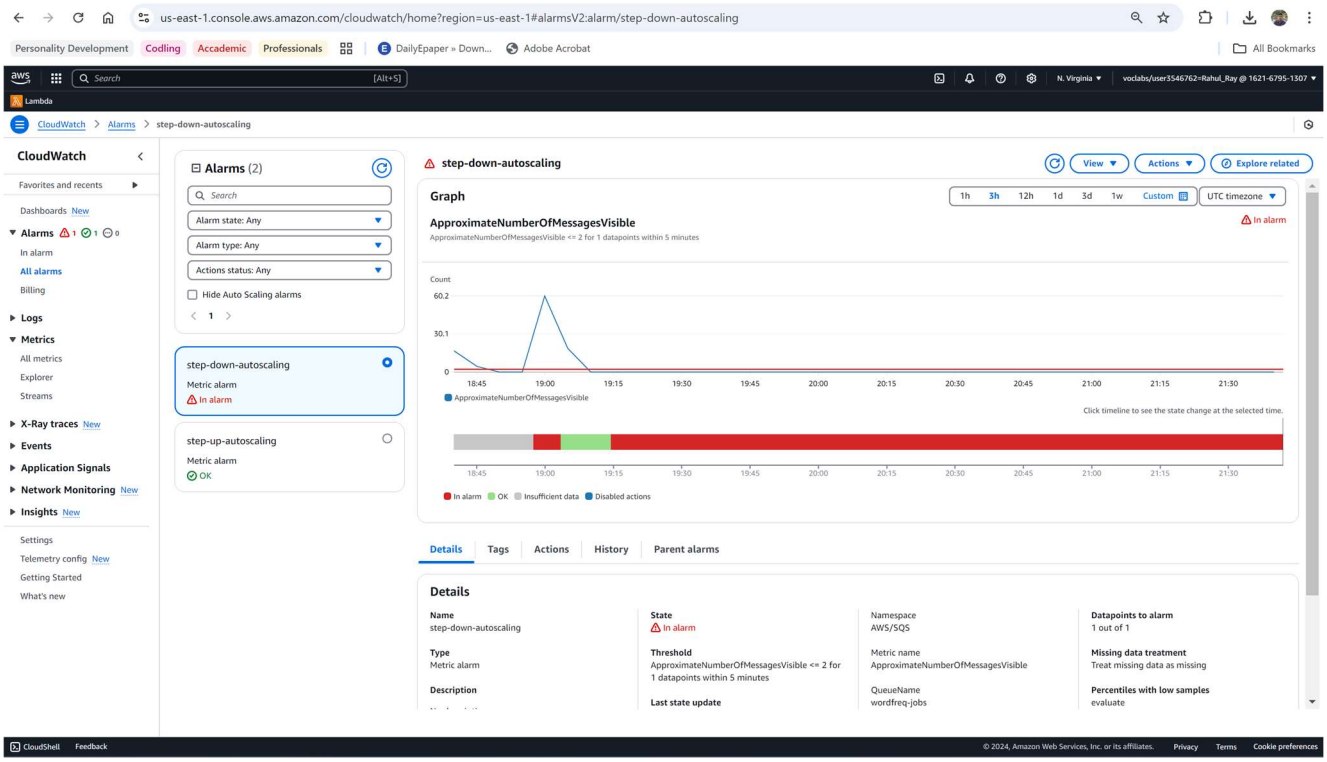
CloudWatch Metrics:

The CloudWatch Metrics used for scaling are based on the SQS

ApproximateNumberOfMessagesVisible:

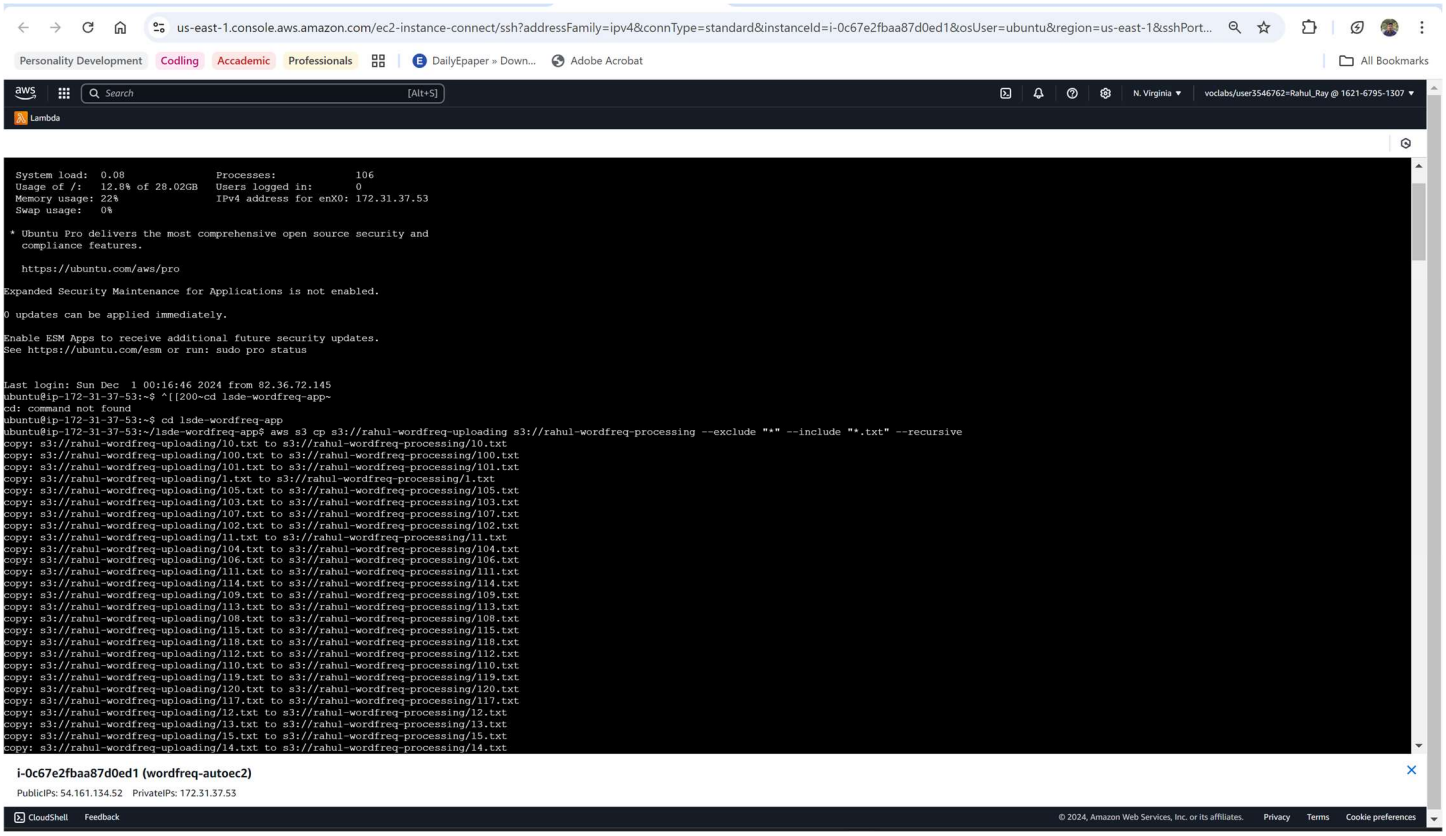
- **Metric Name**: ApproximateNumberOfMessagesVisible
- **Statistic**: Average
- **Period**: 5 minutes

This metric monitors the message count in the queue to determine when to scale up or down, ensuring efficient resource allocation.

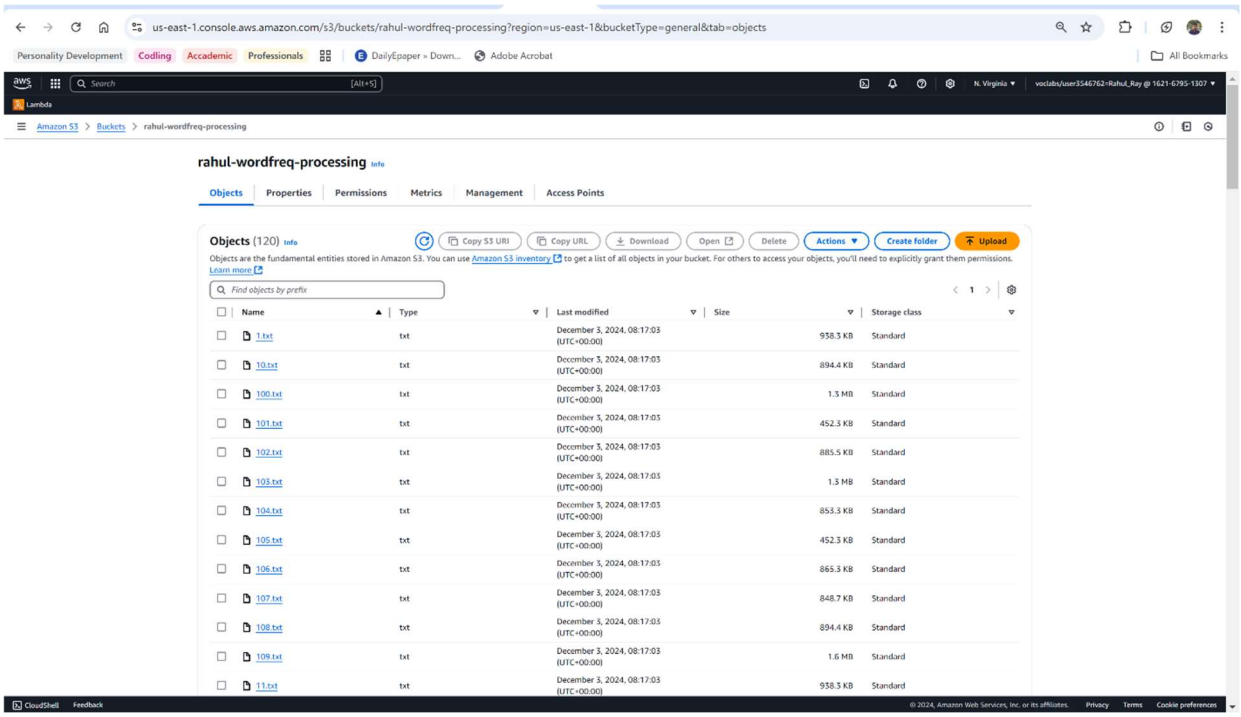


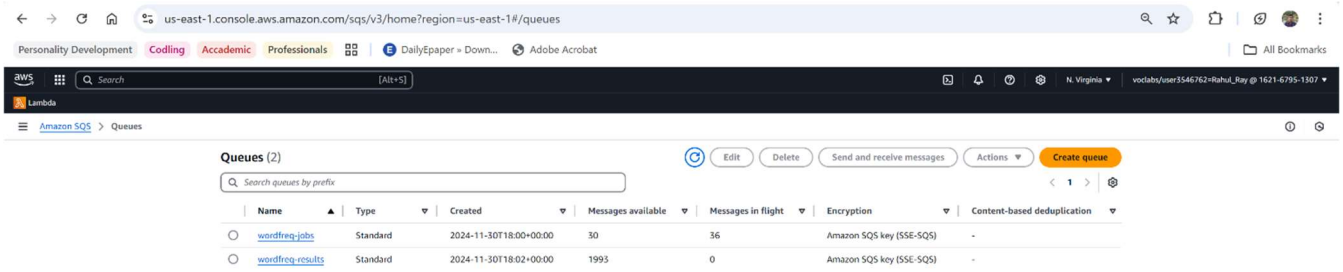
Task C - Perform Load Testing

After set up autoscaling infrastructure for WordFreq Application, it is time to check the load balance. All the file has been deleted from processing bucket and then we have Connect to one of your instances that in your Auto Scaling Group (via SSH connection)

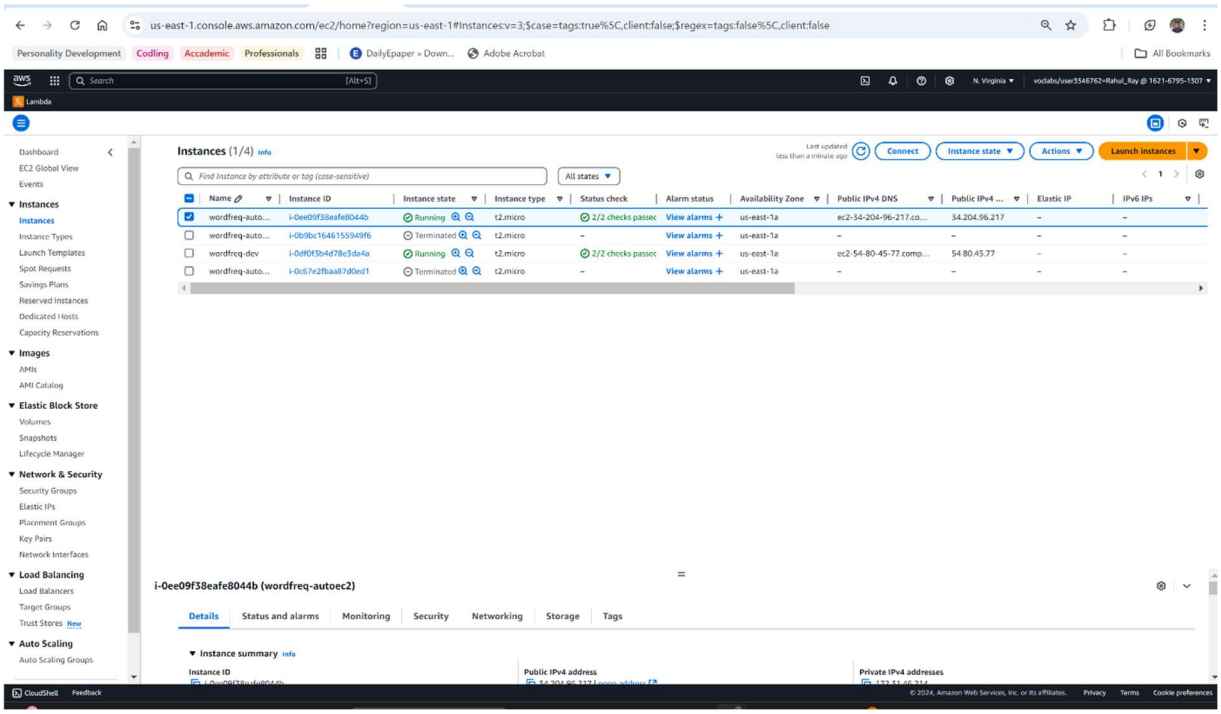


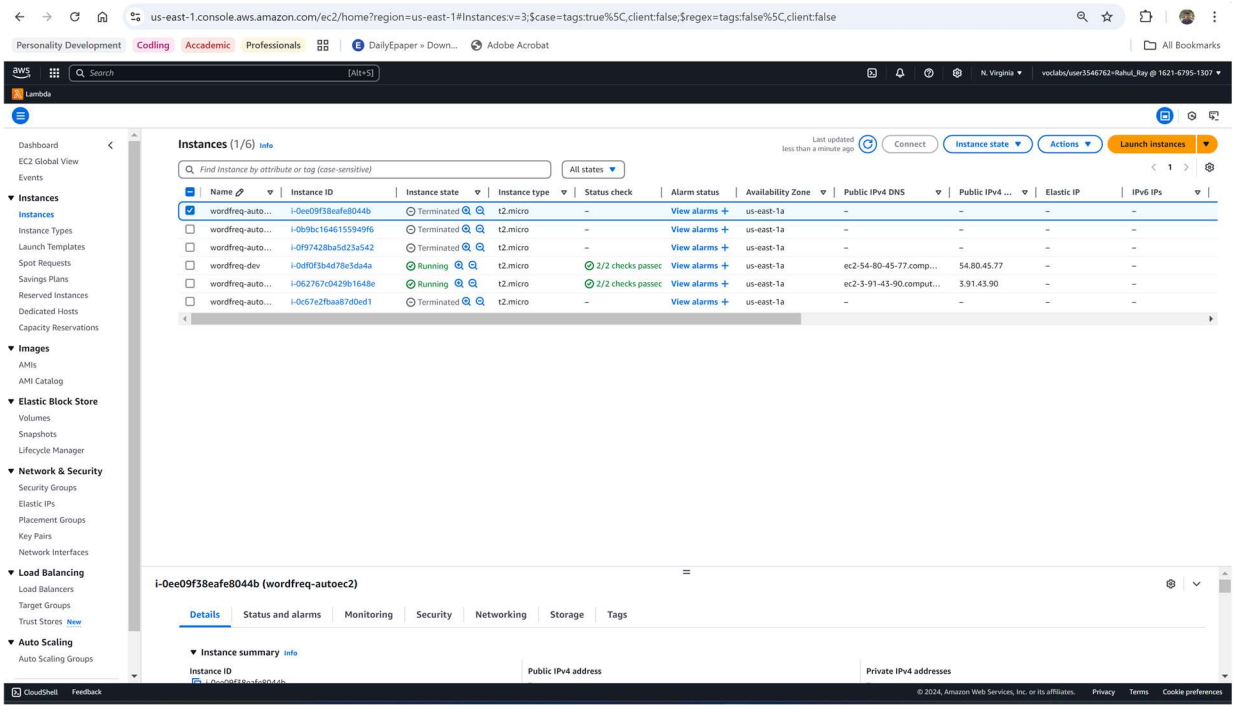
Screenshots of copied files in the S3 bucket, the SQS queue page showing message status, the Auto Scaling Group page showing instance status and the EC2 instance page showing launched / terminated instances during this process.



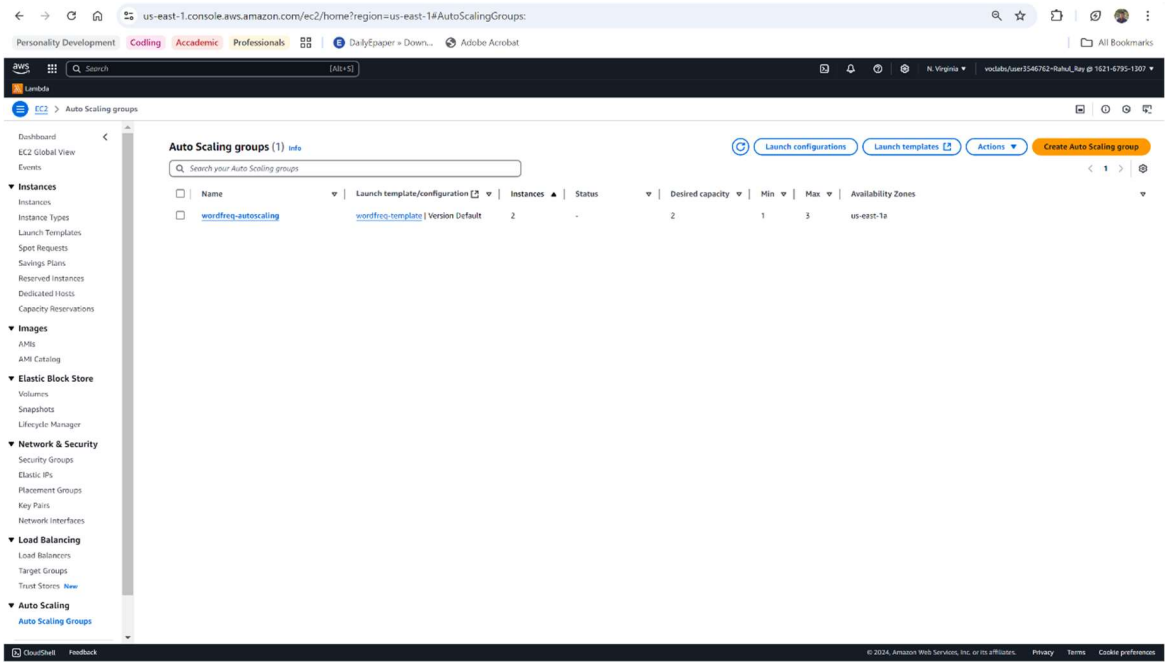


EC2 instance page during the auto scaling process

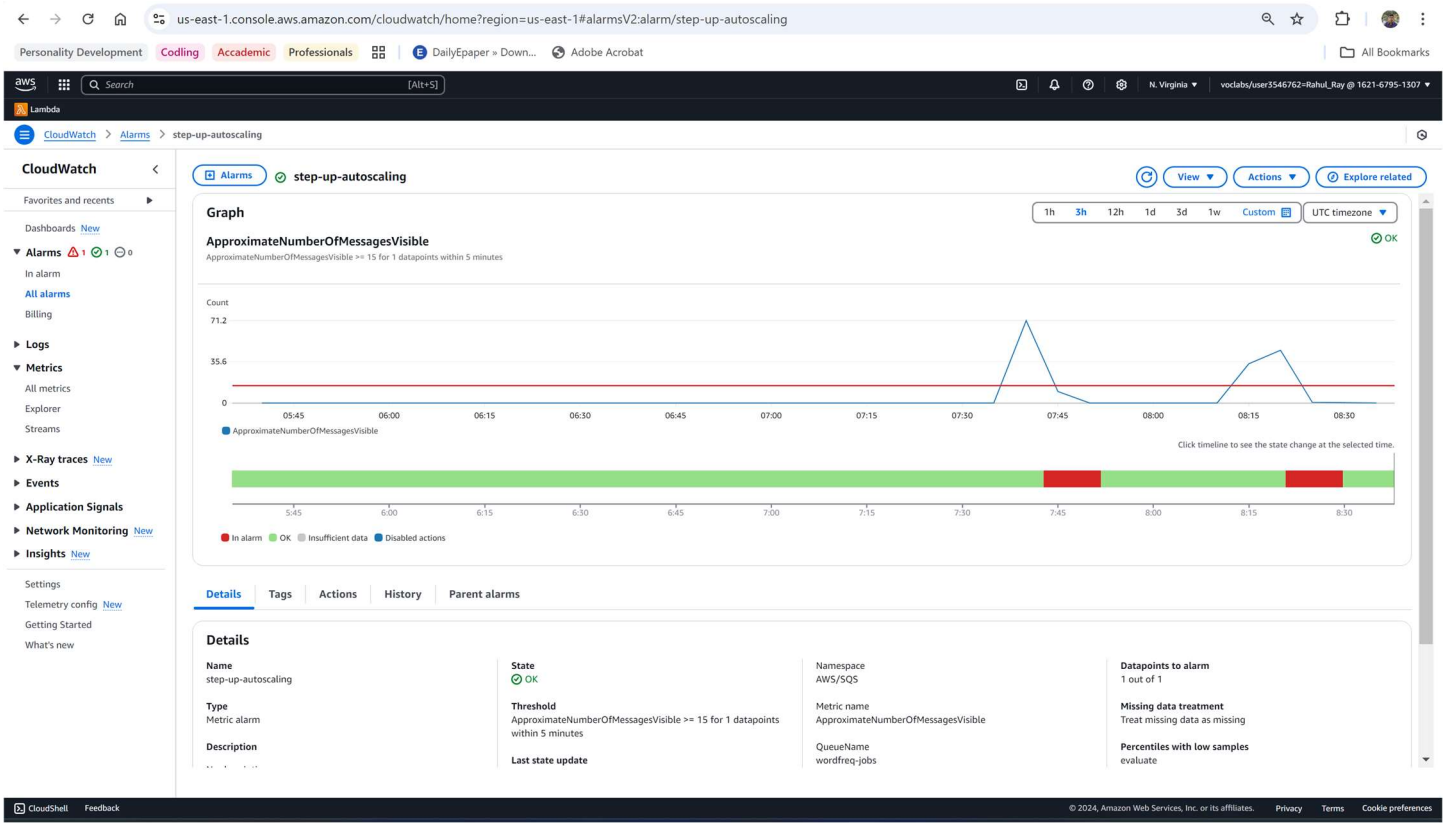




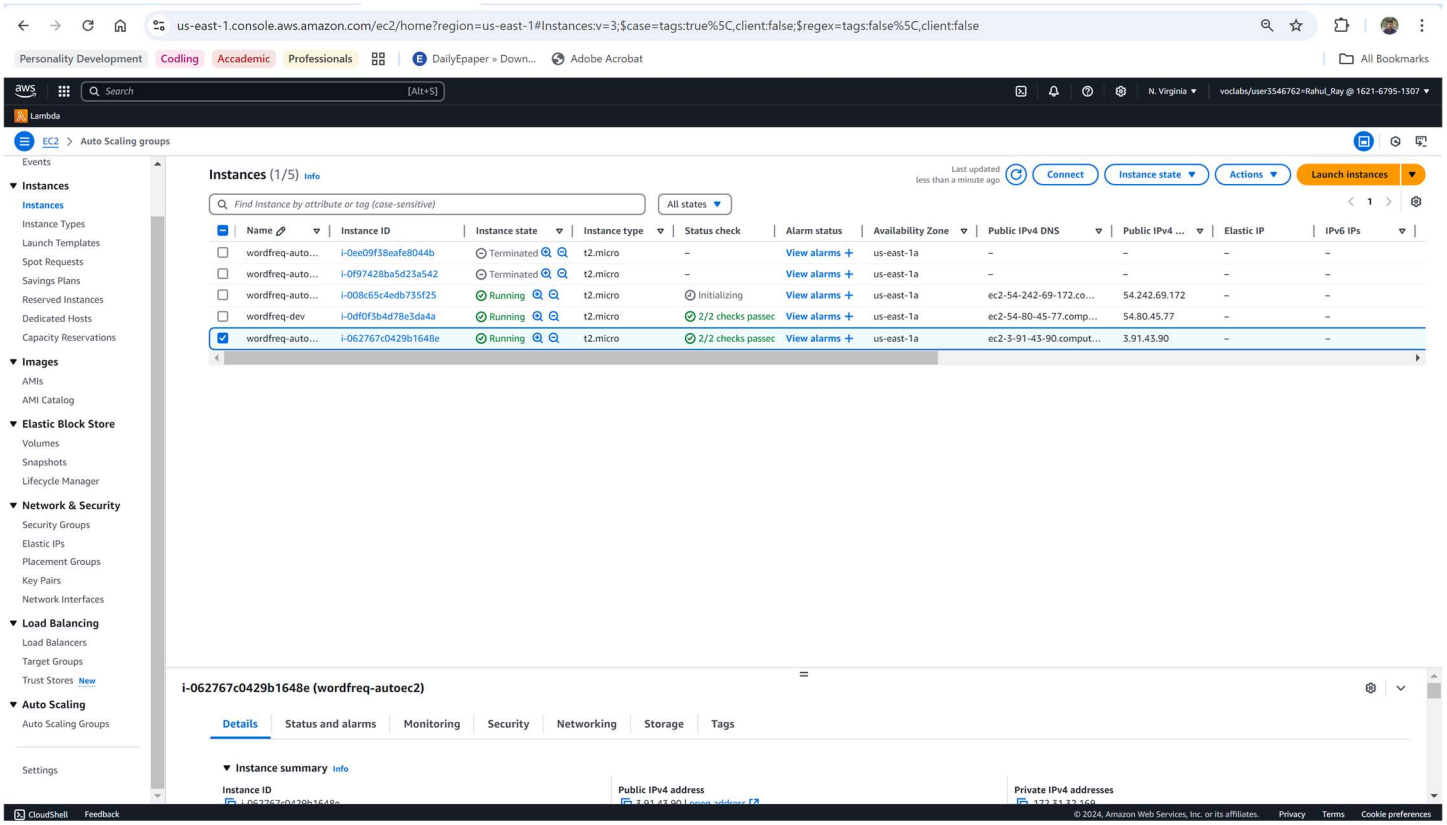
Auto scaling group page during the process:

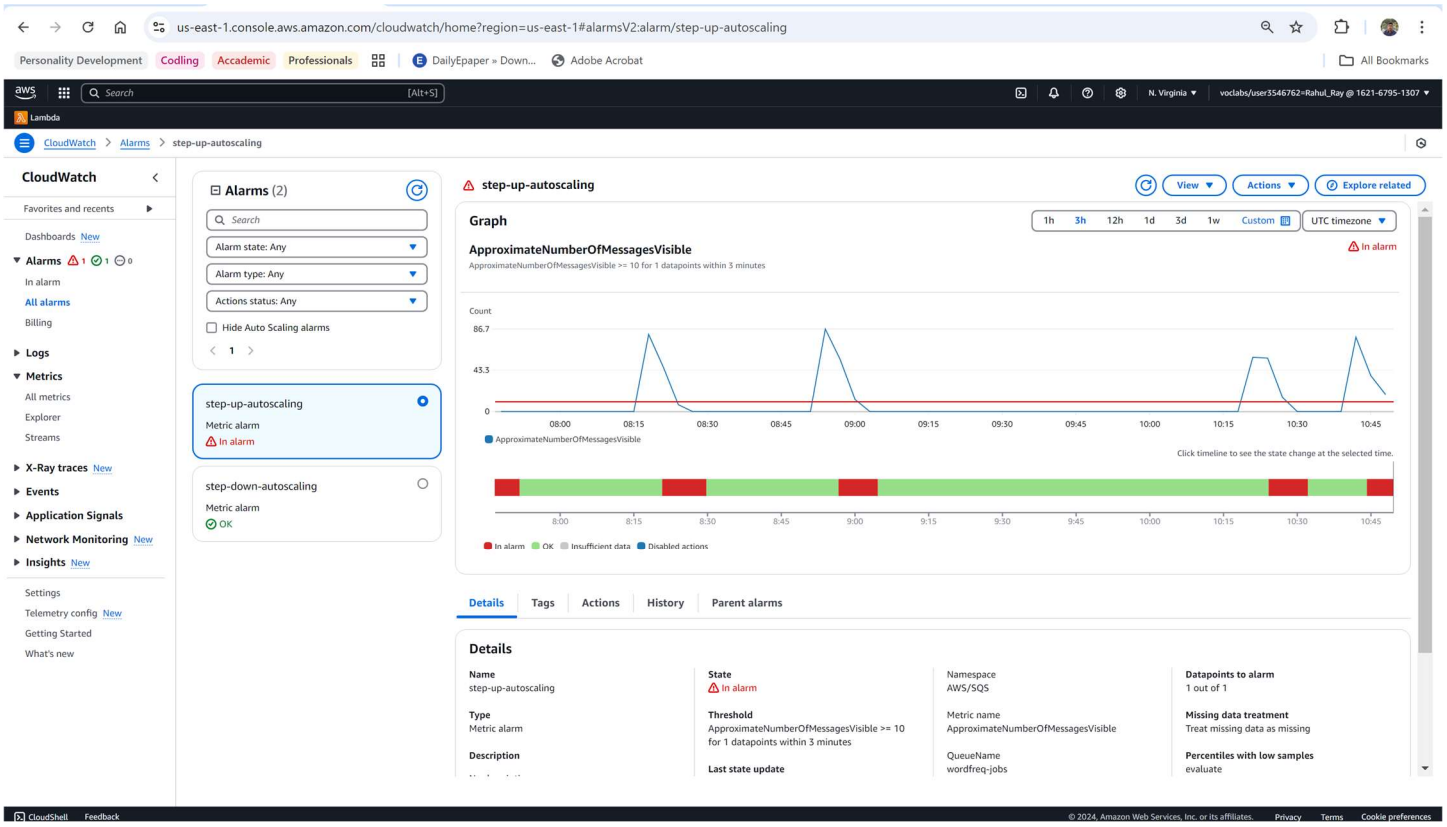


Observations in cloudwatch :



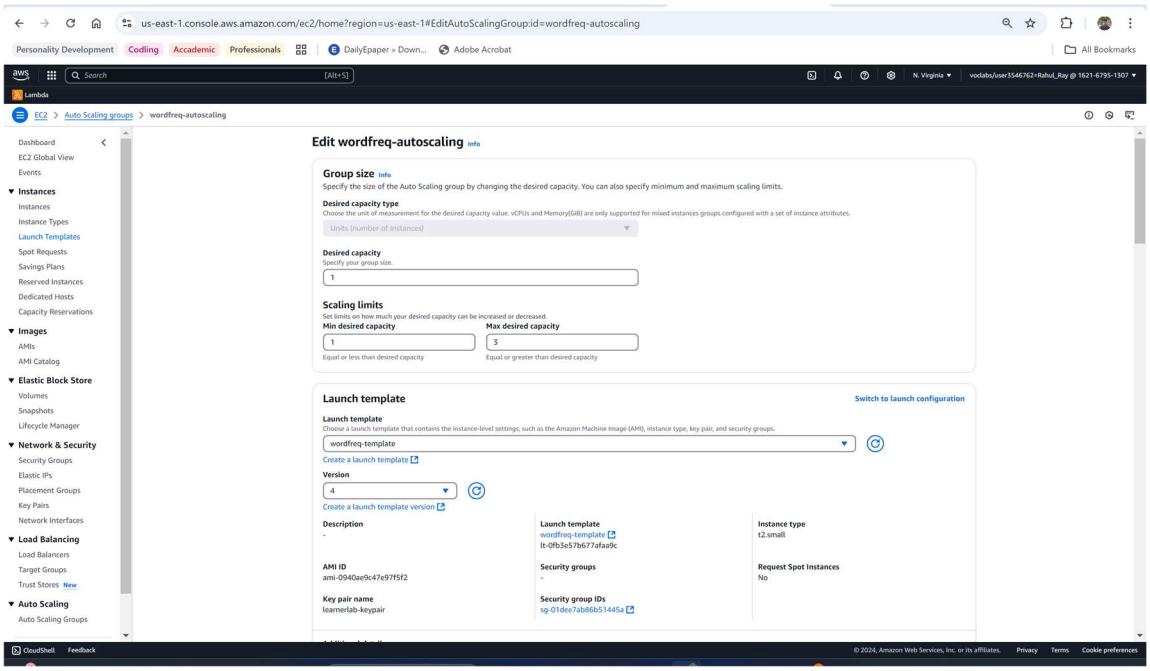
Now let’s Try to optimise the scaling operation, for example so that instances are launched quickly when required and terminated soon.

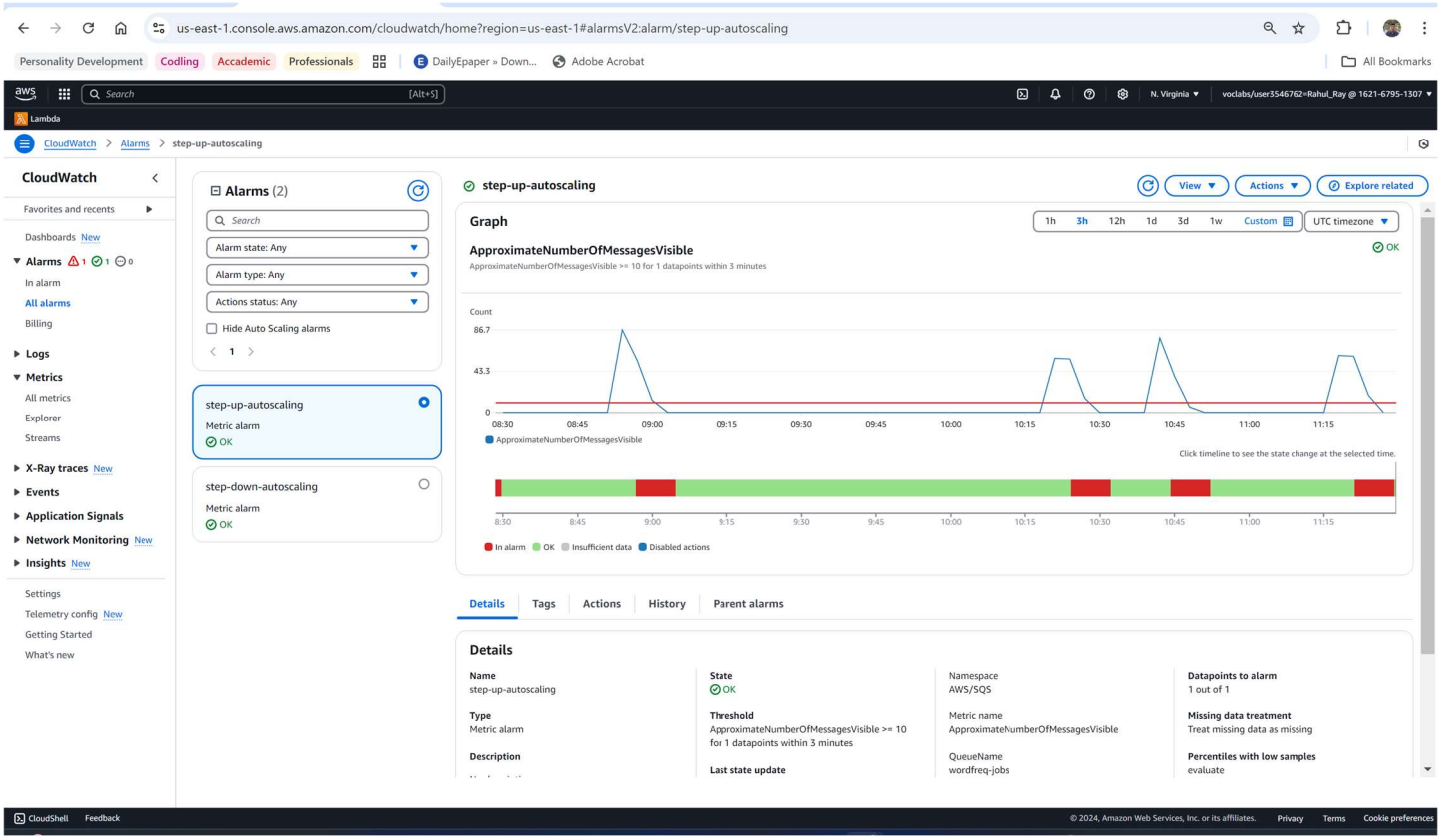




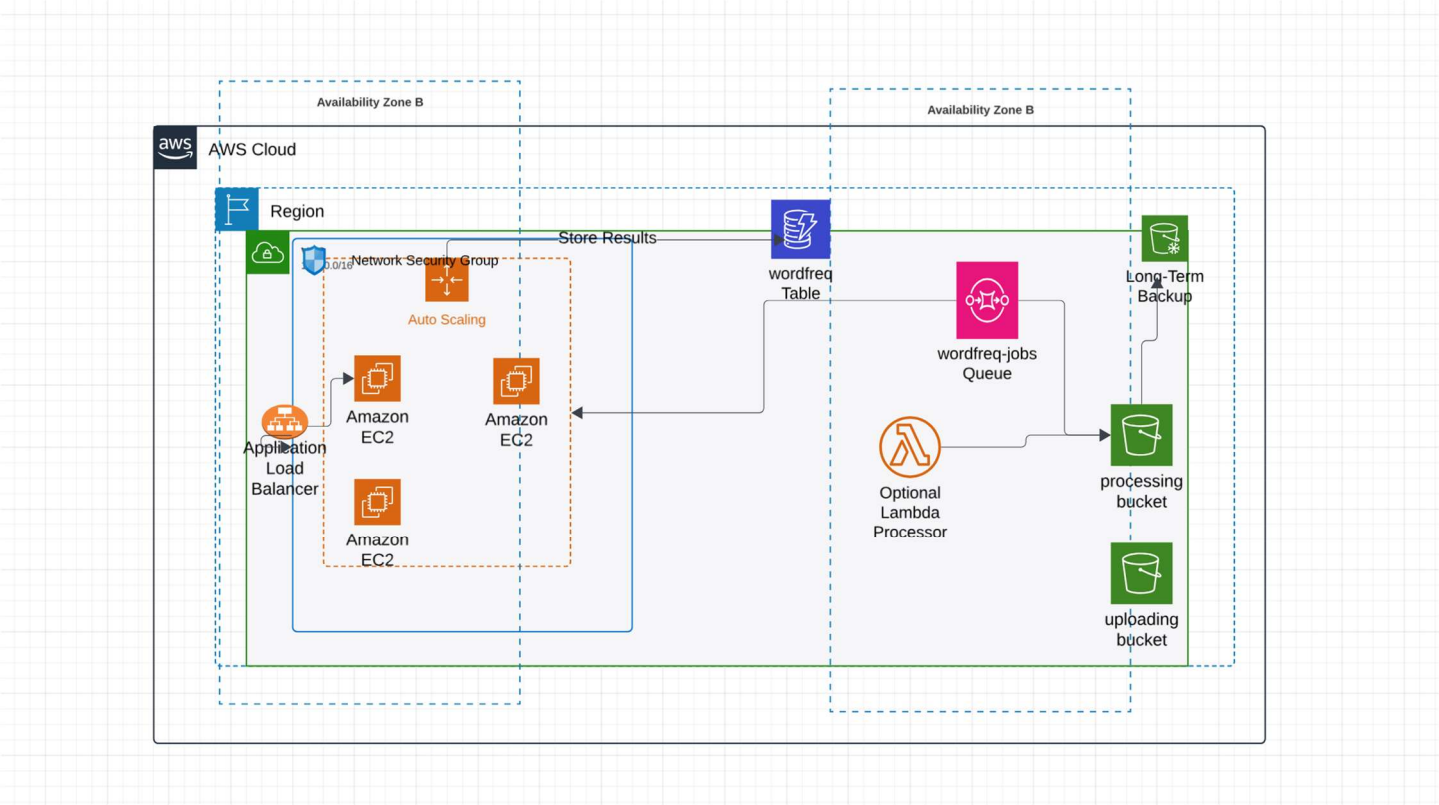
Let’s try using a few different EC2 instance types – with more CPU power, memory, etc-

Using Instance type : t2.micro





Task D - Optimise the WordFreq Architecture



The current architecture of the WordFreq application efficiently handles text file uploads, processing, and result storage. But the WordFreq application's current architecture has several week points. It is based on just the west, so if one zone of availability stops working then everything else is also affected. There is no long-term backup mechanism for data, running the risk of gradual loss over time. Its lack of use efficiency goes without saying, as EC2 instances must always be running which brings about uneccessarily nonzero costs during idle periods. This same laxity in construction security makes the system liable to unauthorized visits and data breaches, for instance through failure of encryption IAM role-based access or a VPC. It is clear that these inefficient areas need architectural improvement to enhance resistance to failure, achieve economy gains and make the system safer. We will re-designed architecture leveraging AWS services to meet these goals while keeping the application functionality unchanged.

- Increase Resilience and Availability
- Long-Term Backups
- Cost-Effective and Efficient for Occasional Use
- Prevent Unauthorised Access

Increase Resilience and Availability: we will use Elastic Load Balancer (ELB) to distribute the traffics across the multiple EC2 instances to prevent downtime. Also we can deploy instances across multiple availability zones, enable multi-region replication for data durability.

Long-Term Backups: Amazon S3 Glacier is cost effective storage for long term data archival, which we will use as long term backup option.

Cost-Effective and Efficient for Occasional Use:

- we will replace EC2 instance with Lambda as it is a serverless compute service that only charges for execution time and eliminating cost when idle. We will use lambda functions to process SQS massage instead of EC2 worker instances
- we will use EC2 spot instances, this is because spot instances offer significant cost savings (upto 90% lower) compared to on demand instances.
- Optimize auto scaling policies so that it minimise running instances during low usages and reduce cost.
- We will enable DynamoDB On-Demand mode as on demand mode charges only for actual reads and writes, making it ideal for intermittent usage.

Prevent Unauthorised Access: To enhance the security of the WordFreq application and prevent unauthorized access, we can implement following configurations

- Assign specific IAM role to EC2 instances, S3 buckets and other services.
- We will use **fine-grained policies** to grant only the permissions required for each role (e.g., read-only for S3 or write-only for DynamoDB).
- Isolate resources from public internet for improved security using virtual private clous (VPC) and we can use Use **NAT Gateways** to allow outbound internet traffic while blocking inbound access.
- Protect data at rest and transit to prevent unauthorized access enable server side Encryption
- Enable AWS WAF (Web Application Firewall, it will protect against common web exploits like SQL injection and DDoS attacks.
- Restrict Network Access with Security Groups. Control inbound and outbound traffic to AWS resources. Restrict outbound traffic to specific AWS resources or internet endpoints.

Task E – Further Improvements

Currently, WordFreq uses S3, EBS, EC2, and DynamoDB to input text files. This kind of system works, if not perfectly. By increasing the scale, though, the needs of your application may handle overwhelming workloads and more complex text-processing tasks without losing efficiency. We parsed two other data-processing services at the WordFreq user conference. One was for Apache Spark on Amazon EMR and the other one was AWS Glue. Each of these two alternative methods, when compared to those in existence and also presented in your current setup for WordFreq, are superior in reliability and speed.

I. Amazon EMR runs Apache Spark

Amazon EMR (Elastic MapReduce) launches Apache Spark. Spark is a distributed computing framework for handling large datasets. It can make use of the parallelism provided by multiple nodes, enabling it to handle very compute-intensive tasks very efficiently. As a result, data can be processed quickly across many nodes.

Benefits:

- **Scalability:** In parallel on the entire cluster of nodes, Spark processes data hundreds or thousands of times faster than one EC2 machine.
- **Performance:** Spark, with its in-memory computing, can speed up answer generation by orders of magnitude compared to all those huge files requiring long waits of disk input and output.
- **Flexibility:** Support for multiple data formats (e.g. JSON, Parquet, CSV). With S3 integration, data ingestion can become easy even when handling terabytes of data.

- **Fault Tolerance:** This distributed architecture means that once a node goes down, the next one can take over.
- **Cost-Effectiveness:** Amazon EMR allows you to use Spot Instances as Worker Nodes, thus saving money while maintaining high availability.

  With Spark running on Amazon EMR, WordFreq can now do even larger or more complex jobs. After all, a giant load of data will crank away calmly 24 hours a day and 7 days a week.

  II. AWS Glue

  AWS Glue was designed as a serverless data integration tool that moves the hassle to the cloud. It has managed ETL (Extract, Transform, Load) capacity suitable for applications like WordFreq.

  Merits:
- **Serverless Architecture:** With AWS Glue, you do not have to set up EC2 instances. Your application will automatically scale itself as necessary, depending on the workload.
- **Simplification of Process:** Glue can identify data schema and then turn your data straight from S3. This saves the need for special configuration.
- **Economical:** You pay only for the time where compute resources are actually being used. Glue's pay-as-you-go model is a good fit for constant, cost-effective duties, a far cry from normal processing tools or meter-based charging.
- **Integration:** Glue can talk to S3 and DynamoDB perfectly easily, as it can with other AWS services. No wonder it is becoming the main backbone of your system--instead of scattering everywhere else to check on these installations all over, Glue itself becomes the nerve center for control.
- **All Batch Welcome:** Glue is structured particularly well for batch treatment. This matches the characteristics of a job like WordFreq. It only works for weeks without really inputting, though.

  If WordFreq can use AWS Glue, that would reduce its existing resource consumptions to almost zero.