Large Scale Data Engineering Coursework

Section 1: Discussing whether the NHS and social care services should use public cloud for data processing

The National Health Service (NHS) and social care sector in the United Kingdom handle vast amounts of sensitive data, which must be processed rapidly, safely, and cost-effectively. Public cloud services offer a compelling response to these needs, with a solid infrastructure and increased features that boost data processing capabilities.. However, some situations warrant care. This article covers the benefits of using public cloud services for NHS and social care data processing, as enabled by Amazon Web Services (AWS) features, as well as instances in which public cloud use should be avoided.

There are several advantages to adopting public cloud services, including their unparalleled scalability. AWS Auto Scaling enables NHS and social care apps to automatically modify capacity to provide consistent and predictable performance at the lowest feasible cost. This functionality is critical for managing fluctuating workloads, such as during health emergencies or seasonal peaks, without requiring a large upfront investment in hardware.

Furthermore, public cloud services operate on a pay-as-you-go model, eliminating the requirement for significant capital investment in physical equipment. AWS offers cost management tools for monitoring and controlling spending, such as AWS Budgets and Cost Explorer. This financial flexibility is especially useful to the NHS and social care organisations, which often work within strict budget limits.

AWS provides a complete suite of security capabilities that meet NHS data protection regulations, including encryption at rest and in transit, Identity and Access Management (IAM), and regular security assessments. AWS Shield and AWS WAF (Web Application Firewall) services safeguard critical patient data from DDoS assaults and other threats while assuring its confidentiality, integrity, and availability.

AWS's worldwide infrastructure consists of numerous Availability Zones (AZs) inside each region, ensuring redundancy and fault tolerance. Services like Amazon RDS (Relational Database Service) and Amazon S3 (Simple Storage Service) guarantee data durability and high availability, which is essential for continuous access to health and social care information.

AWS provides many disaster recovery alternatives through services such as AWS Backup and AWS CloudEndure. These services allow NHS and social care businesses to create automatic backups and disaster recovery plans, ensuring that data is recovered quickly in the case of an incident, decreasing downtime and data loss.

However, there are certain situations in which the public cloud should be avoided. Regardless of the high security requirements of public cloud providers, certain types of sensitive data may be subject to unique legal or regulatory constraints that need on-premises storage. For example, some datasets might be restricted by regional data sovereignty laws, making public cloud storage impractical or illegal.

While public cloud services offer robust security, the highest levels of sensitive or classified data might require an additional layer of security that is more easily managed on-premises. This includes data that, if breached, could result in severe consequences for individuals or national security.

The reliance on internet connectivity for accessing cloud services can be a critical point of failure. In regions with unreliable or limited internet access, relying on public cloud services could impede access to crucial data and services, negatively impacting patient care and operational efficiency.

The use of public cloud services provides various benefits for NHS and social care data processing, including scalability, cost-effectiveness, strong security, high availability, and extensive disaster recovery capabilities. AWS features like Auto Scaling, IAM, multi-AZ deployments, and AWS Backup give the tools required to improve data processing capacity while maintaining data security and compliance with NHS requirements. However, there are several scenarios in which using

the public cloud may not be acceptable, such as meeting certain regulatory requirements, processing extremely sensitive data, and guaranteeing dependable internet access. By carefully analysing these considerations, NHS and social care organisations may make educated judgements about how to maximise the benefits of public cloud services while minimising possible hazards.

Section 2: Scaling the WordFreq Application

Task A – Application Summary

The aim of this paper is to deliver a thorough explanation of my attempts to create a scalable and effective environment for the WordFreq application. Amazon Web Services' (AWS) flexible architecture was used to develop, implement and administrate a cloud computing application, specifically focussing on the architecture of the WordFreq application using the AWS Learner Lab. I used various AWS features and tools to do this, such as the Simple Queue Service (SQS), Elastic Compute Cloud (EC2) and Simple Storage Service (S3). The WordFreq application is designed such that it automatically monitors and modifies the capacity of its serverless application through the use of Amazon CloudWatch, based on the number of messages in an SQS queue.

SQS is a fully managed and scalable service, which receives and queues messages from other applications or systems. Amazon CloudWatch was used to analyse the SQS queue, continuously keeping an eye on queue-related metrics, such as the total number of messages in the queue. There were two CloudWatch alerts which were set up to track the SQS queue's "ApproximateNumberOfMessagesVisible" statistic, which counts the number of messages that have not been removed and are still available for processing. One warning was designed to appear when the quantity of visible messages fell below 20 and the other appeared when the quantity surpassed 50. These alerts triggered auto-scaling activities such that EC2 instances were terminated upon breach of the lower threshold and activated upon breach of the upper threshold, allowing the application to cope with reducing or increasing activity loads.

The messages in the SQS queue were processed by the EC2 instances within the auto-scaling group. The instances ran scripts or software that examine the signals based on the logic of the

application. The final results were subsequently delivered to DynamoDB, a NoSQL database service that was fully monitored and provided the storage for processed results. These results remain in DynamoDB for long-term storage and access from other tools and services.

I designed this cloud architecture to automatically modify the number of EC2 instances in response to changes in the SQS queue's message count. It ensured dynamic increasing or decreasing of the number of instances to handle the workload based on demand. The procedure of auto-scaling greatly aids the system by guaranteeing the optimal use of resources and maintaining system responsiveness as message volumes fluctuate.

Task B – Design and Implementation of Auto-scaling

The first step of my auto-scaling design and implementation was to access the auto-scaling groups tab within the EC2 platform. I clicked “create auto scaling group" and was prompted to generate a launch template before continuing. A launch template was necessary in this scenario because it allowed me to define the key details for configuring and launching an EC2 instance, including the instance type, Amazon Machine Image (AMI) ID, security groups and key pair. I named my template “rorybennett_autoscaling” and identified the latest instance of the wordfreq-dev AMI that was produced. I used a “t2.micro” instance type, selected “learnerlab-keypair” as my PEM key and “wordfreq-dev-sg” as my security group. In the advanced details section of the setup procedure I enabled detailed CloudWatch monitoring data and created an IAM instance profile, set up as “EMR_EC2_defaultrole”. Finally, I clicked "create launch template" which generated the template and allowed it to be attached to my auto-scaling group.

With the this in place I was able to create my auto-scaling group, named “rorybennett_autoscalinggroup”, and attach version 1 of my launch template. In order to create the virtual private network I had to choose an available Virtual Private Cloud alongside the availability zones which I desired. I also made sure to enable group matrices collection inside CloudWatch. Specify the required capacity for auto scaling instances. Finally, I established a desired and minimum capacity of 1 EC2 instance with a maximum capacity of 5. Before clarifying the correct

implementation of my auto-scaling group settings, I ensured not to include a load balancer in this exercise as it was not required.

The formation of the auto-scaling group automatically generated a new EC2 instance and I used my computer's Terminal (SSH) application to connect to it. The purpose of connecting to this instance at this point was to test the functionality of my worker by processing a request, verifying the capability of the new autoscaling system.

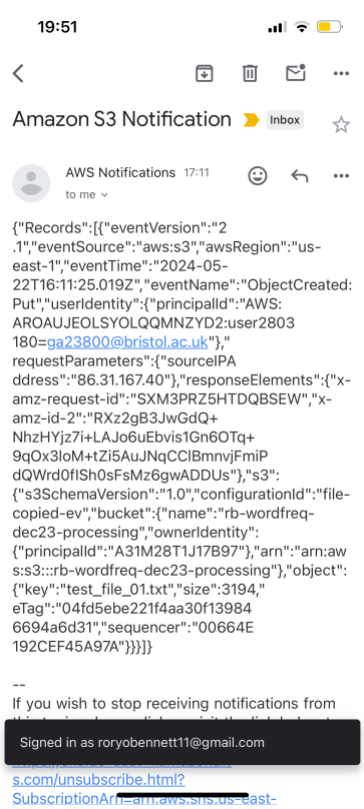
Task C – Performing Load Testing and Results

In order to perform load testing and view the results, I needed to set up CloudWatch alarms and dynamic scaling policies. Dynamic scaling policies are triggered by CloudWatch alarms, which can be created using the CloudWatch service. I investigated all of the available metrics for each of the AWS services that I was using in order to choose the most suitable one to generate alarms. I decided to use "Approximate Number of Messages Visible" within the SQS list of measures, a metric which gives users an estimate of how many messages are displayed in the queue. Using this as my measure, I designed one alarm to be triggered when the approximate number of messages visible was greater than 50 and another to be triggered when they were less than 20.

Once the alarms were in place, I accessed the auto-scaling group (designed in Task B) to design the dynamic scaling policies via the "Automatic scaling" tab. I chose to use a simple scaling rule for removing idle EC2 instances (scaling in) and a step scaling rule for adding instances (scaling out) when the workload of the current ones becomes heightened. For the latter, I navigated to "create dynamic scaling policies" and selected "step scaling" when prompted. I named it "max_message" and proceeded to select the CloudWatch alarm that alerts the user when the approximate number of messages visible exceeded 50. I set the policy to add one EC2 instance when this threshold was breached and set 120 seconds as the warm up time between each step so that instances are launched quickly when required but not at unnecessary quantities. The steps for simple scaling were much the same with the key differences being that you select "simple scaling" when prompted and select the alarm that alerts the user when the approximate number of messages visible was less than 20. I set the

policy name to “min_messagereceived” and configured it to remove one instance when this threshold is breached. There was a 120 second delay before allowing another scaling activity, meaning that instances were terminated soon (but not immediately) when not required.

The images below show the successful functionality of my application architecture, with an example of an Amazon S3 Notification, the files being copied from the uploading to processing buckets, the SQS queue page showing the status of the messages and the EC2 and auto-scaling group pages showing the launched and terminated instances during the process.



19:51

Amazon S3 Notification Inbox

AWS Notifications 17:11
to me

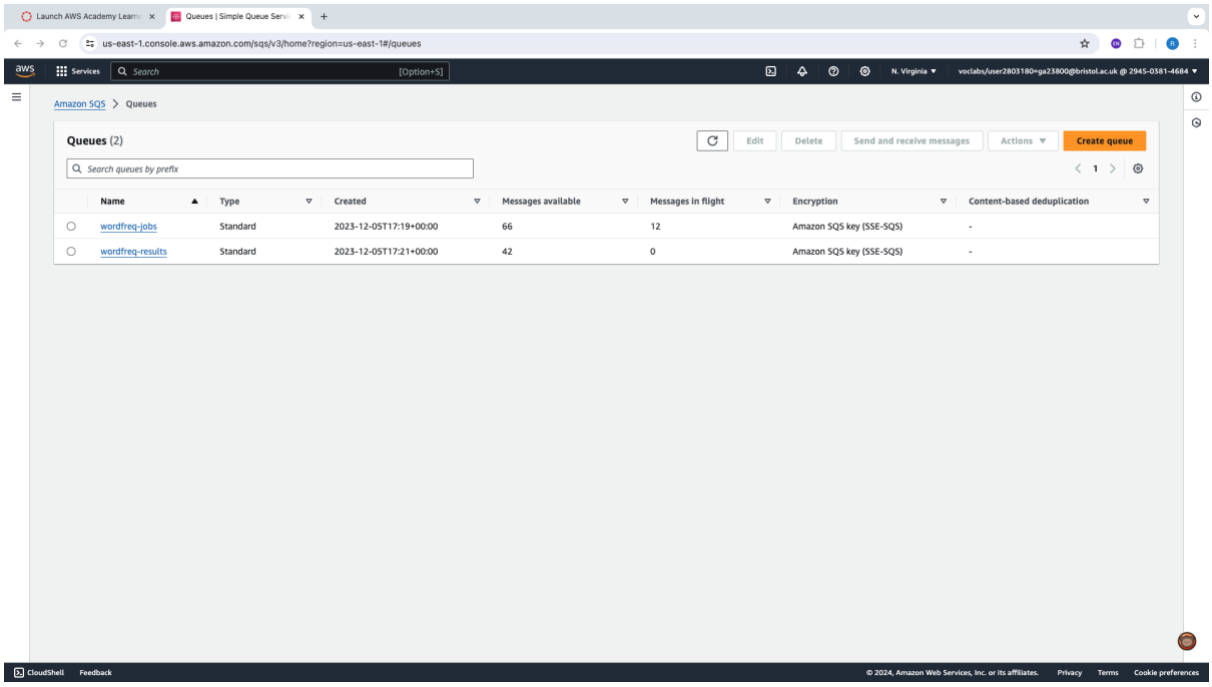
{"Records":[{"eventVersion":"2
.1","eventSource":"aws:s3","awsRegion":"us-
east-1","eventTime":"2024-05-
22T16:11:25.019Z","eventName":"ObjectCreated:
Put","userIdentity":{"principalId":"AWS:
AROAUJEOLSYOLQQMNZYD2:user2803
180=ga23800@bristol.ac.uk"},"
requestParameters":{"sourceIPA
ddress":"86.31.167.40"},"responseElements":{"x-
amz-request-id":"SXM3PRZ5HTDQBSEW","x-
amz-id-2":"RXz2gB3JwGdQ+
NhzHYjz7i+LAJo6uEbvis1Gn6OTq+
9qOx3loM+tZi5AuJNqCCIBmnvjFmiP
dQWrd0fISh0sFsMz6gwADDUs"},"s3":
{"s3SchemaVersion":"1.0","configurationId":"file-
copied-ev","bucket":{"name":"rb-wordfreq-
dec23-processing","ownerIdentity":
{"principalId":"A31M28T1J17B97"},"arn":"arn:aw
s:s3:::rb-wordfreq-dec23-processing"},"object":
{"key":"test_file_01.txt","size":3194,"
eTag":"04fd5ebe221f4aa30f13984
6694a6d31","sequencer":"00664E
192CEF45A97A"}}}]}

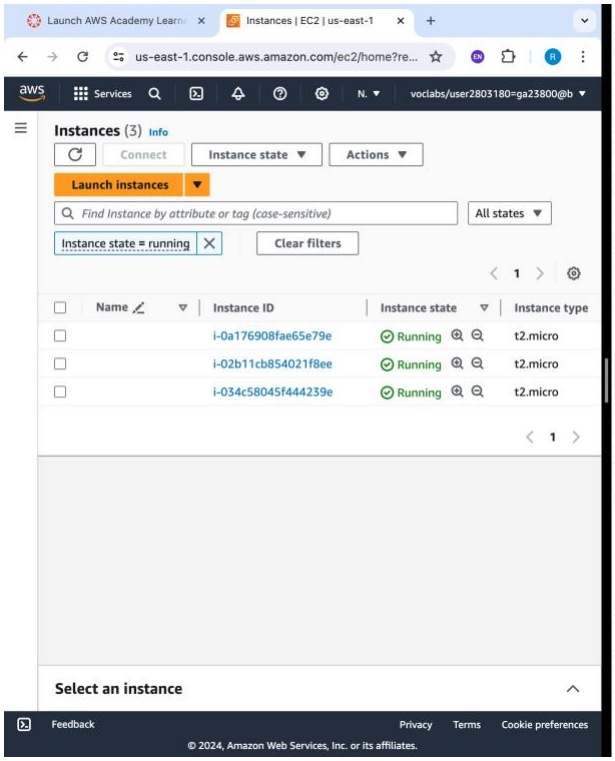
--
If you wish to stop receiving notifications from

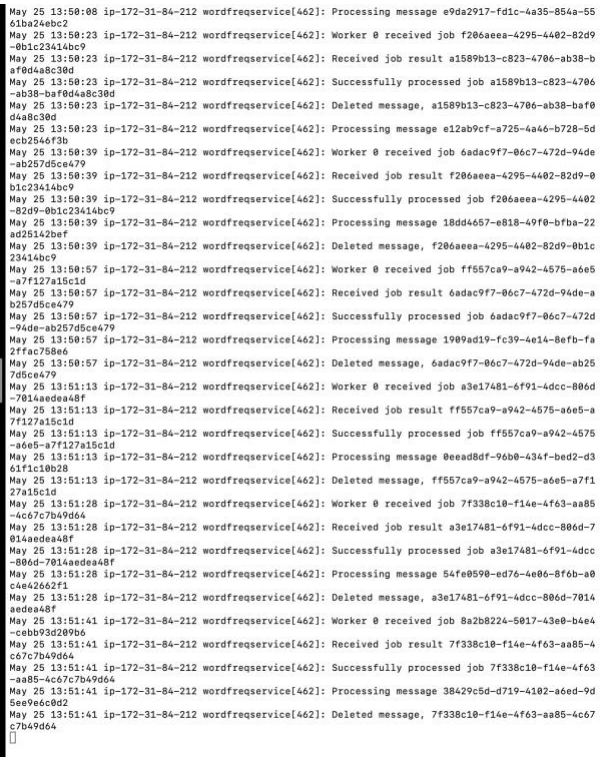
Signed in as roryobennett11@gmail.com

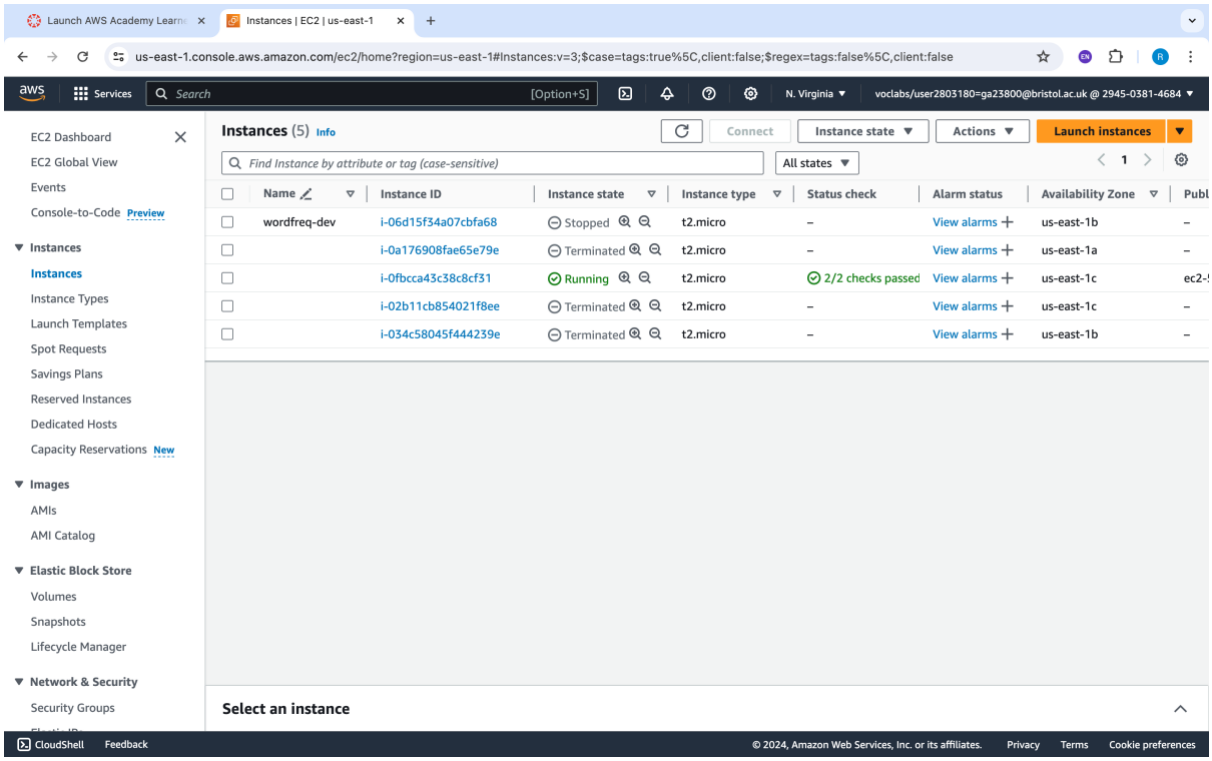
s.com/unsubscribe.html?
SubscriptionArn=arn:aws:sns:us-east-

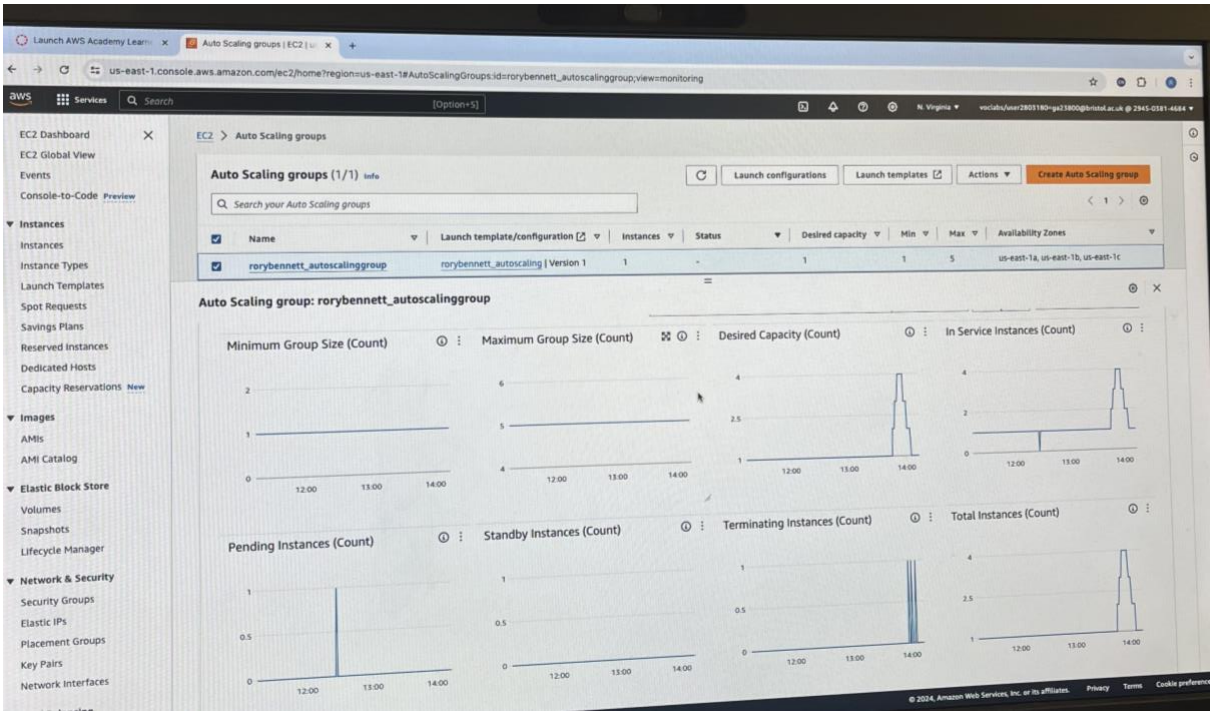
```
copy: s3://rb-wordfreq-dec23-uploading/43.txt to s3://rb-wordfreq-dec23-processing/43.txt
copy: s3://rb-wordfreq-dec23-uploading/42.txt to s3://rb-wordfreq-dec23-processing/42.txt
copy: s3://rb-wordfreq-dec23-uploading/47.txt to s3://rb-wordfreq-dec23-processing/47.txt
copy: s3://rb-wordfreq-dec23-uploading/48.txt to s3://rb-wordfreq-dec23-processing/48.txt
copy: s3://rb-wordfreq-dec23-uploading/46.txt to s3://rb-wordfreq-dec23-processing/46.txt
copy: s3://rb-wordfreq-dec23-uploading/5.txt to s3://rb-wordfreq-dec23-processing/5.txt
copy: s3://rb-wordfreq-dec23-uploading/53.txt to s3://rb-wordfreq-dec23-processing/53.txt
copy: s3://rb-wordfreq-dec23-uploading/44.txt to s3://rb-wordfreq-dec23-processing/44.txt
copy: s3://rb-wordfreq-dec23-uploading/54.txt to s3://rb-wordfreq-dec23-processing/54.txt
copy: s3://rb-wordfreq-dec23-uploading/51.txt to s3://rb-wordfreq-dec23-processing/51.txt
copy: s3://rb-wordfreq-dec23-uploading/49.txt to s3://rb-wordfreq-dec23-processing/49.txt
copy: s3://rb-wordfreq-dec23-uploading/55.txt to s3://rb-wordfreq-dec23-processing/55.txt
copy: s3://rb-wordfreq-dec23-uploading/50.txt to s3://rb-wordfreq-dec23-processing/50.txt
copy: s3://rb-wordfreq-dec23-uploading/56.txt to s3://rb-wordfreq-dec23-processing/56.txt
copy: s3://rb-wordfreq-dec23-uploading/58.txt to s3://rb-wordfreq-dec23-processing/58.txt
copy: s3://rb-wordfreq-dec23-uploading/57.txt to s3://rb-wordfreq-dec23-processing/57.txt
copy: s3://rb-wordfreq-dec23-uploading/59.txt to s3://rb-wordfreq-dec23-processing/59.txt
copy: s3://rb-wordfreq-dec23-uploading/52.txt to s3://rb-wordfreq-dec23-processing/52.txt
copy: s3://rb-wordfreq-dec23-uploading/60.txt to s3://rb-wordfreq-dec23-processing/60.txt
copy: s3://rb-wordfreq-dec23-uploading/6.txt to s3://rb-wordfreq-dec23-processing/6.txt
copy: s3://rb-wordfreq-dec23-uploading/65.txt to s3://rb-wordfreq-dec23-processing/65.txt
copy: s3://rb-wordfreq-dec23-uploading/63.txt to s3://rb-wordfreq-dec23-processing/63.txt
copy: s3://rb-wordfreq-dec23-uploading/61.txt to s3://rb-wordfreq-dec23-processing/61.txt
copy: s3://rb-wordfreq-dec23-uploading/66.txt to s3://rb-wordfreq-dec23-processing/66.txt
copy: s3://rb-wordfreq-dec23-uploading/68.txt to s3://rb-wordfreq-dec23-processing/68.txt
copy: s3://rb-wordfreq-dec23-uploading/62.txt to s3://rb-wordfreq-dec23-processing/62.txt
copy: s3://rb-wordfreq-dec23-uploading/69.txt to s3://rb-wordfreq-dec23-processing/69.txt
copy: s3://rb-wordfreq-dec23-uploading/7.txt to s3://rb-wordfreq-dec23-processing/7.txt
copy: s3://rb-wordfreq-dec23-uploading/71.txt to s3://rb-wordfreq-dec23-processing/71.txt
copy: s3://rb-wordfreq-dec23-uploading/70.txt to s3://rb-wordfreq-dec23-processing/70.txt
copy: s3://rb-wordfreq-dec23-uploading/67.txt to s3://rb-wordfreq-dec23-processing/67.txt
copy: s3://rb-wordfreq-dec23-uploading/72.txt to s3://rb-wordfreq-dec23-processing/72.txt
copy: s3://rb-wordfreq-dec23-uploading/64.txt to s3://rb-wordfreq-dec23-processing/64.txt
copy: s3://rb-wordfreq-dec23-uploading/74.txt to s3://rb-wordfreq-dec23-processing/74.txt
copy: s3://rb-wordfreq-dec23-uploading/73.txt to s3://rb-wordfreq-dec23-processing/73.txt
copy: s3://rb-wordfreq-dec23-uploading/77.txt to s3://rb-wordfreq-dec23-processing/77.txt
copy: s3://rb-wordfreq-dec23-uploading/78.txt to s3://rb-wordfreq-dec23-processing/78.txt
copy: s3://rb-wordfreq-dec23-uploading/75.txt to s3://rb-wordfreq-dec23-processing/75.txt
copy: s3://rb-wordfreq-dec23-uploading/79.txt to s3://rb-wordfreq-dec23-processing/79.txt
copy: s3://rb-wordfreq-dec23-uploading/76.txt to s3://rb-wordfreq-dec23-processing/76.txt
copy: s3://rb-wordfreq-dec23-uploading/80.txt to s3://rb-wordfreq-dec23-processing/80.txt
copy: s3://rb-wordfreq-dec23-uploading/8.txt to s3://rb-wordfreq-dec23-processing/8.txt
copy: s3://rb-wordfreq-dec23-uploading/81.txt to s3://rb-wordfreq-dec23-processing/81.txt
copy: s3://rb-wordfreq-dec23-uploading/83.txt to s3://rb-wordfreq-dec23-processing/83.txt
copy: s3://rb-wordfreq-dec23-uploading/82.txt to s3://rb-wordfreq-dec23-processing/82.txt
copy: s3://rb-wordfreq-dec23-uploading/84.txt to s3://rb-wordfreq-dec23-processing/84.txt
copy: s3://rb-wordfreq-dec23-uploading/85.txt to s3://rb-wordfreq-dec23-processing/85.txt
copy: s3://rb-wordfreq-dec23-uploading/86.txt to s3://rb-wordfreq-dec23-processing/86.txt
copy: s3://rb-wordfreq-dec23-uploading/87.txt to s3://rb-wordfreq-dec23-processing/87.txt
copy: s3://rb-wordfreq-dec23-uploading/89.txt to s3://rb-wordfreq-dec23-processing/89.txt
copy: s3://rb-wordfreq-dec23-uploading/92.txt to s3://rb-wordfreq-dec23-processing/92.txt
copy: s3://rb-wordfreq-dec23-uploading/93.txt to s3://rb-wordfreq-dec23-processing/93.txt
copy: s3://rb-wordfreq-dec23-uploading/9.txt to s3://rb-wordfreq-dec23-processing/9.txt
copy: s3://rb-wordfreq-dec23-uploading/88.txt to s3://rb-wordfreq-dec23-processing/88.txt
copy: s3://rb-wordfreq-dec23-uploading/94.txt to s3://rb-wordfreq-dec23-processing/94.txt
copy: s3://rb-wordfreq-dec23-uploading/91.txt to s3://rb-wordfreq-dec23-processing/91.txt
copy: s3://rb-wordfreq-dec23-uploading/95.txt to s3://rb-wordfreq-dec23-processing/95.txt
copy: s3://rb-wordfreq-dec23-uploading/90.txt to s3://rb-wordfreq-dec23-processing/90.txt
copy: s3://rb-wordfreq-dec23-uploading/96.txt to s3://rb-wordfreq-dec23-processing/96.txt
copy: s3://rb-wordfreq-dec23-uploading/99.txt to s3://rb-wordfreq-dec23-processing/99.txt
copy: s3://rb-wordfreq-dec23-uploading/97.txt to s3://rb-wordfreq-dec23-processing/97.txt
copy: s3://rb-wordfreq-dec23-uploading/98.txt to s3://rb-wordfreq-dec23-processing/98.txt
ubuntu@ip-172-31-84-111:~$
```




Launch AWS Academy Learn
Instances | EC2 | us-east-1
us-east-1.console.aws.amazon.com/ec2/home?re...
Services
N.
voclabs/user2803180=ga23800@b
Instances (3) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
i-0a176908fae65e79e
Running
t2.micro
i-02b11cb854021f8ee
Running
t2.micro
i-034c58045f444239e
Running
t2.micro
Select an instance
Feedback
Privacy
Terms
Cookie preferences
© 2024, Amazon Web Services, Inc. or its affiliates.


May 25 13:50:08 ip-172-31-84-212 wordfreqservice[462]: Processing message e9da2917-fd1c-4a35-854a-5561ba24ebc2
May 25 13:50:23 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job f206aeea-4295-4402-82d9-0b1c23414bc9
May 25 13:50:23 ip-172-31-84-212 wordfreqservice[462]: Received job result a1589b13-c823-4706-ab38-baf0d4a8c30d
May 25 13:50:23 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job a1589b13-c823-4706-ab38-baf0d4a8c30d
May 25 13:50:23 ip-172-31-84-212 wordfreqservice[462]: Deleted message, a1589b13-c823-4706-ab38-baf0d4a8c30d
May 25 13:50:23 ip-172-31-84-212 wordfreqservice[462]: Processing message e12ab9cf-a725-4a46-b728-5decb2546f3b
May 25 13:50:39 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job 6adac9f7-06c7-472d-94de-ab257d5ce479
May 25 13:50:39 ip-172-31-84-212 wordfreqservice[462]: Received job result f206aeea-4295-4402-82d9-0b1c23414bc9
May 25 13:50:39 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job f206aeea-4295-4402-82d9-0b1c23414bc9
May 25 13:50:39 ip-172-31-84-212 wordfreqservice[462]: Processing message 18dd4657-e818-49f0-bfba-22ad25142bef
May 25 13:50:39 ip-172-31-84-212 wordfreqservice[462]: Deleted message, f206aeea-4295-4402-82d9-0b1c23414bc9
May 25 13:50:57 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job ff557ca9-a942-4575-a6e5-a7f127a15c1d
May 25 13:50:57 ip-172-31-84-212 wordfreqservice[462]: Received job result 6adac9f7-06c7-472d-94de-ab257d5ce479
May 25 13:50:57 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job 6adac9f7-06c7-472d-94de-ab257d5ce479
May 25 13:50:57 ip-172-31-84-212 wordfreqservice[462]: Processing message 1909ad19-fc39-4e14-8efb-fa2ffac758e6
May 25 13:50:57 ip-172-31-84-212 wordfreqservice[462]: Deleted message, 6adac9f7-06c7-472d-94de-ab257d5ce479
May 25 13:51:13 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job a3e17481-6f91-4dcc-806d-7014aedea48f
May 25 13:51:13 ip-172-31-84-212 wordfreqservice[462]: Received job result ff557ca9-a942-4575-a6e5-a7f127a15c1d
May 25 13:51:13 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job ff557ca9-a942-4575-a6e5-a7f127a15c1d
May 25 13:51:13 ip-172-31-84-212 wordfreqservice[462]: Processing message 0eead8df-96b0-434f-bed2-d361f1c10b28
May 25 13:51:13 ip-172-31-84-212 wordfreqservice[462]: Deleted message, ff557ca9-a942-4575-a6e5-a7f127a15c1d
May 25 13:51:28 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job 7f338c10-f14e-4f63-aa85-4c67c7b49d64
May 25 13:51:28 ip-172-31-84-212 wordfreqservice[462]: Received job result a3e17481-6f91-4dcc-806d-7014aedea48f
May 25 13:51:28 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job a3e17481-6f91-4dcc-806d-7014aedea48f
May 25 13:51:28 ip-172-31-84-212 wordfreqservice[462]: Processing message 54fe0590-ed76-4e06-8f6b-a0c4e42662f1
May 25 13:51:28 ip-172-31-84-212 wordfreqservice[462]: Deleted message, a3e17481-6f91-4dcc-806d-7014aedea48f
May 25 13:51:41 ip-172-31-84-212 wordfreqservice[462]: Worker 0 received job 8a2b8224-5017-43e0-b4e4-cebb93d209b6
May 25 13:51:41 ip-172-31-84-212 wordfreqservice[462]: Received job result 7f338c10-f14e-4f63-aa85-4c67c7b49d64
May 25 13:51:41 ip-172-31-84-212 wordfreqservice[462]: Successfully processed job 7f338c10-f14e-4f63-aa85-4c67c7b49d64
May 25 13:51:41 ip-172-31-84-212 wordfreqservice[462]: Processing message 38429c5d-d719-4102-a6ed-9d5ee9e6c0d2
May 25 13:51:41 ip-172-31-84-212 wordfreqservice[462]: Deleted message, 7f338c10-f14e-4f63-aa85-4c67c7b49d64


Launch AWS Academy Learn
Instances | EC2 | us-east-1
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:v=3;$case=tags:true%5C,client:false;$regex=tags:false%5C,client:false
Services
Search
[Option+S]
N. Virginia
voclabs/user2803180=ga23800@bristol.ac.uk @ 2945-0381-4684
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Instances (5) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
wordfreq-dev
i-06d15f34a07cbfa68
Stopped
t2.micro
View alarms
us-east-1b
i-0a176908fae65e79e
Terminated
t2.micro
View alarms
us-east-1a
i-0fbcca43c38c8cf31
Running
t2.micro
2/2 checks passed
View alarms
us-east-1c
i-02b11cb854021f8ee
Terminated
t2.micro
View alarms
us-east-1c
i-034c58045f444239e
Terminated
t2.micro
View alarms
us-east-1b
Select an instance
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



Task D – Optimising the WordFreq Architecture

Using AWS services and features from the Cloud Foundations course, it is possible to redesign the WordFreq application's cloud architecture to increase resilience and availability, ensure long-term backups, maintain cost-effectiveness, and prevent unauthorized access.

Firstly, for improved data processing performance I would opt to use Lambda to replace EC2 instances in processing text files. Lambda is a serverless compute service that automatically scales on demand. Lambda functions would be triggered by S3 events (file uploads), fetching files from S3, processing them and then storing results in DynamoDB. Scaling would be automatic against the number of incoming requests, ensuring efficient processing without the need for manual scaling. Using this instead of EC2 would reduce the cost by using serverless compute that only charges for actual usage. Amazon S3 is a crucial part of the architecture which must remain for its highly durable and scalable object storage. An uploading bucket would trigger Lambda and a processing bucket would store the results. It plays an important role in the resilience of the application because it allows data to be stored across multiple availability zones.

Similarly, Amazon SQS would still need to be used for its ability to decouple microservices, distributed systems, and serverless applications. SQS queues hold jobs for processing, allowing them to be retried in the event of failure. The impact of SQS on resilience is obvious, since it ensures message durability and availability. Amazon DynamoDB is a fully managed NoSQL database service that offers fast and consistent performance while allowing for easy scaling. In this design, DynamoDB is utilised to store file processing results, which are auto-scaled to manage variable demands. It increases application performance by providing high throughput and low latency for read/write operations. Amazon SNS is a fully managed pub/sub messaging service that helps you decouple and grow microservices, distributed systems, and serverless applications. SNS sends timely notifications about processing status to users and improves the resilience of an application due to its high availability and fault tolerance for notifications.

AWS IAM manages access to AWS services and resources securely. In this example, I would implement least privilege access with IAM roles and policies which, in turn improves security by minimising the risk of unauthorised access. Amazon VPC provides isolated cloud resources within the AWS cloud which can be used for Lambda and DynamoDB to control network access. VPC provides network isolation which would enhance the security of the architecture.

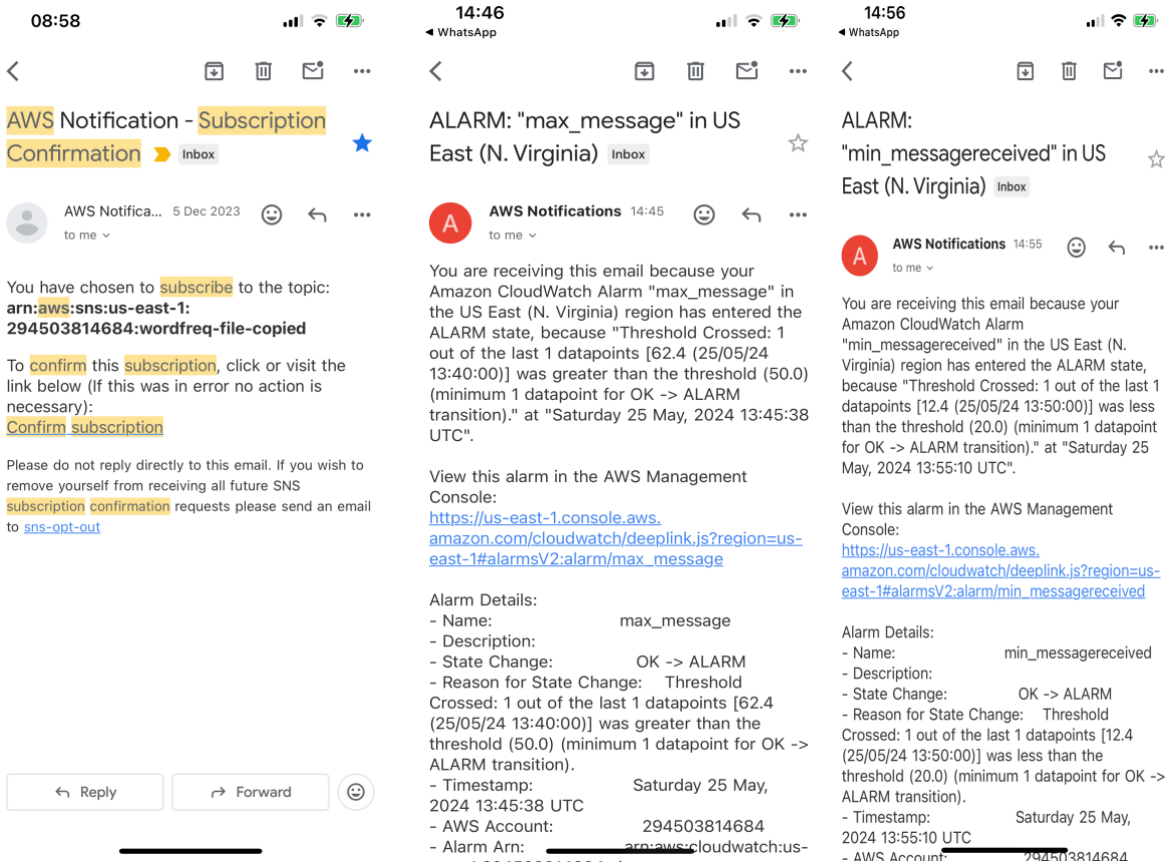
Amazon S3 Glacier is a secure, durable, and low-cost storage for data archiving and long-term backup. In this case, I'd use lifecycle policies to migrate WordFreq data to Glacier. Glacier lowers storage costs for long-term data preservation as compared to standard S3. AWS Cost Management Tools may be used to manage and optimise this application's AWS expenditures. I would create budgets and notifications to track and restrict expenditure. This contributes to the architecture's cost-effectiveness by allowing for cost awareness and budget control.

By integrating these AWS services and configurations, the revised architecture will improve the WordFreq application's resilience and availability, ensure long-term data backups, keep costs low, and protect against unauthorised access.

Task E – Resolving Issues

The only major issue that I experienced during this assignment was with CloudWatch alarms. I had initially chosen “Number of Messages Received” as my alarm metric, confusing it with “Approximate Number of Messages Visible”. Upon testing the auto-scaling and alarm functionality, it became clear that this metric was incorrect as it never went above the minimum value threshold, which I had set with “Approximate Number of Messages Visible” in mind. This meant that the minimum value alarm was constantly in alarm state and therefore the program never scaled out extra instances to manage the load of the application. Once I changed the metric to “Approximate Number of Messages Visible” the auto-scaling and alarm policies both began to work as planned and this is shown below.

I also experienced a minor issue related to the SNS service. After going through the setup instructions in December, I forgot to confirm my subscription to the SNS topic related to my work and was not receiving notifications about my files being processed or my alarms. This was easy to resolve as I found the confirmation email and clicked the link which subscribed me to the SNS topic, as seen below.



Task F - Further Improvements

Based on the services and frameworks covered in the full LSDE course, I have identified Apache Spark (on Amazon EMR) and Google Cloud Dataflow (with Apache Beam) as two alternative data processing applications that would provide more performance and robustness for this task.

Apache Spark (AS) is a robust open-source unified analytics engine for large data processing that includes modules for streaming, SQL, machine learning, and graph analysis. Running Spark on Amazon EMR (Elastic MapReduce) might significantly improve the performance and reliability of the WordFreq application. The goal of a managed cluster platform is to make it easier to run big data frameworks like Apache Hadoop and Spark. I would start by using Amazon EMR to create a cluster with Spark installed and configuring it with the appropriate instance types and auto-scaling features. For data processing, I'd use Spark to get text files from S3, count word frequencies, and then publish the results to S3 or DynamoDB. In terms of resilience and availability, EMR provides fault-tolerant clusters with automatic retries of failed processes, and data is distributed across several nodes to increase robustness. The benefits of AS span improvements in performance, scalability and flexibility. Spark's in-memory processing significantly speeds up data processing tasks, EMR allows easy scaling of the cluster to handle varying workloads and overall it supports a variety of big data use cases beyond simple word counting.

Google Cloud Dataflow (GCD) is a fully managed streaming analytics service that minimizes latency, processing time, and cost through autoscaling and batch or stream processing. It utilizes Apache Beam, a unified programming model for batch and stream data processing. As a fully managed service, it's purpose is to run Apache Beam pipelines. I'd start by establishing Dataflow tasks that read text files from Google Cloud Storage. Apache Beam pipelines provide batch and stream processing, enabling real-time or near-real-time processing. They would be used to parse text files, count word frequencies, and save results to Google BigQuery or Cloud Storage. In terms of robustness and availability, Dataflow dynamically manages resources, scaling up and down as needed, and has built-in fault tolerance to guarantee processing continues even if individual tasks fail. GCD

promises improved performance and flexibility, in addition to the benefits of being a managed service. Managed services decrease operational costs by automating infrastructure management and scalability. It is optimised for low-latency stream processing, can handle large-scale data efficiently and supports a wide range of data processing tasks with a single unified model.

By leveraging Apache Spark on Amazon EMR or Google Cloud Dataflow with Apache Beam, you can significantly enhance the performance, scalability, and robustness of the WordFreq application. Both solutions offer advanced features for distributed data processing, fault tolerance, and auto-scaling, making them ideal for handling large-scale data processing tasks efficiently.

Learner Lab Details

Username: ga23800@bristol.ac.uk

Password: Hannah05081!