Part 2

Overview of The Application

WordFreq is a complete application designed using the Go programming language. Counting words in a text file is the basic feature of the programme. The top ten most common words in the document are returned. It has been migrated to AWS cloud using four main services connected with each other. The application runs on a virtual server (EC2 instance). Files are uploaded to AWS cloud via Amazon Simple Storage Service (Amazon S3). When the files are uploaded successfully to S3 bucket, an alert is sent to the job queue on Simple Queue Service (SQS) where the application processes the available messages. Another results queue holds the top ten results of the processed jobs. Eventually, all the output results are loaded into a NoSQL database table (DynamoDB).

Auto-Scaling Design Architecture

The purpose of Auto-Scaling architecture is to allow the application to scale up and down to meet demand. So, at this stage my goal was to support the application to handle greater loads. I did this by trying to vertically scale the application. For example, increasing the size of the instance from t2.micro to t2.large. Also, horizontally scaling by increasing the number of instances for the WordFreq application. When setting up the auto-scaling architecture for the WordFreq application, first I created Cloudwatch alarms attached to scaling policies in the auto-scaling group. The alarms' purpose is to be triggered once a threshold is breached to take the action of launching worker instances specified in the scaling policy. The scaling depends on the SQS jobs queue because that is where the messages are picked up by the consumer and get processed. The process here begins with uploading test files to S3 bucket, then, an event-notification sends an alert to the jobs queue indicating that there are available messages ready to be processed.

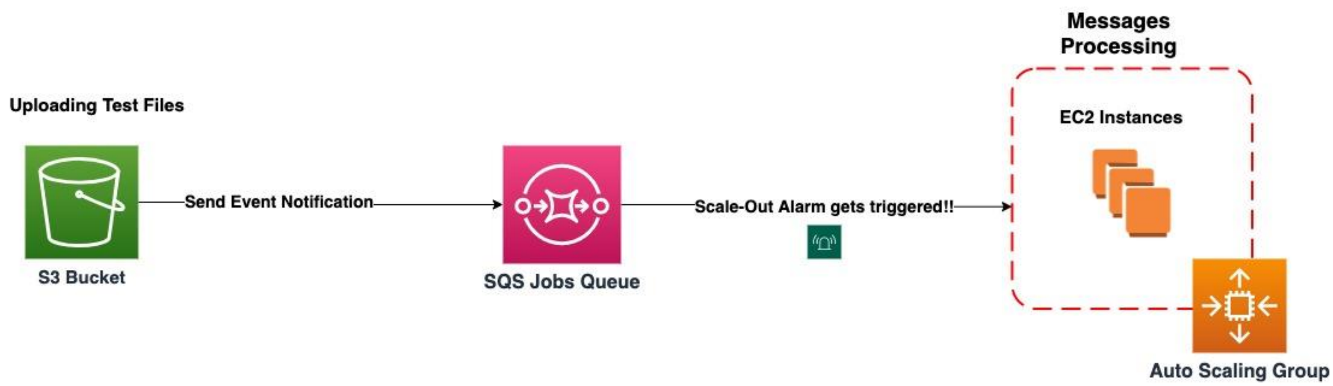


Figure 1: Jobs Queue Scaling Architecture.

I have specified the threshold that if the average number of visible messages on the jobs queue is equal to or more than three messages for one data point within a minute, the scale-out alarm will be triggered to launch three servers for processing, and if there are no more messages remaining on the jobs queue, the scale-in alarm will be activated to leave no instances running for cost saving.

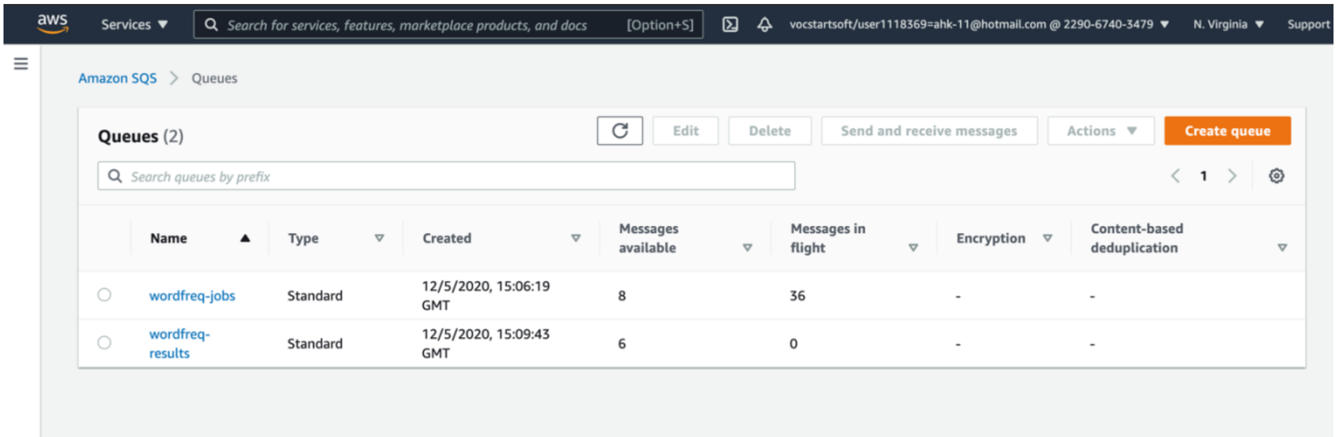


Figure 8: SQS Queues Status

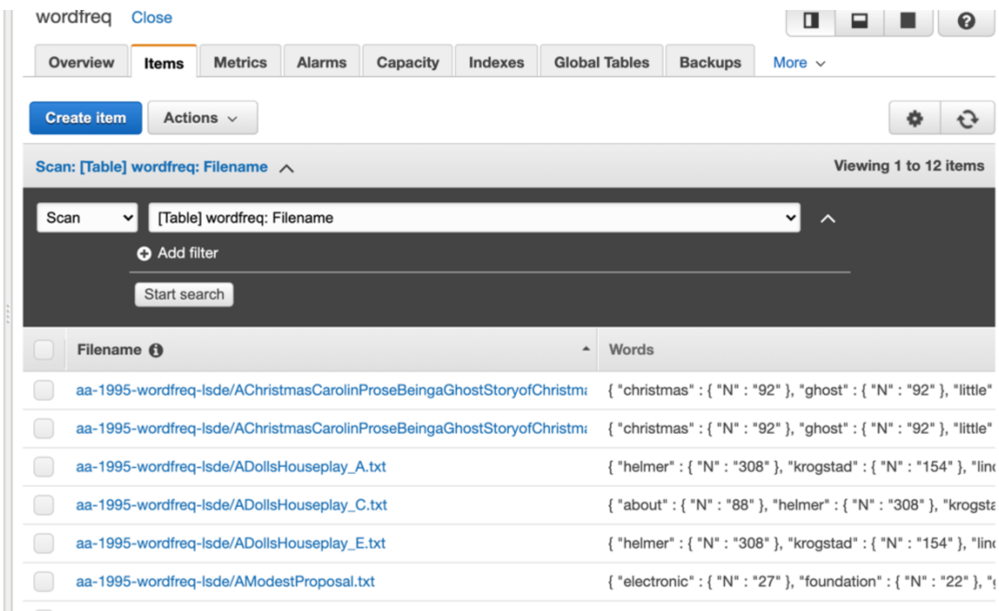


Figure 9: Saving the Results in DynamoDB

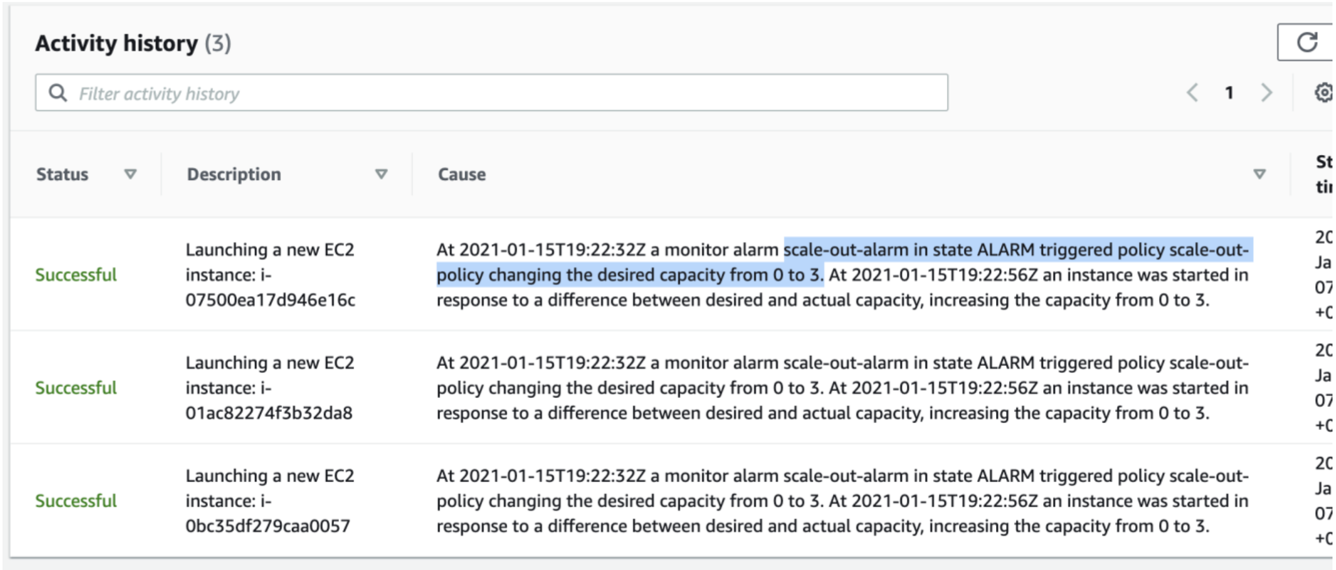


Figure 10: Auto-Scaling group Activity Monitoring

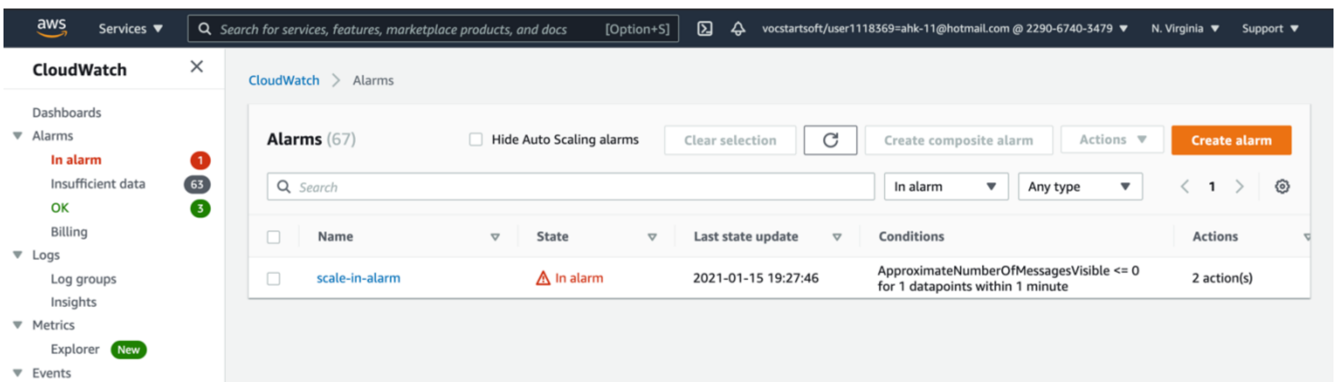


Figure 11: Scale-In Alarm Response

Securing and Optimising the WordFreak Architecture

To secure and optimise the WordFreq architecture, I have decided to create a new Virtual Private Network (VPC) called (wordfreq-vpc) to deploy my resources in it. This VPC has three Availability Zones with public and private subnets for each Availability Zone (AZ). I have chosen to do this because by launching instances in separate Availability Zones I can protect the WordFreq application from the failure of a single location. I have also set up public and private route tables and their purpose is to define access to the internet and between subnets.

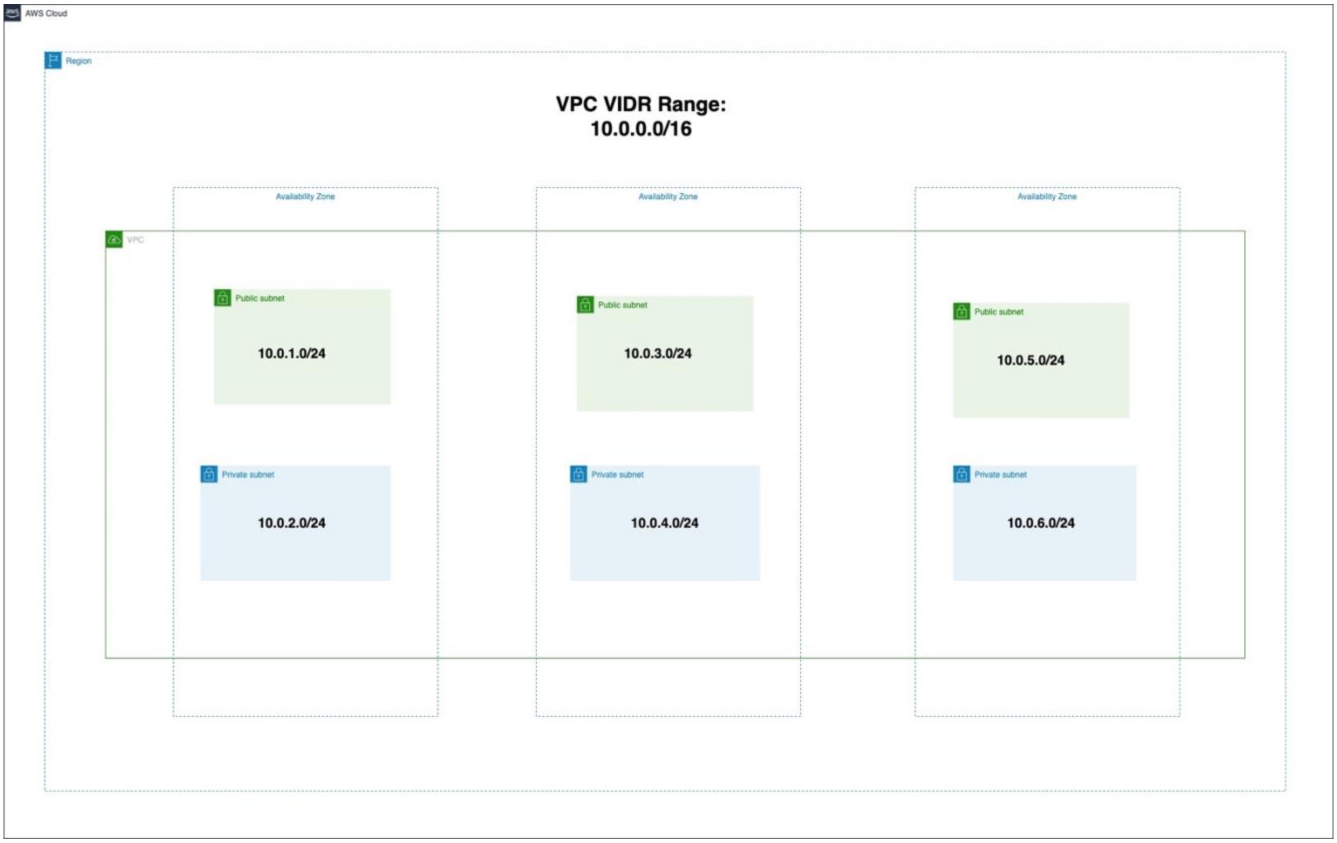


Figure 12: WordFreq Basic Network Architecture

I have also set up Network ACL which acts as a firewall that controls traffic from and to subnets. I have only allowed my IP address in the inbound rules to access the subnets so as to not make it open to the world. To protect the instances access, I have created a Security Group for it. Both Network ACL and Security Group control the traffic flow but the difference is that Network ACL is attached to the subnet level while security groups are attached to instances level.

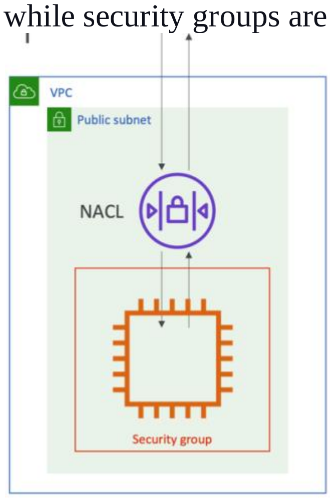


Figure 13: NACL Architecture

To protect the App data in the private subnet, I thought of setting up a NAT Gateway in the public subnet to allow the private subnet to access the internet while remaining private. I could not implement this feature due to limitations in the AWS Educate account, so instead I have set up a NAT instance in the public subnet to enable the worker instances in the private subnet to send traffic to the NAT instance in the public subnet. Internet traffic from the worker instances in the private subnet will be routed to the NAT instance, which then connects to the internet. The traffic is routed to the public IP address of the NAT instance.

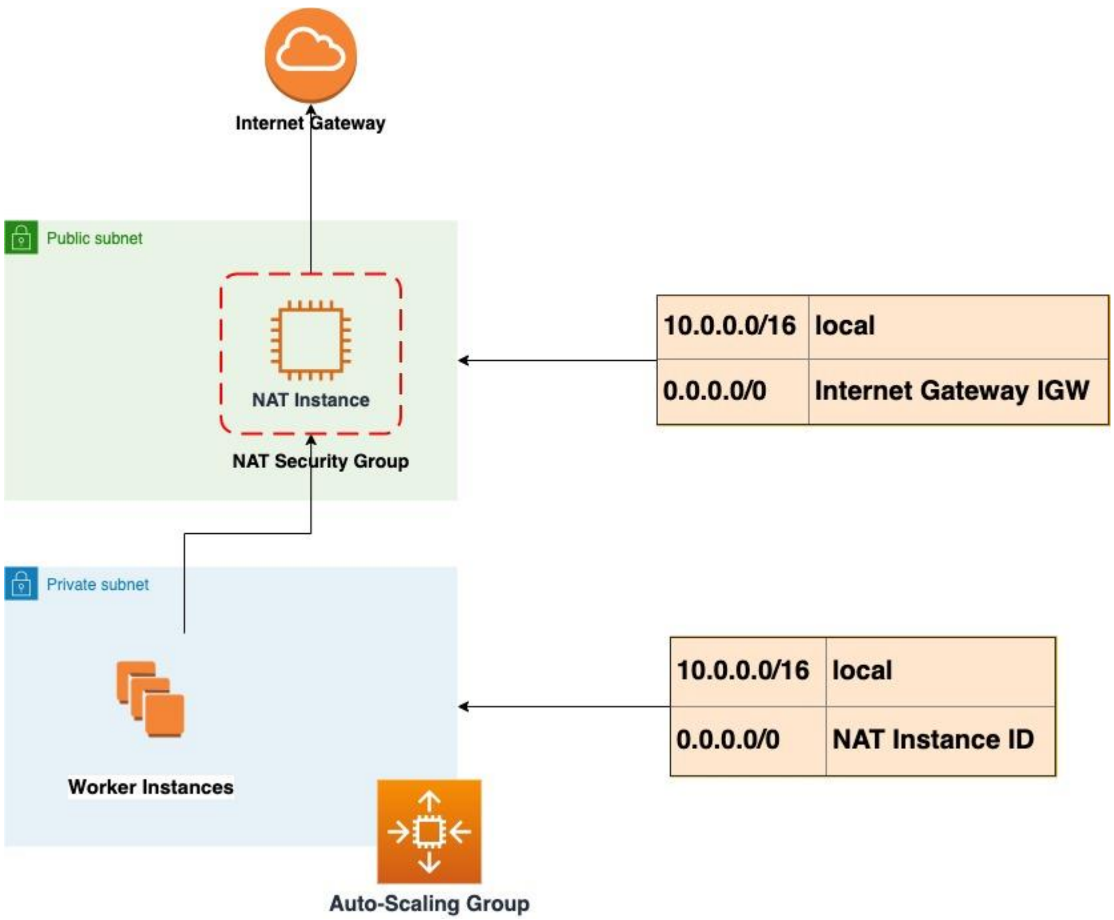
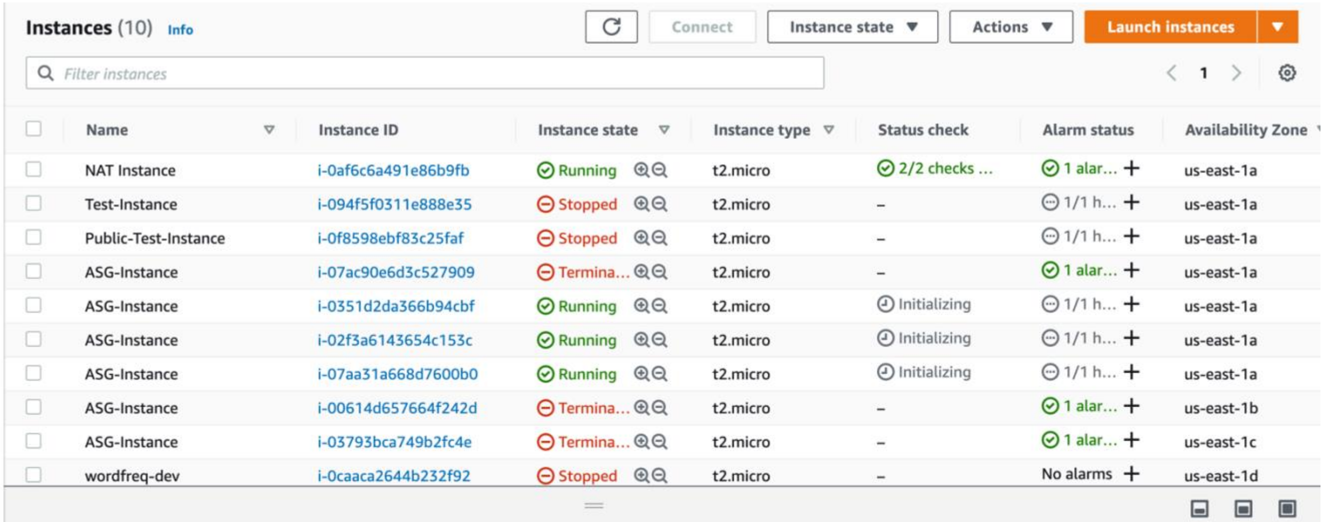


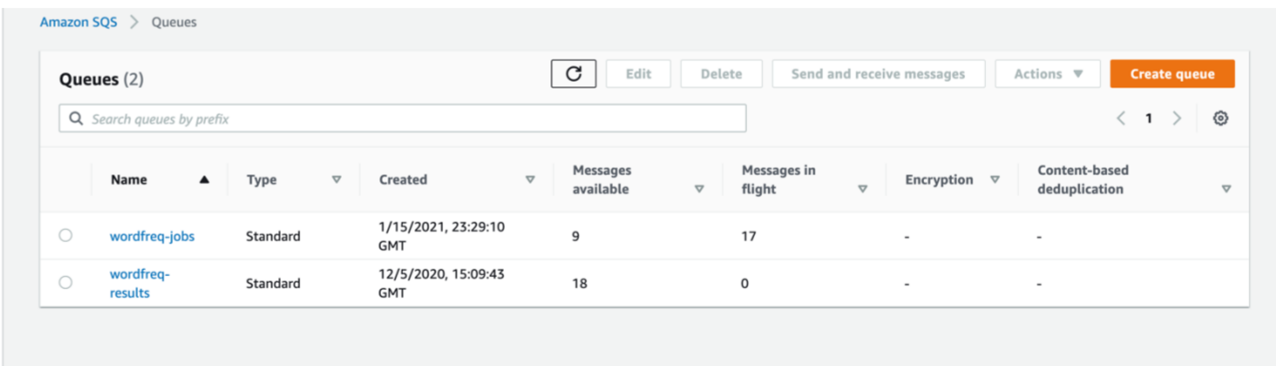
Figure 14: NAT Instance Architecture



Instances (10) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
NAT Instance	i-0af6c6a491e86b9fb	Running	t2.micro	2/2 checks ...	1 alar...	us-east-1a
Test-Instance	i-094f5f0311e888e35	Stopped	t2.micro	–	1/1 h...	us-east-1a
Public-Test-Instance	i-0f8598ebf83c25faf	Stopped	t2.micro	–	1/1 h...	us-east-1a
ASG-Instance	i-07ac90e6d3c527909	Termina...	t2.micro	–	1 alar...	us-east-1a
ASG-Instance	i-0351d2da366b94cbf	Running	t2.micro	Initializing	1/1 h...	us-east-1a
ASG-Instance	i-02f3a6143654c153c	Running	t2.micro	Initializing	1/1 h...	us-east-1a
ASG-Instance	i-07aa31a668d7600b0	Running	t2.micro	Initializing	1/1 h...	us-east-1a
ASG-Instance	i-00614d657664f242d	Termina...	t2.micro	–	1 alar...	us-east-1b
ASG-Instance	i-03793bca749b2fc4e	Termina...	t2.micro	–	1 alar...	us-east-1c
wordfreq-dev	i-0caaca2644b232f92	Stopped	t2.micro	–	No alarms	us-east-1d

Figure 15: Auto-Scaling group worker instances status with NAT Instance



Amazon SQS > Queues

Queues (2)

Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
wordfreq-jobs	Standard	1/15/2021, 23:29:10 GMT	9	17	-	-
wordfreq-results	Standard	12/5/2020, 15:09:43 GMT	18	0	-	-

Figure 16: SQS Queues Status

NAT instances are great for establishing a private connection to AWS services. However, deploying NAT instances in many AZs will not be cost-effective. So, I decided to do an alternative and more efficient solution which is creating VPC Endpoints to enable the instances in the private subnet to connect privately to AWS services and thus optimising the security of the App's data. VPC Endpoints are virtual devices that allow me to connect to AWS Services using a private network instead of a public www network. There will not be any need for Internet Gateway, NAT instance, virtual private gateway, VPN connection, or AWS Direct Connect for the connection which is a good advantage. Gateway VPC Endpoints for the services S3, DynamoDB are free while Interface VPC endpoint for SQS service costs $0.01 per hour, but still cheaper than a NAT gateway.

For the WordFreq architecture, I think VPC Endpoints are an optimal way because they enhance security and lower latency to access AWS services. I also needed to add them in the private route table in order to be able to route the traffic to the VPC Endpoints which then connects to the service.

When building a VPC Endpoint Interface for a service such as SQS, an elastic network interface (ENI) with a private IP address that serves as an entry point for service traffic is generated on the selected subnet.

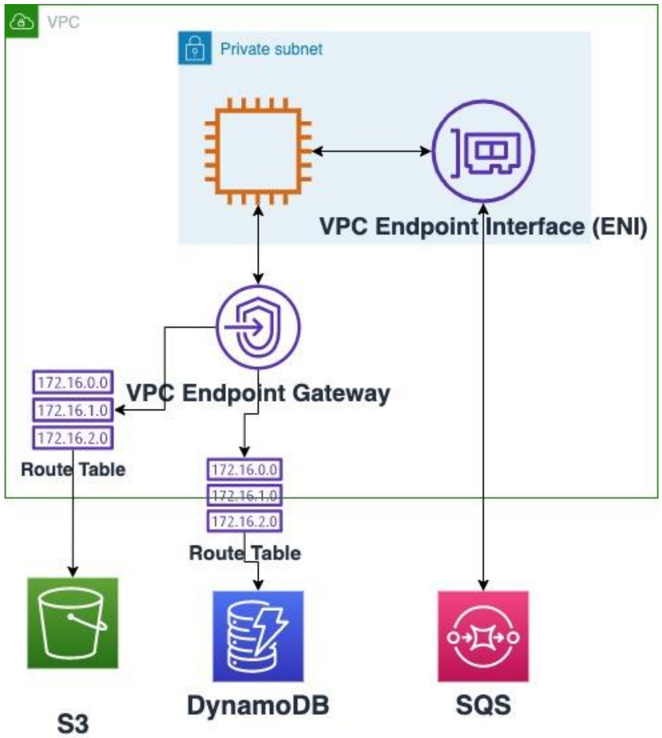
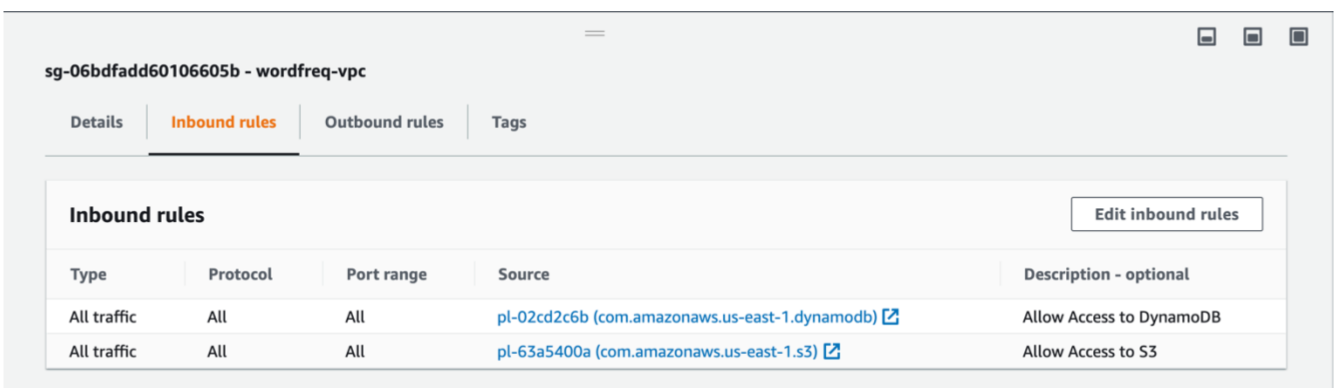


Figure 17: Changing the inbound rules for the wordfreq-vpc security group



Type	Protocol	Port range	Source	Description - optional
All traffic	All	All	pl-02cd2c6b (com.amazonaws.us-east-1.dynamodb)	Allow Access to DynamoDB
All traffic	All	All	pl-63a5400a (com.amazonaws.us-east-1.s3)	Allow Access to S3

Figure 18: Changing the inbound rules for the wordfreq-vpc security group

Since the security group of the VPC is open to the world at port 22, it is not best practice to leave it like that. I had to restrict the access to make the application more secure. Therefore, I only enabled access for the VPC endpoints that I created to gain access to AWS services privately.

Issues

There are some issues I encountered during my implementation. When I created a launch configuration for the ASG to deploy the application in a new VPC environment, I had an error