Assignment 3

Part A

Scenario: In a remote and disaster-prone region, one organization faces challenges during natural calamities like earthquakes and floods. Communication breakdowns lower their ability to access critical health and location data, making it difficult to provide immediate assistance during emergencies. The organization plans to deploy a real-time health monitoring and emergency response system to tackle these issues. This system will quickly collect important health data from affected individuals using wearable devices and medical sensors, analyze it in real time, and coordinate emergency responses effectively to ensure that urgent medical aid reaches those in urgent need. The main goal of this system is to save lives, enhance the efficiency of disaster relief operations, and improve the overall resilience of the disaster effected community.

Use-Case:

Health Data Collection: Equipped with AWS IoT Core, the system gathers real-time health data, including heart rate, blood pressure, body temperature, and respiratory rate, from individuals affected by the disaster. This data is instantly transmitted through AWS Kinesis Data Streams, allowing immediate processing and analysis[1].

Real-Time Health Analytics: Powered by AWS Kinesis Data Analytics and advanced machine learning models from Amazon SageMaker, the system processes the incoming health data in real time[1]. It can accurately detect critical health conditions like cardiac arrest, respiratory distress, or abnormal vital signs, allowing it to prioritize emergency cases based on severity. When emergencies are seen, AWS Lambda functions, integrated with Amazon SNS, trigger real-time alerts to emergency response teams and medical personnel, enabling them to take swift action and allocate resources efficiently. The system securely stores historical health data in Amazon S3 using Kinesis Data Firehose for future analysis and disaster preparedness, further enhancing the community's resilience during future calamities[1].

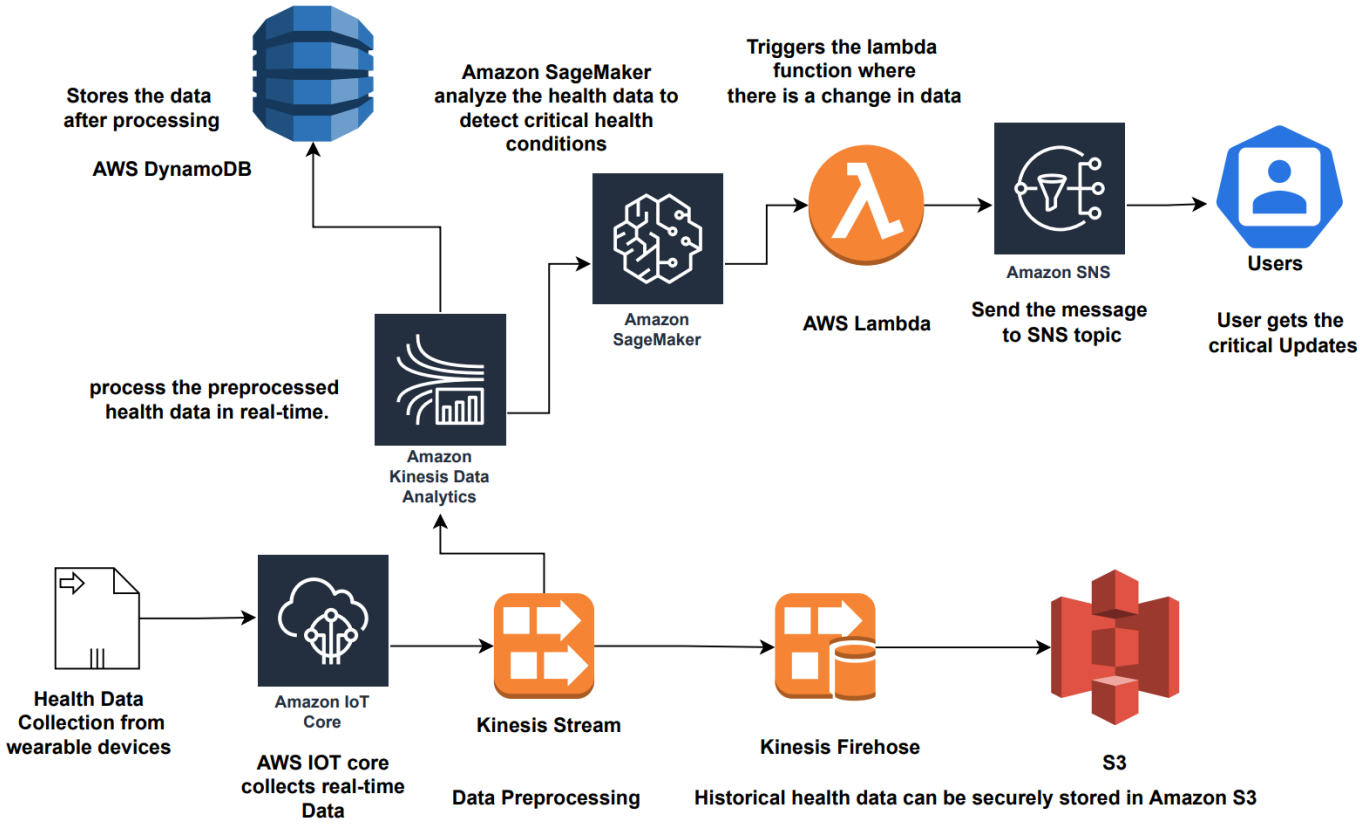

Fig 1: Workflow Diagram (Draw.io - https://app.diagrams.net/)

Part B:

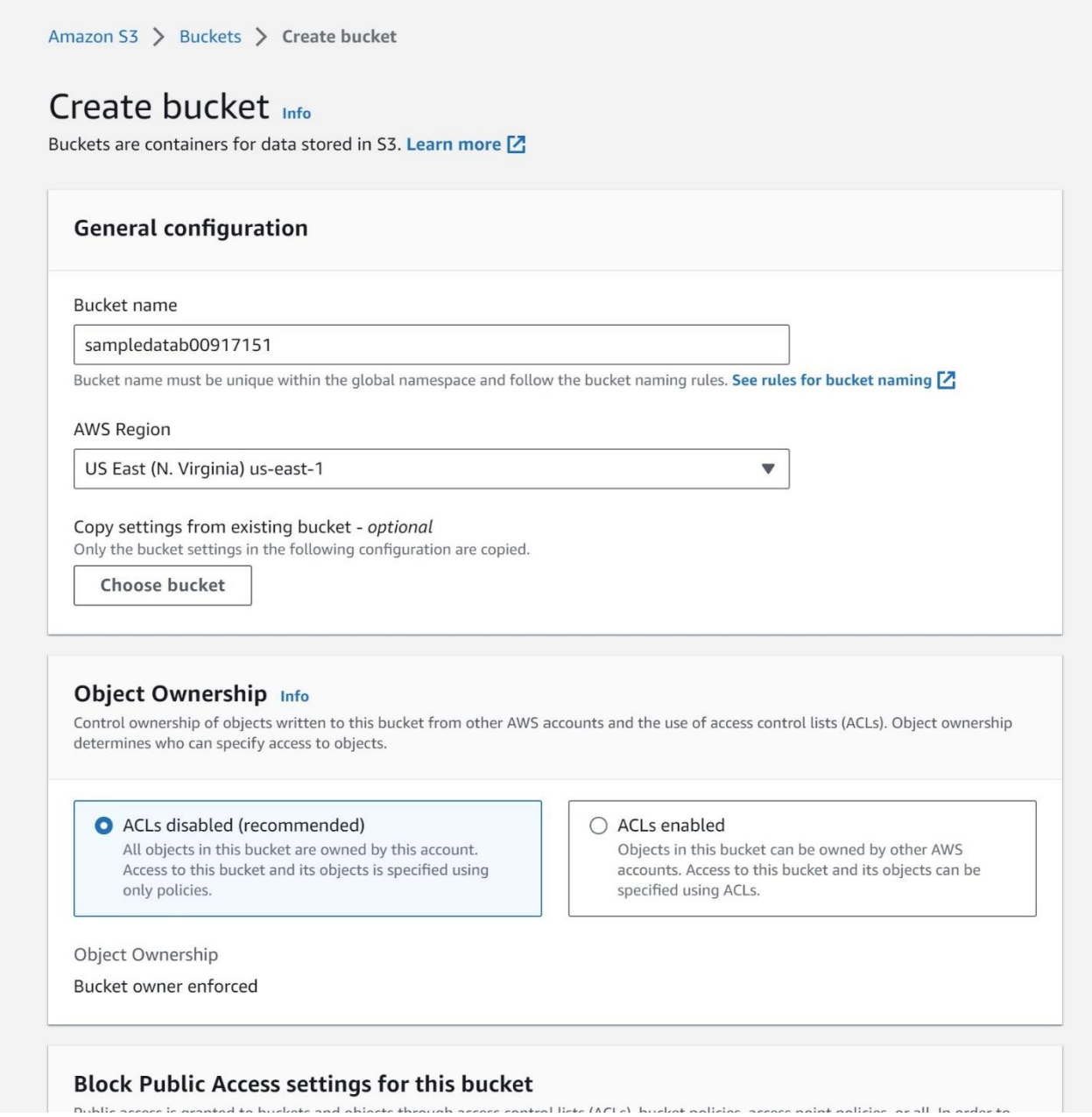

Fig 2: Creating 1st Bucket sampledatab00917151

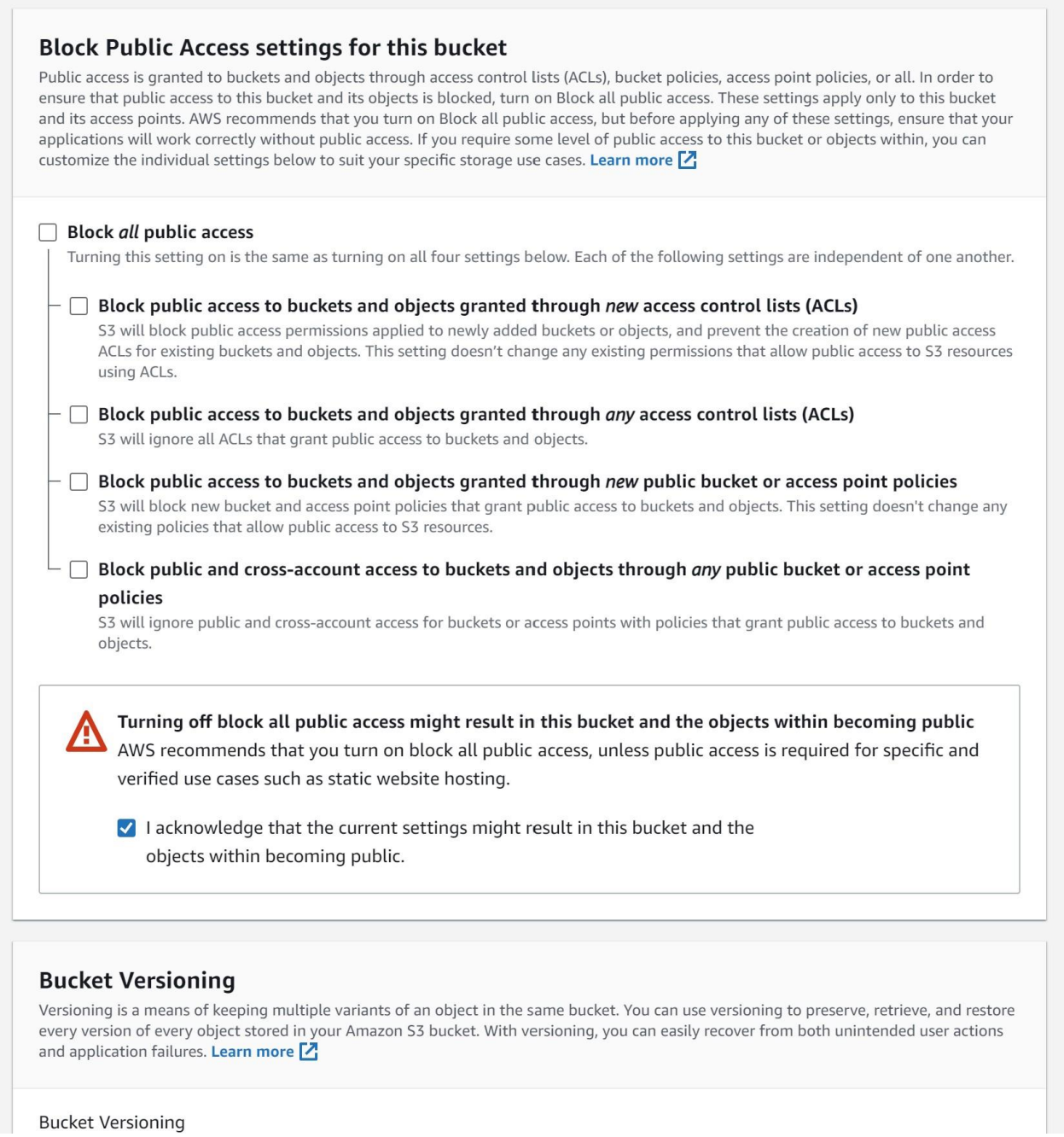

Fig 3: Configuration of the bucket

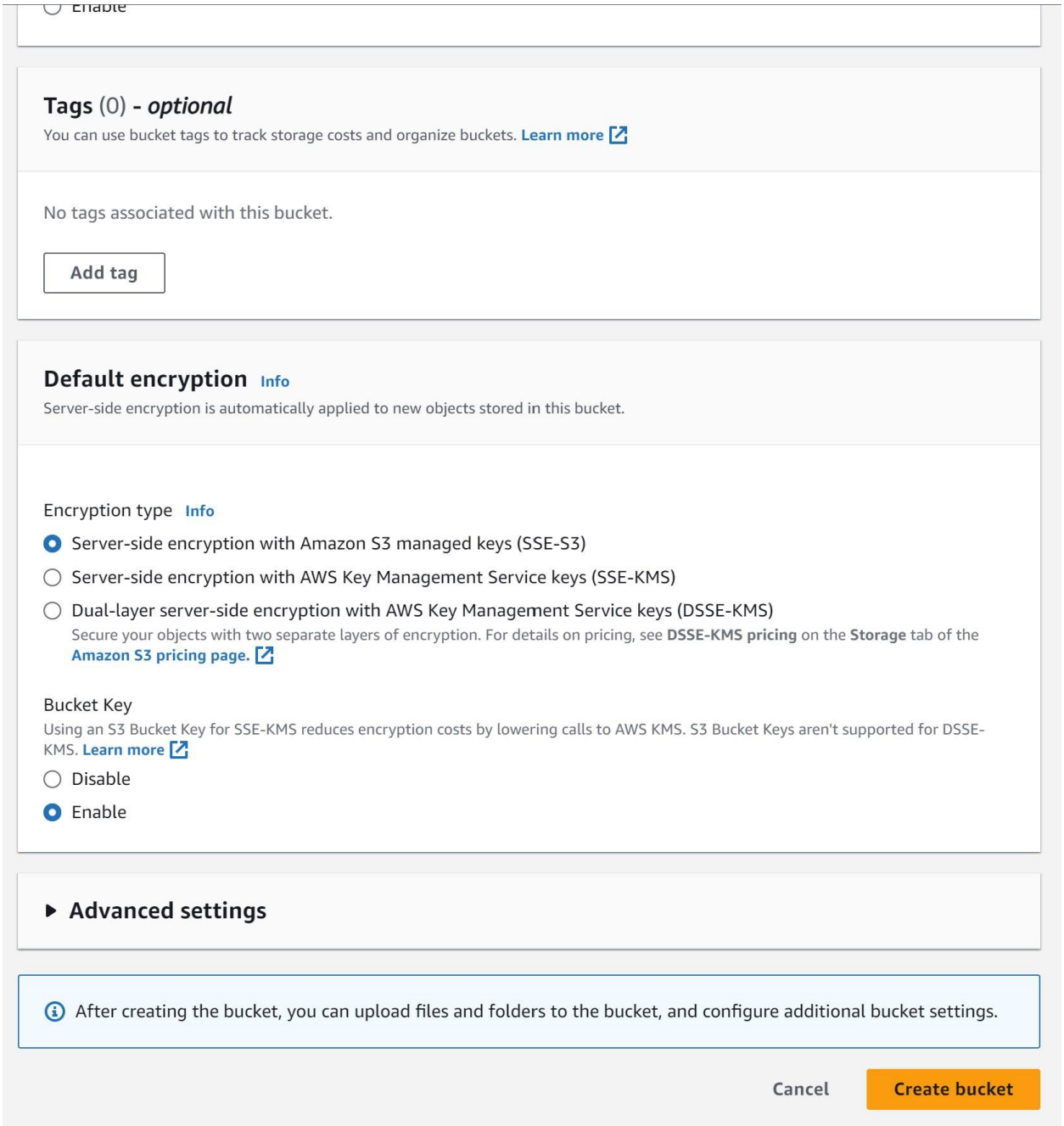

Fig 4: Creating bucket

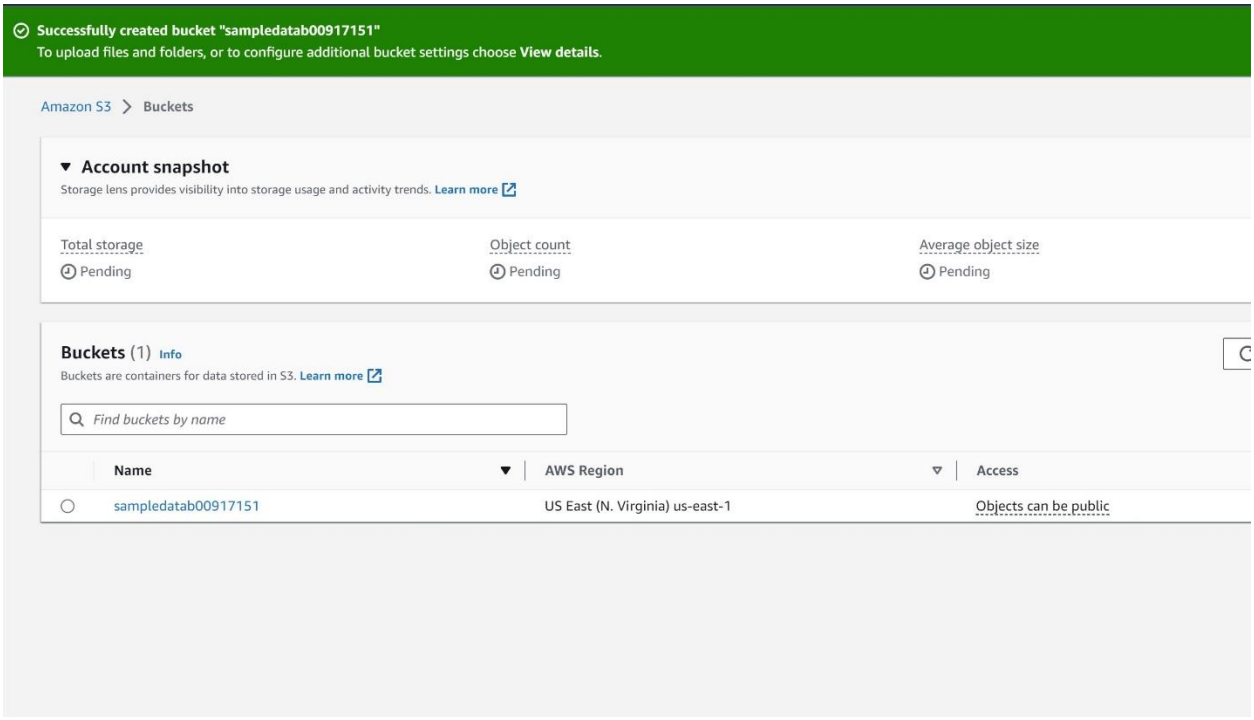

Fig 5: 1st bucket Created

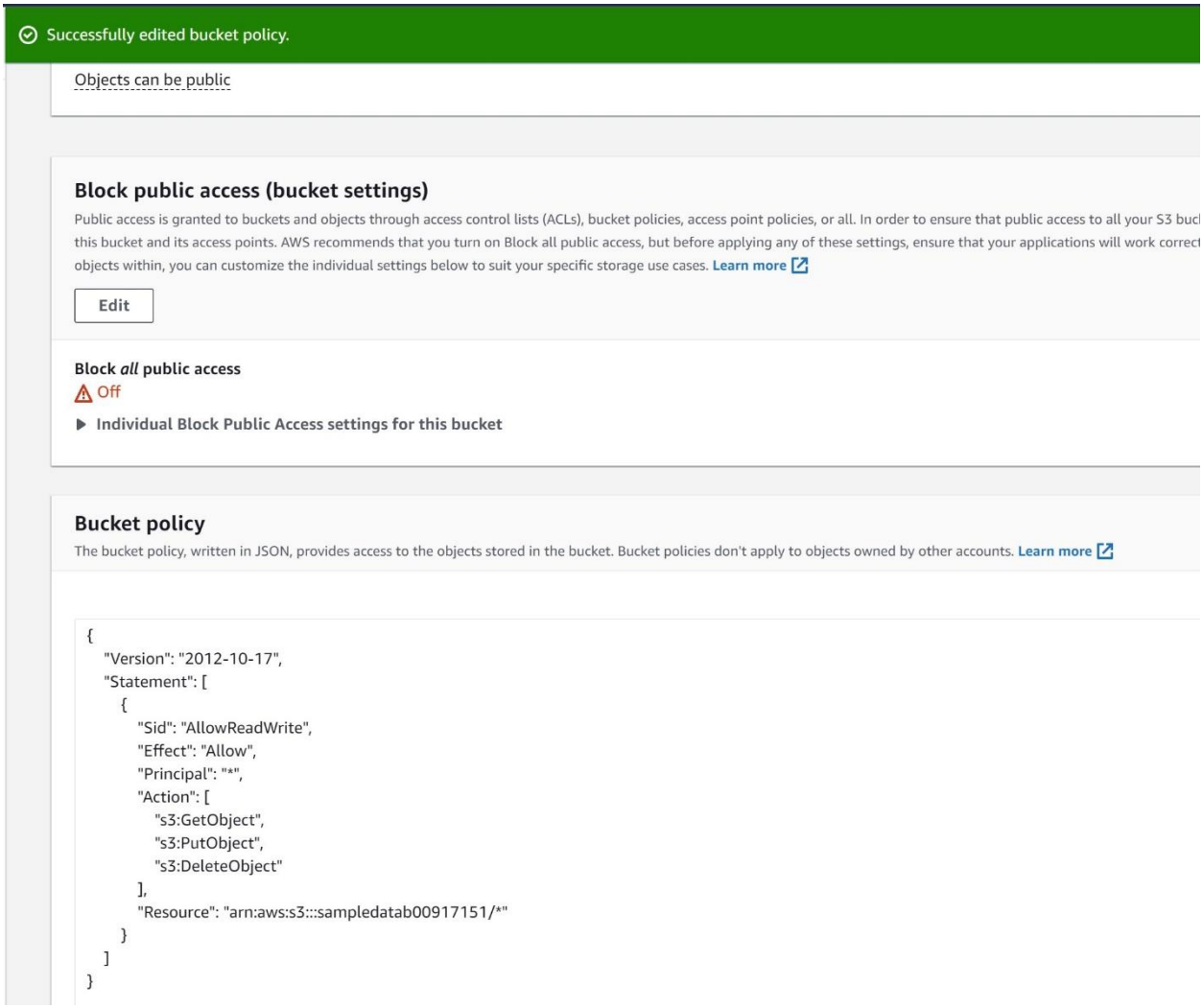

Fig 6: changing the bucket policy of the 1st bucket

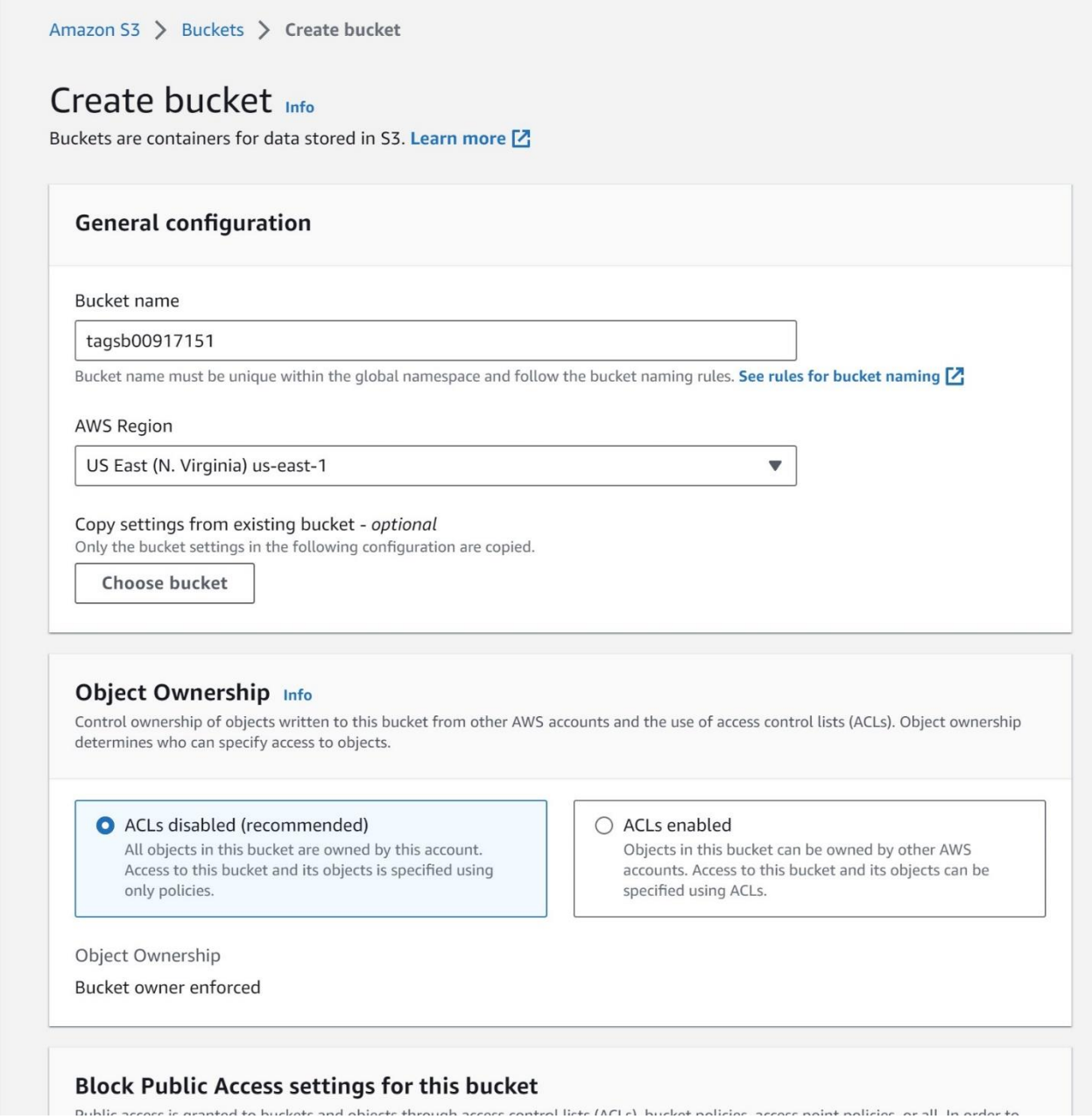

Fig 7: creating 2nd bucket tagsb00917151 with default setting as previous bucket

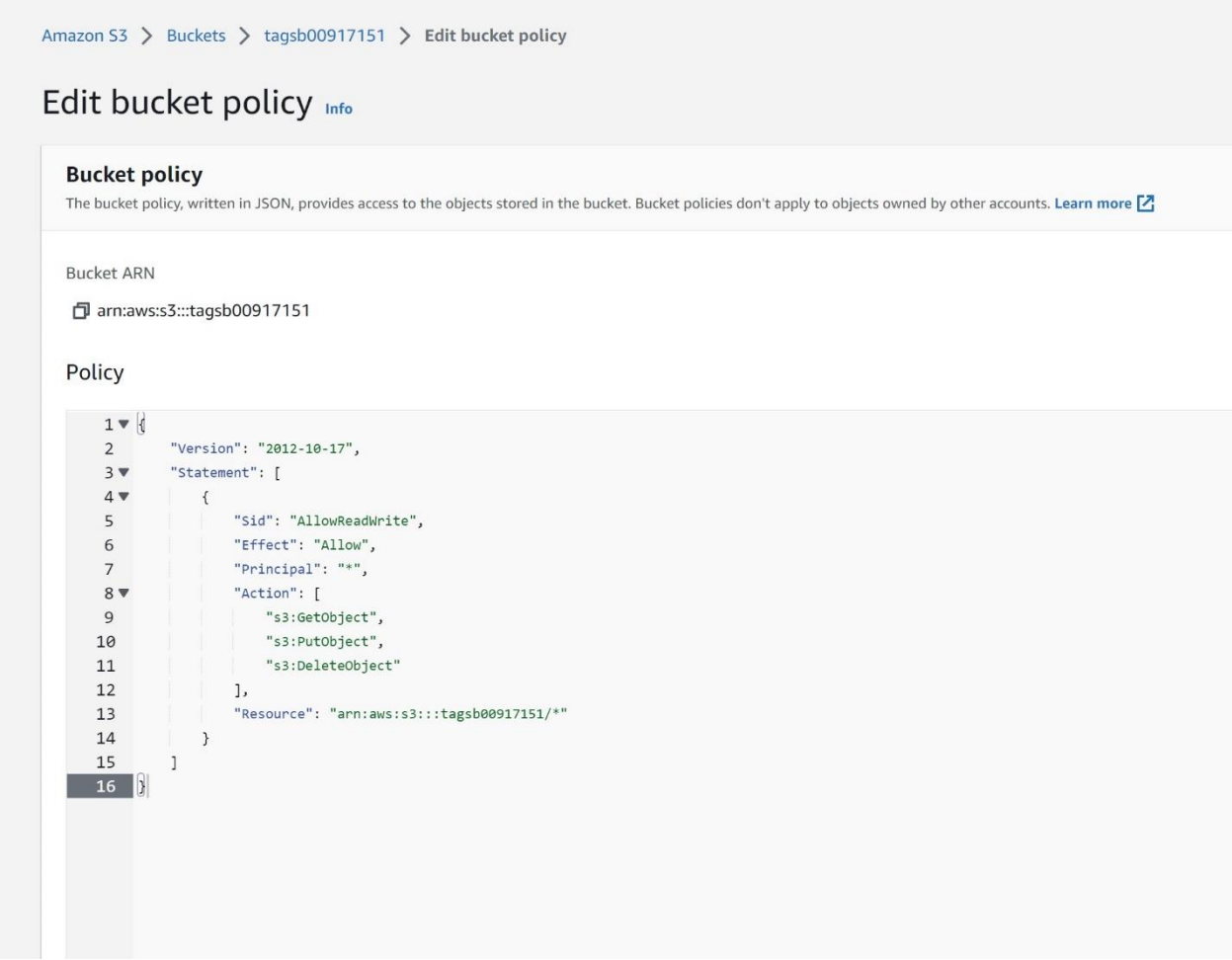

Fig 8: changing the bucket policy of 2nd bucket to allow public access

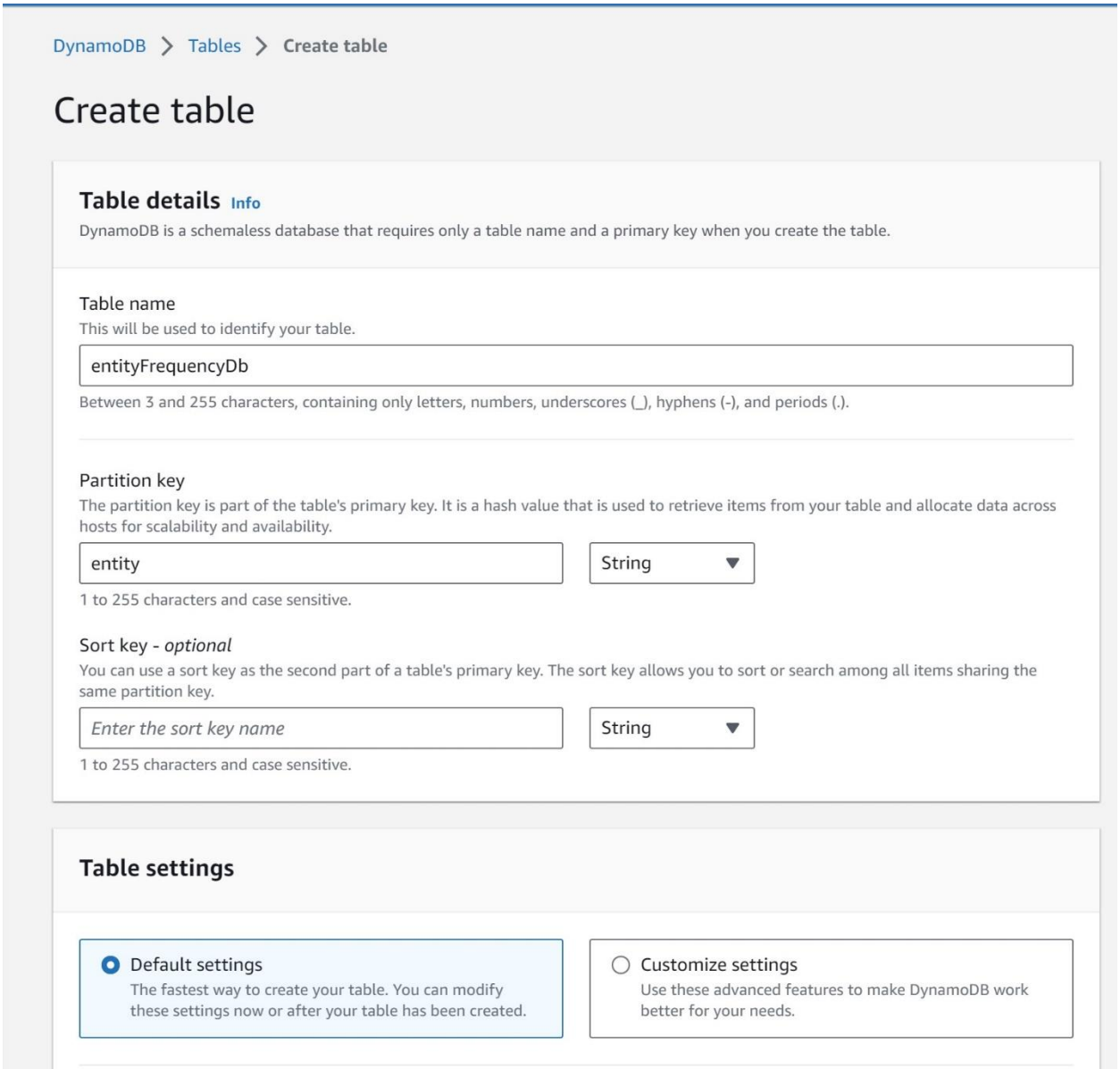

Fig 9: Creating a DynamoDb table called entityFrequencyDb

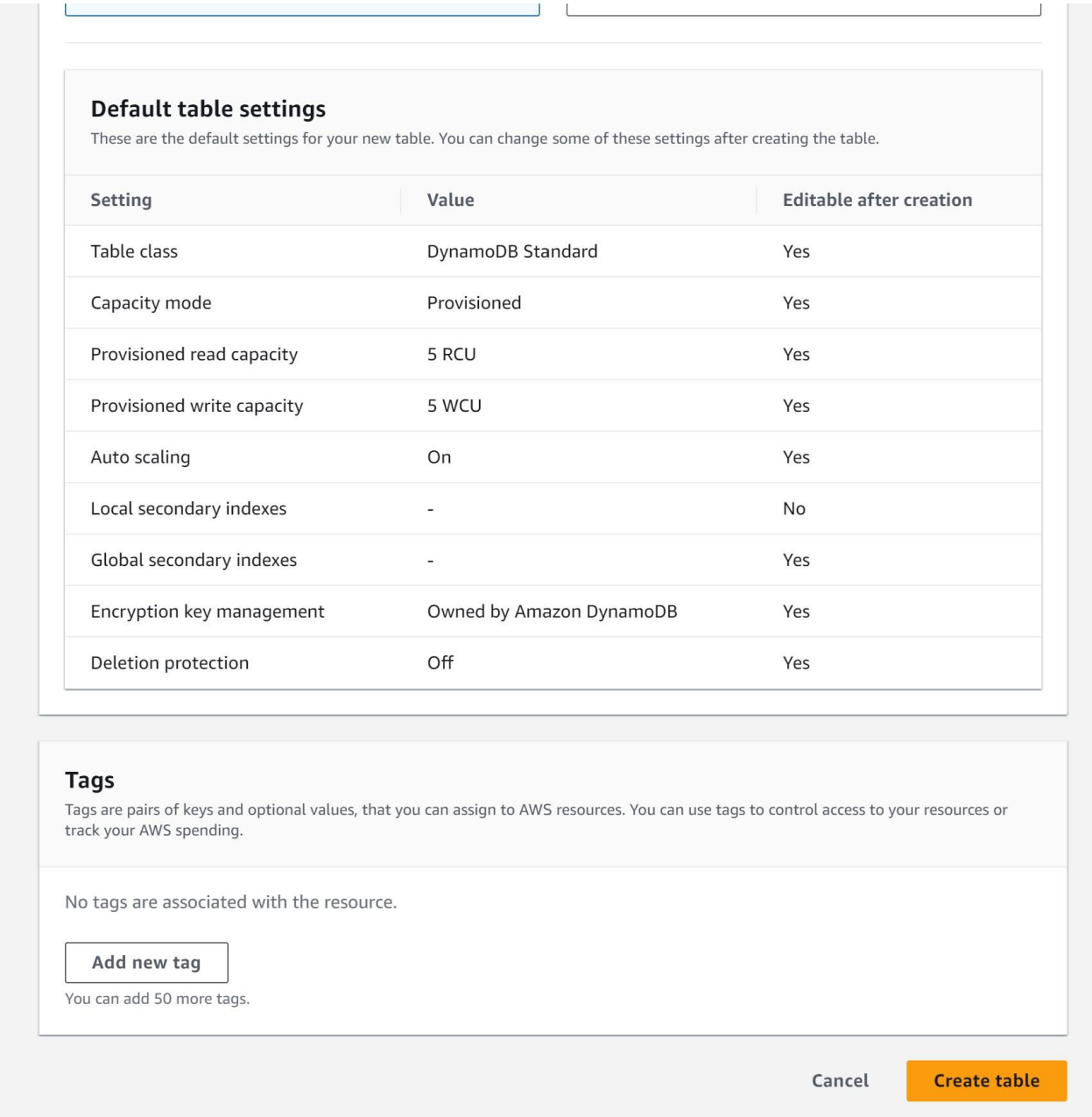

Fig 10 : creating table with default config

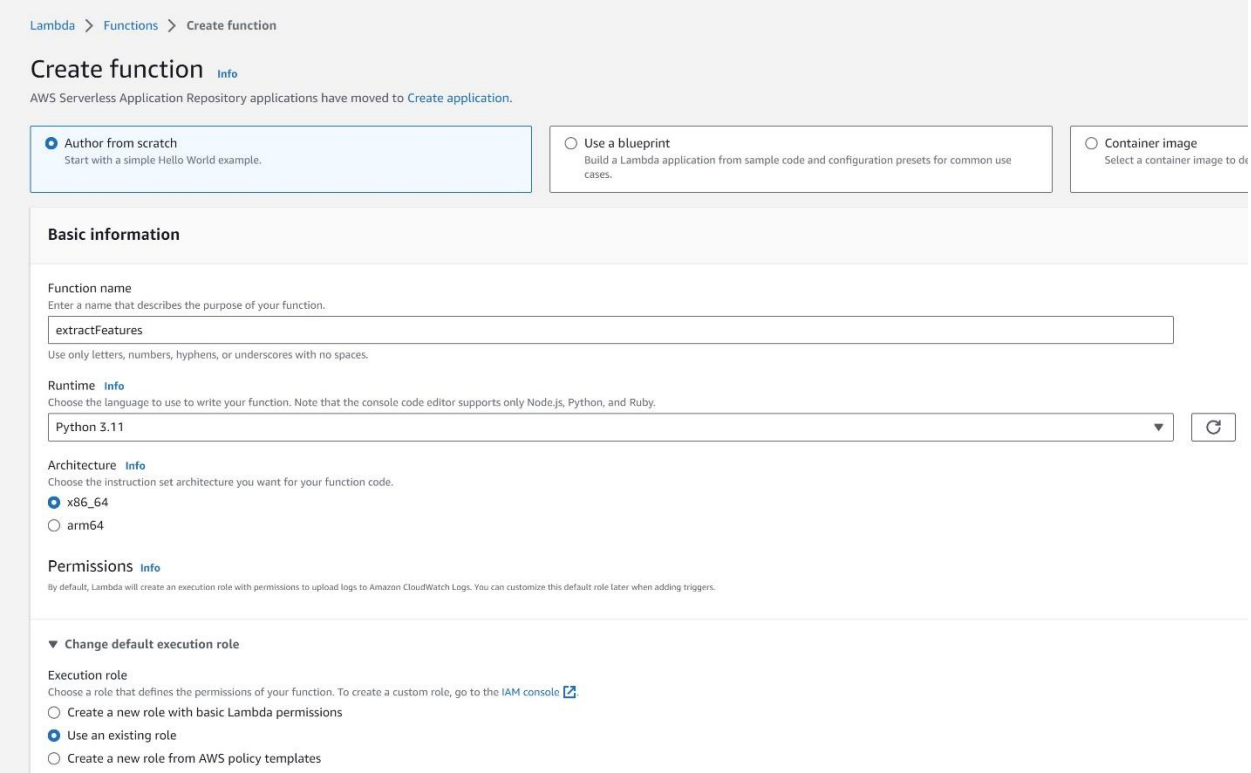

Fig 11: creating 1st lambda function called extractFeatures

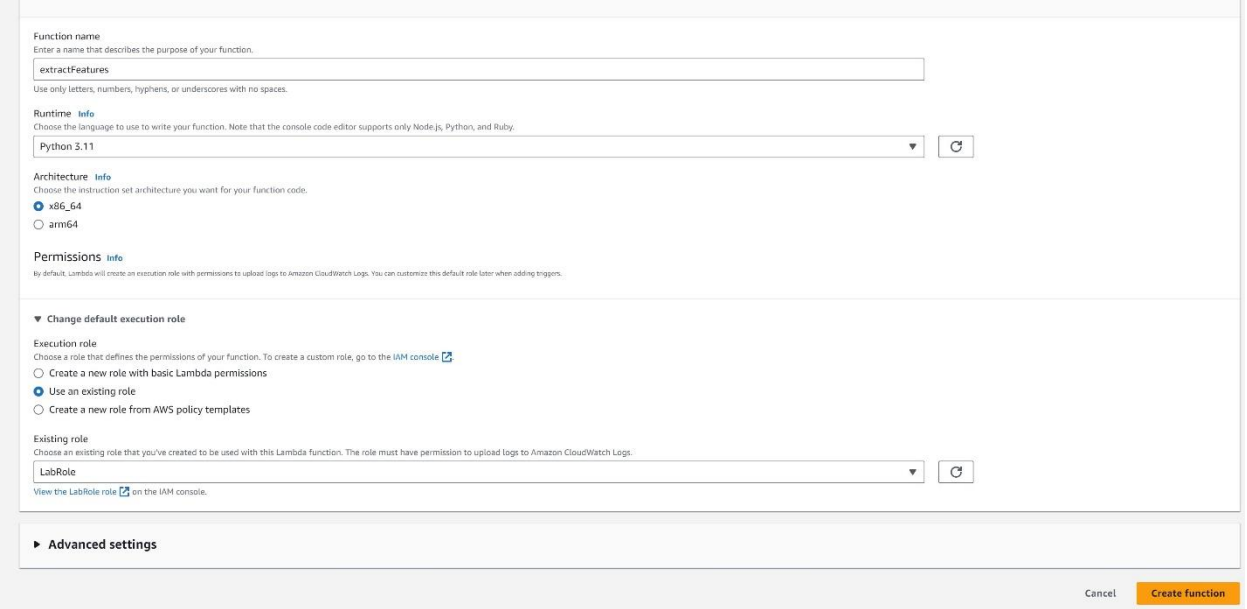

Fig 12: Assigning a default LabRole and language selection

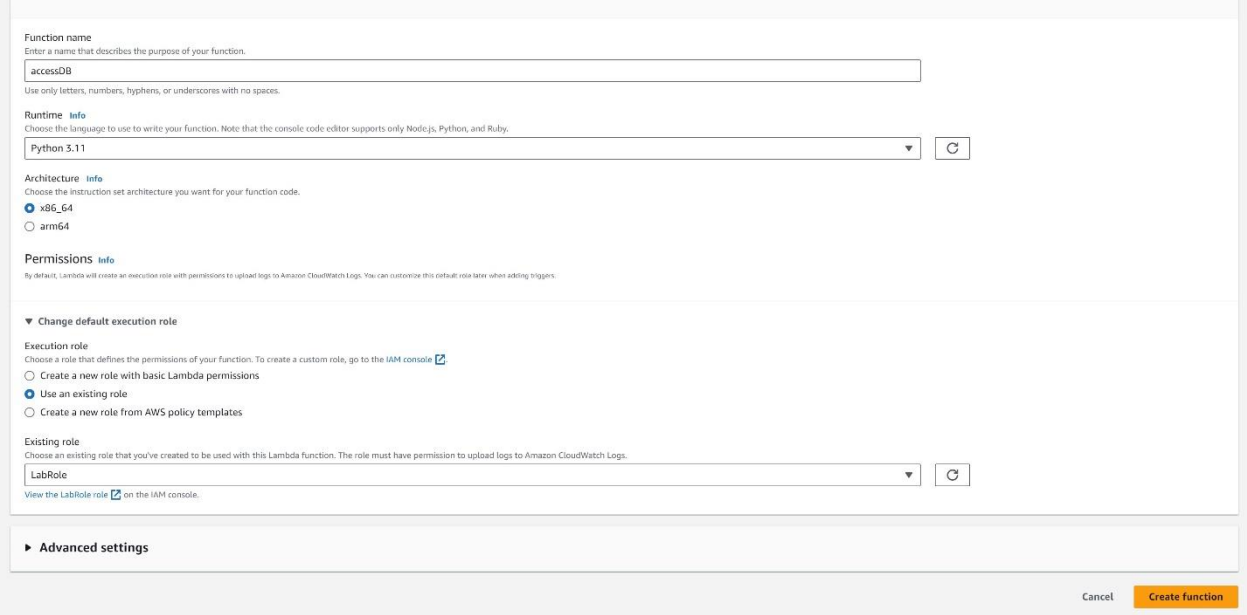

Fig 13: creating a 2nd lambda function called accessDb with default config as 1st function.

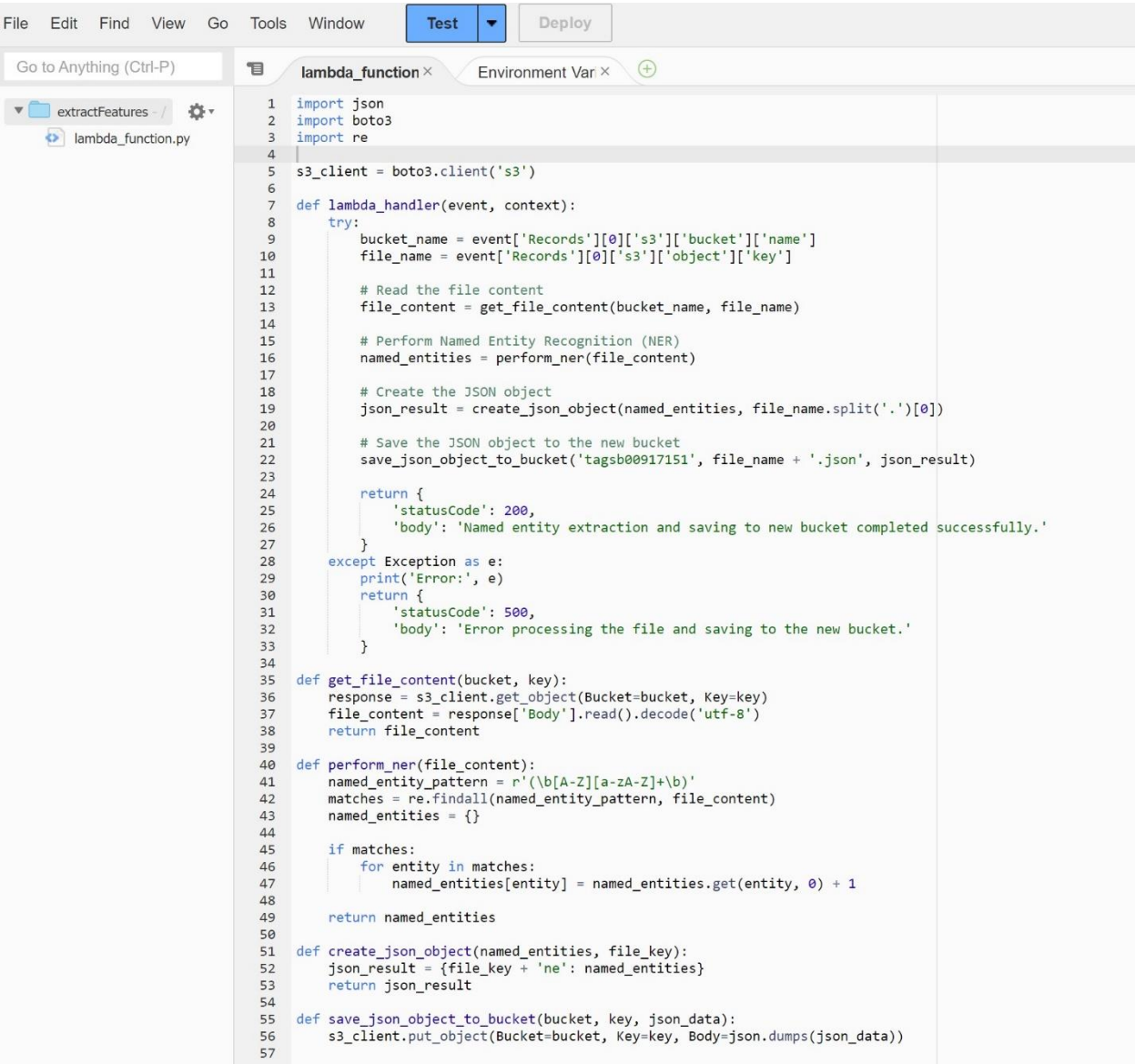

Fig 14: lambda code for extractFeatures

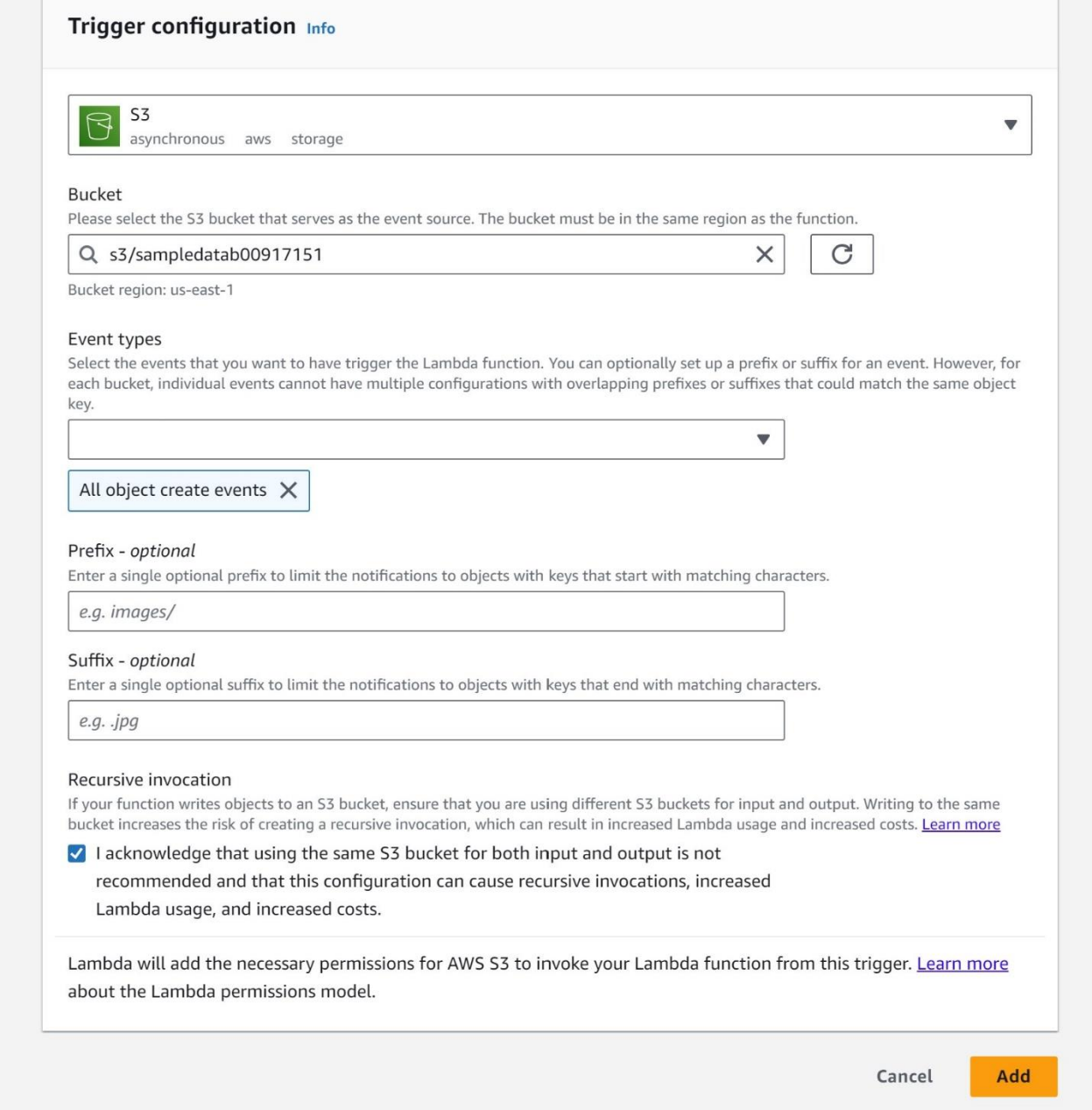

Fig 14: setting trigger for extractFeatures lambda code with s3 bucket sampledatab00917151

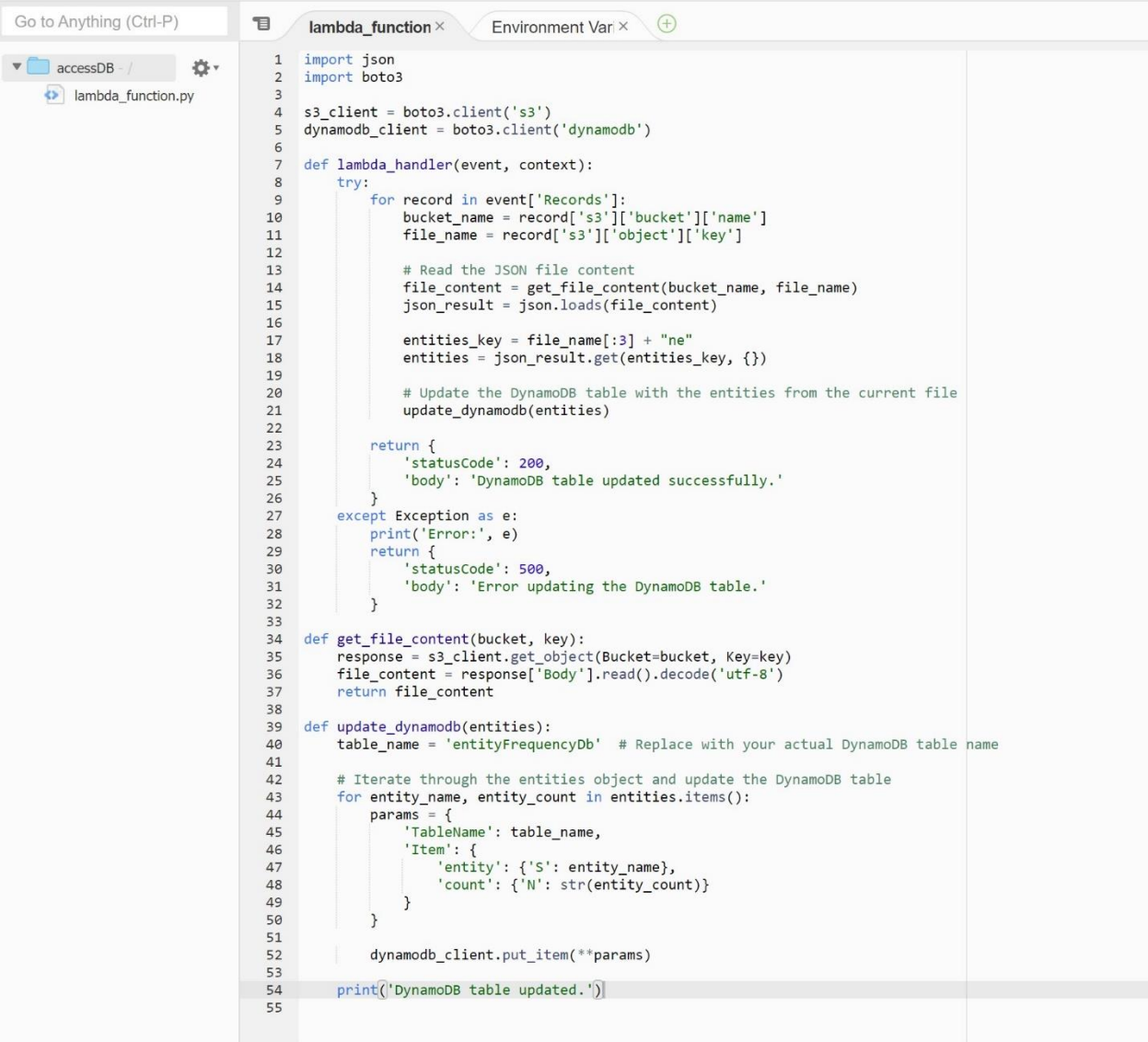

Fig 15: lambda code for accessDB

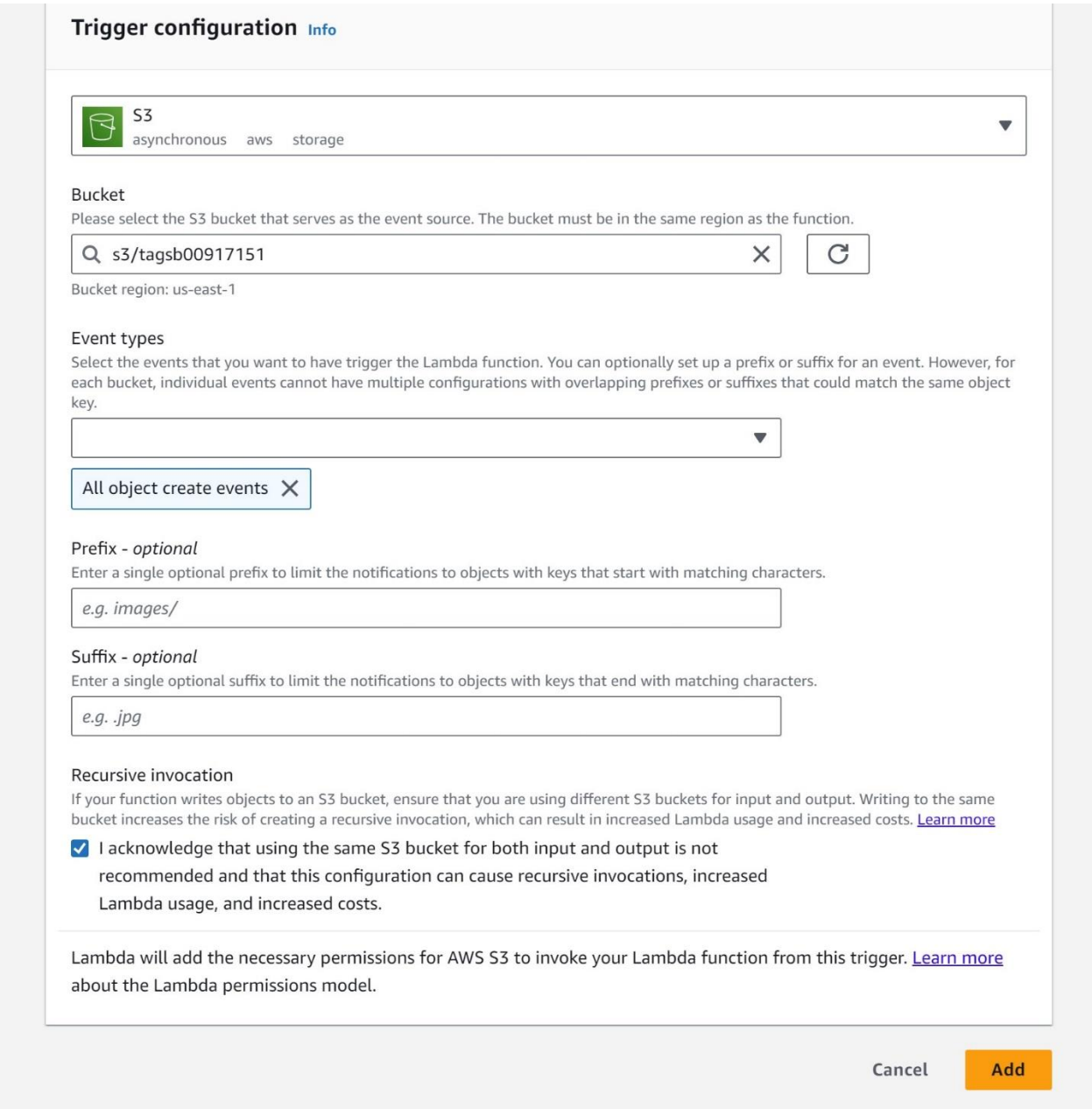

Fig 16: setting trigger for accessDb lambda code with s3 bucket tagsb00917151

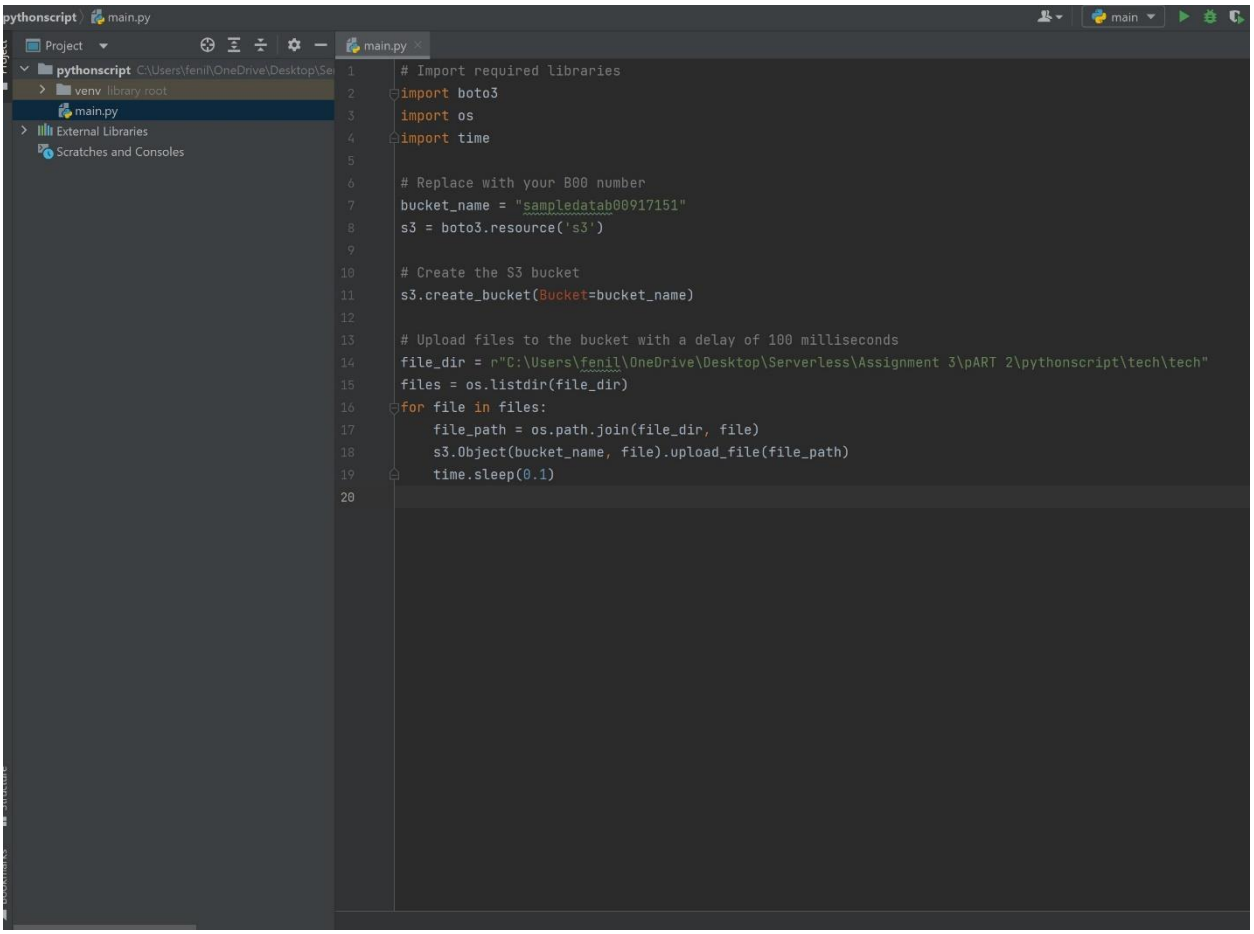

Fig 17: python script for pushing 400 text file of zip to sampledatab00917151 bucket

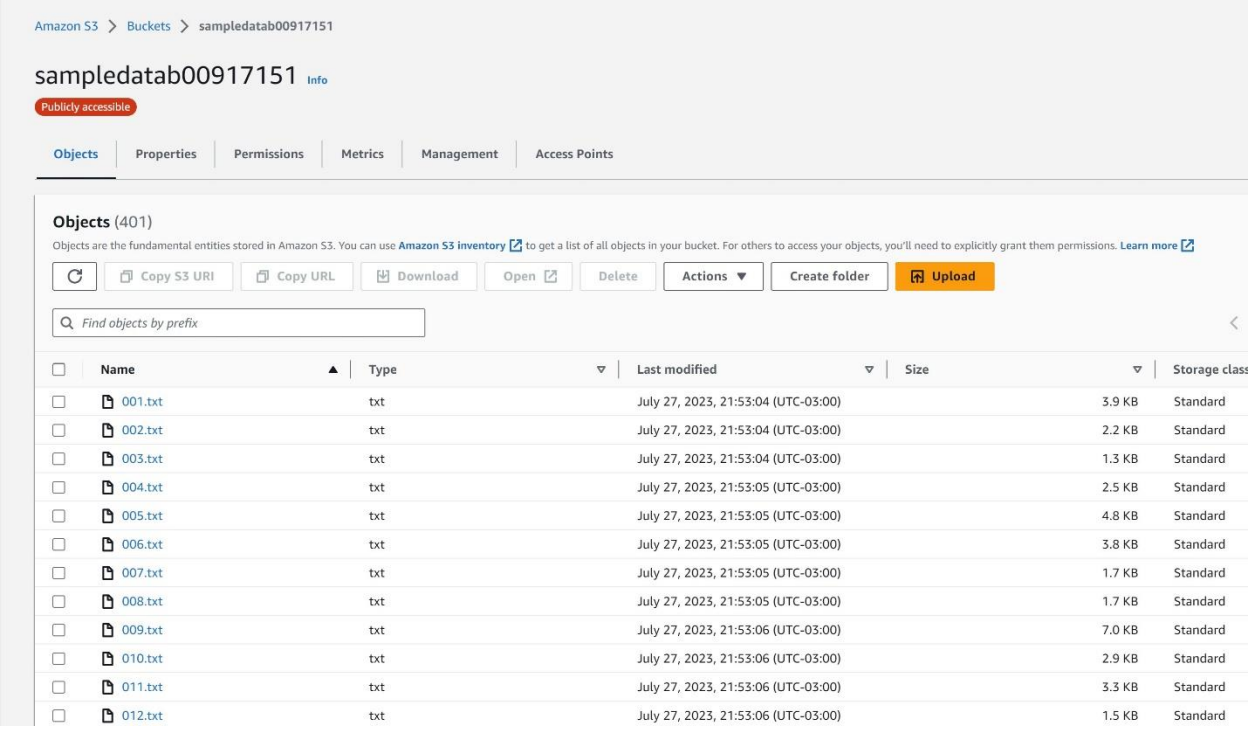

Fig 18 : sampledatab00917151 uploaded text files

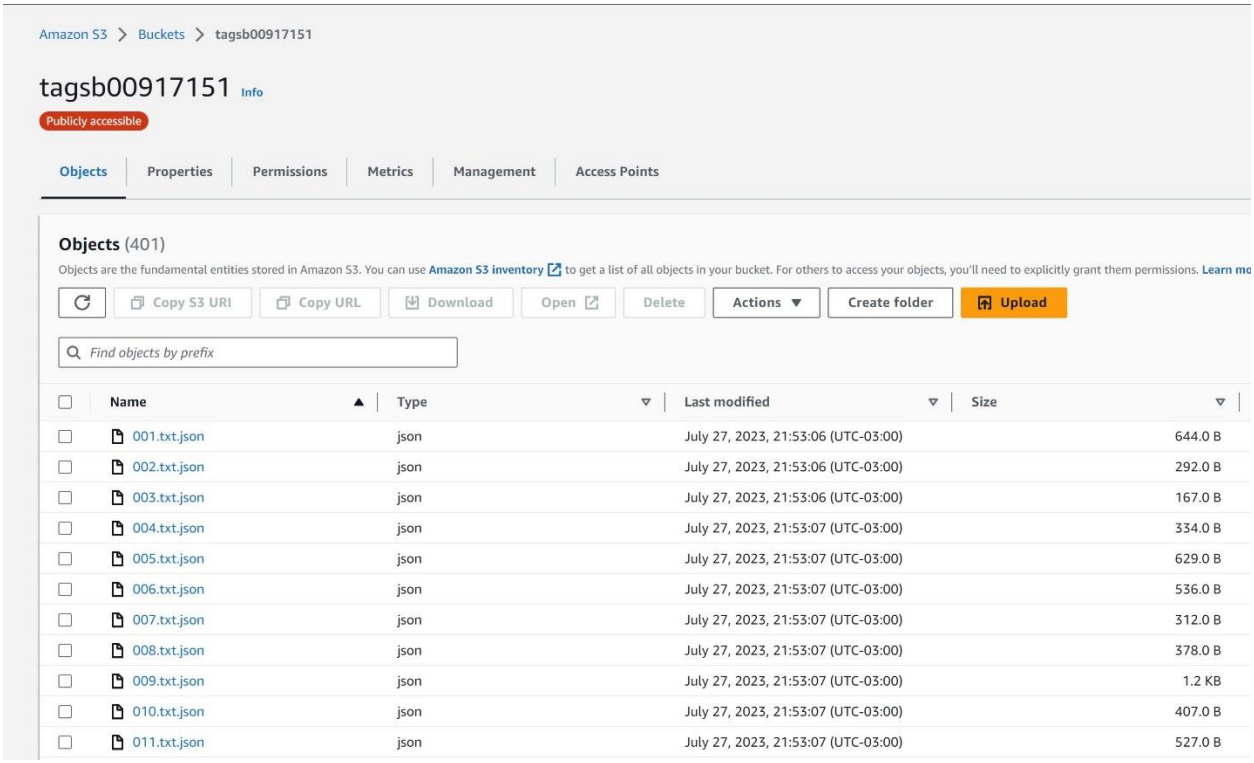

Fig 19: converted text files from sampledatab00917151 to tagsb00917151

Fig 20: tagsb00917151 s3 bucket's one file structure

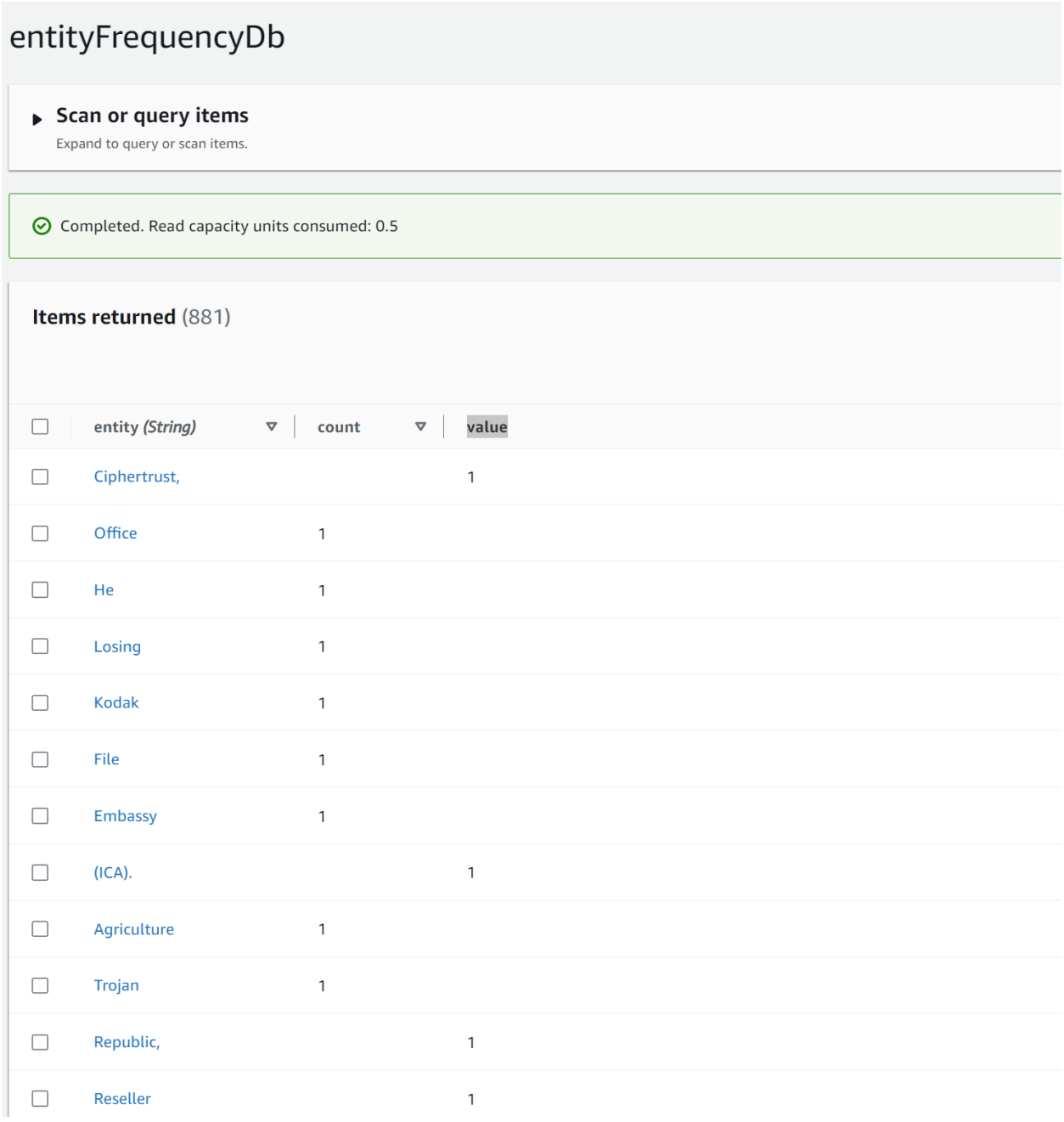

Fig 21:named entity stored in entityFrequencyDb

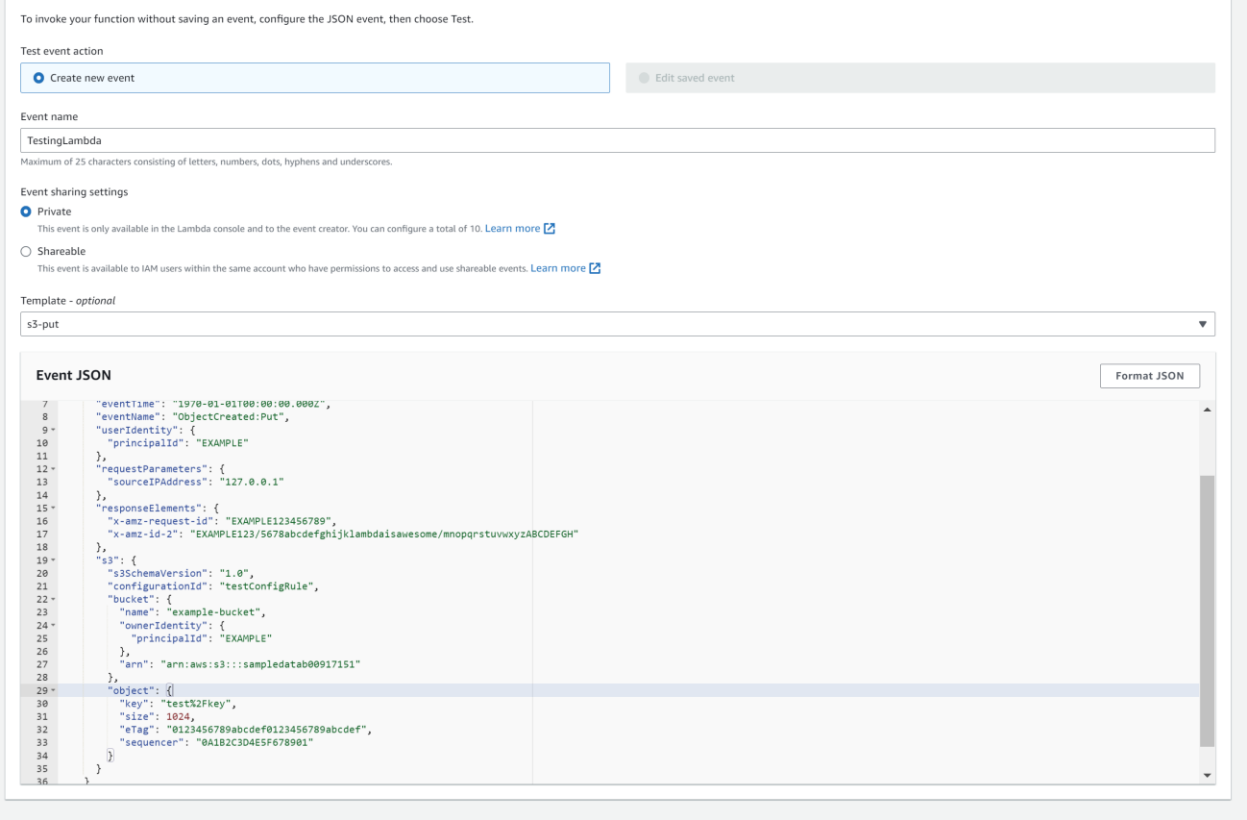

Fig 22: Functional Testing of the lambda function

Part C:

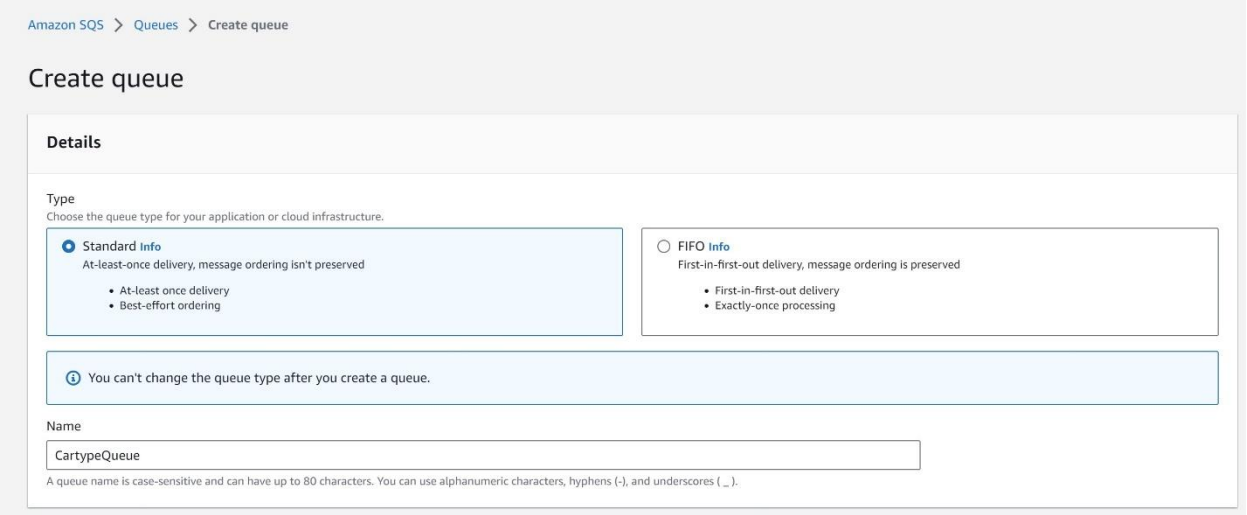

Fig 23: Creating a CartypeQueue

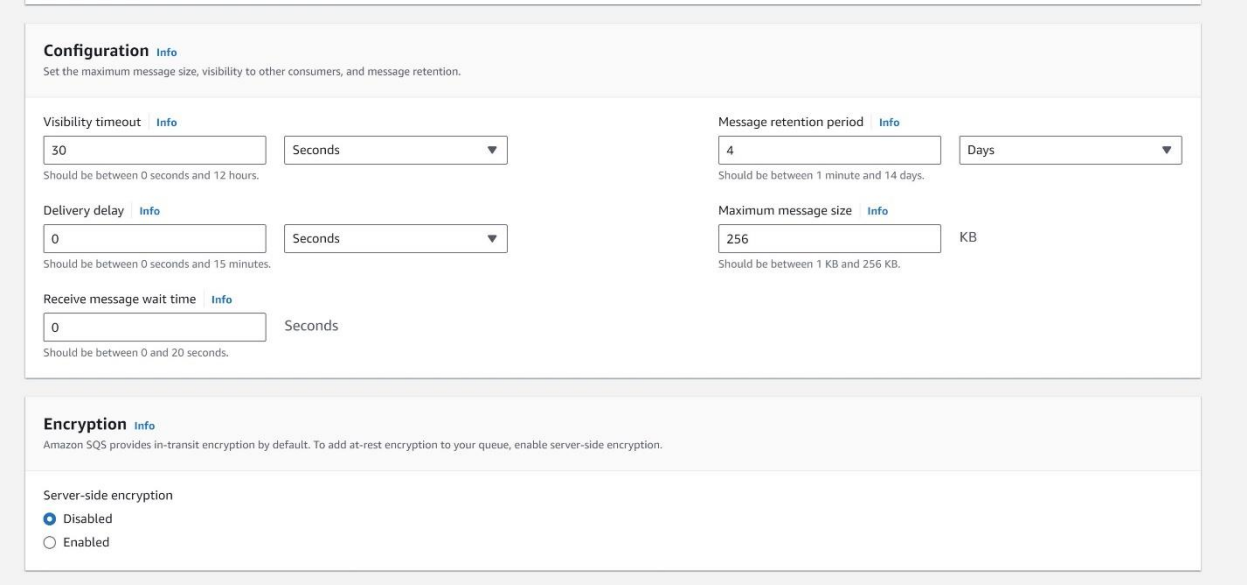

Fig 24: setting configuration for CartypeQueue

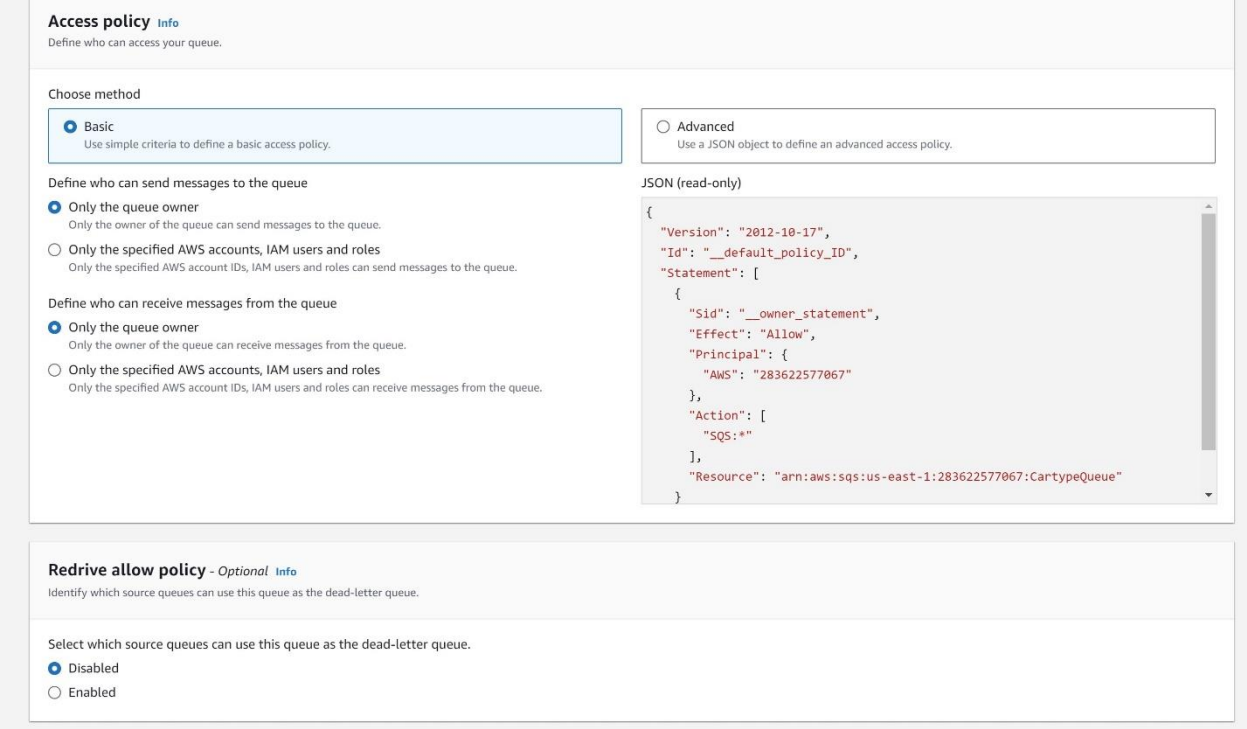

Fig 25: all the setting of the queue as default

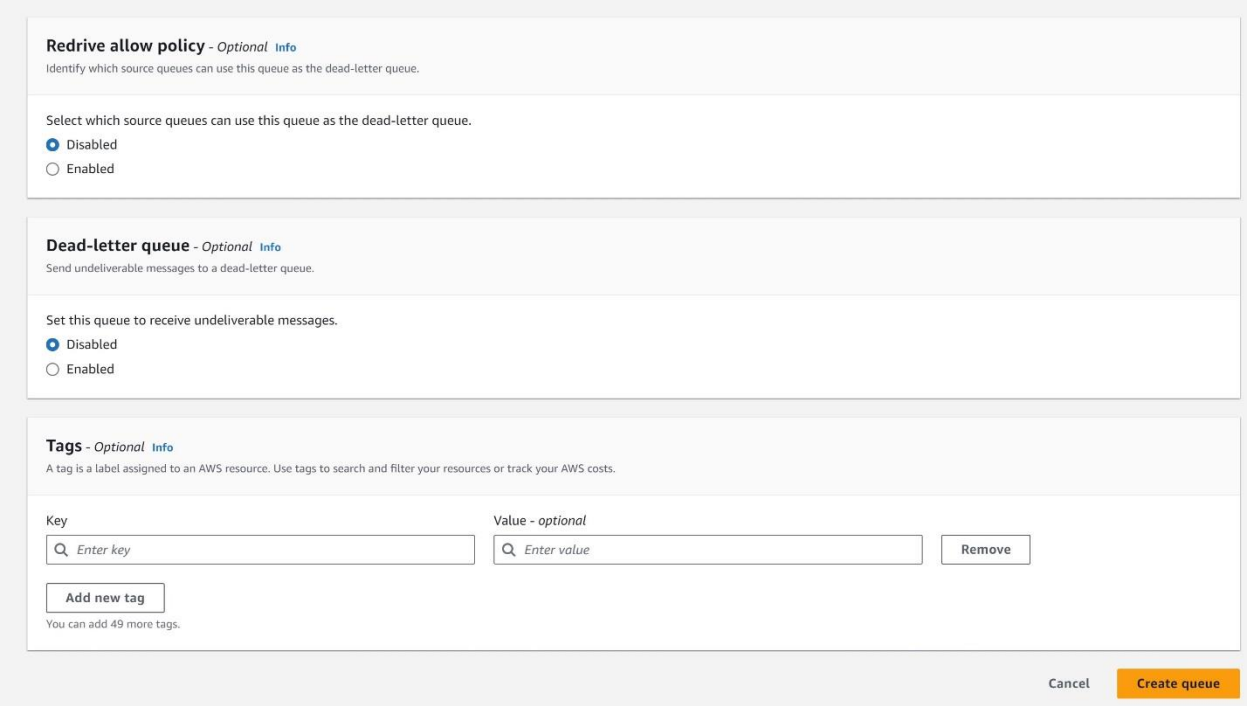

Fig 26: creating a queue with default setting

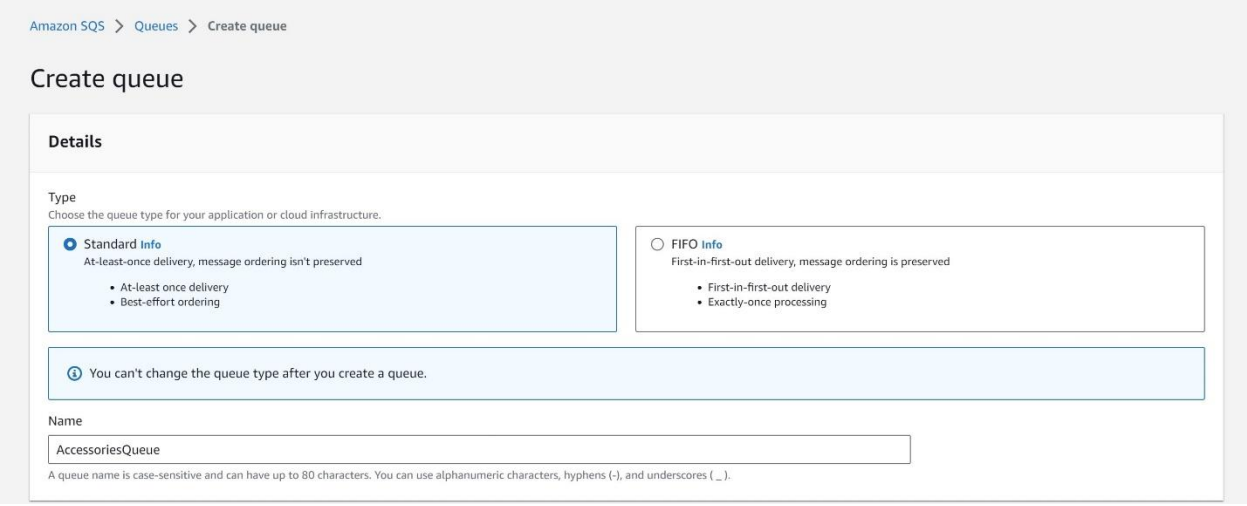

Fig 27: Creating a 2nd queue called AccessoriesQueue

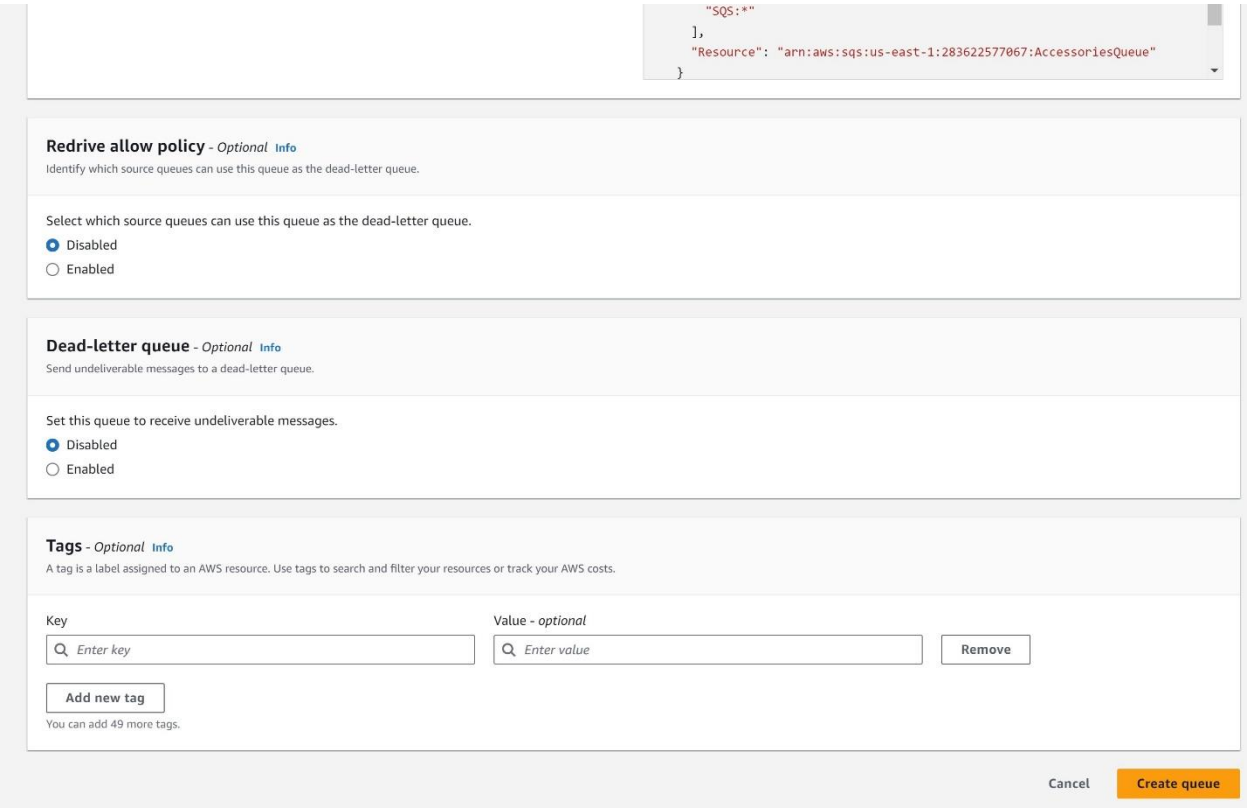

Fig 28: creating queue with default setting

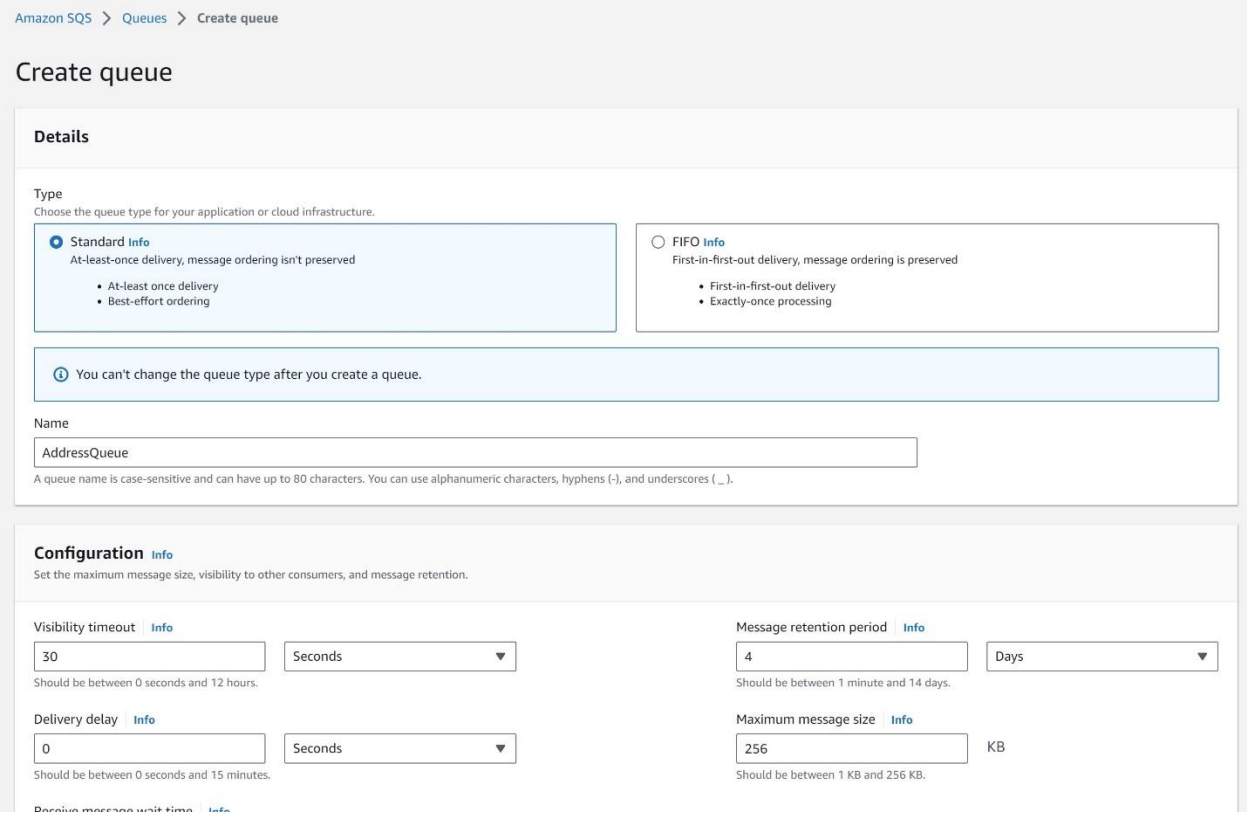

Fig 29: Creating a 3rd queue called AddressQueue

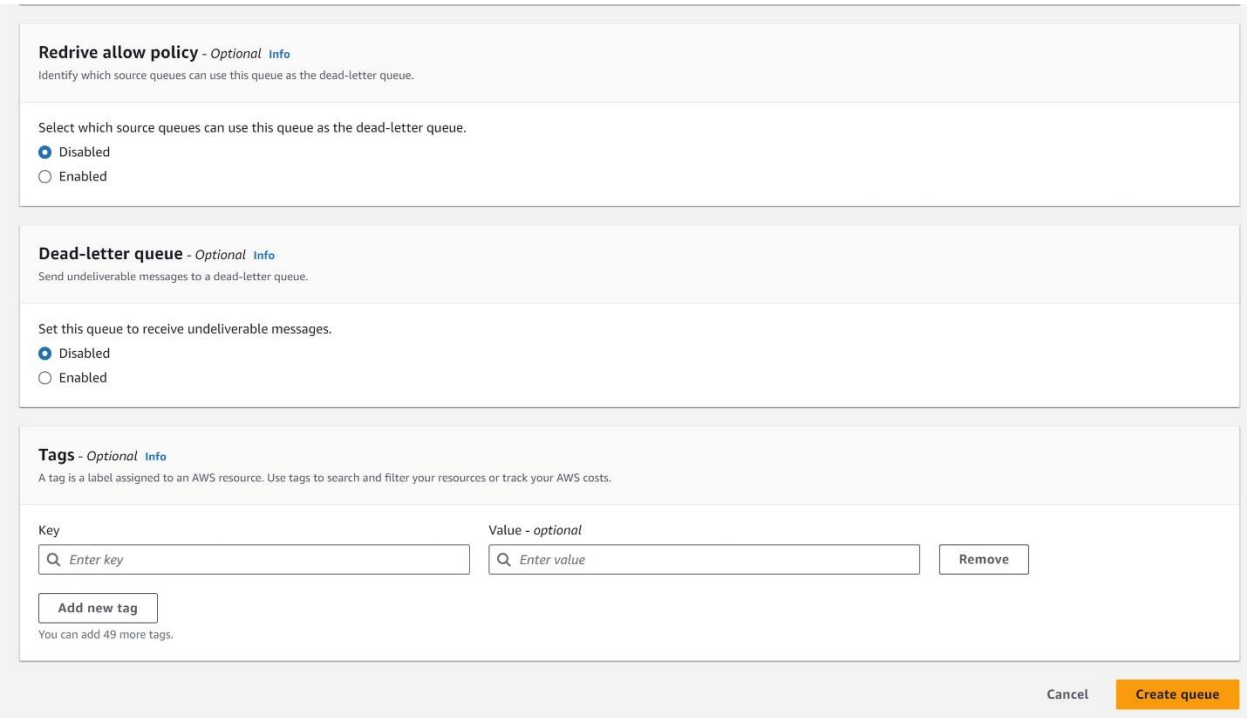

Fig 30: Creating queue with other default setting

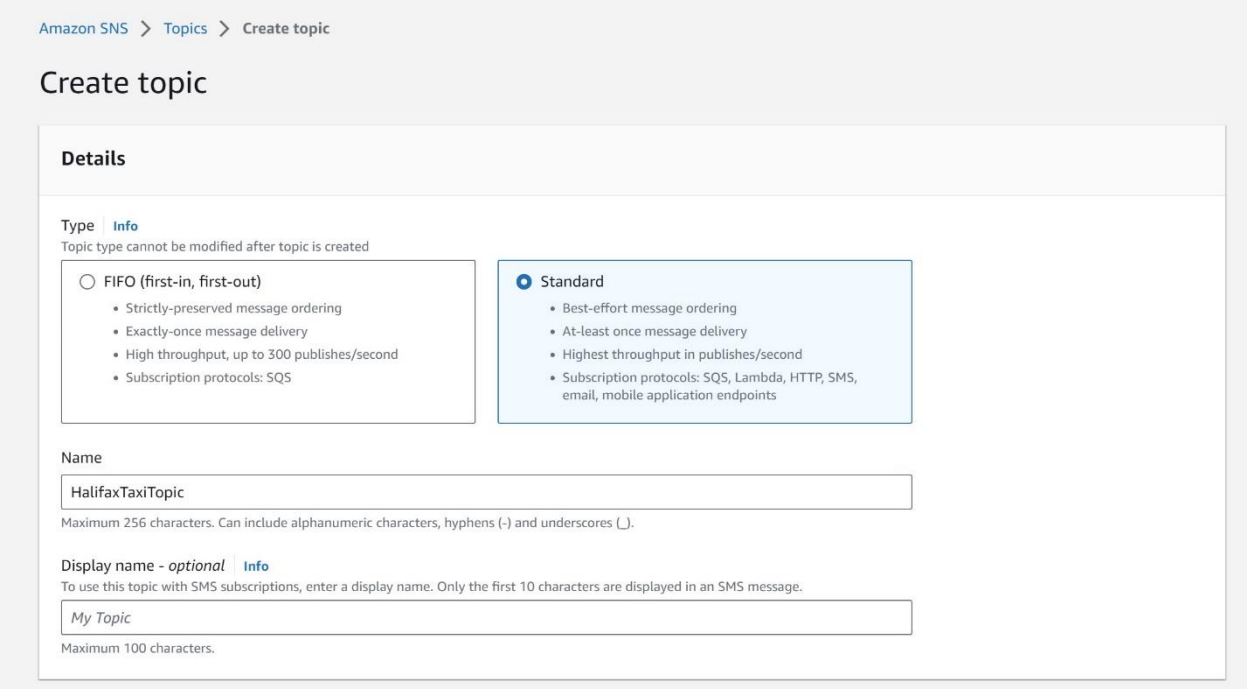

Fig 31 : Creating a standard SNS topic HalifaxTaxiTopic

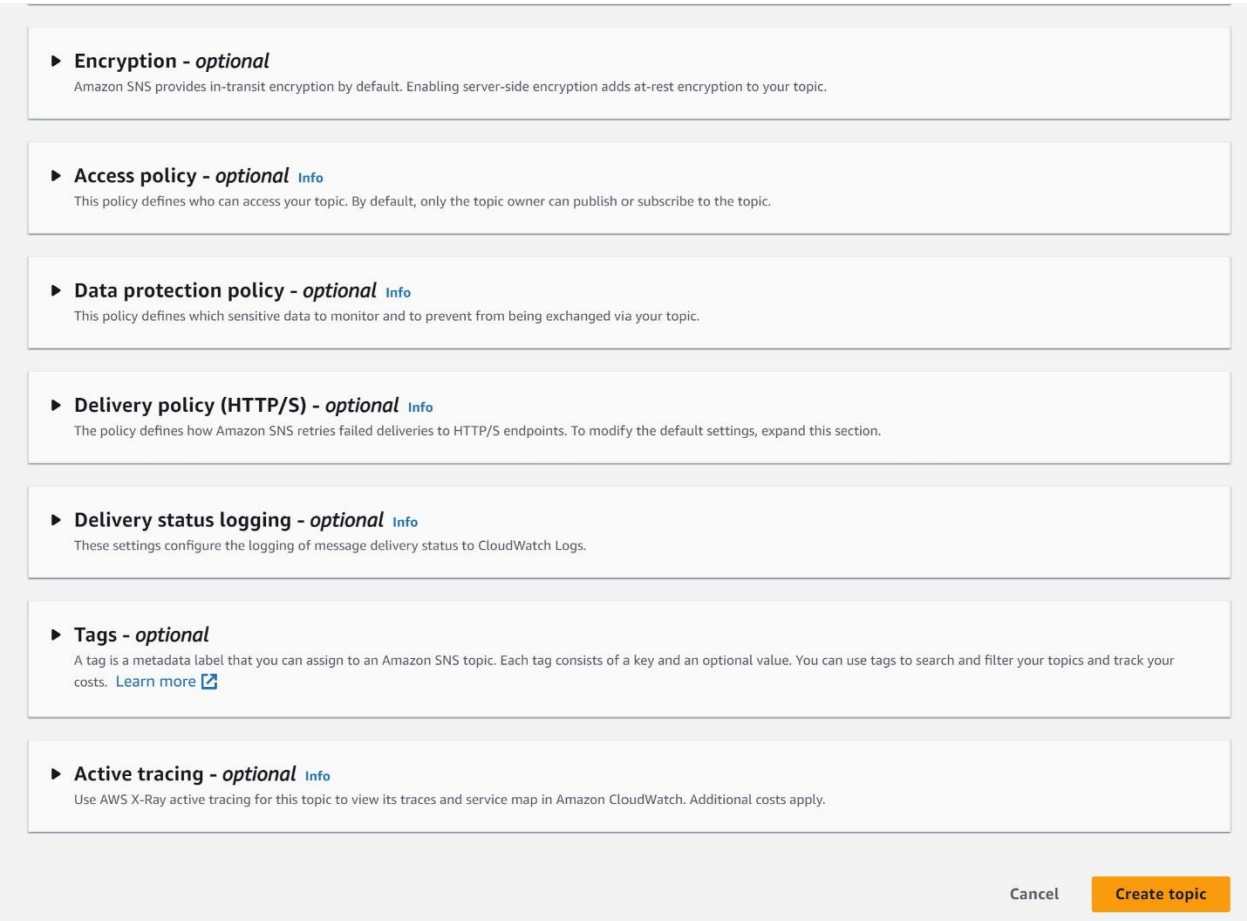

Fig 32: setting it as default setting and creating the topic

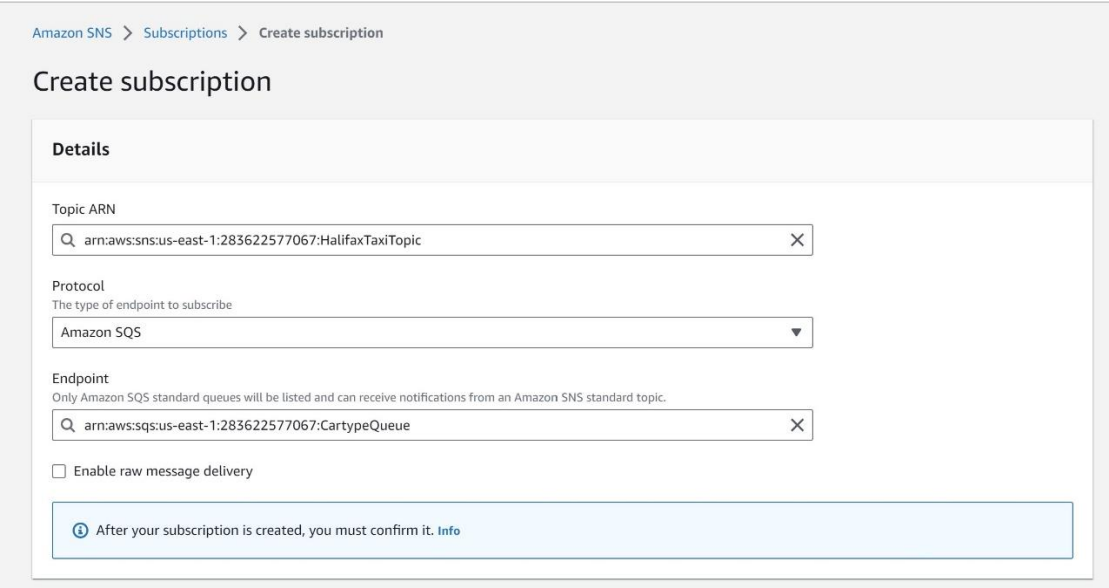

Fig 33: Creating a subscription with sqs queue CartypeQueue

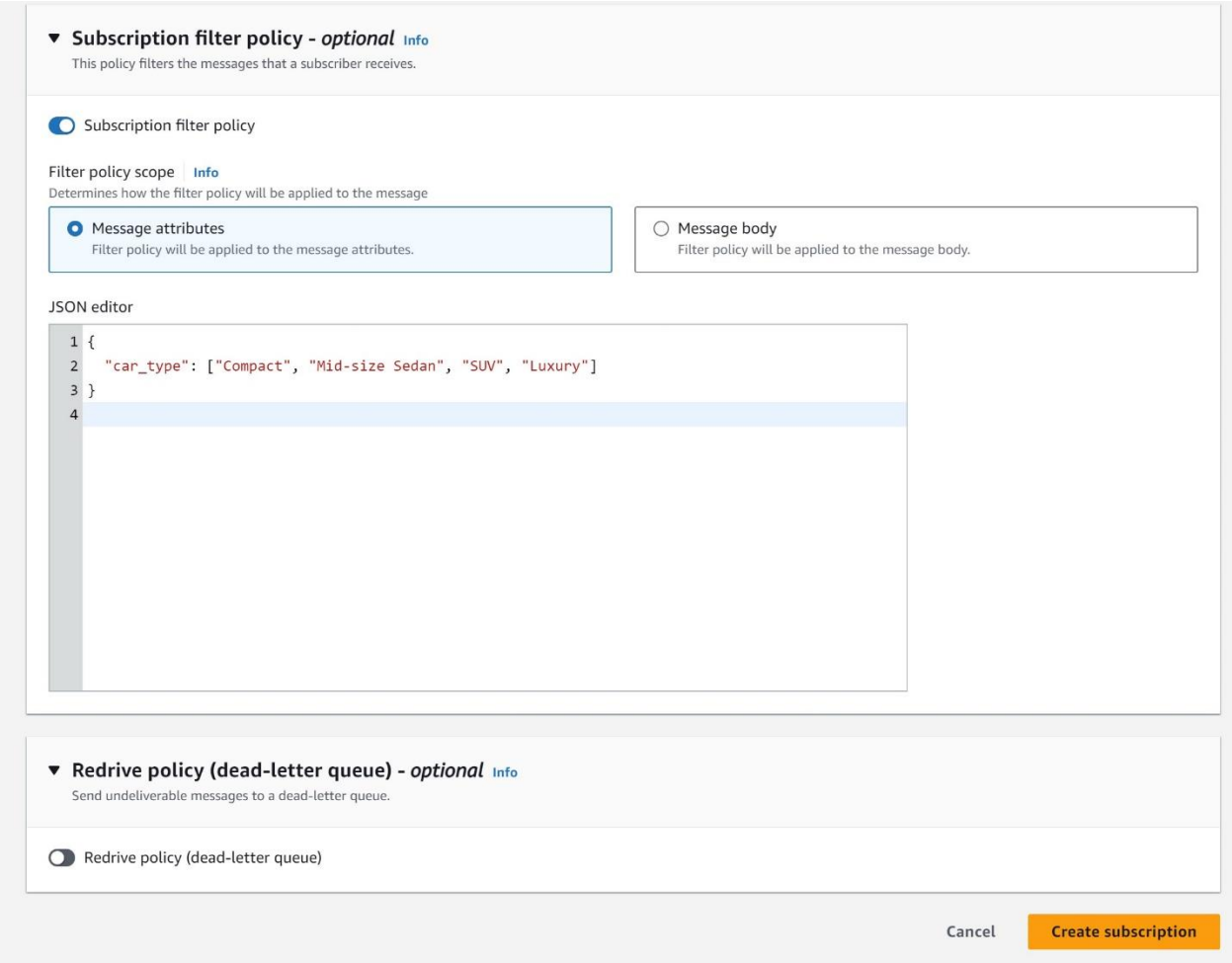

Fig 34: applying filter policy for that subscription

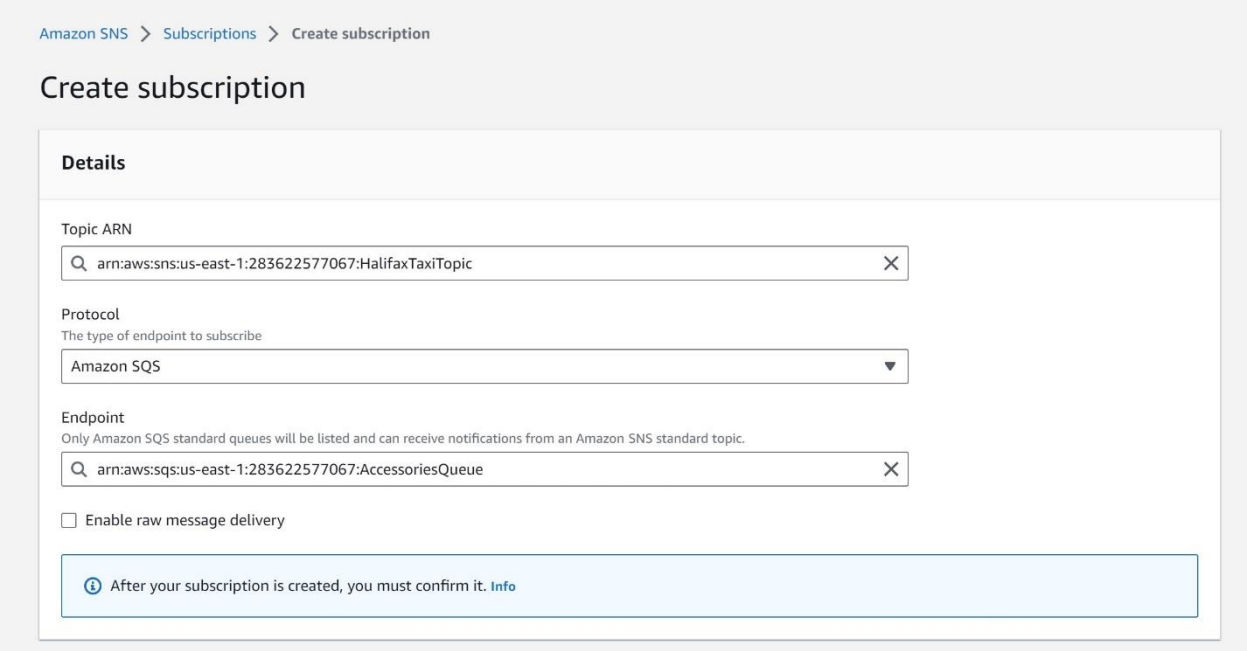

Fig 35: Creating a 2nd subscription with sqs queue AccessoriesQueue

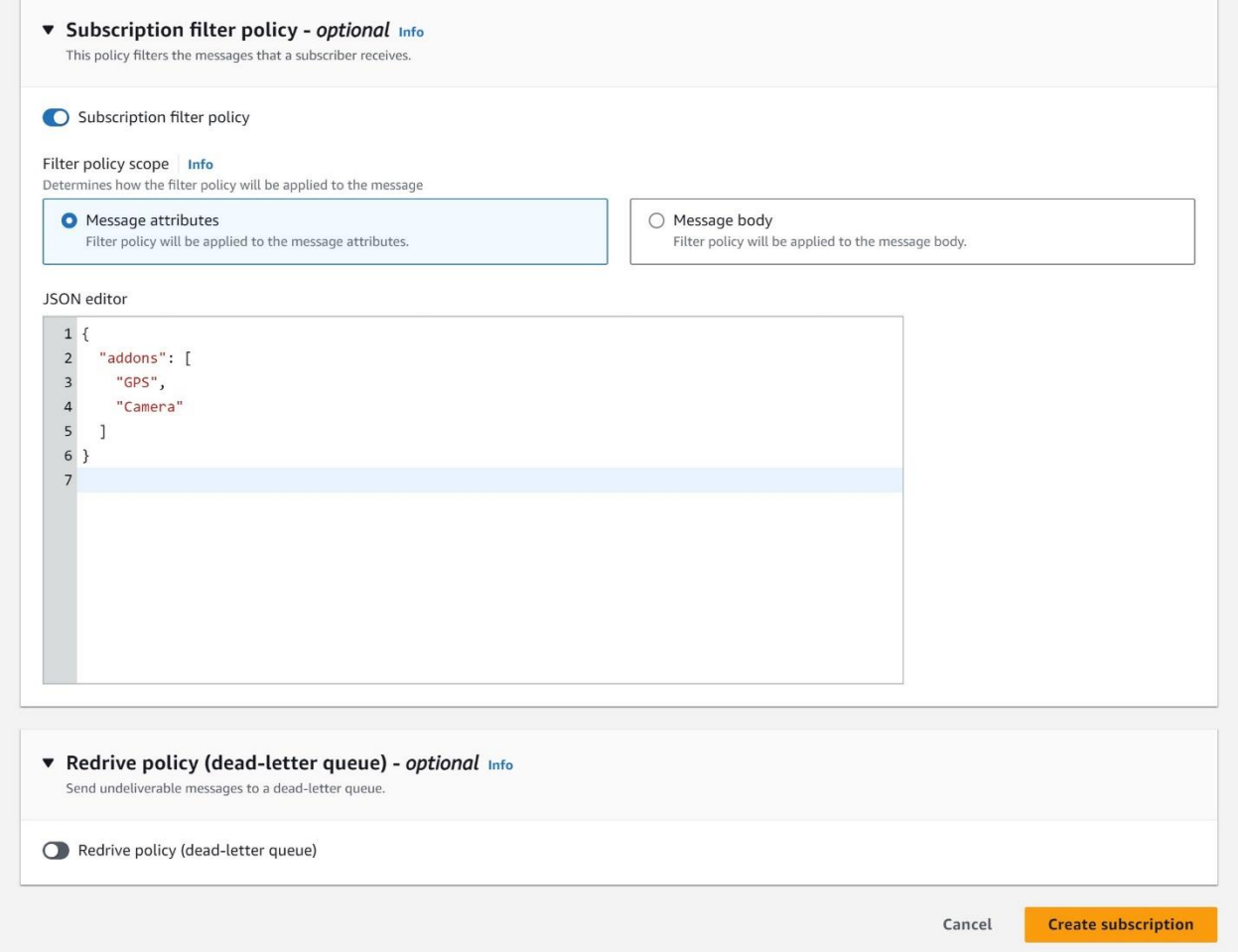

Fig 36: applying filter policy for that subscription

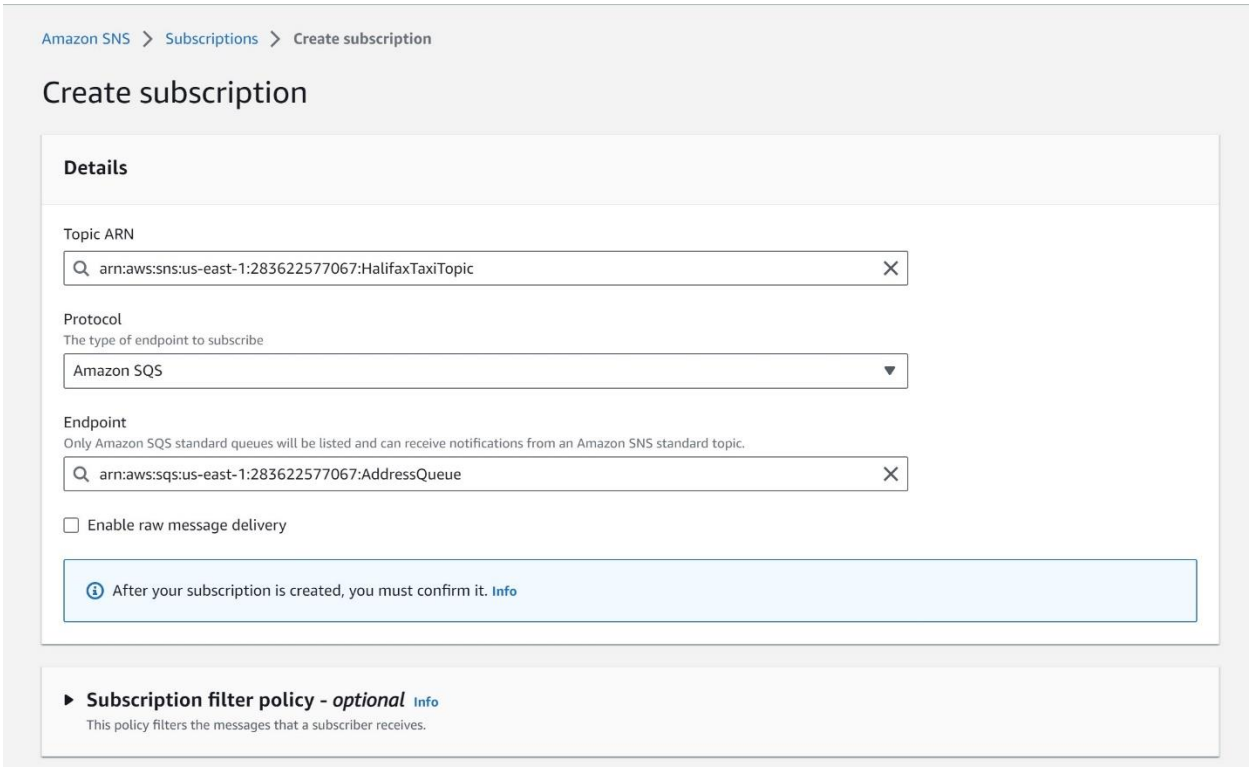

Fig 37: Creating a 3rd subscription with sqs queue AddressQueue

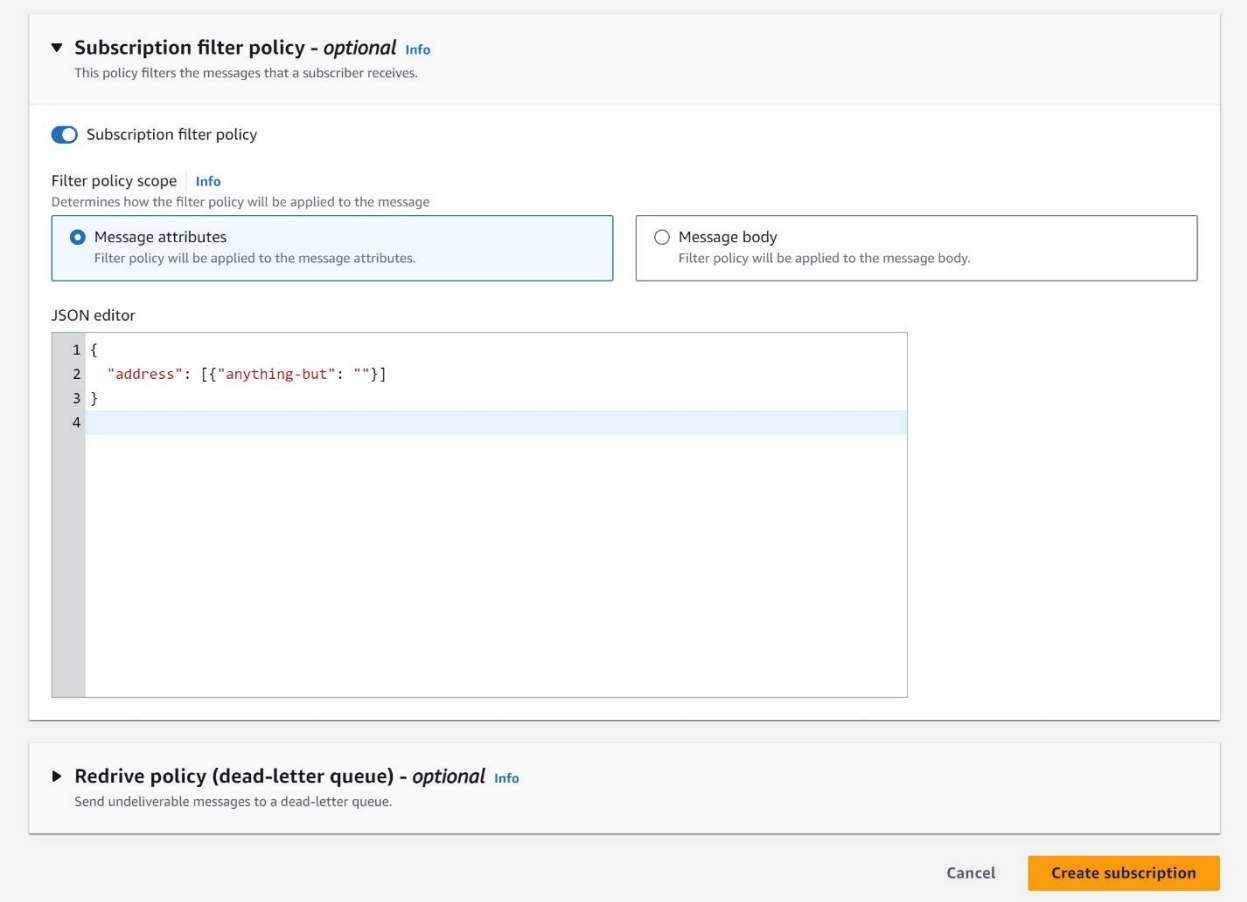

Fig 38: applying filter policy for that subscription

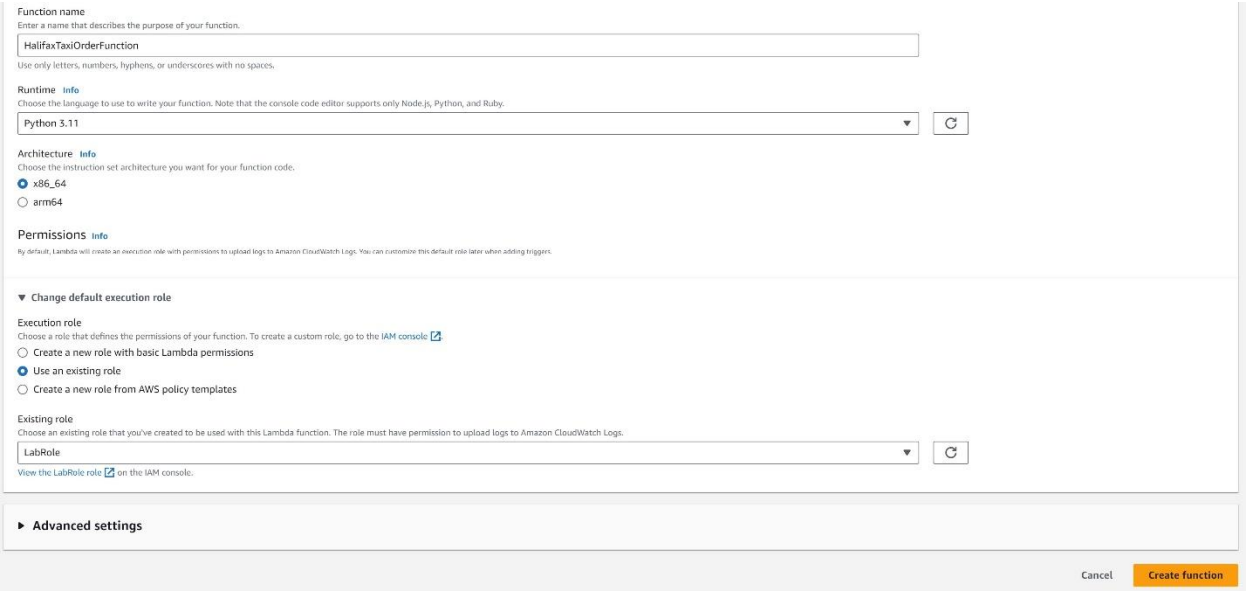

Fig 39: creating HalifaxTaxiOrderFunction with LabRole

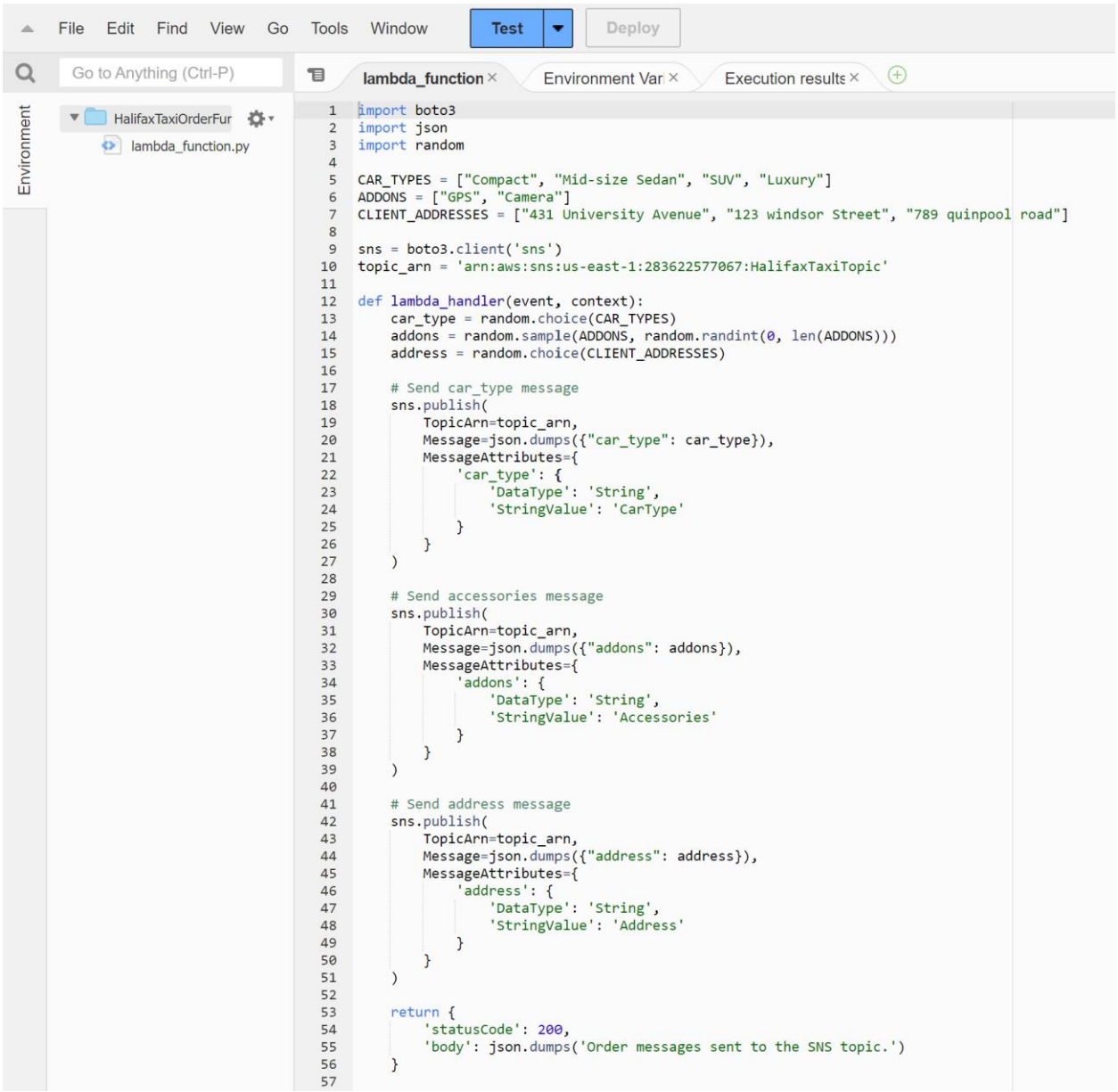

Fig 40: python code for HalifaxTaxiOrderFunction

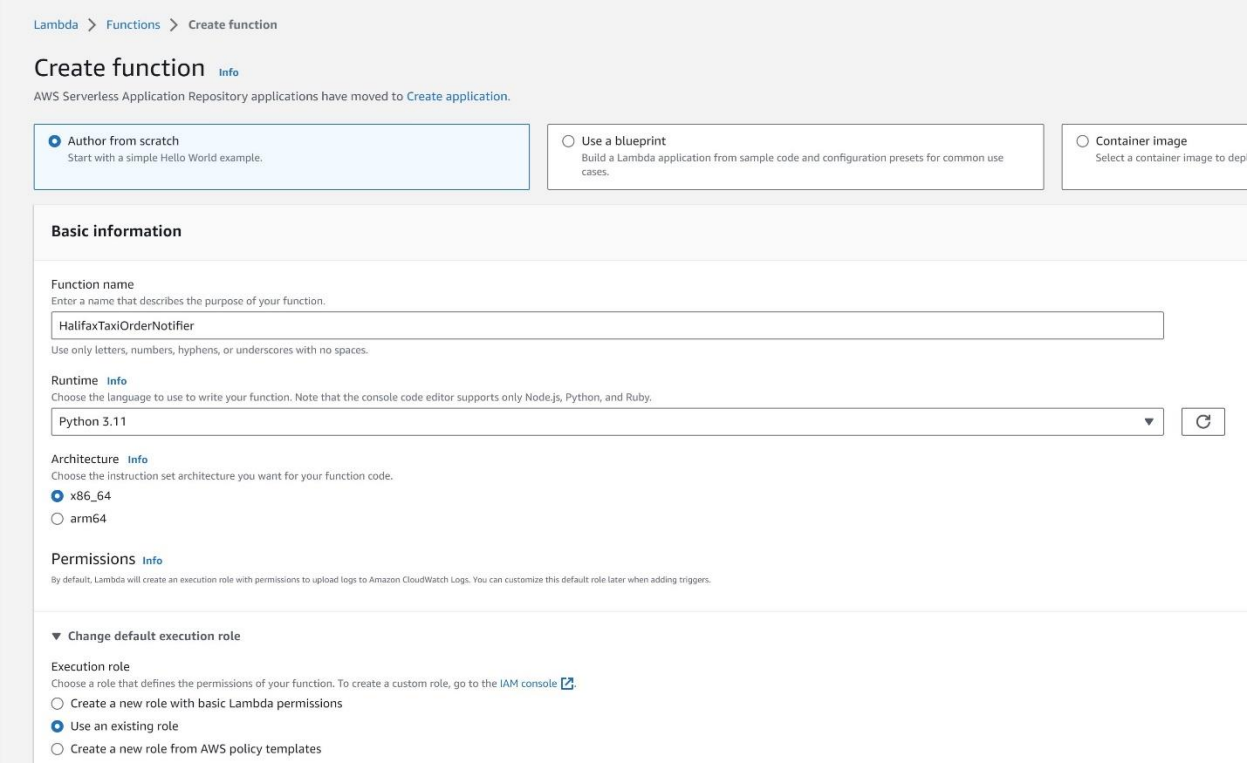

Fig 41: creating HalifaxTaxiOrderNotifier with LabRole

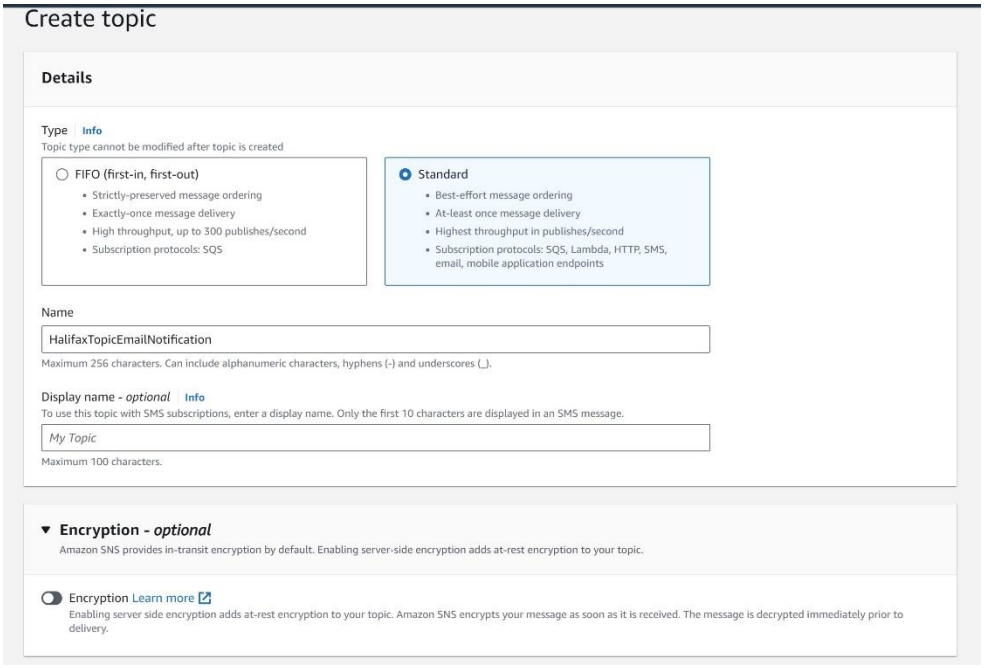

Fig 42: creating a SNS topic for email notification called HalifaxTopicEmailNotification

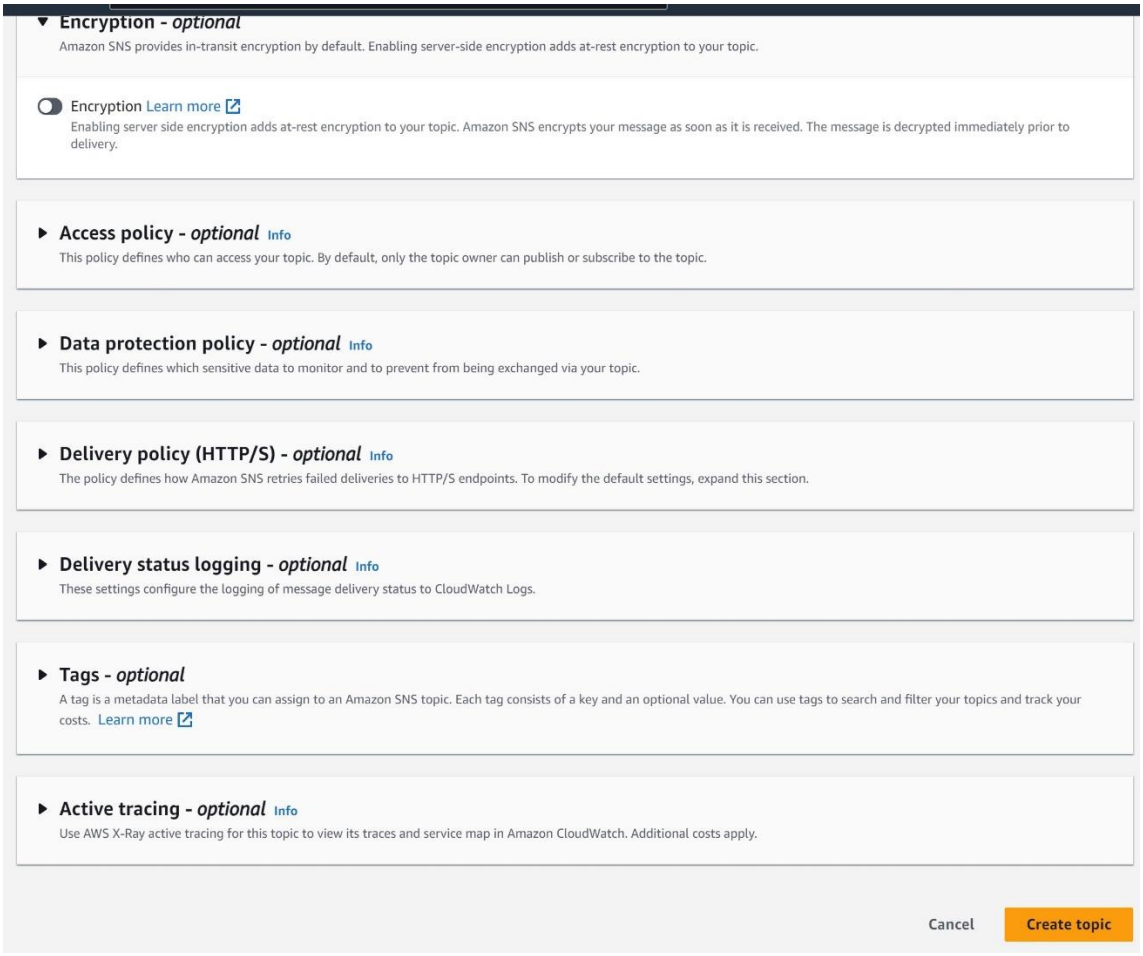

Fig 43: Setting up sns with default settings

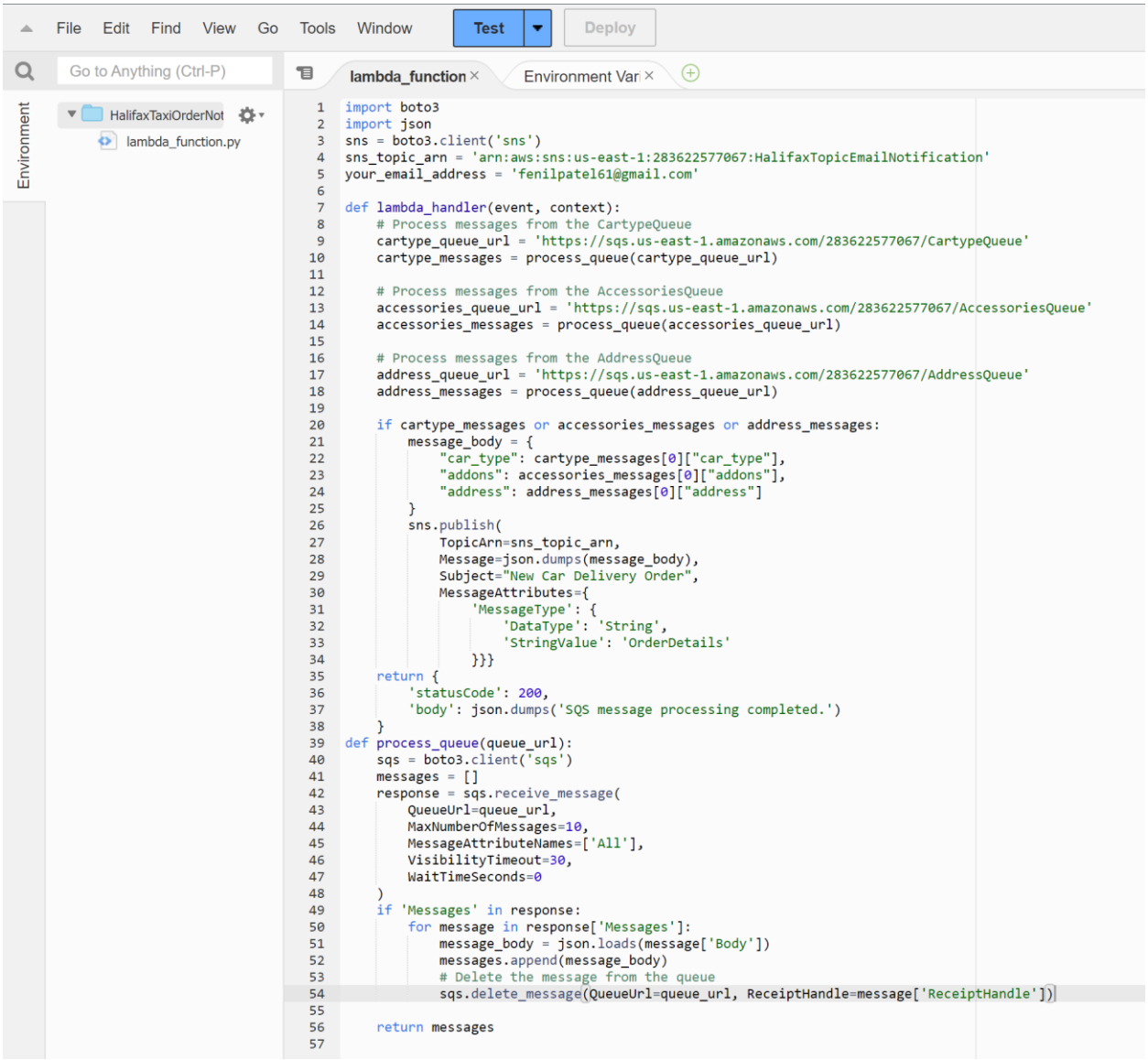

Fig 44: Lambda code of HalifaxTaxiOrderNotification for fetching messages from 3 sqs and sending sns notification to email user

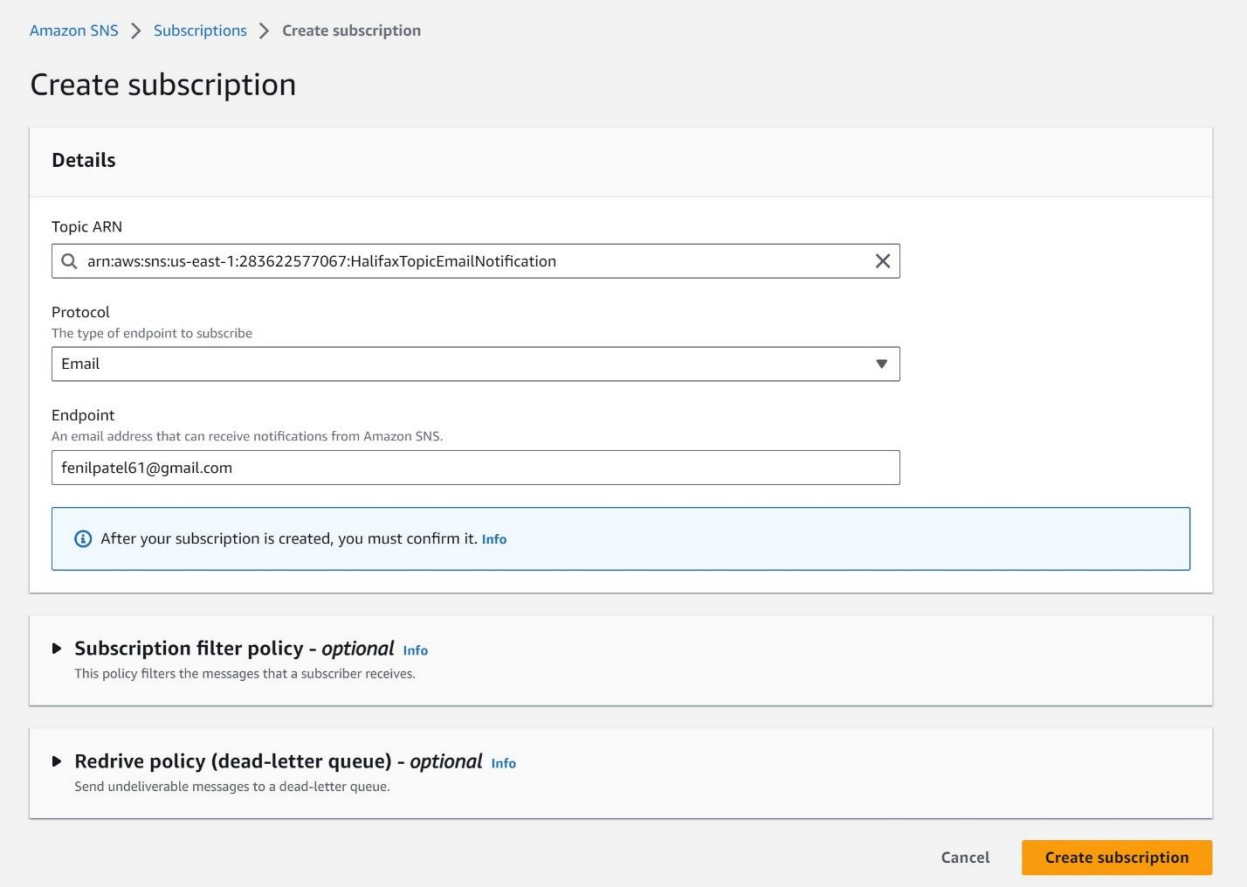

Fig 45: creating an email subscription for the user

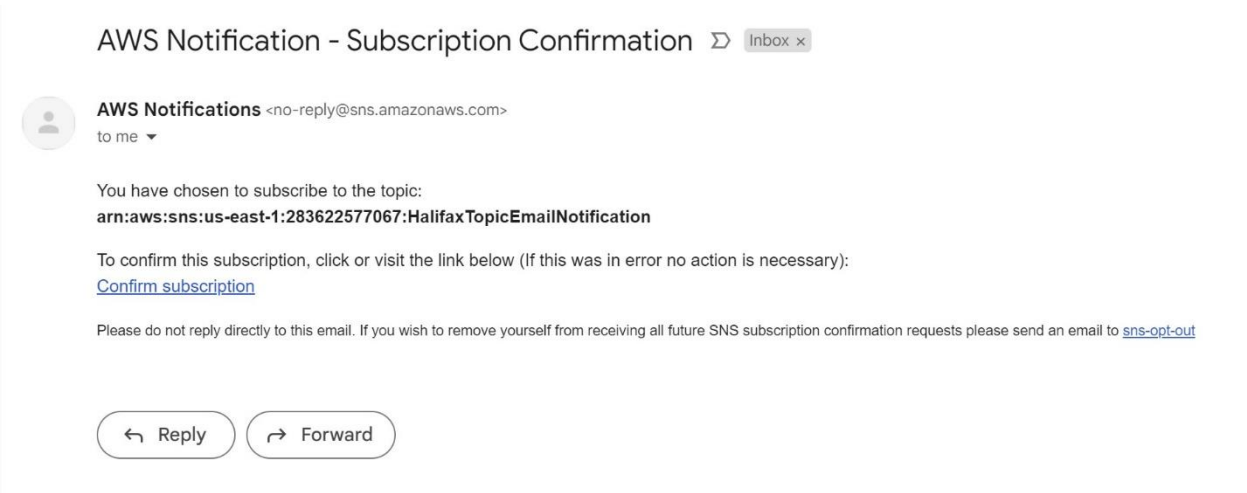

Fig 46: user receives the confirmation through mail

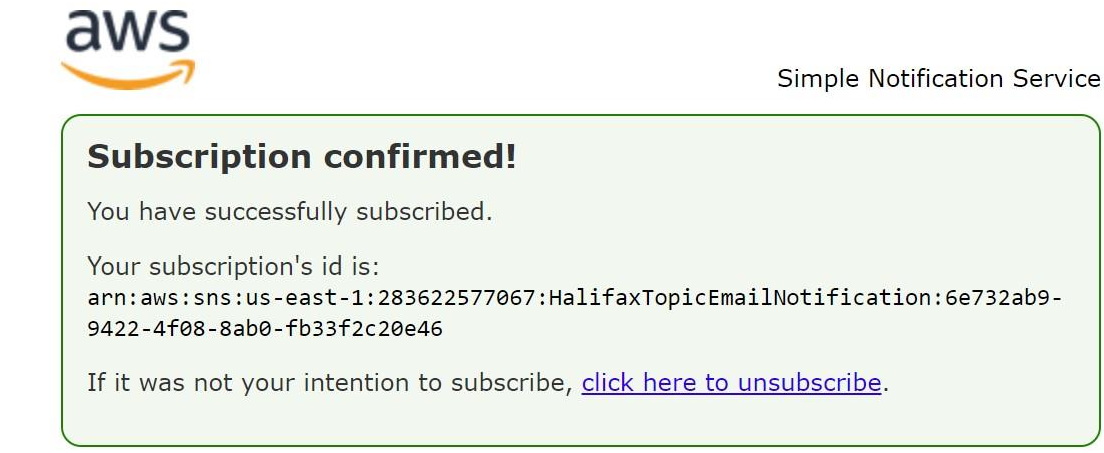

Fig 47: User approve confirmation

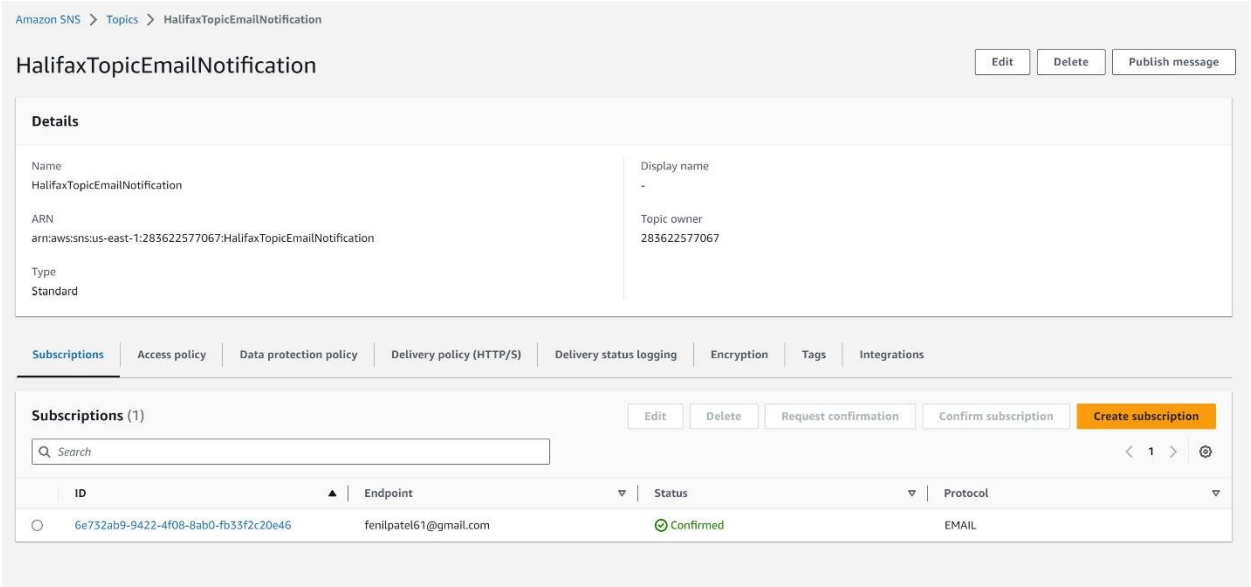

Fig 48: Subscription activated

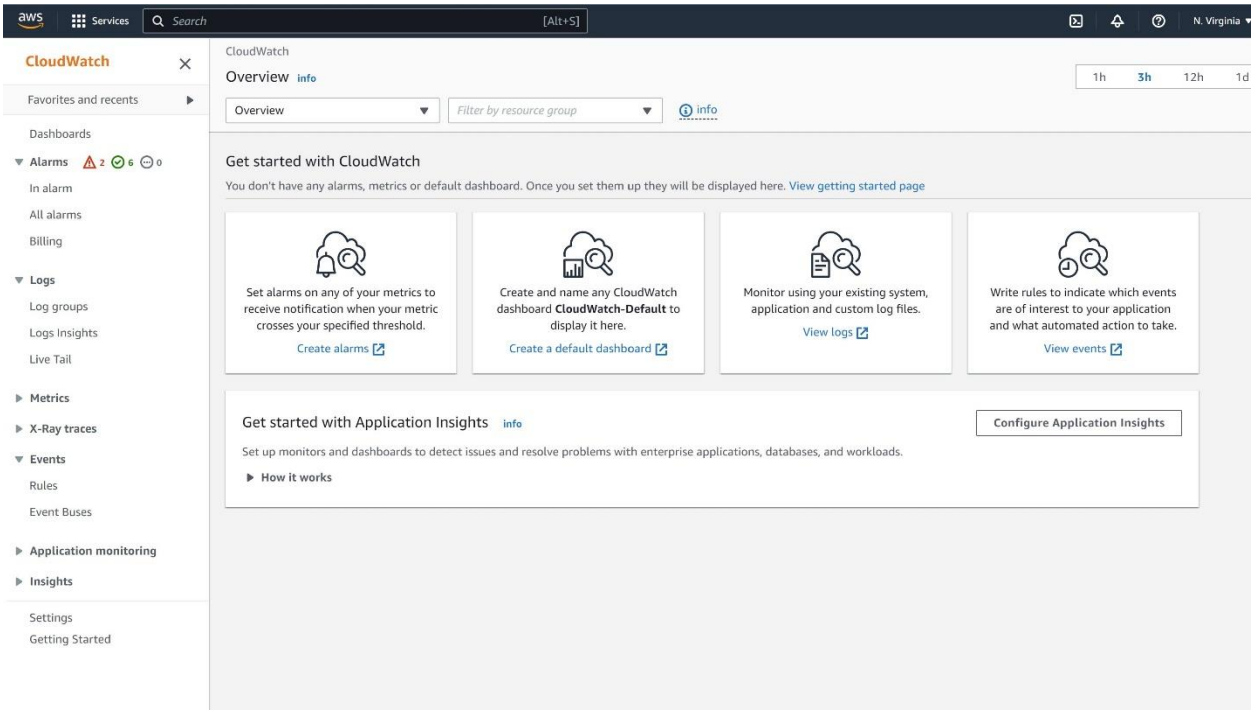

Fig 49: setting an event for lambda trigger every 2 minutes through CloudWatch

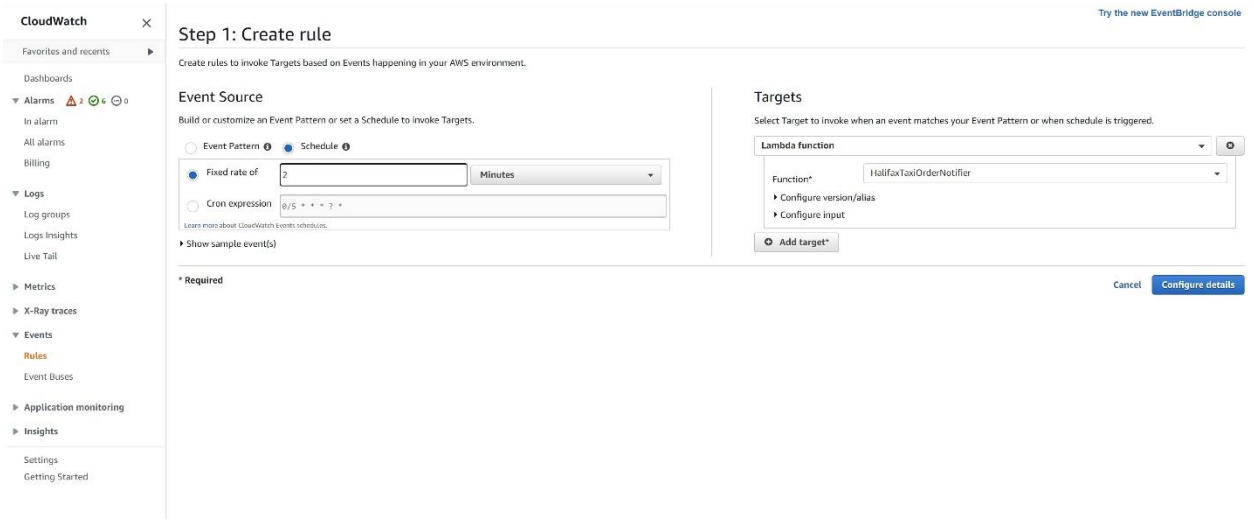

Fig 50: Creating a rule for that lambda trigger in cloudwatch

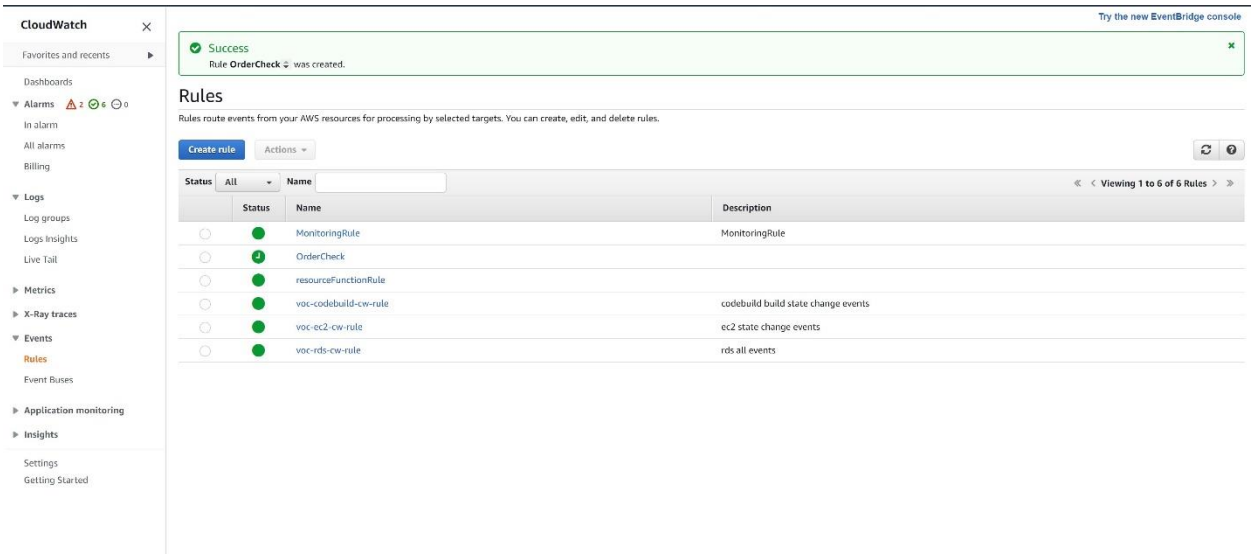

Fig 51: OrderCheck Rule has been created

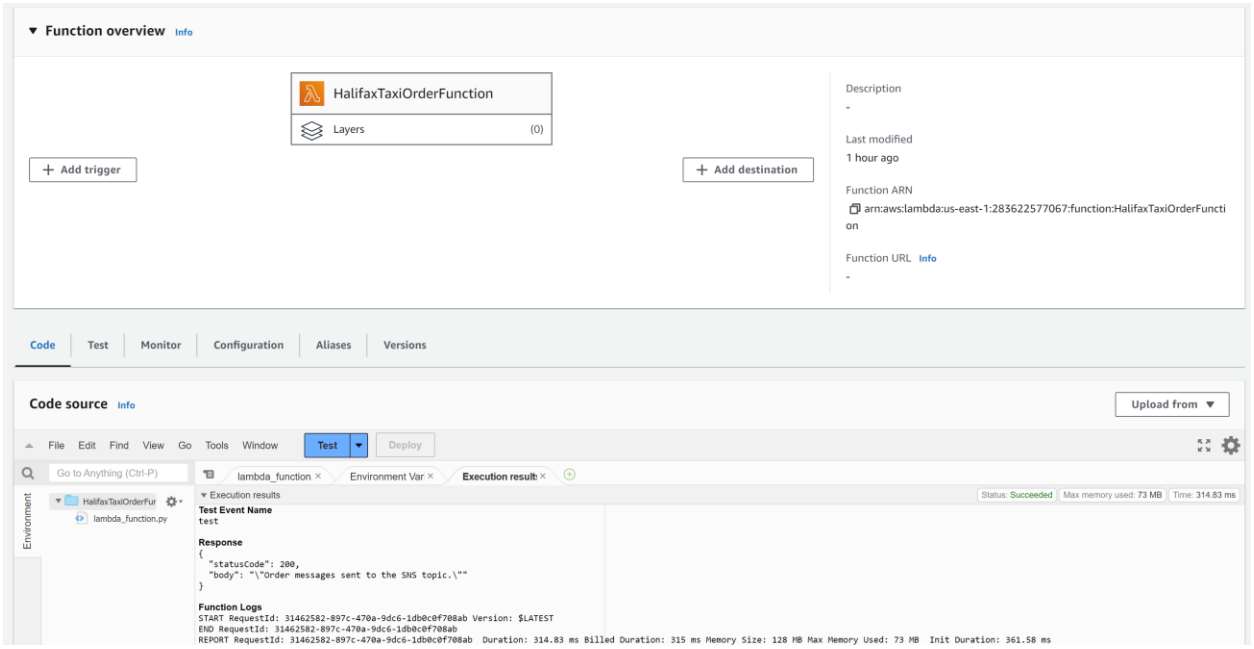

Fig 52: testing 1st lambda function HalifaxTaxiOrderFunction

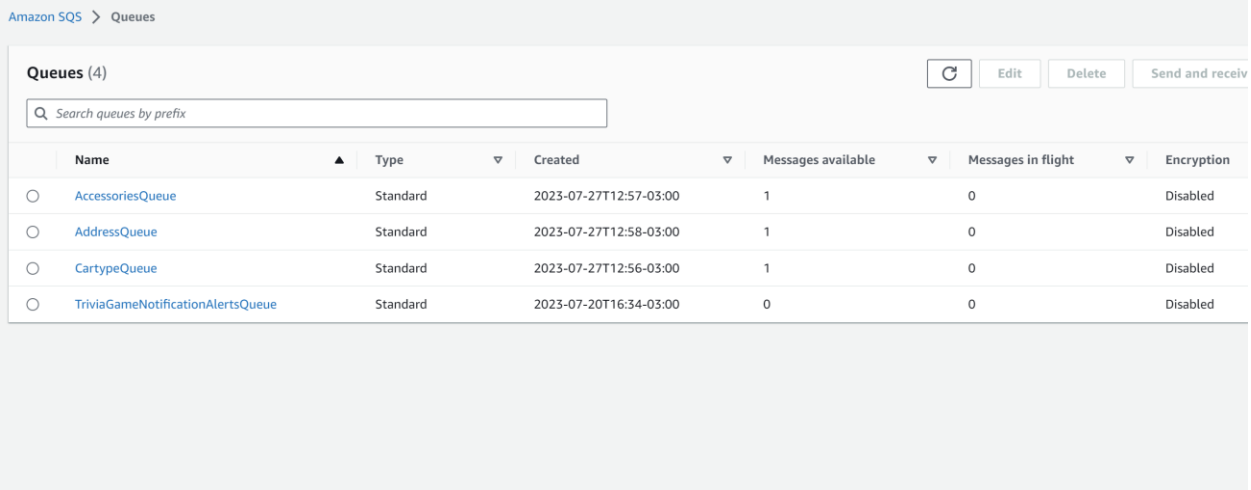

Fig 53: sns topic submitted from 1st lambda and then the message has been received to 3 sqs queues with filter policy applied

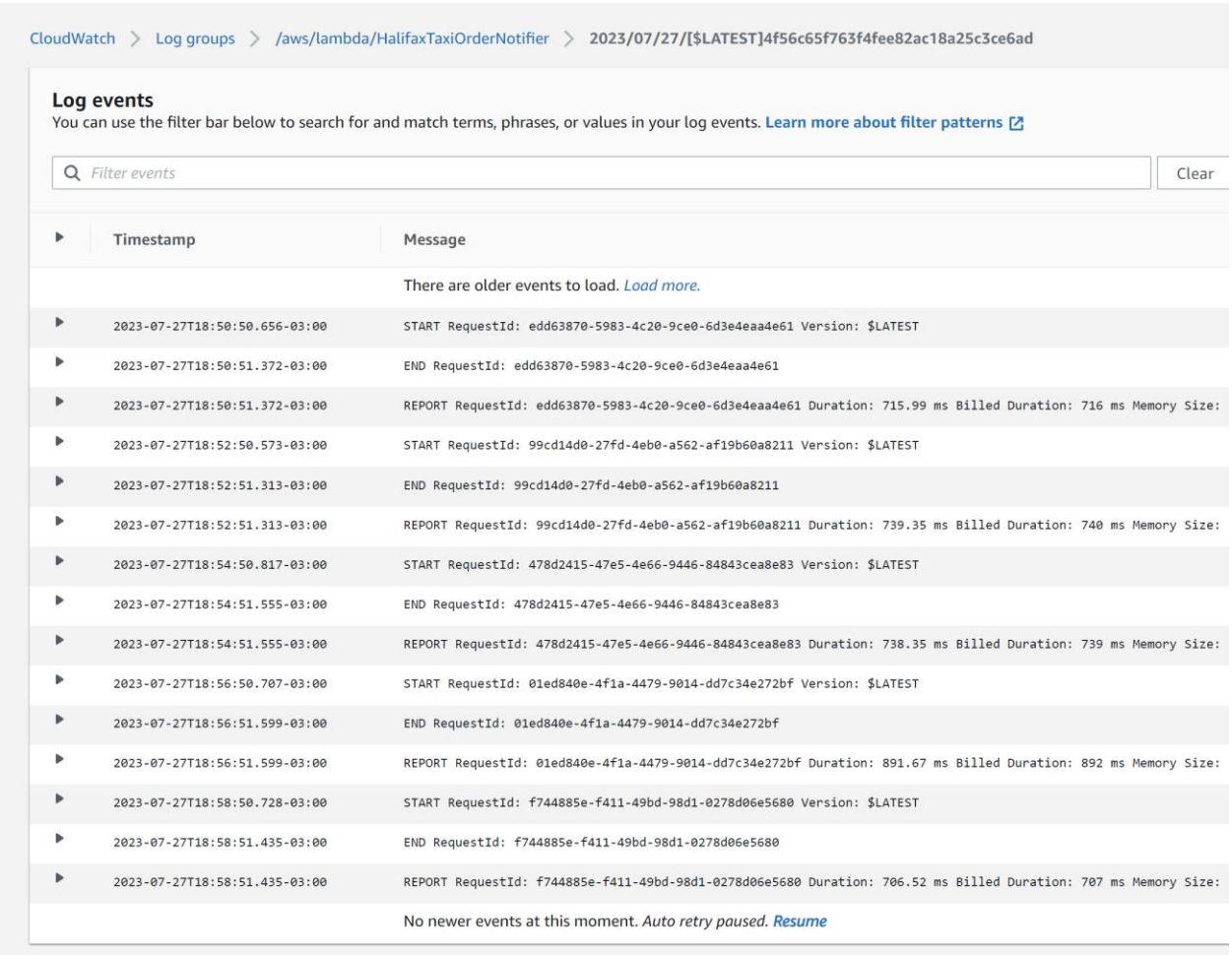

Fig 54: Cloud watch logs for the 2nd lambda

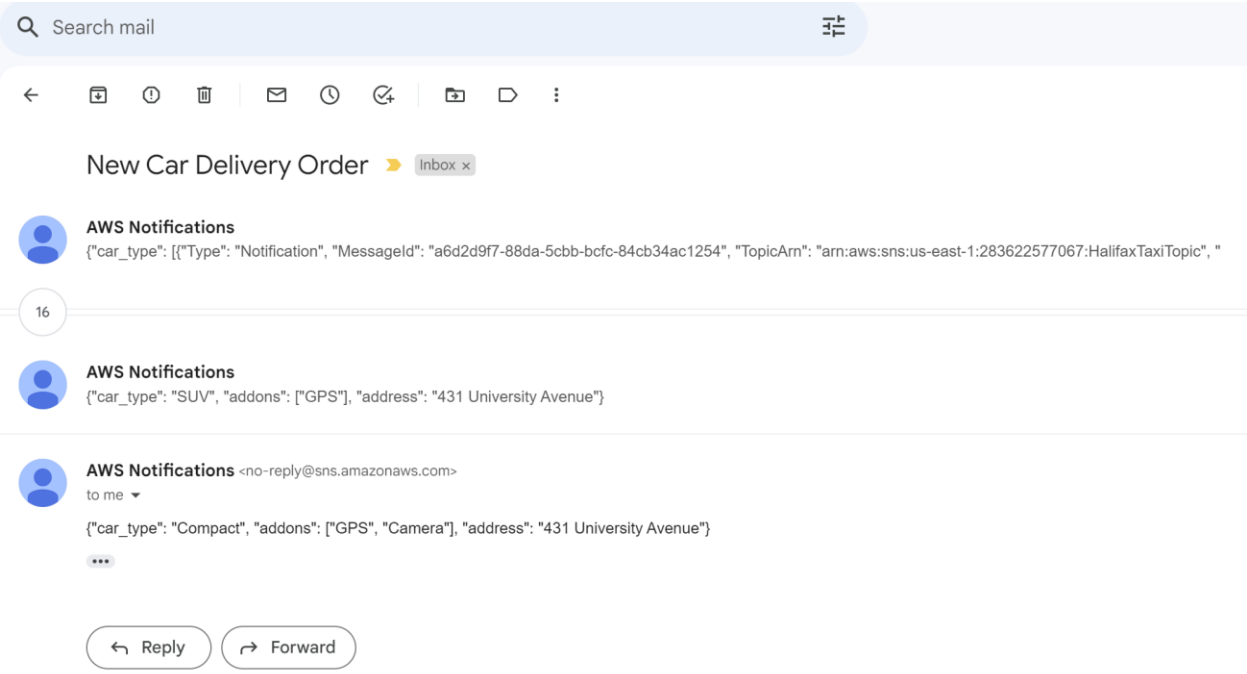

Fig 55: users receive the email for the notification

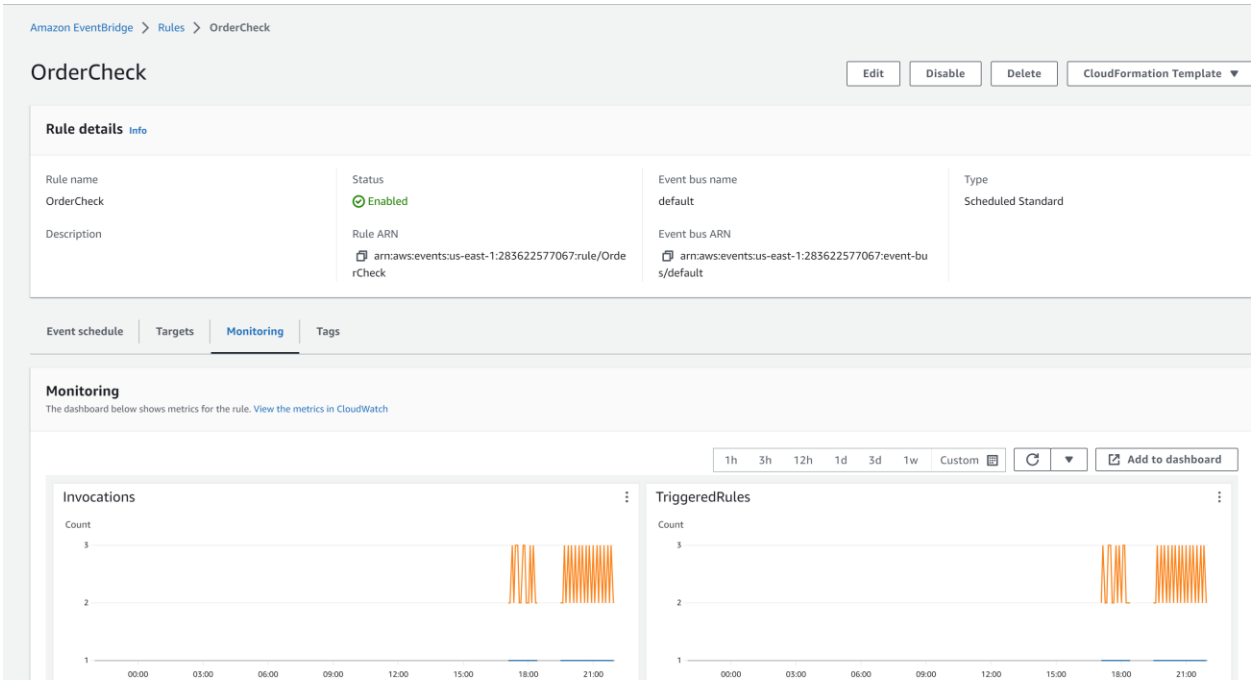

Fig 56: monitoring of OrderCheck rule for 2nd lambda trigger

References :

[1] Walmsley, G., & Bie, J. (1983). *Kinesis*. Amazon. https://aws.amazon.com/kinesis/