AWS Capstone Use Case 6

Send Fanout Event Notifications with Amazon Simple Queue Service (SQS) and Amazon Simple Notification Service (SNS)

Each use case report should have following sections:

Sections	Points
Implemented Solution Architecture diagram with detailed explanation for each component / service used.	8
Implementation Screenshot with brief description for every screenshot. Code / Script files (if applicable) *Screen shots must have your AWS / Azure account ID clearly visible along with your laptop's date and time*	8
Cost Analysis of the implemented Solution – Assume your solution is used by 1000 users for a month, and give monthly billing estimates.	2
Lessons & Observations	2
TOTAL (for 1 use case)	**20**

A. Implemented Solution Architecture diagram with detailed explanation for each component / service used.

Send Fanout Event Notifications with Amazon Simple Queue Service (SQS) and Amazon Simple Notification Service (SNS):

This use case is implemented in two ways:

I) Without SNS message Filtering

In this scenario, identical messages are "pushed" to multiple subscribers, which eliminates the need to periodically check or poll for updates and enables parallel asynchronous processing of the message by the subscribers.

II) With SNS message Filtering

Types of message filtering:

- Topic-based filtering
- Attribute Based filtering

message filtering functionality for SNS simplifies the pub/sub messaging architecture by offloading the filtering logic from subscribers, as well as the routing logic from publishers, to SNS.

Components/Services Used:

- **Amazon SNS:** Fully managed publish/subscribe service for Application to Person (A2P) and Application to Application (A2A) messaging. This service is available in **AWS free tier.**
    - **SNS message publisher:** Sends a message to an Amazon SNS topic, a text message (SMS message) directly to a phone number, or a message to a mobile platform endpoint
    - **SNS Topic:** An Amazon SNS topic is a logical access point that acts as a communication channel.
    - **SNS endpoints:** SNS supports multiple endpoints such as AWS Lambda, Amazon SQS, HTTP/S, Mobile SMS or an email address.
    - **JSON Script:** SNS uses JSON script to publish a topic. JSON script defines which conditions an event must meet in order to be published to a topic.
- **Amazon SQS:** Fully managed message queuing for microservices, distributed systems, and serverless applications. This service is available in **AWS free tier.**
- **AWS Lambda:** Its a serverless compute service from AWS. It is used as an endpoint in SNS application service to process the notifications.
- **Amazon CloudWatch:** It's a monitoring service from AWS. It is used to monitor logs generated by AWS lambda.
- **Amazon IAM role:** This service helps to assign permission policies that determine what the identity/component can and cannot do in AWS e.g. Amazon SNS and Amazon CloudWatch services can be assigned full access permissions to Amazon lambda.

I) Without SNS message Filtering-Implementation

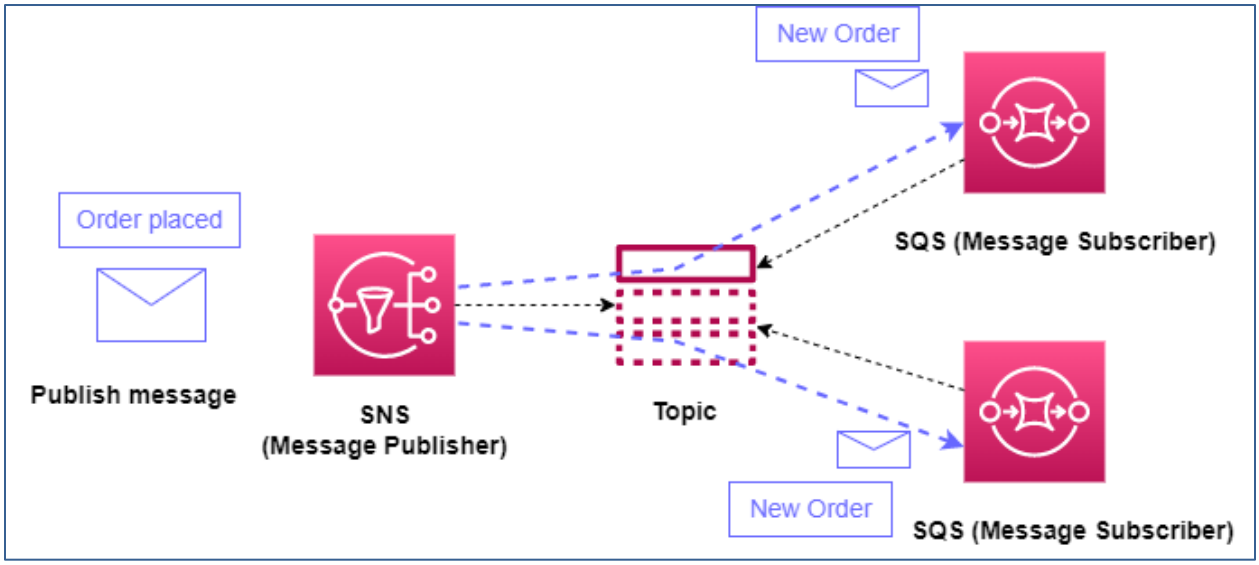


Fig 1: Fanout messaging scenario using Amazon Simple Notification Service (SNS) and Amazon Simple Queue Service (SQS) (without message filtering)

Fig 1 illustrates developing a cloud-native application that sends an Amazon SNS message to a topic whenever an order is placed on an online store. The Amazon SQS queues that are subscribed to that topic will each receive identical notifications for the new order.

Steps for Fig 1 Implementation (Fanout identical messages):

- **Browse SNS in AWS console, create a topic named New Orders**
- **Browse SQS in AWS console in separate tab, create two queues: Queues of inventory and Queues for Analytics**
- **Subscribe the queues to the SNS topic, check for confirmed status for subscriptions**
- **Check both subscriptions are listed under SNS topic**
- **Publish message to be sent to the queues by selecting SNS topic. Choose appropriate topic ARN, SQS endpoint, SQS ARN and edit message body in this step.**
- **Verify message received in each of the queues by polling messages from send receive message menu.**
- **Verify whether same message is received as sent one in each of the queues**

II) With SNS message Filtering

II a) Topic-based filtering

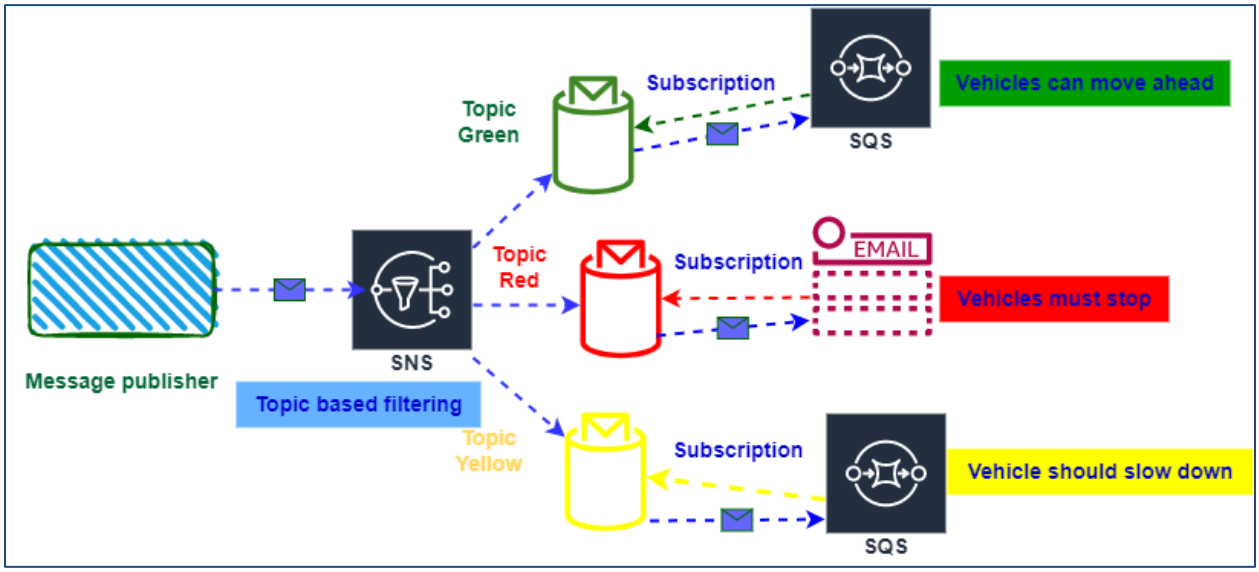


Fig 2: Messaging scenario using Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS) and Email (with topic-based message filtering for traffic light signals)

Steps for Fig 2 Implementation (Topic based filtering):

1. Browse SNS in AWS console, create three different topics named Green, Red and yellow respectively.
2. Browse SQS in AWS console in separate tab, create two queues: green signal and yellow signal
3. From SNS console click on create subscription for SNS topic named Green, choose appropriate ARN from dropdown for topic green, select endpoint as Amazon SQS protocol, select appropriate ARN for SQS and hit create subscription
4. Click on Publish message in SNS topic named Green, edit message to be sent for green signal and hit publish message
5. Check for subscription confirmed status for green topic
6. Verify message received in green signal queue by polling messages from send receive message menu.
7. Verify whether same message is received as sent.
8. Repeat steps 3 to 7 for SNS topic named Yellow with Amazon SQS endpoint
9. Create subscription with email endpoint choosing topic named red. choose appropriate ARN from dropdown for topic red, and provide email address
10. Verify subscription from received notification in email, confirm subscription from email, check subscription confirmation status in SNS topic.
11. Click on Publish message in SNS topic named Red, edit message to be sent for Red signal and hit publish message
12. Verify message received in email.

II b) Attribute Based filtering

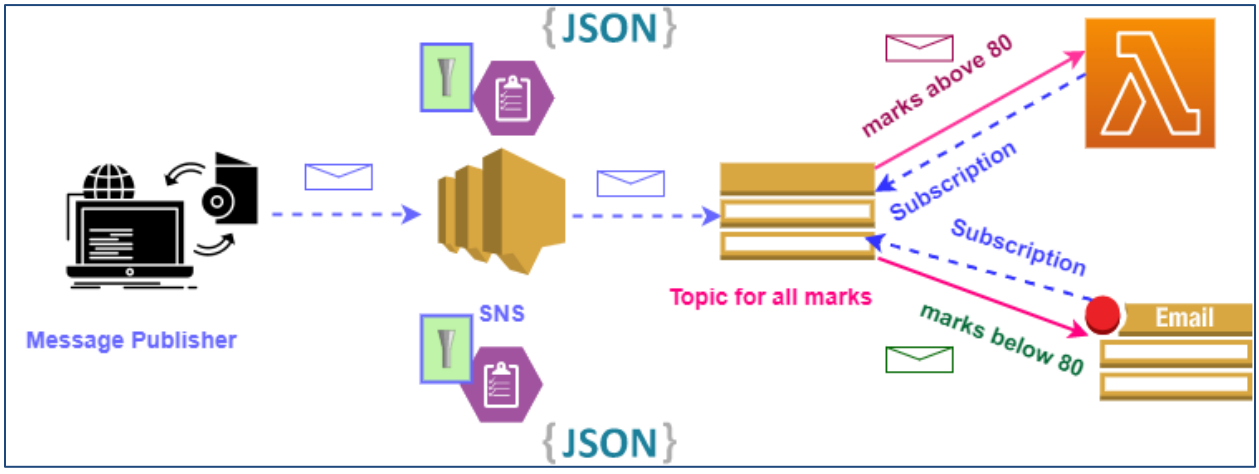


Fig 3: Messaging scenario using Amazon Simple Notification Service (SNS), AWS Lambda and Email (with attribute-based message filtering for sending student marks)

Steps for Fig 3 Implementation (Attribute based filtering):

1. Browse SNS in AWS console, create a topic named marks
2. Create subscription with email endpoint choosing topic named marks. choose appropriate ARN from dropdown for topic marks, and provide email address
3. Verify subscription from received notification in email, confirm subscription from email, check subscription confirmation status in SNS topic.
4. Click on Publish message in SNS topic named marks, edit message to be sent (marks for all) and hit publish message
5. Verify message received in email.
6. Create IAM role to assign full access rights to Amazon SNS and Amazon CloudWatch services to AWS lambda
7. Create lambda function named lambdasns
8. Create subscription from SNS topic with AWS lambda endpoint, choose appropriate topic ARN, lambda function ARN and IAM role
9. Publish message for lambda endpoint choosing SNS topic named marks, edit message to be sent (marks for all) and hit publish message
10. Edit JSON script in lambda function to print logs. Verify message received in lambda using cloudwatch logs.
11. By following above steps both email and AWS lambda endpoints will recive identical messages
12. Edit subscription filter policy in SNS, edit JSON script to add condition, provide appropriate message attributes, edit message to be sent to email endpoint (marks below 80). Create subscription. Check for subscription confirmed status.
13. Publish message for email endpoint
14. Verify message received in email.
15. Edit subscription filter policy in SNS, edit JSON script to add condition, provide appropriate message attributes, edit message to be sent to AWS lambda endpoint (marks above 80). Create subscription. Check for subscription confirmed status.
16. Publish message for AWS lambda endpoint
17. Verify message received in AWS lambda using CloudWatch logs
18. Steps 12 to 17 provide attribute-based filtering for SNS messages. Different messages are received in the two endpoints based on the conditions applied in the JSON scripts

---

B. Implementation Screenshot with brief description for every screenshot. Code/ Script files (if applicable) Screen shots must have your AWS / Azure account ID clearly visible along with your laptop's date and time

I) Without SNS message Filtering-Implementation

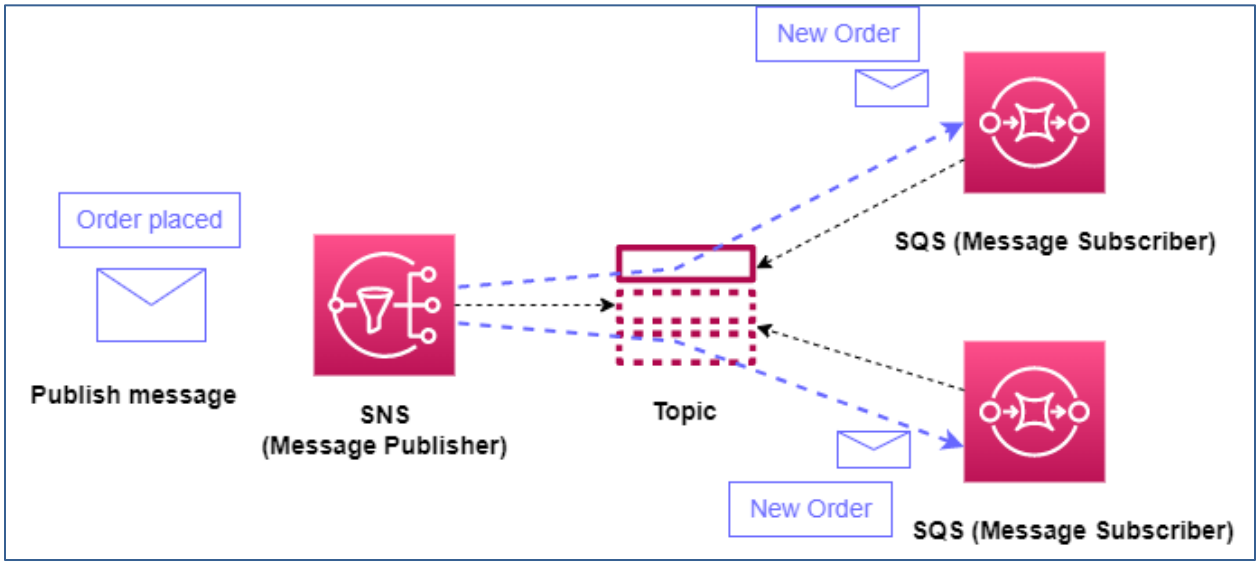
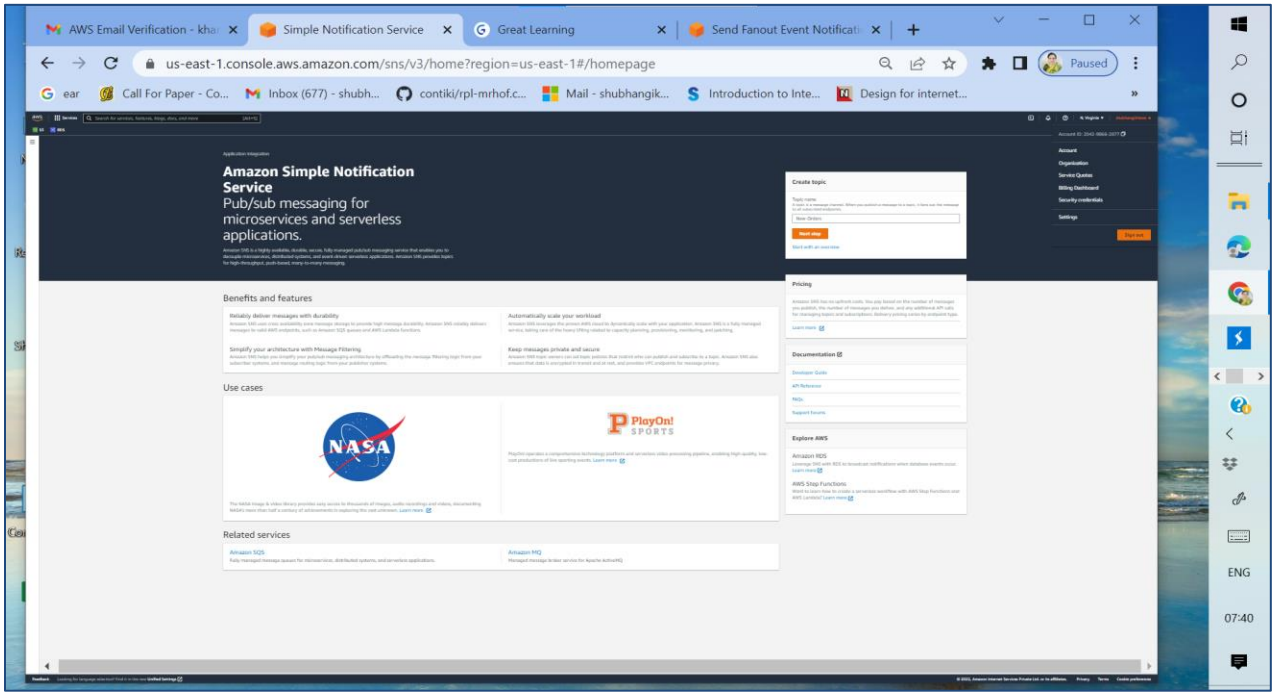
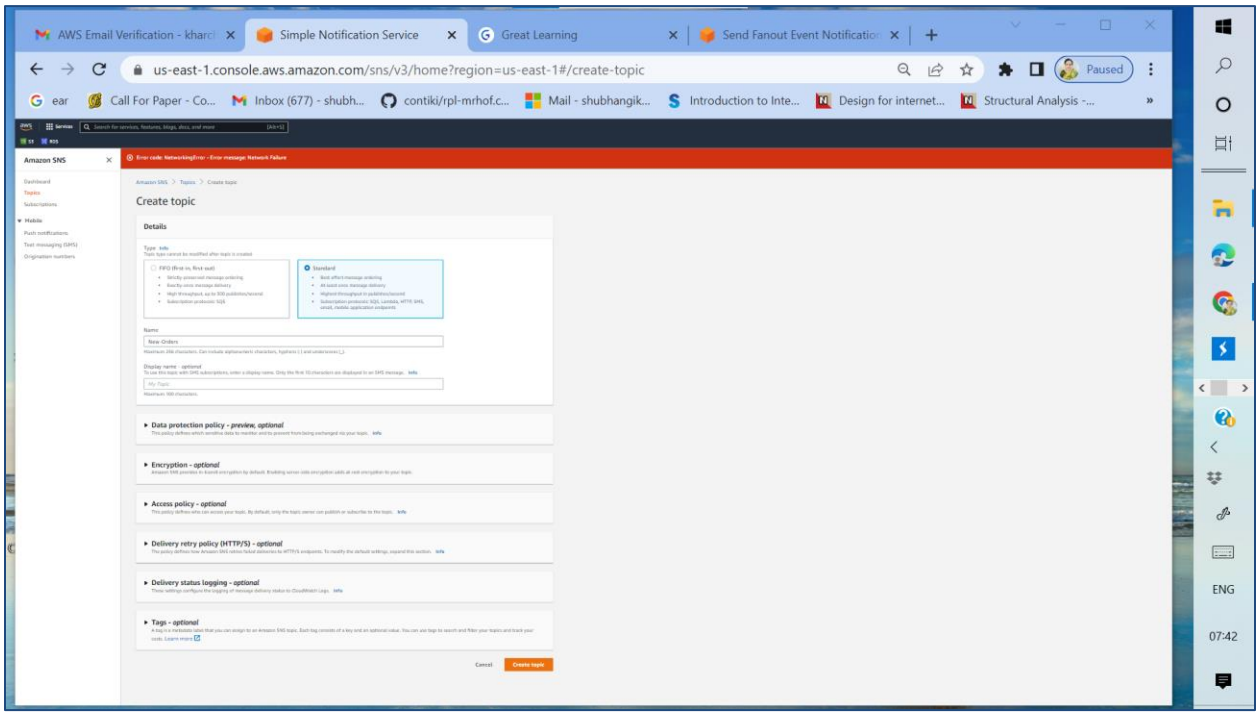


Fig 1: Fanout messaging scenario using Amazon Simple Notification Service (SNS) and Amazon Simple Queue Service (SQS) (without message filtering)

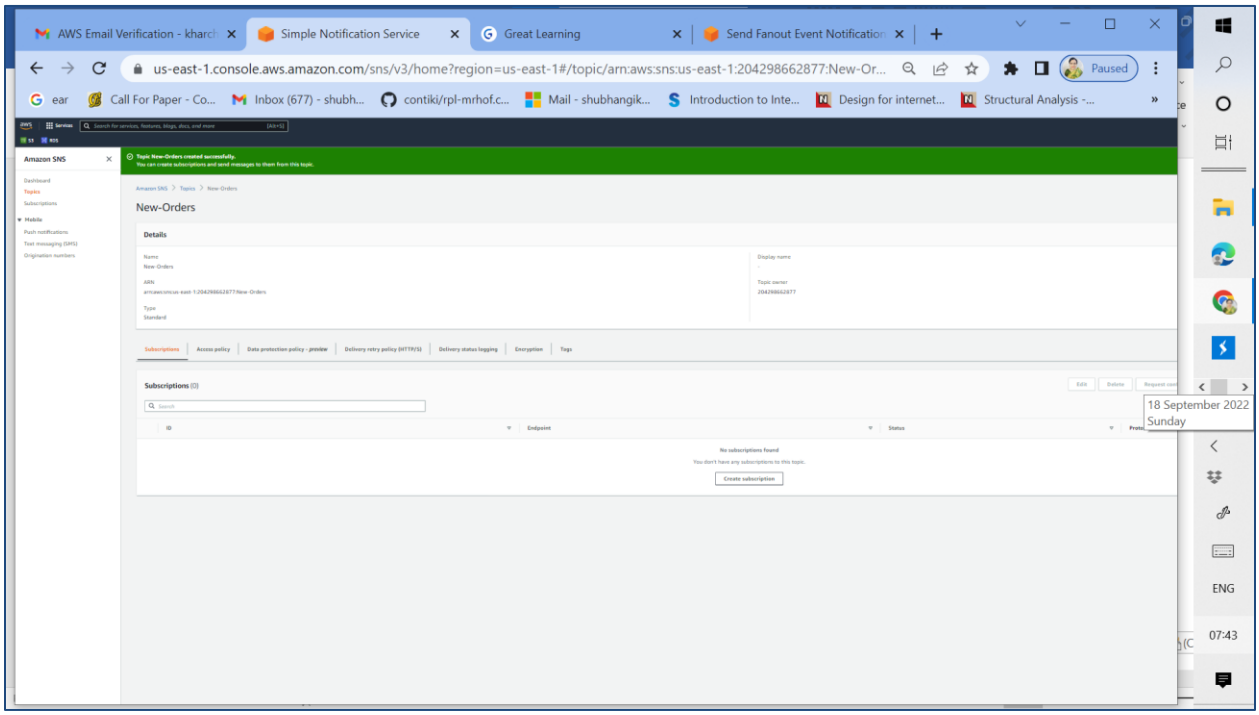
Fig 1: Implementation Screenshots:



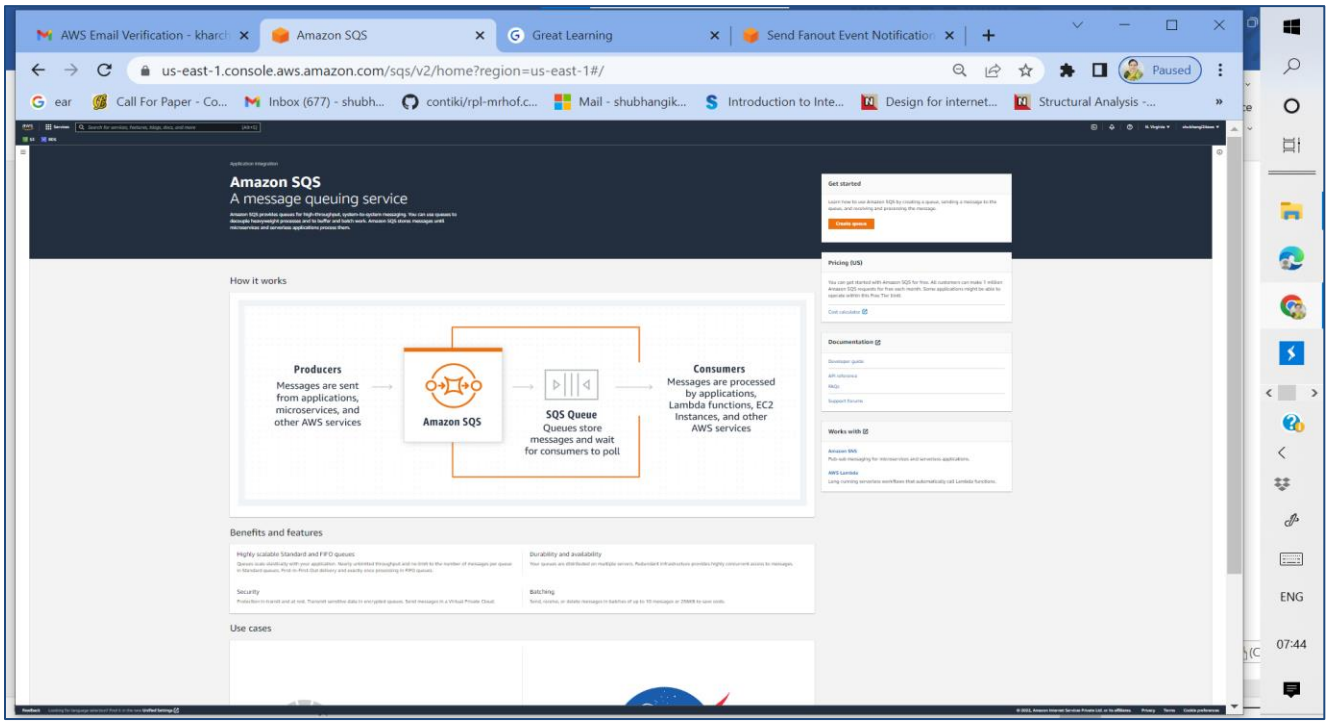
1a: Browse SNS from AWS console



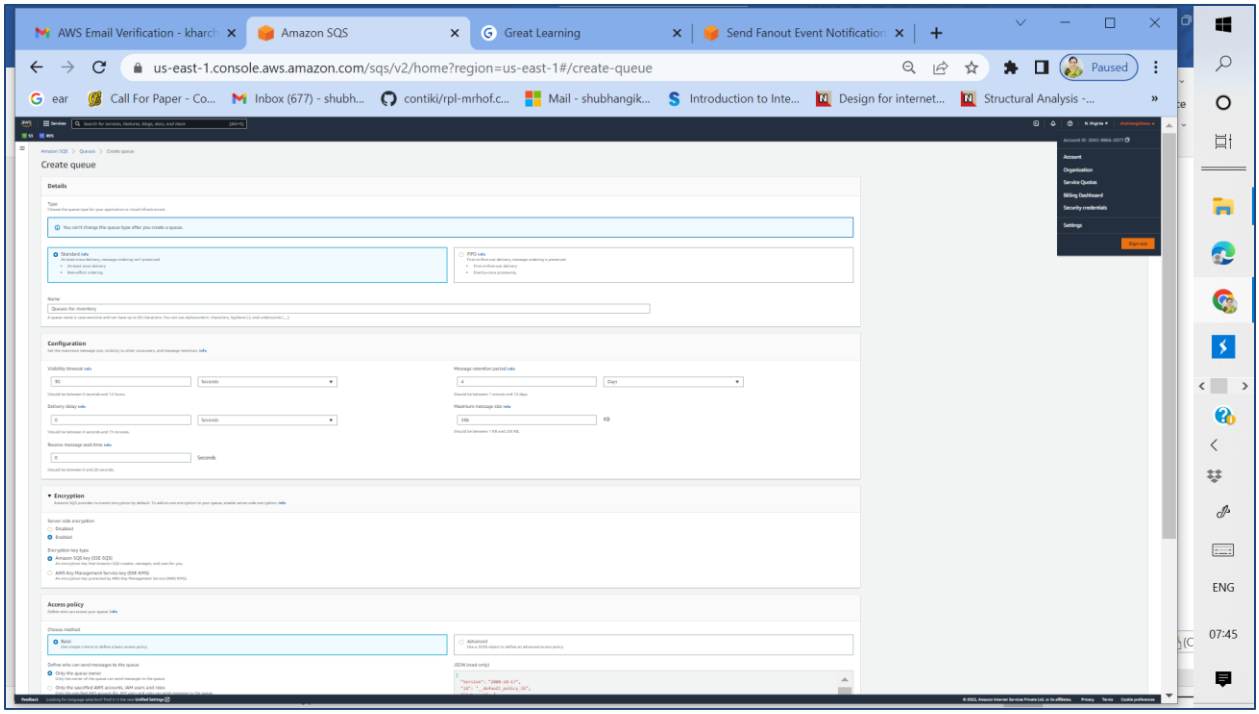
1b: Create SNS topic named New-Orders (standard topic)



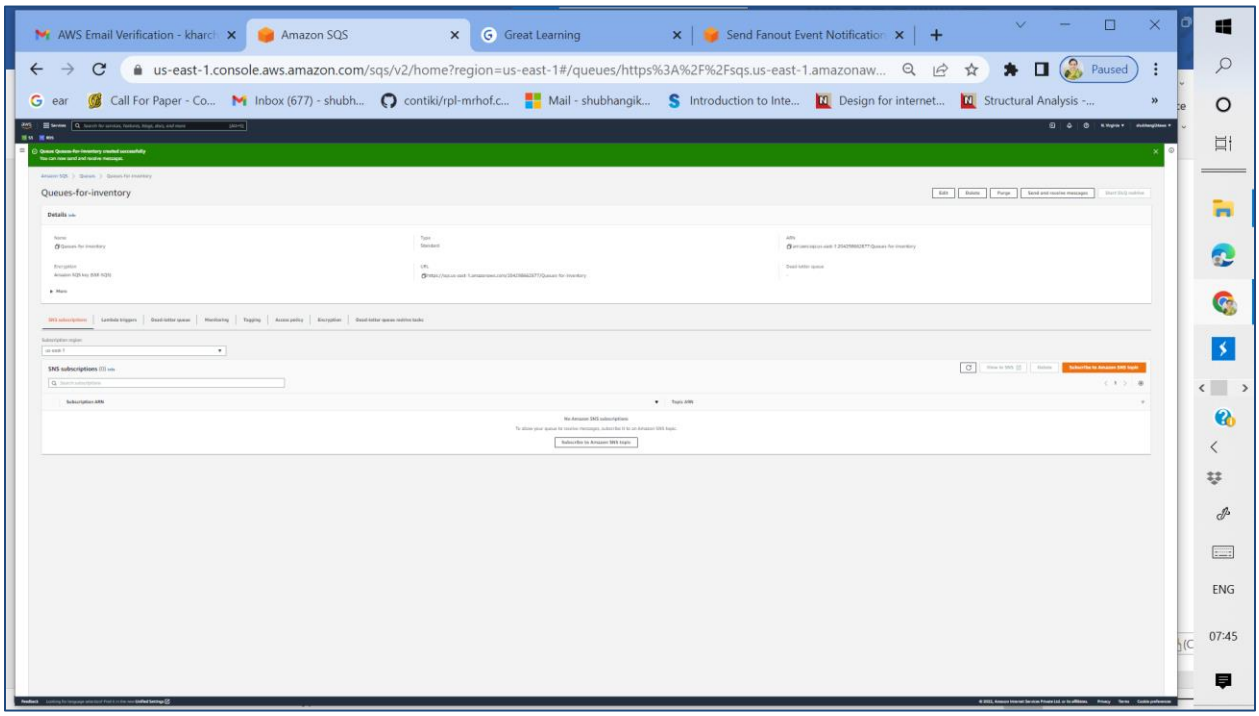
1c: SNS topic named New-Orders created successfully



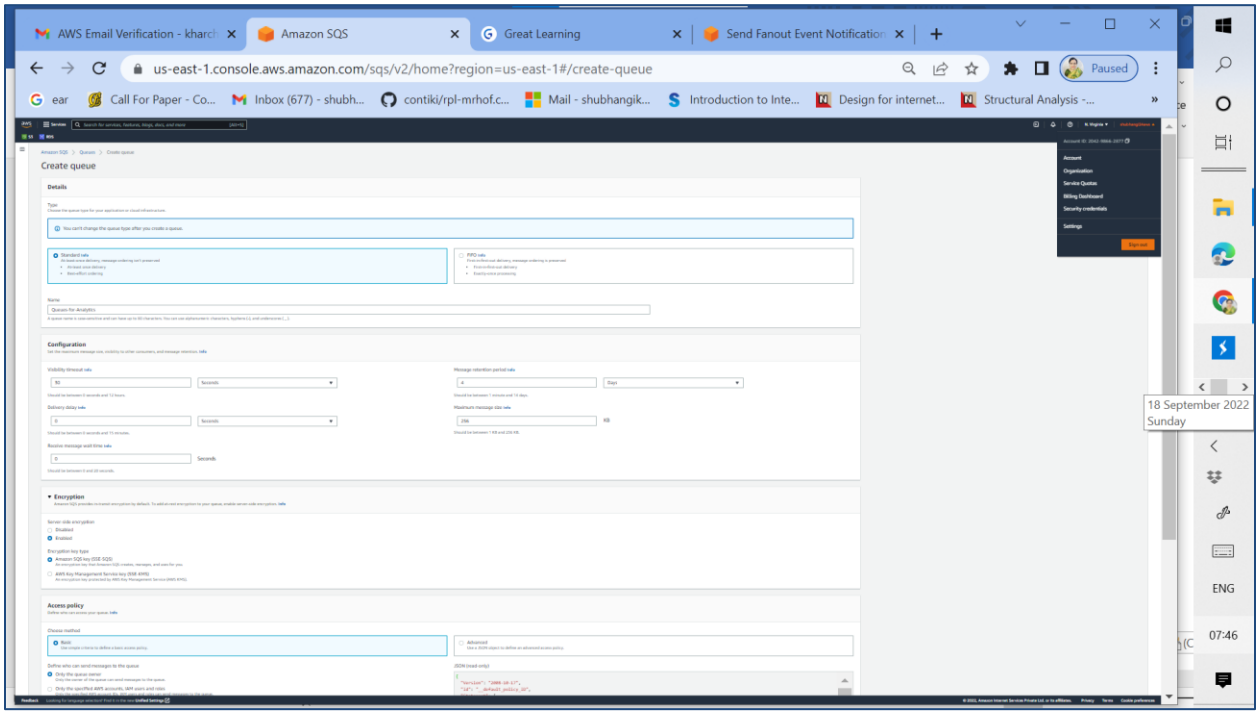
1d: Browse Amazon SQS from AWS console



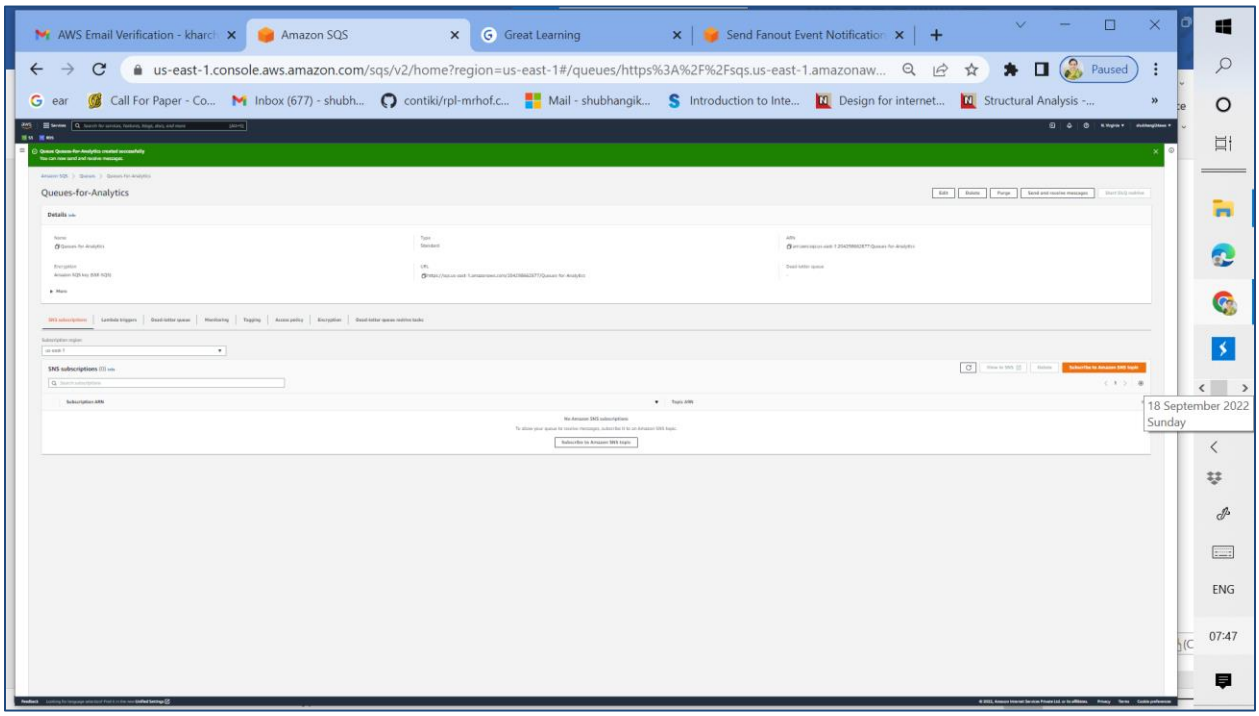
1e: Create standard SQS Queue named Queues for inventory



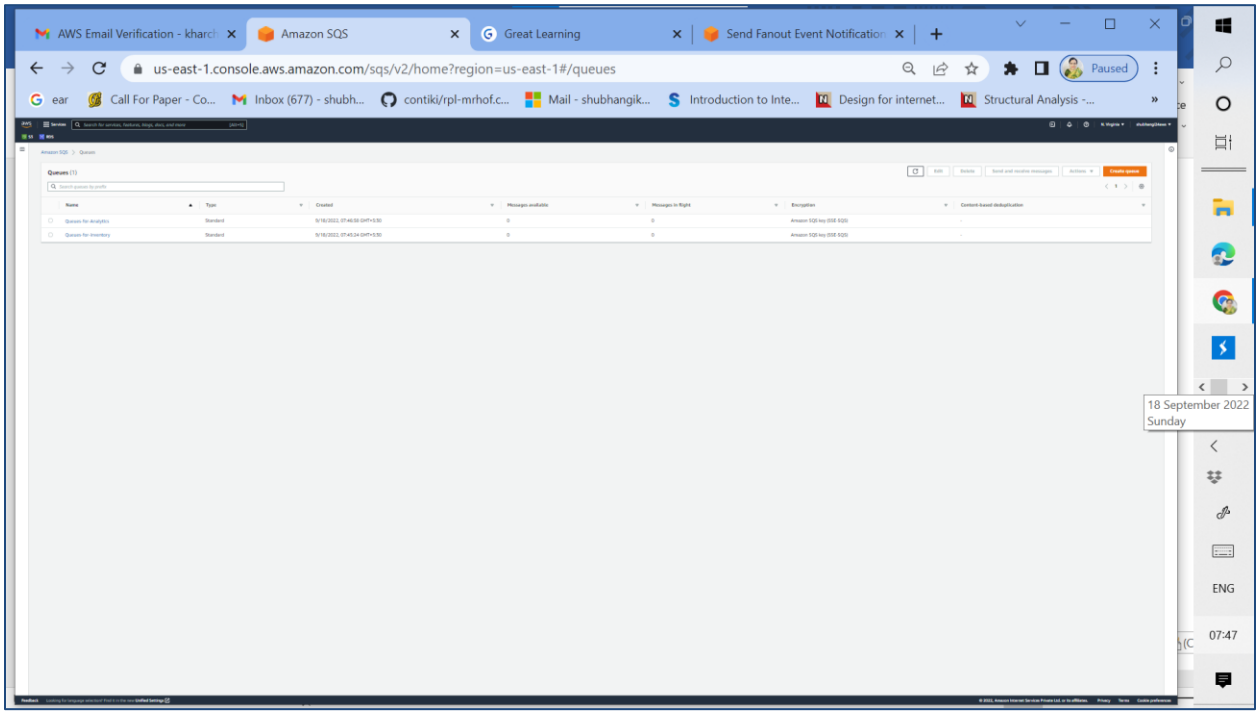
1f: standard SQS Queue named Queues for inventory successfully created



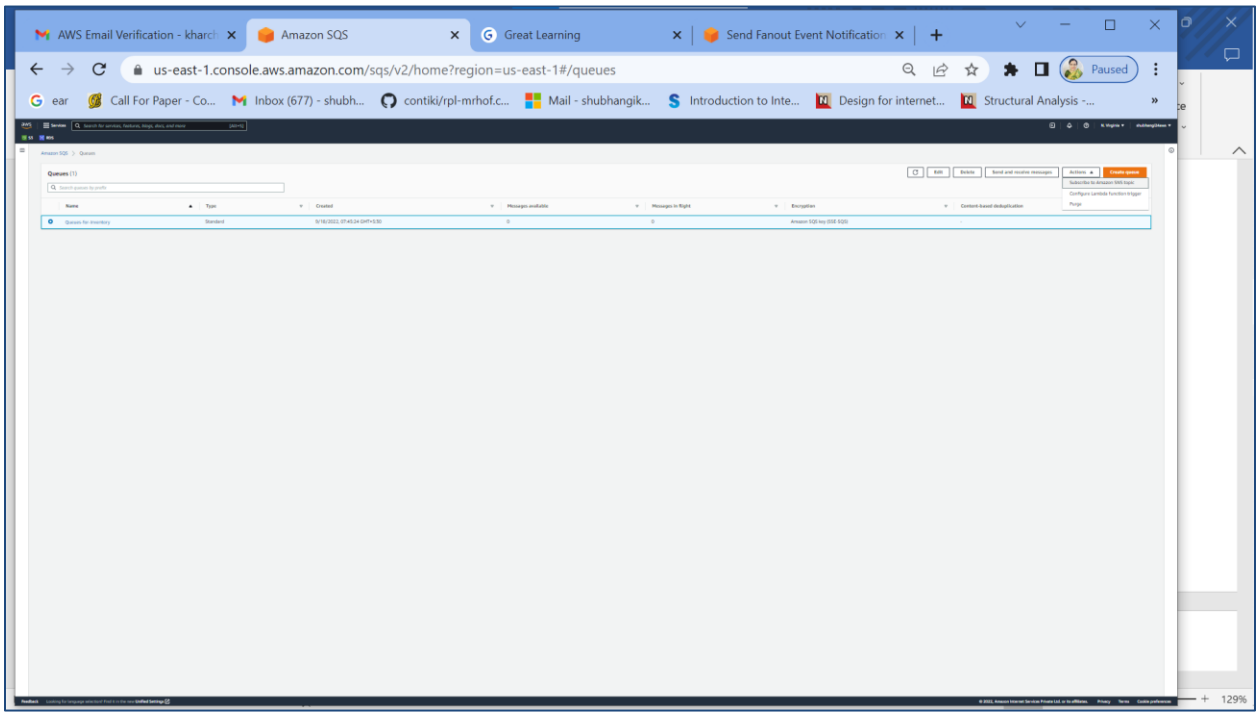
1g: Create standard SQS Queue named Queues for Analytics



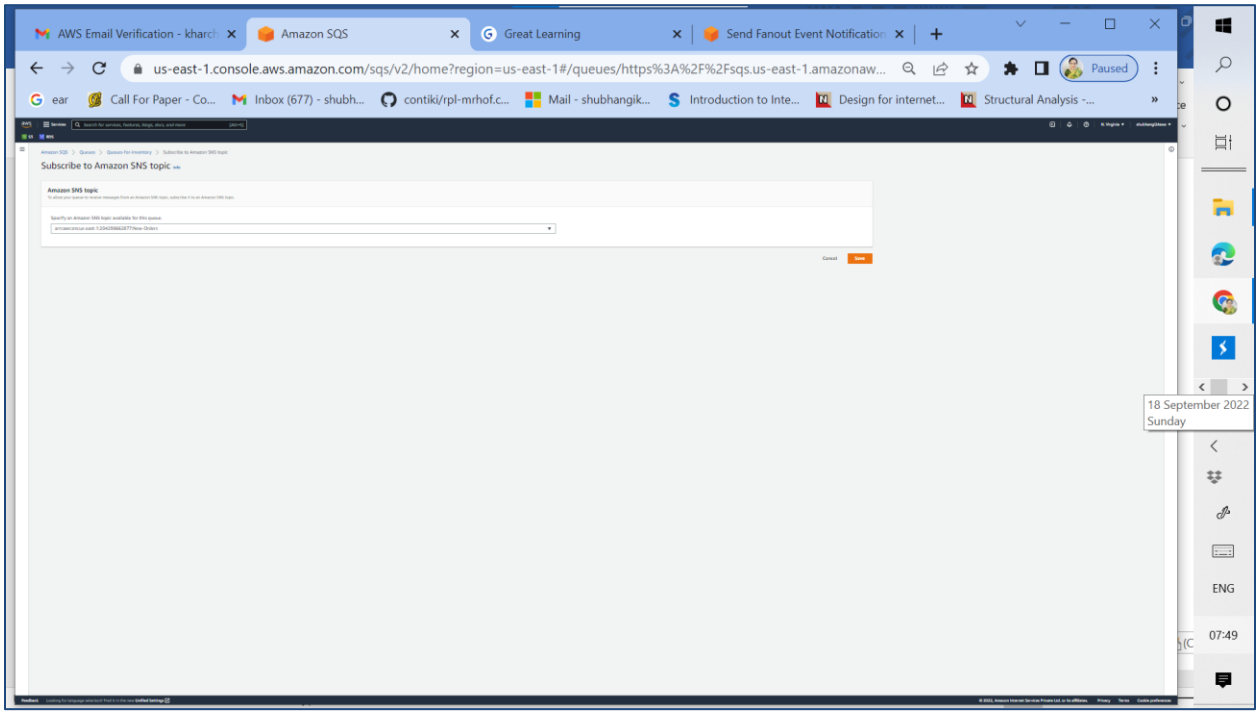
1h: Standard SQS Queue named Queues for Analytics successfully created



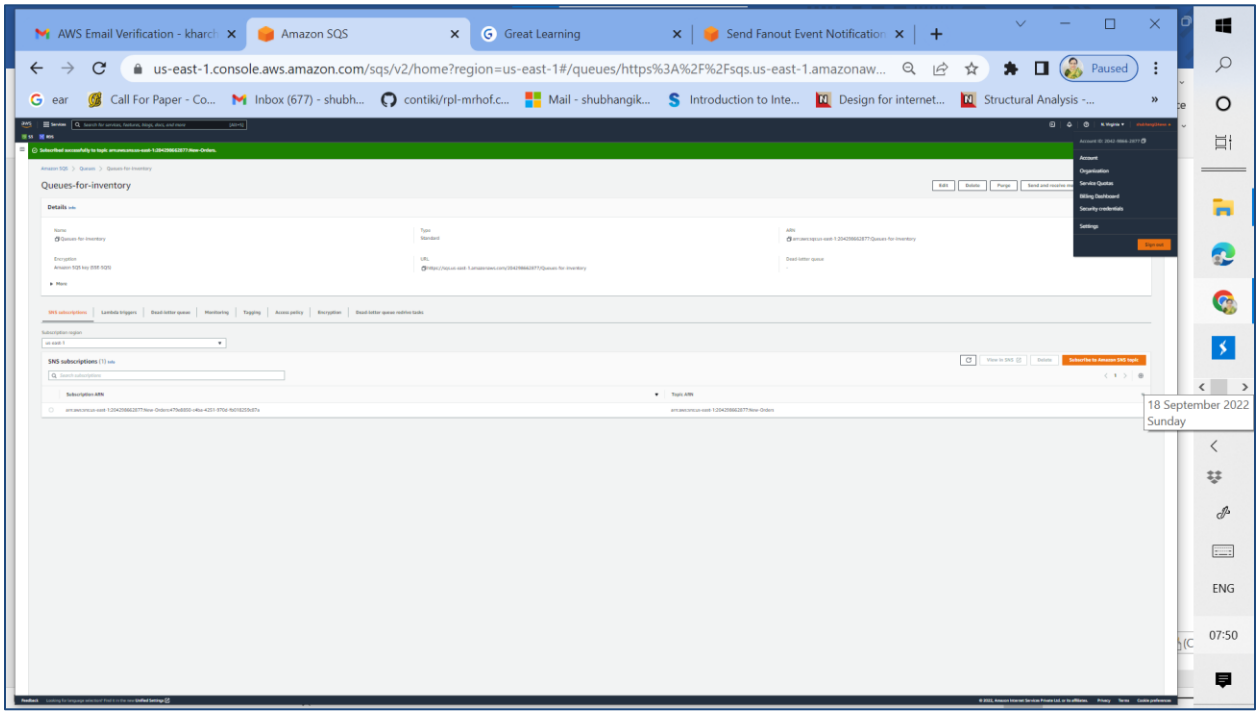
1i: Both Standard Queues listed in Amazon SQS



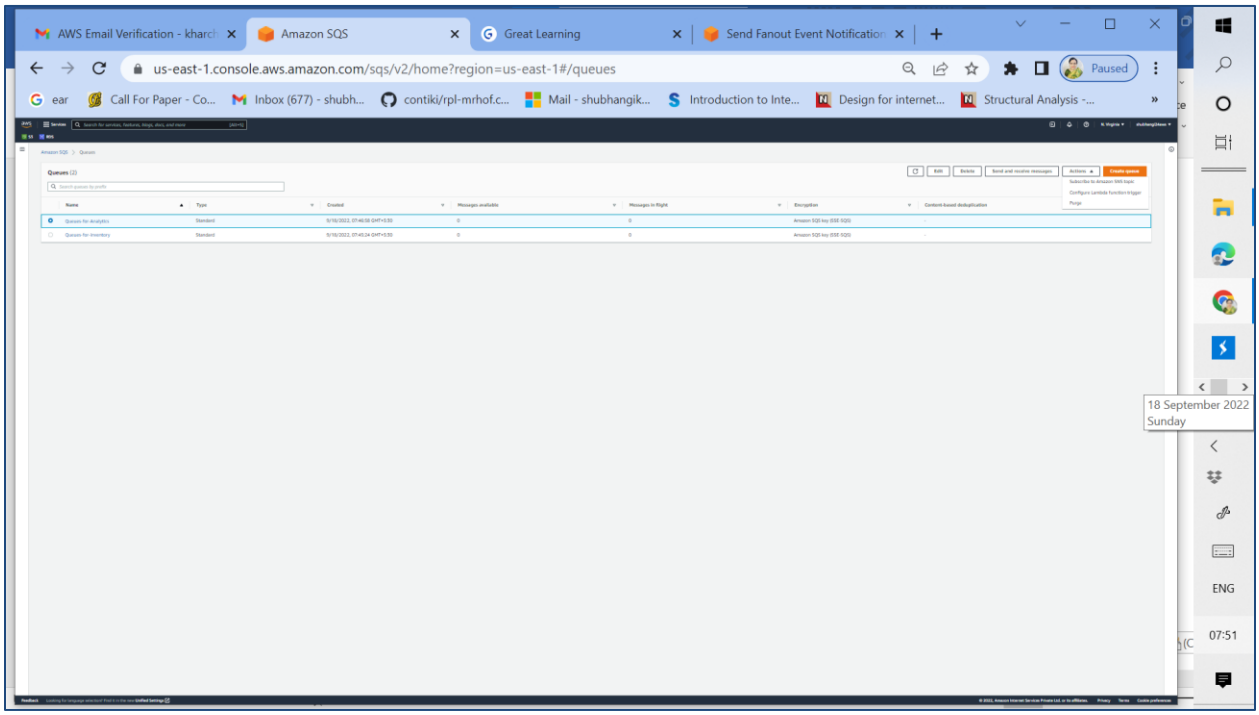
1j: Subscribe Queues for inventory to Amazon SNS topic New-Orders



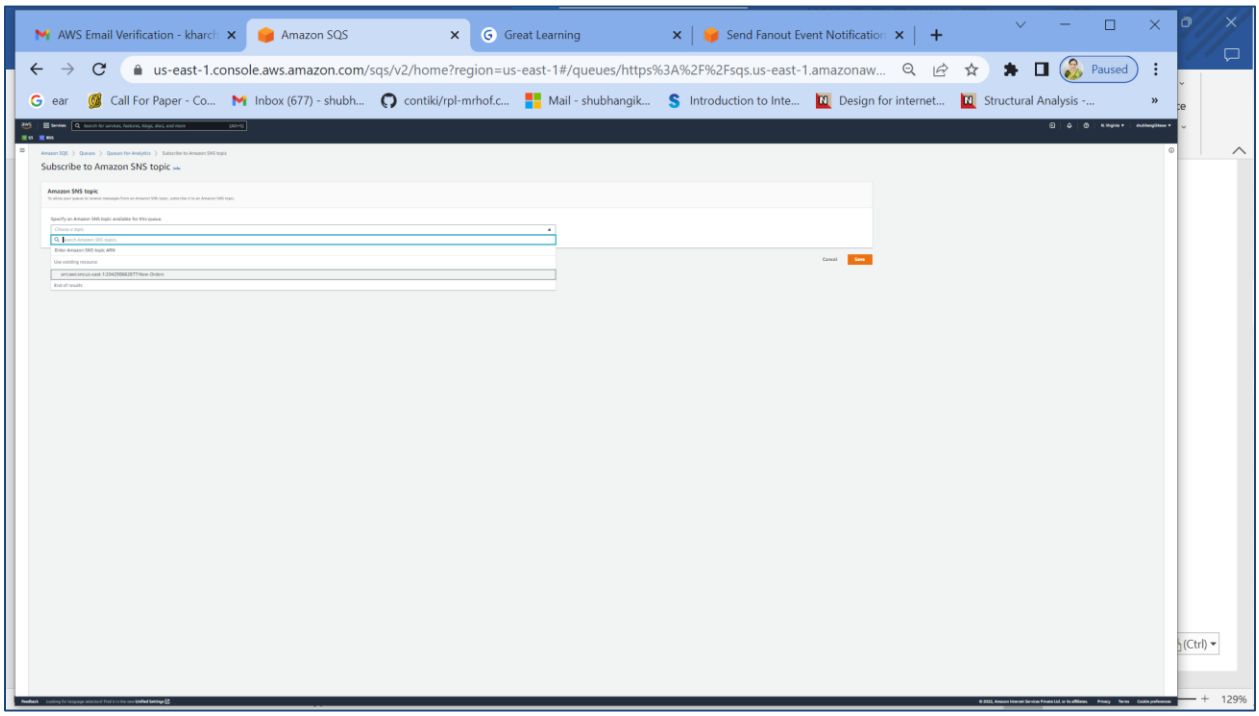
1k: Choose SNS topic ARN while subscribing Queues for inventory to the topic New-Orders and save



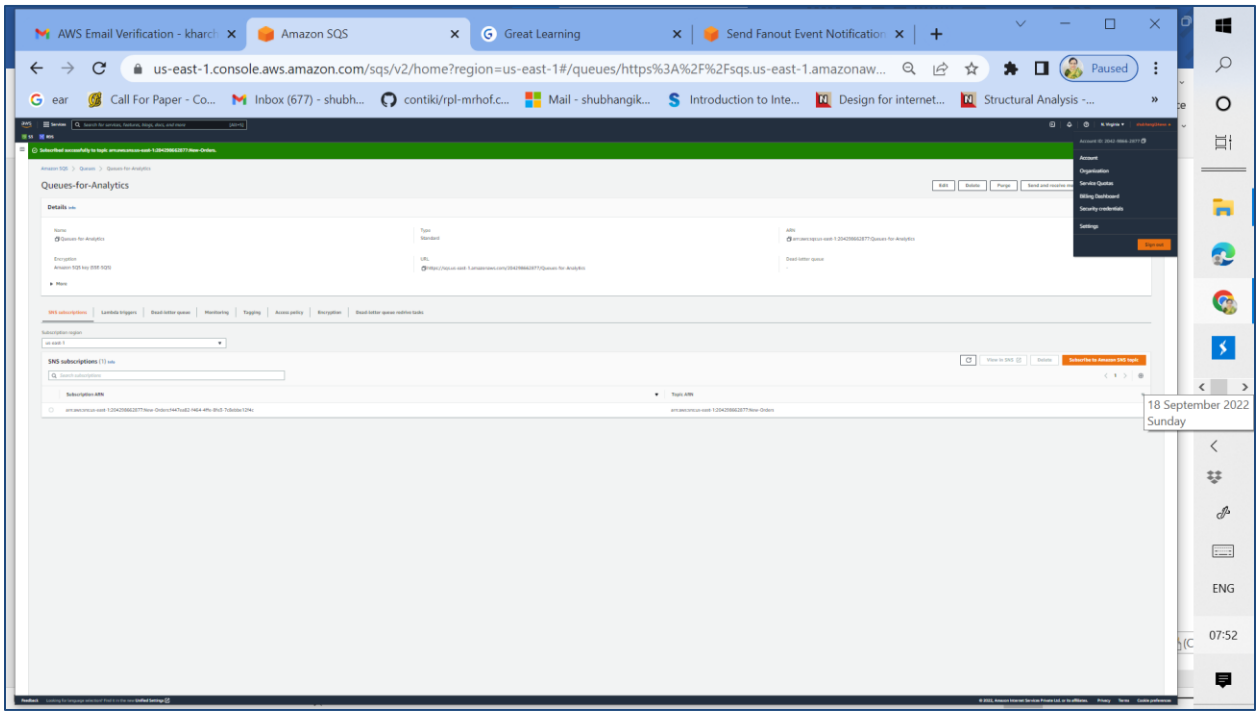
1l: Subscribing Queues for inventory to the topic New-Orders is successful



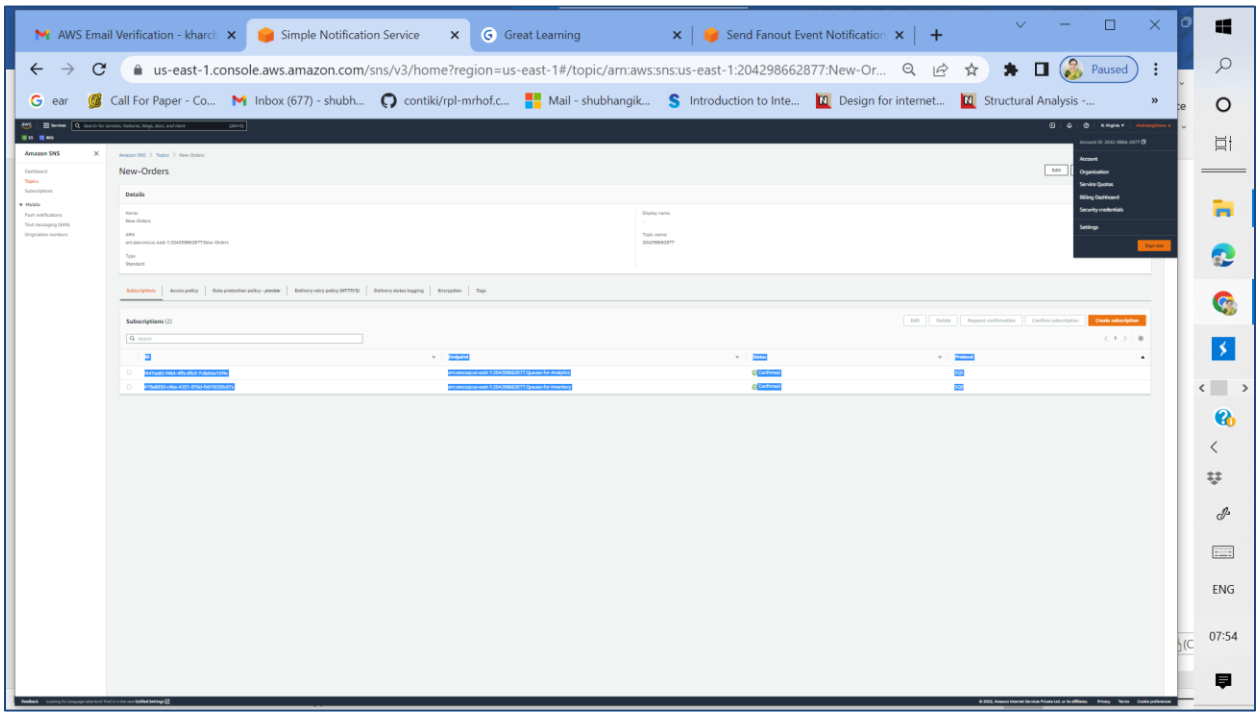
1m: Subscribe Queues for Analytics to the Amazon SNS topic New-Orders



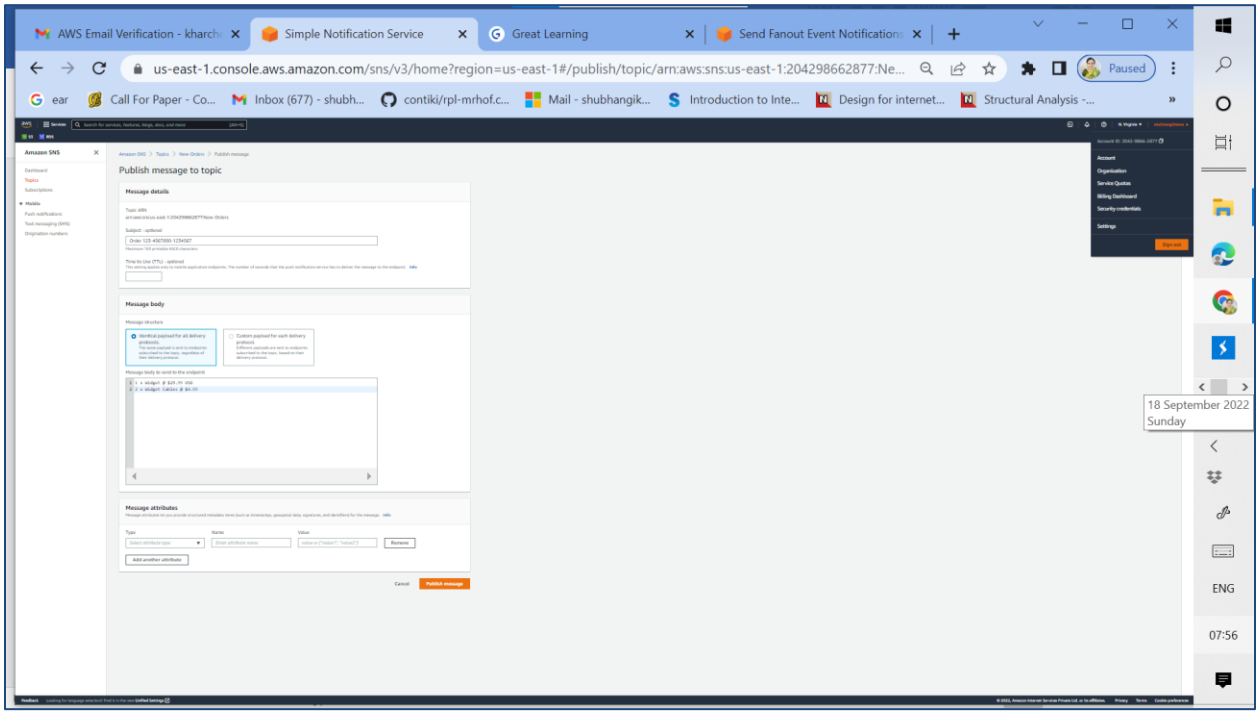
1n: Choose SNS topic ARN while subscribing Queues for Analytics to the topic New-Orders and save



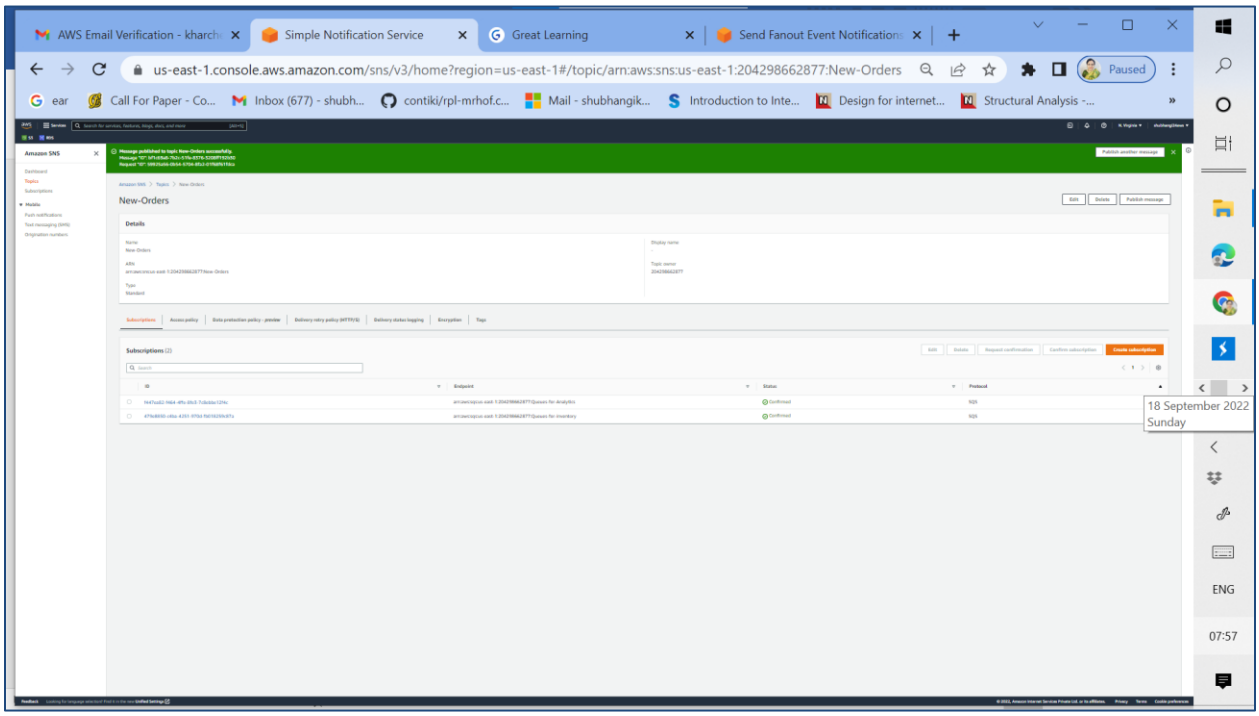
1o: Queues for Analytics successfully subscribed to the topic New-Orders



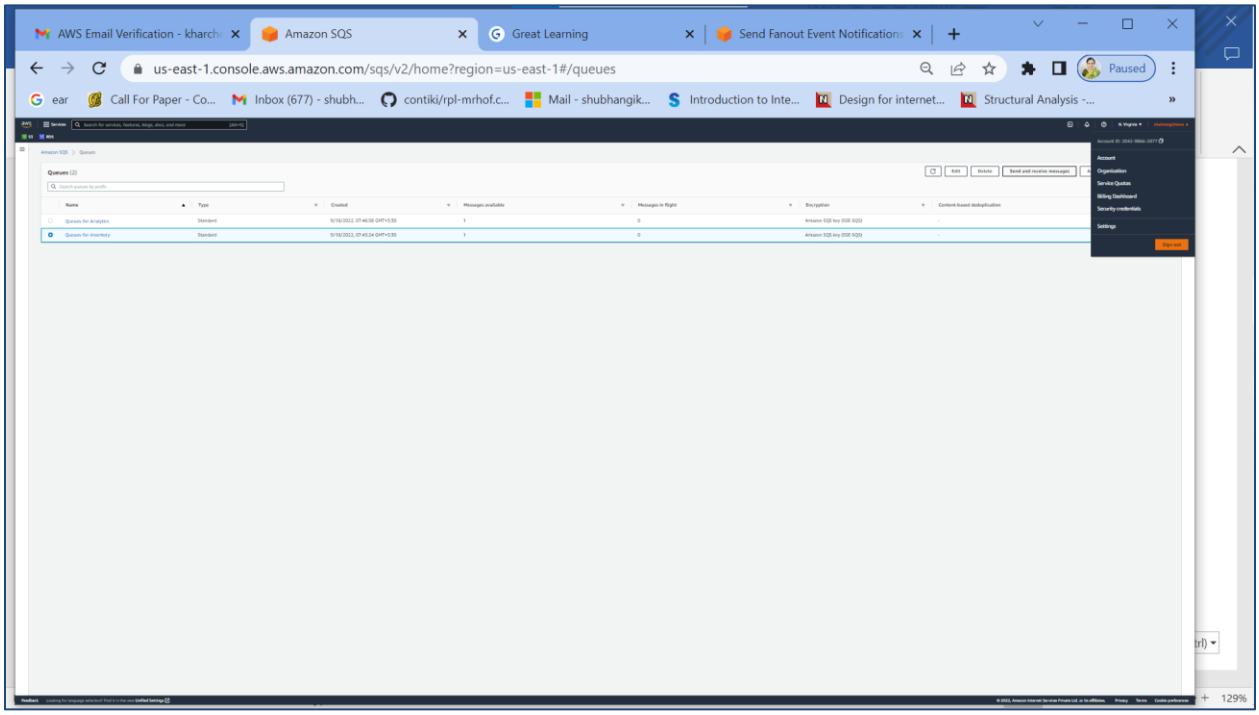
1p: Both SQS Queues subscription is now listed under the SNS topic New-Orders



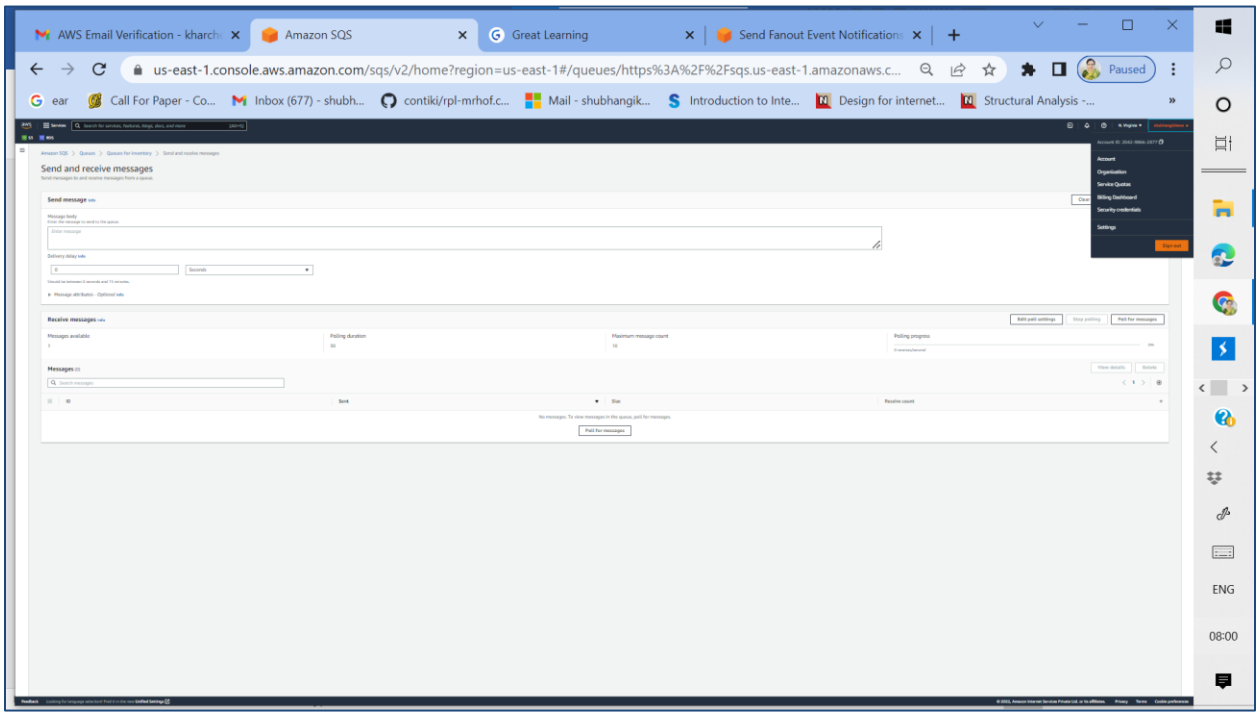
1q: Click Publish message from SNS topic New-Orders using identical payload for all delivery. Edit the message body for the particular message which needs to be sent to the SQS queues and hit publish message



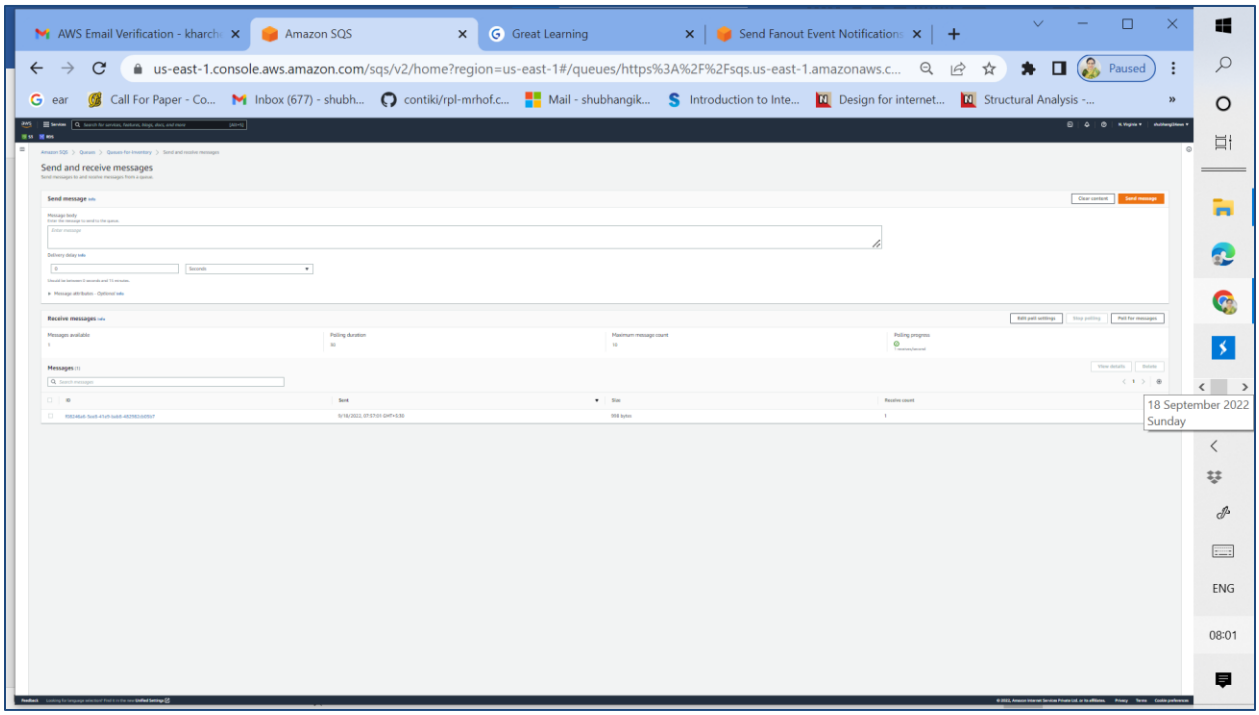
1r: Published message from topic New-orders for both the Queues is successful



1s: Verify whether Published message is successful for Queues for inventory or not using send receive messages

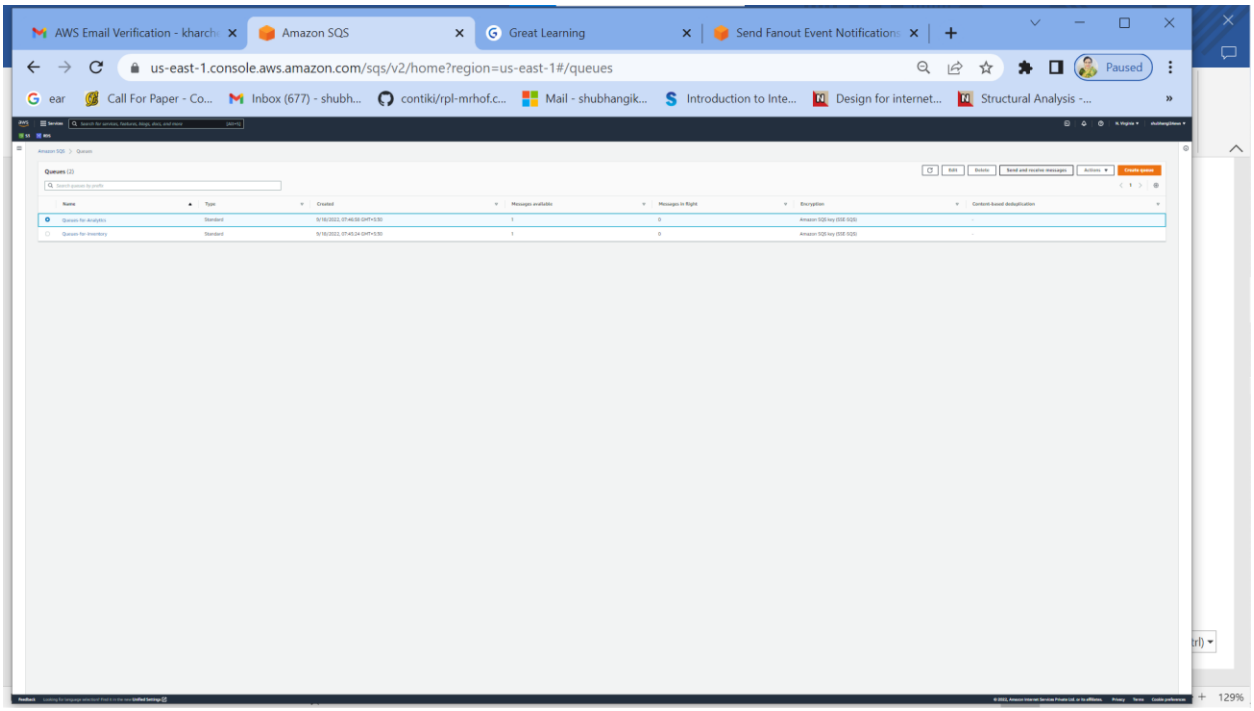


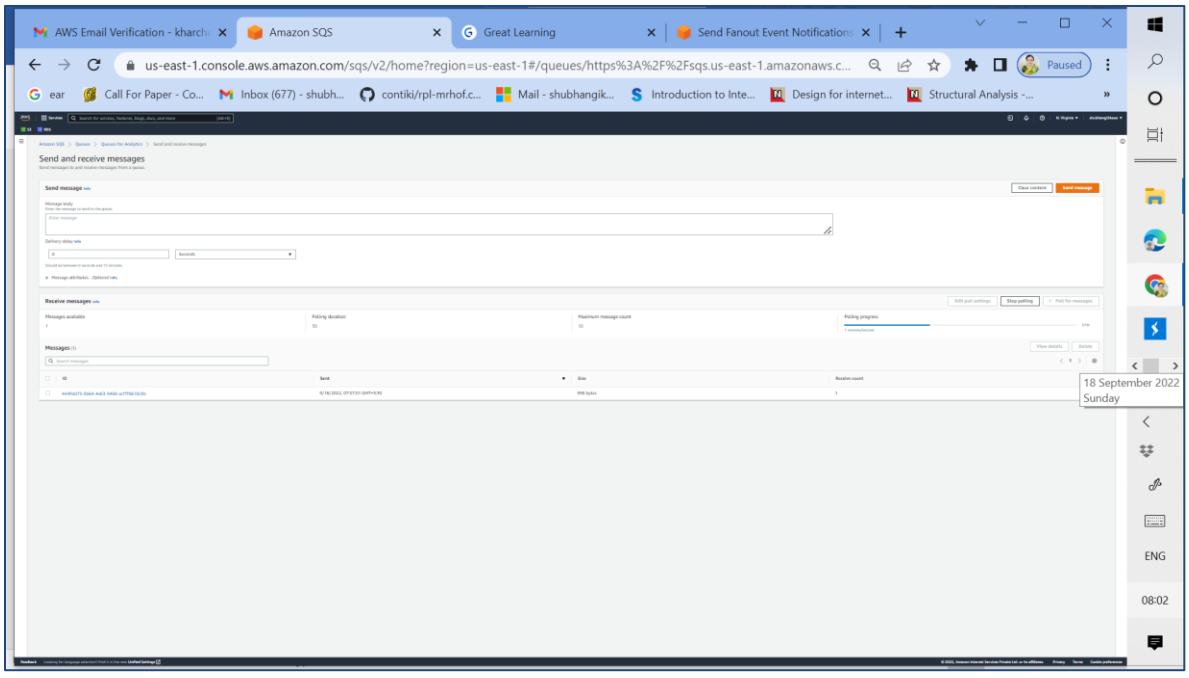
1t: Start polling messages for Queues for inventory. At the start zero messages are received



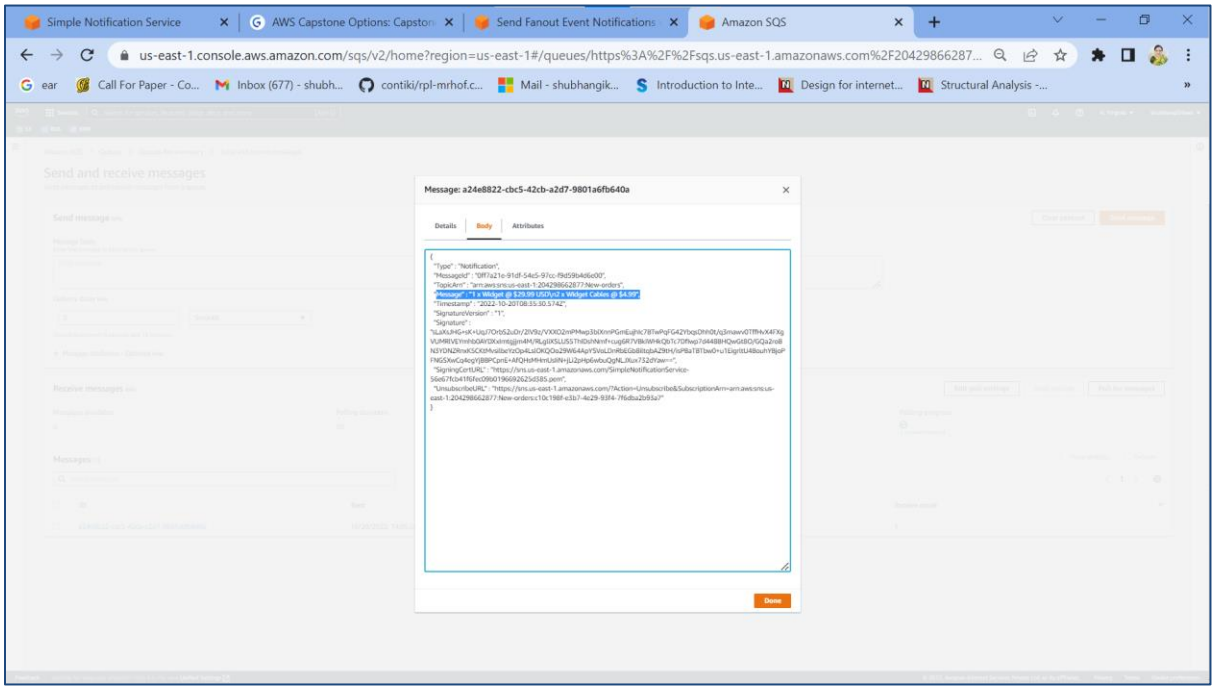
1u: As polling progresses for Queues for inventory, one message is received

Repeat 1s to 1u for Queues for Analytics:

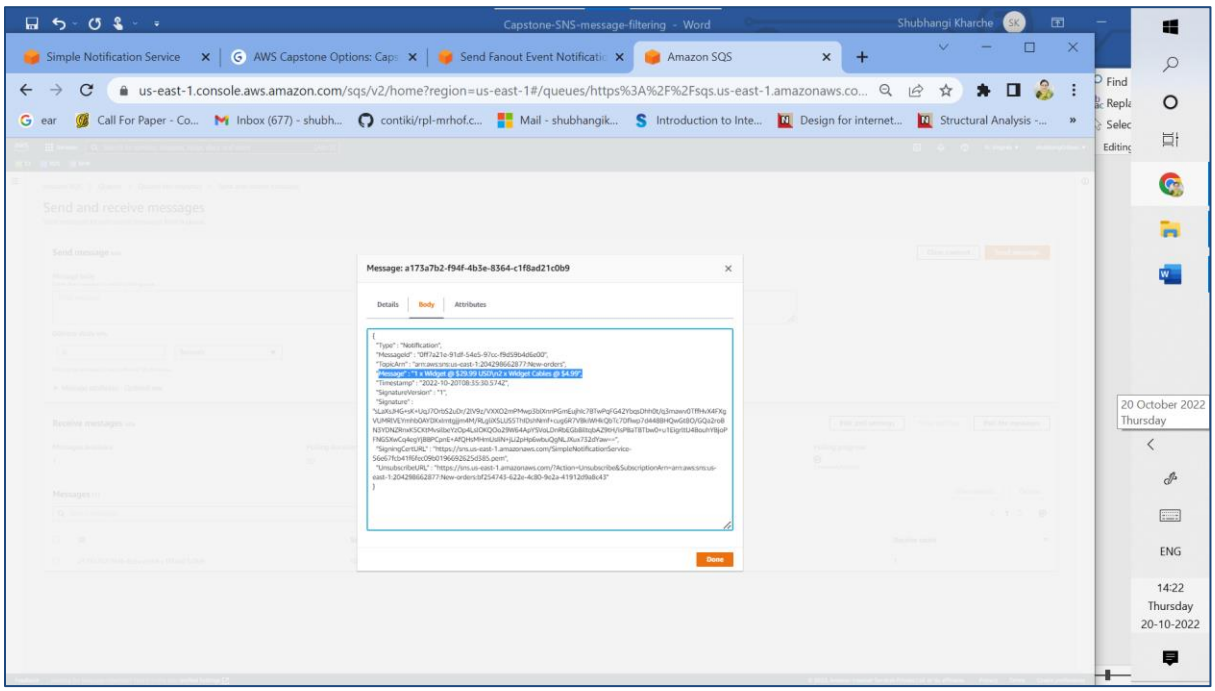




1v: As polling progresses for Queues for Analytics, one message is received



1w: Verification of actual message received in Queues for inventory (it is same as sent message)



1x: Verification of actual message received in Queues for Analytics (it is same as sent message)

Next step is to successfully delete all aws resources.

II a) Topic-based filtering

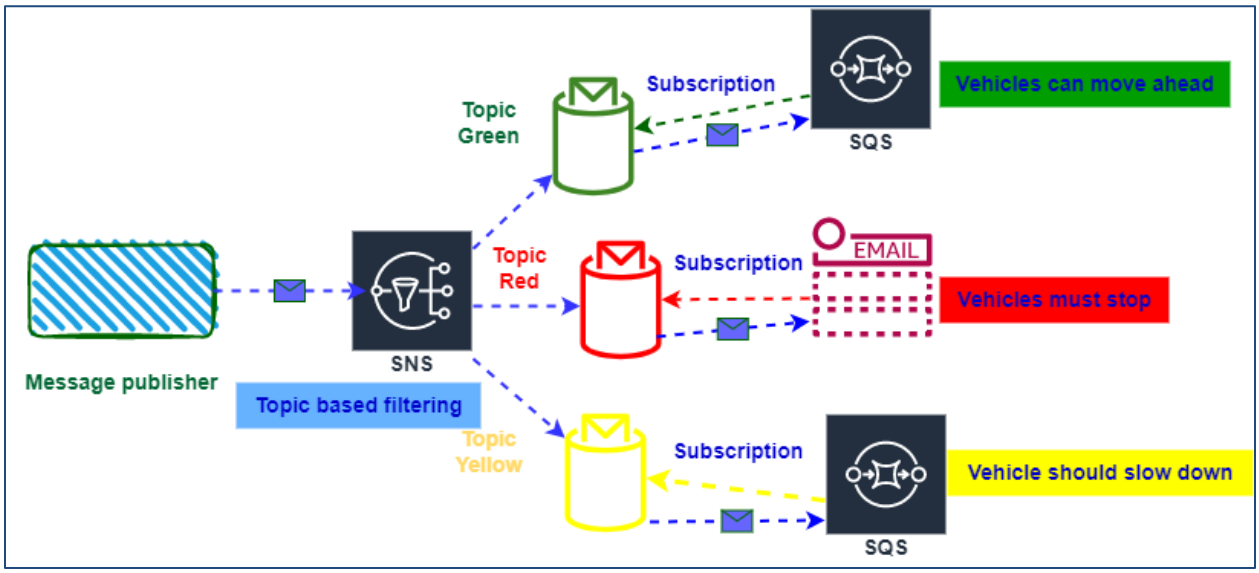
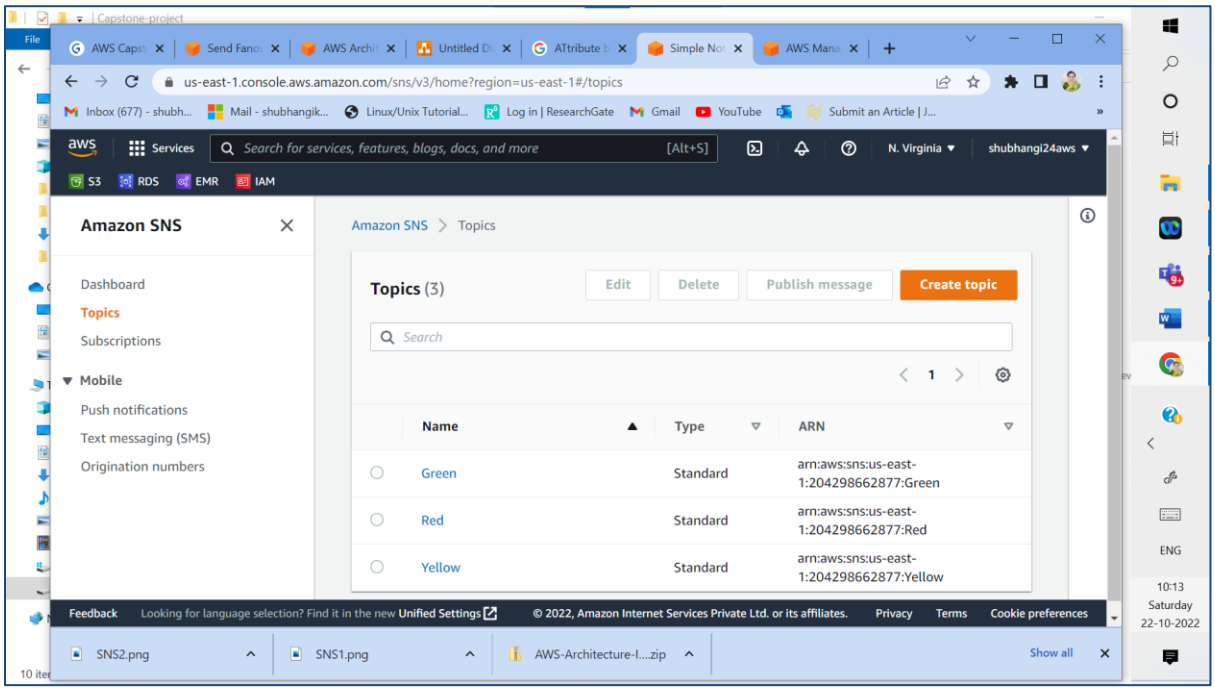
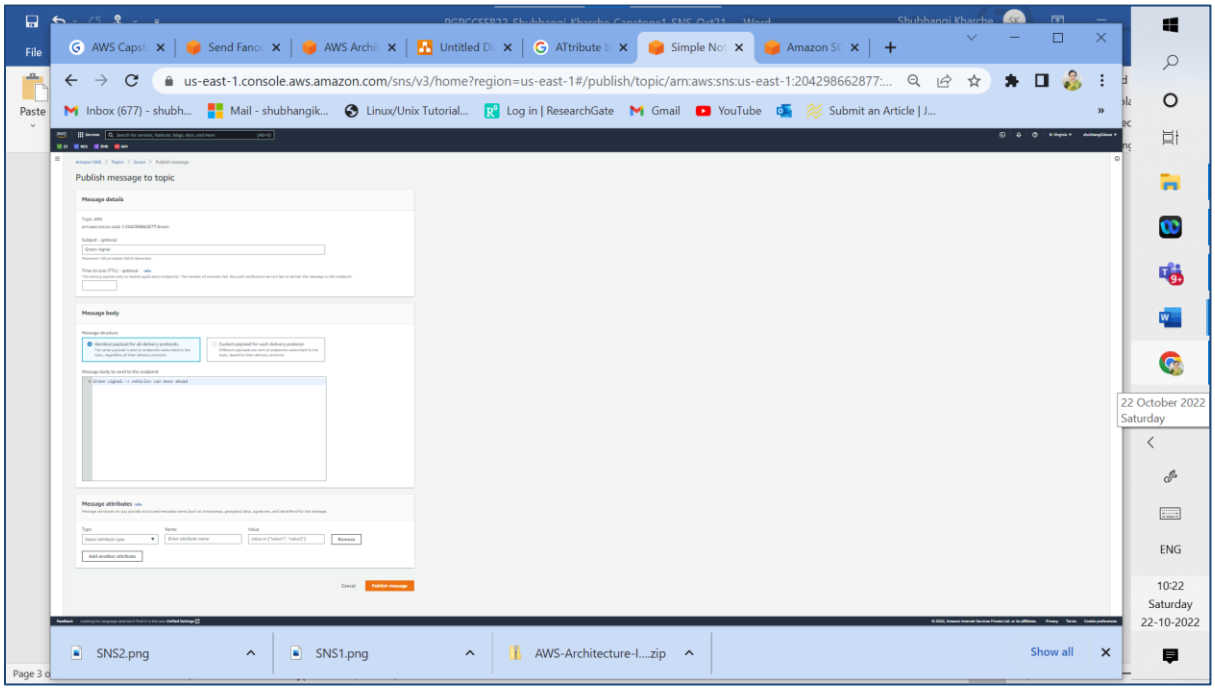


Fig 2: Messaging scenario using Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS) and Email (with topic based message filtering for traffic light signals)

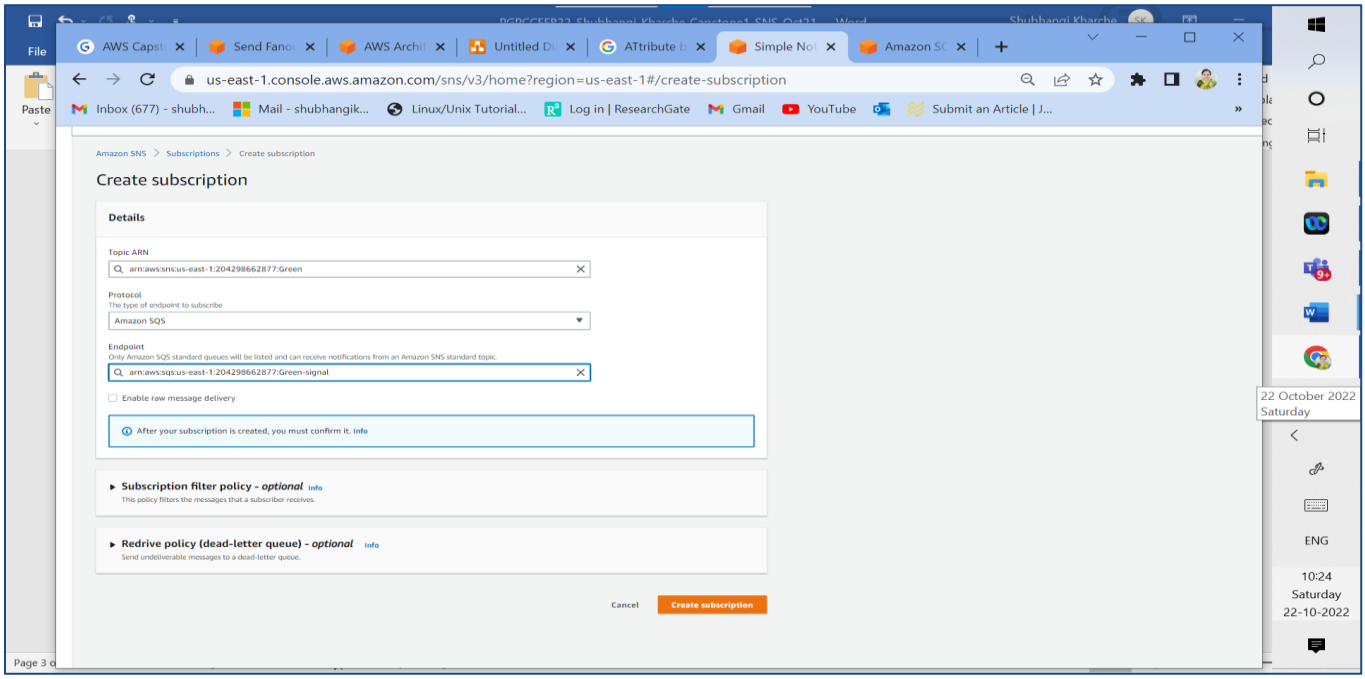
Implementation Screenshots for Fig 2:



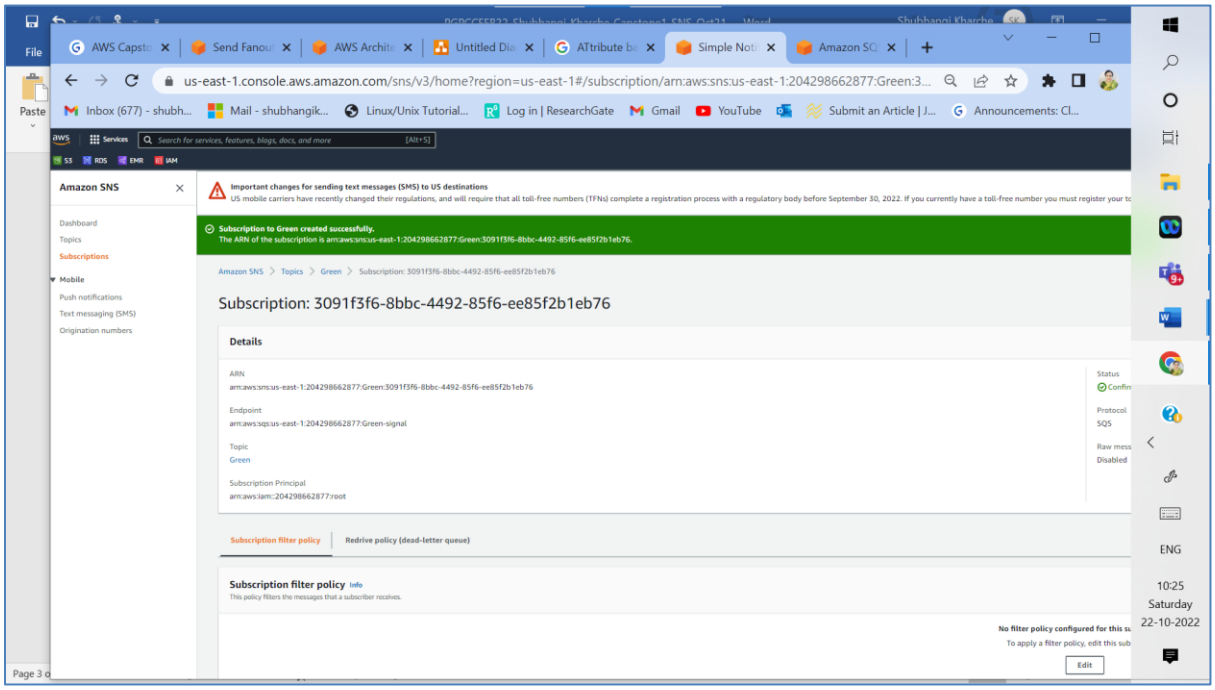
2a: Create three different SNS topics (Green, Red and Yellow)



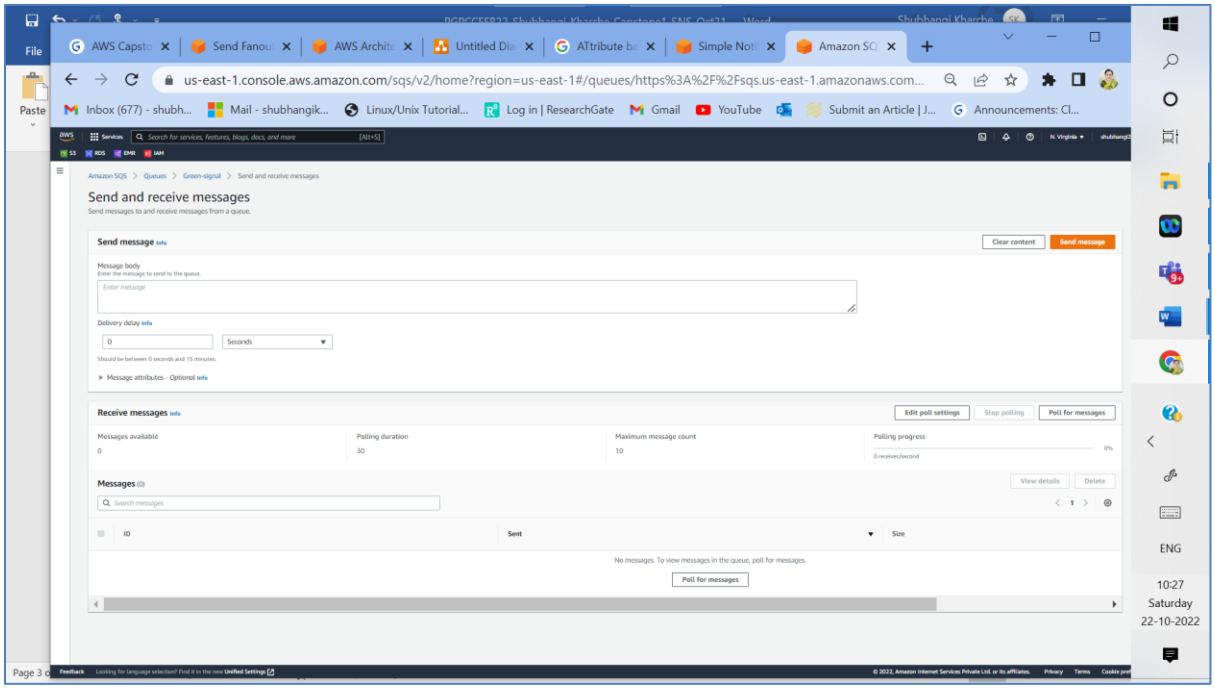
2b: Click on Publish message in SNS topic named Green, edit message to be sent for green signal and hit publish message



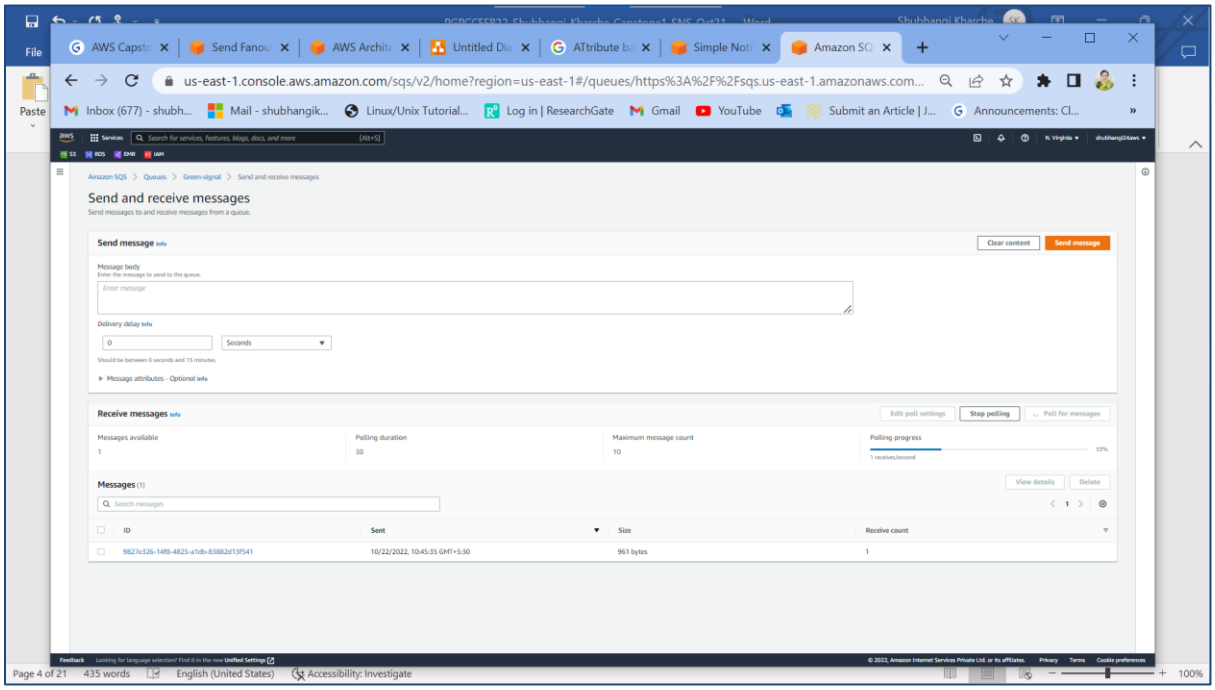
2c: Create SQS. Click on create subscription for SNS topic named Green, choose appropriate ARN from dropdown for topic green, select endpoint as Amazon SQS protocol, select appropriate ARN for SQS and hit create subscription



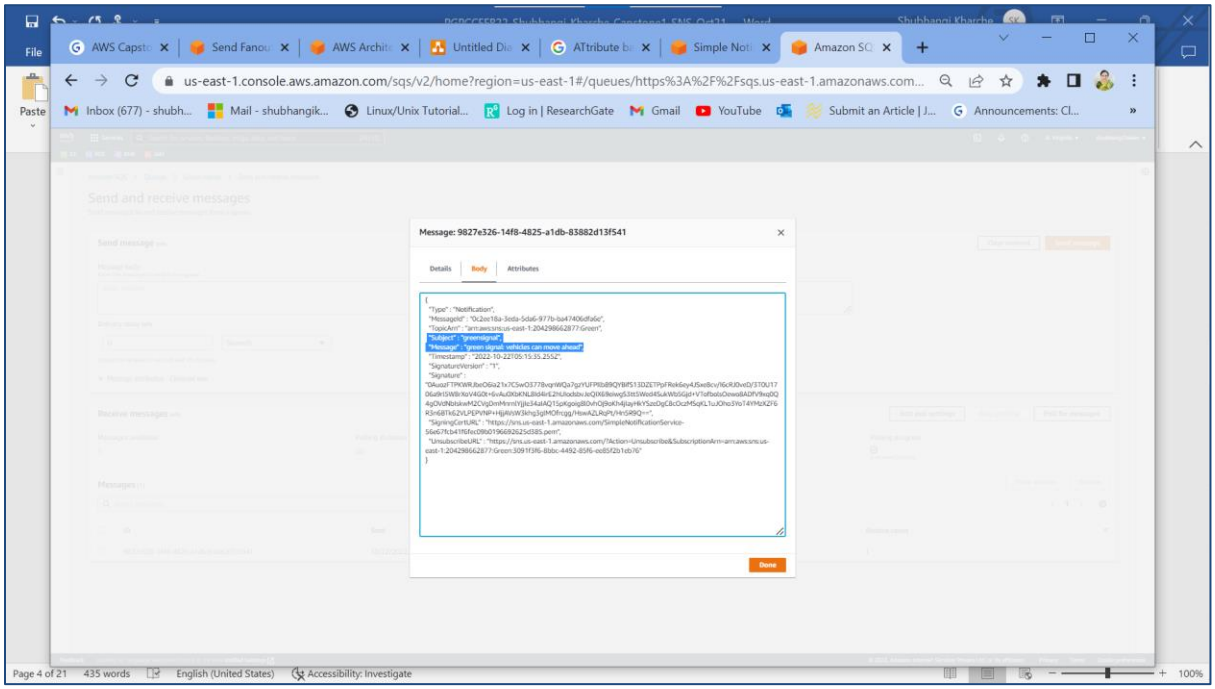
2d: Subscription created successfully for SNS topic named Green (status confirmed)



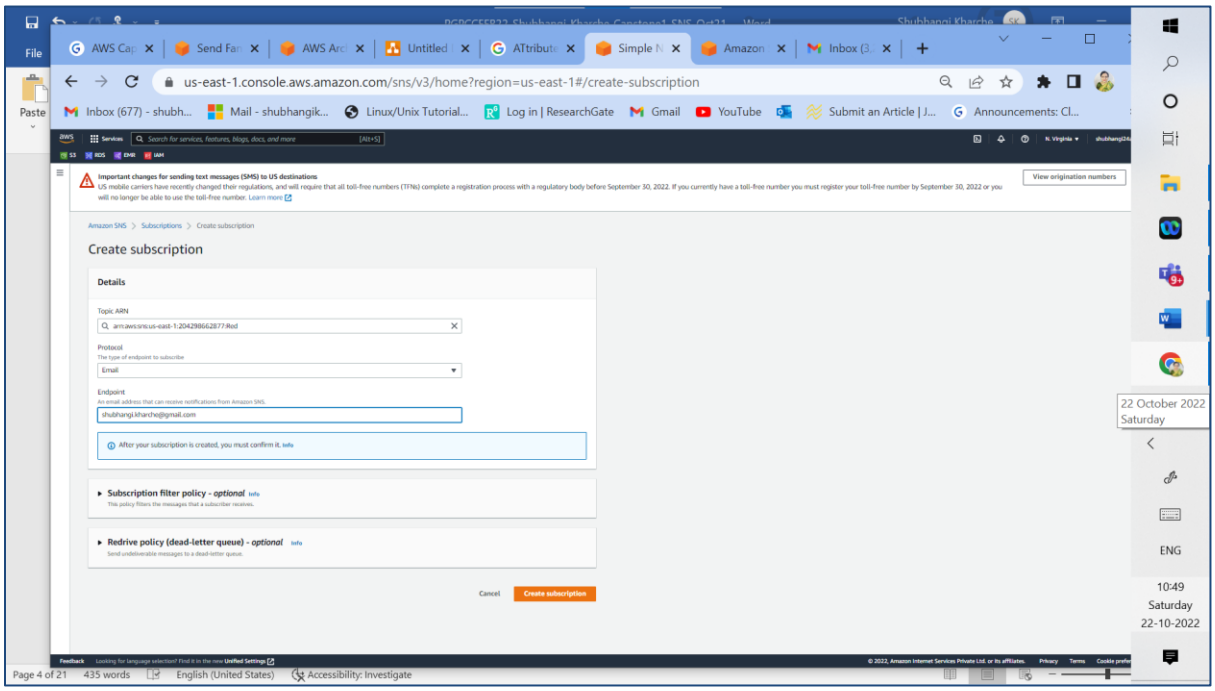
2e: Start polling messages for topic named Green from Amazon SQS



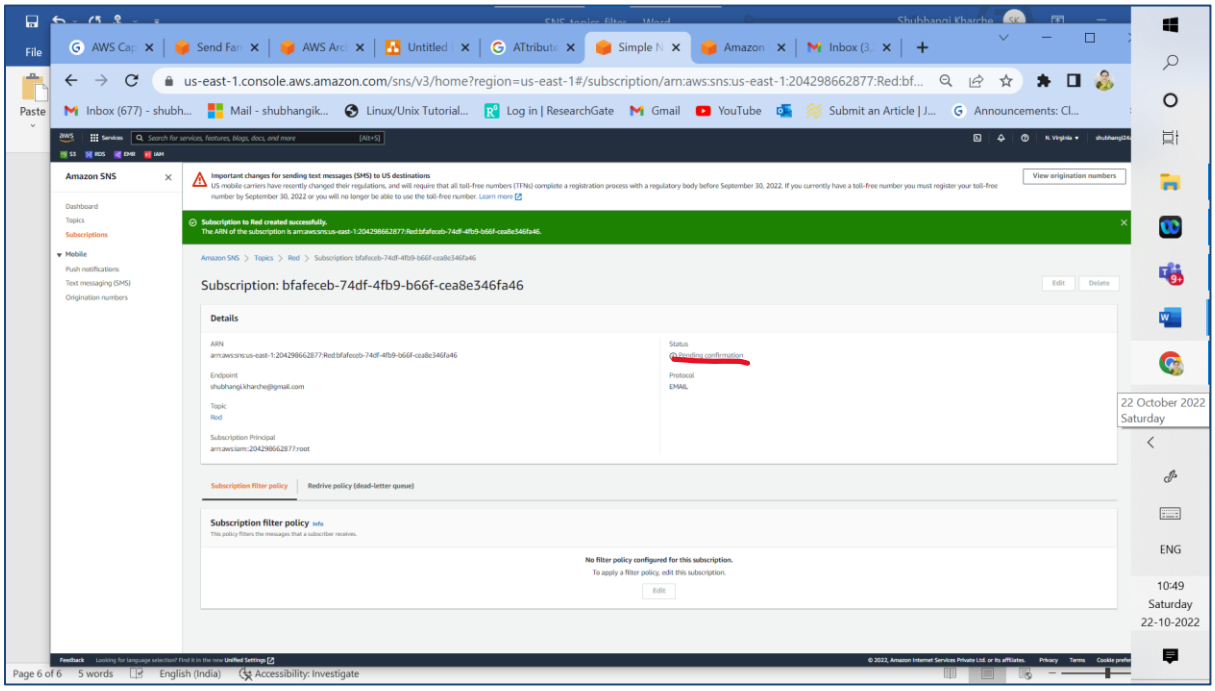
2f: As polling message progresses for topic named Green from Amazon SQS, one message is received.



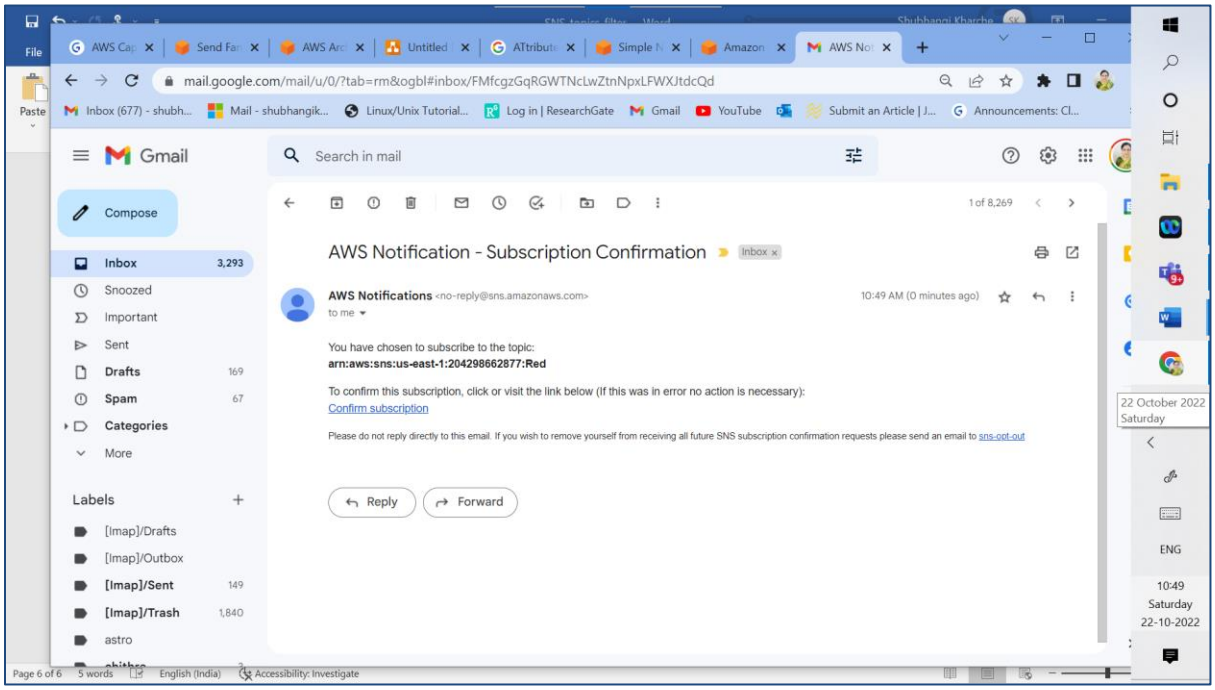
2g: Verified actual message received at Amazon SQS for green signal which is same as the sent one.



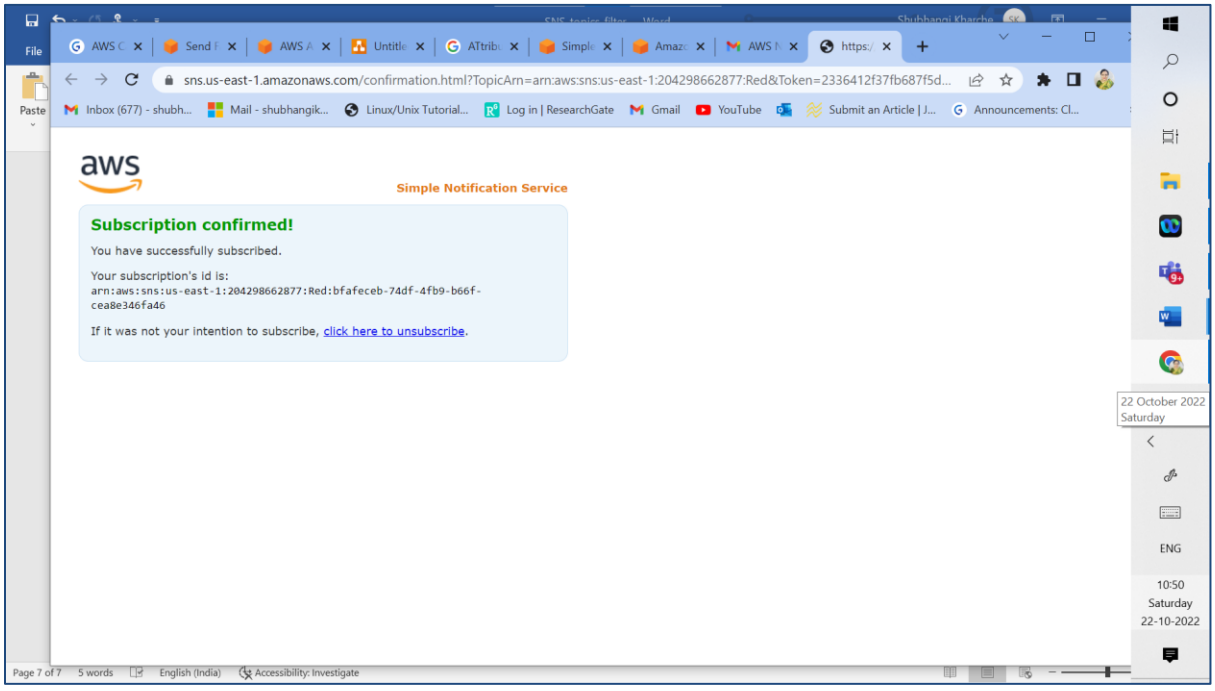
2h: Click on create subscription for topic named Red from Amazon SNS, select appropriate topic ARN from dropdown, select Email protocol as type of endpoint, enter email id in the endpoint address and hit create subscription.



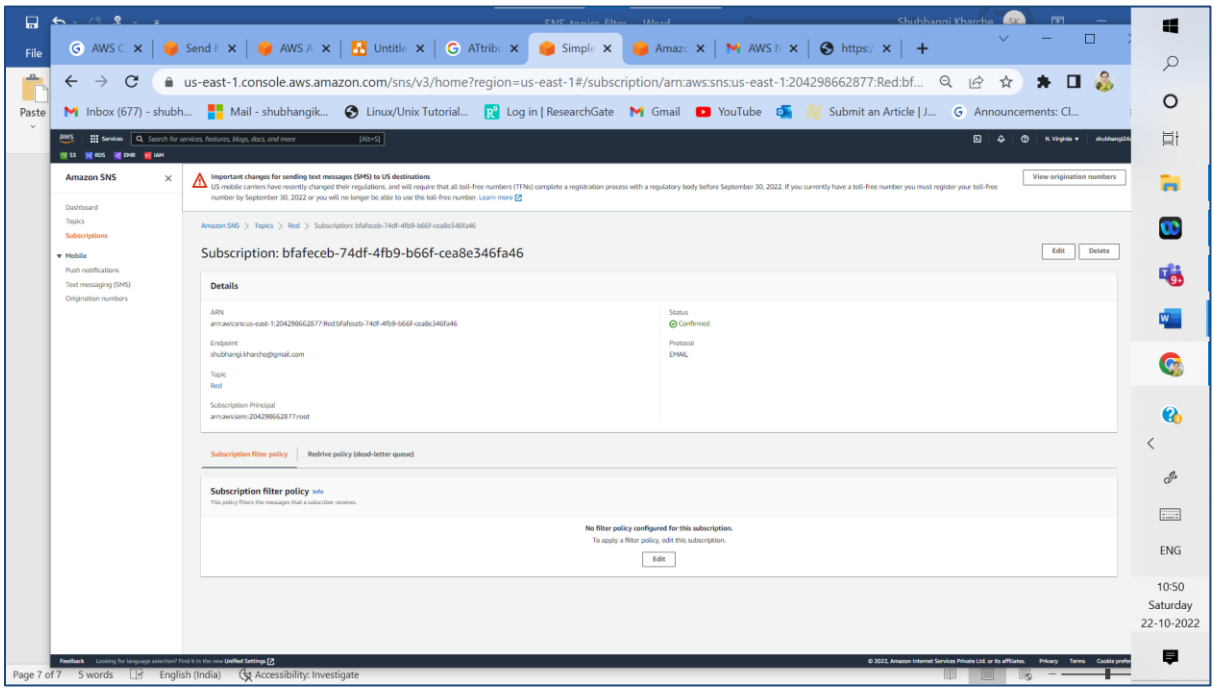
2i: Subscription to SNS topic named Red is successful but confirmation is pending



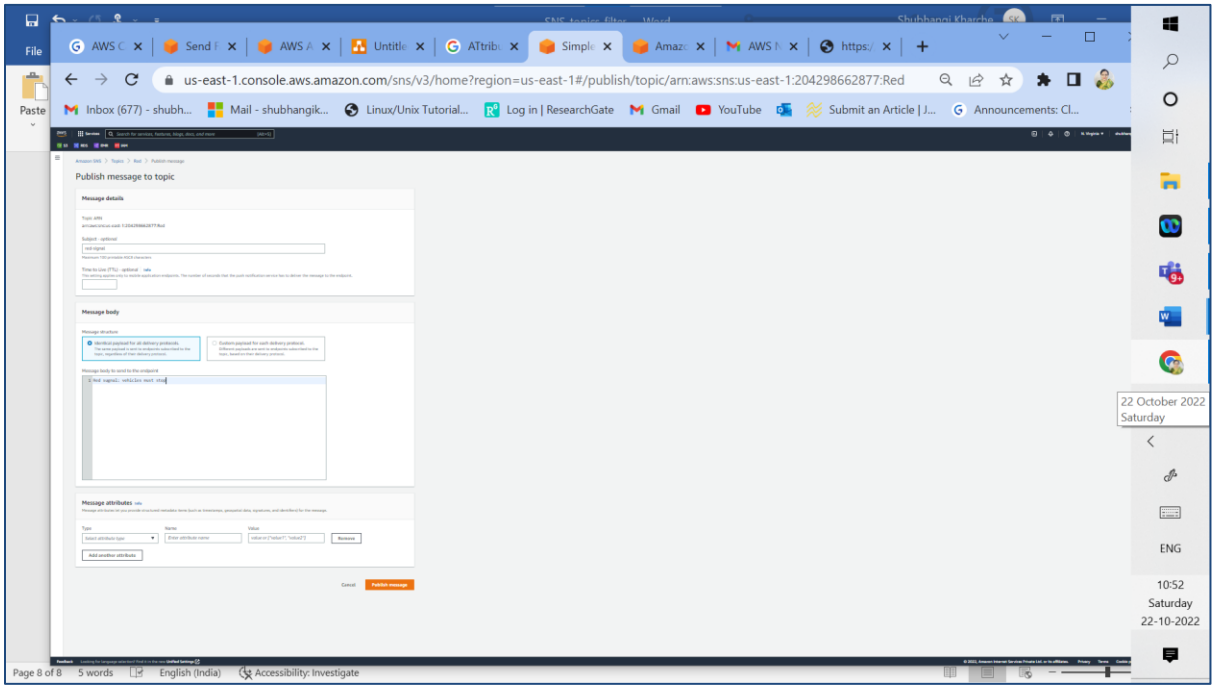
2i: Confirm pending subscription from received email notification



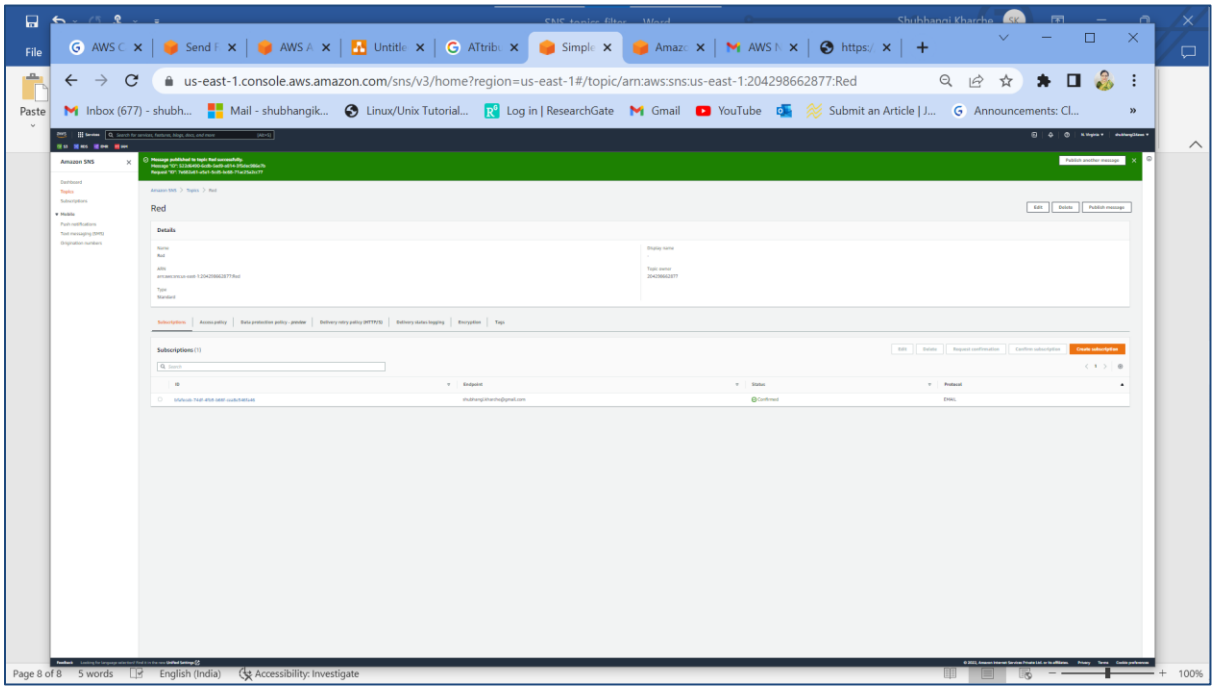
2j: Pending Subscription confirmed



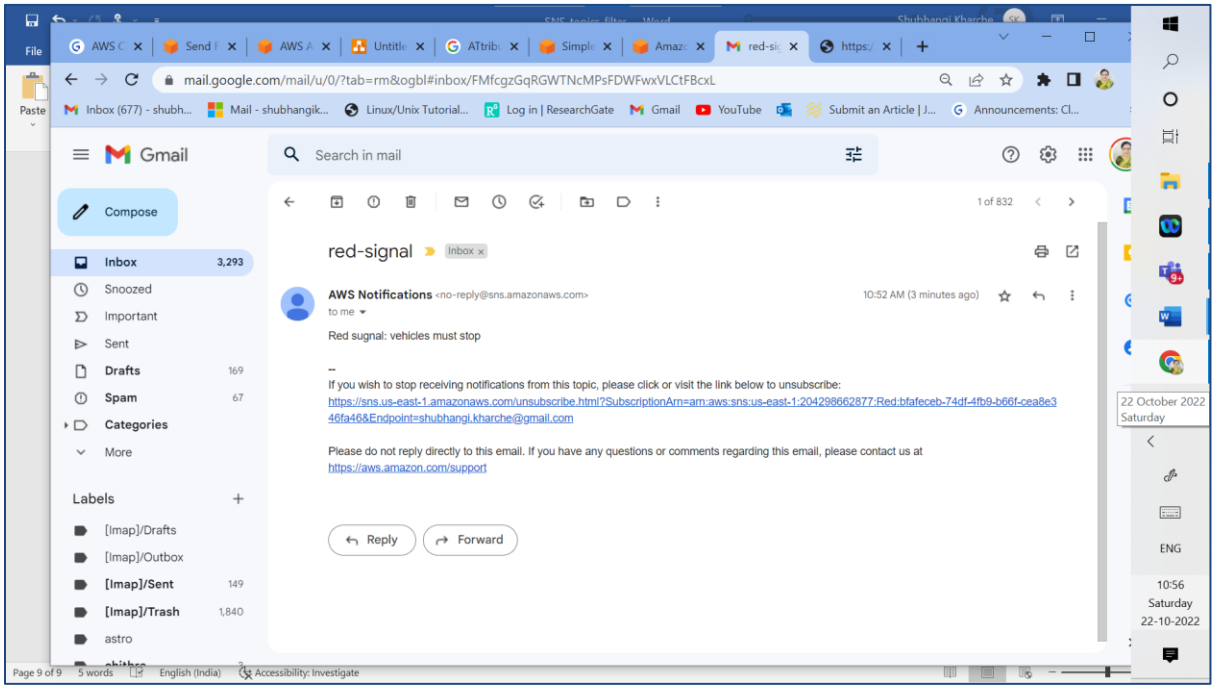
2k: Pending Subscription confirmed status for topic named Red



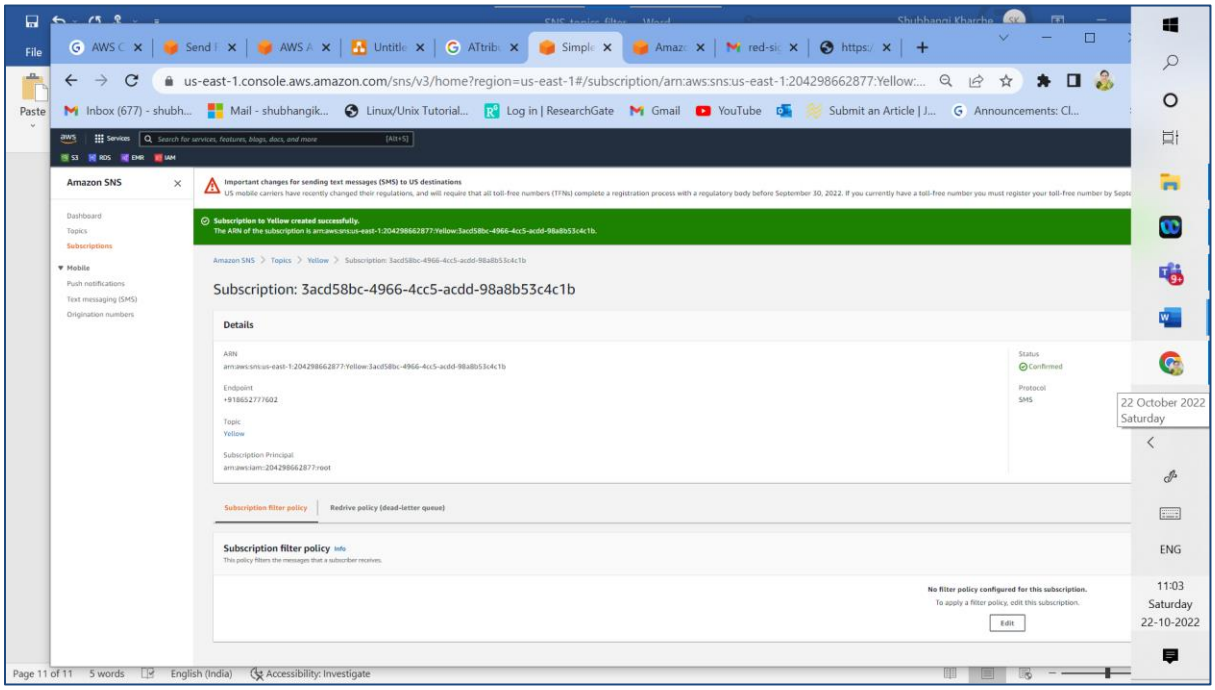
2k: Publish message Red Signal: vehicles must stop for topic named Red



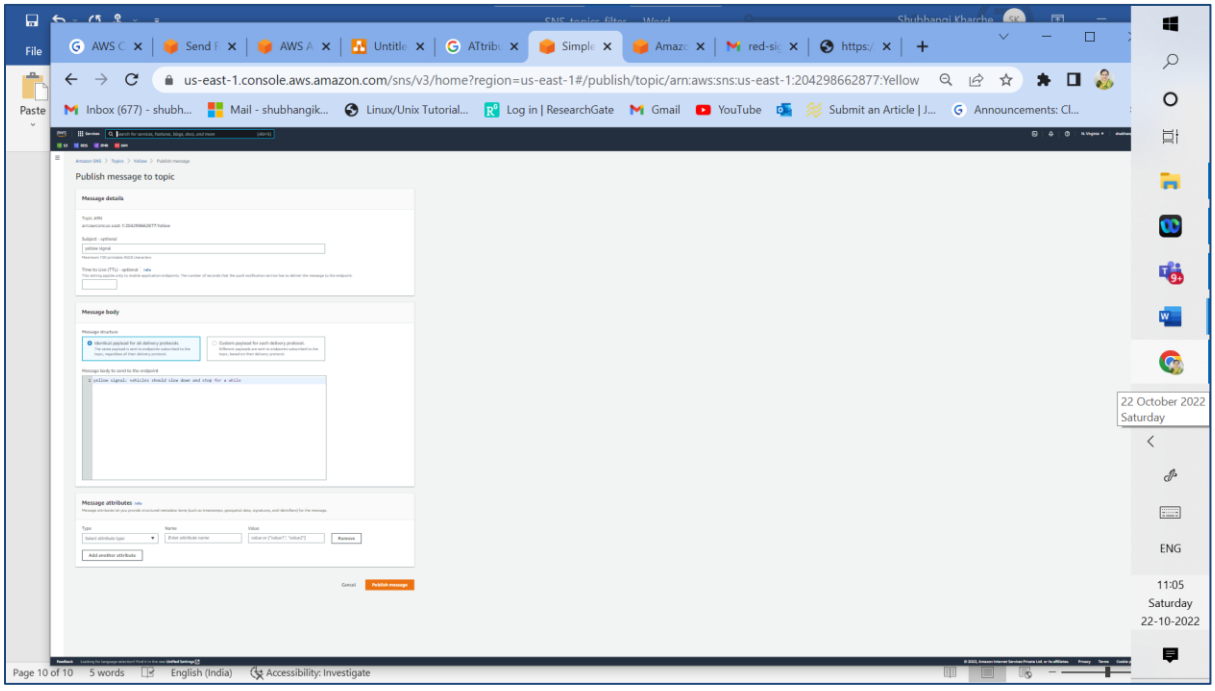
2l: Message published to topic named Red successfully



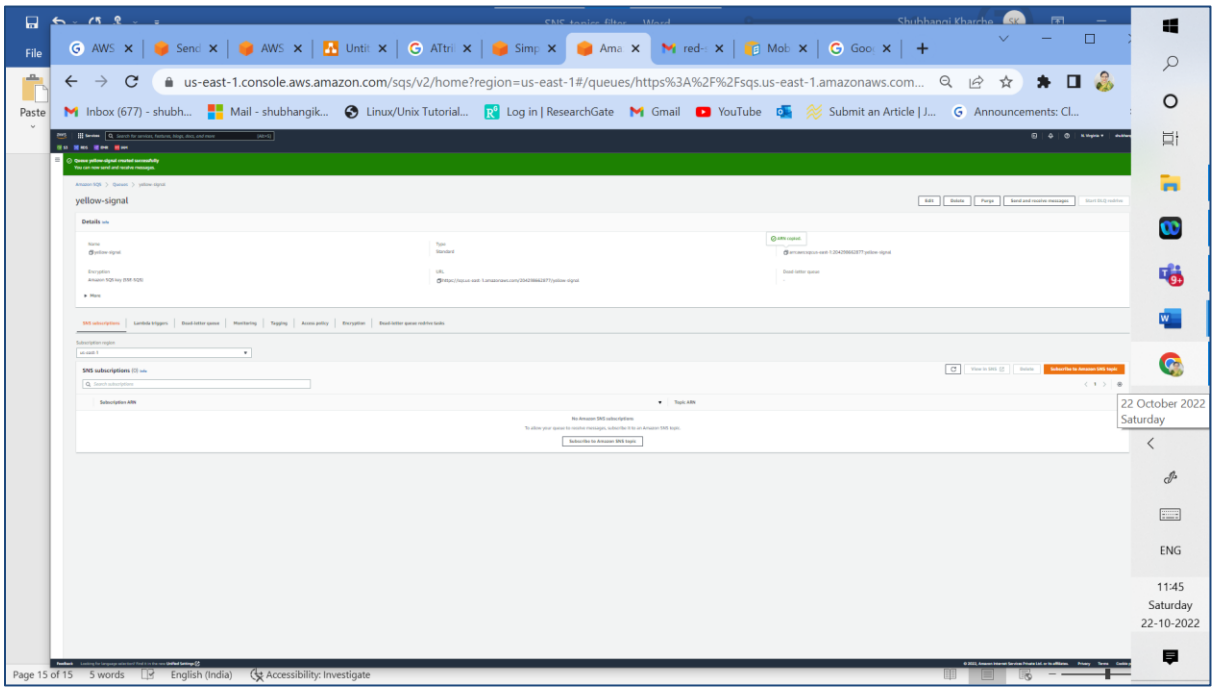
2m: Message received successfully via email for subscribed red topic with subject red-signal and message Red signal: vehicles must stop



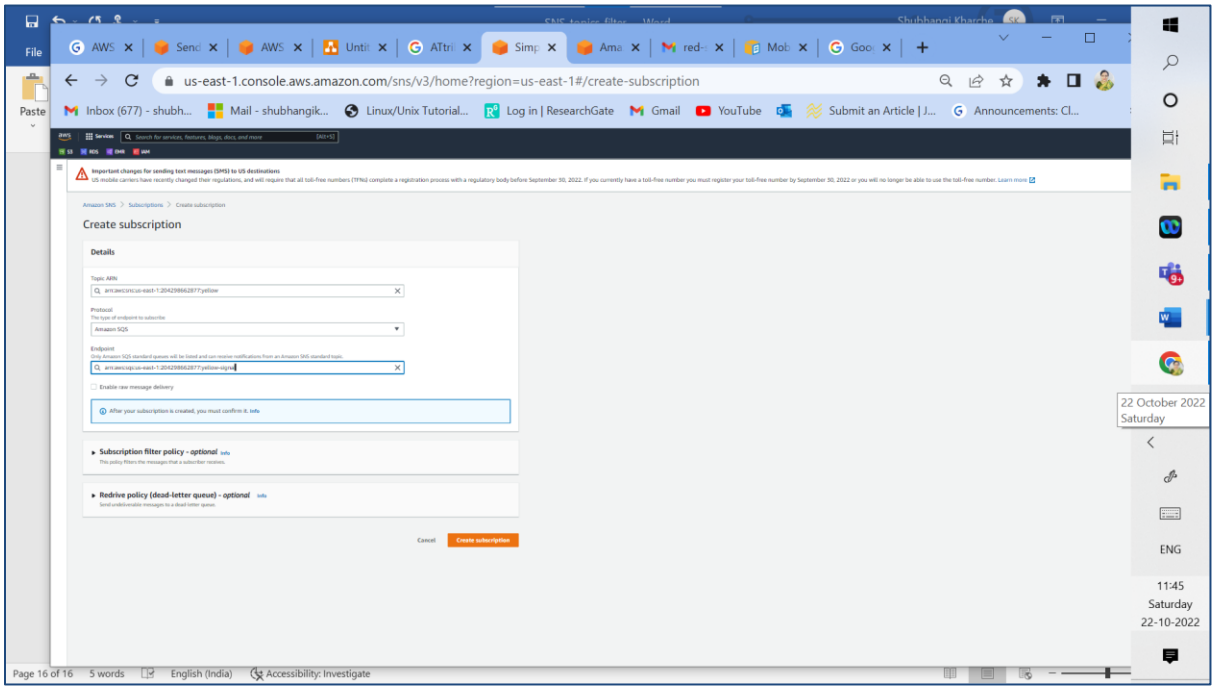
2n: Subscription to yellow topic created successfully



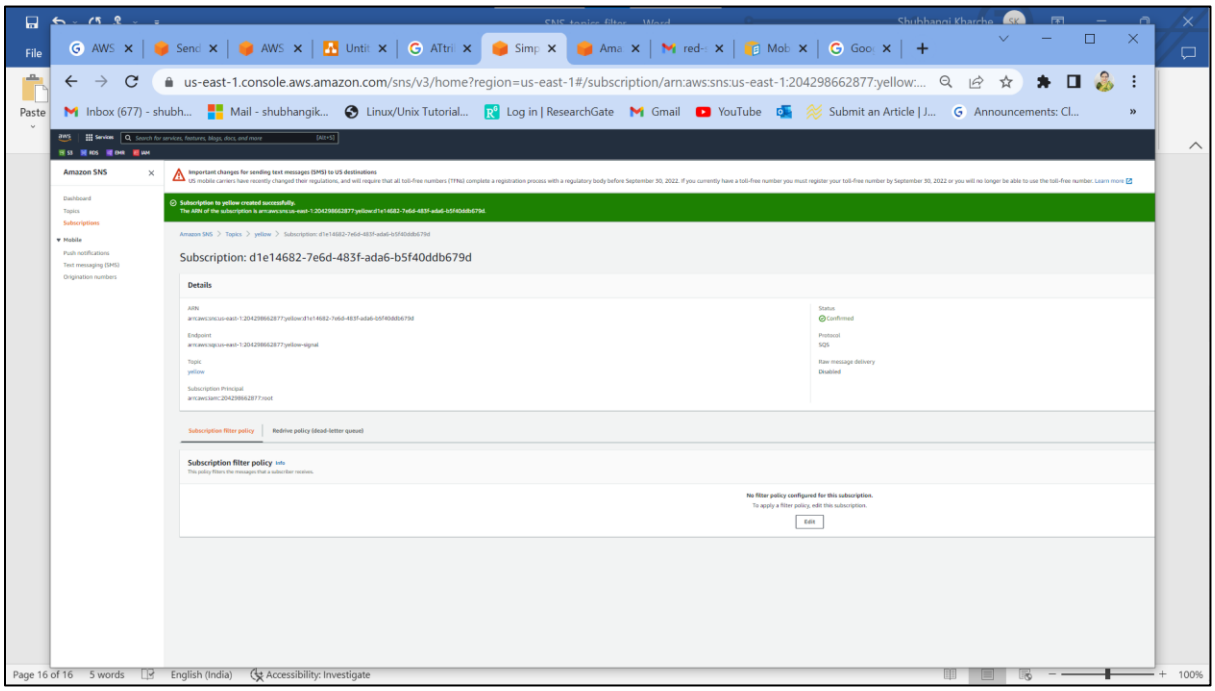
2o: Publish message to yellow topic with subject: yellow signal and message: yellow signal: vehicles should slow down and stop for a while



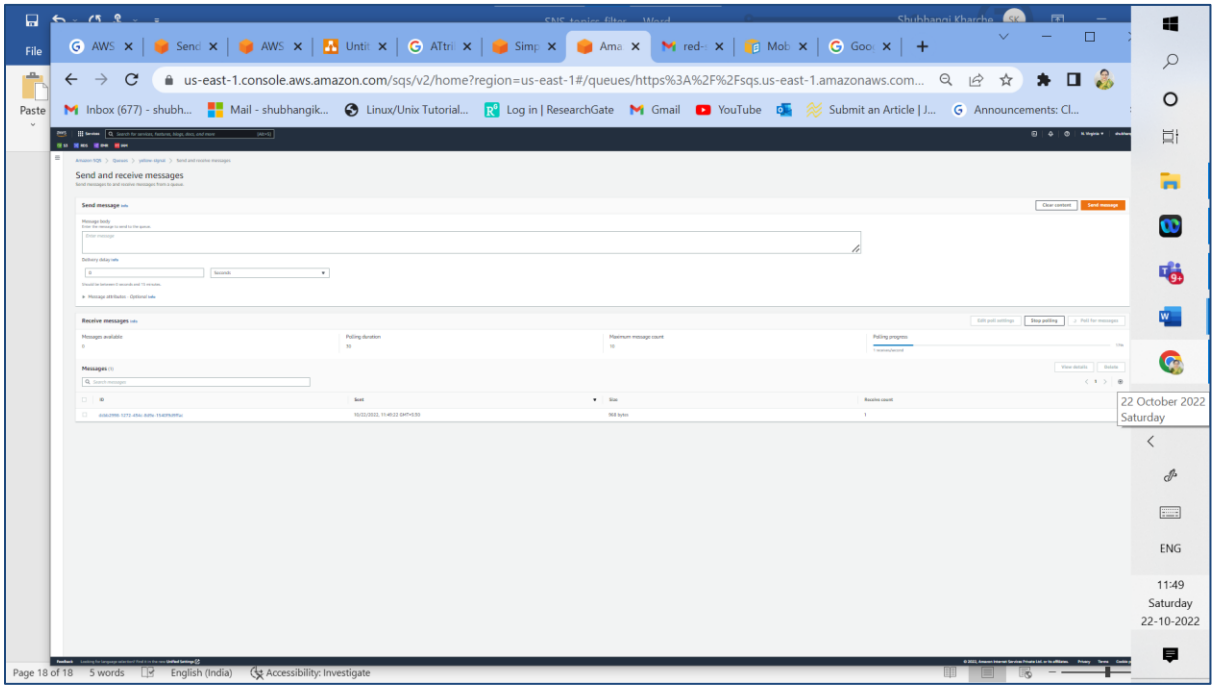
2p: Create SQS Queue Yellow-signal



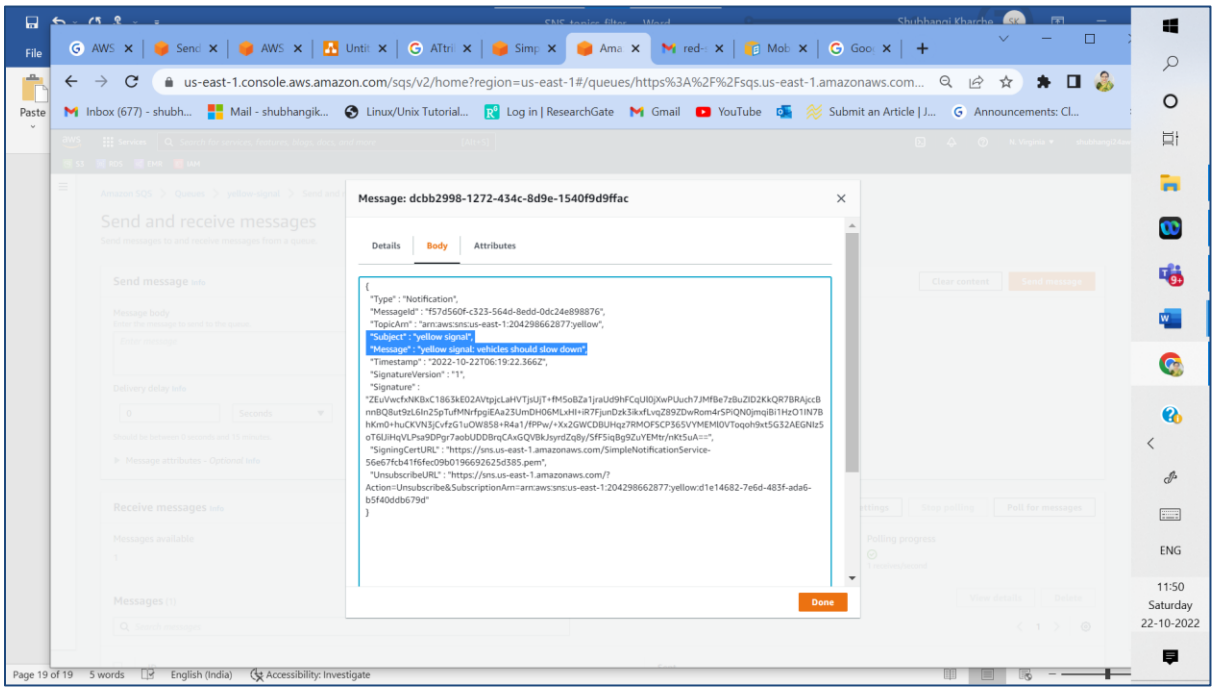
2q: Create subscription for yellow topic, select appropriate topic ARN, type of endpoint as Amazon SQS, endpoint as ARN for Amazon SQS



2r: Subscription created and confirmed successfully for yellow topic



2s: Start polling messages from send and receive messages menu of Amazon SQS for yellow topic, one message received as polling progresses.



2t: message received successfully in Amazon SQS from yellow topic with subject: yellow signal and message: yellow signal: vehicles should slow down (same as sent)

II b) Attribute Based filtering

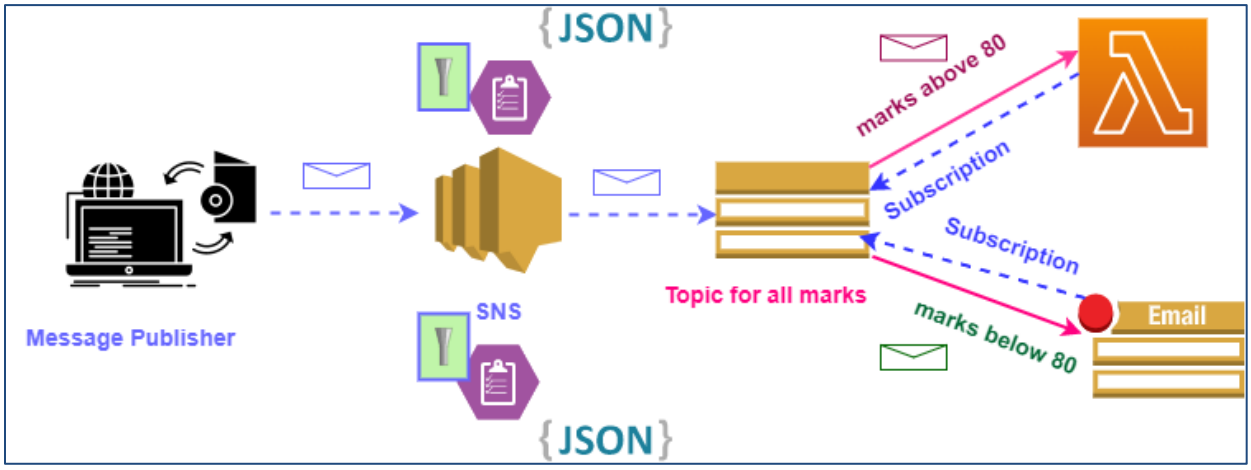
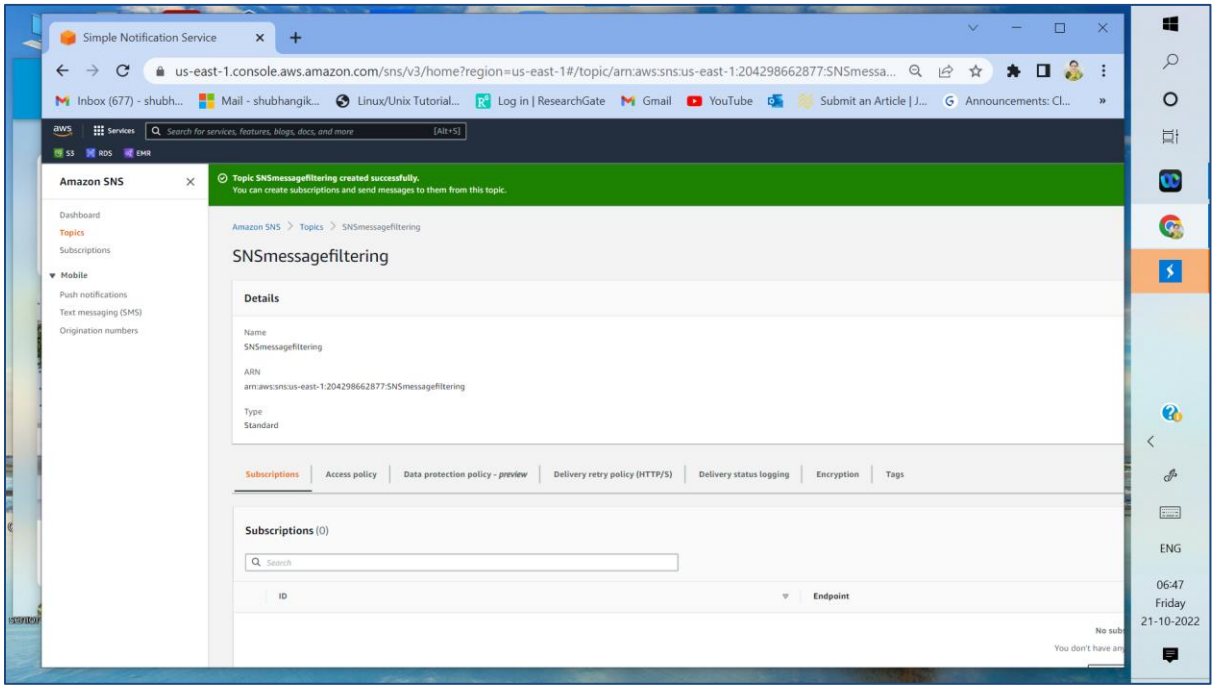
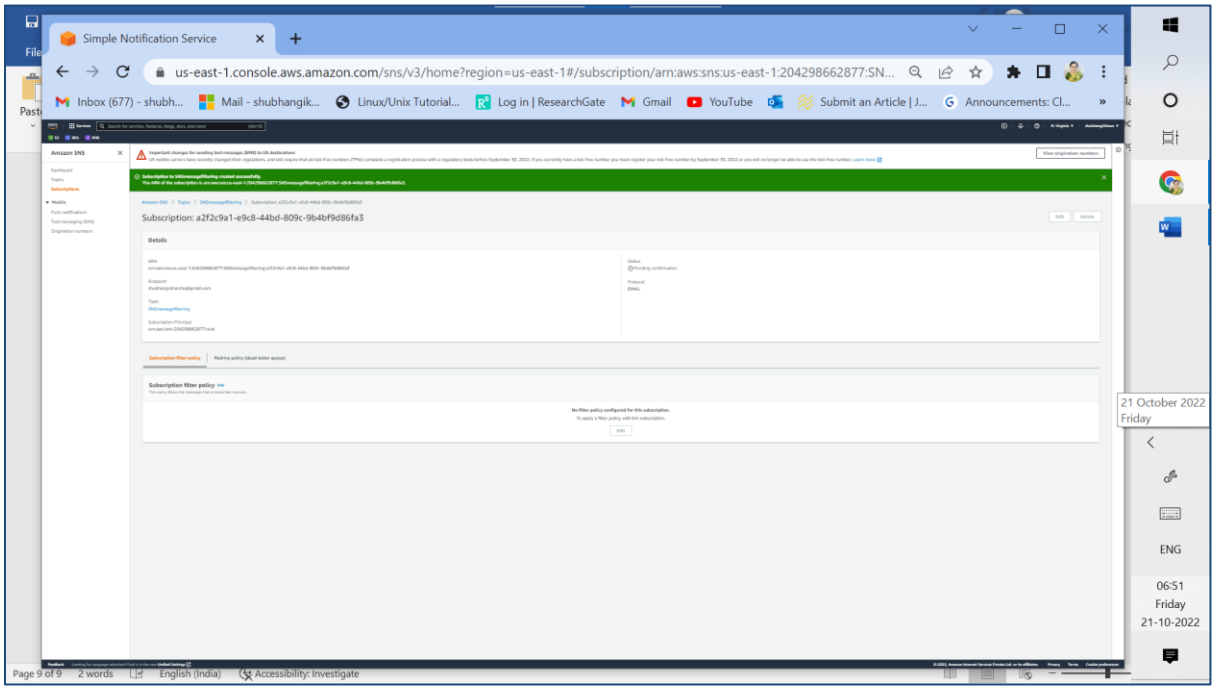


Fig 3: Messaging scenario using Amazon Simple Notification Service (SNS), AWS Lambda and Email (with attribute-based message filtering for sending student marks)

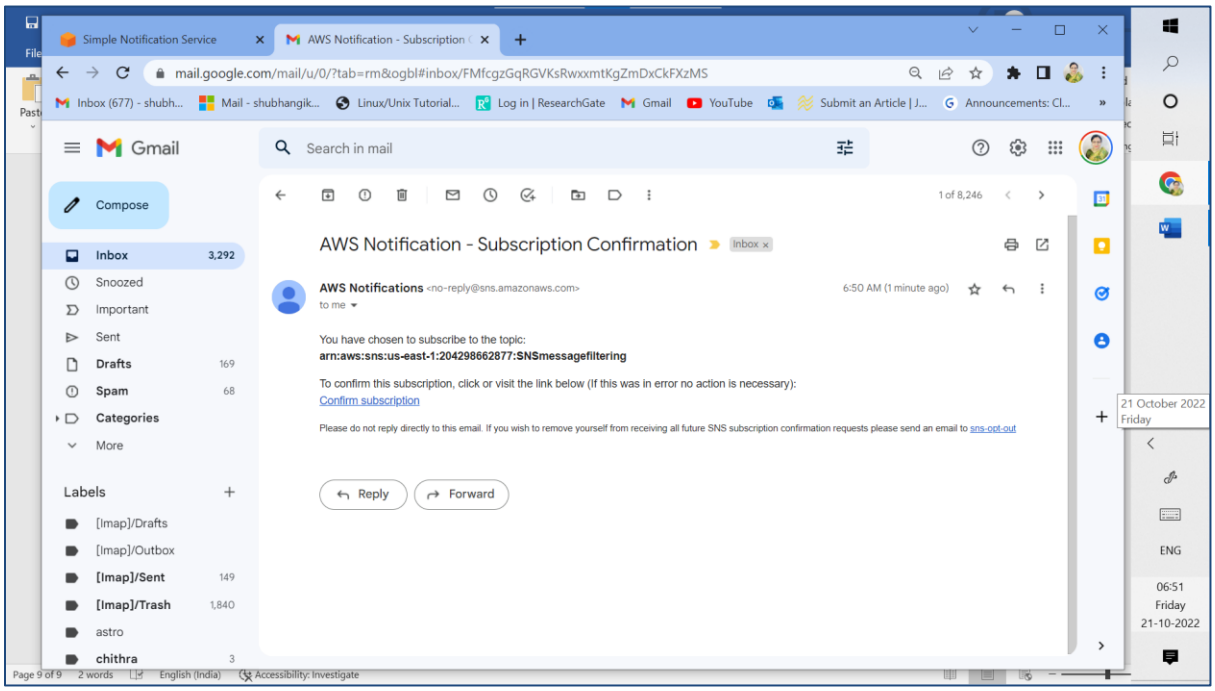
Implementation Screenshots for Fig 3:



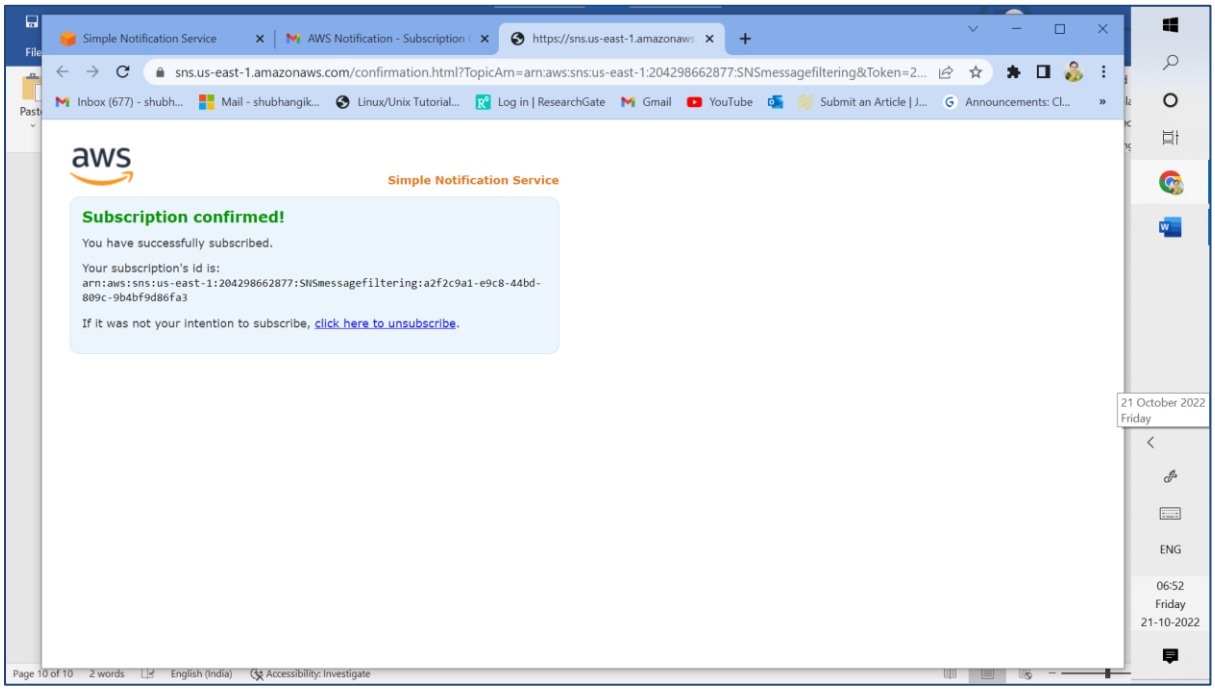
3a: Create SNS topic named SNSmessagefilering



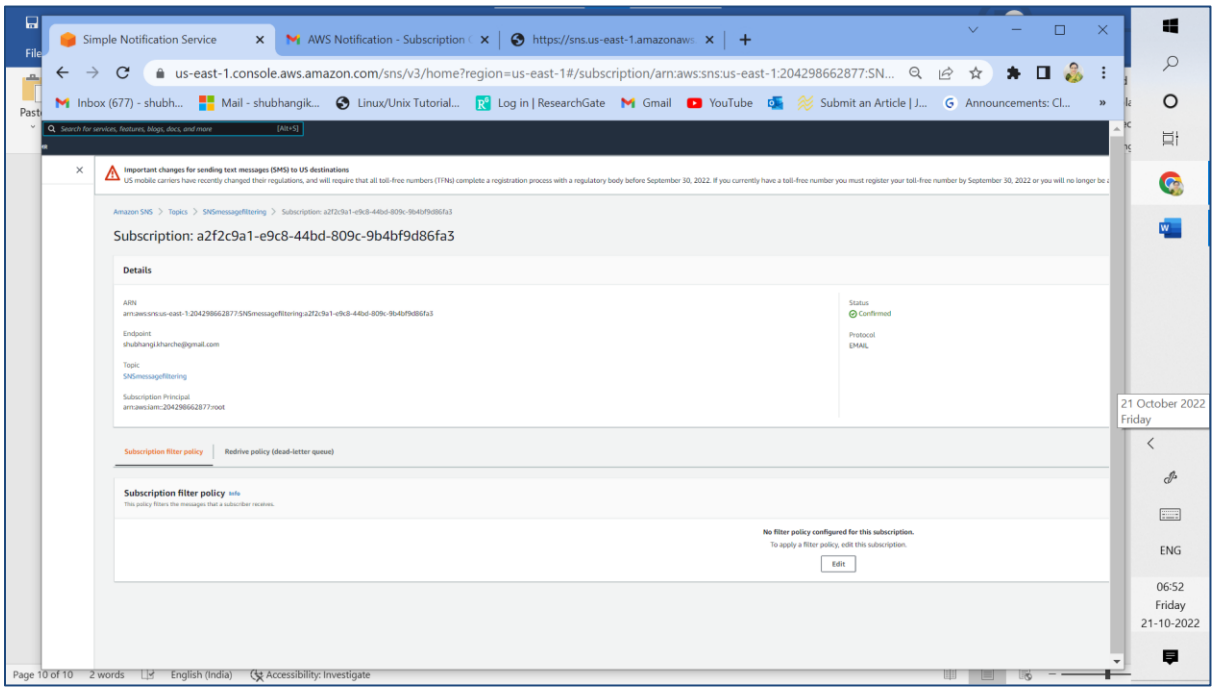
3b: Subscription to SNSmessagefiltering topic created successfully for email endpoint but confirmation is pending



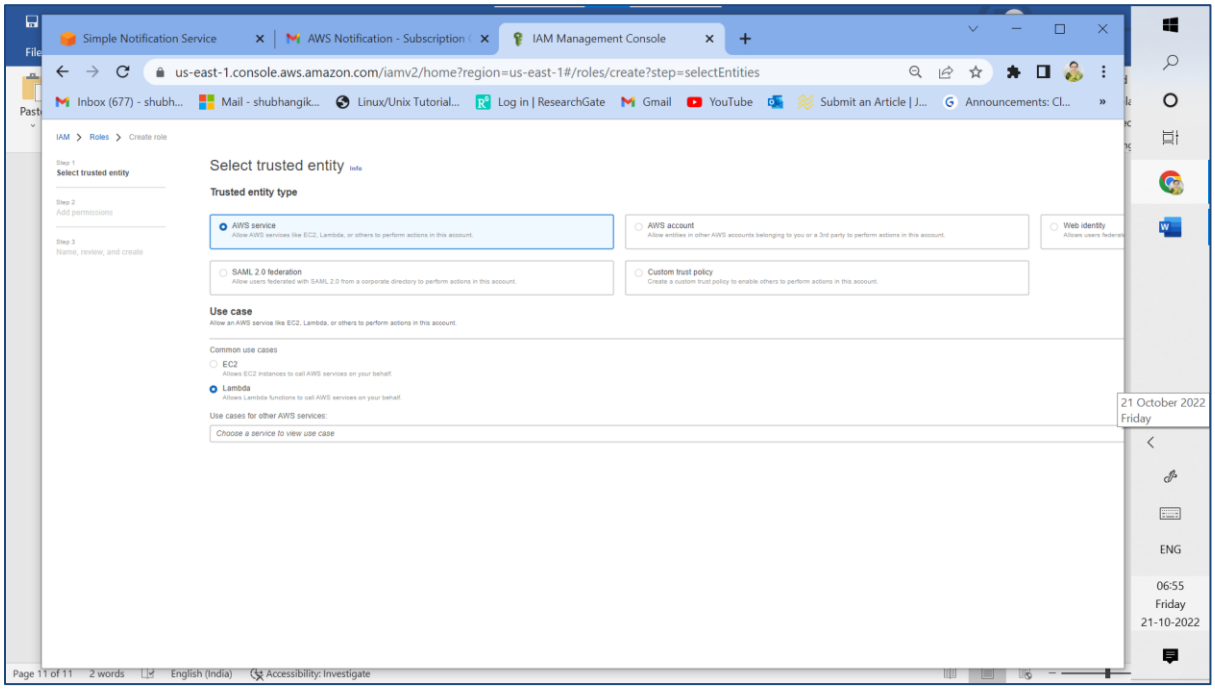
3c: AWS notification received to confirm subscription



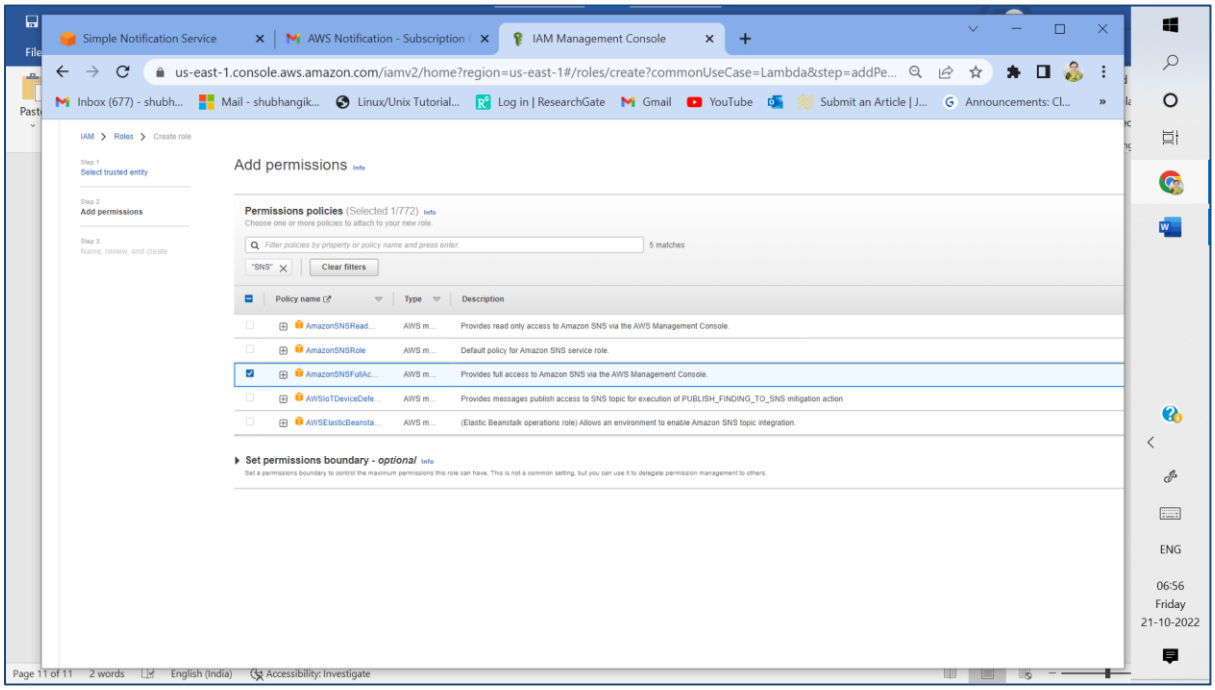
3d: subscription confirmed from received email



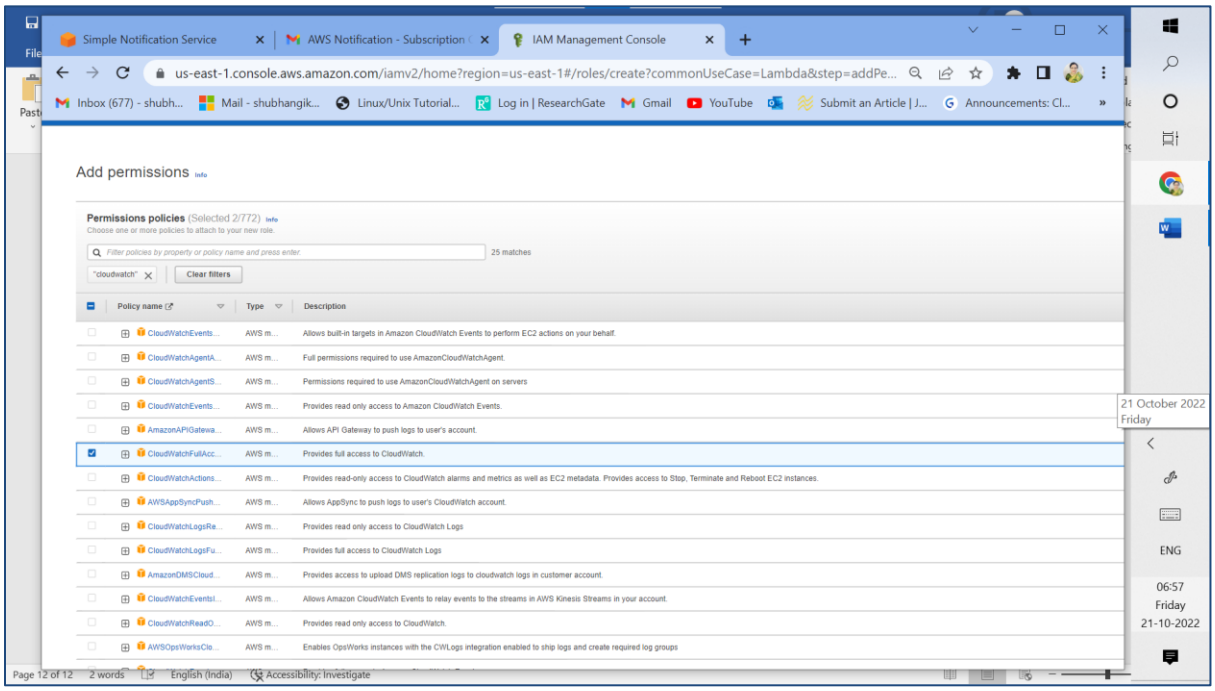
3e: Confirmed Subscription for SNSmessagefiltering topic



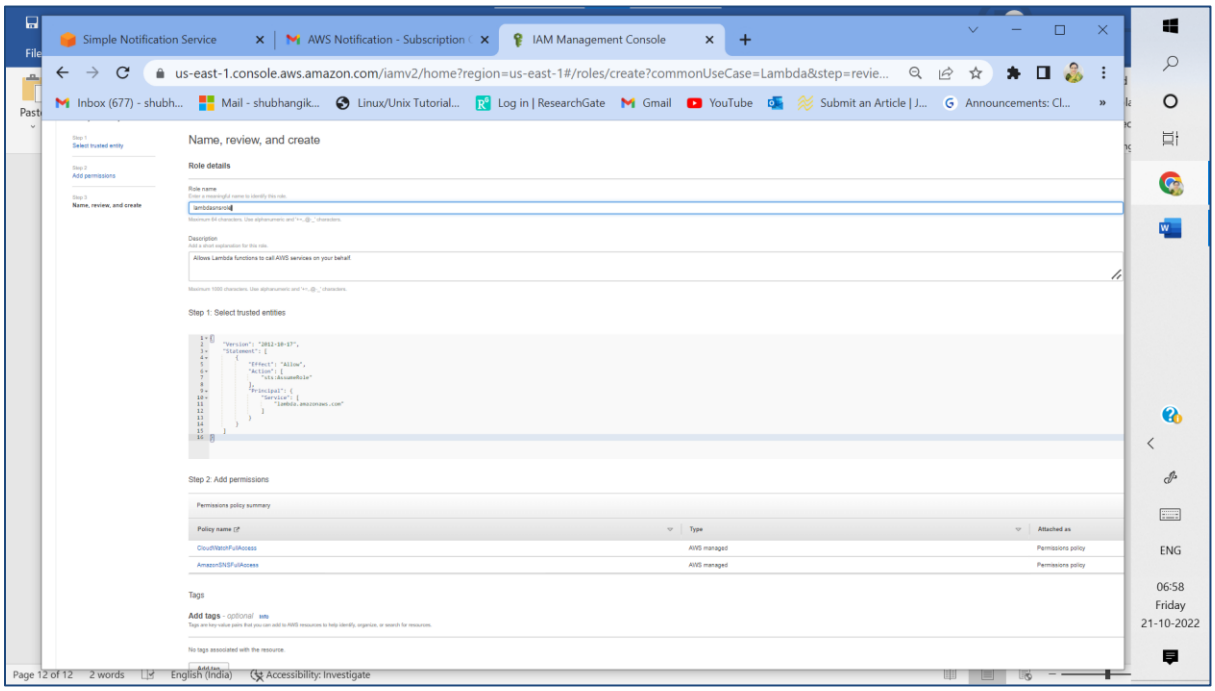
3f: Create IAM role for AWS Lambda



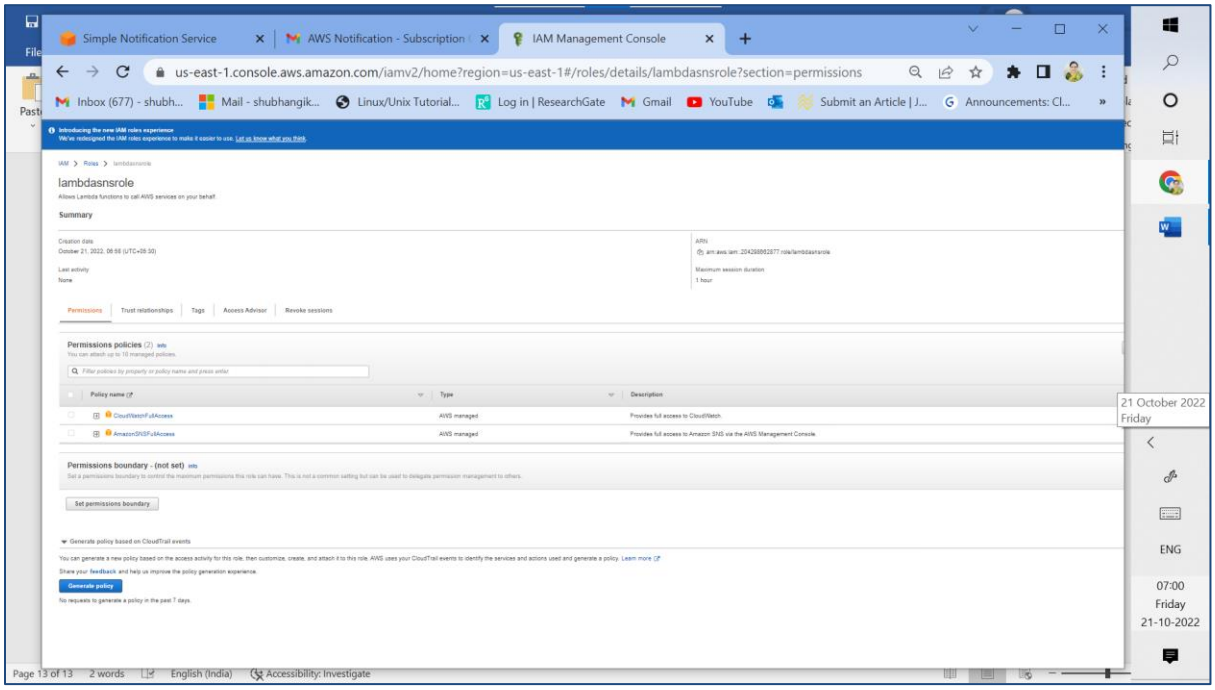
3g: Add full access permission to Amazon SNS



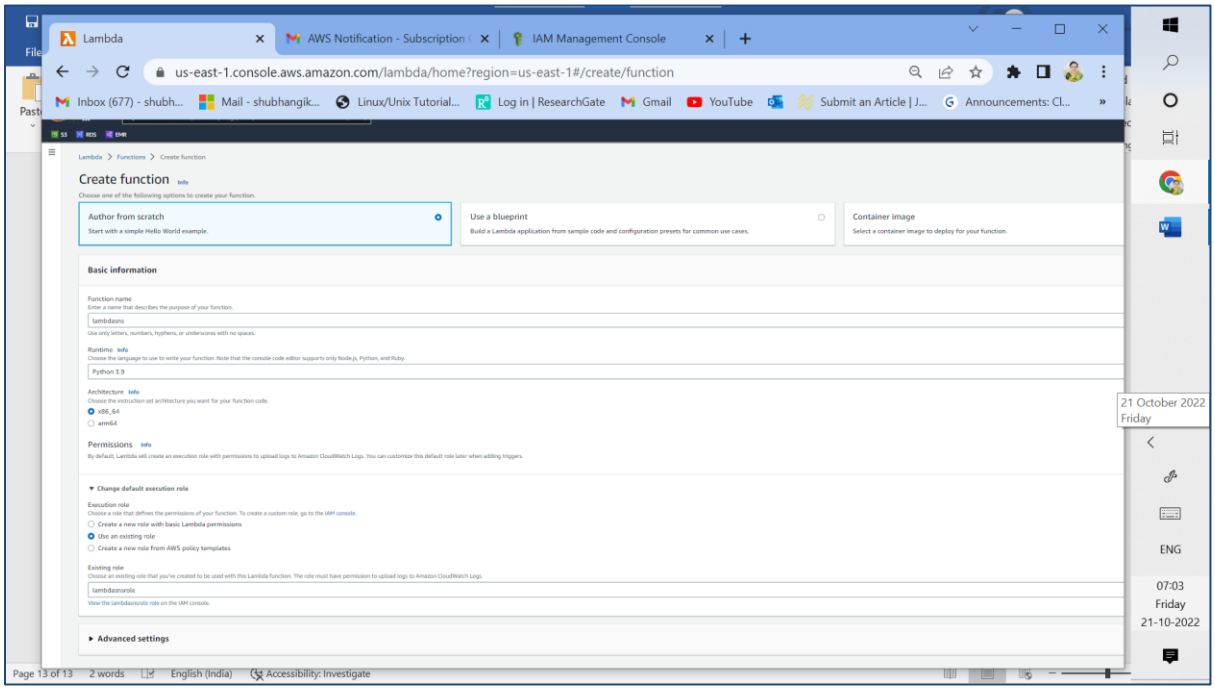
3h: Add full access permission to AWS cloudwatch



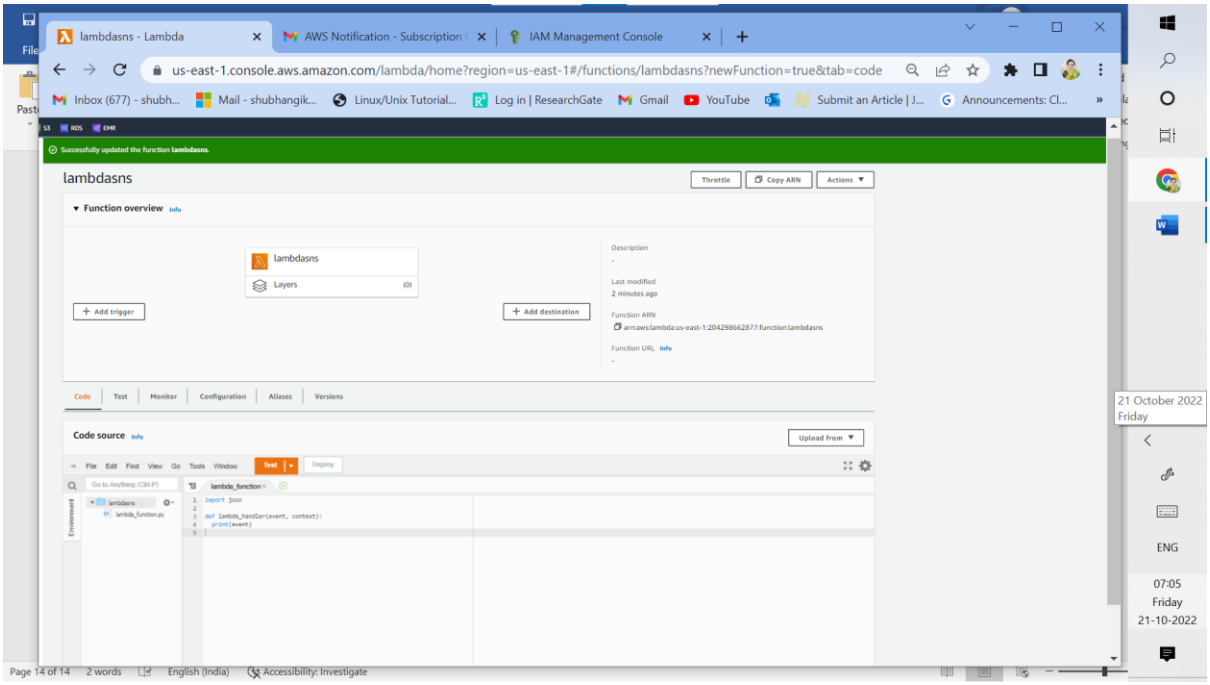
3i: Add role name as lambdasnsrole



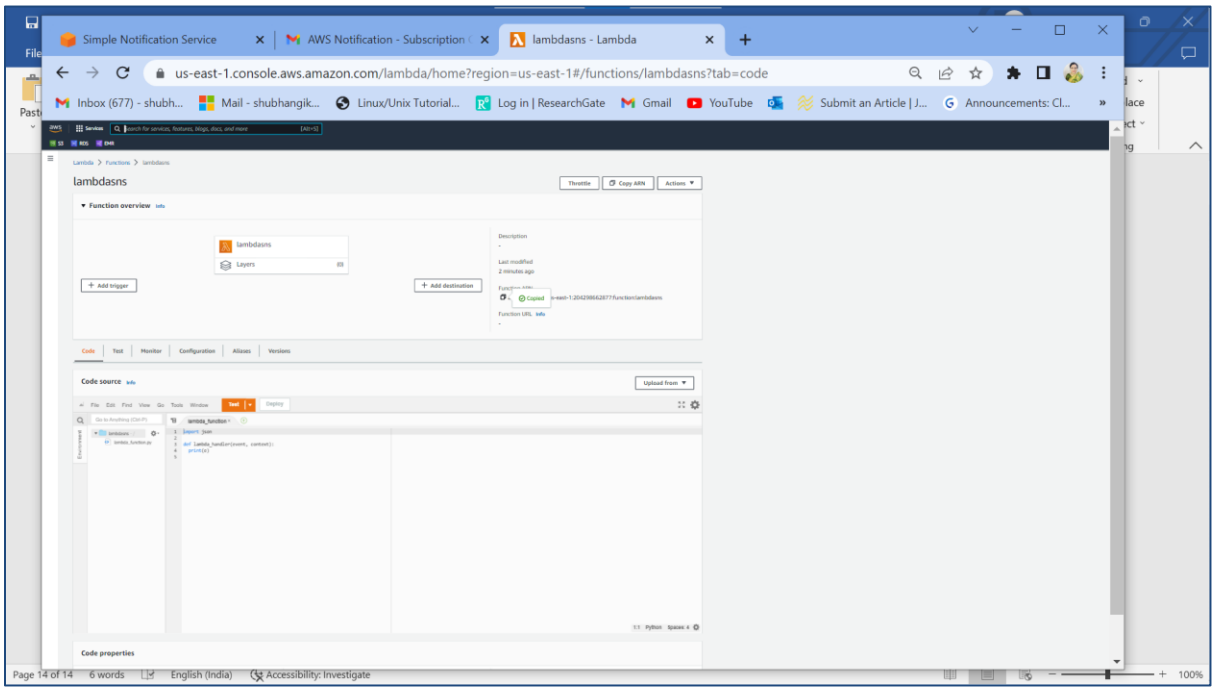
3j: lambdasnsrole IAM role created successfully with full access permissions to lambda for cloudwatch and SNS services



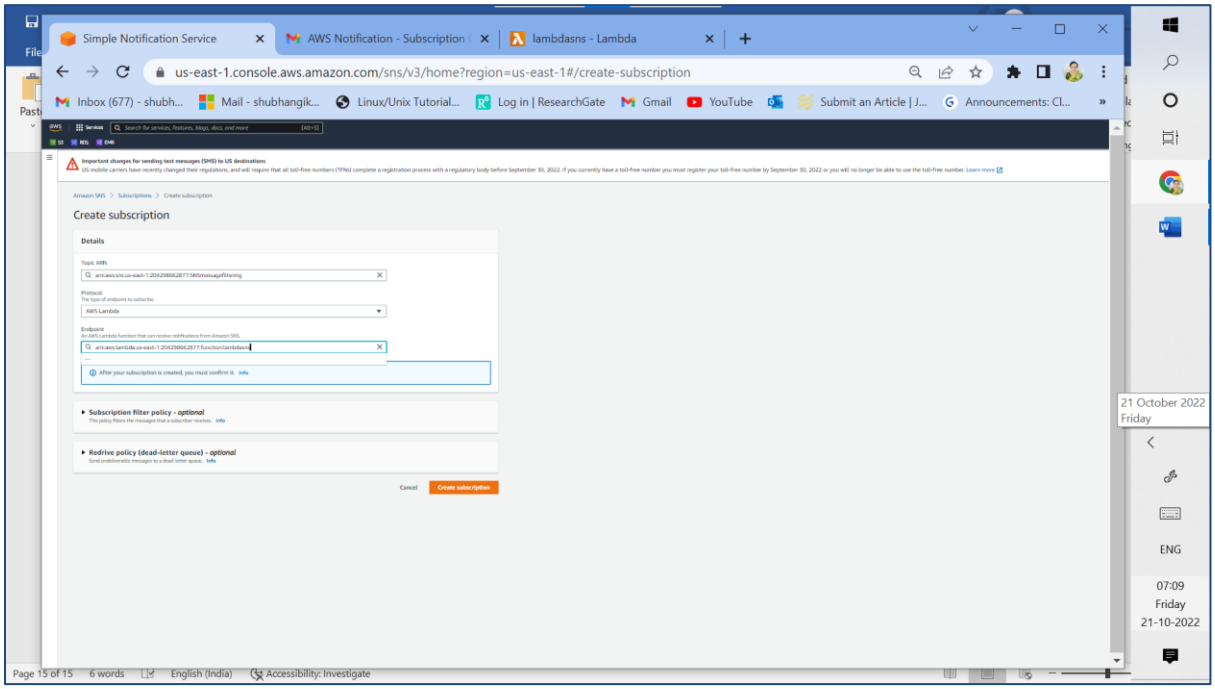
3k: Create lambda function lambdasns with python 3.9 as runtime, existing role lambdasnsrole and using author from scratch option



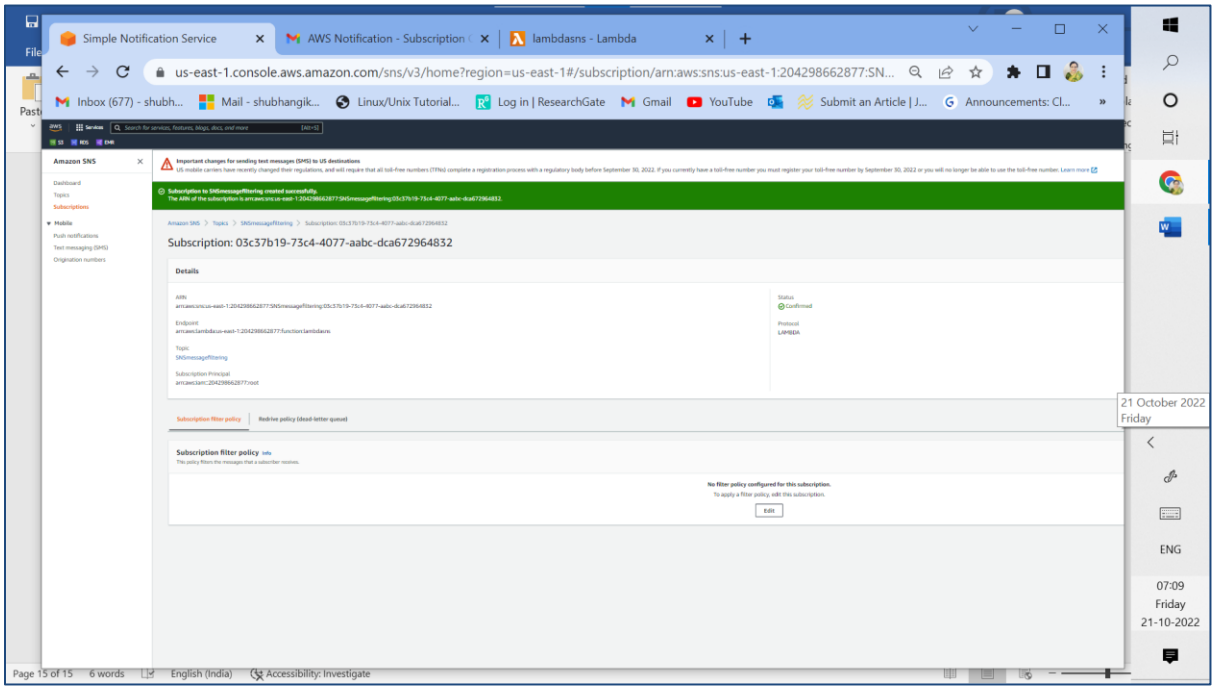
3l: Edit lambda function JSON script to print event. After editing the script deploy it.



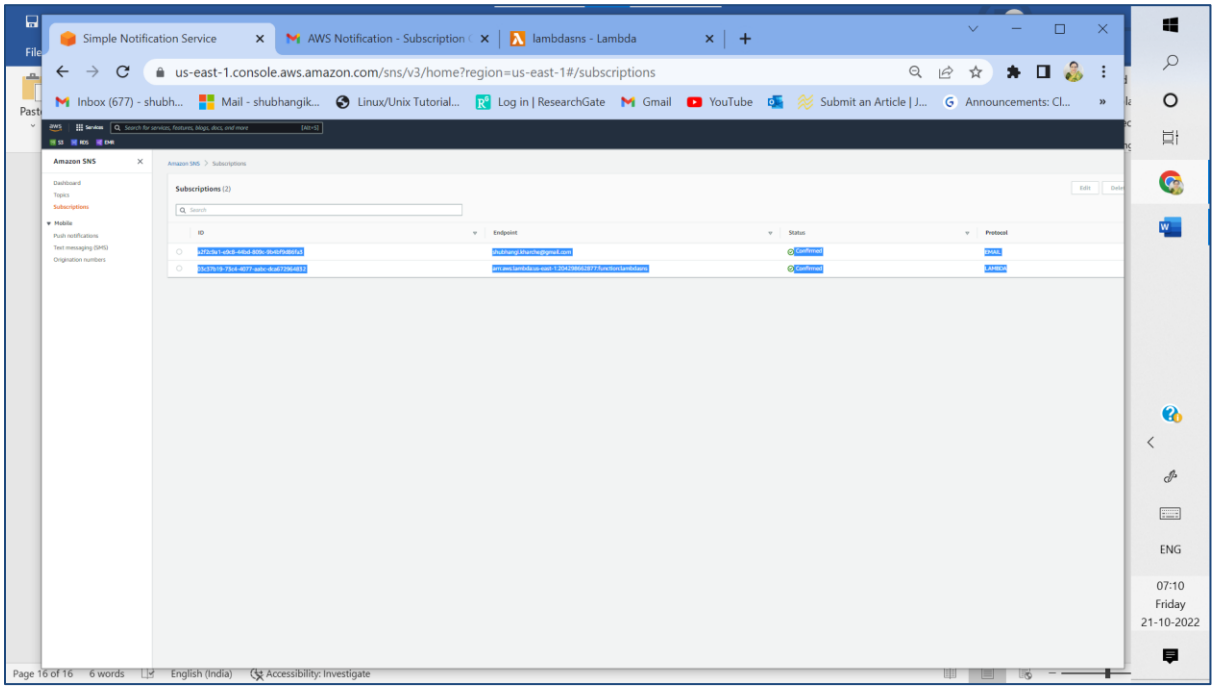
3m: Copy lambda function ARN which is required while subscribing topic to lambda



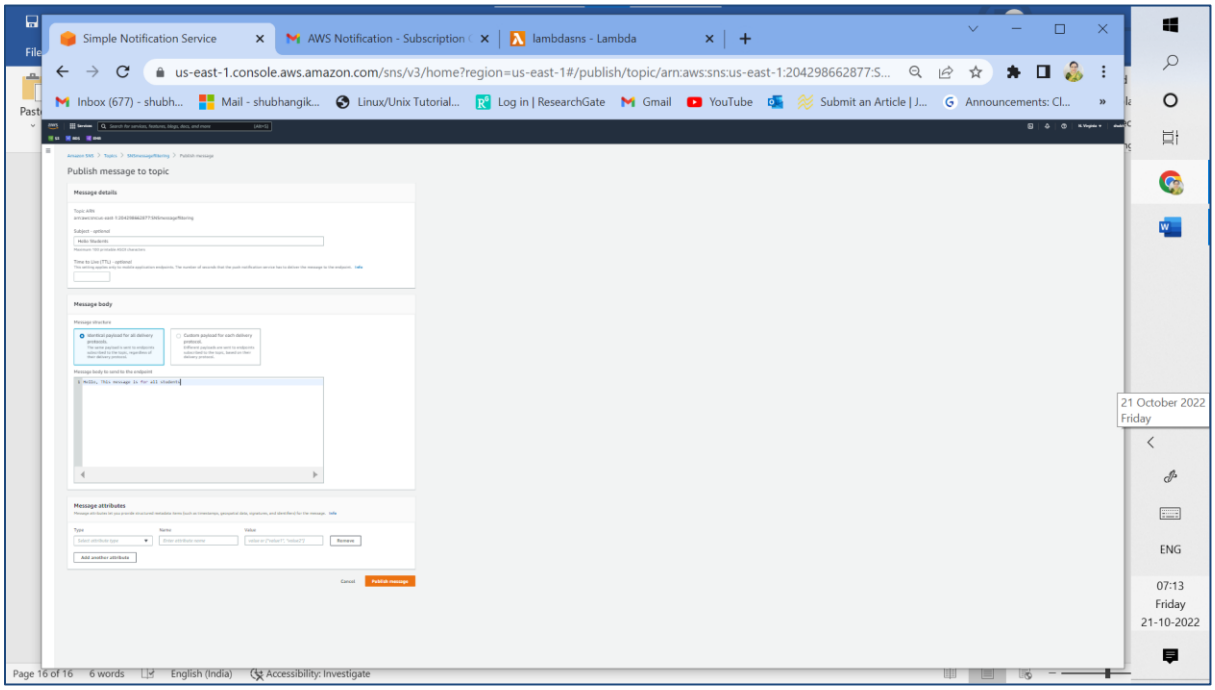
3n: Subscribe SNS topic to AWS lambda, select appropriate topic ARN and lambda function ARN



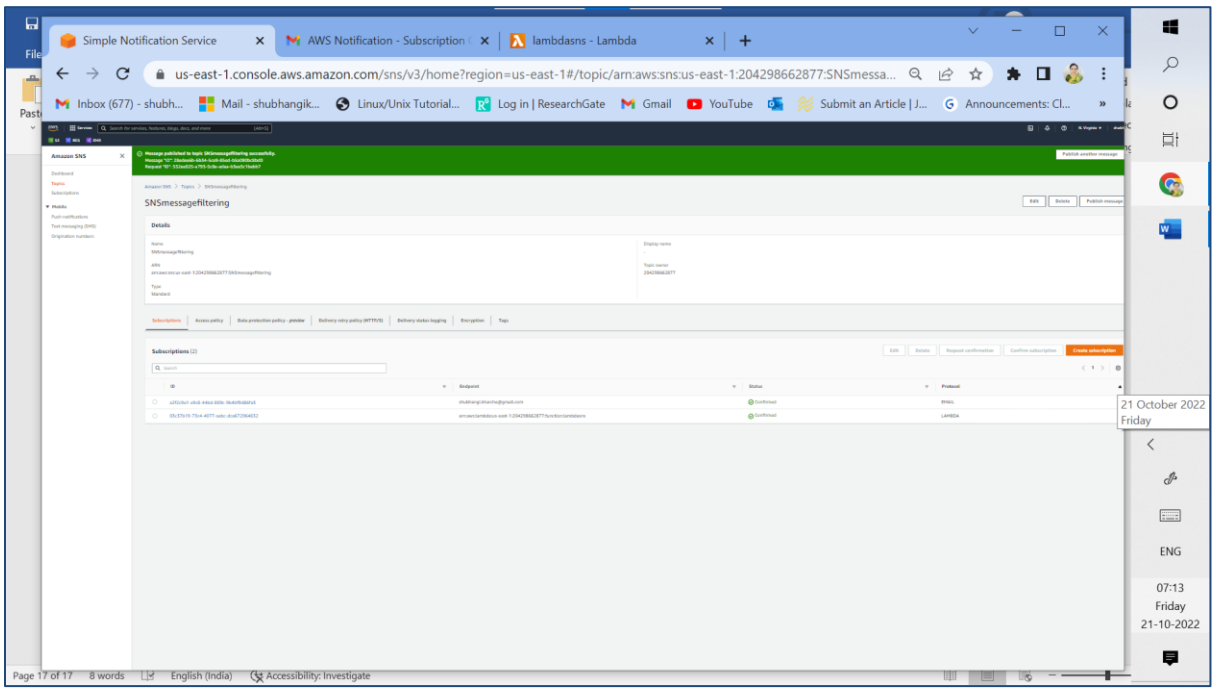
3o: Successfully created subscription to the topic SNSmessagefiltering for lambda function with confirmed status



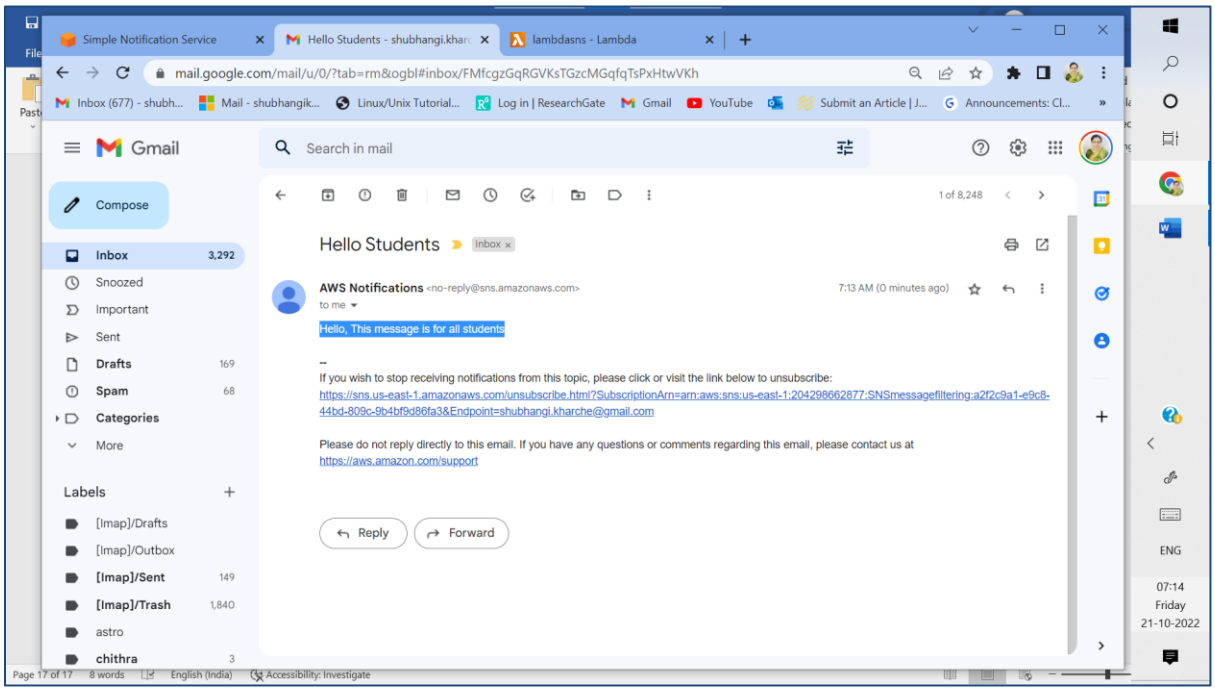
3p: Both email and lambda subscriptions are now seen as confirmed under SNS



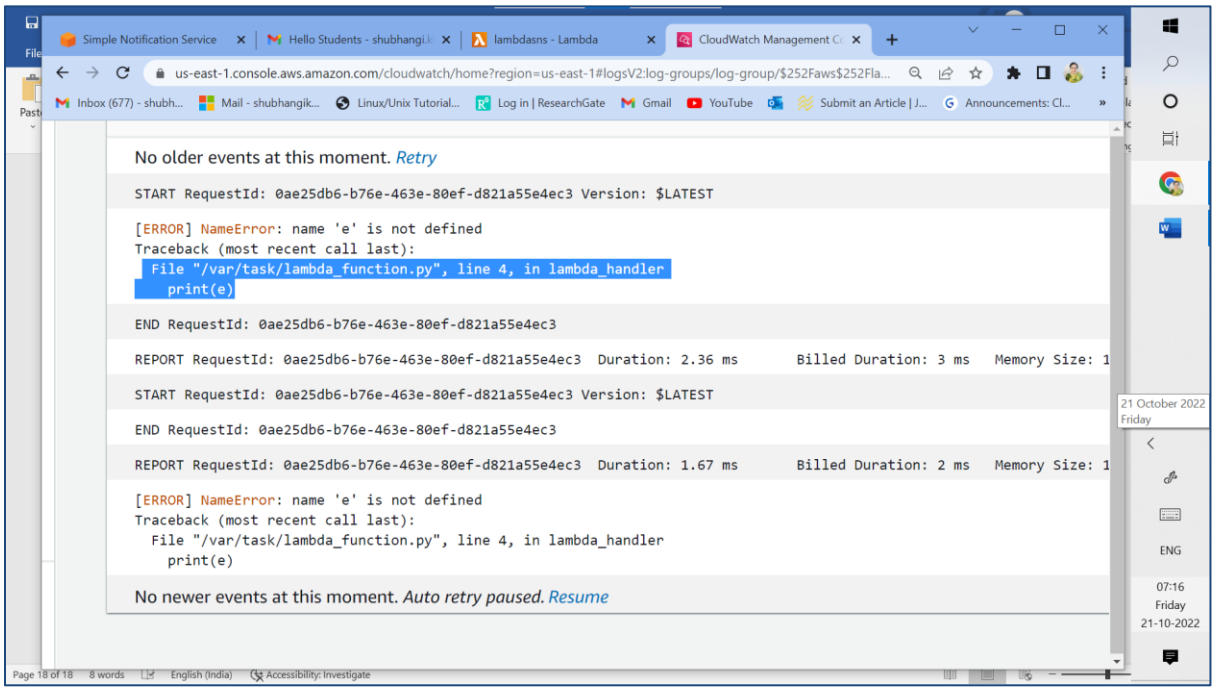
3q: Publish message (Hello, This message is for all students) to be sent to both email and AWS lambda endpoints (No filtering)



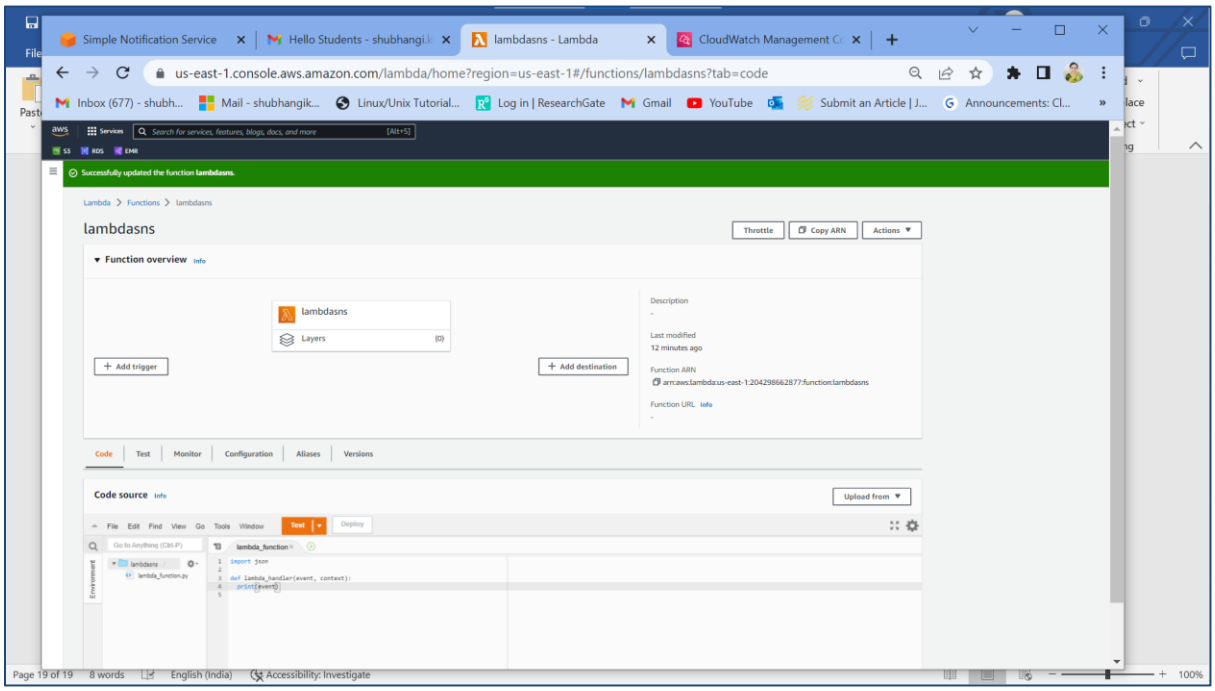
3r: Message published successfully



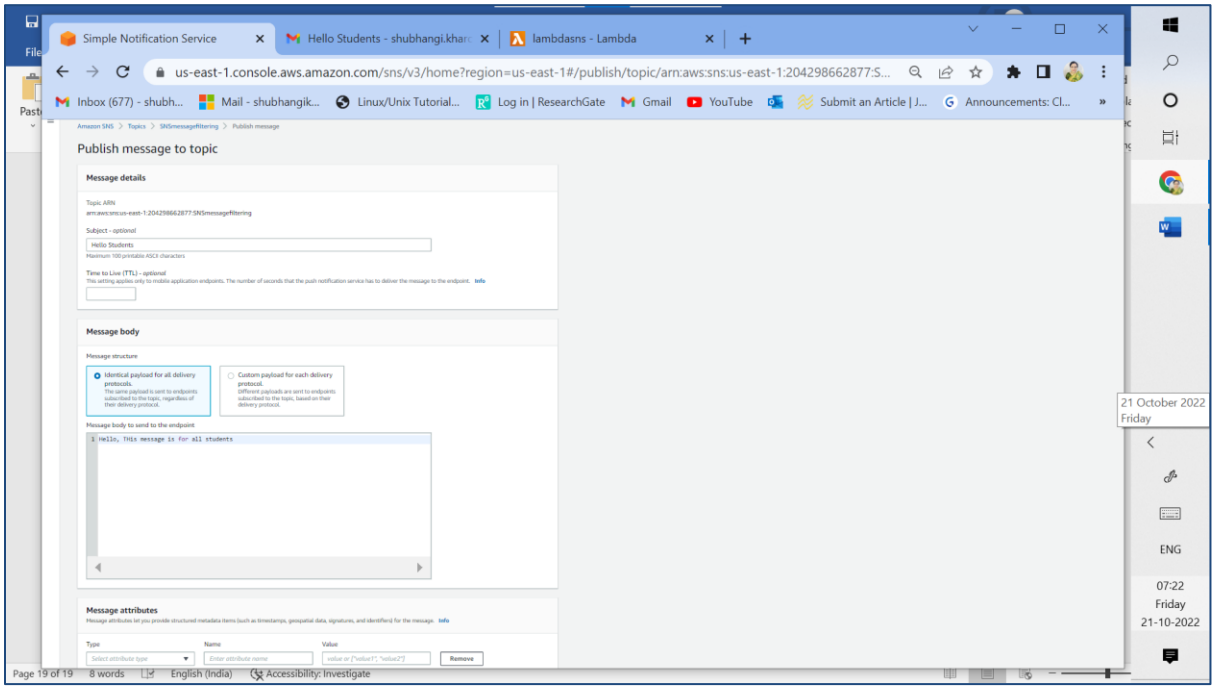
3s:Successfully received message via email



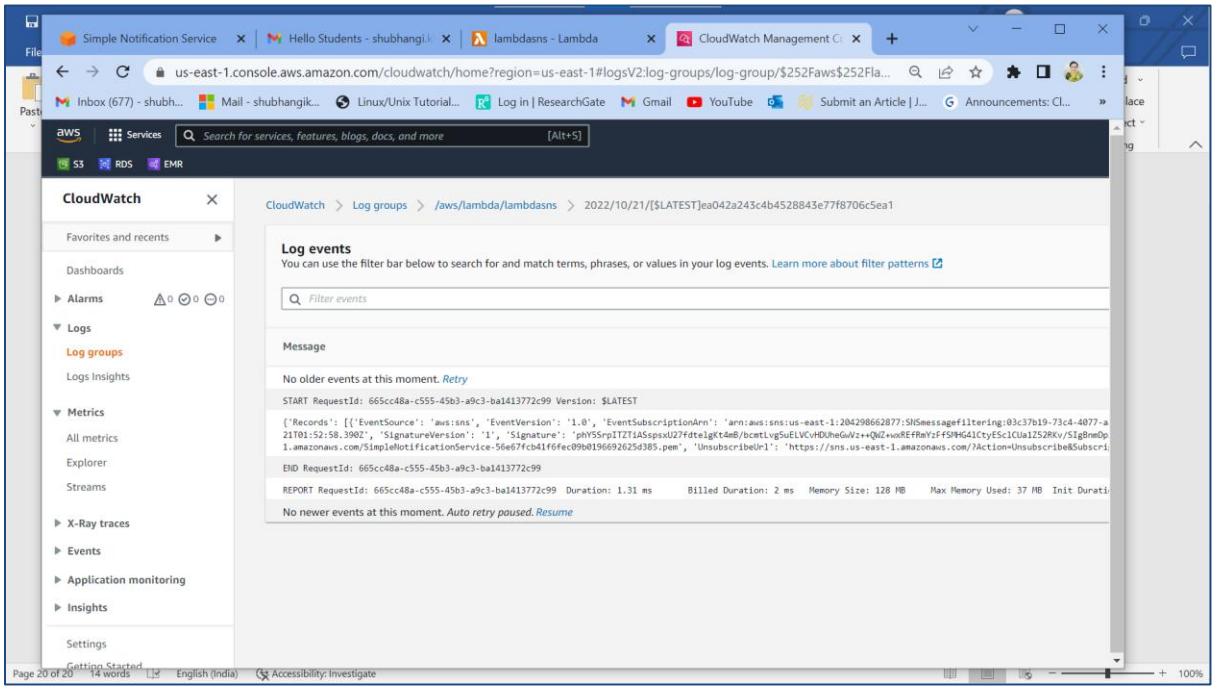
3t: Check message for lambda function using cloudwatch monitoring. As seen, Message not received in lambda function due to print error in JSON

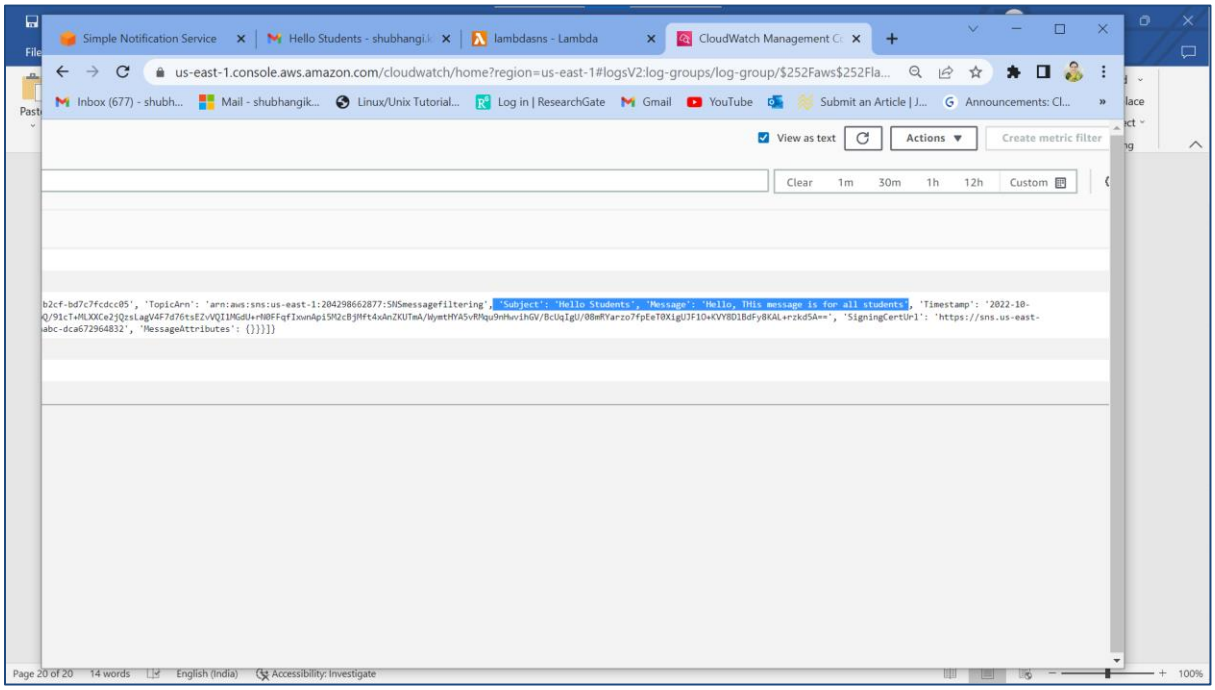


3u: Successfully corrected print error in lambda function JSON script

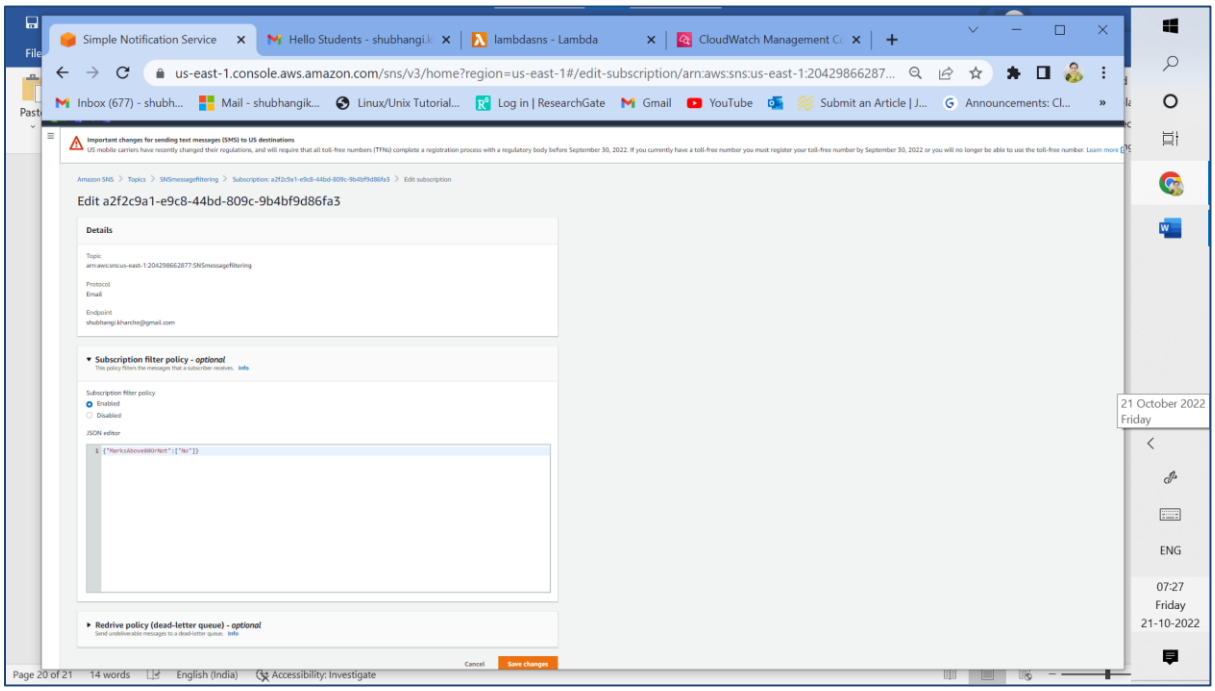


3v: After removing error again publish message

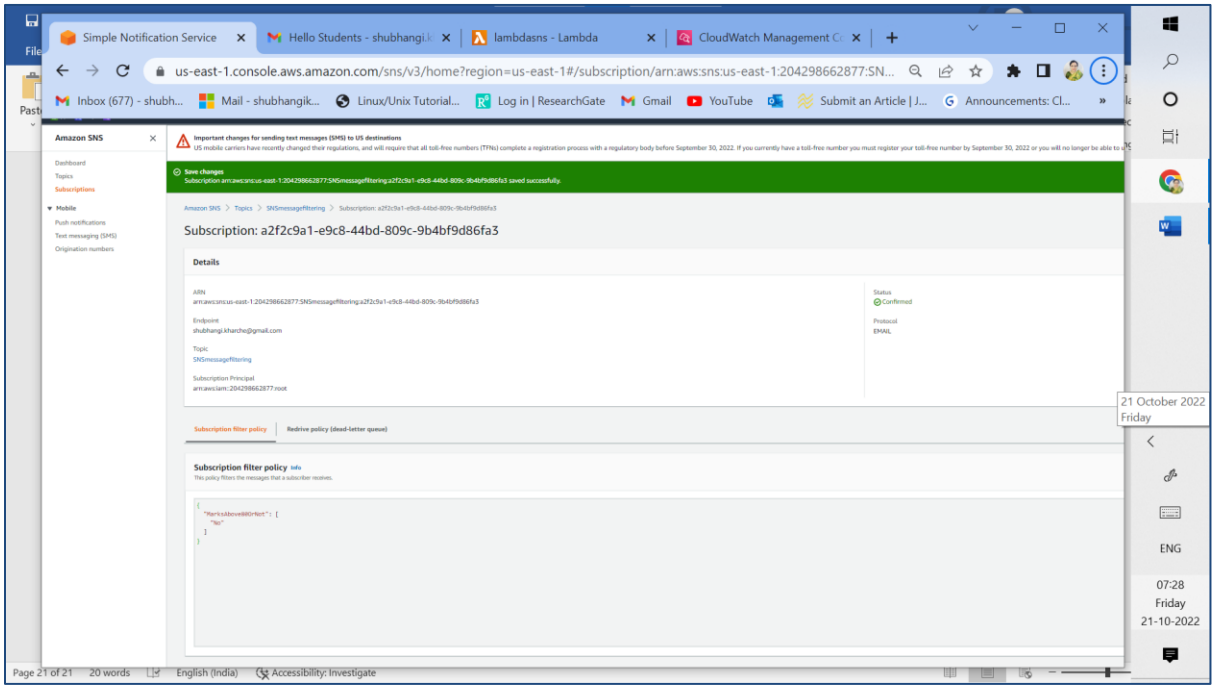




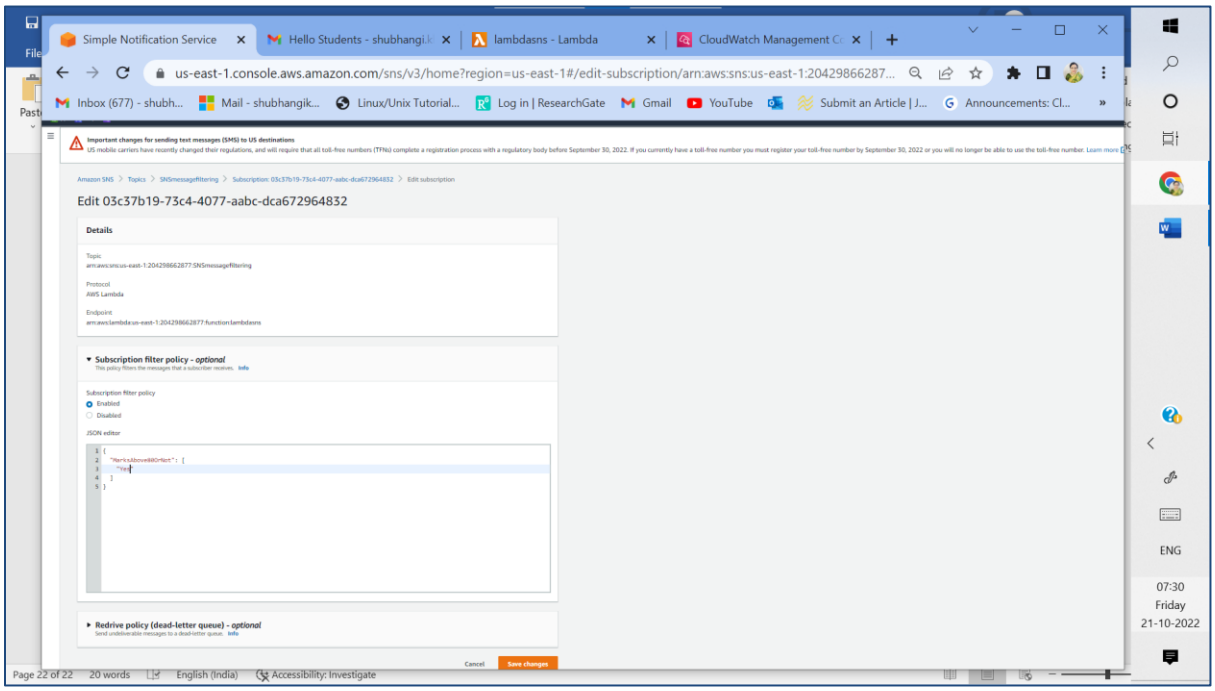
3w: Monitor cloudwatch events for lambda function,message received successfully



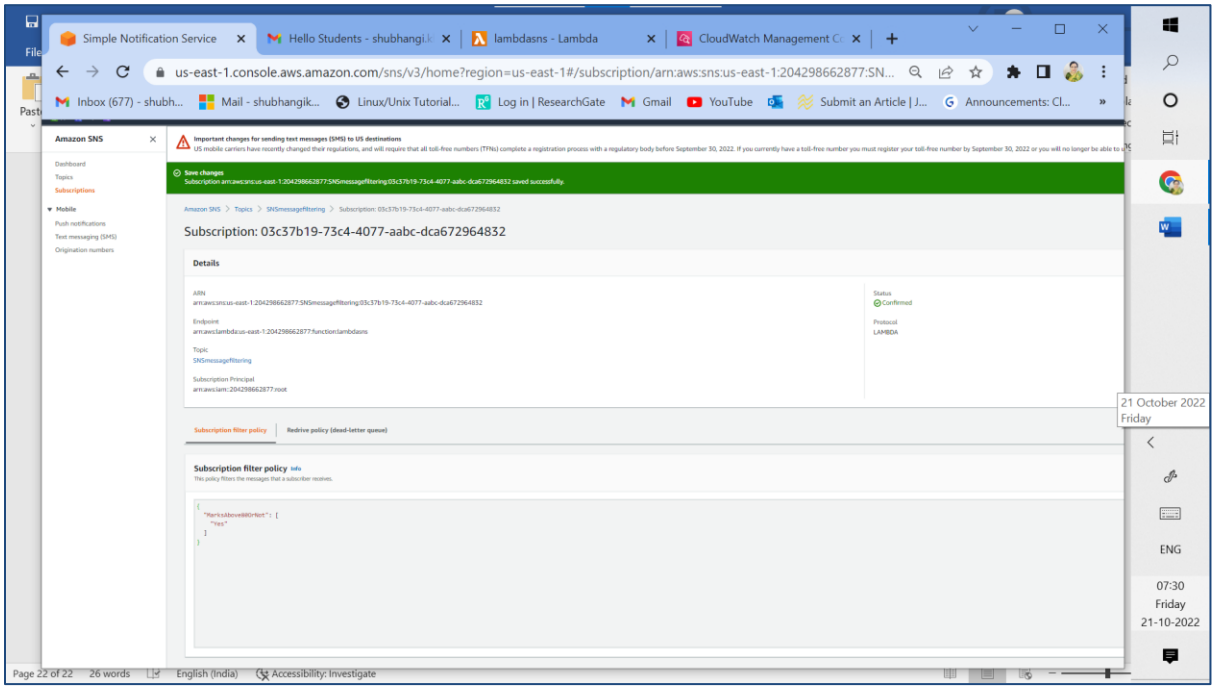
3x: Edit subscription filter policy for email endpoint. Email will be sent only when marks are below 80.



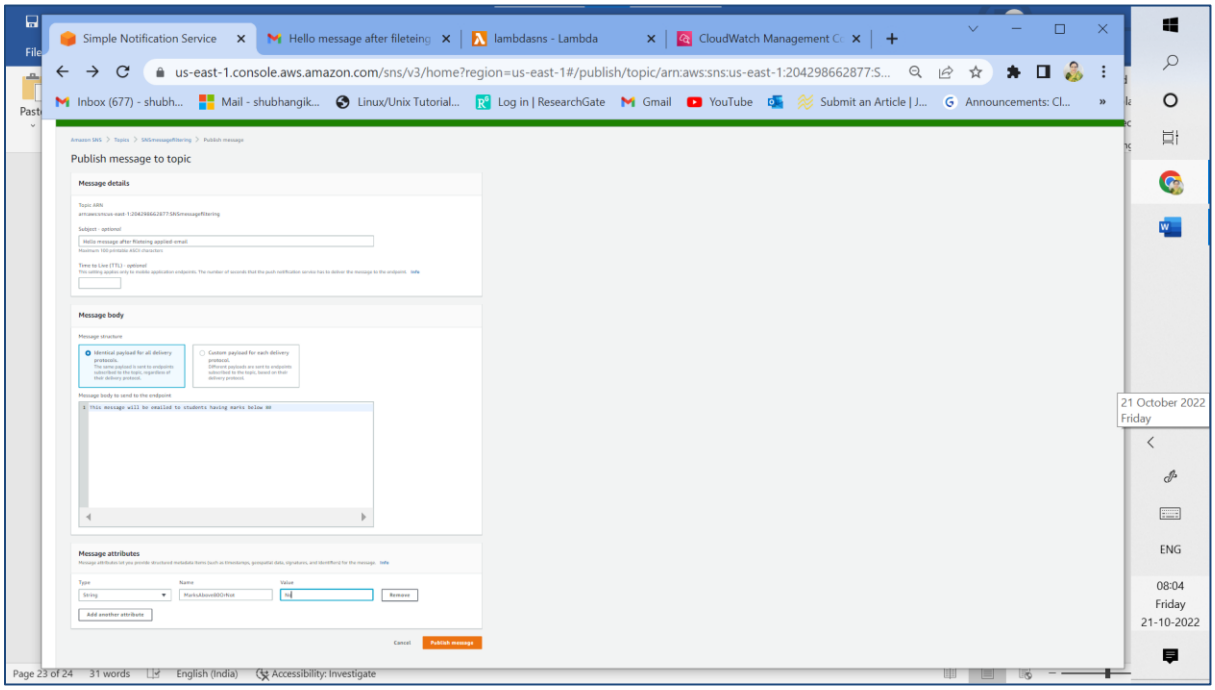
3y: Status confirmed for subscription filter policy (email endpoint)



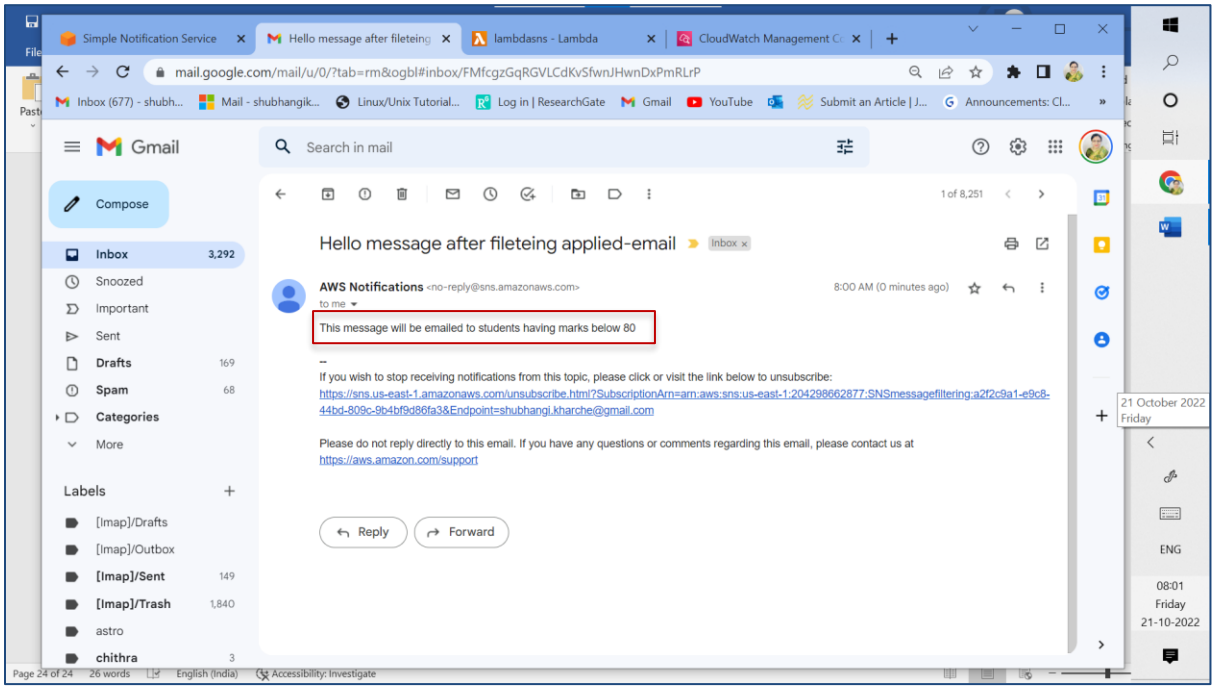
3z: Edit subscription filter policy for AWS lambda endpoint. Lambda function will be triggered only when marks are above 80



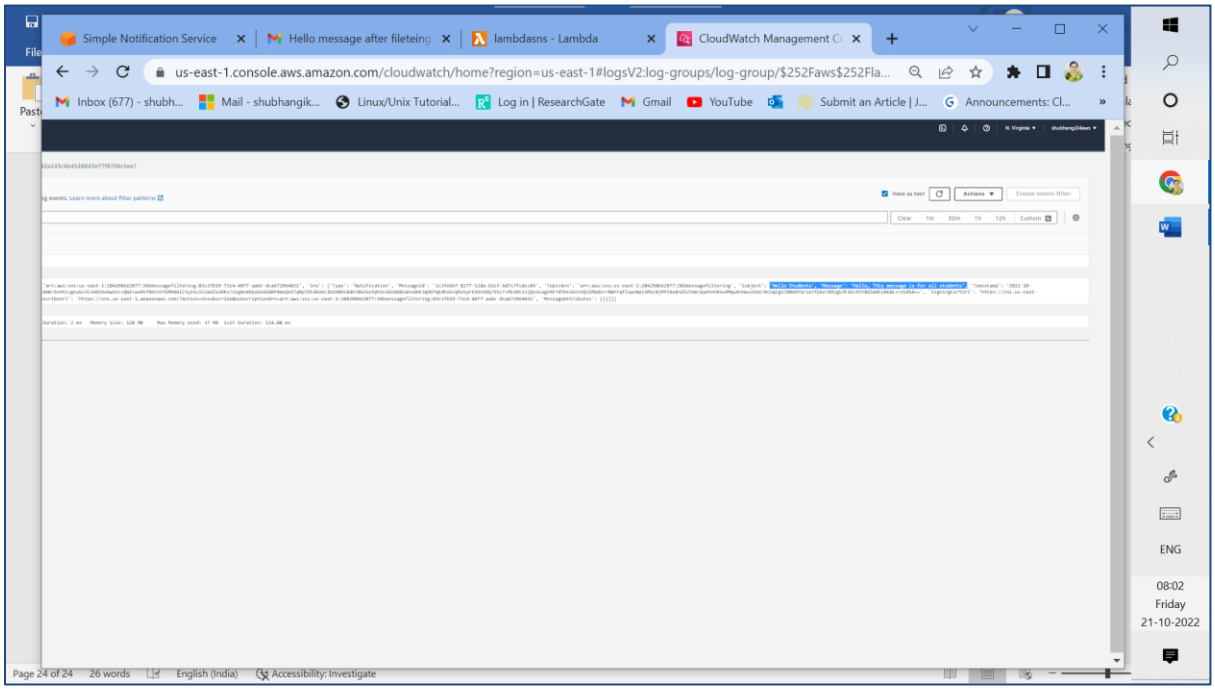
3za: Status confirmed for subscription filter policy (AWS lambda endpoint)



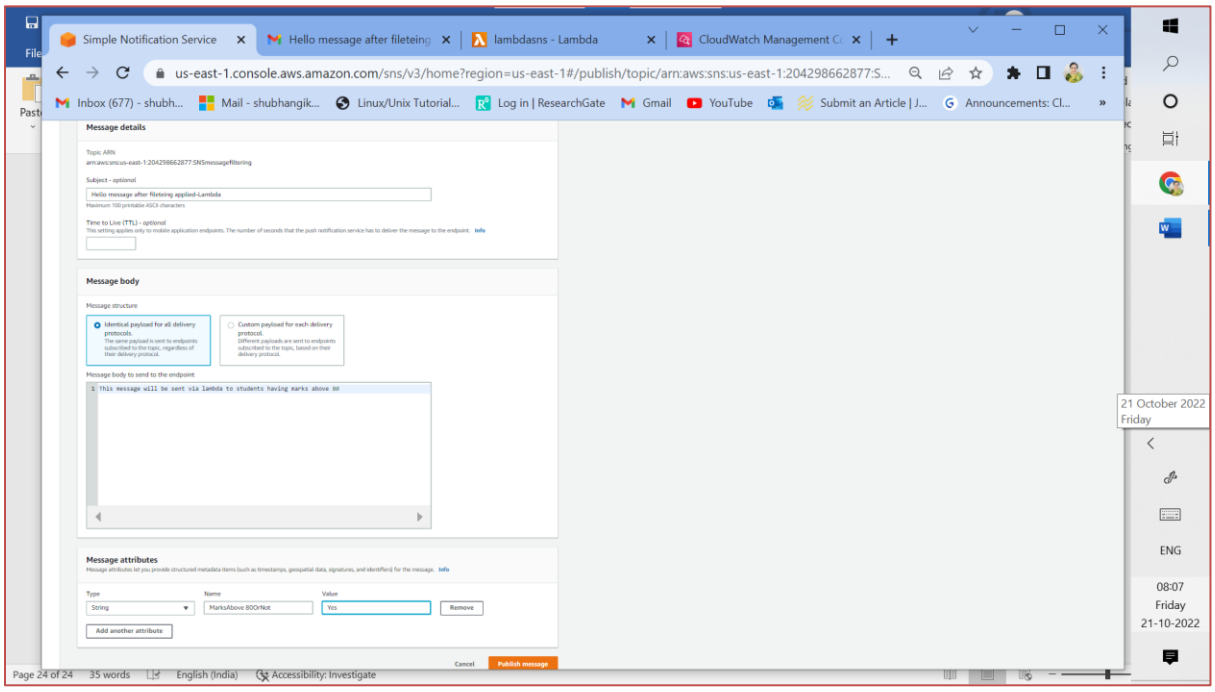
3zb: Publish message for email with attributes as seen



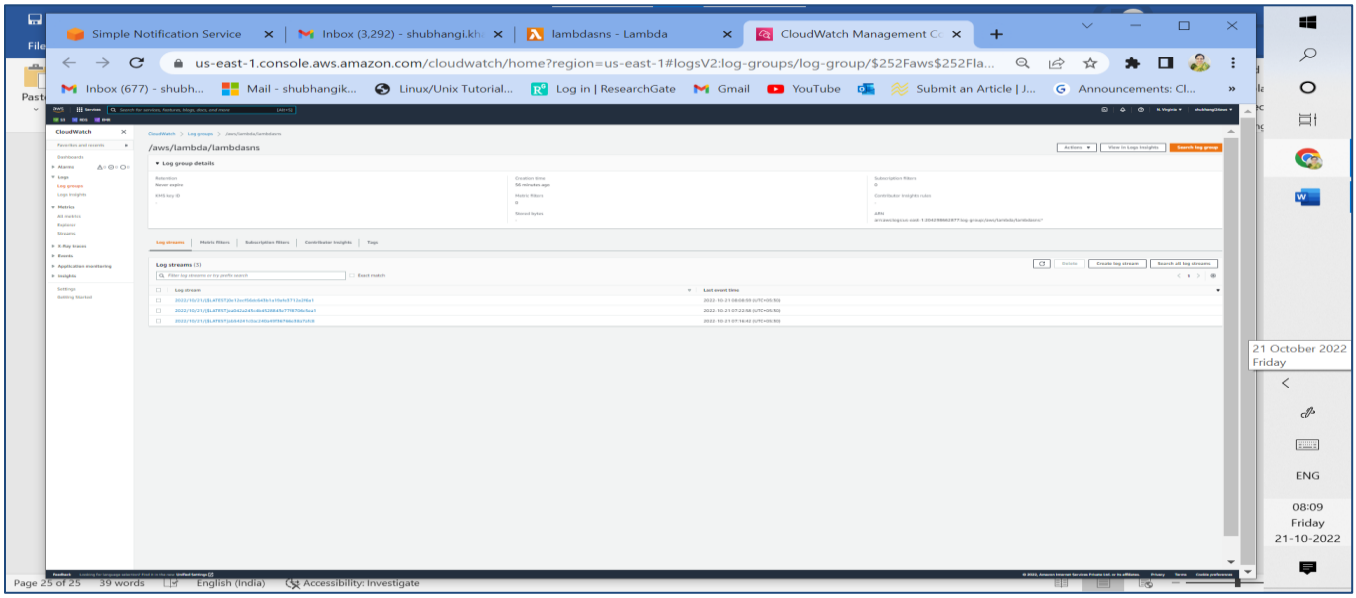
3zc: Successfully received appropriate message via email after applying SNS filtering



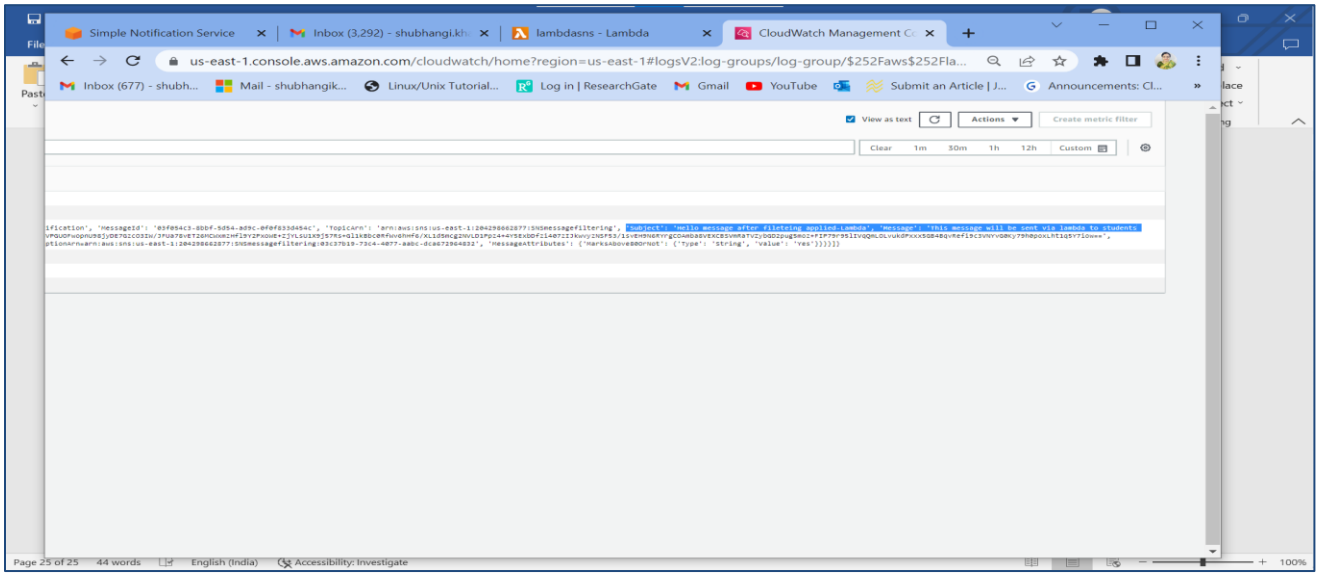
3zd: Message in AWS lambda is unchanged (as expected) after applying SNS message filtering



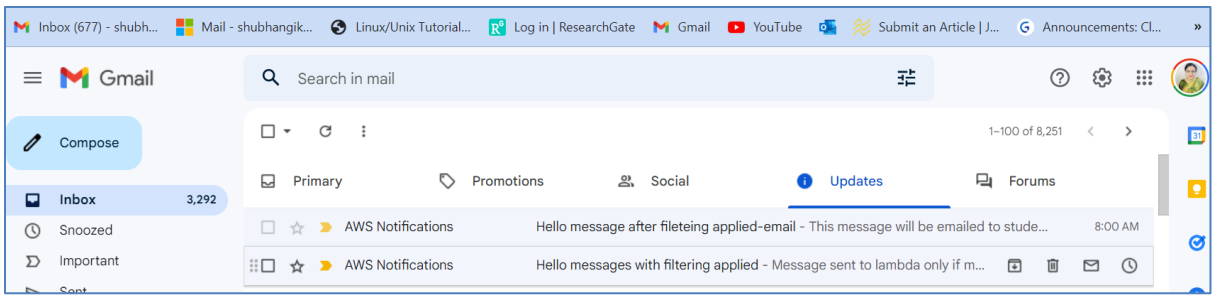
3ze: Publish message for AWS Lambda endpoint with attributes as seen. Lambda function will be triggered only when marks are above 80;



3zf: New log visible for Lambda



3zg: Subject and message updated for lambda as expected after SNS filtering is applied



3zh: New email message not received as expected after SNS filtering is applied.

C. Cost Analysis of the implemented Solution –Assume your solution is used by 1000 users for a month, and give monthly billing estimates.

Sr. No	Region	Services	Upfront	monthly	For 1000 users	First 12 months	currency	Configuration summary	Calculations
1	US East (N virginia)	Amazon SNS	0.00	0.00	0.00	0.00	USD	Standard topic, 1000 Email/Email JSON notifications, 1000 SQS notifications, 1000 lambda requests (all per month)	1,000 requests - 1000000 free tier requests = -999,000.00 billable SNS requests per month Max (-999000.000000 requests, 0 requests) = 0.00 requests Tiered price for: 1000 calls 1000 calls x 0.0000000000 USD = 0.00 USD Total tier cost = 0.00 USD (EMAIL/EMAIL-JSON Notifications cost) 1,000 notifications x 0.00 USD = 0.00 USD (SQS Notifications cost) 1,000 deliveries x 0.00 USD = 0.00 USD (Amazon Web Services Lambda cost) SNS Requests and Notifications cost (monthly): 0.00 USD
2	US East (N virginia)	Amazon SQS	0.00	0.00	0.00	0.00	USD	Standard queue (1000 requests per month), all data transfer in	First 1 Million Requests/Month is Free for std queue, data transfer in at $0.00 per GB
3	US East (N virginia)	AWS Lambda	0.00	0.00	0.00	0.00	USD	Lambda function with 1000 requests per month	Data transfer cost between SNS and lambda in same region is 0.00 USD
4	US East (N virginia)	Amazon cloudwatch	0.00	0.00	0.00	6.00	USD	1000 user logs will require < 1GB	1 GB x 0.50 USD = 0.50 USD Standard Logs Data Ingested cost: 0.50 USD CloudWatch Logs Data Ingested tiered pricing cost: 0 USD 1.00 GB per month x 0.15 Storage compression factor x 1 Logs retention factor x 0.03 USD = 0.0045 USD Standard/Vended Logs data storage cost: 0.0045 USD Logs delivered to S3 Data Ingested cost: 0 USD Logs converted to Apache Parquet format cost: 0 USD 0.50 USD + 0.0045 USD = 0.5045 USD. CloudWatch Logs Ingested and Storage cost (monthly): 0.50 USD
5	US East (N virginia)	IAM role	0.00	0.00	0.00	0.00	USD	IAM is offered at no additional charge.	IAM role for aws lambda to provide full access to SNS and cloudwatch
		Total	**0.00**	**0.00**	**0.00**	**6.00**			

D. Lessons & Observations

- SNS is an application service that is used to send identical messages to various endpoints viz; Amazon SQS, Amazon lambda, Amazon kinesis data firehose, email/email JSON, HTTP/HTTPs, mobile SMS.
- Learnt that SNS has message filtering capability: 1) filtering based on topics and 2) filtering based on conditions defined in JSON scripts.
  - SNS sets asynchronous communication with endpoints over Internet
  - SNS and SQS services are free up to million requests per month
  - Learnt to create IAM role for AWS lambda to have full access to SNS and cloudwatch.
- Learnt to read cloudwatch logs
  - Learnt to edit JSON scripts.
  - Learnt SNS and SQS as publish subscribe application services
  - Learnt to estimate cost for each service
  - Learnt that Amazon SNS supports IoT protocol like MQTT
  - AWS application services like SNS and SQS are used to send bulk messages from ecommerce websites, BFSI websites etc