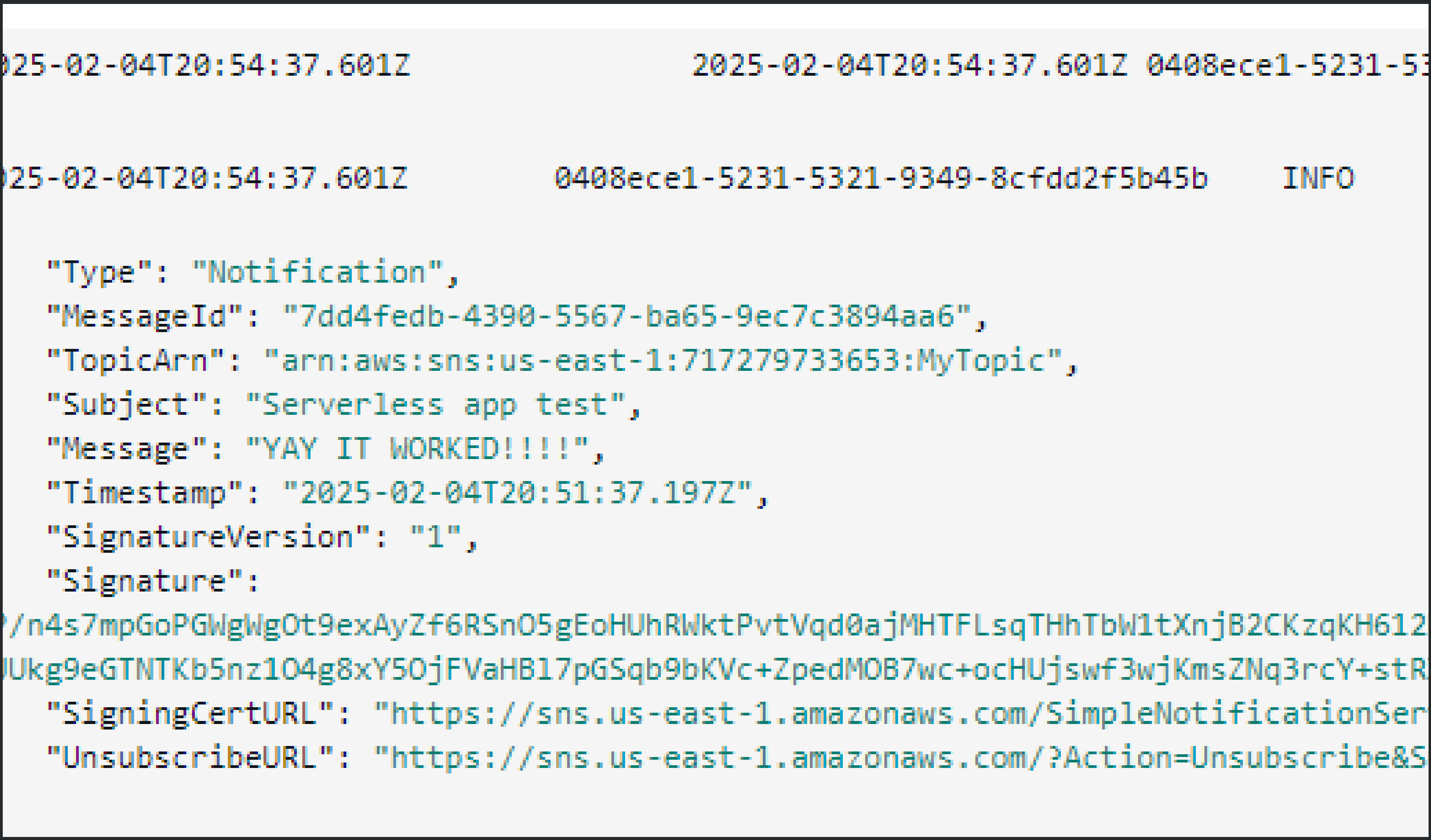
Kanika Mathur
github.com/KanikaGenesis

Serverless Event-Driven Application

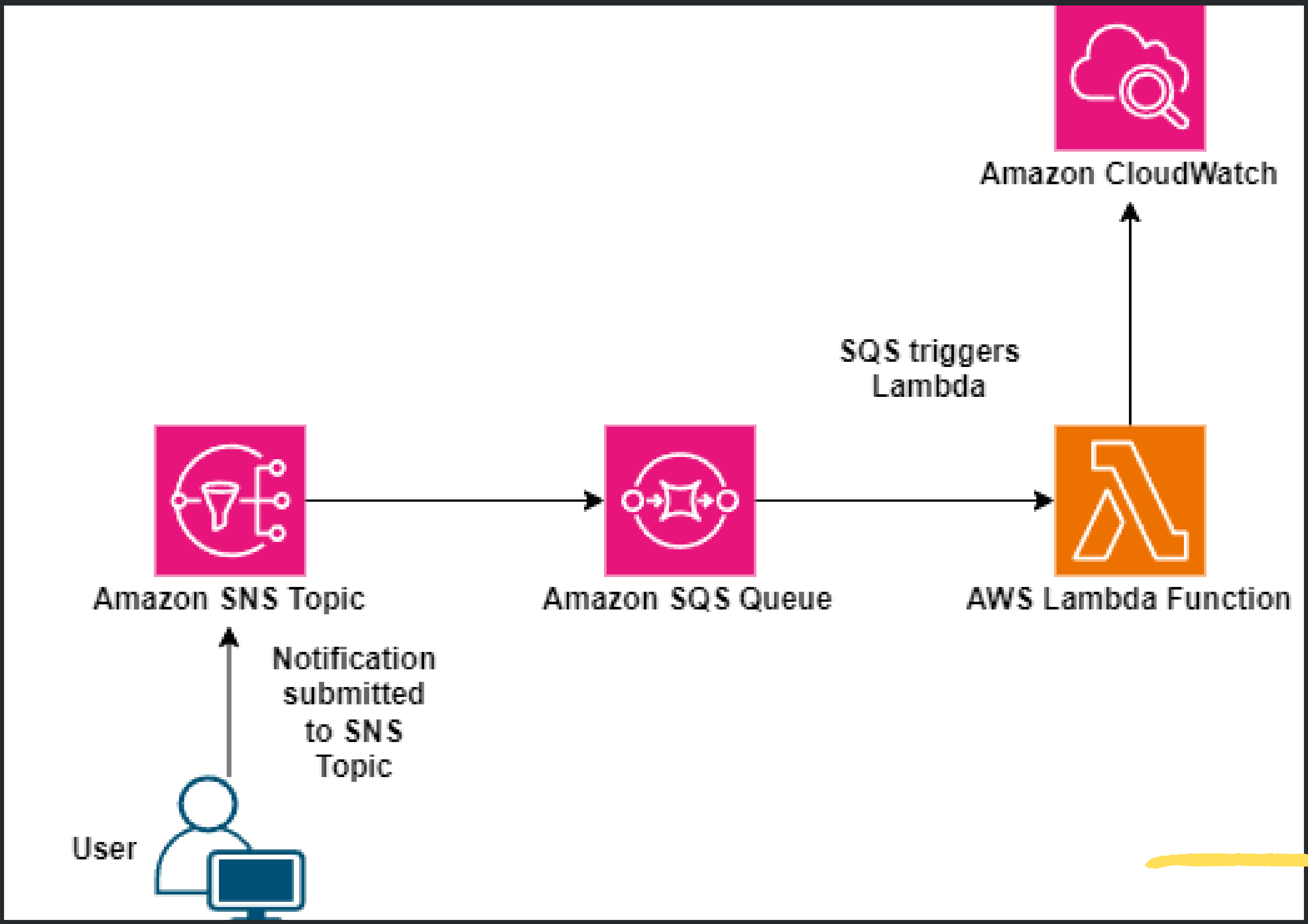


```
)25-02-04T20:54:37.601Z            2025-02-04T20:54:37.601Z 0408ece1-5231-53

)25-02-04T20:54:37.601Z       0408ece1-5231-5321-9349-8cfdd2f5b45b    INFO

   "Type": "Notification",
   "MessageId": "7dd4fedb-4390-5567-ba65-9ec7c3894aa6",
   "TopicArn": "arn:aws:sns:us-east-1:717279733653:MyTopic",
   "Subject": "Serverless app test",
   "Message": "YAY IT WORKED!!!!",
   "Timestamp": "2025-02-04T20:51:37.197Z",
   "SignatureVersion": "1",
   "Signature":
'/n4s7mpGoPGWgWgOt9exAyZf6RSnO5gEoHUhRWktPvtVqd0ajMHTFLsqTHhTbW1tXnjB2CKzqKH612
JUkg9eGTNTKb5nz104g8xY5OjFVaHB17pGSqb9bKVc+ZpedMOB7wc+ocHUjswf3wjKmsZNq3rcY+stR
   "SigningCertURL": "https://sns.us-east-1.amazonaws.com/SimpleNotificationSer
   "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&S
```

Kanika Mathur
github.com/KanikaGenesis

Project Overview

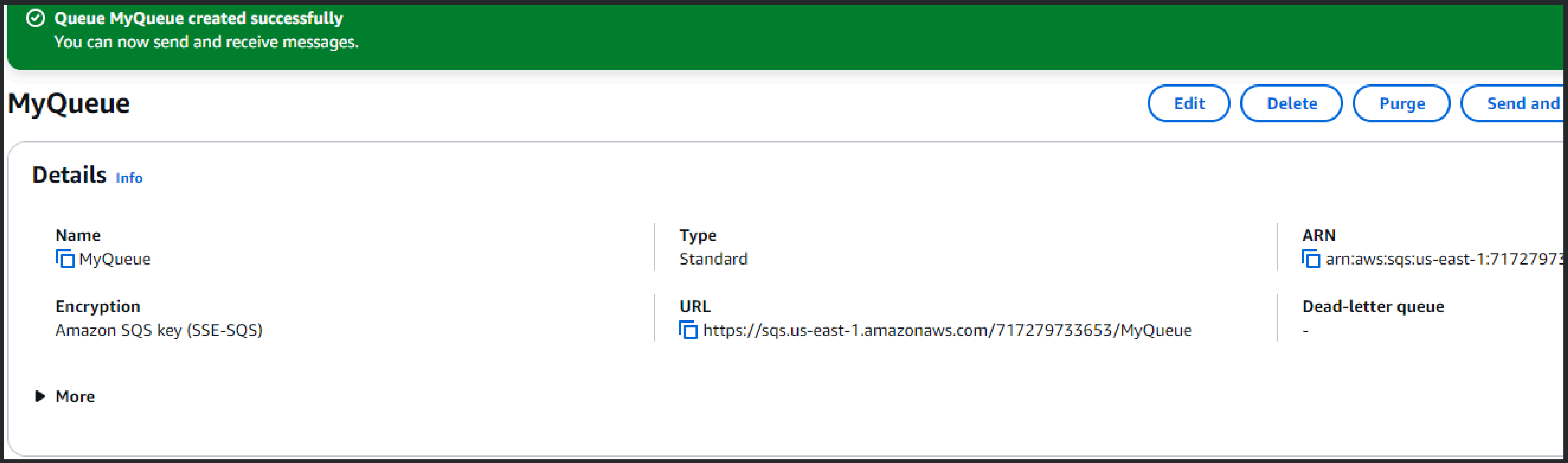
In this project, I created a system where users submitted notifications to an Amazon SNS topic. These notifications were then forwarded to an Amazon SQS queue, which acted as a buffer and stored messages until they were processed. The queue was set up to trigger an AWS Lambda function, which retrieved messages and wrote them into Amazon CloudWatch logs. This setup ensured that my system was event-driven and operated efficiently.



Kanika Mathur
github.com/KanikaGenesis

Creating an SQS Queue

I started by creating an SQS queue. Amazon Simple Queue Service (SQS) is a message queuing service that enables asynchronous communication between services. The queue temporarily held messages before they were processed by the Lambda function. SQS ensures reliable message delivery and decouples the producer (SNS) from the consumer (Lambda), making the system more resilient to failures.



Queue MyQueue created successfully
You can now send and receive messages.

MyQueue

Details Info

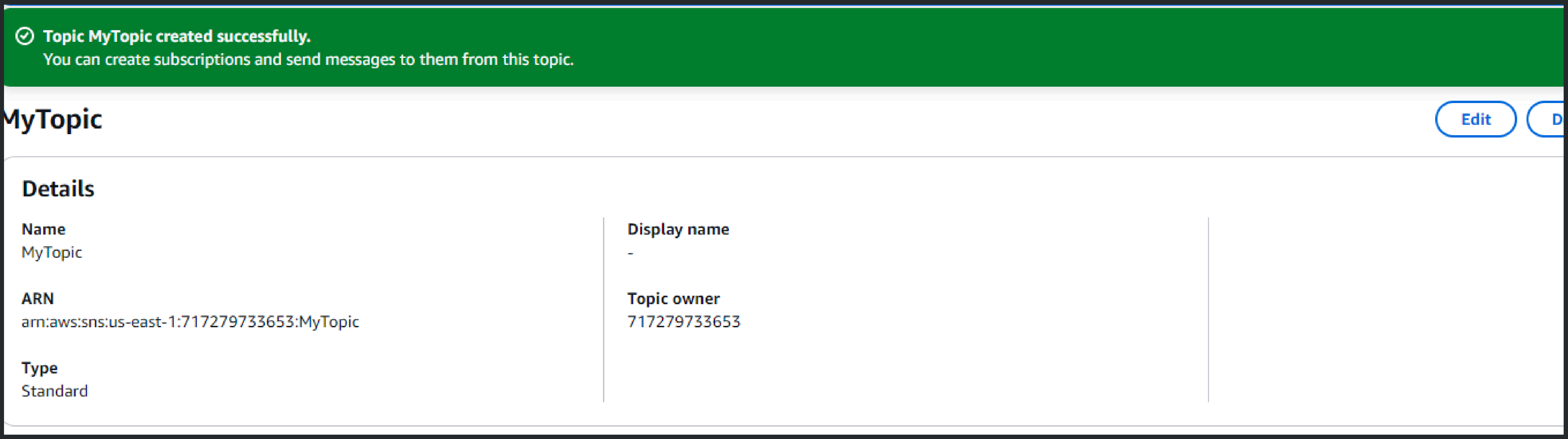
Name MyQueue	Type Standard	ARN arn:aws:sqs:us-east-1:71727973
Encryption Amazon SQS key (SSE-SQS)	URL https://sqs.us-east-1.amazonaws.com/717279733653/MyQueue	Dead-letter queue -

▶ More

Kanika Mathur
github.com/KanikaGenesis

Creating an SNS Topic

Next, I created an Amazon Simple Notification Service (SNS) topic to distribute messages to multiple endpoints. By doing this, I ensured that my messages could be sent to various subscribers, including the SQS queue I had just created. SNS acts as a publisher-subscriber model, allowing multiple services to receive the same message efficiently.



Topic MyTopic created successfully.
You can create subscriptions and send messages to them from this topic.

MyTopic

Details

Name
MyTopic

ARN
arn:aws:sns:us-east-1:717279733653:MyTopic

Type
Standard

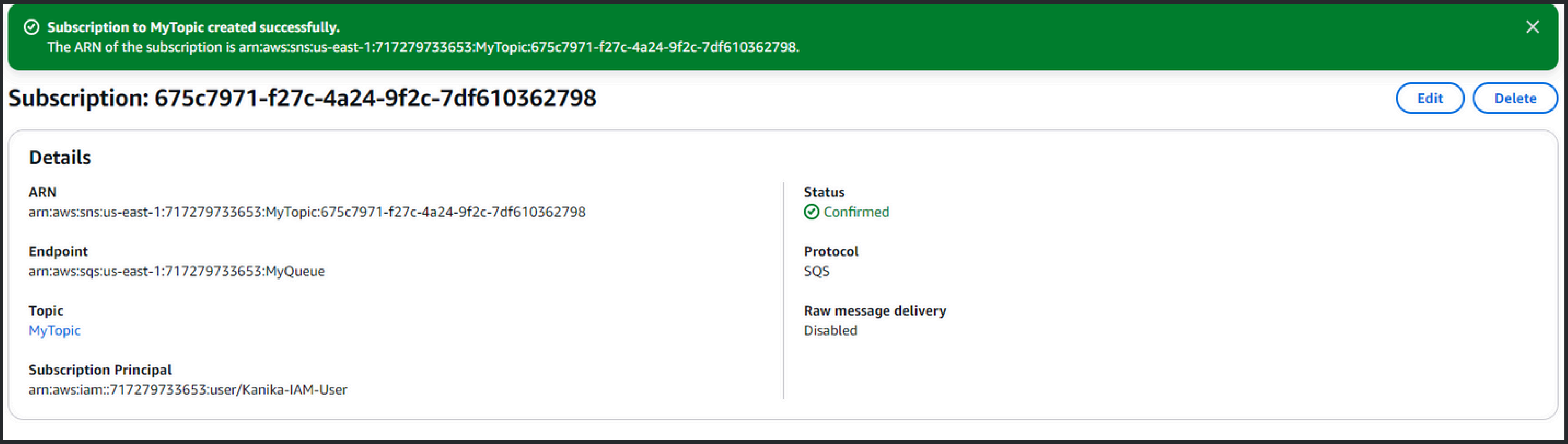
Display name
-

Topic owner
717279733653

Kanika Mathur
github.com/KanikaGenesis

Creating an SNS Subscription to SQS

After setting up the SNS topic, I subscribed my SQS queue to it. This ensured that any message published to SNS was automatically delivered to the queue. By setting up this subscription, I ensured event-driven messaging, where messages are automatically relayed without the need for manual intervention.



Subscription to MyTopic created successfully.
The ARN of the subscription is arn:aws:sns:us-east-1:717279733653:MyTopic:675c7971-f27c-4a24-9f2c-7df610362798.

Subscription: 675c7971-f27c-4a24-9f2c-7df610362798

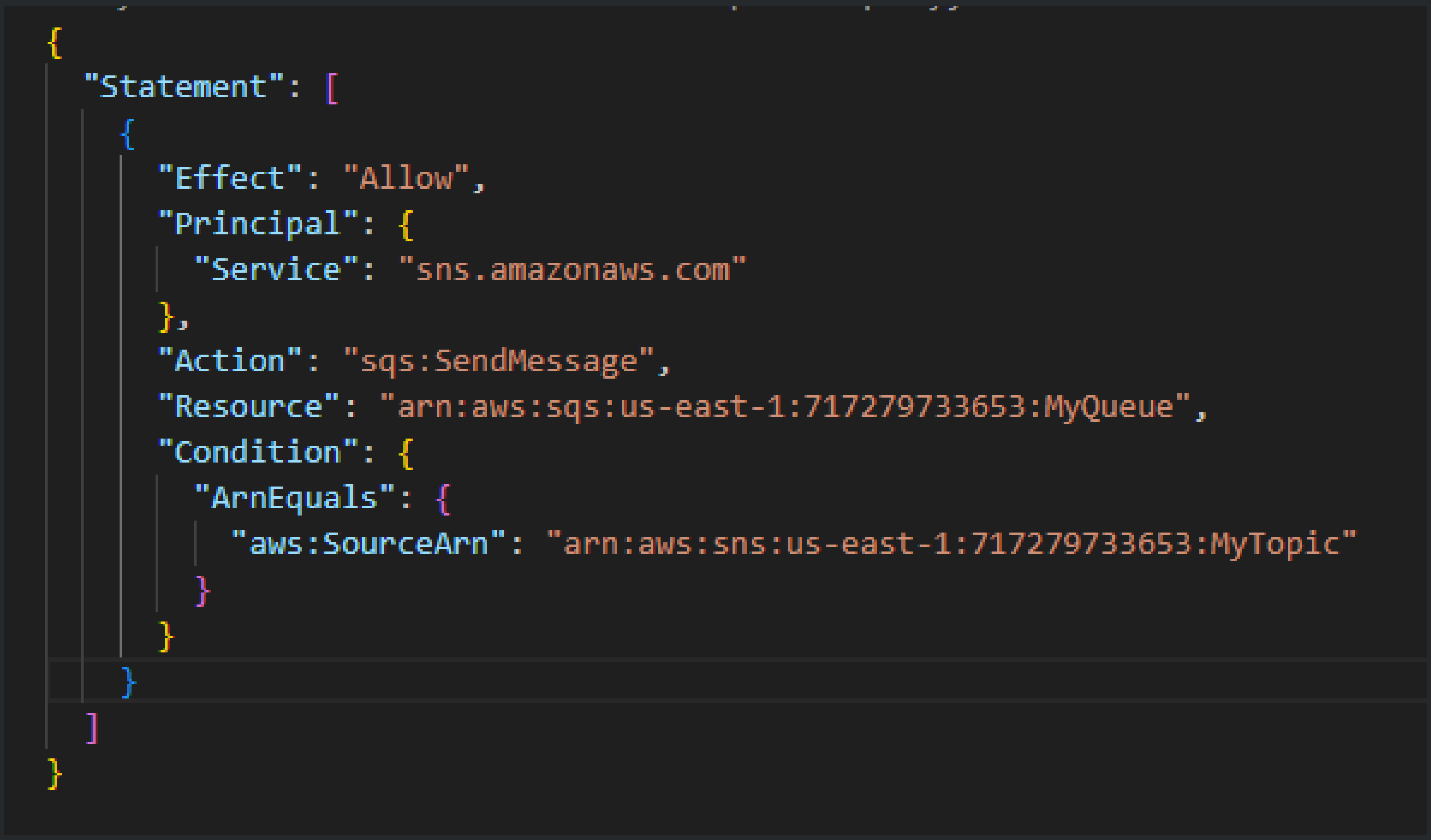
Details

ARN arn:aws:sns:us-east-1:717279733653:MyTopic:675c7971-f27c-4a24-9f2c-7df610362798	**Status** Confirmed
Endpoint arn:aws:sqs:us-east-1:717279733653:MyQueue	**Protocol** SQS
Topic MyTopic	**Raw message delivery** Disabled
Subscription Principal arn:aws:iam::717279733653:user/Kanika-IAM-User	

Kanika Mathur
github.com/KanikaGenesis

Granting SNS Permissions to SQS

By default, SNS does not have permission to send messages to SQS, so I explicitly granted these permissions by updating the queue's access policy. This step was necessary to allow SNS to publish messages directly to SQS without encountering permission errors.

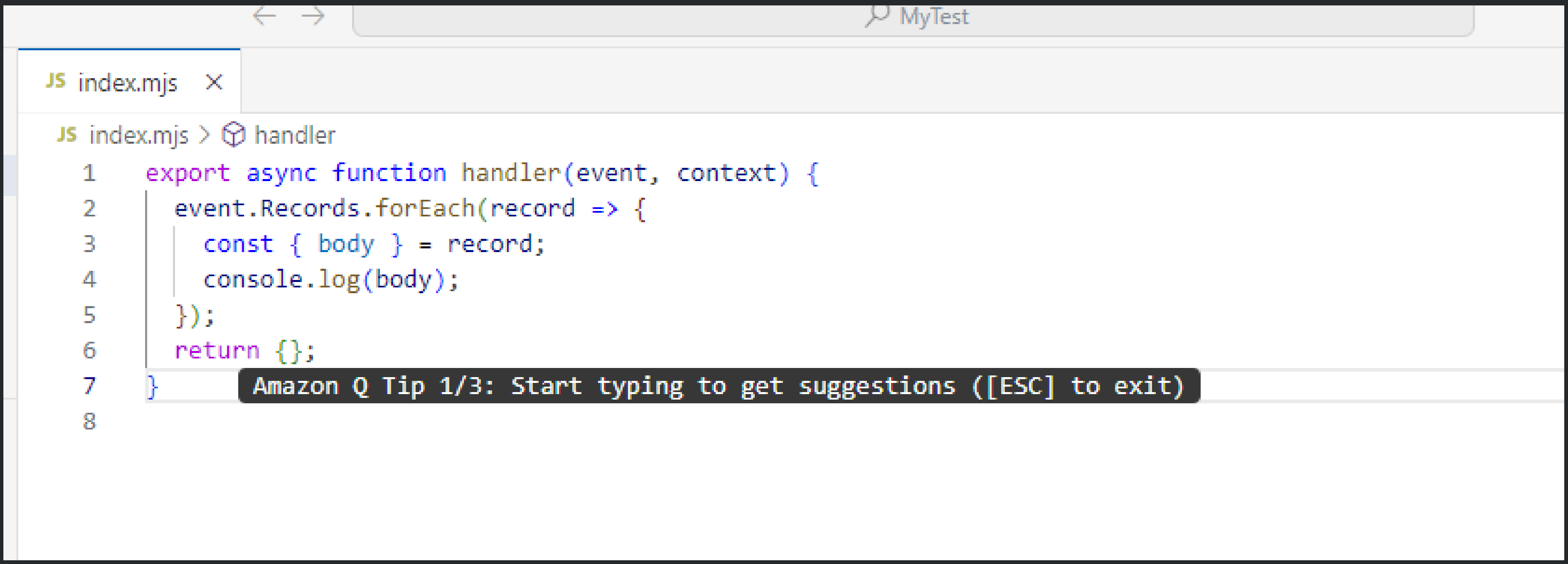


```json
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:717279733653:MyQueue",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:717279733653:MyTopic"
        }
      }
    }
  ]
}
```

Kanika Mathur
github.com/KanikaGenesis

Setting Up an AWS Lambda Function

I then created an AWS Lambda function to process messages from the SQS queue and log them to CloudWatch. AWS Lambda allows serverless execution of code in response to events, ensuring cost efficiency by only running when triggered.

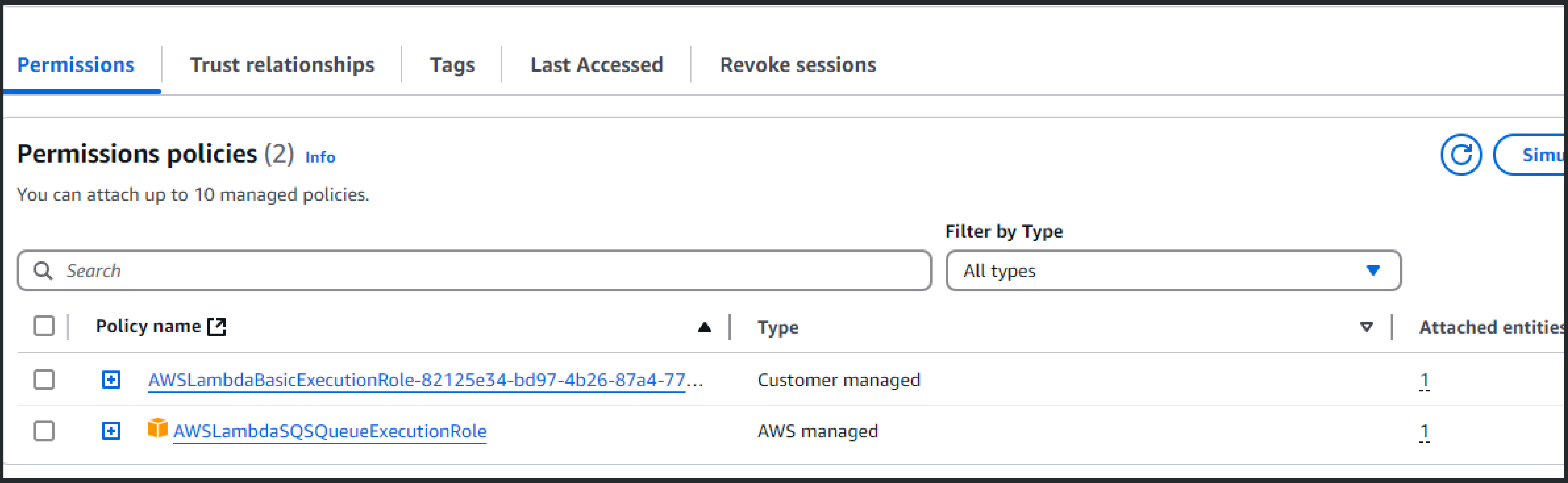


```
export async function handler(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}      Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
```

Kanika Mathur
github.com/KanikaGenesis

Granting Lambda Permissions to Read from SQS

To enable Lambda to read and delete messages from SQS, I assigned the necessary IAM permissions. Without these permissions, Lambda would not be able to retrieve messages from the queue, causing delays or failures in message processing.



Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (2) Info

You can attach up to 10 managed policies.

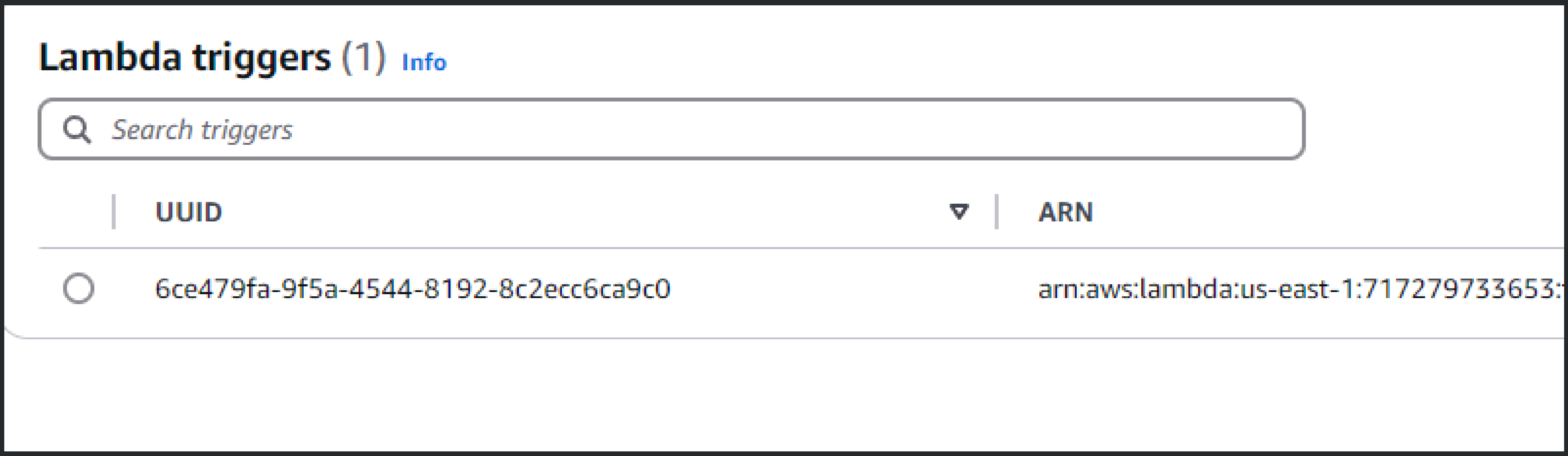
Filter by Type: All types

Policy name	Type	Attached entities
AWSLambdaBasicExecutionRole-82125e34-bd97-4b26-87a4-77...	Customer managed	1
AWSLambdaSQSQueueExecutionRole	AWS managed	1

Kanika Mathur
github.com/KanikaGenesis

Configuring SQS to Trigger Lambda

Next, I configured the SQS queue to automatically invoke my Lambda function whenever a new message arrived. This setup allowed the system to process messages in real-time.



Lambda triggers (1) Info

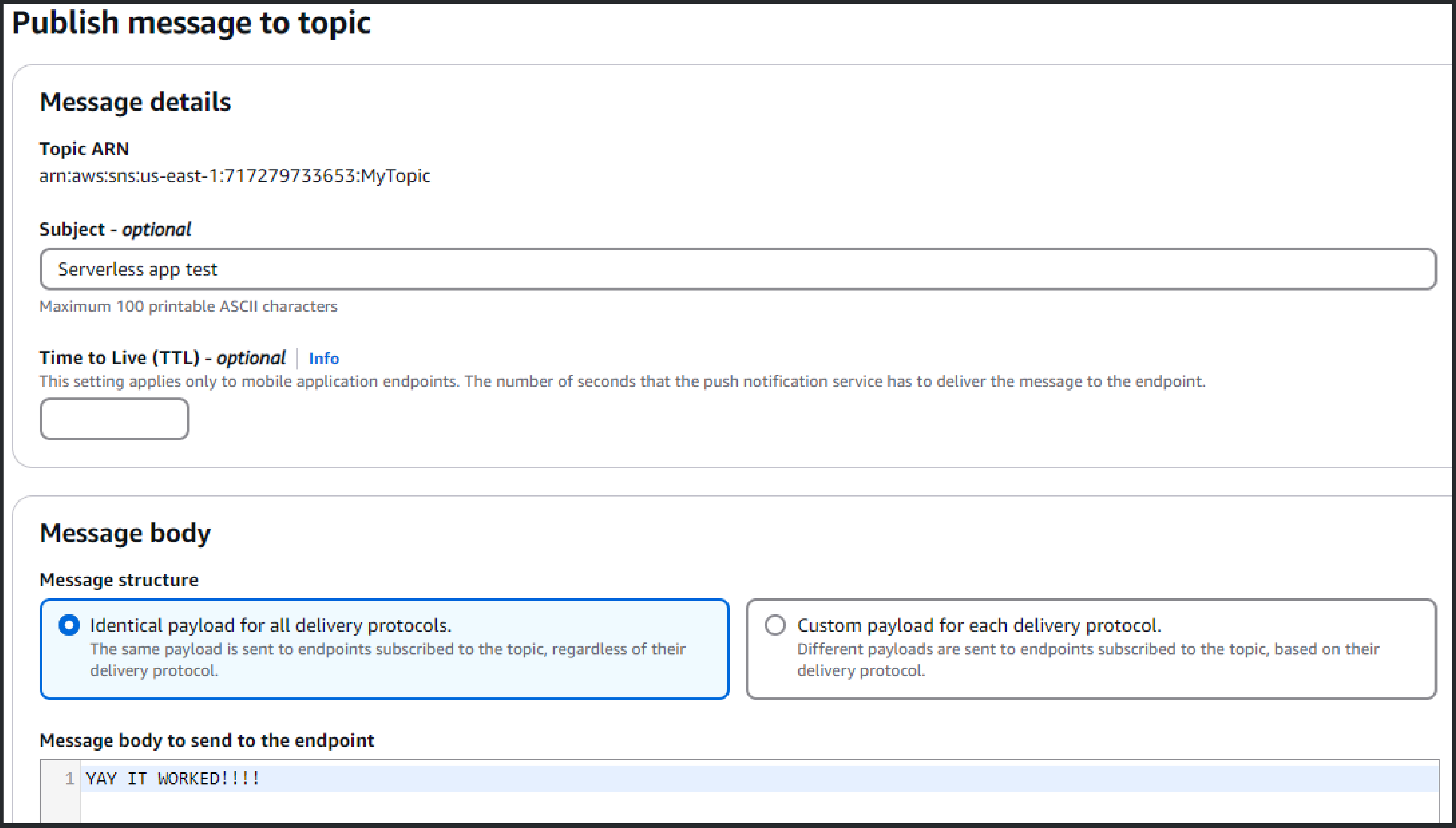
Search triggers

	UUID	ARN
○	6ce479fa-9f5a-4544-8192-8c2ecc6ca9c0	arn:aws:lambda:us-east-1:717279733653:

Kanika Mathur
github.com/KanikaGenesis

Testing the System

Once everything was set up, I needed to test whether the workflow functioned as expected. I tested the system by publishing a message to SNS and verifying that it appeared in CloudWatch.



Publish message to topic

Message details

Topic ARN
arn:aws:sns:us-east-1:717279733653:MyTopic

Subject - *optional*
Serverless app test
Maximum 100 printable ASCII characters

Time to Live (TTL) - *optional* | Info
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint.

Message body

Message structure

[✓ Identical payload for all delivery protocols.
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.]

Custom payload for each delivery protocol.
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

```
YAY IT WORKED!!!!
```

Kanika Mathur
github.com/KanikaGenesis

Thank You