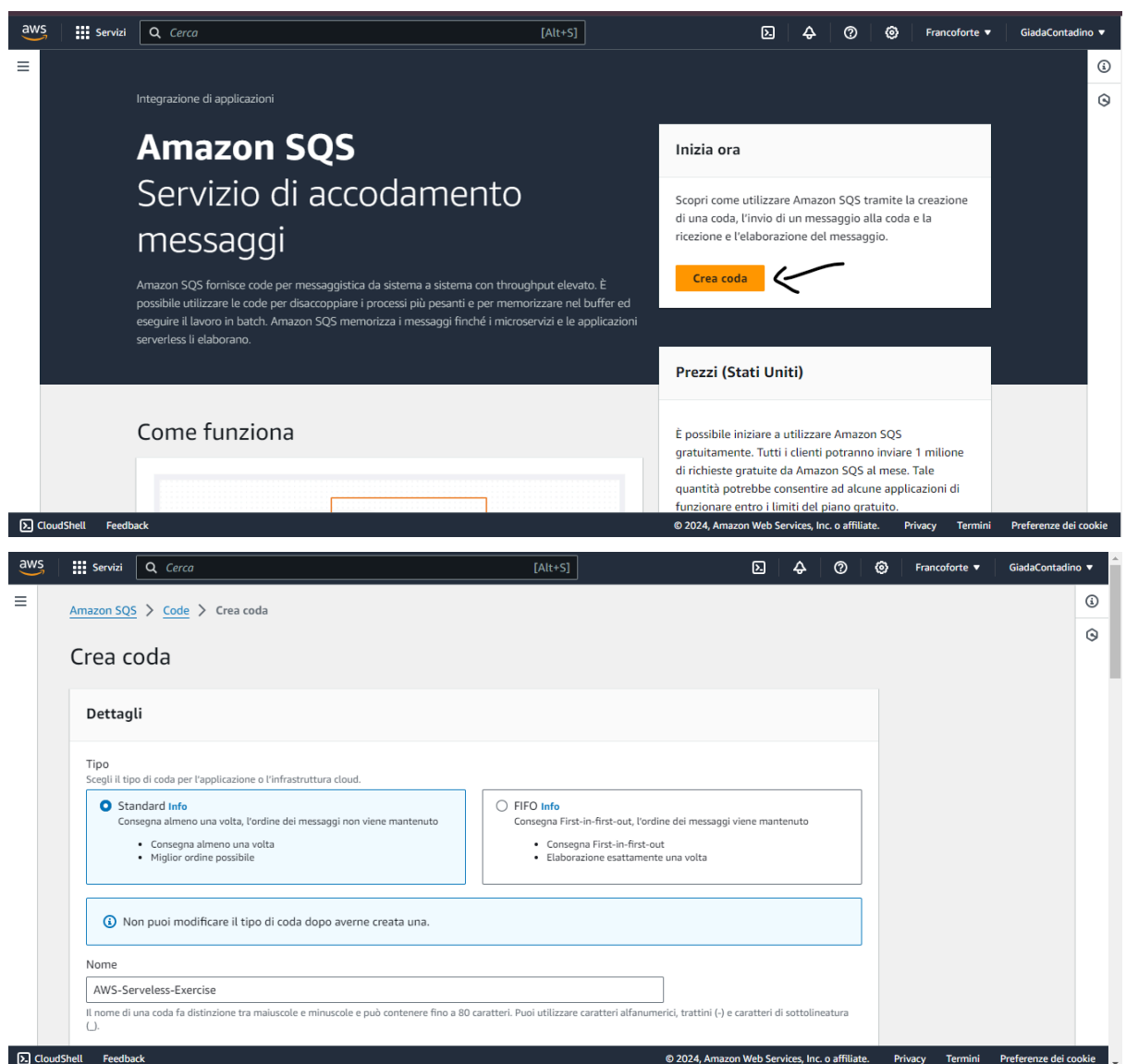


# Progetto AWS Serverless Exercise – Lambda, SQS and DynamoDB

## 1) Primo step: Creare la coda SQS

Come primo passo, ho effettuato l'accesso al mio account personale e creato la coda SQS attraverso la quale passeranno i messaggi destinati alla tabella DynamoDB. Ho configurato una coda standard con le impostazioni predefinite, come illustrato negli screenshot seguenti.



aws

Servizi

Cerca

[Alt+S]

Francoforte

GiadaContadino

Configurazione

Info

Imposta la dimensione massima del messaggio, la visibilità per altri consumatori e la conservazione dei messaggi.

Timeout visibilità

Info

30

Secondi

Deve essere compreso tra 0 secondi e 12 ore.

Periodo di conservazione dei messaggi

Info

7

Giorni

Deve essere compreso tra 1 minuto e 14 giorni.

Ritardo nella consegna

Info

0

Secondi

Deve essere compreso tra 0 secondi e 15 minuti.

Dimensione massima dei messaggi

Info

256

KB

Deve essere compreso tra 1 KB e 256 KB.

Tempo di attesa per la ricezione dei messaggi

Info

0

Secondi

Deve essere compreso tra 0 e 20 secondi.

Crittografia

Info

Amazon SQS fornisce la crittografia dei dati in transito per impostazione predefinita. Per aggiungere la crittografia dei dati inattivi alla coda, abilita la crittografia lato server.

CloudShell

Feedback

© 2024, Amazon Web Services, Inc. o affiliate.

Privacy

Termini

Preferenze dei cookie

aws

Servizi

Cerca

[Alt+S]

Francoforte

GiadaContadino

0

Secondi

Deve essere compreso tra 0 e 20 secondi.

Crittografia

Info

Amazon SQS fornisce la crittografia dei dati in transito per impostazione predefinita. Per aggiungere la crittografia dei dati inattivi alla coda, abilita la crittografia lato server.

Crittografia lato server

☐ Disabilitato

☒ Abilitato

Encryption key type

☒ Chiave Amazon SQS (SSE-SQS)

Una chiave di crittografia che Amazon SQS crea, gestisce e utilizza per te.

☐ Chiave AWS Key Management Service (SSE-KMS)

Una chiave di crittografia protetta da AWS Key Management Service (AWS KMS).

Policy di accesso

Info

Definisci chi può accedere alla coda.

Scegli metodo

CloudShell

Feedback

© 2024, Amazon Web Services, Inc. o affiliate.

Privacy

Termini

Preferenze dei cookie

aws

Servizi

Cerca

[Alt+S]

Francoforte

GiadaContadino

Policy di accesso

Info

Definisci chi può accedere alla coda.

Scegli metodo

☒ Base

Utilizza criteri semplici per definire una policy di accesso di base.

☐ Avanzato

Utilizza un oggetto JSON per definire una policy di accesso avanzata.

Definisci chi può inviare messaggi alla coda

☒ Solo il proprietario della coda

Solo il proprietario della coda può inviare messaggi alla coda.

☐ Solo gli account AWS, gli utenti e i ruoli IAM specificati

Solo gli ID account AWS, gli utenti e i ruoli IAM specificati possono inviare messaggi alla coda.

Definisci chi può ricevere messaggi dalla coda

☒ Solo il proprietario della coda

Solo il proprietario della coda può ricevere messaggi dalla coda.

☐ Solo gli account AWS, gli utenti e i ruoli IAM specificati

Solo gli ID account AWS, gli utenti e i ruoli IAM specificati possono ricevere messaggi dalla coda.

JSON (sola lettura)

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "339712995867"
      },
      "Action": [
        "SQS:*"
      ],
      "Resource": "arn:aws:sqs:eu-central-1:339712995867:AWS-Serveless-Exercise"
    }
  ]
}
```

CloudShell

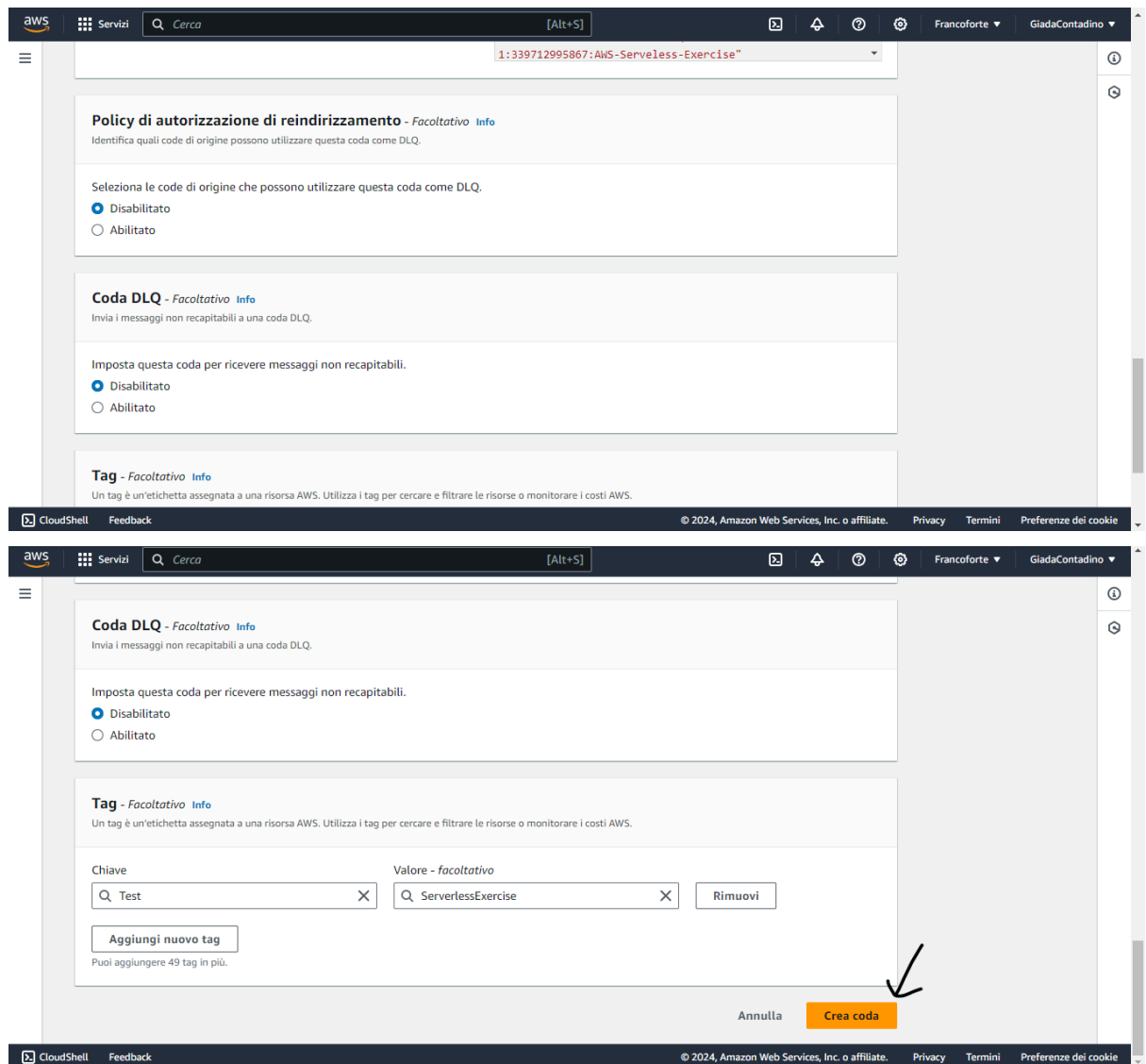
Feedback

© 2024, Amazon Web Services, Inc. o affiliate.

Privacy

Termini

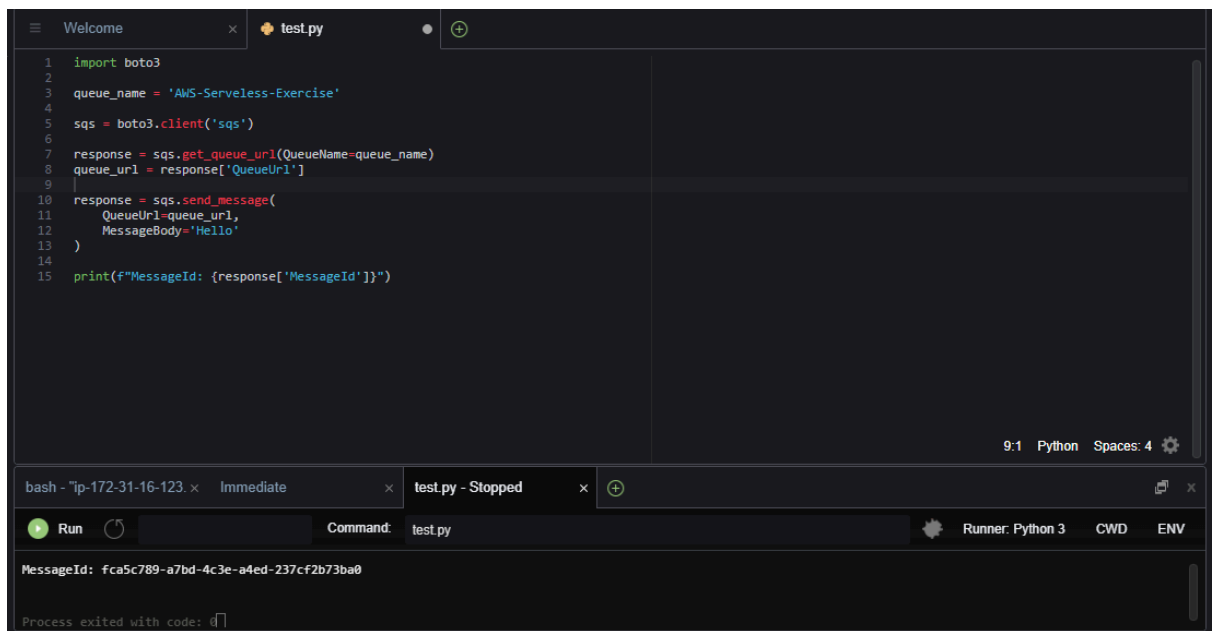
Preferenze dei cookie



## 2) Secondo step: Scrivere lo script in Python per il message producer

Come indicato nella email, ho riscontrato difficoltà nell'utilizzare Python sul mio computer a causa di problemi di path e Boto3. Per risolvere questo problema, ho optato per l'uso di AWS Cloud9, installando Boto3 tramite il terminale fornito dal servizio e caricando lo script nel file Python. Nello screenshot seguente è presente lo script utilizzato per inviare messaggi alla coda SQS precedentemente creata. Innanzitutto, ho importato boto3 e ho creato il client SQS, seguendo le indicazioni trovate nella documentazione. Successivamente ho inserito il nome della coda SQS e ho fatto in modo che avviato lo script, l'URL della coda venisse automaticamente ottenuto. Infine ho inserito il testo del messaggio e attraverso la funzione print, ho fatto sì che ogni volta che

lo script viene avviato, posso ottenere l'ID del messaggio per essere sicura che esso sia stato inviato alla coda.



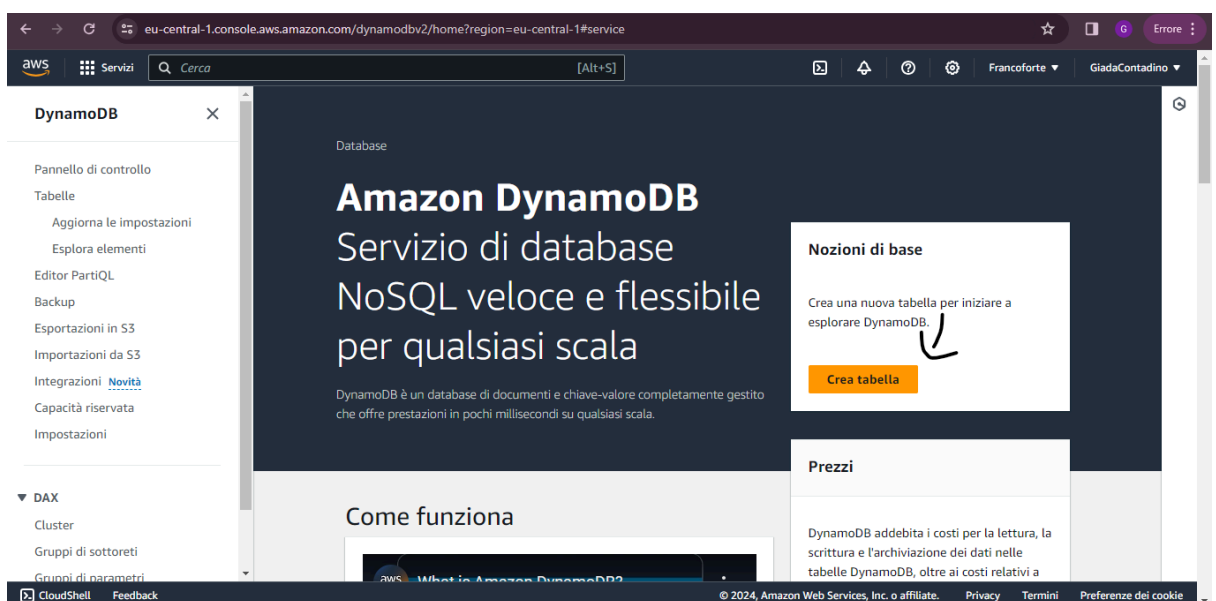
```
1 import boto3
2
3 queue_name = 'AWS-Serveless-Exercise'
4
5 sqs = boto3.client('sqs')
6
7 response = sqs.get_queue_url(QueueName=queue_name)
8 queue_url = response['QueueUrl']
9
10 response = sqs.send_message(
11     QueueUrl=queue_url,
12     MessageBody='Hello'
13 )
14
15 print(f"MessageId: {response['MessageId']}")
```

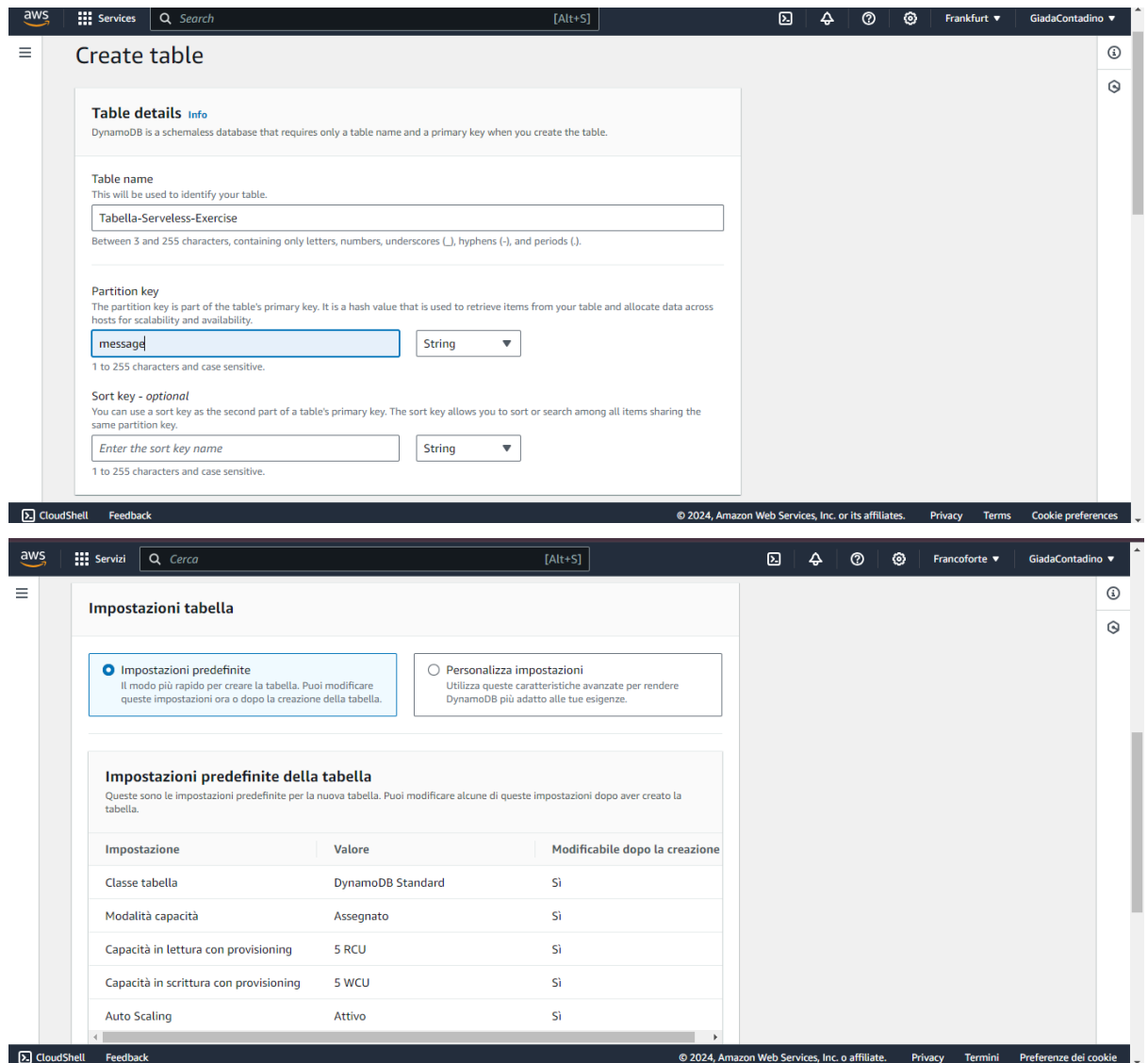
MessageId: fca5c789-a7bd-4c3e-a4ed-237cf2b73ba0

Process exited with code: 0

### 3) Terzo step: Creare la tabella di DynamoDB

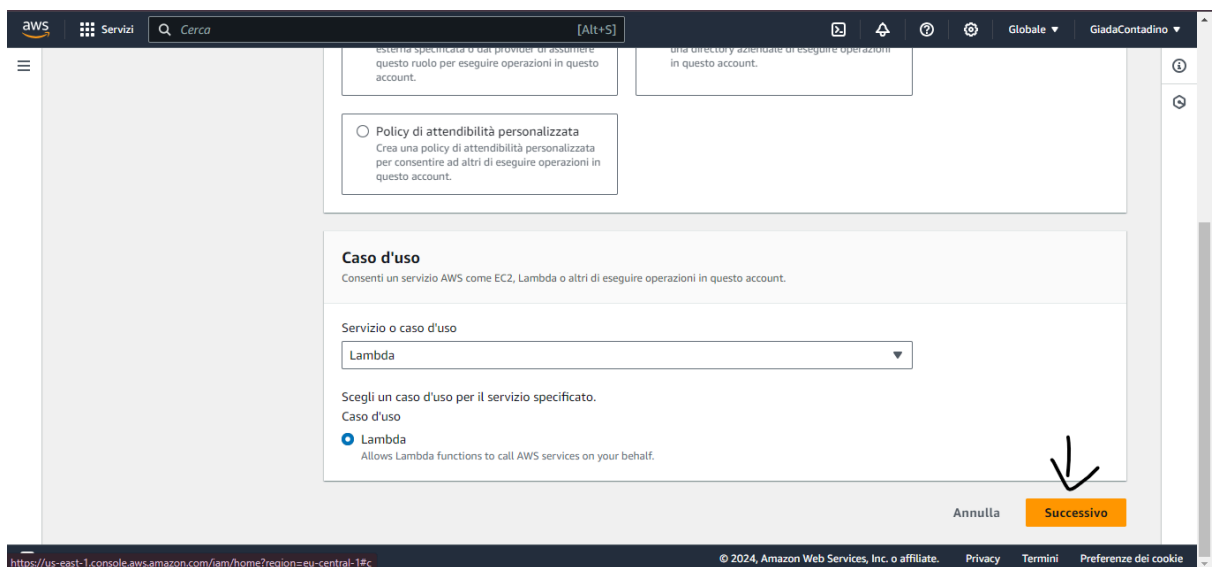
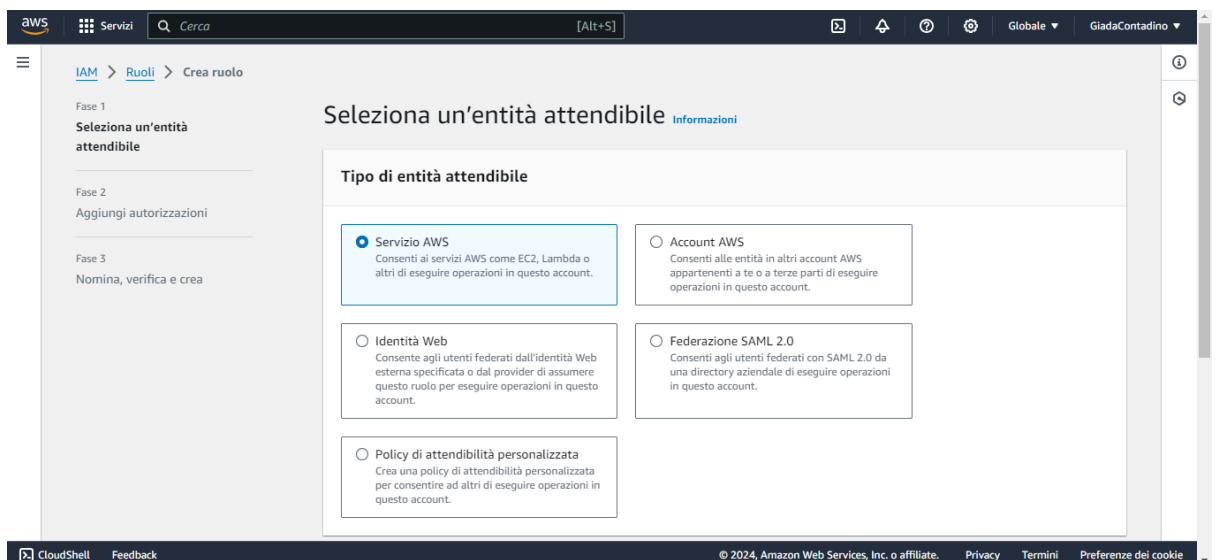
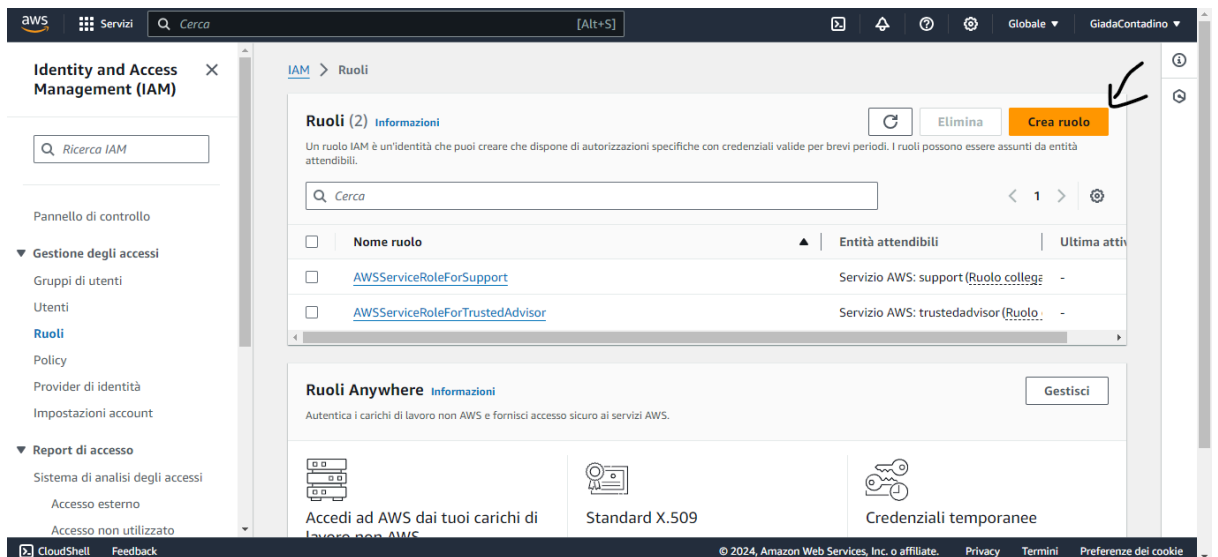
Ho creato la tabella di DynamoDB per far sì che i messaggi inviati dallo script (message producer) vengono processati dalla funzione Lambda (consumer) e caricati automaticamente nella tabella. Ho lasciato le configurazioni di default e ho indicato come chiave di partizione: 'message', come negli screenshots che seguono.

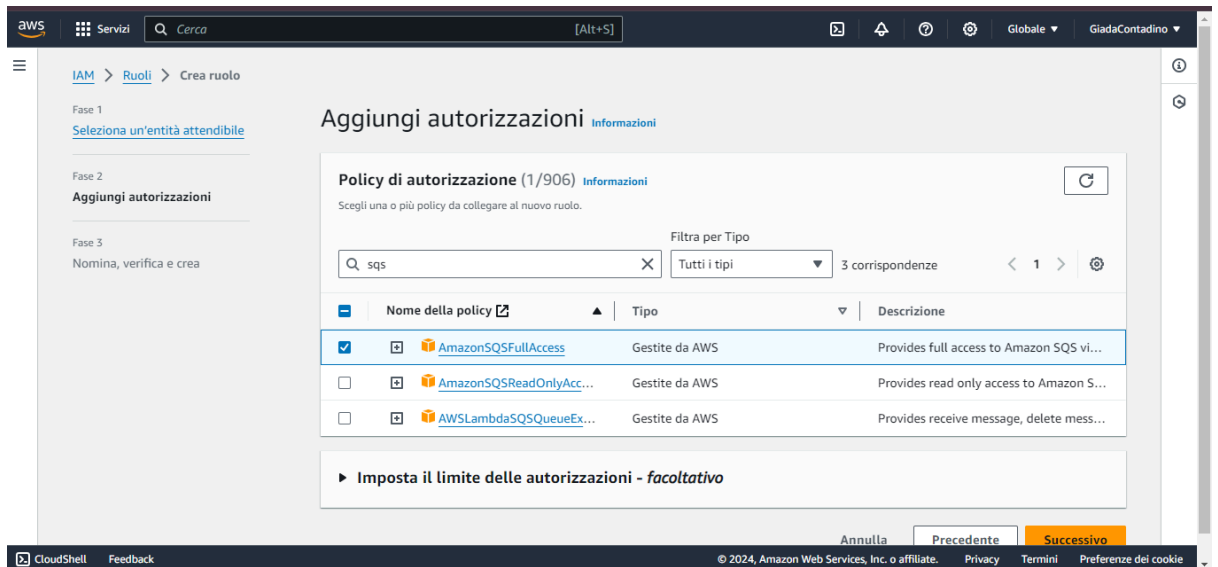




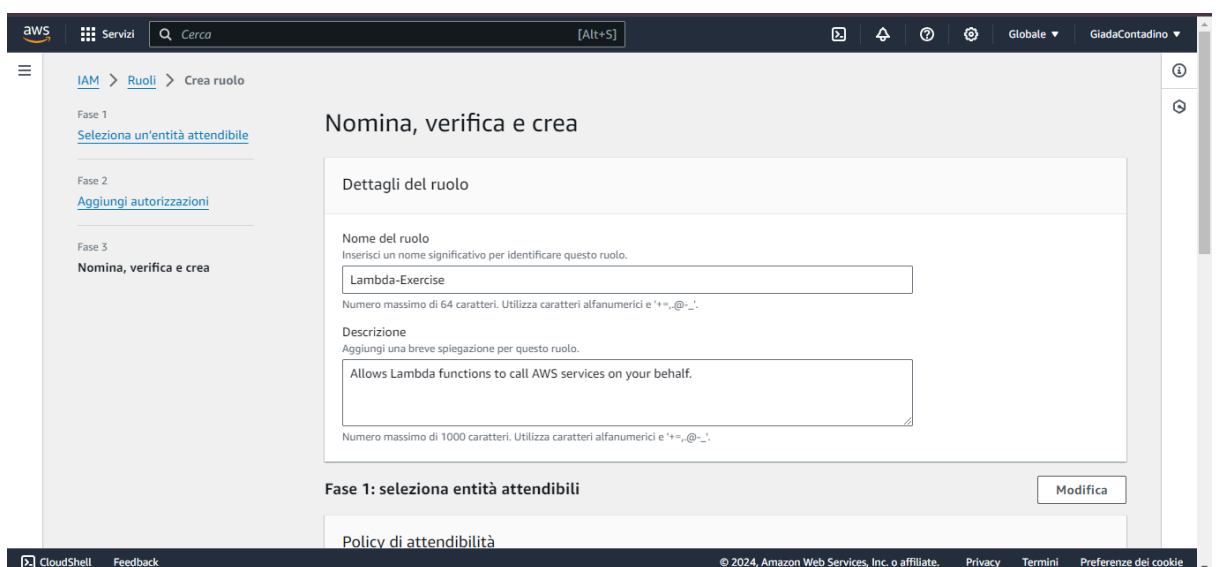
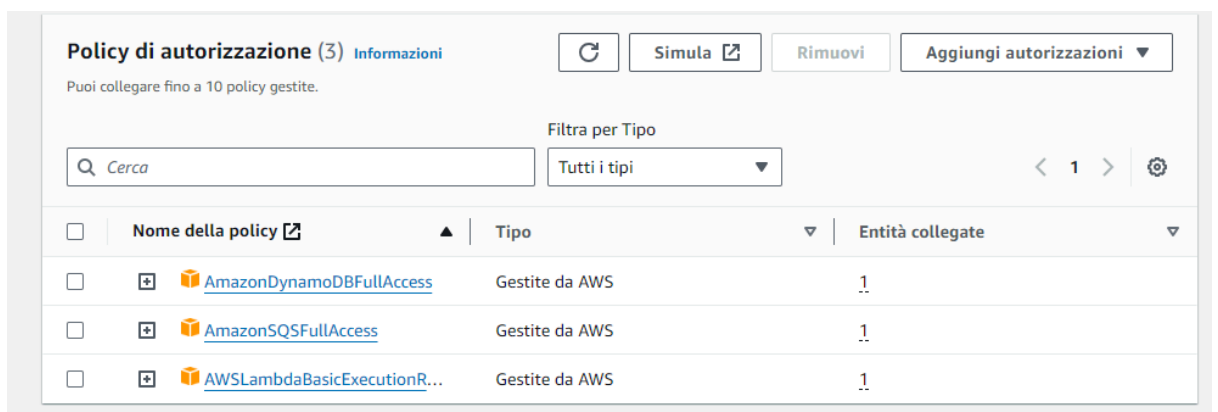
#### 4) Quarto step: Creare IAM role per Lambda con le policy di autorizzazione

Prima di creare la funzione Lambda, ho creato il ruolo IAM da specificare durante il processo di creazione della funzione. Come negli screenshots che seguono, ho creato un ruolo IAM collegato al servizio Lambda. Ad esso ho collegato le policy di autorizzazione per SQS e DynamoDB e la policy AWS Lambda BasicExecutionRole per poter visualizzare i Cloudwatch metrics e i file log di Cloudwatch legati all'esecuzione della funzione.



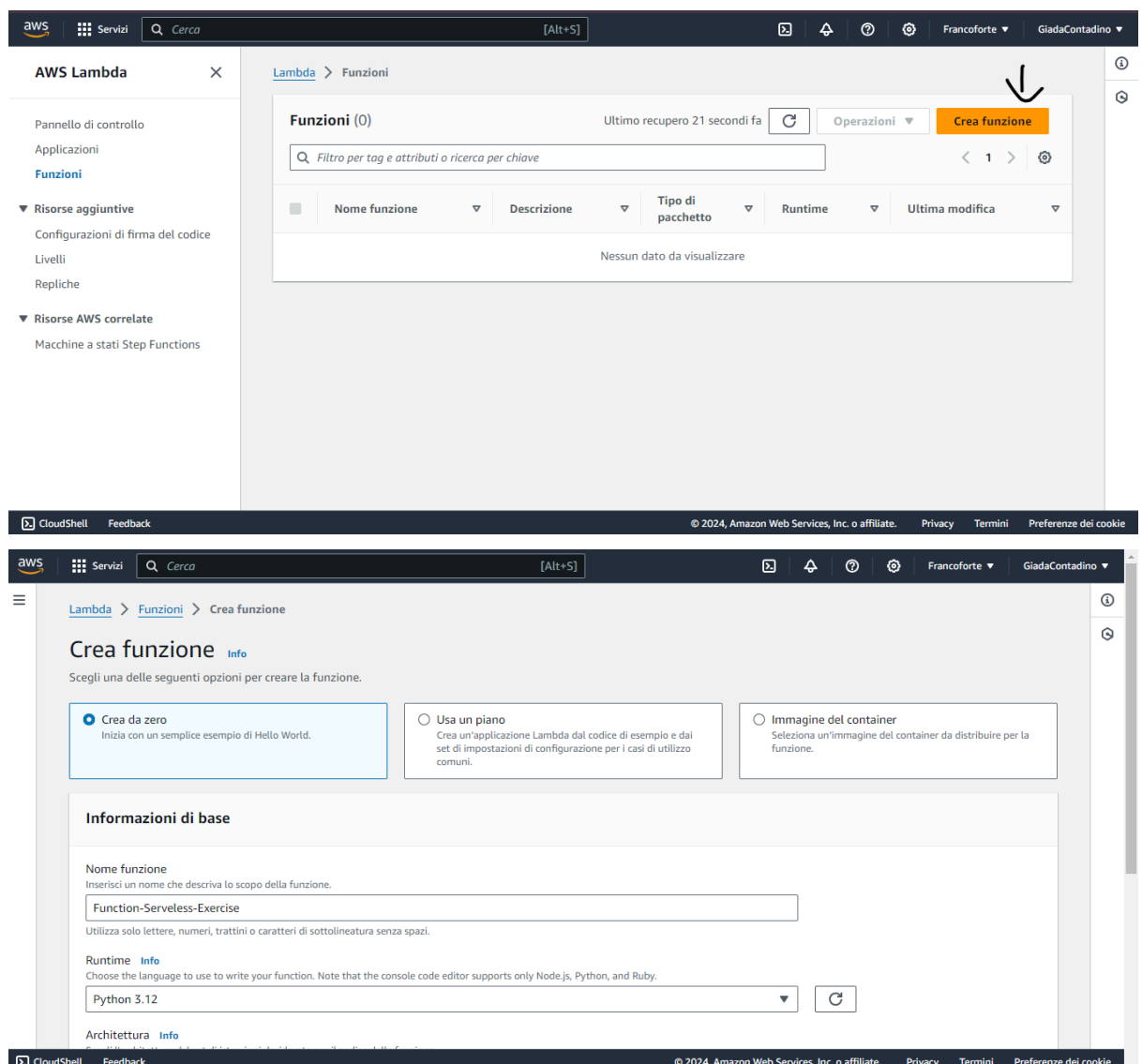


Qui riportate tutte le policy collegate al ruolo IAM:

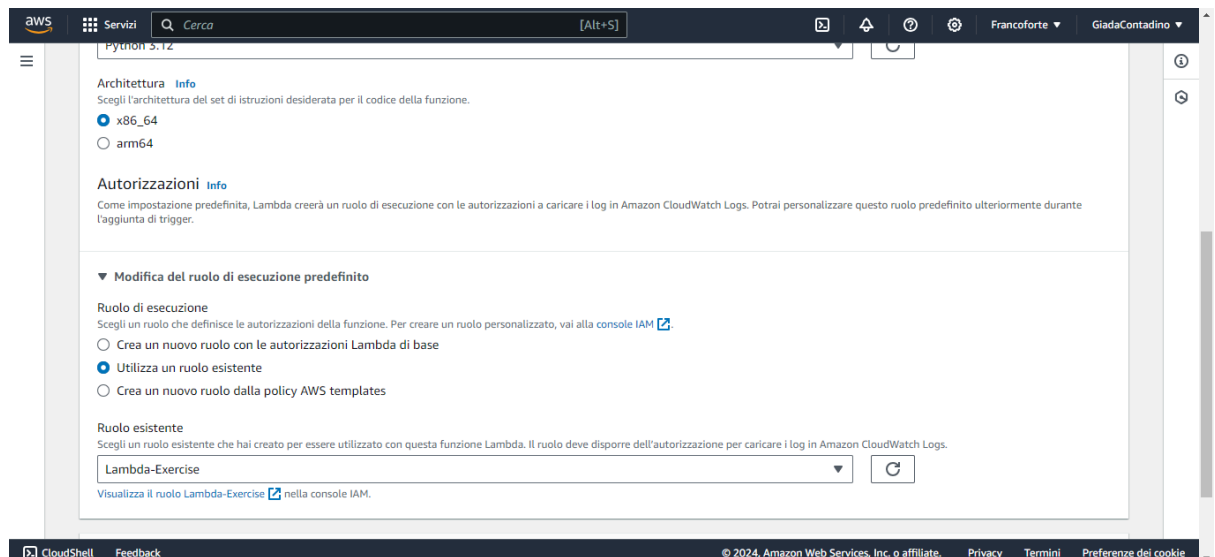


## 5) Quinto step: Creare la funzione di Lambda

Dopo la creazione del ruolo IAM, ho creato la funzione Lambda per registrare i messaggi che passano per la coda SQS nella tabella di DynamoDB. La funzione viene attivata ogni volta che viene avviato lo script, grazie al trigger che ho impostato dopo la creazione della funzione. Ho impostato python 3.12 per il runtime della funzione, dato che è il linguaggio di programmazione usato nel progetto e ho collegato il ruolo IAM creato nello step precedente.



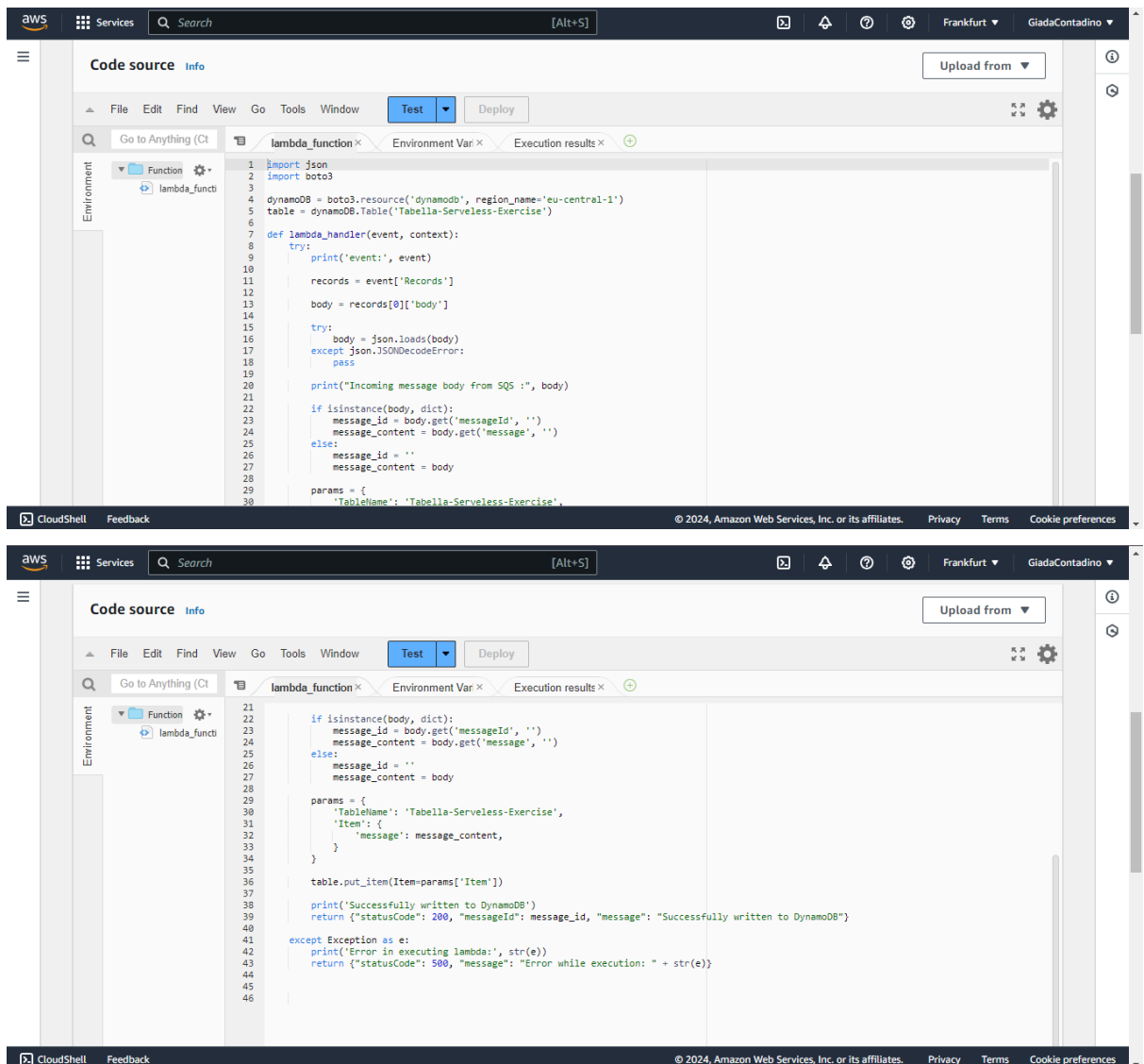




## 6) Sesto step: Scrivere la funzione Lambda (consumer) per conservare il messaggio su DynamoDB

Dopo la creazione della funzione Lambda, ho caricato lo script python nella tab 'code' che dovrà registrare il messaggio inviato dal message producer (AWS Cloud9) nella tabella di DynamoDB. Innanzitutto ho importato json e boto3 e ho specificato la resource (DynamoDB) e le caratteristiche della tabella (nome e regione). Successivamente troviamo la funzione `lambda_handler` che prende il parametro `event`. Nella funzione, `event` contiene i dettagli dell'evento che ha innescato la funzione. Dato che la funzione è chiamata dalla coda SQS, `event` contiene i record della coda. Successivamente, avviene l'estrazione dei record dall'evento. In questo caso, i record sono i messaggi nella coda. La variabile `body` viene utilizzata per estrarre il corpo del record. Ho aggiunto il parsing del corpo del messaggio come JSON, a causa di errori trovati nei log di Cloudwatch, quando ho testato la funzione. Viene tentato il parsing del corpo come JSON utilizzando `json.loads(body)`. Se il parsing ha successo, significa che il corpo è in formato JSON e viene sostituito alla variabile `body`. Se il parsing fallisce, il codice assume che il corpo sia una stringa normale e continua senza modifiche (non viene sostituito alla variabile `body`). La funzione 'stampa' (`print`) il corpo del messaggio ricevuto ed estrae i valori di `messageId` e `message`. Se il corpo è una stringa normale, il corpo del messaggio viene utilizzato come contenuto del messaggio, e `messageId` viene impostato su una stringa vuota. Ho scelto di utilizzare questa distinzione cercata su internet per adattare la funzione a diverse strutture di messaggi provenienti dalla coda SQS, evitando alcuni errori che ho riscontrato

durante i test. La funzione utilizza i valori estratti (params) per creare l'elemento da scrivere nella tabella DynamoDB. La funzione scrive quindi questo elemento nella tabella utilizzando `table.put_item()`. Se si verifica un'eccezione durante l'esecuzione di qualsiasi parte del codice, viene stampato un messaggio di errore e la funzione restituisce uno stato di errore (`statusCode: 500`) con un messaggio descrittivo, dato da `+str(5)`. Se tutto il processo di scrittura su DynamoDB ha successo, la funzione restituisce uno stato di successo (`statusCode: 200`) con un messaggio che indica che la funzione è stata eseguita con successo.



The image displays two screenshots of the AWS Lambda console, showing the code source for a function named 'lambda\_function'.

The top screenshot shows the initial code, which imports `json` and `boto3`, initializes a `dynamoDB` resource and a `table` object, and defines a `lambda_handler` function. The function prints the event, extracts the body from the event, and attempts to load it as JSON. If successful, it prints the body. If an exception occurs, it prints an error message and returns a status code of 500.

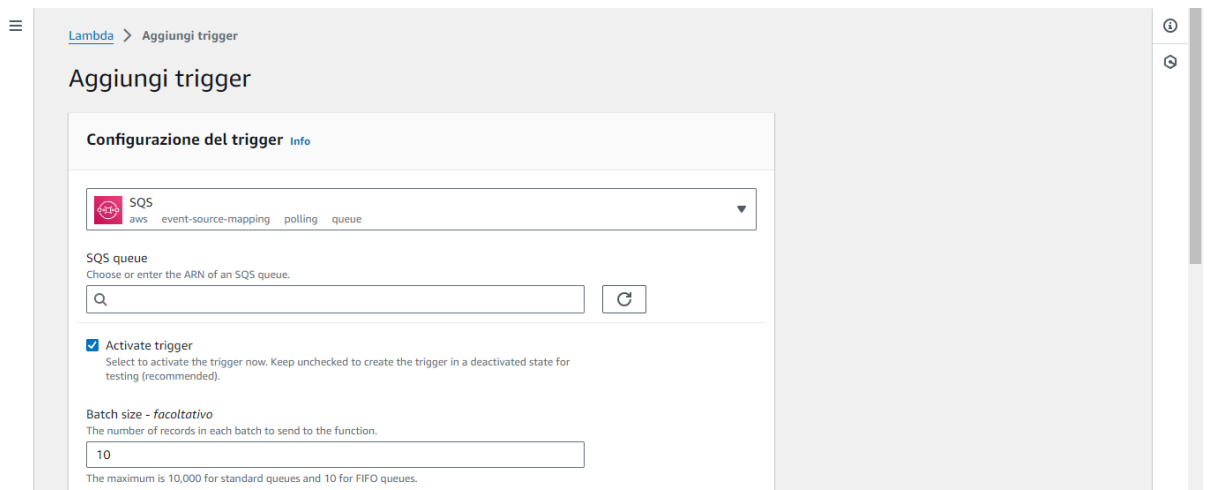
```
1 import json
2 import boto3
3
4 dynamoDB = boto3.resource('dynamodb', region_name='eu-central-1')
5 table = dynamoDB.Table('Tabella-Serveless-Exercise')
6
7 def lambda_handler(event, context):
8     try:
9         print('event:', event)
10
11         records = event['Records']
12         body = records[0]['body']
13
14         try:
15             body = json.loads(body)
16         except json.JSONDecodeError:
17             pass
18
19         print("Incoming message body from SQS :", body)
20
21         if isinstance(body, dict):
22             message_id = body.get('messageId', '')
23             message_content = body.get('message', '')
24         else:
25             message_id = ''
26             message_content = body
27
28         params = {
29             'TableName': 'Tabella-Serveless-Exercise',
30             'Item': {
31                 'message': message_content,
32             }
33         }
34
35         table.put_item(Item=params['Item'])
36
37         print('Successfully written to DynamoDB')
38         return {'statusCode': 200, 'messageId': message_id, 'message': 'Successfully written to DynamoDB'}
```

The bottom screenshot shows the updated code, which includes the `table.put_item()` call and the return statement. The function now returns a status code of 200 and a message indicating successful execution.

```
21
22 if isinstance(body, dict):
23     message_id = body.get('messageId', '')
24     message_content = body.get('message', '')
25 else:
26     message_id = ''
27     message_content = body
28
29 params = {
30     'TableName': 'Tabella-Serveless-Exercise',
31     'Item': {
32         'message': message_content,
33     }
34 }
35
36 table.put_item(Item=params['Item'])
37
38 print('Successfully written to DynamoDB')
39 return {'statusCode': 200, 'messageId': message_id, 'message': 'Successfully written to DynamoDB'}
40
41 except Exception as e:
42     print('Error in executing lambda:', str(e))
43     return {'statusCode': 500, 'message': 'Error while execution: ' + str(e)}
```

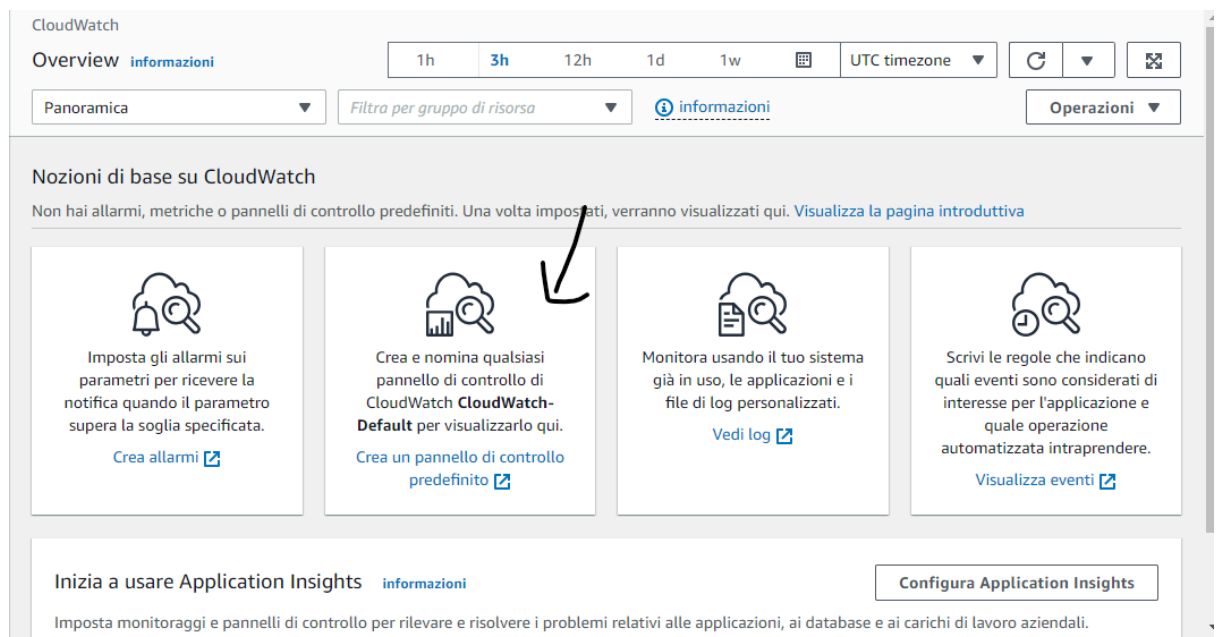
- 7) Settimo step: Attivare il trigger SQS per far sì che la funzione Lambda venga attivata ogni volta che lo script su AWS Cloud9 viene avviato

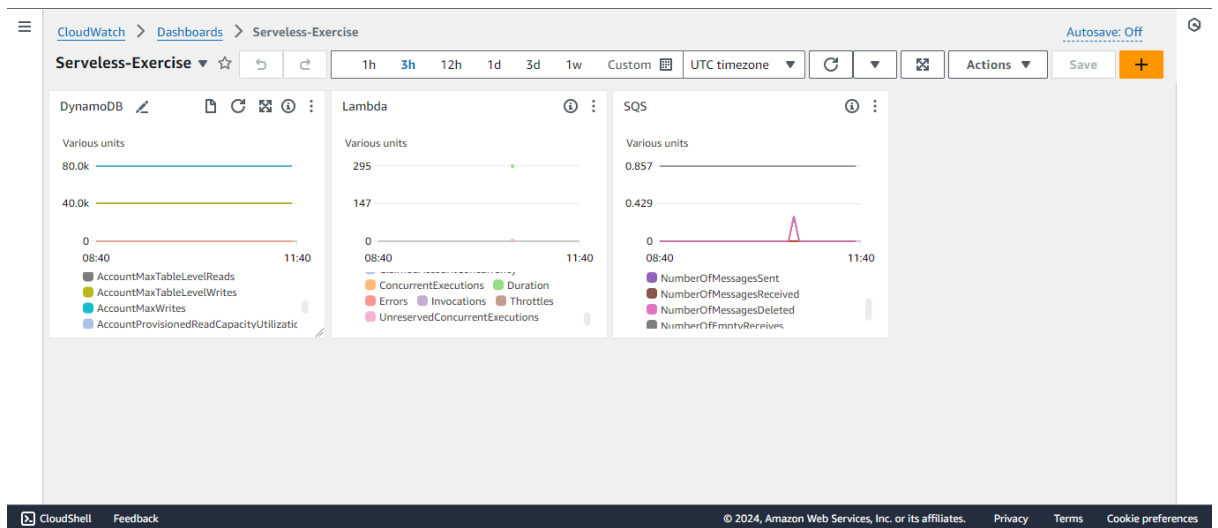
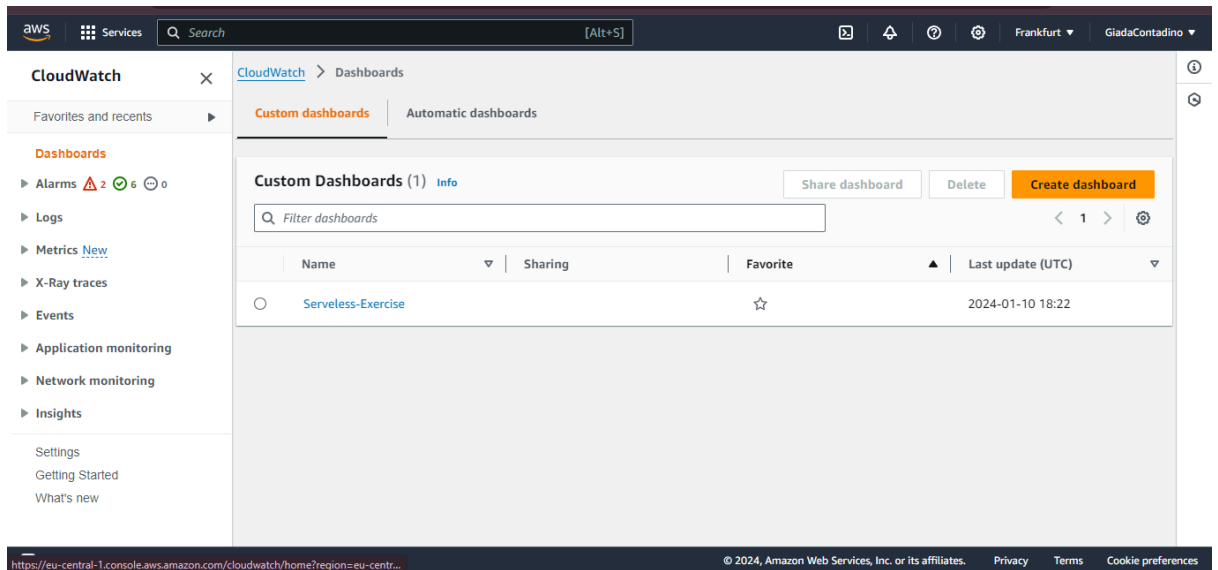
Per garantire che la funzione venga attivata ogni volta che SQS riceve un messaggio (in questo caso, il messaggio è inviato dallo script su AWS Cloud9), ho aggiunto il trigger di SQS alla funzione.



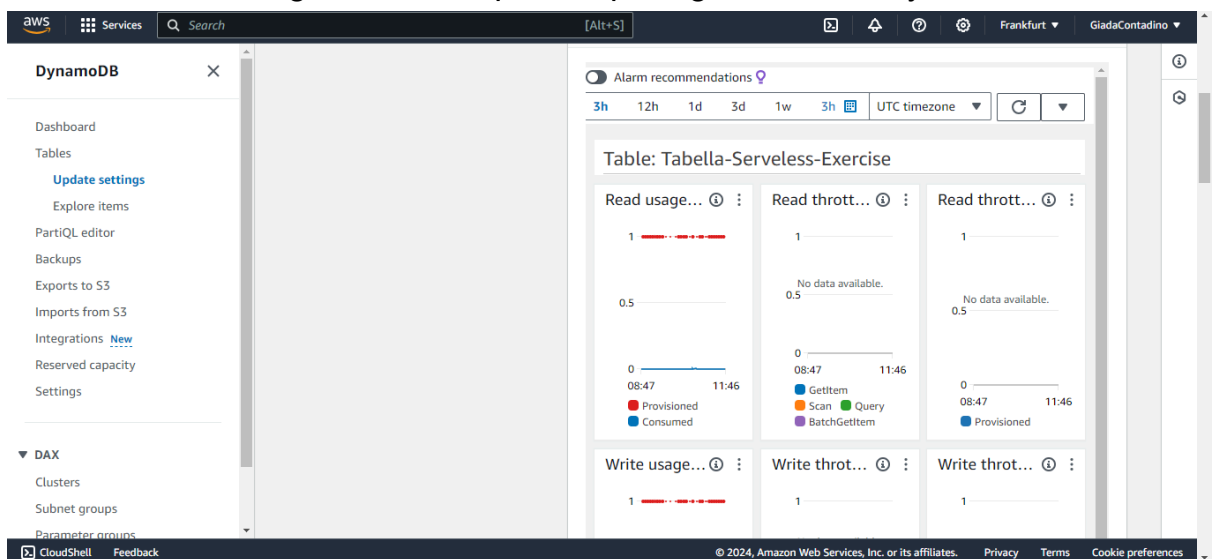
- 8) Ottavo step: Collegare Cloudwatch per effettuare il monitoring

Al fine di monitorare le prestazioni dei servizi utilizzati nel progetto (SQS, DynamoDB e Lambda), ho creato una dashboard su CloudWatch specificando i servizi da monitorare.

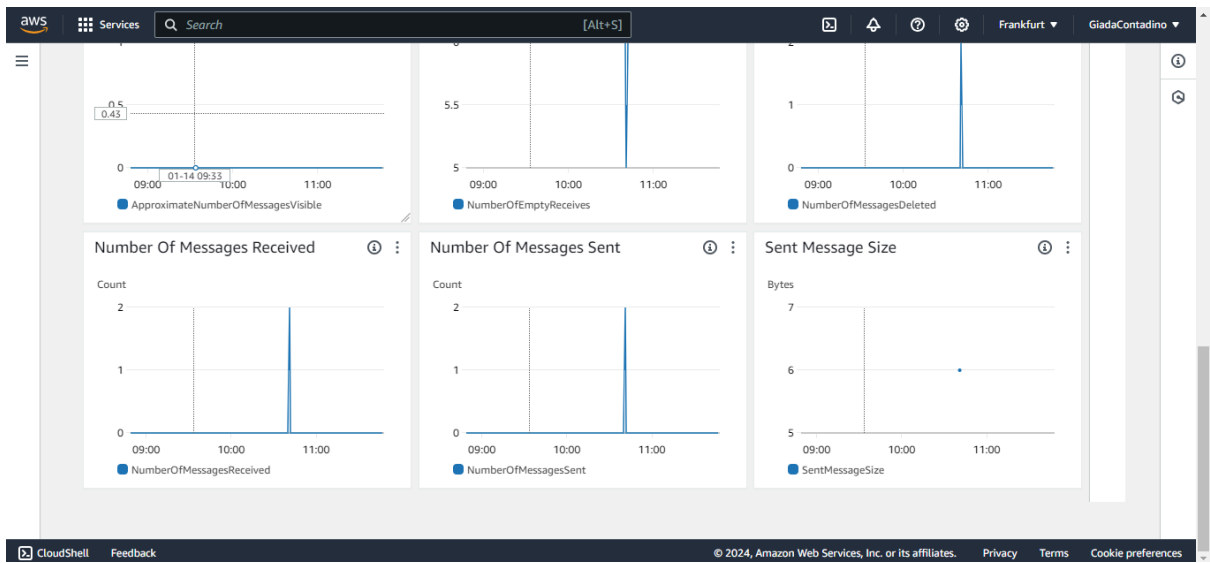




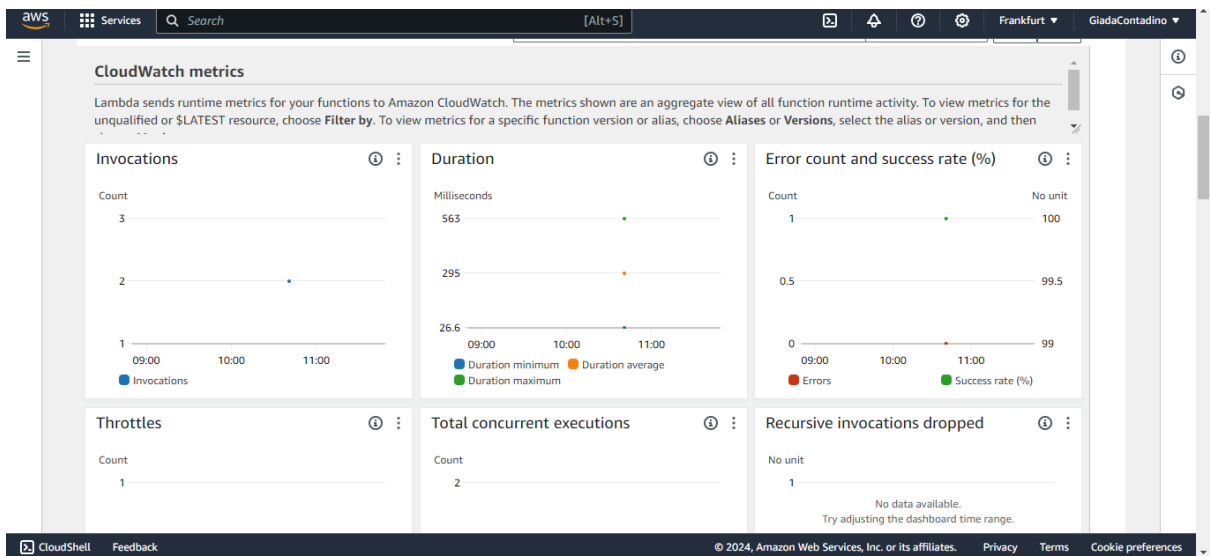
Gli screenshots seguenti sono specifici per ogni servizio. DynamoDB:



SQS:

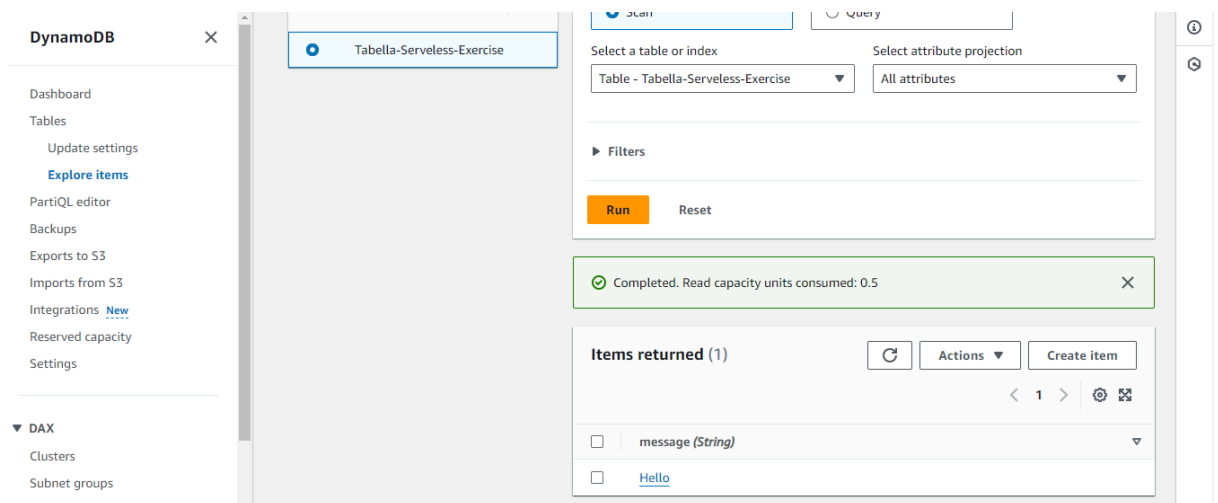
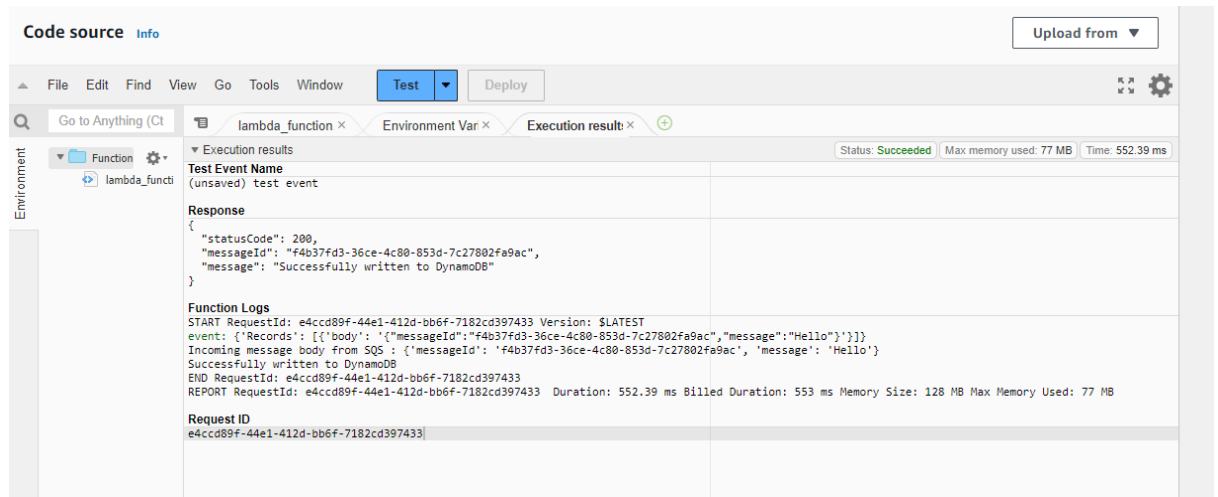


Lambda:

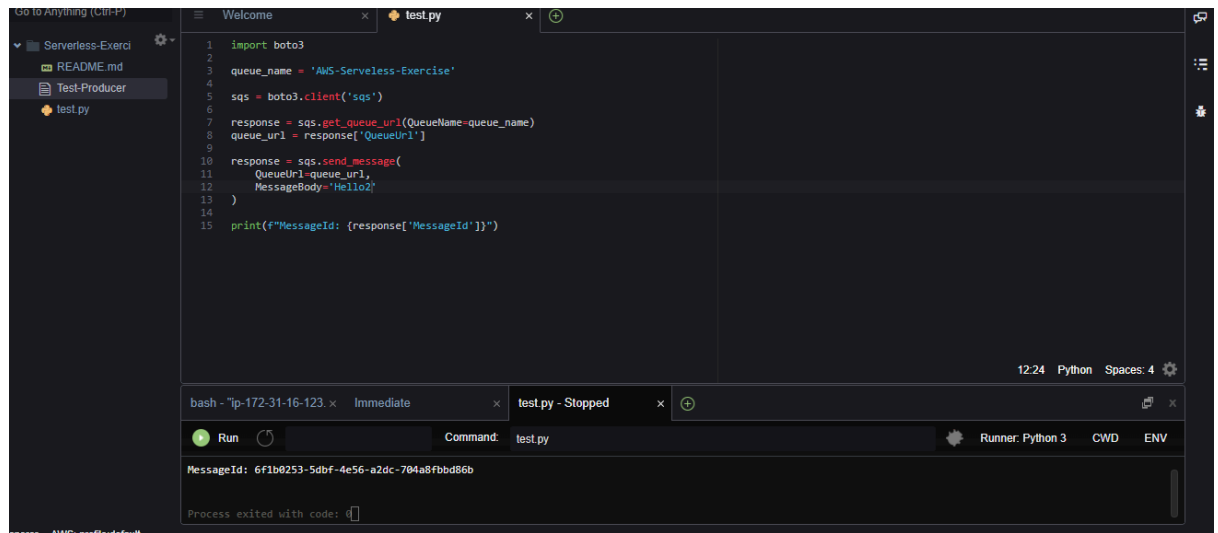


- 9) Nono step: Verificare il corretto funzionamento della funzione attraverso i test e controllare i log di CloudWatch

Inizialmente, ho testato la funzione creando un evento di test con un messageId casuale e un body del messaggio contenente "Hello". Ho potuto constatare che la funzione ha lavorato correttamente e che il messaggio è stato salvato nella tabella DynamoDB.



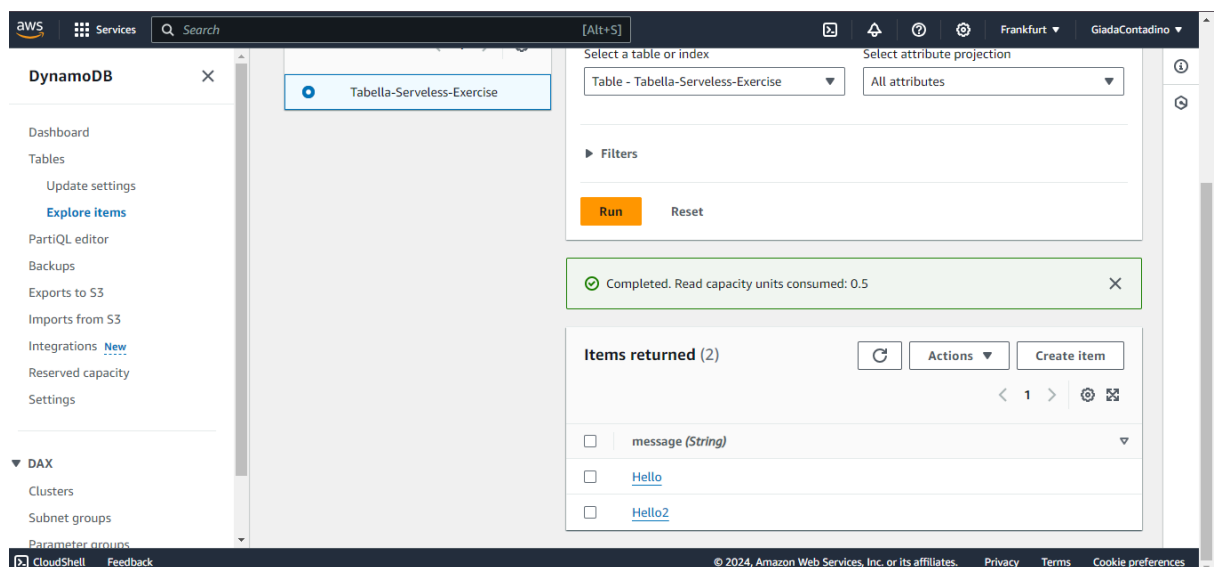
Successivamente, ho eseguito un test effettivo avviando lo script Python salvato su AWS Cloud9, modificando il body del messaggio in "Hello2" per verificare che il messaggio venisse salvato nella tabella. Attraverso la tabella e i log di CloudWatch, ho verificato che la funzione è stata attivata all'invio del messaggio alla coda e che il messaggio è stato salvato nella tabella DynamoDB.



```
1 import boto3
2
3 queue_name = 'AWS-Serveless-Exercise'
4
5 sqs = boto3.client('sqs')
6
7 response = sqs.get_queue_url(QueueName=queue_name)
8 queue_url = response['QueueUrl']
9
10 response = sqs.send_message(
11     QueueUrl=queue_url,
12     MessageBody='Hello2'
13 )
14
15 print(f"MessageId: {response['MessageId']}")
```

MessageId: 6f1b0253-5dbf-4e56-a2dc-704a8fbbd86b

Process exited with code: 0



# Optional requirements

## 1) Implement dead-letter queues for handling processing failures.

Per poter utilizzare una coda come dlq collegata alla main queue (in caso di errori durante il processo, i messaggi vengono indirizzati a questa coda e non alla principale), ho creato innanzitutto una nuova coda abilitata per essere utilizzata come dlq, come segue negli screenshots.

Amazon SQS > Queues > Create queue

### Create queue

**Details**

Type  
Choose the queue type for your application or cloud infrastructure.

☒ **Standard** [Info](#)  
At-least-once delivery, message ordering isn't preserved

- At-least-once delivery
- Best-effort ordering

☐ **FIFO** [Info](#)  
First-in-first-out delivery, message ordering is preserved

- First-in-first-out delivery
- Exactly-once processing

**ⓘ** You can't change the queue type after you create a queue.

Name

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (\_).

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Services Search [Alt+S] Frankfurt GiadaContadino

### Configuration

Set the maximum message size, visibility to other consumers, and message retention.

Visibility timeout [Info](#)  
 Seconds  
Should be between 0 seconds and 12 hours.

Message retention period [Info](#)  
 Days  
Should be between 1 minute and 14 days.

Delivery delay [Info](#)  
 Seconds  
Should be between 0 seconds and 15 minutes.

Maximum message size [Info](#)  
 KB  
Should be between 1 KB and 256 KB.

Receive message wait time [Info](#)  
 Seconds  
Should be between 0 and 20 seconds.

### Encryption

Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption.

Server-side encryption  
☐ Disabled

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



aws

Services

Search

[Alt+S]

Frankfurt

GiadaContadino

Encryption

Info

Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption.

Server-side encryption

☐ Disabled

☒ Enabled

Encryption key type

☒ Amazon SQS key (SSE-SQS)

An encryption key that Amazon SQS creates, manages, and uses for you.

☐ AWS Key Management Service key (SSE-KMS)

An encryption key protected by AWS Key Management Service (AWS KMS).

Access policy

Info

Define who can access your queue.

Choose method

☒ Basic

Use simple criteria to define a basic access policy.

☐ Advanced

Use a JSON object to define an advanced access policy.

Define who can send messages to the queue

JSON (read-only)

CloudShell

Feedback

© 2024, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

aws

Services

Search

[Alt+S]

Frankfurt

GiadaContadino

Access policy

Info

Define who can access your queue.

Choose method

☒ Basic

Use simple criteria to define a basic access policy.

☐ Advanced

Use a JSON object to define an advanced access policy.

Define who can send messages to the queue

☒ Only the queue owner

Only the owner of the queue can send messages to the queue.

☐ Only the specified AWS accounts, IAM users and roles

Only the specified AWS account IDs, IAM users and roles can send messages to the queue.

Define who can receive messages from the queue

☒ Only the queue owner

Only the owner of the queue can receive messages from the queue.

☐ Only the specified AWS accounts, IAM users and roles

Only the specified AWS account IDs, IAM users and roles can receive messages from the queue.

JSON (read-only)

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "339712995867"
      },
      "Action": [
        "SQS:*"
      ],
      "Resource": "arn:aws:sqs:eu-central-1:339712995867:dead_letter_queue"
    }
  ]
}
```

aws

Services

Search

[Alt+S]

Frankfurt

GiadaContadino

Redrive allow policy - Optional

Info

Identify which source queues can use this queue as the dead-letter queue.

Select which source queues can use this queue as the dead-letter queue.

☐ Disabled

☒ Enabled

Redrive permission

☐ Allow all

Allow all source queues from the same account and in the same region to use this queue as the dead-letter queue

☒ By queue

Allow a list of source queues from the same account and in the same region to use this queue as the dead-letter queue

☐ Deny all

Deny all source queues from using this queue as the dead-letter queue

Warning: if any source queue that is not listed below uses this queue as the dead-letter queue, all messages moved from the source queue to the dead-letter queue will fail.

Select the ARN for each source queue.

arn:aws:sqs:eu-central-1:339712995867:AWS-Serveless-Exercise

Add new source queue

CloudShell

Feedback

© 2024, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

Choose queue  
arn:aws:sqs:eu-central-1:339712995867:AWS-Serveless-Exercise

Maximum receives  
10  
Should be between 1 and 1000

**Tags - Optional** [Info](#)  
A tag is a label assigned to an AWS resource. Use tags to search and filter your resources or track your AWS costs.

Key  Value - optional  [Remove](#)

[Add new tag](#)  
You can add 49 more tags.

[Cancel](#) [Create queue](#)

Dopo la creazione della dead letter queue, ho selezionato su SQS la coda principale e ho cliccato edit nella tab 'dead letter queue'. Ho abilitato l'opzione di mandare i 'undeliverable messages' alla dead letter queue creata precedentemente.

Amazon SQS > Queues > AWS-Serveless-Exercise

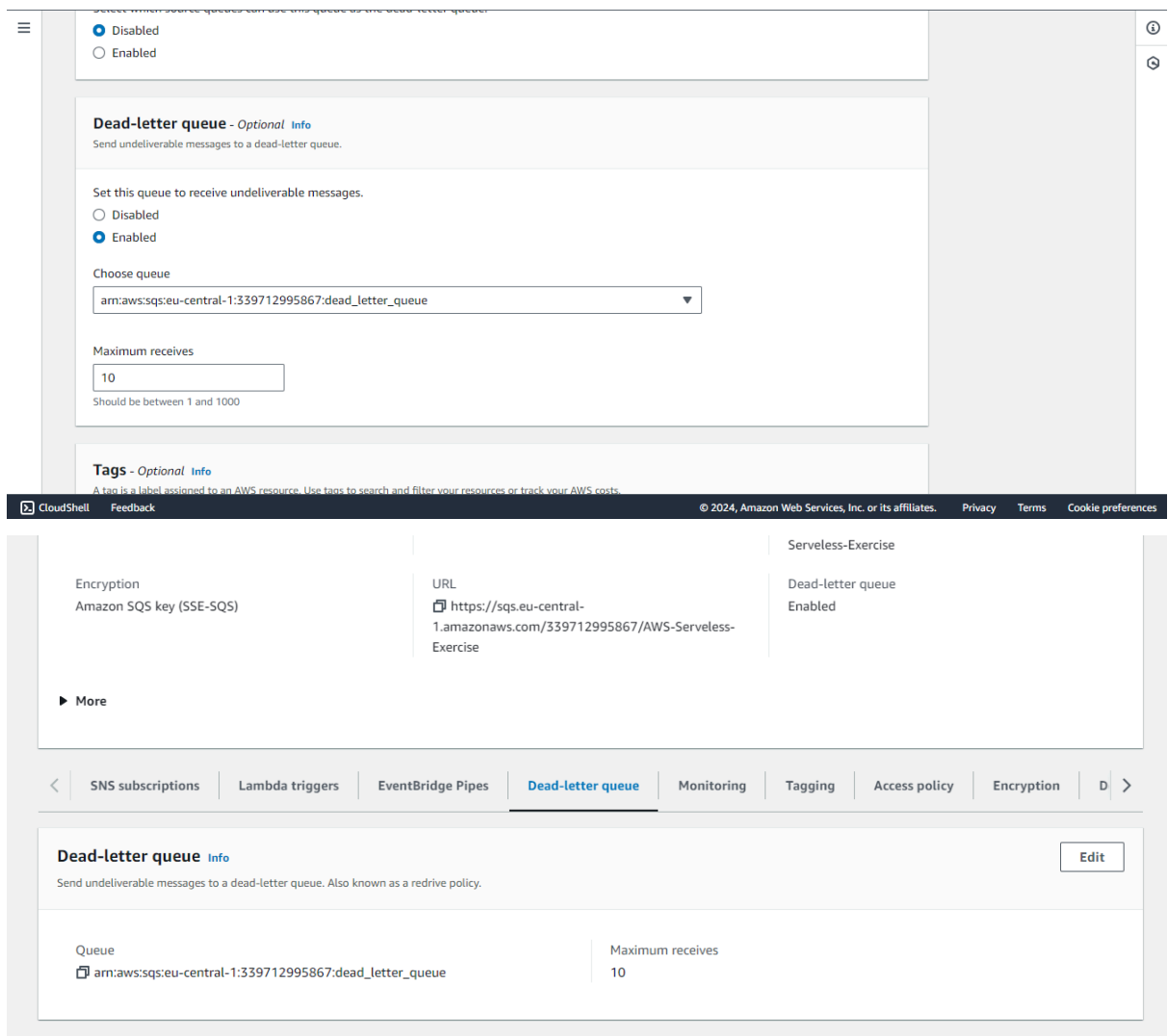
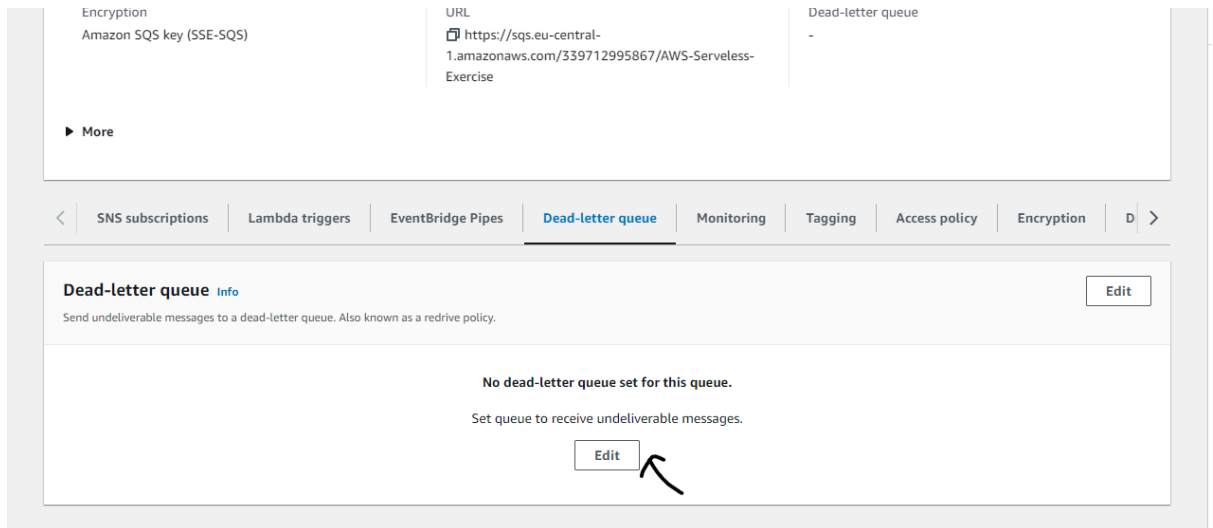
[Edit](#) [Delete](#) [Purge](#) [Send and receive messages](#) [Start DLQ redrive](#)

**Details** [Info](#)

Name AWS-Serveless-Exercise	Type Standard	ARN arn:aws:sqs:eu-central-1:339712995867:AWS-Serveless-Exercise
Encryption Amazon SQS key (SSE-SQS)	URL <a href="https://sqs.eu-central-1.amazonaws.com/339712995867/AWS-Serveless-Exercise">https://sqs.eu-central-1.amazonaws.com/339712995867/AWS-Serveless-Exercise</a>	Dead-letter queue -

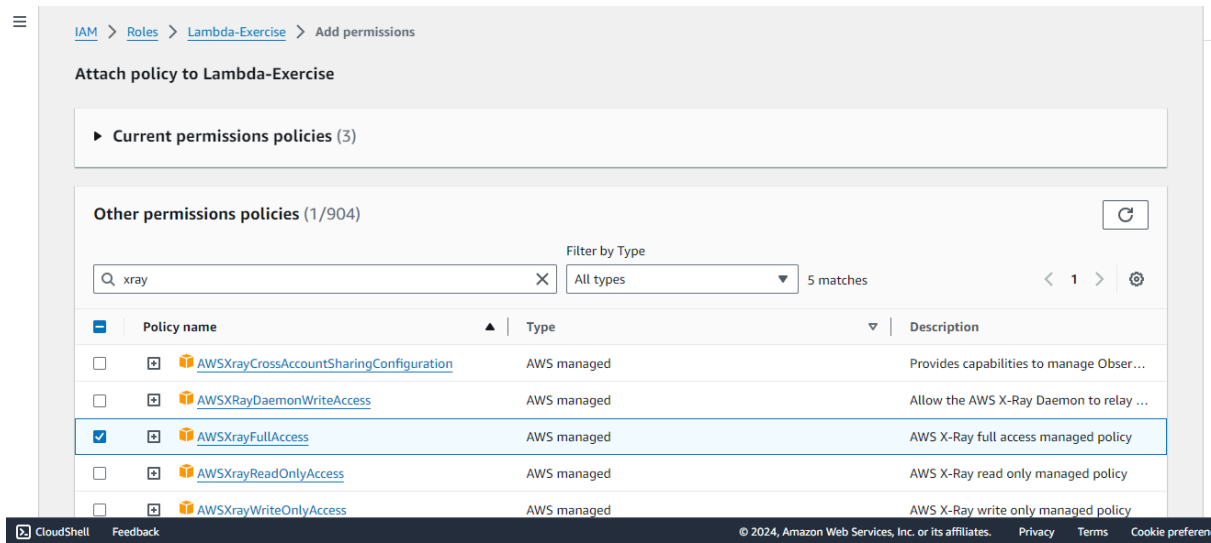
[More](#)

[SNS subscriptions](#) [Lambda triggers](#) [EventBridge Pipes](#) [Dead-letter queue](#) [Monitoring](#) [Tagging](#) [Access policy](#) [Encryption](#) [D](#)

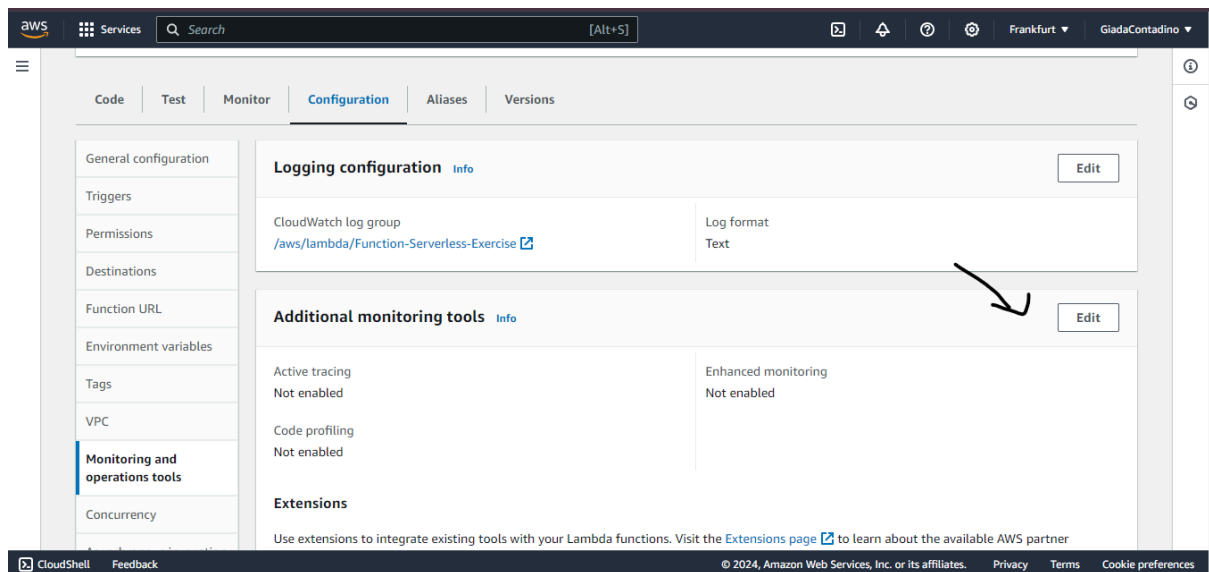


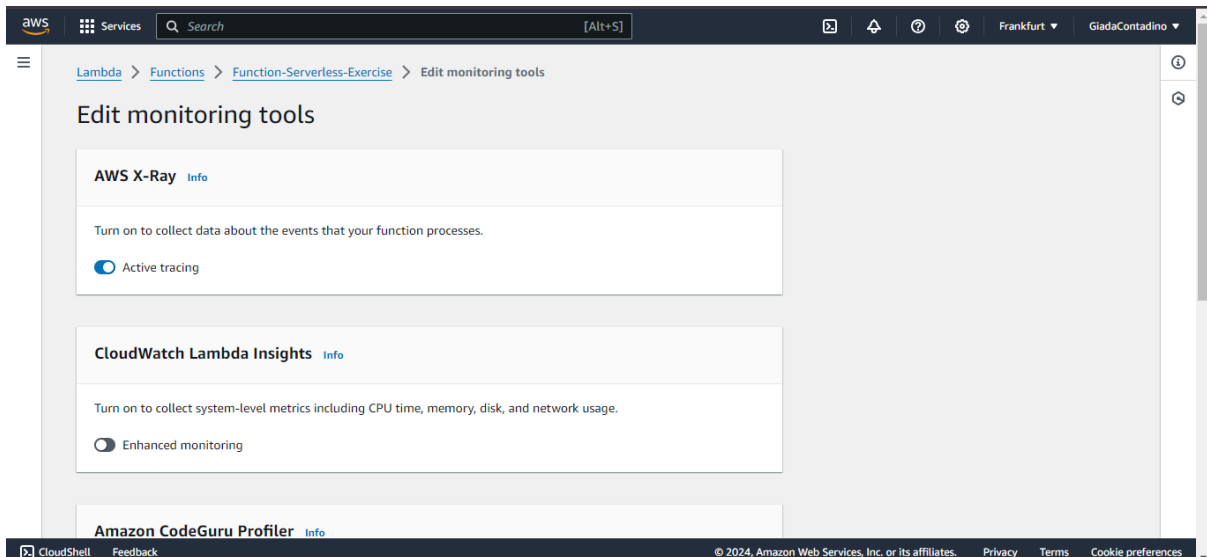
2) Implement AWS X-Ray for tracing and analyzing the behavior of the Lambda, SQS and DynamoDB.

Innanzitutto modifico il ruolo IAM, aggiungendo la policy di autorizzazione per AWS X-Ray.

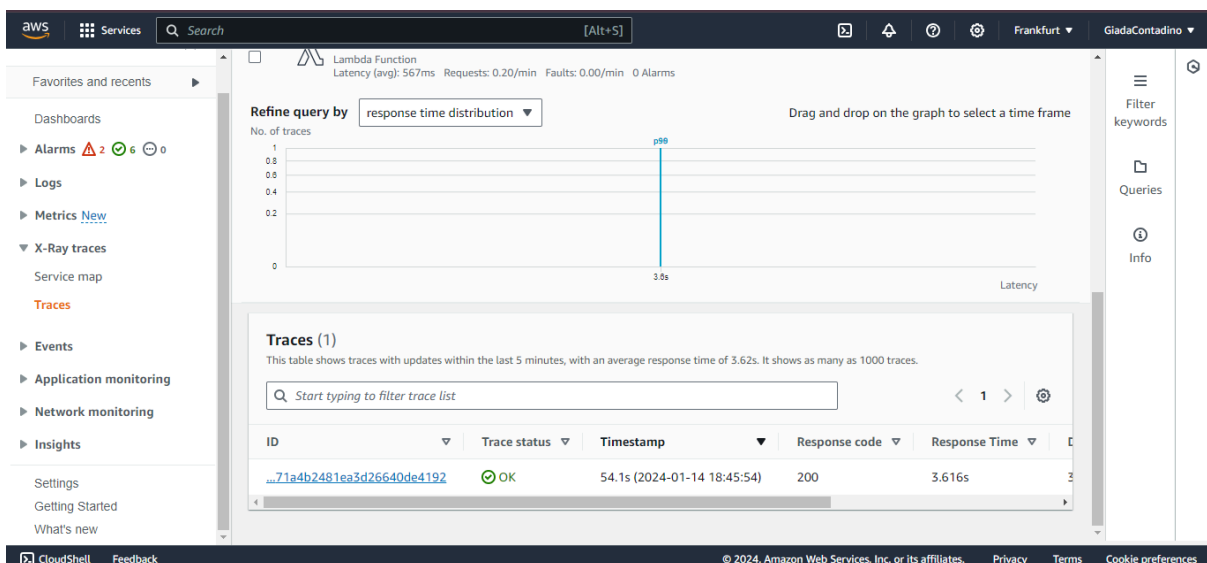
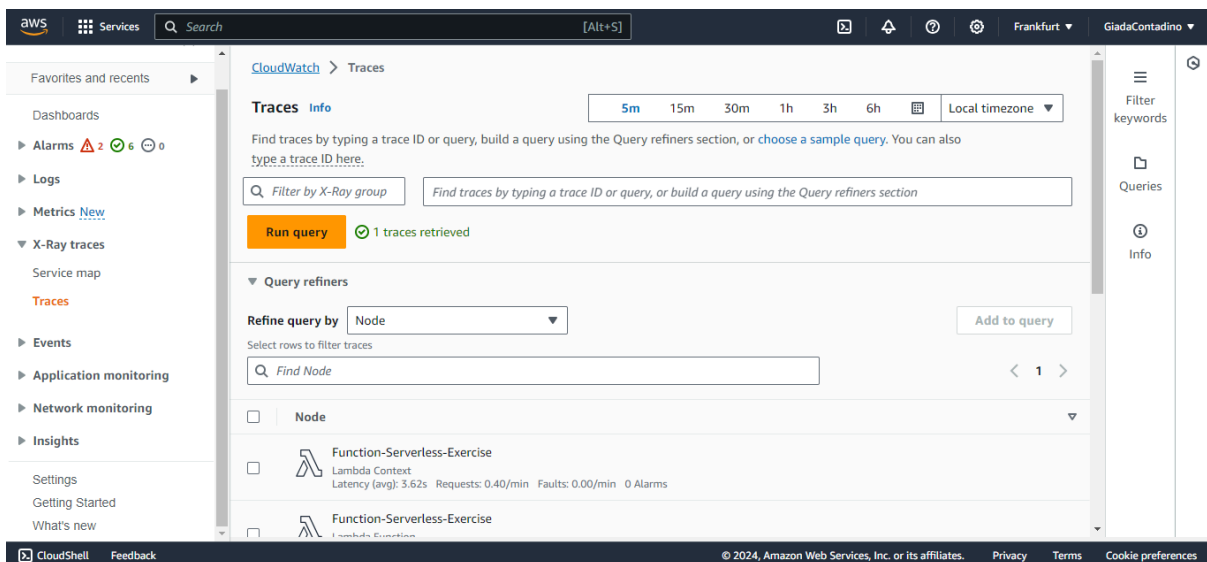


Successivamente vado su Lambda, clicco nella tab Configuration della mia funzione, clicco a sinistra su monitoring and operations tools e infine clicco su edit per Additional monitoring tools. Abilito Active Tracing (AWS X-Ray).





Infine controllo se nelle tracce di AWS X-Ray è stata registrata correttamente l'invocazione della funzione.



aws

Services

Search

[Alt+S]

Frankfurt

GiadaContadino

CloudWatch

Traces

Trace 1-65a41dd2-71a4b2481ea3d26640de4192

Trace details

Raw data

Method: -. Response Code: 200. Duration: 3.6s. Age: 10 minutes (2024-01-14 18:45:58)

Trace details

Select a node to see its details

Legend and options

Client

O Function-...s-Exercise  
Lambda Context

O Function-...s-Exercise  
Lambda Function

aws

Services

Search

[Alt+S]

Frankfurt

GiadaContadino

Segments Timeline

Info

Name	Segment status	Response code	Duration	Hosted in
▼ Function-Serverless-Exercise AWS::Lambda				
Function-Serverless-Exerc...	OK	200	3.62s	
▼ Function-Serverless-Exercise AWS::Lambda::Function				
Function-Serverless-Exerc...	OK	-	567ms	
Initialization	OK	-	461ms	
Invocation	OK	-	566ms	
Overhead	OK	-	0ms	

Logs

Info

All logs for this trace

View in CloudWatch Logs Insights

#	@log	@timestamp	@message
1	339712995867:/aws/lambda/Function-Serverless-Exercise	2024-01-14T17:45:57.631Z	START RequestId: 3eda241c-0e4e-5ba7-9ed3-7a022e4391a5 Version: \$LATEST
2	339712995867:/aws/lambda/Function-Serverless-Exercise	2024-01-14T17:45:58.198Z	REPORT RequestId: 3eda241c-0e4e-5ba7-9ed3-7a022e4391a5 Duration: 567.08 ms Billed...