Soliz, Joseph
Simplilearn Post-Graduate Cloud Computing
AWS Solutions Architect
8/15/2024

AWS Solution Architect Course Project

Infrastructure Deployment for Real-time Data Management Requirements on the AWS Cloud

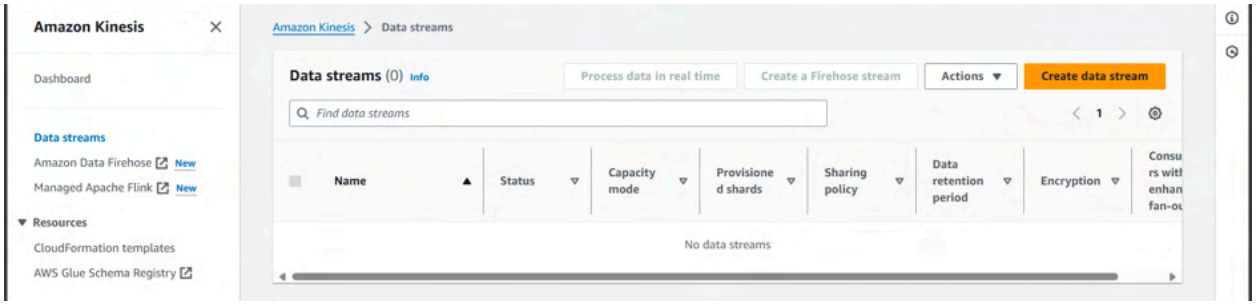
Objective

Create data in a Kinesis stream that can be copied to the DynamoDB database.

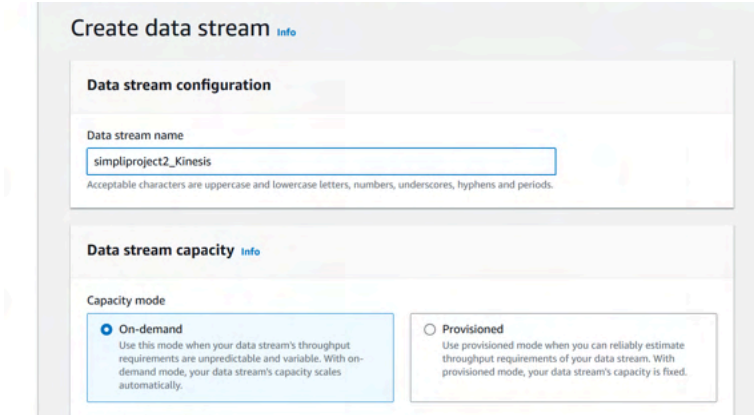
Tasks:

1. Create an AWS Kinesis Data Streams
2. Create an AWS Lambda Function
3. Create an AWS DynamoDB Database
4. Perform Scan Operation

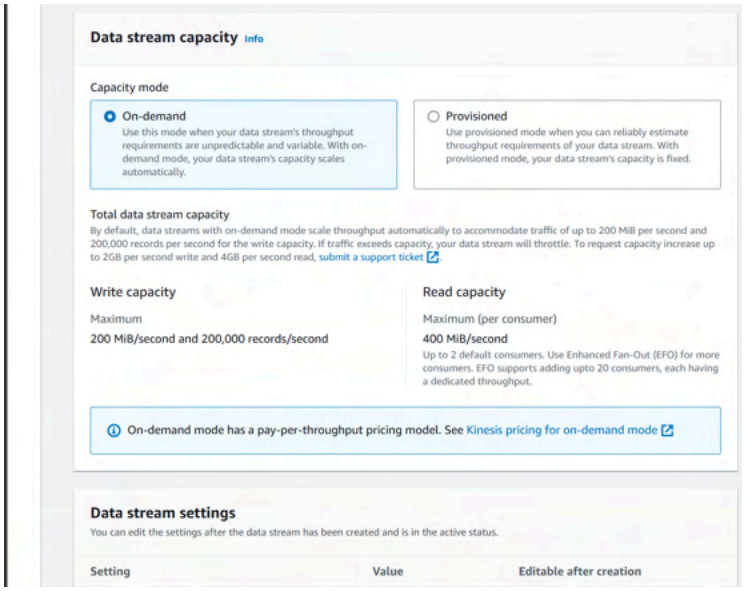
Step 1: Create a Kinesis Data Stream



Go to Kinesis. Go to Data Streams and click on “Create Data Stream”.

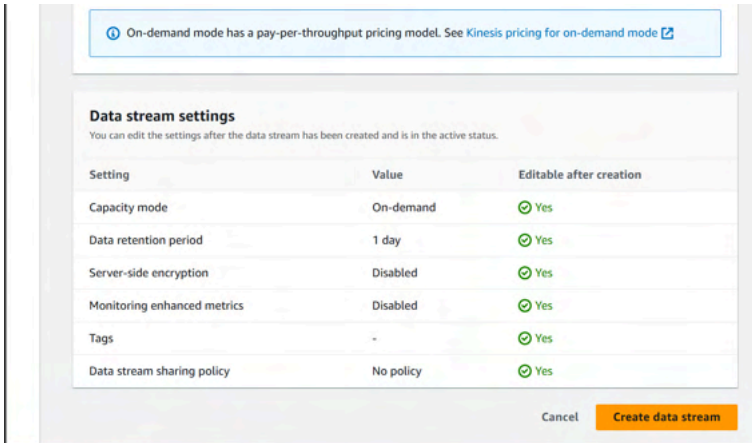


In Create data stream, create a name for the data stream. In this case “simpliproject2_Kinesis”.

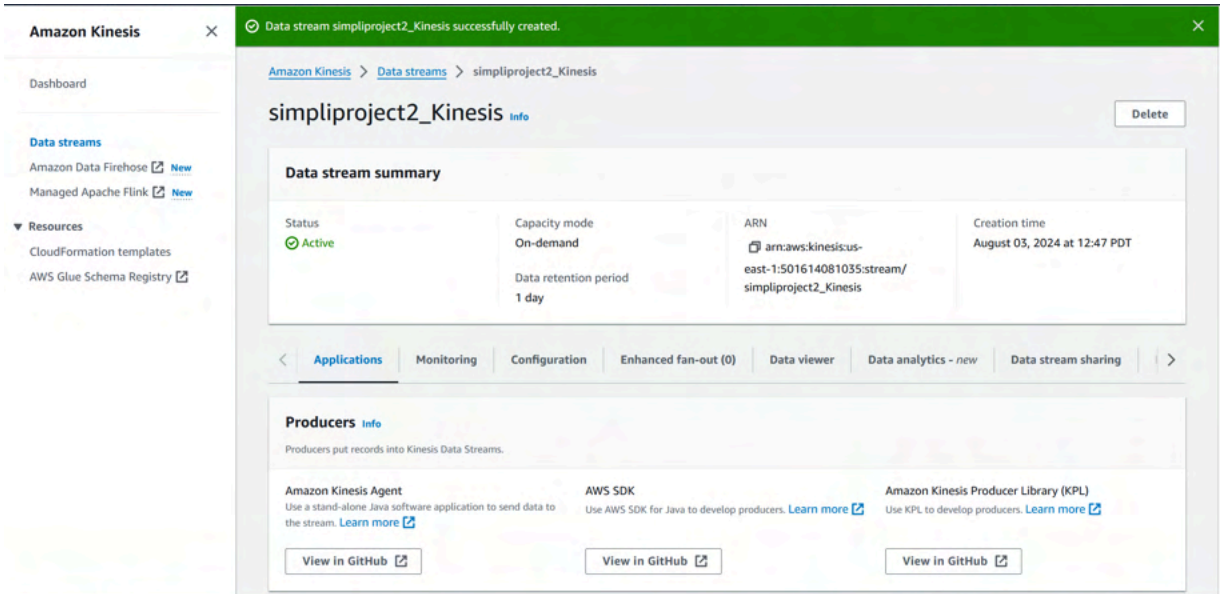


You give two options for Dat stream capacity: On-Demand or Provisioned.

In this case, we will use On-Demand. You can add and edit data streams after creation.



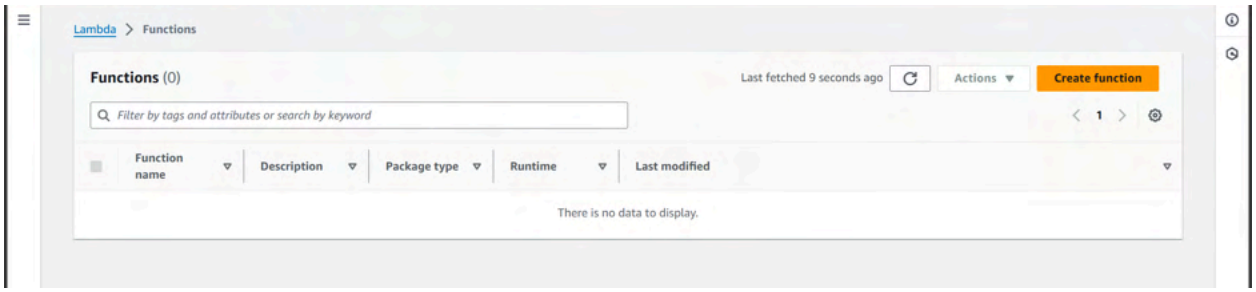
Click on Create data stream.



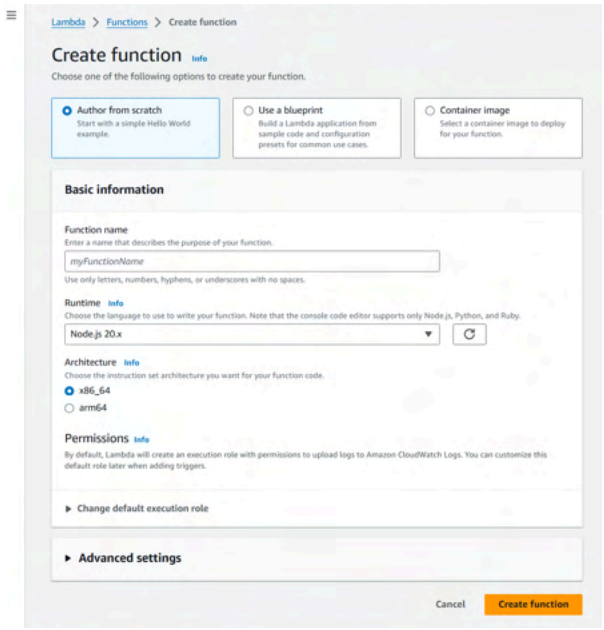
Kinesis data stream created.

Step 2: Create an AWS Lambda Function

2.1: Lamba Function Producer



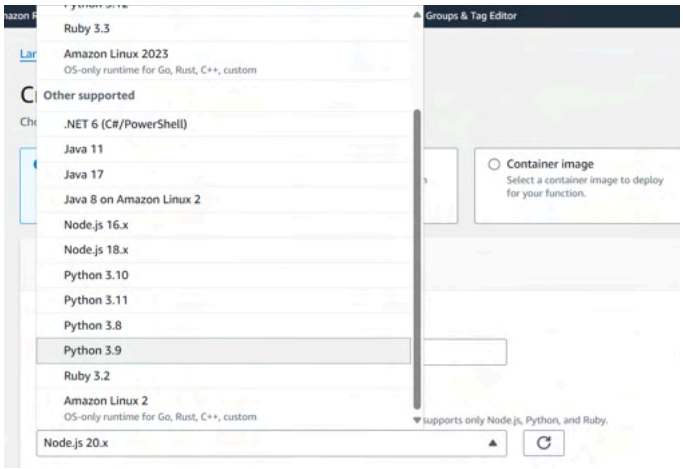
Go to Lambda function. Under Functions and click on ‘Create function’.



Under Create Function, enter the name you want for your function.

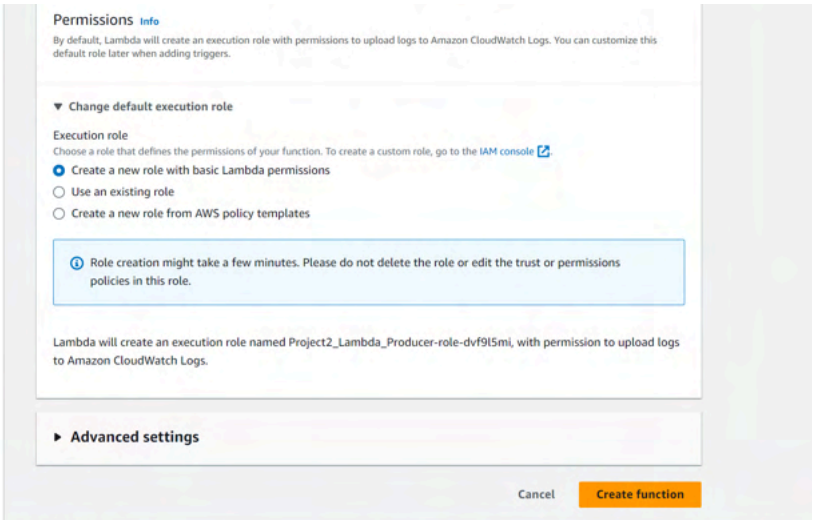
Example: “Project2_Lambda_Producer”.

You will be creating two lambda functions for this project; The first one is the producer for your Kinesis.



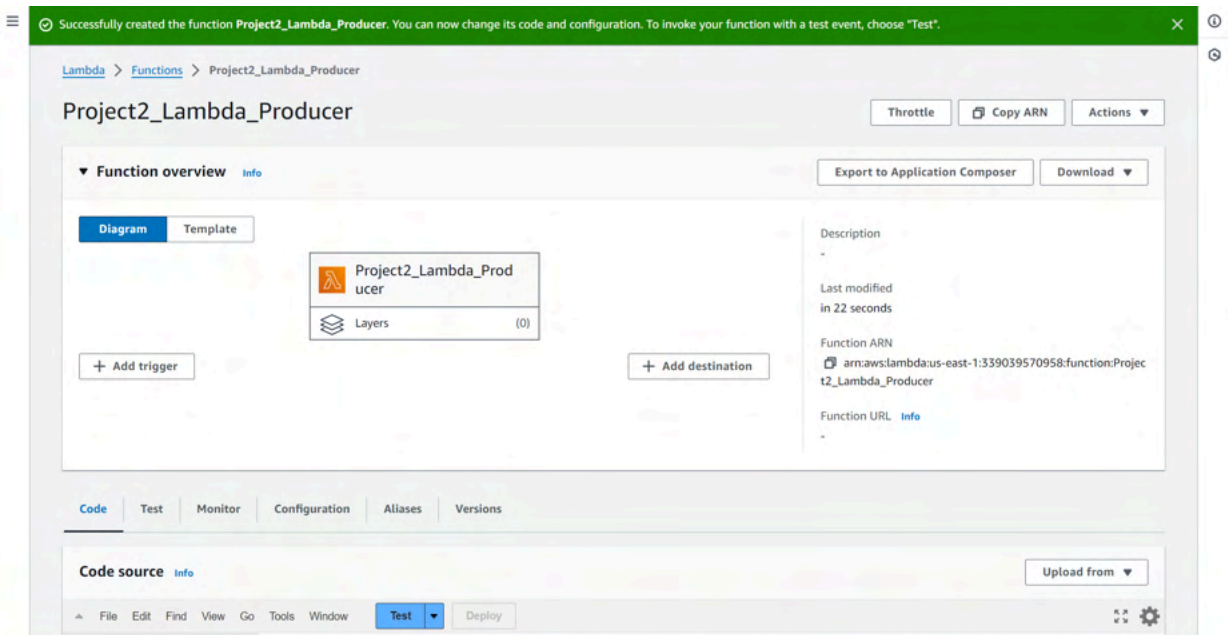
For Runtime, select Python 3.9 as your consumer set.

Keep Architecture at x86_64.



Under ‘Change default execution role’ in ‘Execution Role’. You will receive a new role in IAM.

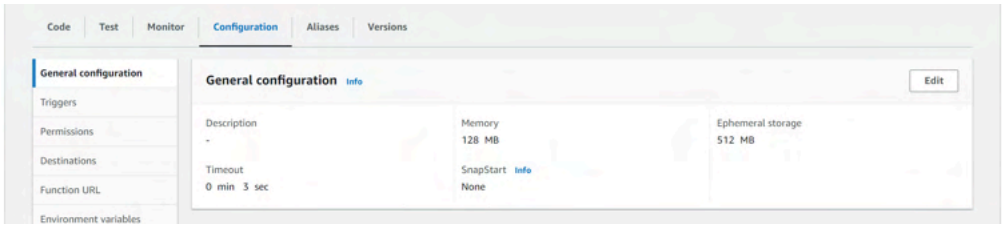
Click on “Create function.”

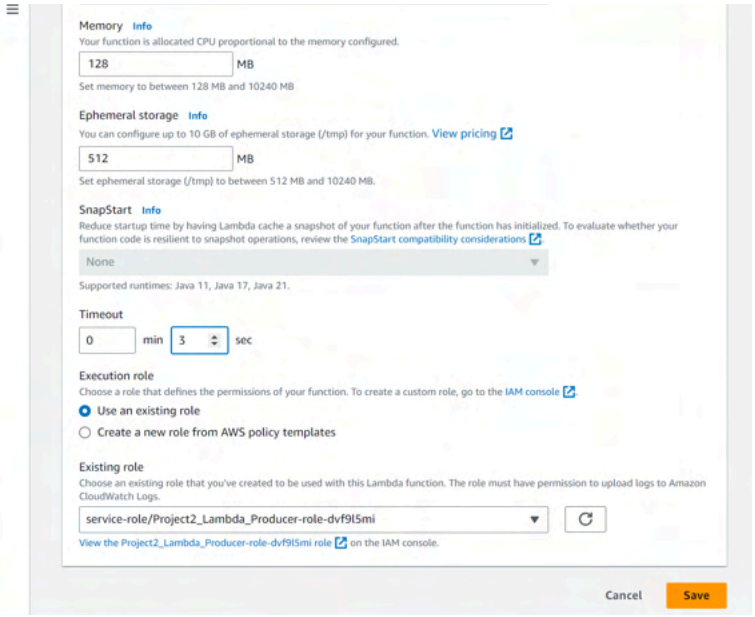


Lambda Function created.

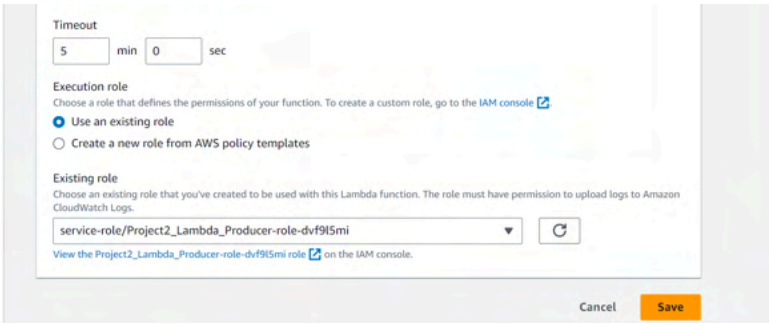
In your function, go to Configuration.

First, go to General Configuration

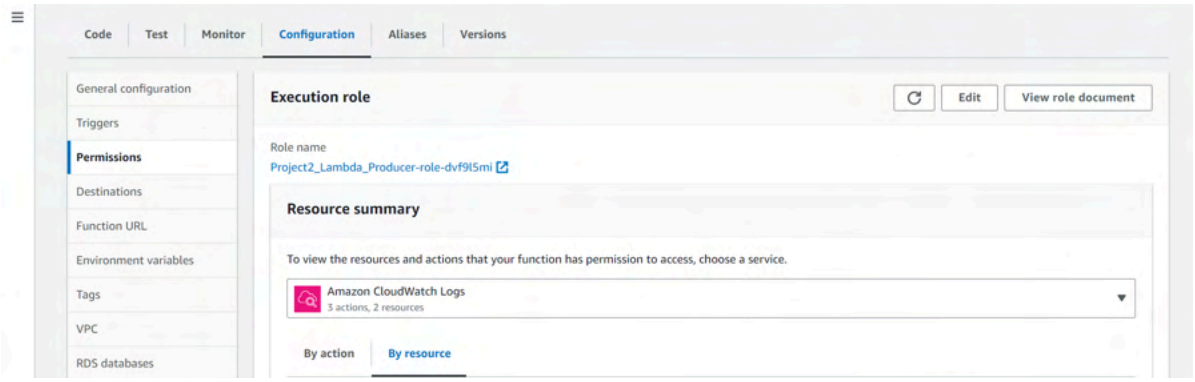




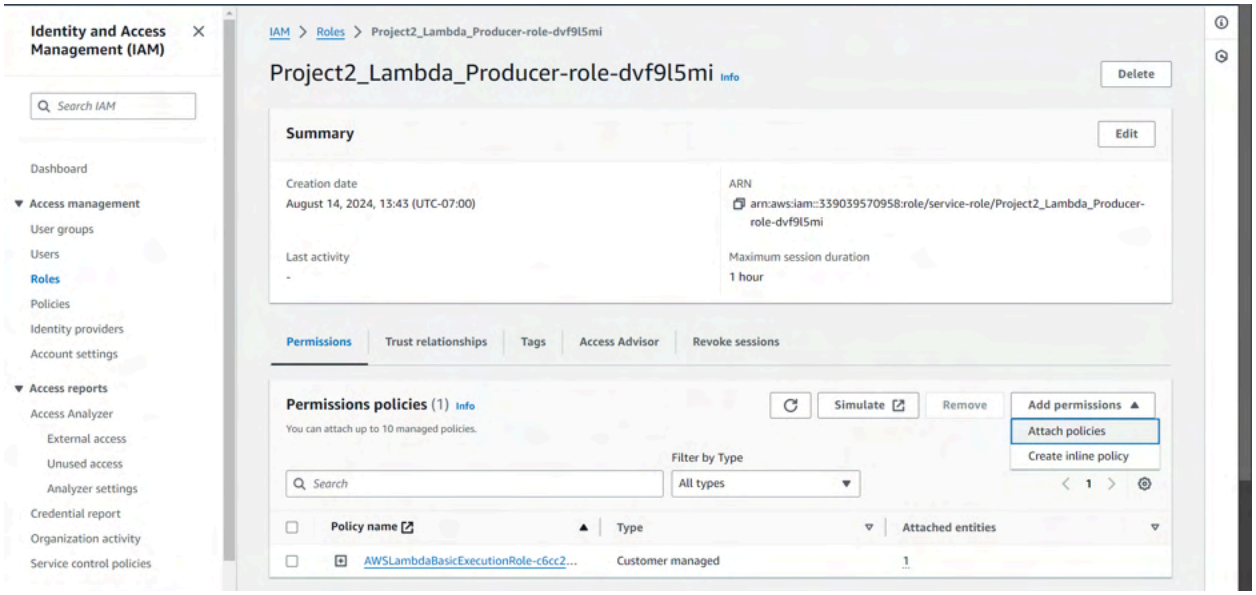
Change the timeout setting from 3 seconds to 5 minutes. This way depending on the data you are streaming, the function won't time out prematurely.



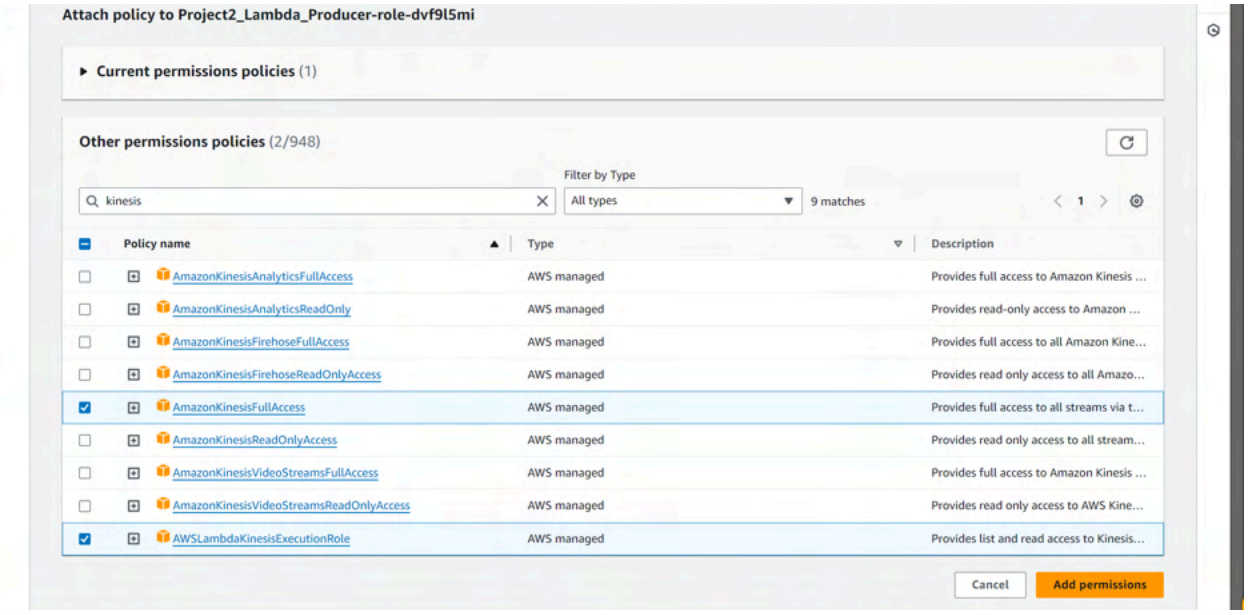
Go to Permissions



Click on role under Role Name. You are sent to your role in IAM: Project2_Lambda_Producer-role-dvf9l5m



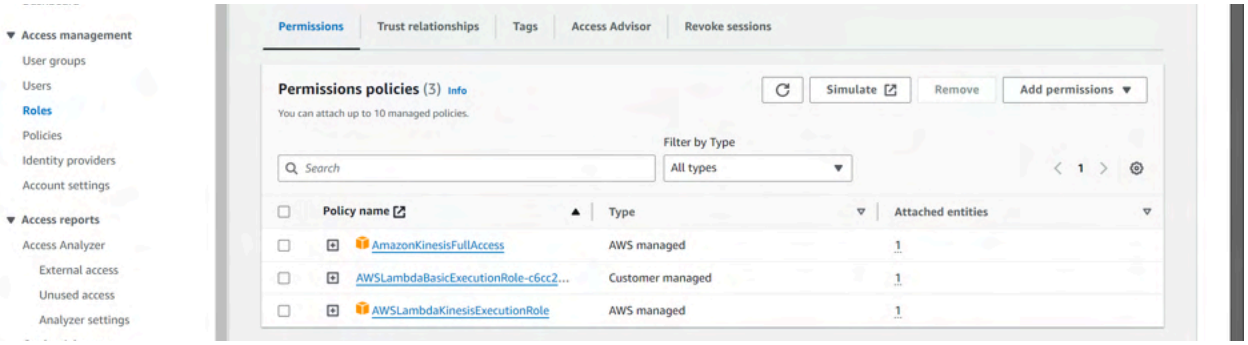
In permission policies, click on Add permissions > Attach policies.



In Add Permissions search for these two policies:

- AmazonKinesisFullAccess
- AWSLambdaKinesisExecutionRole

Click on Add permissions.



Next, you need to add a layer of Python coding to your function. In order to perform Python correctly.

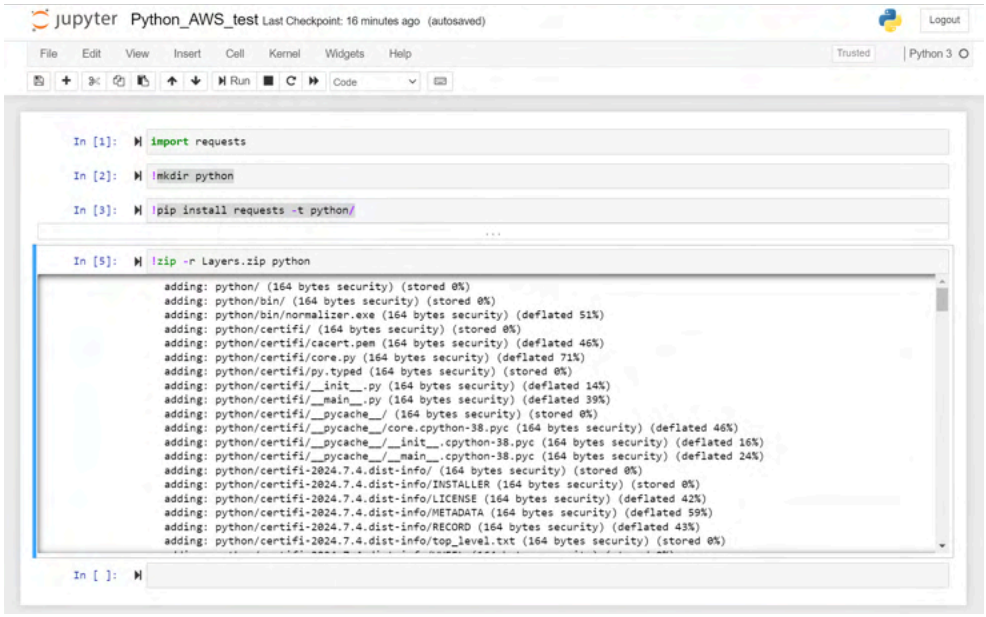
Follow these steps in your command line:

mkdir python #create your directory if you need to

pip install requests -t python/ #install Python Requests, this will provide the files you need

zip -r Layers.zip python #create zip file

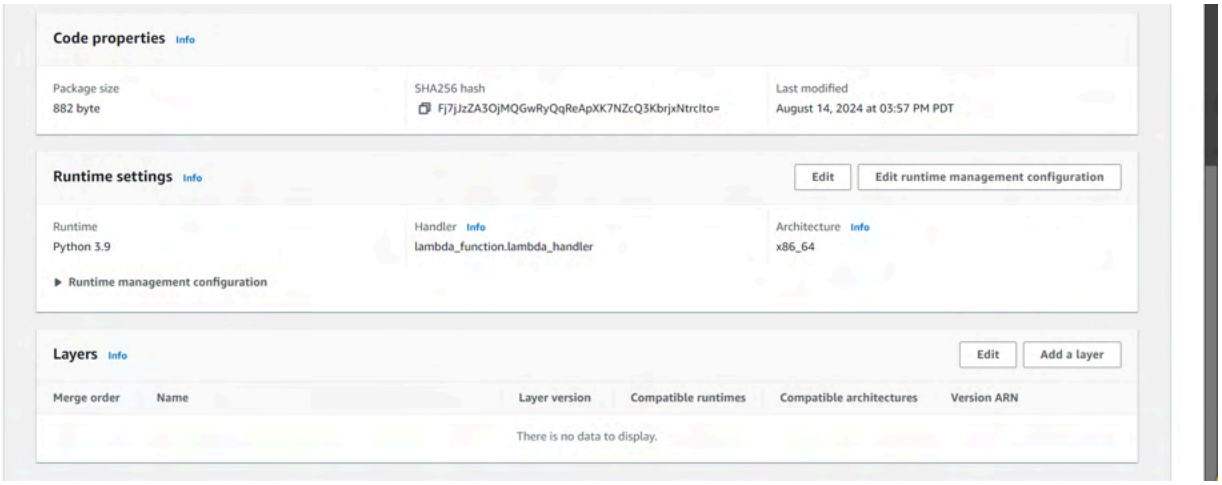
In our case, we used Jupyter Notebook as we already have Python installed.



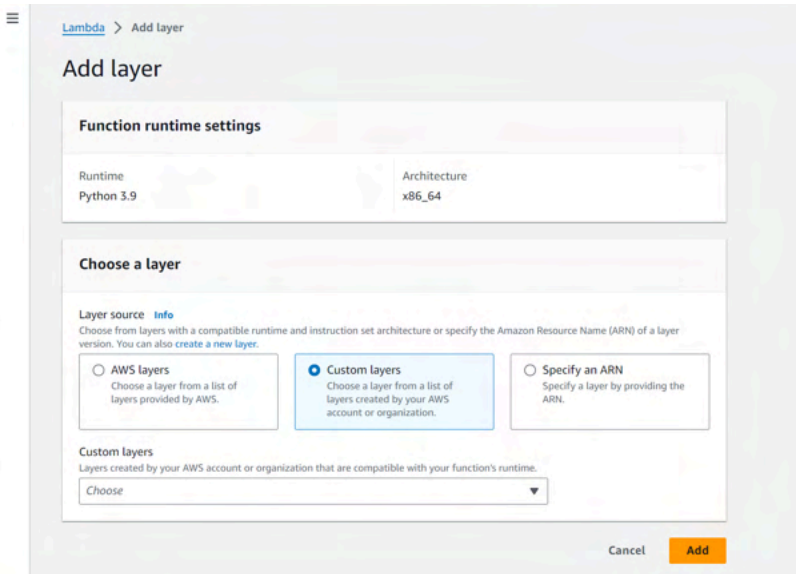
Don’t worry about the error.

Use this zip file for Python layers: Layers.zip

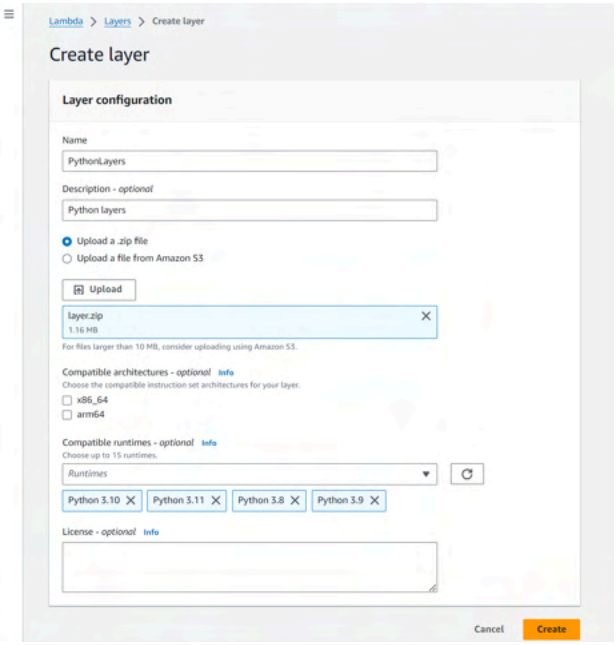
Go to Layers in your function.



Click on Add a layer

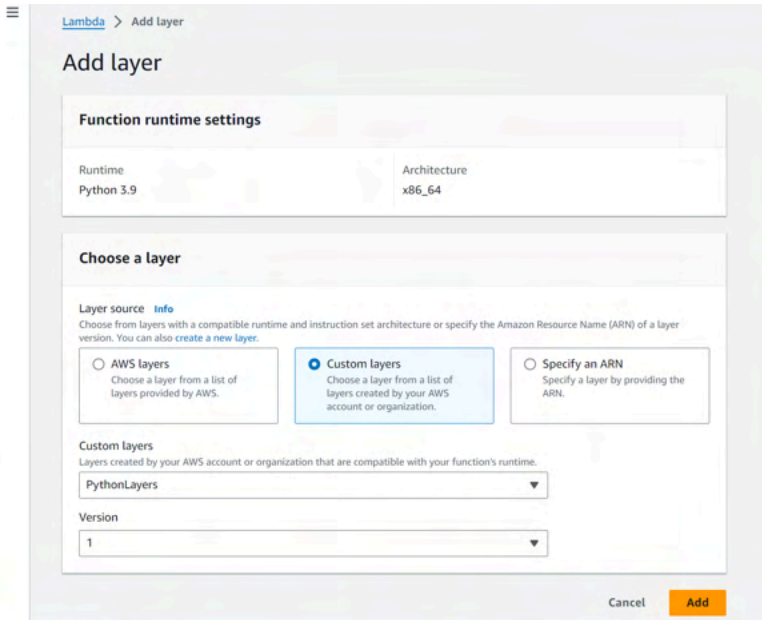


Click on ‘Create a new layer’.



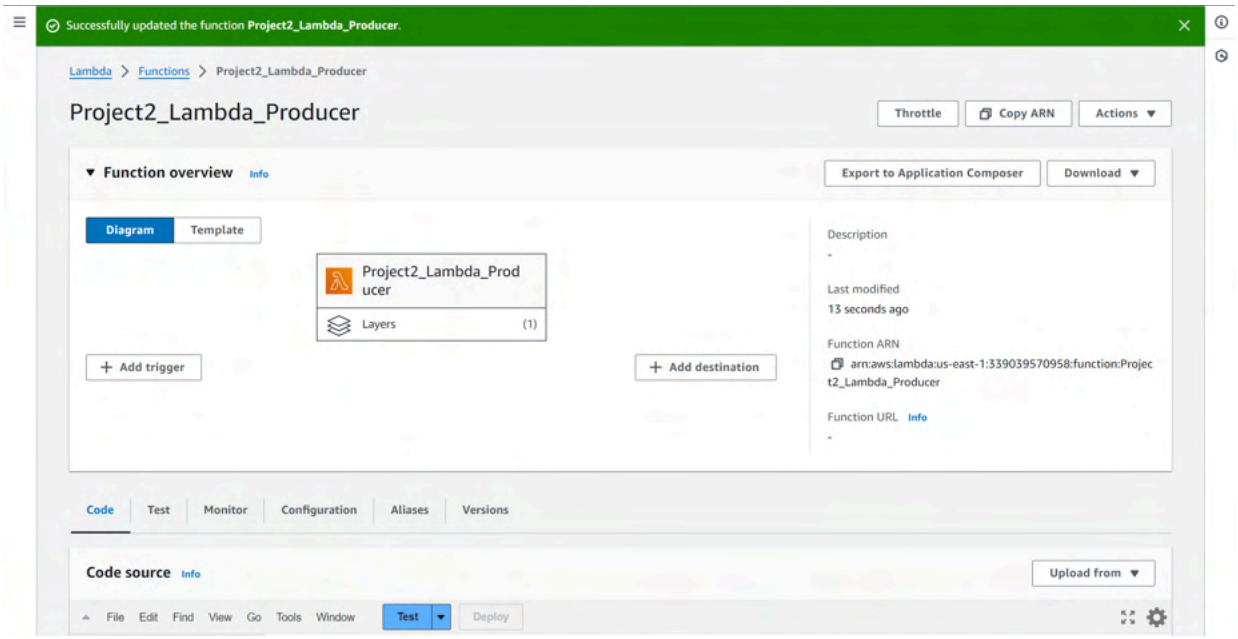
Provide a name for your layers and upload your zip file. Description and Compatible architectures are optional.

Select your compatible runtimes. In this case, select ALL versions of Python. Then click “Create”.



Go back to *Add Layers*. Select custom layers and select the layer you created and the version.

Click “Add”.



Before making your Python code, select a database you will use as your sample test data. In this case, we will use data on the Iris flowers from UC Irvine Machine Learning databases: https://archive.ics.uci.edu/dataset/53/iris

URL LINK: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Enter the following code in Code Source under the lambda_function tab:

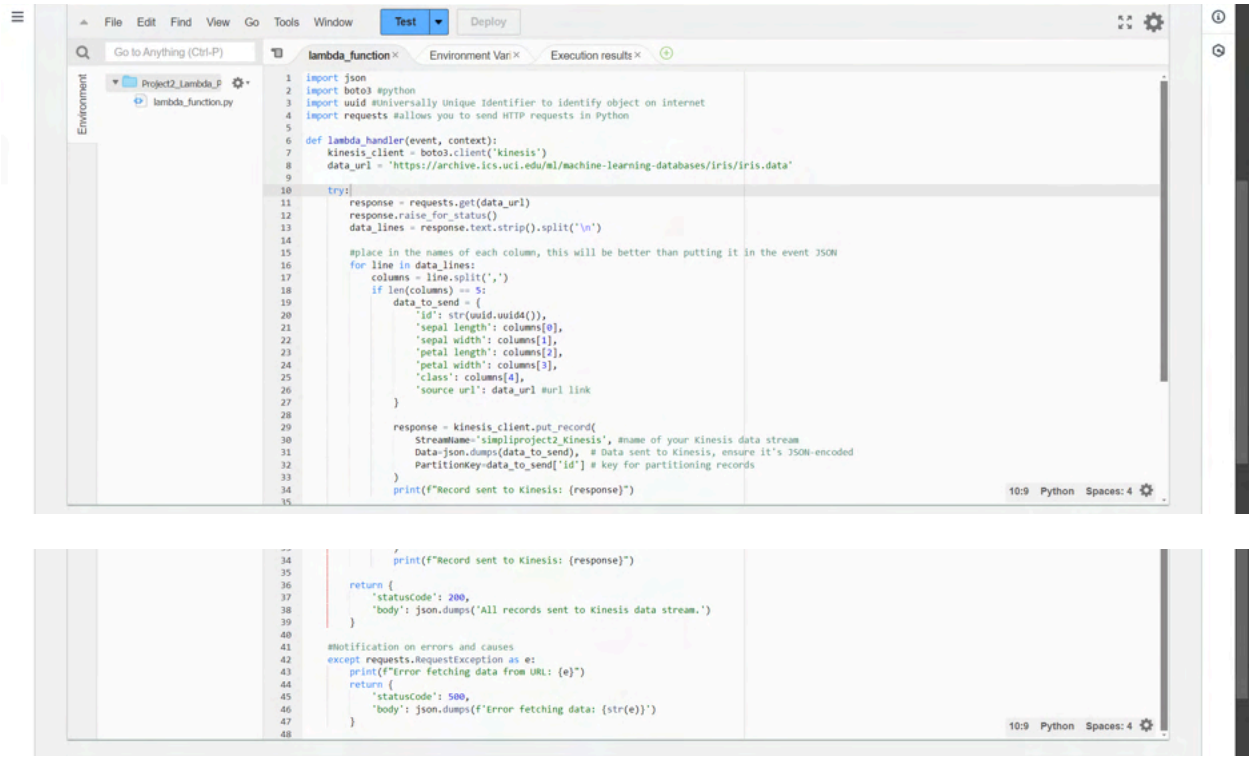
import json

```
import boto3 #python
import uuid #Universally Unique Identifier to identify object on internet
import requests #allows you to send HTTP requests in Python

def lambda_handler(event, context):
    kinesis_client = boto3.client('kinesis')
    data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

    try:
        response = requests.get(data_url)
        response.raise_for_status()
        data_lines = response.text.strip().split('\n')

        #place in the names of each column, this will be better than putting it in the event
JSON
        for line in data_lines:
            columns = line.split(',')
            if len(columns) == 5:
                data_to_send = {
                    'id': str(uuid.uuid4()),
                    'sepal length': columns[0],
                    'sepal width': columns[1],
                    'petal length': columns[2],
                    'petal width': columns[3],
                    'class': columns[4],
                    'source url': data_url #url link
                }

                response = kinesis_client.put_record(
```

```
            StreamName='simpliproject2_Kinesis', #name of your Kinesis data stream
            Data=json.dumps(data_to_send),  # Data sent to Kinesis, ensure it's JSON-encoded
            PartitionKey=data_to_send['id'] # key for partitioning records
        )
        print(f"Record sent to Kinesis: {response}")

    return {
        'statusCode': 200,
        'body': json.dumps('All records sent to Kinesis data stream.')
    }

  #Notification on errors and causes
  except requests.RequestException as e:
    print(f"Error fetching data from URL: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps(f'Error fetching data: {str(e)}')
    }
```

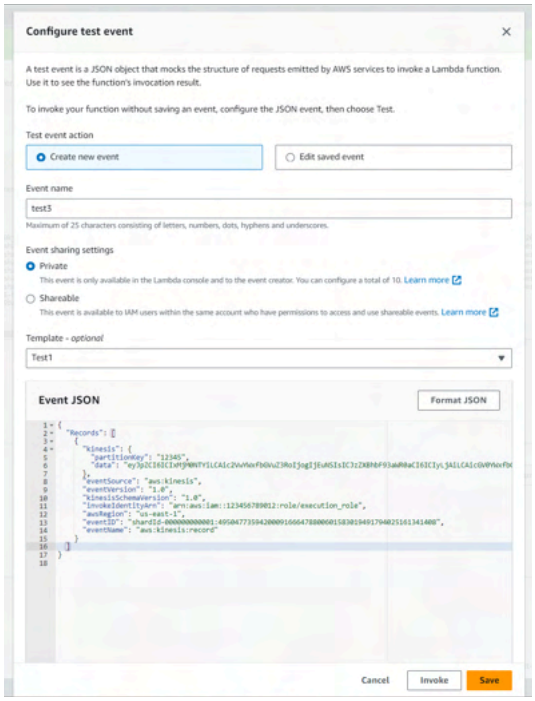
The status code 200 is the HTTP 200 OK, meaning a request has succeeded and was successful.

The status code 500 is HTTP 500 Internal Server Error server error response code. This indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.



Click Deploy.

In Test, click on configure test event.

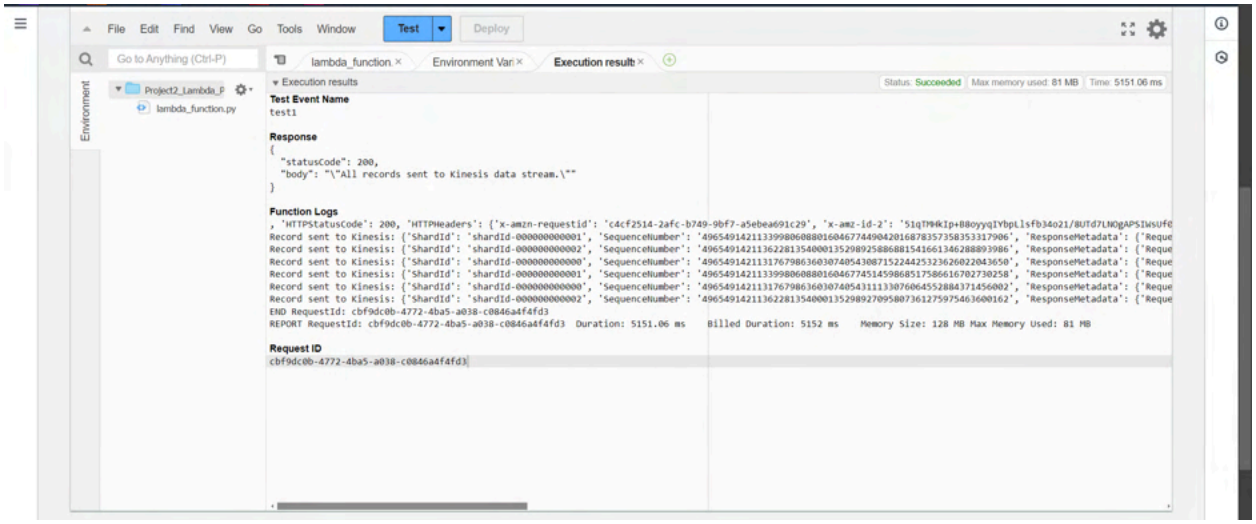


Give a name to your test. In this case, “test3” (multiple tests were used as trial runs).

To ensure that the data will be sent to the DynamnoTable:

```
{
  "Records": [
    {
      "kinesis": {
        "partitionKey": "12345",
        "data":
"eyJpZCI6ICIxMjM0NTYiLCAic2VwYWxfbGVuZ3RoIjogIjEuNSIsICJzZXBhbF93aWR0aCI6ICIyLjAiLCAicGV0YWxfbGVuZ3RoIjogIjEuMCIsICJwZXRhbF93aWR0aCI6ICIwLjAiLCAiY2xhc3MiOiAiU29tZSBDbGFzcyIsICJzb3VyY2VfdXJsIjogImh0dHBzOi8vYXJjaGl2ZS5pY3MuZWN1LmVkdS9tbC9tYWNoaW5lLWxlYXJuaW5nLWRhdGFzYmFzZXMvaXJpcy9pcnMuZGF0YSJ9"
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "kinesisSchemaVersion": "1.0",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/execution_role",
      "awsRegion": "us-east-1",
      "eventID":
"shardId-000000000001:49504773594200091666478800601583019491794025161341408",
      "eventName": "aws:kinesis:record"
    }
  ]
}
```


This took multiple trials. Do not be worried to experiment.

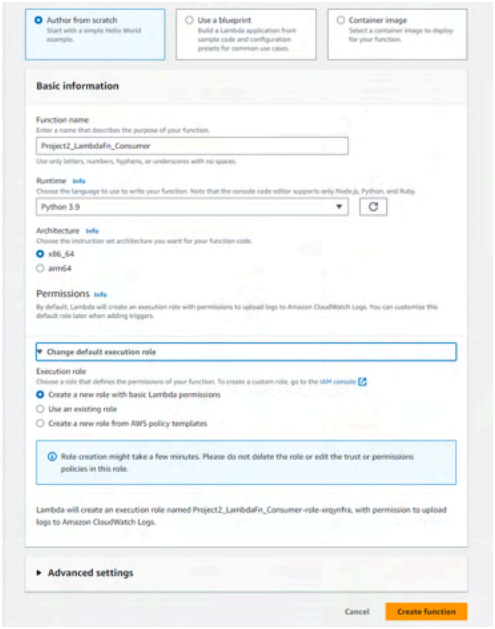


Same successful event. Move on to the next Lambda Function.

2.2 Lambda Consumer

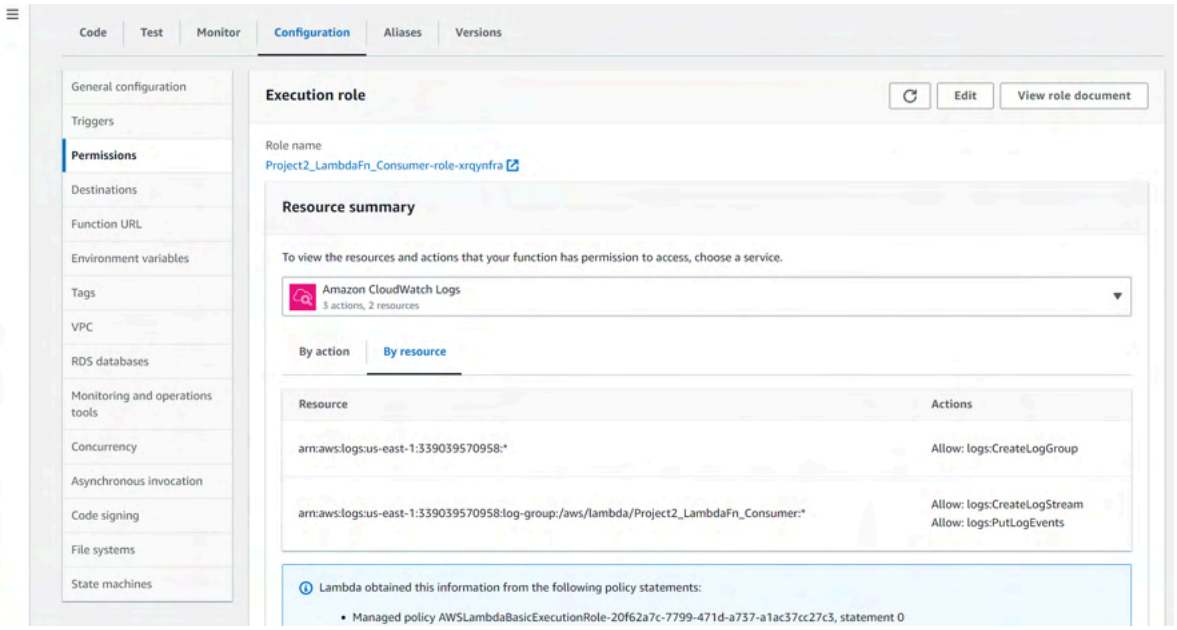
Repeat the same steps in creating the lambda function but have it be named “Project2_LambdaFn_Consumer”.

This is your consumer lambda function that will be connected to your DynamnoDB table and Kinesis trigger.



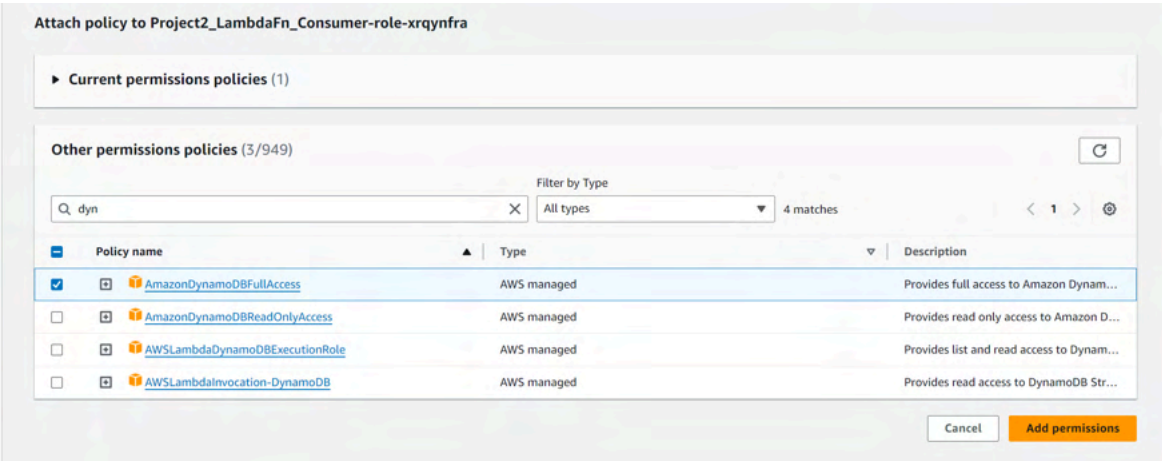
Adjust the timeout to 5 minutes as well.

Then go to your Lambda function Role: Project2_LambdaFn_Consumer-role-xrqynfra.



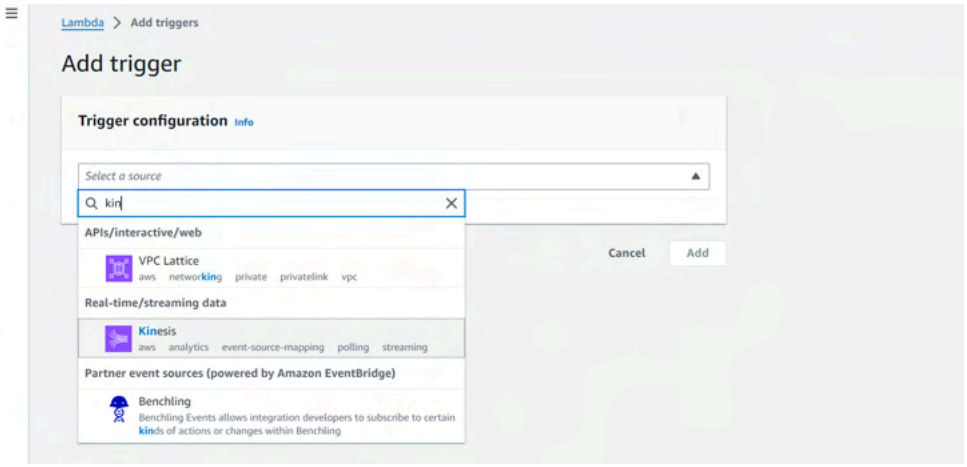
Add these policies to your consumer function:

- AmazonKinesisFullAccess
- AWSLambdaKinesisExecutionRole
- AmazonDynamoDBFullAccess

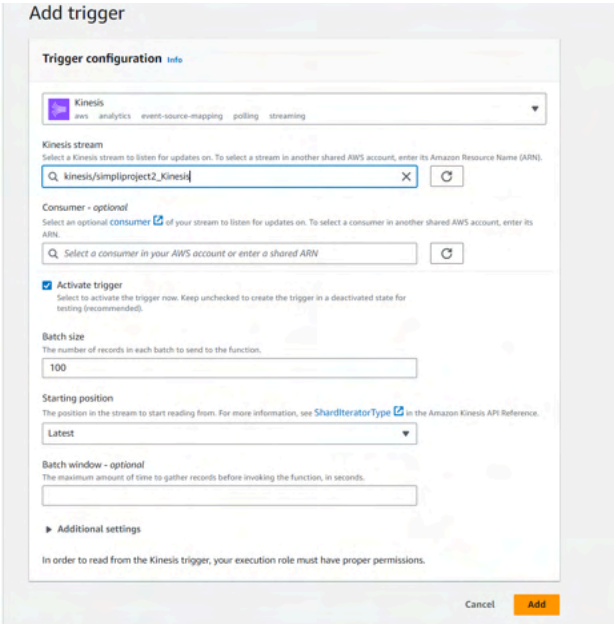


Add permissions. Then return to Lambda consumer function.

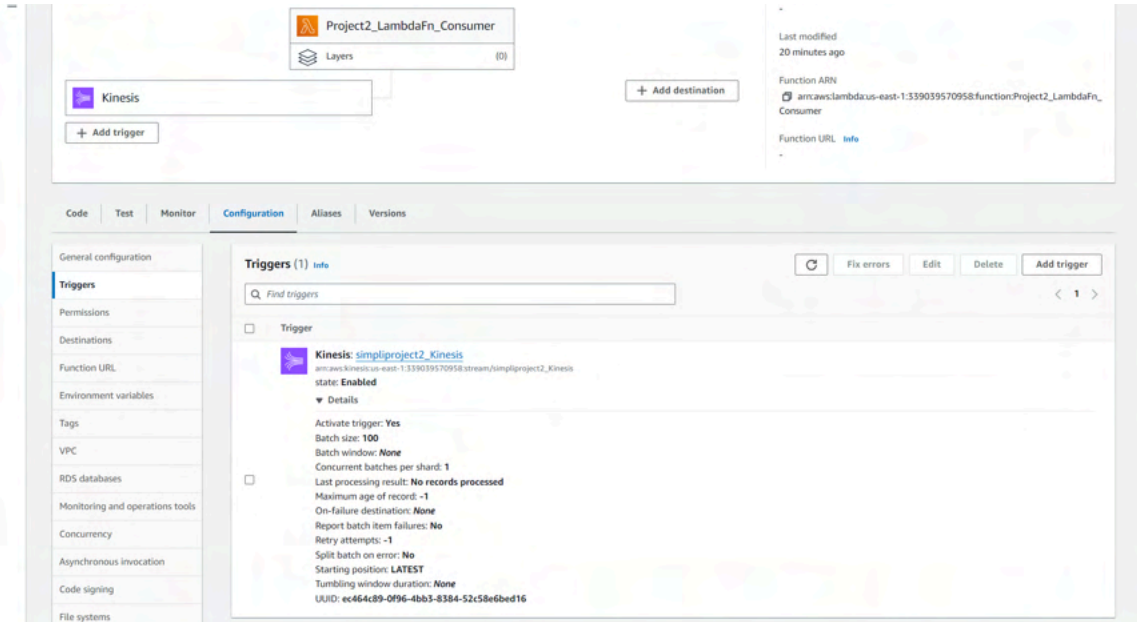
Go to Add Trigger.



Type in Kinesis.



Select your data stream “simpliproject2_Kinesis”. Then click on Add.

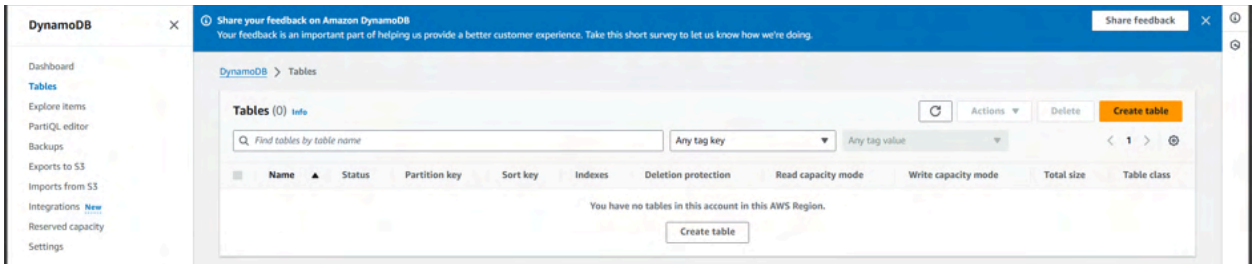


Before continuing, we recommend you make your DynamoDB table first then enter the code for consumer function to test your data.

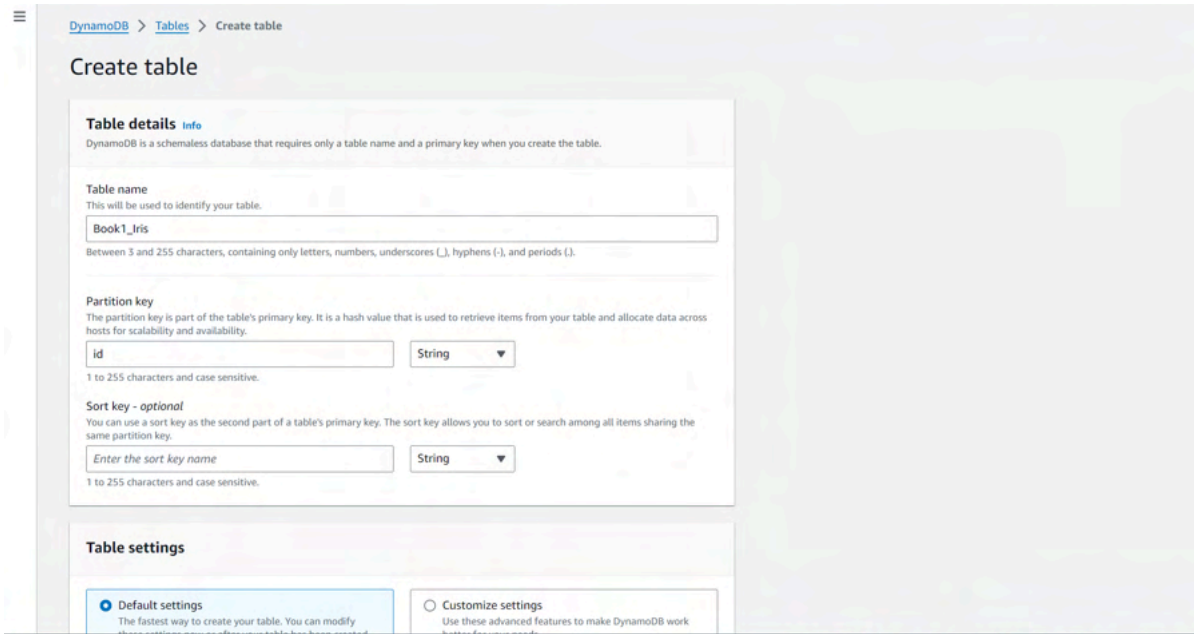
Step 3: Create DynamoDB table

3.1 Create Table

Go to DynamoDB



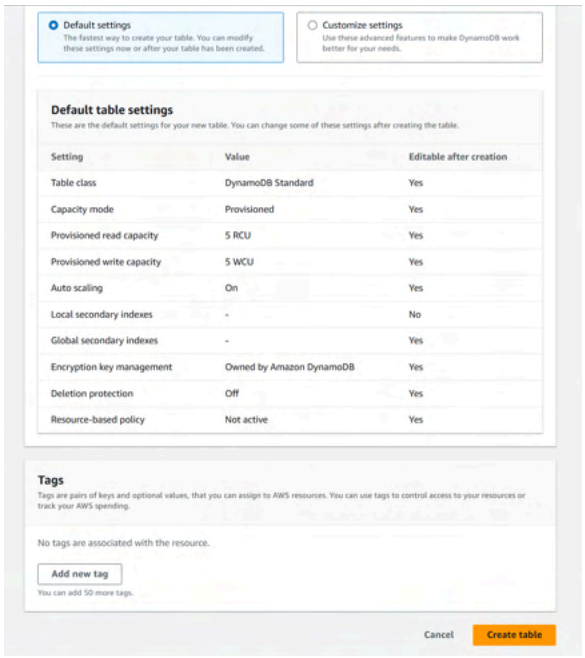
Go to Tables. Click on Create Table.



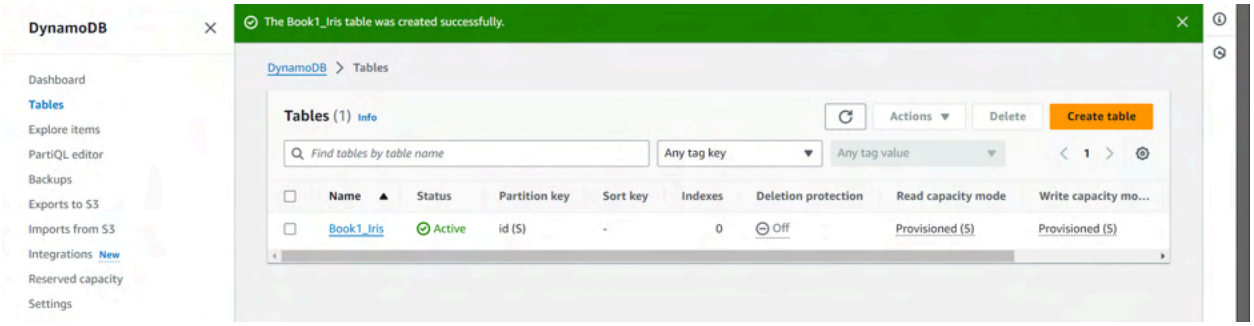
Enter the desired Table name and Partition key. Ignore sort key.

For me: Table name is ‘Book1_Iris’ and Partition Key is ‘id’.

NOTE: BE SURE your partition key matches with the first name column in your data set.



Click on Create table.



DynamoDB created.

3.2 Lambda Consumer Code

Return to your consumer lambda function.

```
import json
import boto3
import base64 #built-in module in Python
from botocore.exceptions import ClientError
#catching botocore exception

# Initialize DynamoDB client
dynamodb_client = boto3.resource('dynamodb')
table = dynamodb_client.Table('Book1_Iris')  #DynamoDB table name

def lambda_handler(event, context):
    for record in event.get('Records', []):
        try:
            # Decode Kinesis data from Base64
            encoded_data = record['kinesis']['data']
            decoded_data = base64.b64decode(encoded_data)
            package = json.loads(decoded_data)

            # Extract data fields
            id = package.get('id')
```

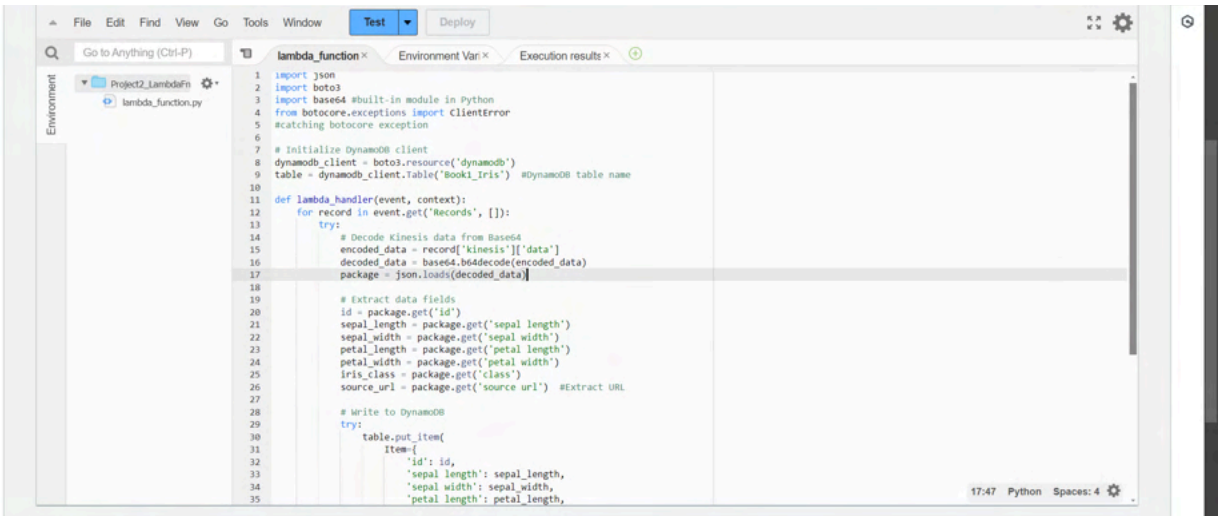
```
        sepal_length = package.get('sepal length')
        sepal_width = package.get('sepal width')
        petal_length = package.get('petal length')
        petal_width = package.get('petal width')
        iris_class = package.get('class')
        source_url = package.get('source url')  #Extract URL

        # Write to DynamoDB
        try:
            table.put_item(
                Item={
                    'id': id,
                    'sepal length': sepal_length,
                    'sepal width': sepal_width,
                    'petal length': petal_length,
                    'petal width': petal_width,
                    'class': iris_class,
                    'source url': source_url  # Store URL in DynamoDB
                }
            )
            print(f"Record wrote with id: {id}")

        except ClientError as e:
            print(f"Error writing record to DynamoDB: {e}")

    except Exception as e:
        print(f"Error processing record: {e}")

return {
```

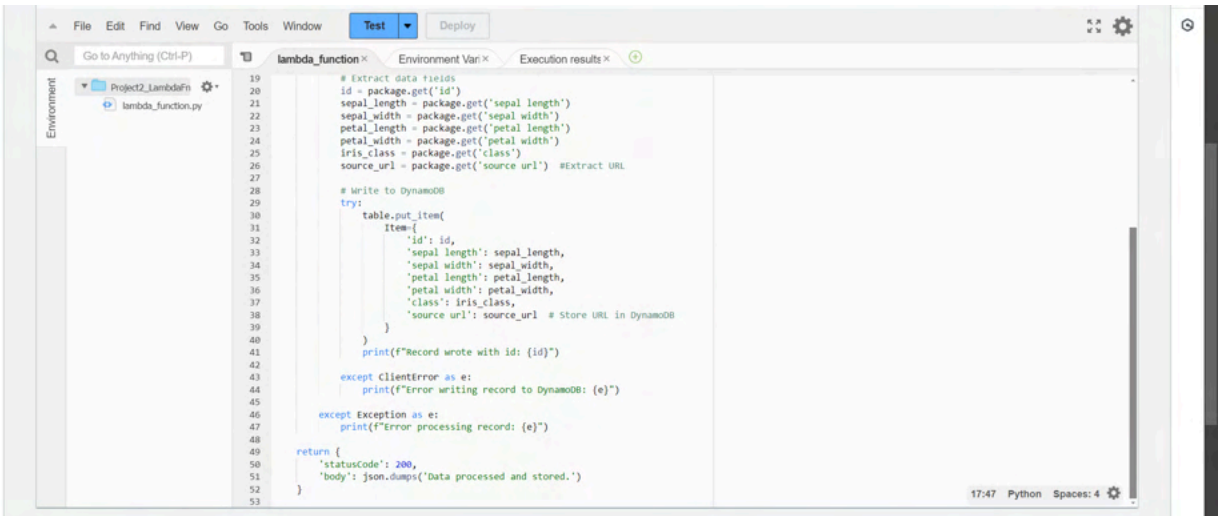
'statusCode': 200,

'body': json.dumps('Data processed and stored.')

}

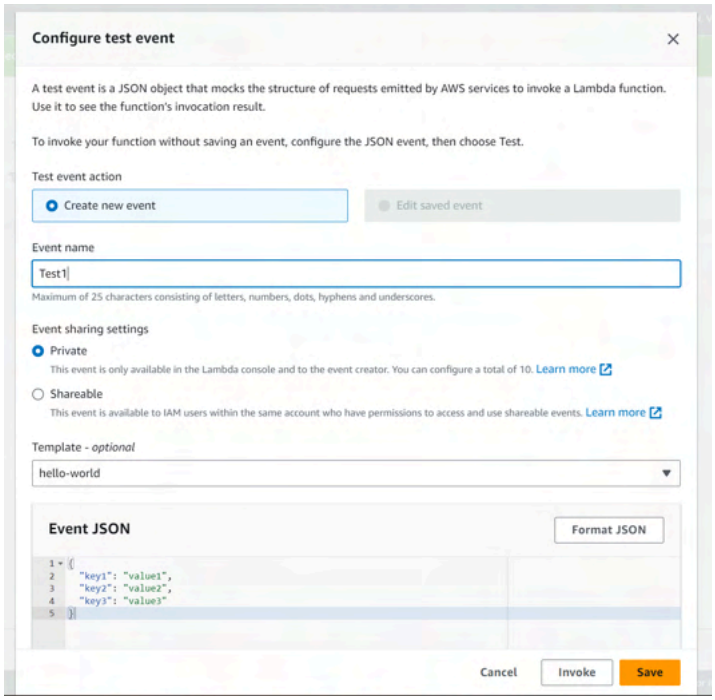
The status code 200 is the HTTP 200 OK, meaning a request has succeeded.



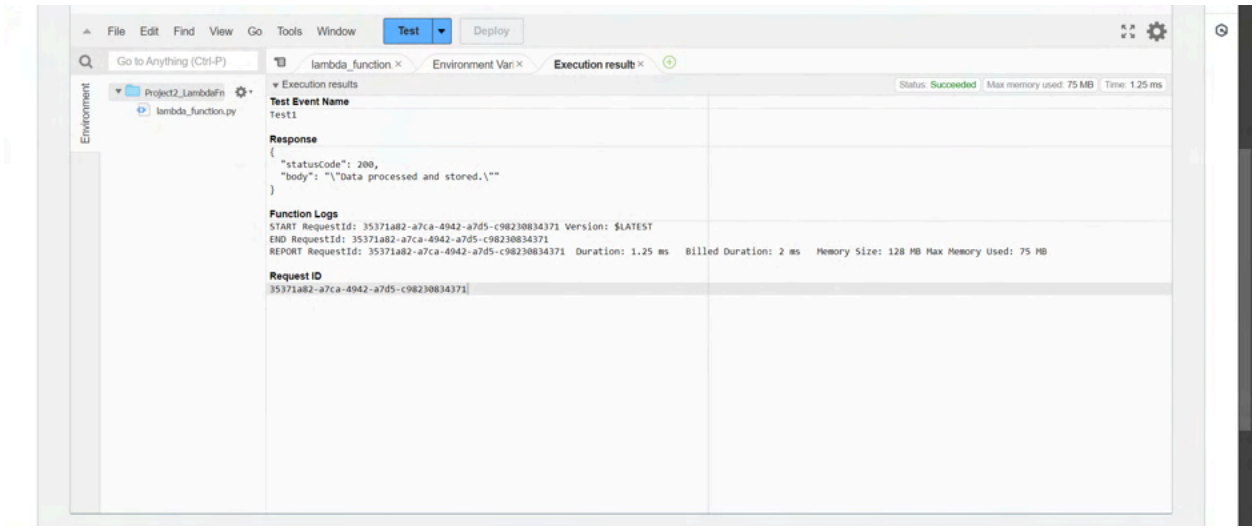


Press Deploy.

For Test, create a test name. Leave the JSON alone.



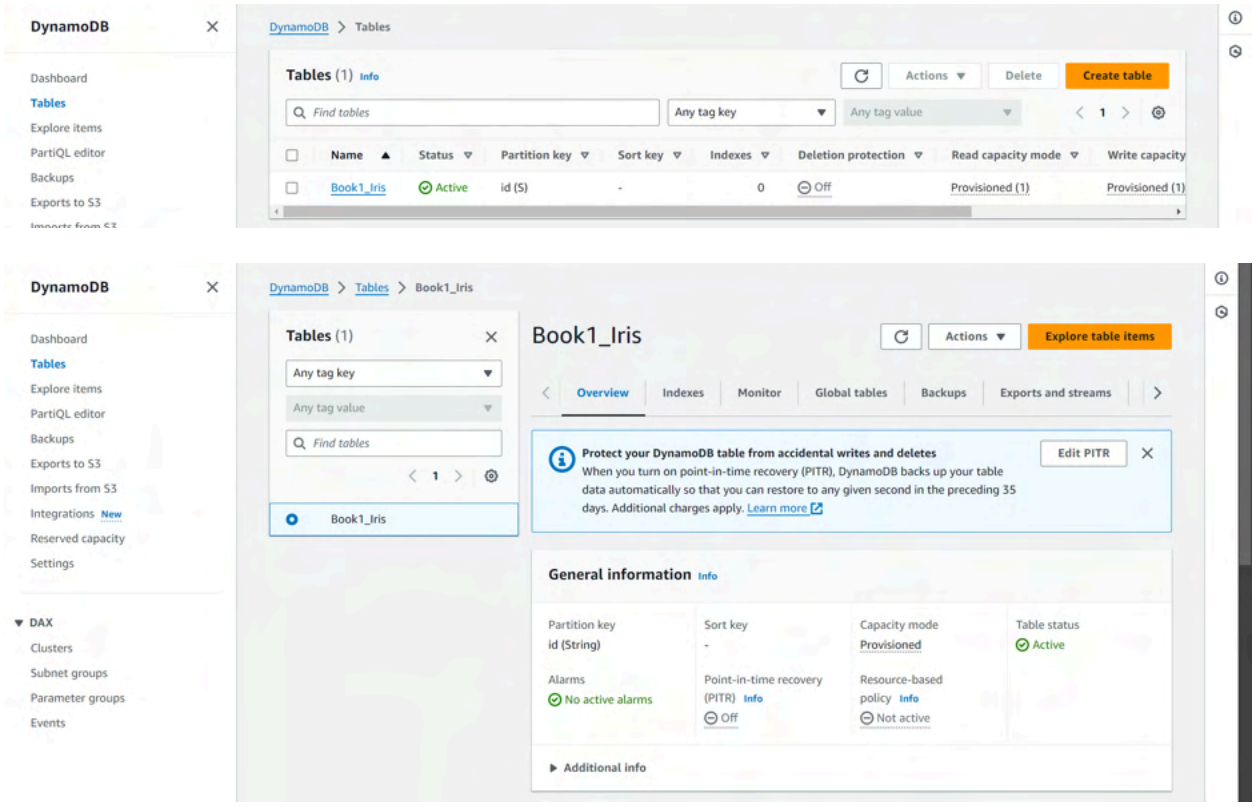
Run Test.



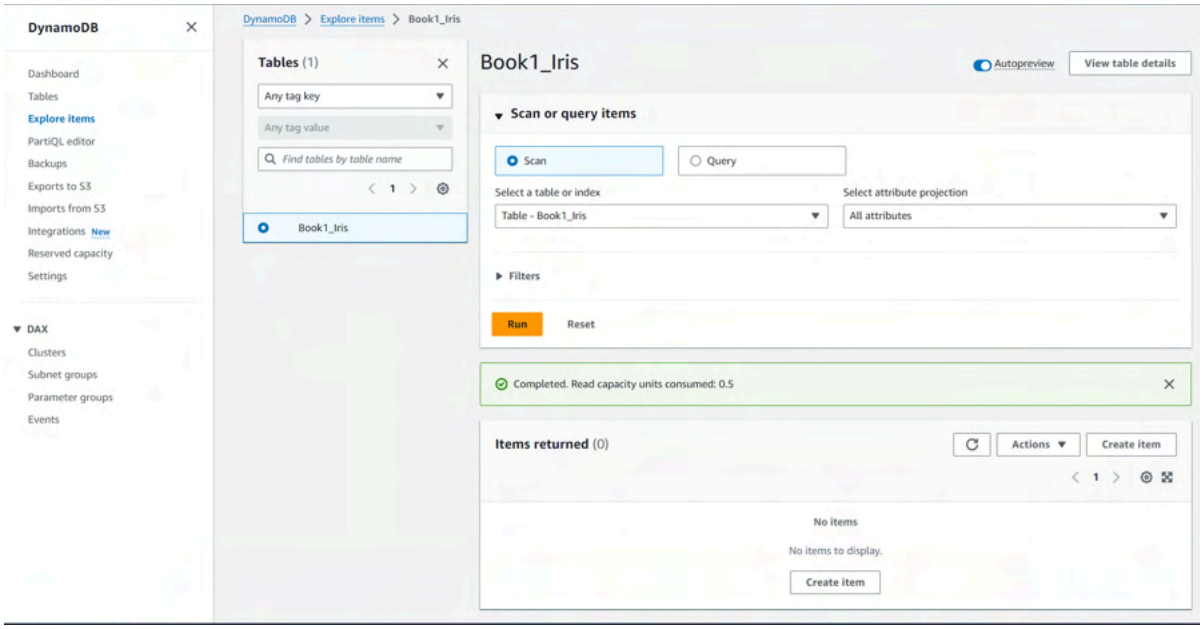
Test successful.

3.3 Upload Data to Table

Go to your DynamoDB table.

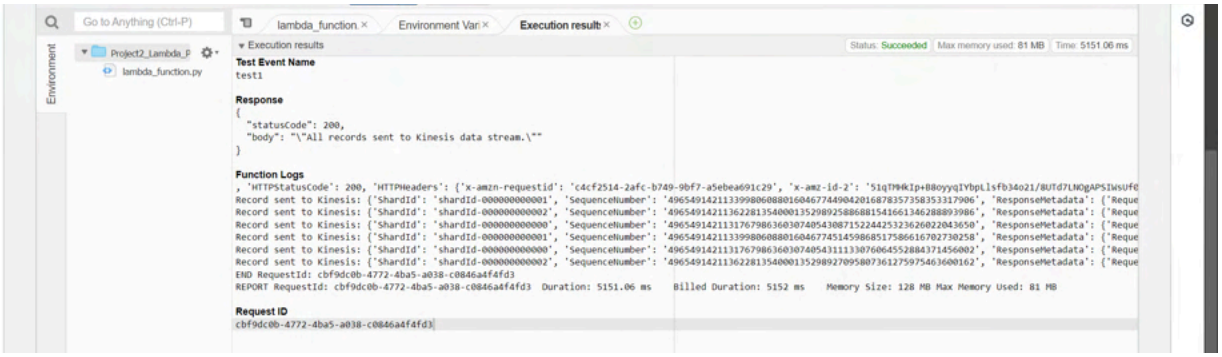


Click on ‘Explore table items’.



You will see **Items returned (0)**.

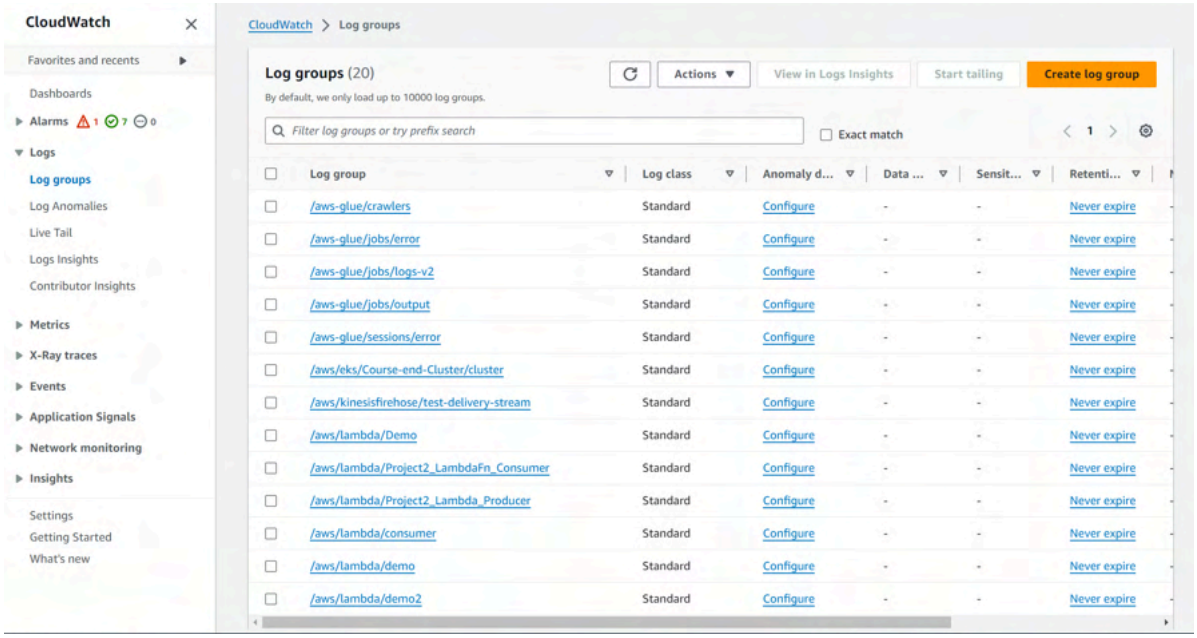
Go to your Lambda producer function. Click on Test, be sure it is set your test event you want.



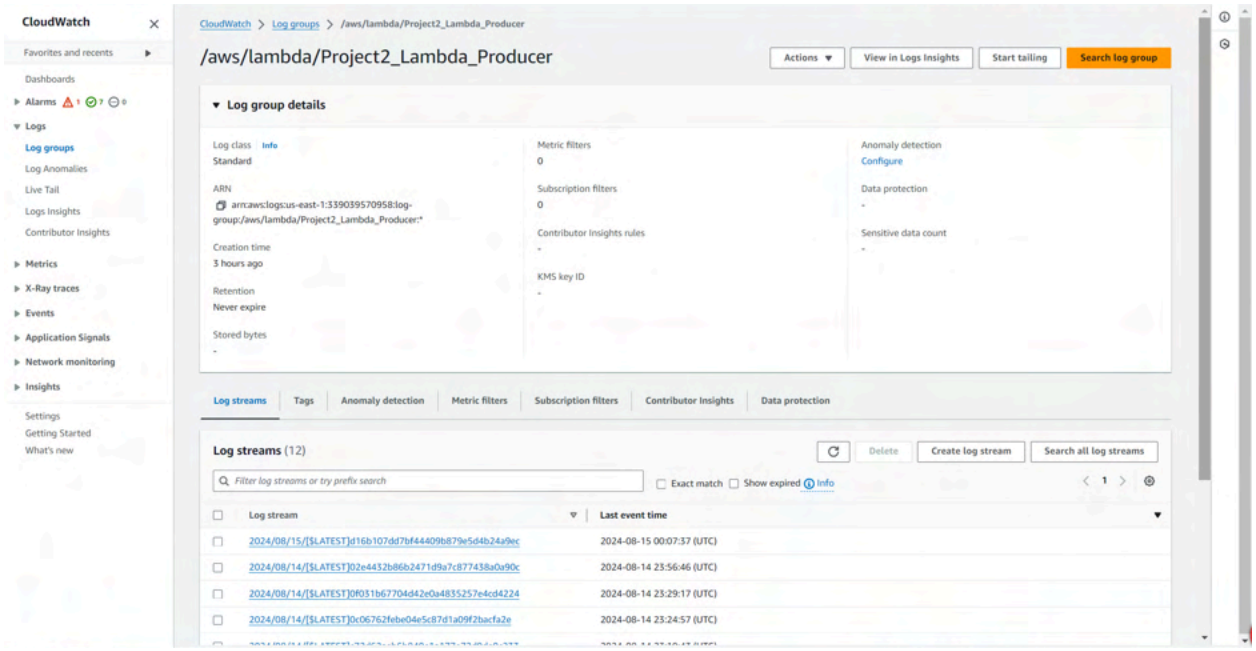
Results indicate that data has been sent to your DynamoDB Table.

To check, go to *CloudWatch*.

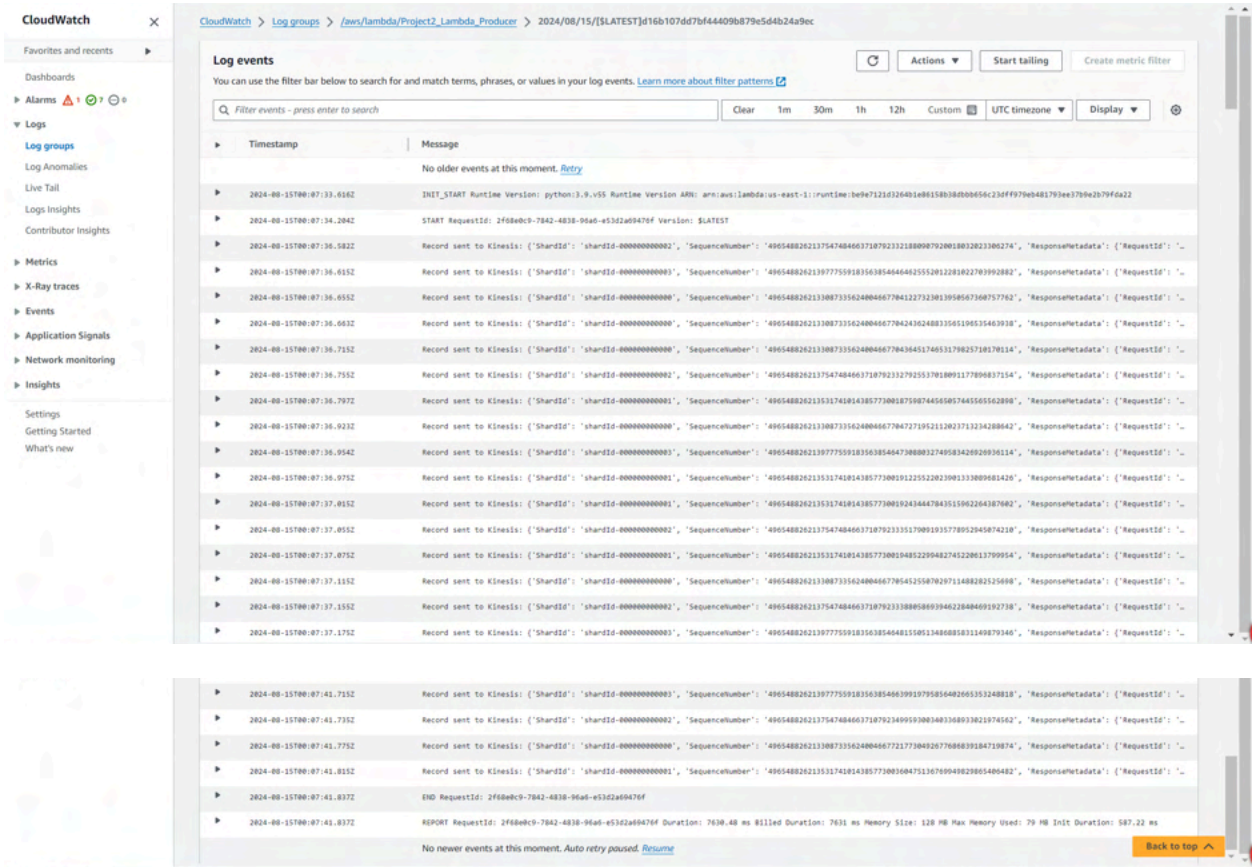
Go to Log Groups.



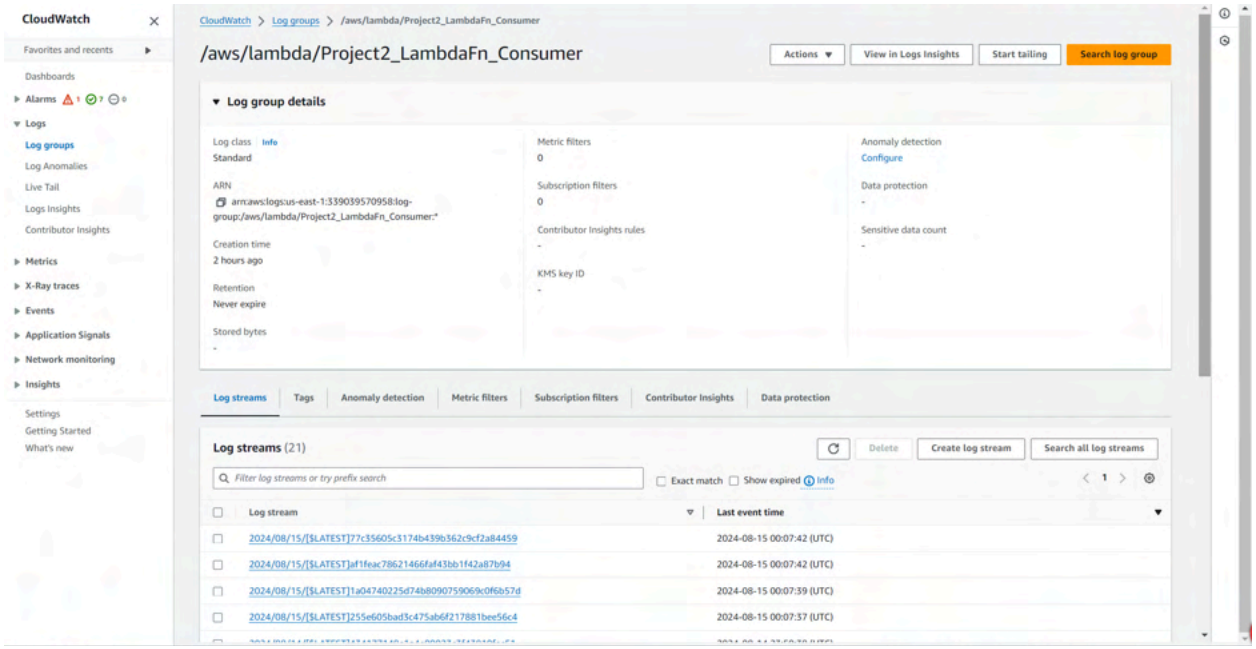
For Lambda Producer



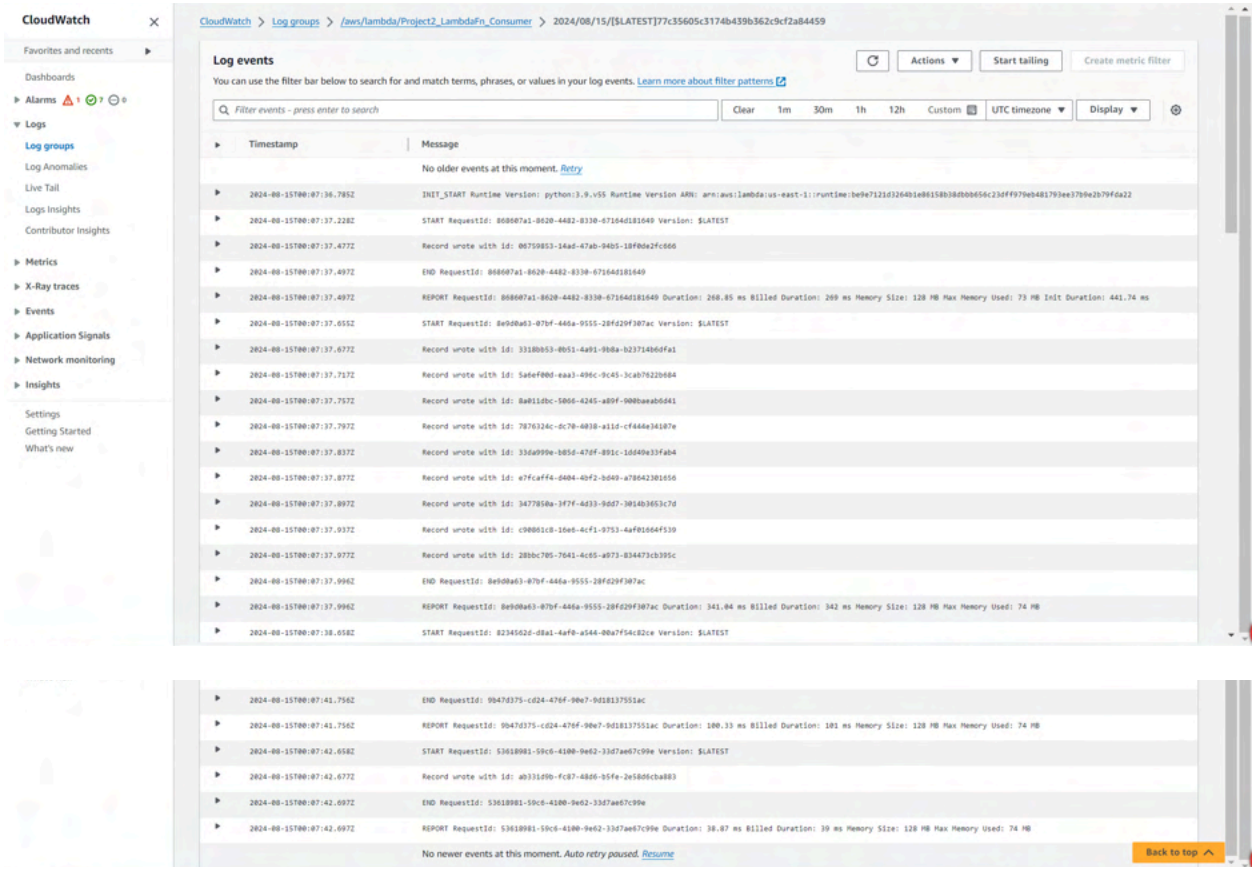
Click on the latest log stream.



Now check your Lambda Consumer.

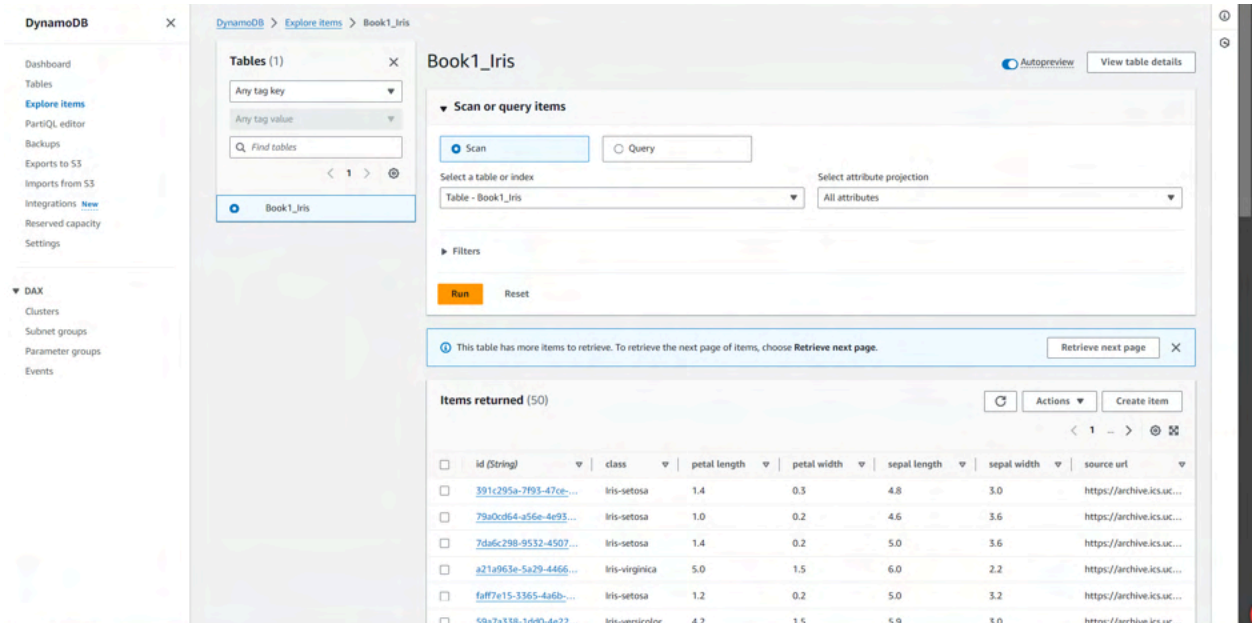


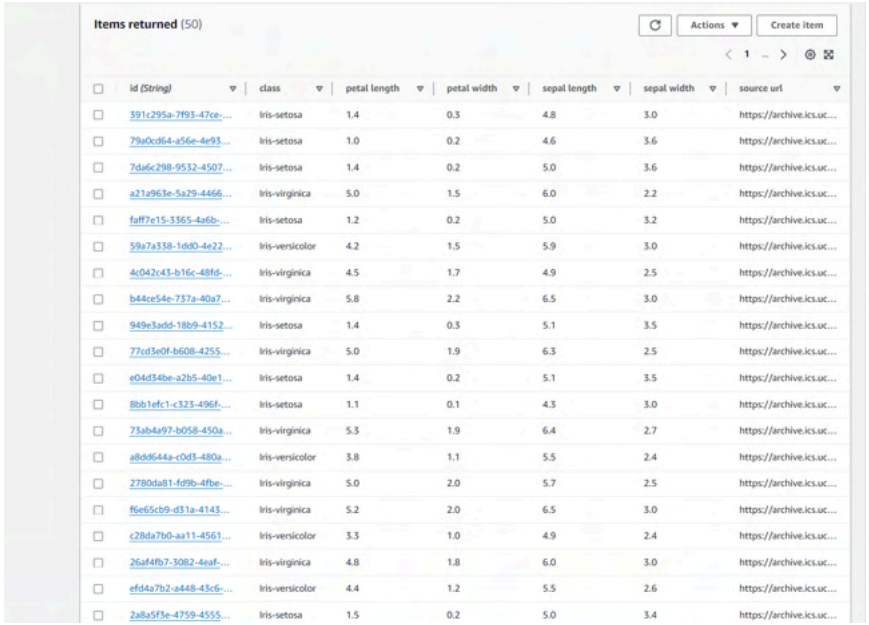
Click on the latest log stream.



Cloud indicates data stream went through.

Now go back to DynamoDB Table.





Data has been successfully. Uploaded.

Step 4: Perform Scan Operation

4.1 Scan Option

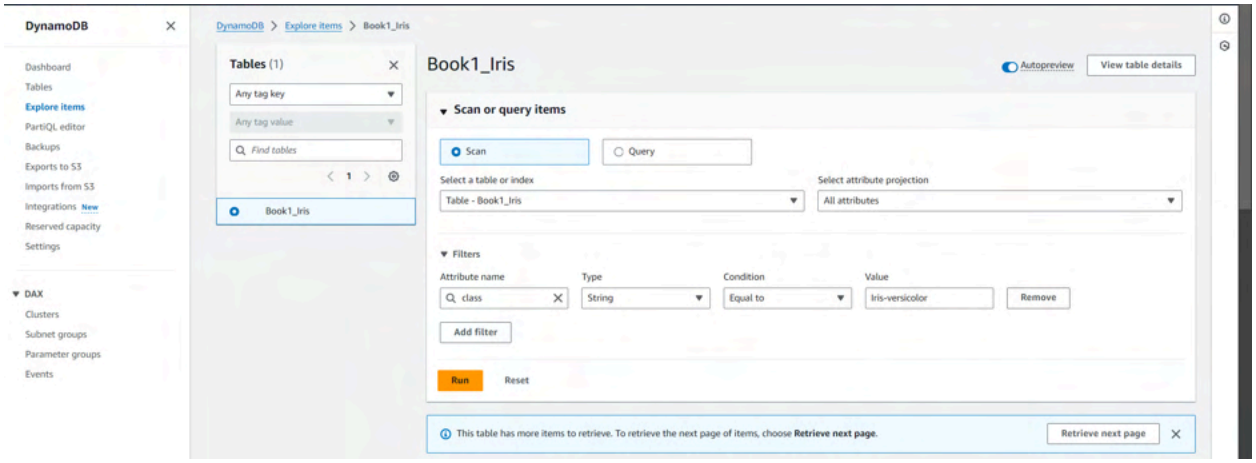
From the upload data, test Scan & Query options.

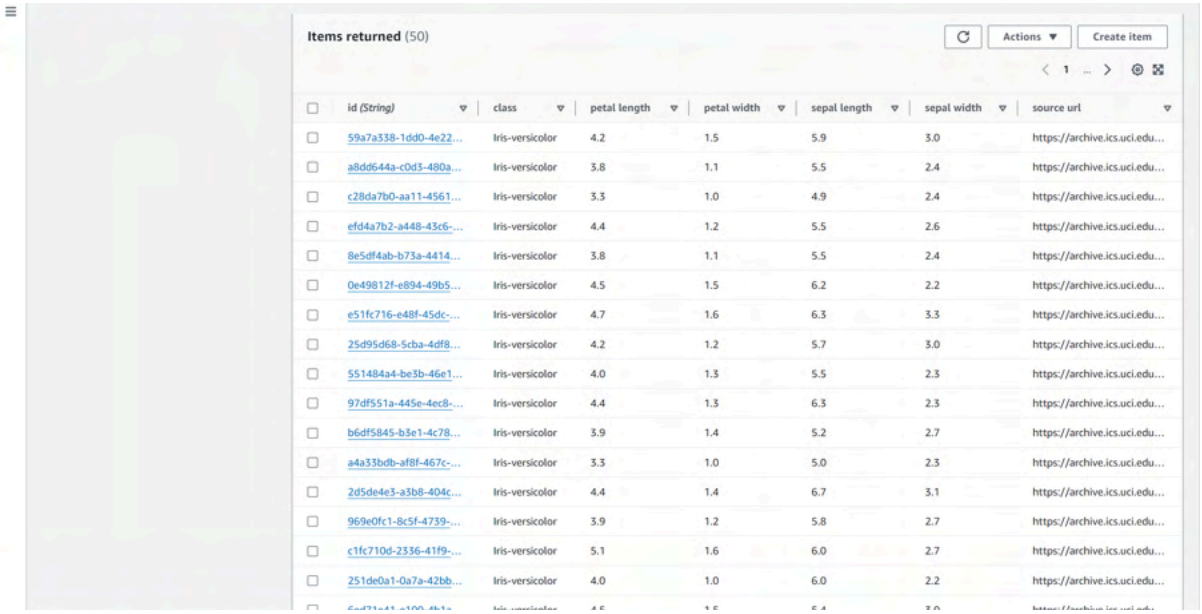
In “Scan or query items”, Select Scan.

Select Filter. With this option, you can filter out specific items in your data table.

First trial, have attribute name be “class”. Type as String, condition to “equal to”, and value put in “Iris-versicolor”.

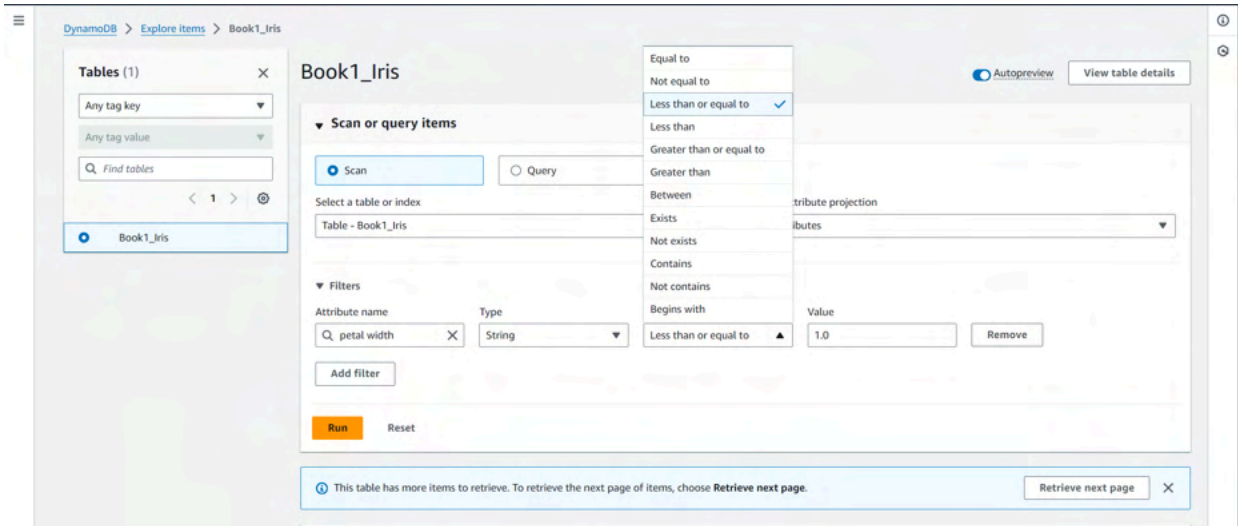
Click on Run.



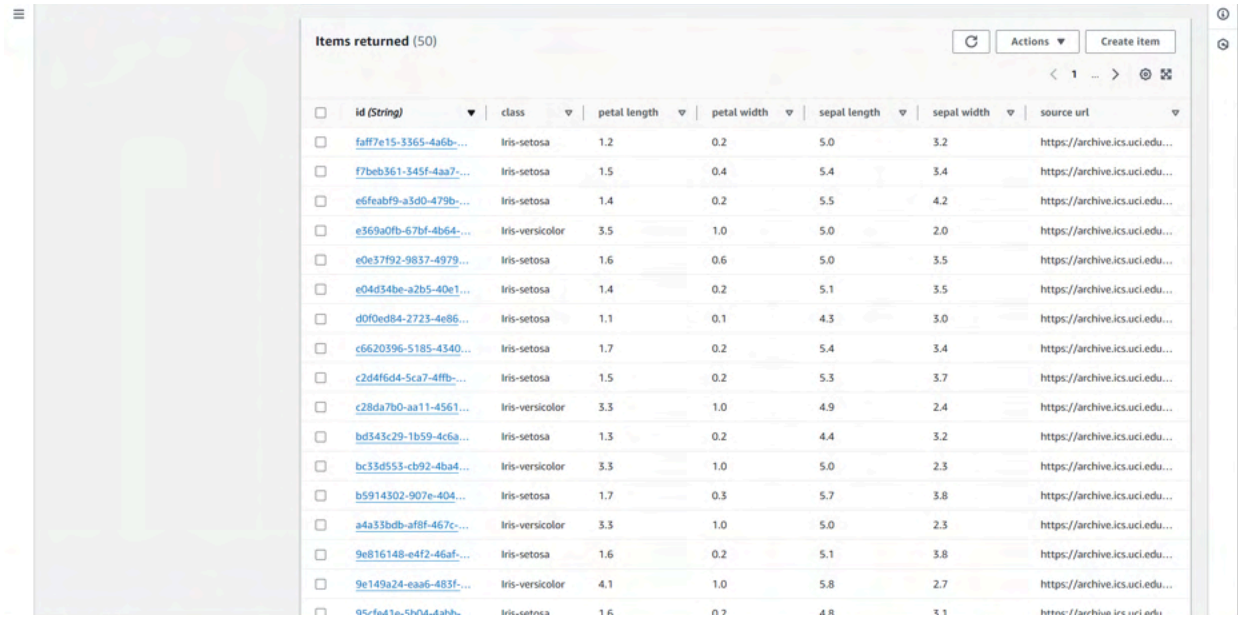


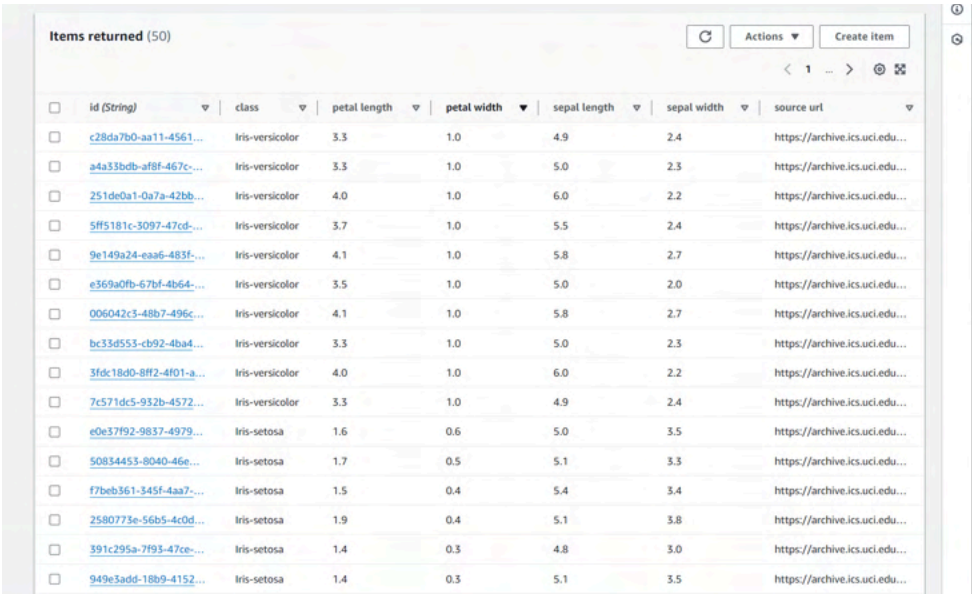
Your data is filtered to a specific class of irises based on your request.

Now try, attribute “petal width” and have condition set to “less than or equal to” at 1.0.


DynamoDB > Explore items > Book1_Iris
Tables (1)
Any tag key
Any tag value
Find tables
Book1_Iris
Book1_Iris
Autopreview
View table details
Scan or query items
Scan
Query
Select a table or index
Table - Book1_Iris
Filters
Attribute name
petal width
Type
String
Value
1.0
Remove
Add filter
Run
Reset
This table has more items to retrieve. To retrieve the next page of items, choose Retrieve next page.
Retrieve next page
Equal to
Not equal to
Less than or equal to
Less than
Greater than or equal to
Greater than
Between
Exists
Not exists
Contains
Not contains
Begins with

Click on Run.


Items returned (50)
Actions
Create item
id (String)
class
petal length
petal width
sepal length
sepal width
source url
faff7e15-3365-4a6b-...
Iris-setosa
1.2
0.2
5.0
3.2
https://archive.ics.uci.edu...
f7beb361-345f-4aa7-...
Iris-setosa
1.5
0.4
5.4
3.4
https://archive.ics.uci.edu...
e6feabf9-a3d0-479b-...
Iris-setosa
1.4
0.2
5.5
4.2
https://archive.ics.uci.edu...
e369a0fb-67bf-4b64-...
Iris-versicolor
3.5
1.0
5.0
2.0
https://archive.ics.uci.edu...
e0e37f92-9837-4979-...
Iris-setosa
1.6
0.6
5.0
3.5
https://archive.ics.uci.edu...
e04d34be-a2b5-40e1-...
Iris-setosa
1.4
0.2
5.1
3.5
https://archive.ics.uci.edu...
d0f0ed84-2723-4e86-...
Iris-setosa
1.1
0.1
4.3
3.0
https://archive.ics.uci.edu...
c6620396-5185-4340-...
Iris-setosa
1.7
0.2
5.4
3.4
https://archive.ics.uci.edu...
c2d4f6d4-5ca7-4ffb-...
Iris-setosa
1.5
0.2
5.3
3.7
https://archive.ics.uci.edu...
c28da7b0-aa11-4561-...
Iris-versicolor
3.3
1.0
4.9
2.4
https://archive.ics.uci.edu...
bd343c29-1b59-4c6a-...
Iris-setosa
1.3
0.2
4.4
3.2
https://archive.ics.uci.edu...
bc33d553-cb92-4ba4-...
Iris-versicolor
3.3
1.0
5.0
2.3
https://archive.ics.uci.edu...
b5914302-907e-404-...
Iris-setosa
1.7
0.3
5.7
3.8
https://archive.ics.uci.edu...
a4a33bdb-af8f-467c-...
Iris-versicolor
3.3
1.0
5.0
2.3
https://archive.ics.uci.edu...
9e816148-e4f2-46af-...
Iris-setosa
1.6
0.2
5.1
3.8
https://archive.ics.uci.edu...
9e149a24-eaa6-483f-...
Iris-versicolor
4.1
1.0
5.8
2.7
https://archive.ics.uci.edu...

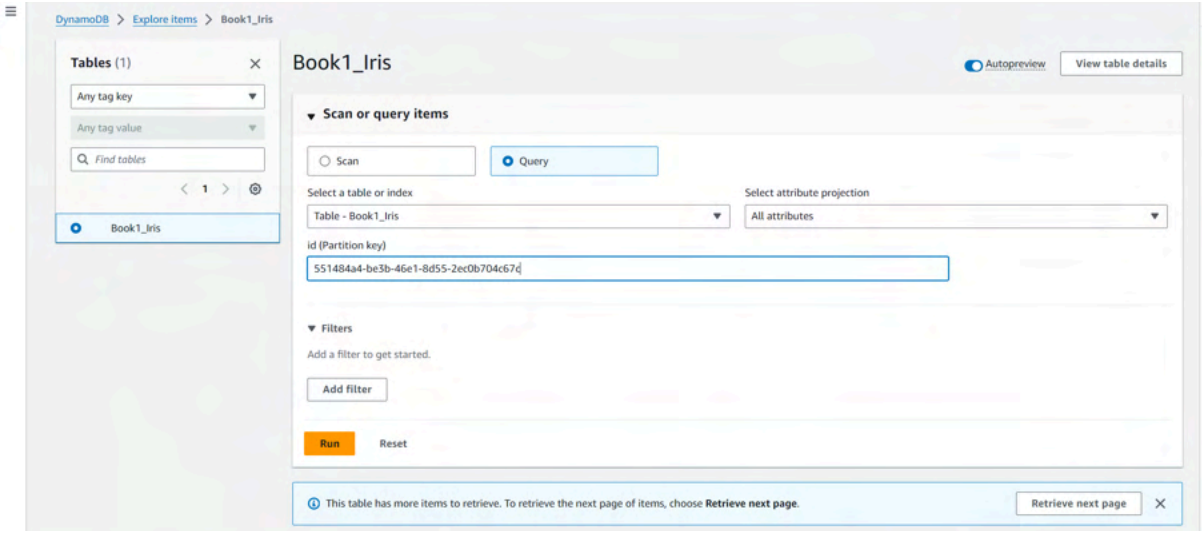


Your database is filtered to irises with petal width less than or equal to 1.0.

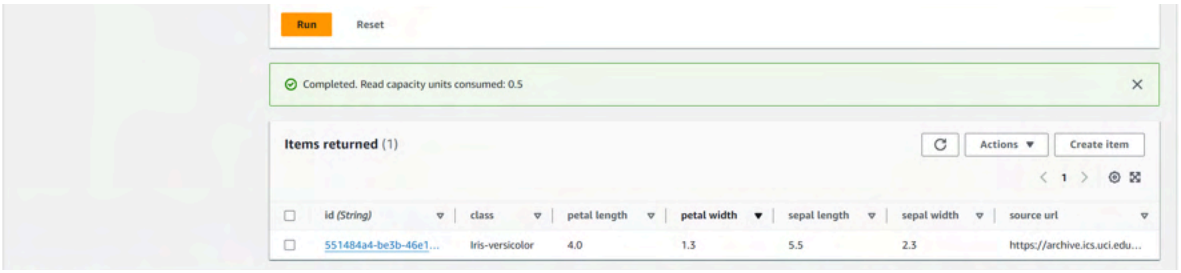
4.2 Query Option

Now try Query.

Select Query. Copy the id (string) of one of the items such as “551484a4-be3b-46e1-8d55-2ec0b704c67c”.



Click on Run.



Your data base was queried to a single item.

Your AWS data base is now managed!

Finish