😊 AWS Kinesis Data Stream using Console

AWS Kinesis Data Streams (KDS) is a real-time, scalable, and fully managed service designed to continuously collect, process, and analyze streaming data at a massive scale. It's part of the broader **Amazon Kinesis** suite, which enables real-time data streaming for various use cases, such as real-time analytics, machine learning, and data transformation.

Key Features of AWS Kinesis Data Streams:

1. **Real-time Data Streaming**: Kinesis Data Streams allows ingestion of data in real-time from multiple sources (IoT devices, application logs, social media feeds, etc.) and processes it continuously.
2. **Massive Throughput**: The service supports high volumes of data, where streams can handle terabytes of data per hour from hundreds of thousands of data producers.
3. **Data Durability and Retention**: Data is stored across multiple availability zones, ensuring high availability and durability. Streams can retain data for up to 7 days (default is 24 hours).
4. **Scalable Architecture**: Kinesis Data Streams is highly scalable. You can add or remove shards (units of capacity) to adjust the throughput capacity of your stream.
5. **Custom Processing**: You can consume the stream in real-time using consumer applications such as **AWS Lambda**, **Amazon EC2**, or **Amazon Kinesis Data Analytics**.
6. **Cost-Effective**: Pricing is based on the volume of data you stream and the resources used for processing it.

Common Use Cases:

- **Real-time analytics**: Analyze logs, events, or sensor data as it arrives.
- **Application monitoring**: Collect and process logs and metrics from applications in real-time.
- **Machine learning**: Stream data to a machine learning model for real-time predictions.
- **Data lakes and warehousing**: Continuously stream data into data lakes (e.g., Amazon S3) or warehouses for further storage and analysis.

How It Works:

1. **Data Producers**: Devices or applications send data to the stream in real time. Each data point sent is referred to as a **record**.
2. **Data Stream**: Consists of **shards**, each capable of ingesting data and enabling consumers to process it.
3. **Data Consumers**: Applications, services, or analytics tools read the data from the stream for further processing, storage, or real-time analysis.

This real-time streaming service helps businesses react to data instantly, making it valuable for time-sensitive applications.

Benefits of AWS Kinesis Data Streams:

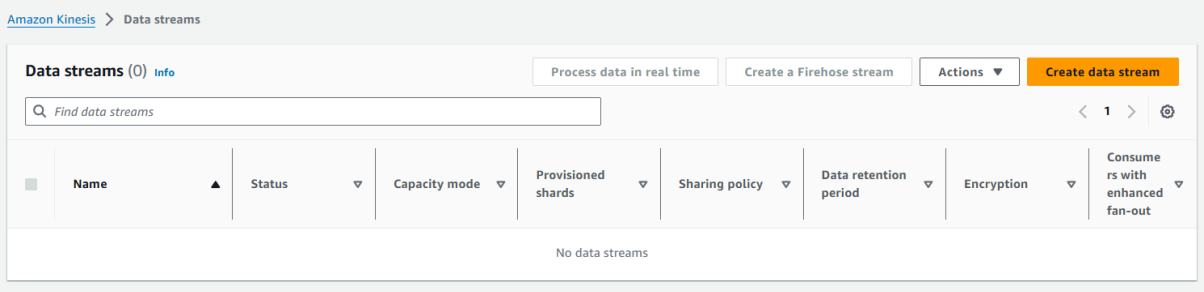
1. **Instant Insights**: Kinesis lets you see and act on data the moment it arrives, which is useful if you want to stay ahead of problems or make real-time decisions.
2. **Scalable**: Whether you have a small or massive amount of data, Kinesis can handle it without slowing down. It grows as your needs grow.
3. **Cost-Effective**: You only pay for what you use, which makes it affordable for businesses of all sizes. You don't need to invest in expensive infrastructure to handle real-time data.
4. **No Need to Manage Servers**: Kinesis is fully managed by AWS, meaning you don't have to worry about setting up or maintaining servers to process your data. This saves you time and effort.
5. **Works with Other Tools**: You can easily connect Kinesis to other AWS services or external tools to store, analyze, or visualize your data in real-time.

In this exercise, you will create a Kinesis Data Stream, an S3 bucket, and three Lambda functions (one for producing data and two for consuming it). The Kinesis Data Stream will collect data from an S3 bucket, where objects are stored. You'll set up event notifications in S3 to trigger a Lambda function whenever a new text file is uploaded. This function reads the file and sends its contents to the Kinesis stream. The other two Lambda functions act as consumers, processing data from the Kinesis stream. You'll monitor the flow of data through logs in CloudWatch and view the processed records using Kinesis Data Viewer.

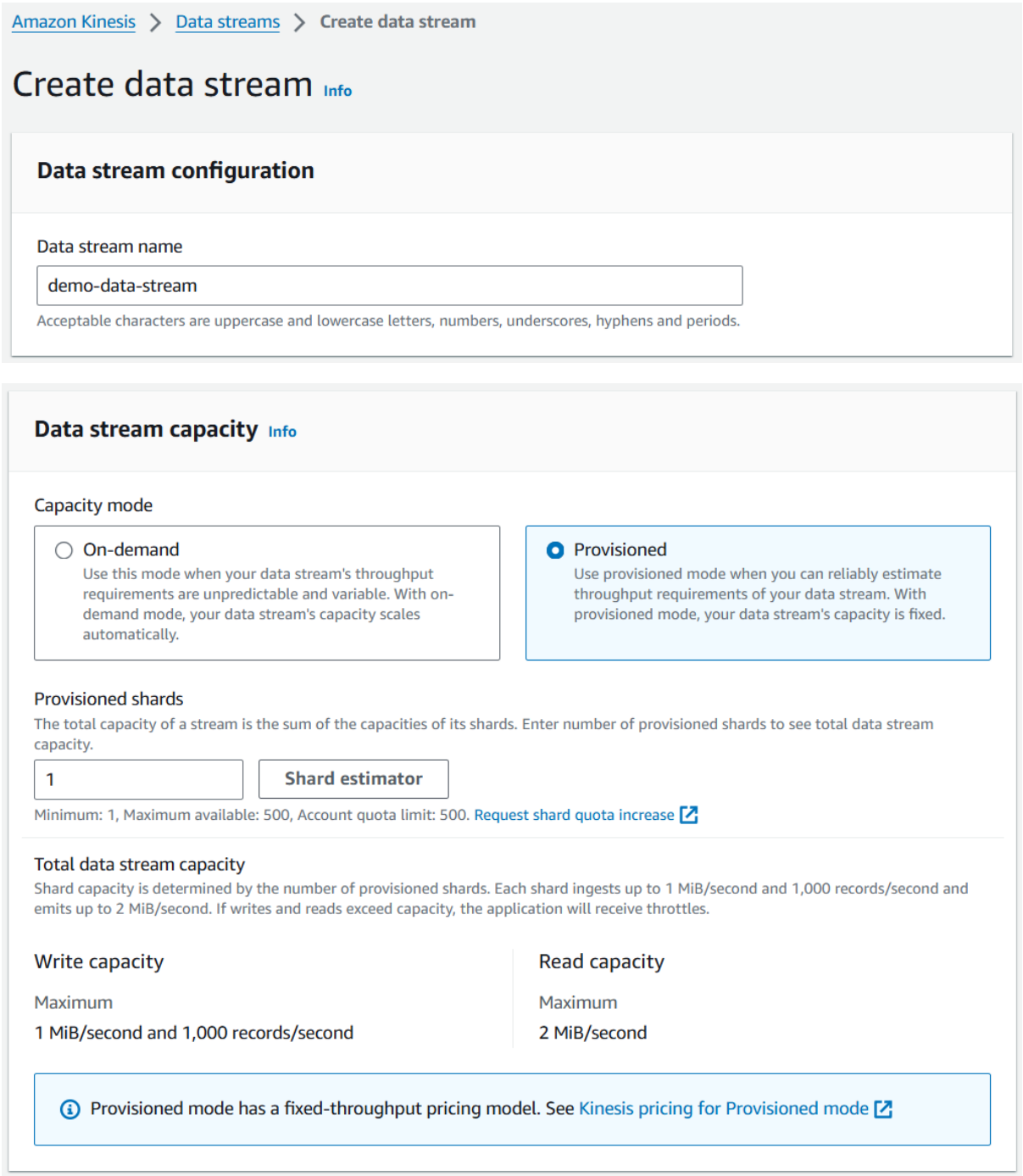
The end goal is to understand how Kinesis streams data from S3 to different consumers using Lambda functions and track this flow with CloudWatch.

😄 To begin with the Lab:

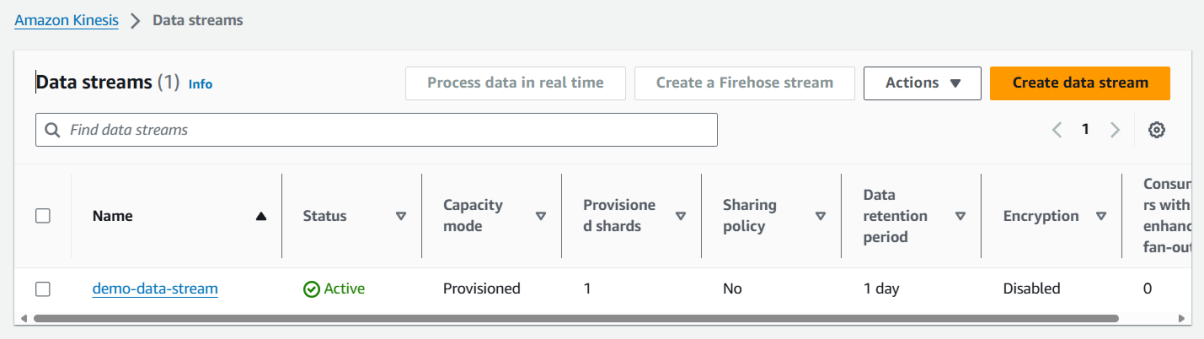
1. In this lab, we are going to create a data stream in Kinesis using Console. In your AWS Console search for Kinesis and go to Data Streams. Click on Create Data stream.



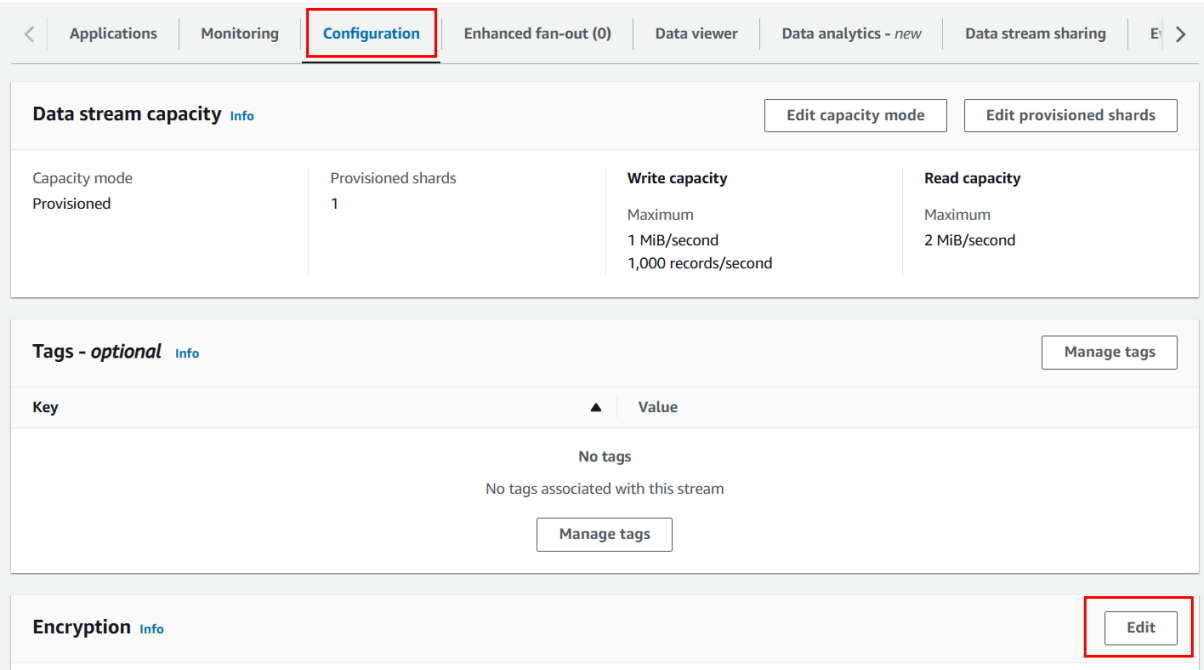
2. First, give your stream a name then choose Provisioned and for provisioned shards capacity choose 1. After that just click on Create Stream.



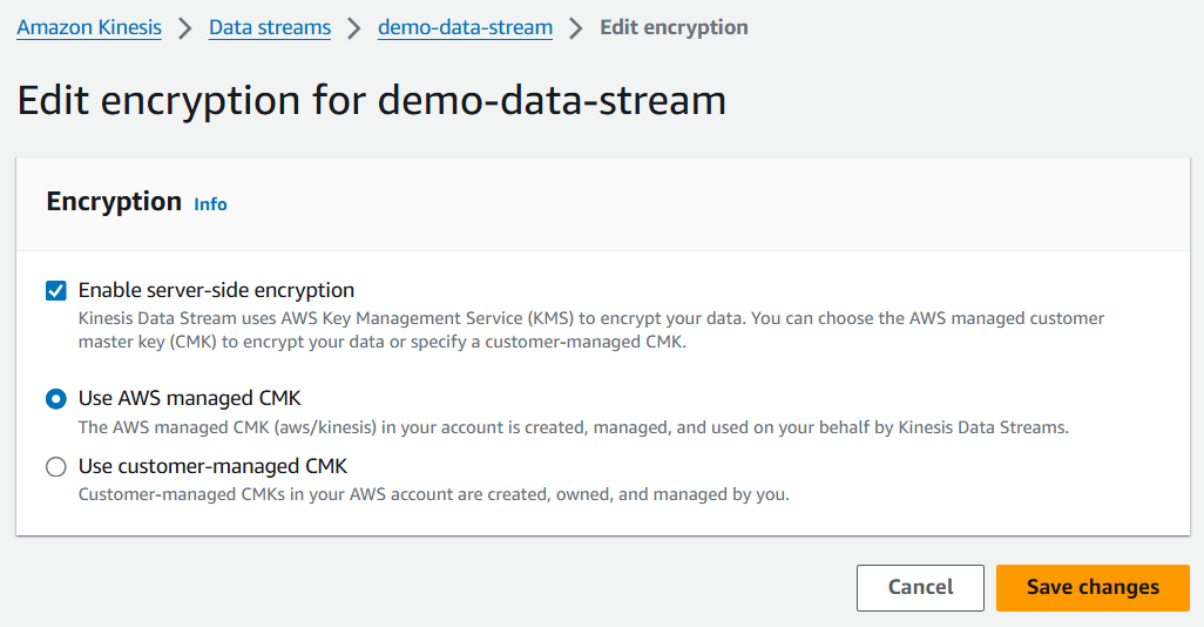
3. Below you can see that your data stream has been created.



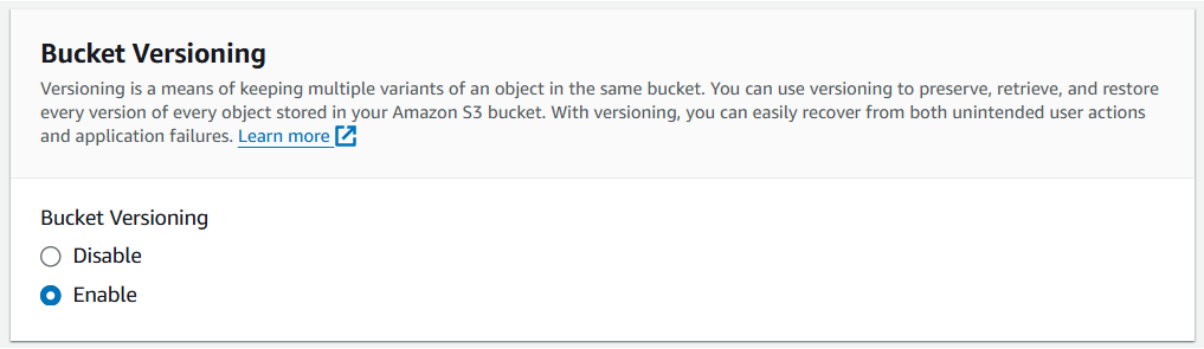
4. Now go inside your data stream and choose configuration tab, here you scroll down to encryption and click on Edit.



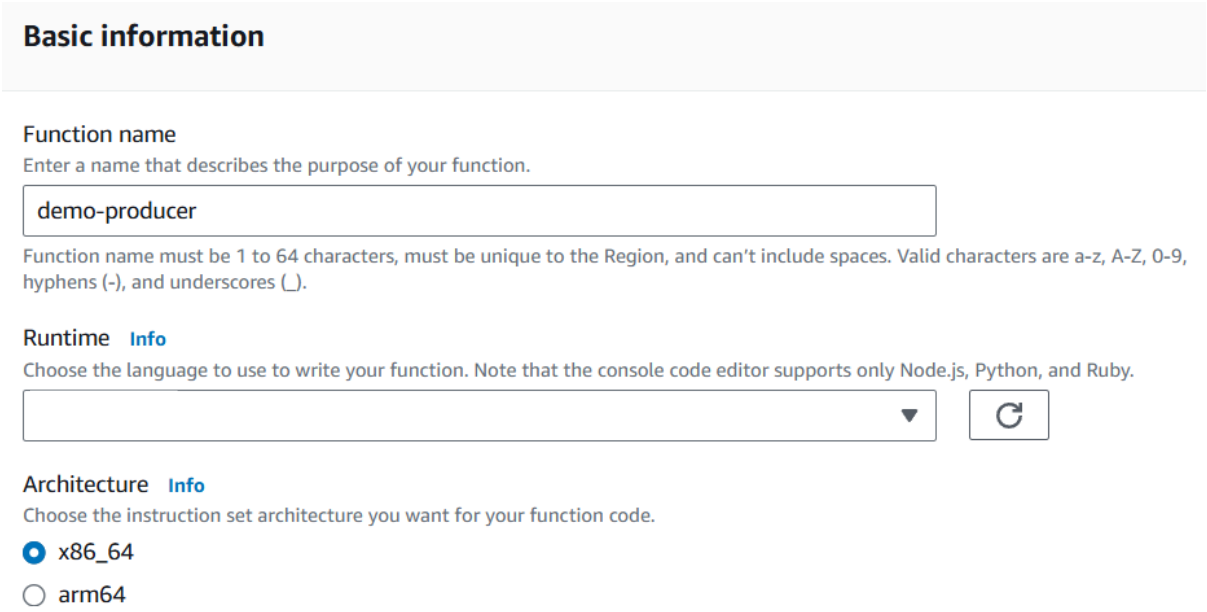
5. Here you need to enable server-side encryption and choose AWS-managed CMK. Then click on Save Changes.



6. Now we are going to create an S3 bucket. For this, you need to give your bucket a unique name and you must enable the Bucket Versioning and just create your bucket.



7. After that we are going to create 3 Lambda functions. We will create 1 lambda function for the Producer and 2 lambda functions for the Consumers.
8. Go to lambda and click on create function then give it a name choose Python 3.11 as your runtime and click on create function.

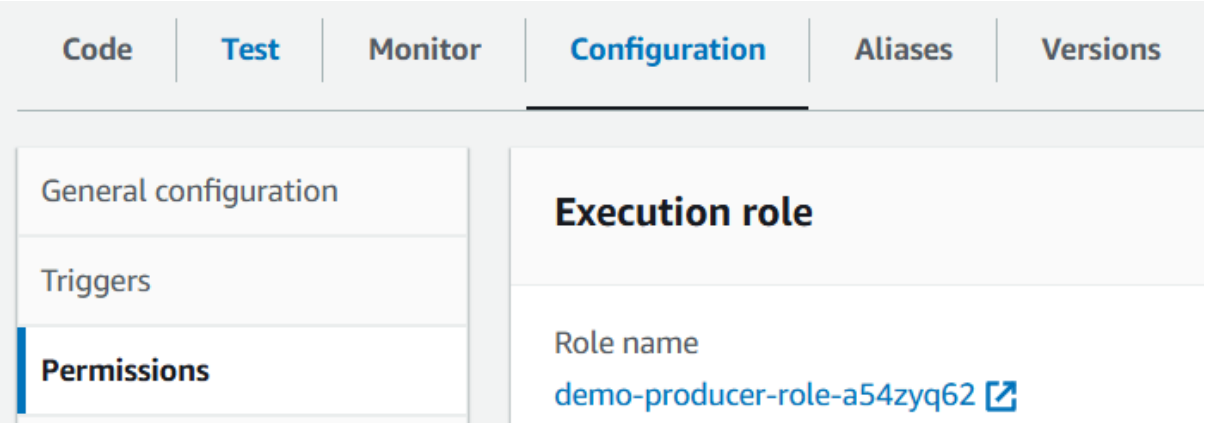


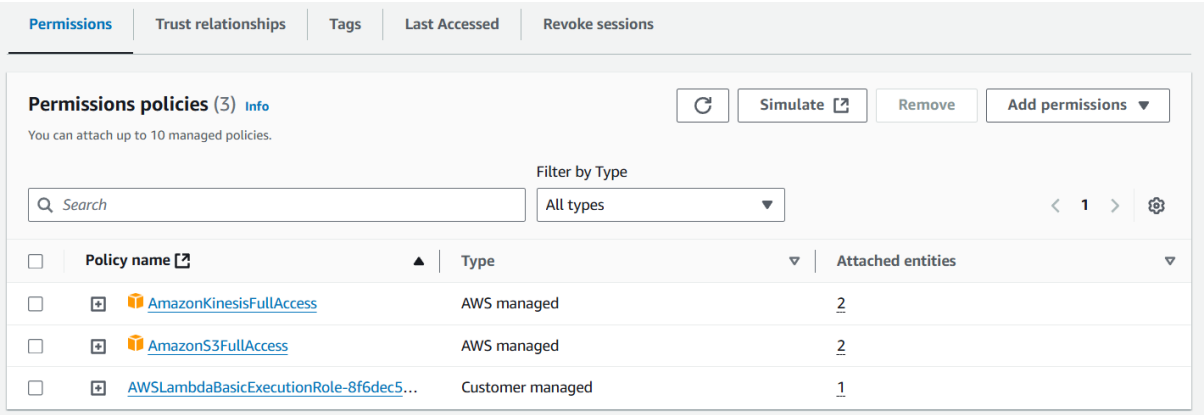
9. Once your function is created then you need to paste the code mentioned below and deploy it in your lambda function. In this code, you just need to change the Kinesis Data Stream name.

```
import json
import boto3
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()

# Initialize AWS clients
s3 = boto3.client('s3', region_name='us-east-1')
kinesis = boto3.client('kinesis', region_name='us-east-1')

def lambda_handler(event, context):
    logger.info(json.dumps(event))

    bucket_name = event['Records'][0]['s3']['bucket']['name']
    key_name = event['Records'][0]['s3']['object']['key']

    try:
        data = s3.get_object(Bucket=bucket_name, Key=key_name)
        data_string = data['Body'].read().decode('utf-8')

        payload = {
            'data': data_string
        }
```

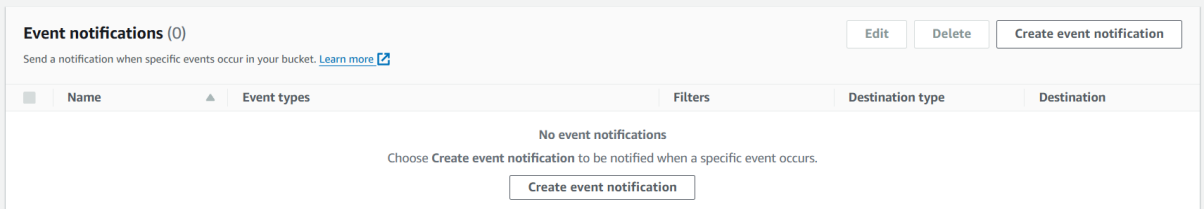
```
            send_to_kinesis(payload, key_name)

        except Exception as e:
            logger.error(e)

    def send_to_kinesis(payload, partition_key):
        params = {
            'Data': json.dumps(payload),
            'PartitionKey': partition_key,
            'StreamName': 'demo-data-stream'
        }

        try:
            response = kinesis.put_record(**params)
            logger.info(response)
        except Exception as e:
            logger.error(e)
```

10. Then go to the configuration tab in your lambda function and open the execution role for it. You need to add S3 full access and Kinesis full access to your IAM role attach with your lambda function.

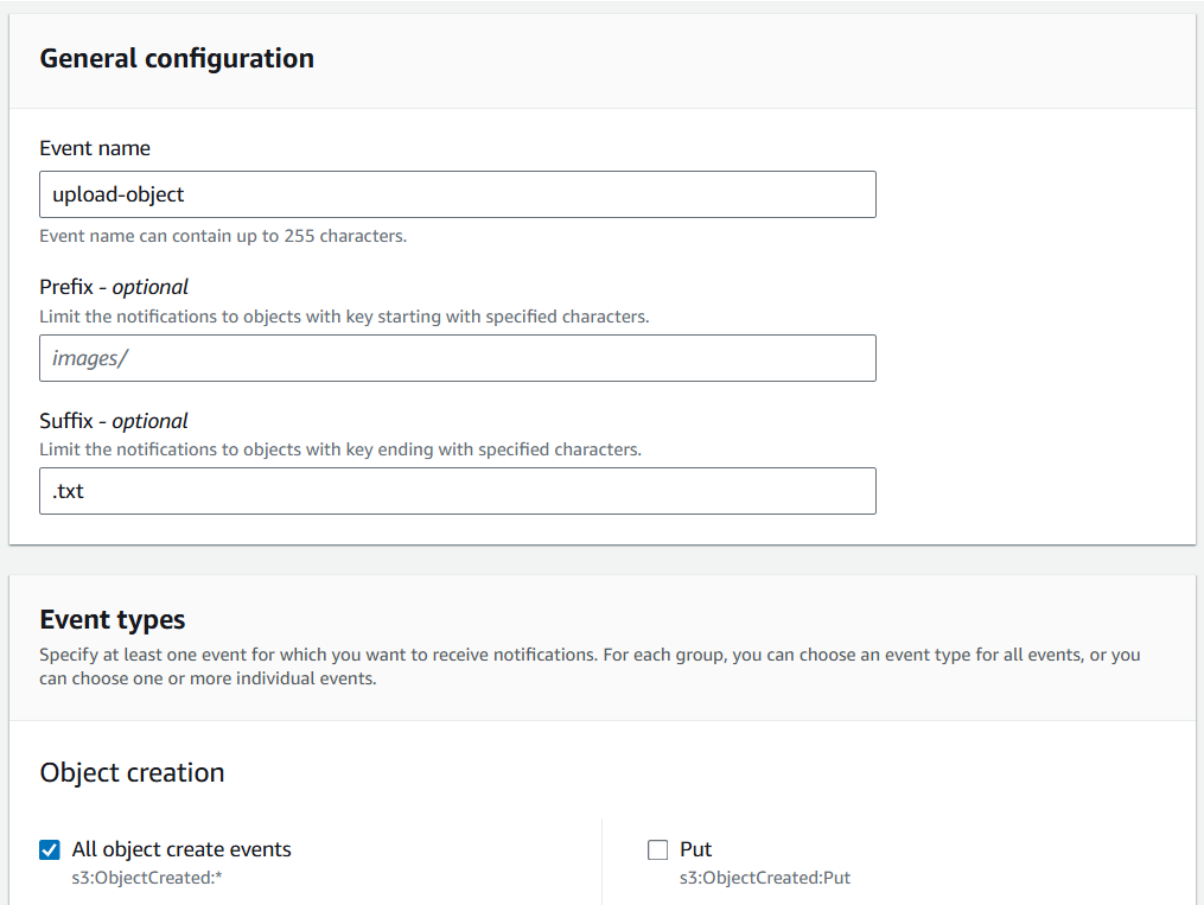




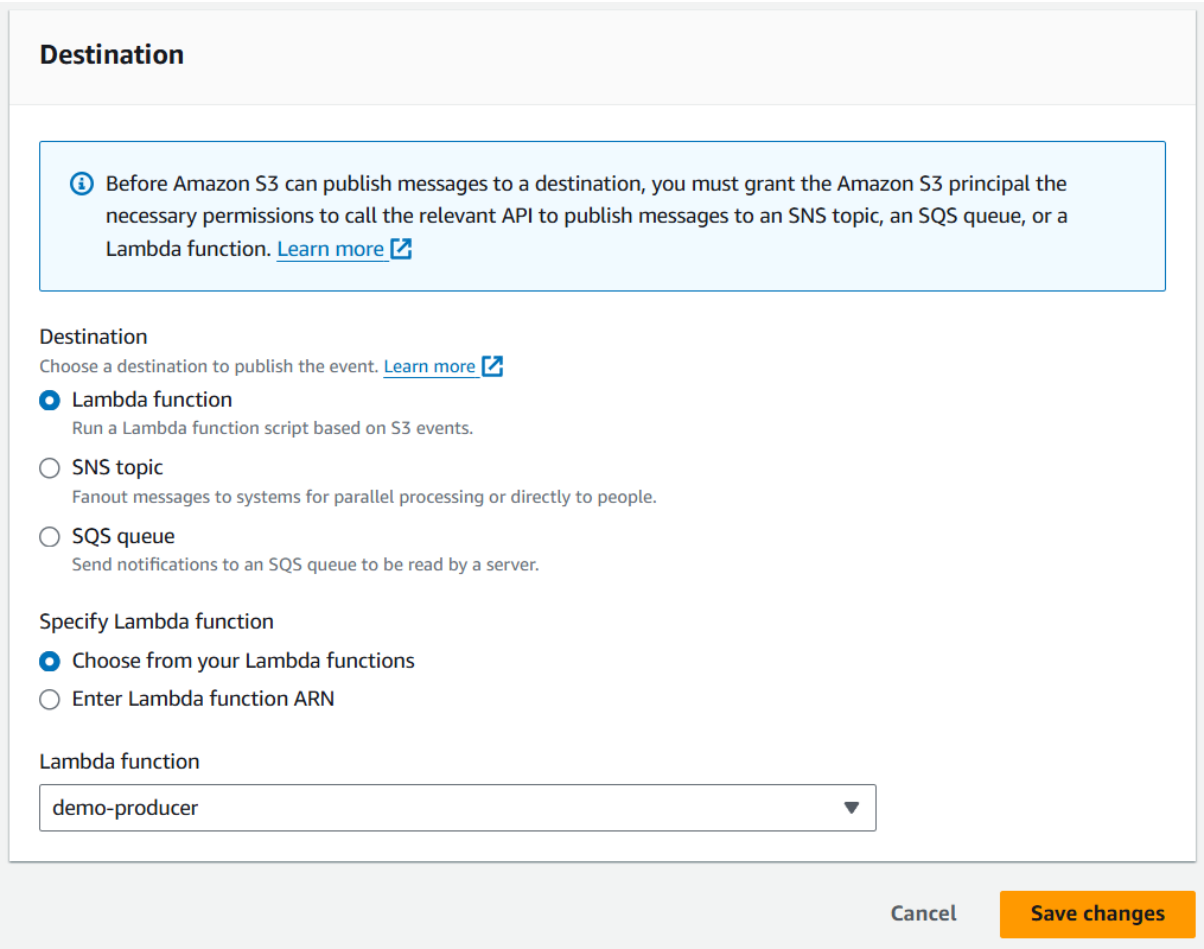
11. Now come back to S3, inside your bucket go to Properties scroll down to event notification and click on Create.



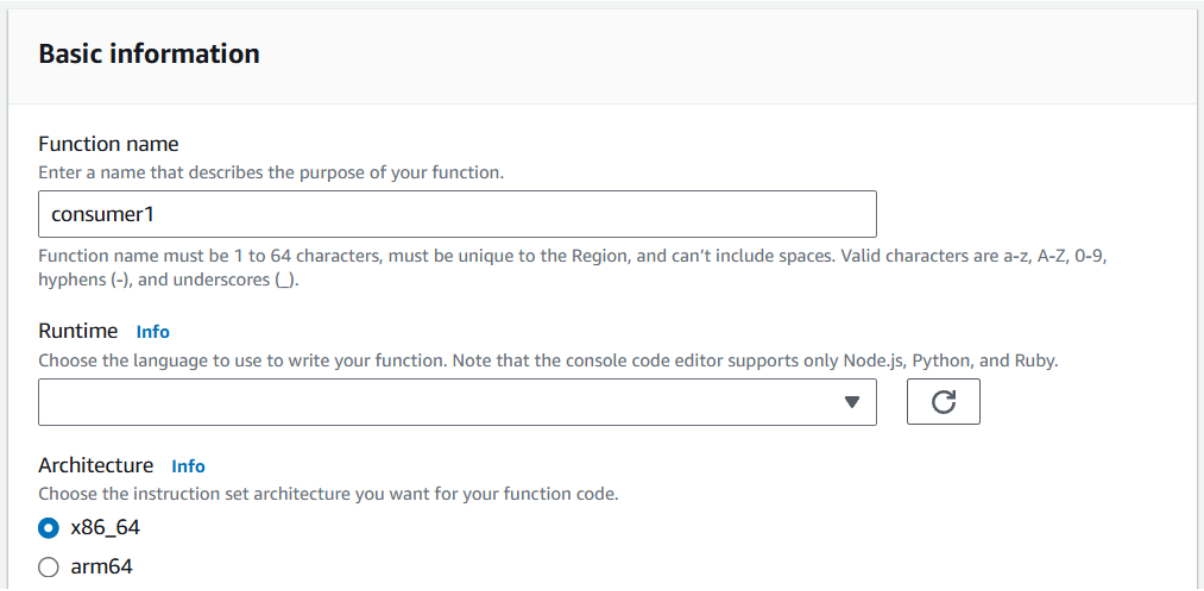
12. Here you need to give your event a name and, in the Suffix, write **.txt** then in the event types just choose All objects to create events. After that scroll down to the bottom.



13. Now you need to choose a destination which is the Lambda function and choose your function. Click on Save Changes.
14. So, what it means is whenever an object is created, the producer lambda function will be triggered.



15. Along with this we are going to create 2 more lambda functions which we are going to call consumer lambda functions. Choose the runtime as Python 3.11.

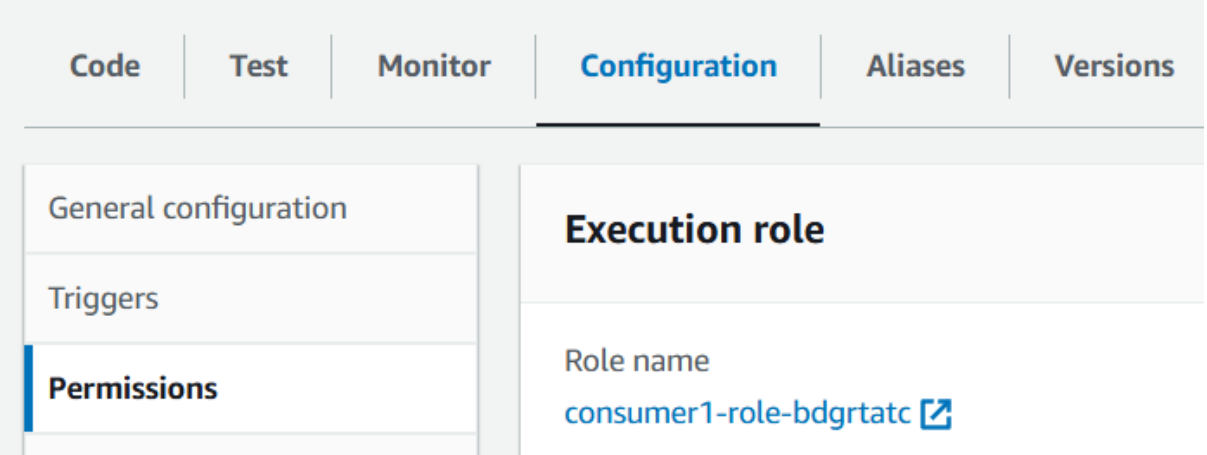


16. Once the function is created then we need to modify the code with the code given below.

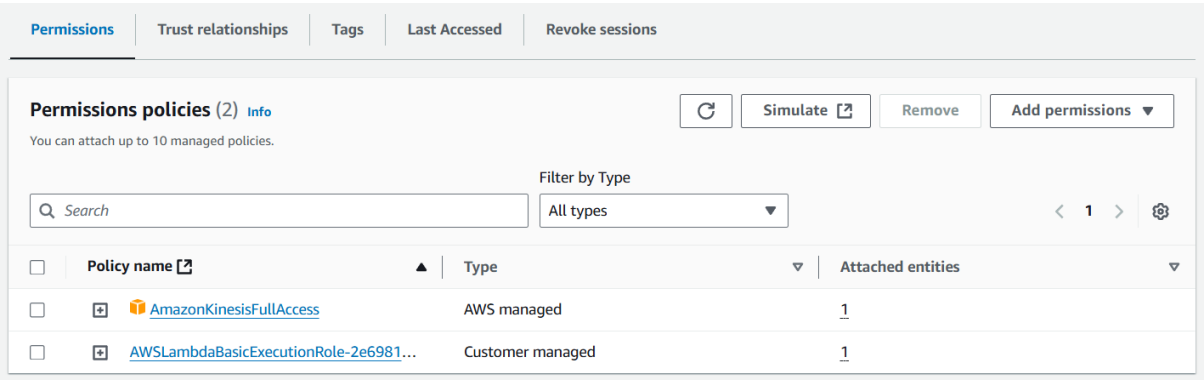
```
import base64
import json
```

```
def lambda_handler(event, context):
    print(json.dumps(event))

    for record in event['Records']:
        data = json.loads(base64.b64decode(record['kinesis']['data']).decode('utf-8'))
        print('This is consumer - 1', data)
```

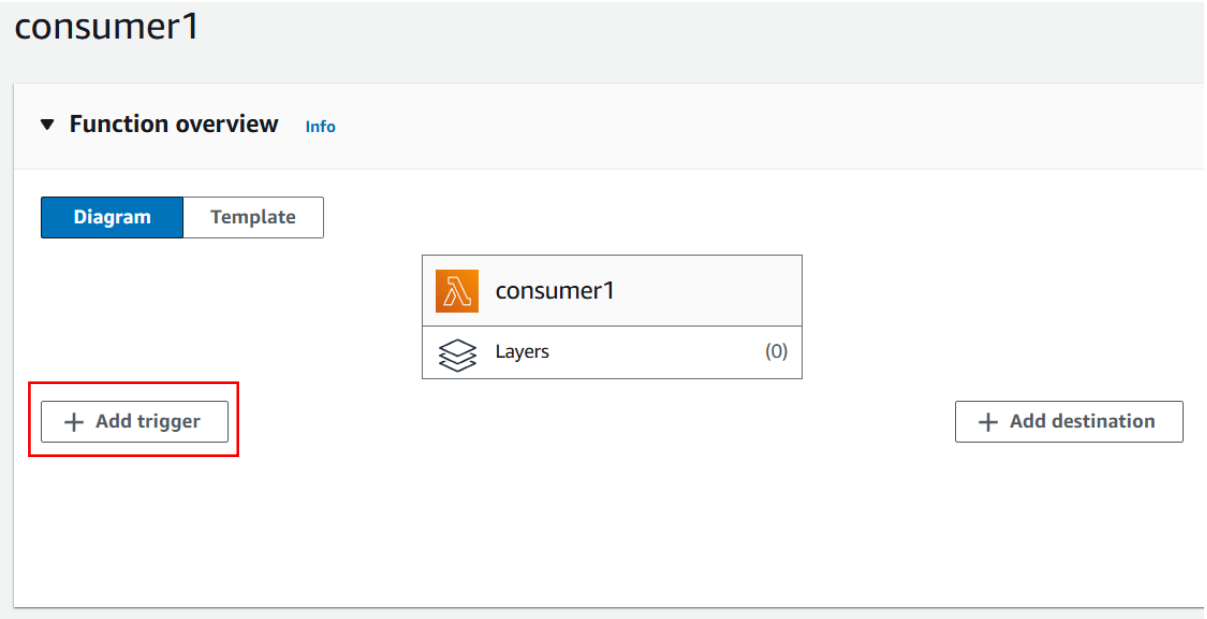
17. For this consumer lambda function we need to modify the IAM role, so go to its configurations tab choose Permissions, and click on the execution role.



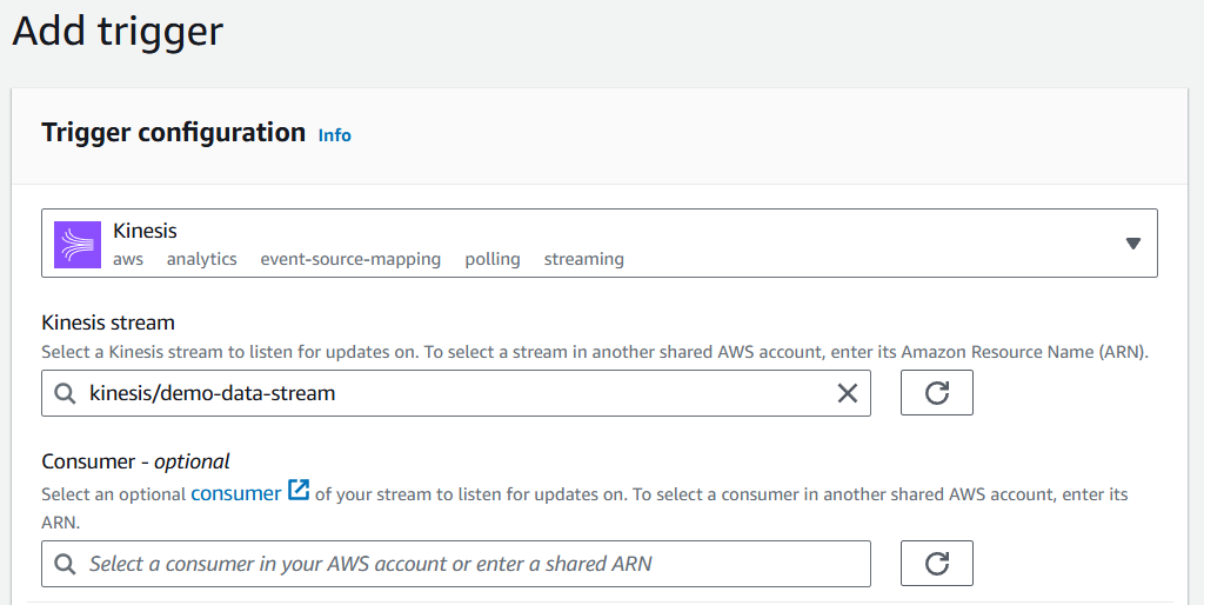
18. For permission you need to provide Kinesis full access. Then come back to lambda.



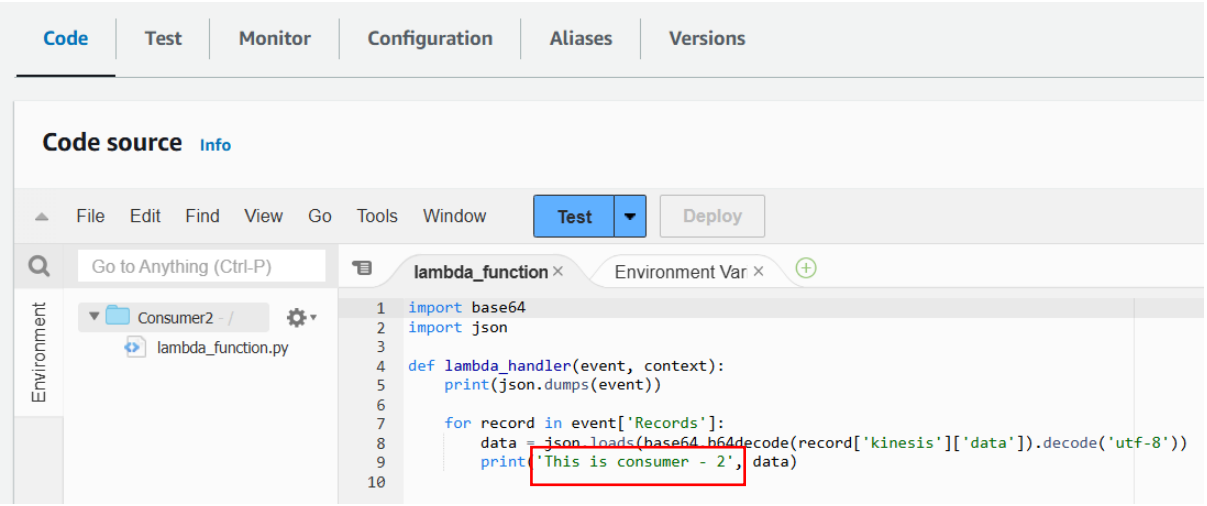
19. Now for the consumer 1 lambda function we are going to add a trigger. Click on Add trigger.



20. Then in the trigger search for Kinesis and choose your data stream, keep everything as it is and click on Add.



21. Similarly, you are going to create another lambda function for Consumer 2 and just add the same IAM role to this new lambda function. Also, modify the code by using the same code as before.
22. Just remember to change the consumer value in the code.



Basic information

Function name

Enter a name that describes the purpose of your function.

consumer2

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

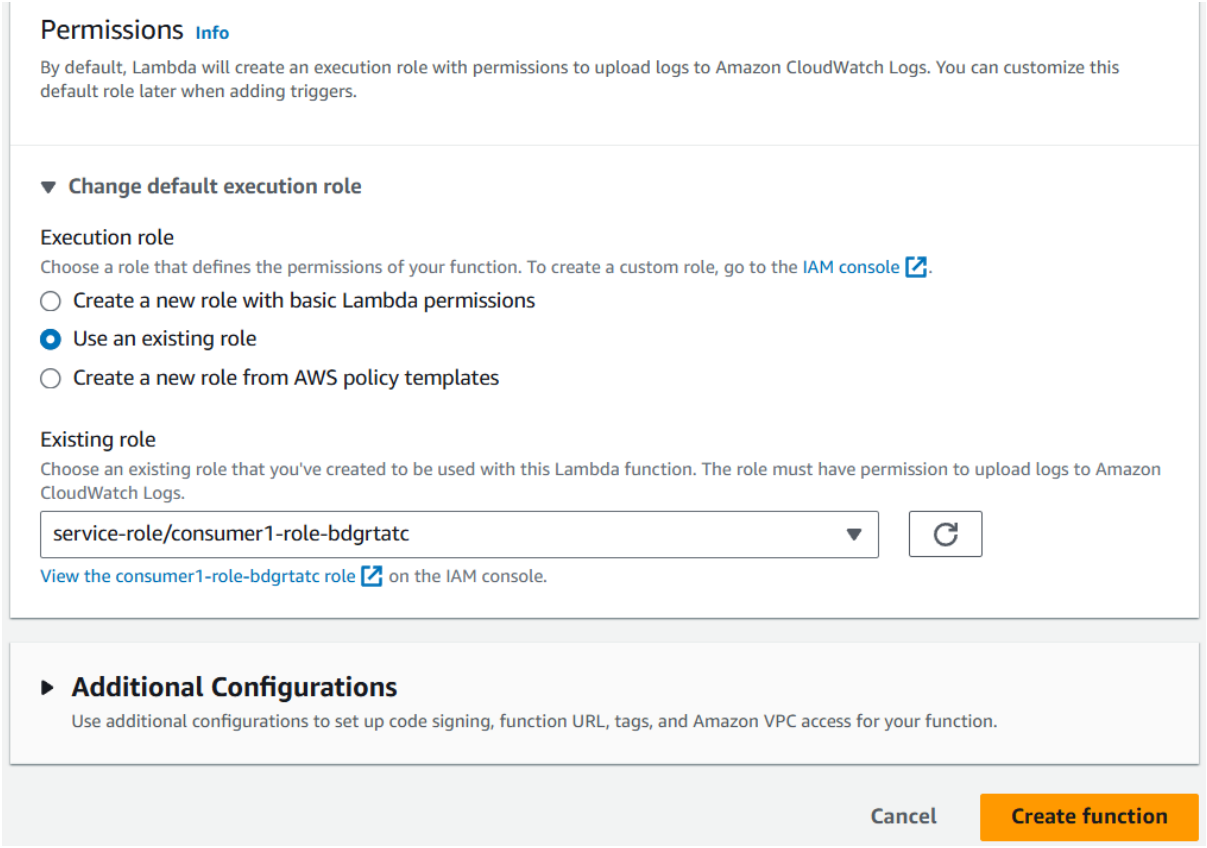
Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

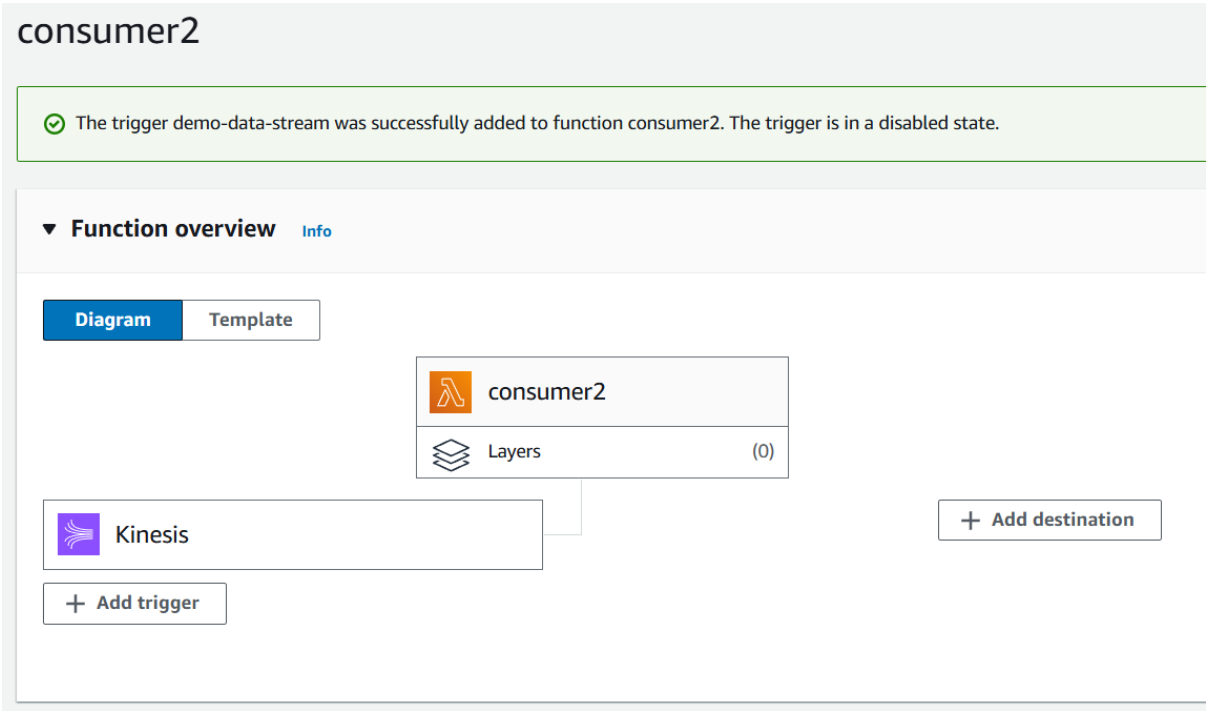
Architecture Info

Choose the instruction set architecture you want for your function code.

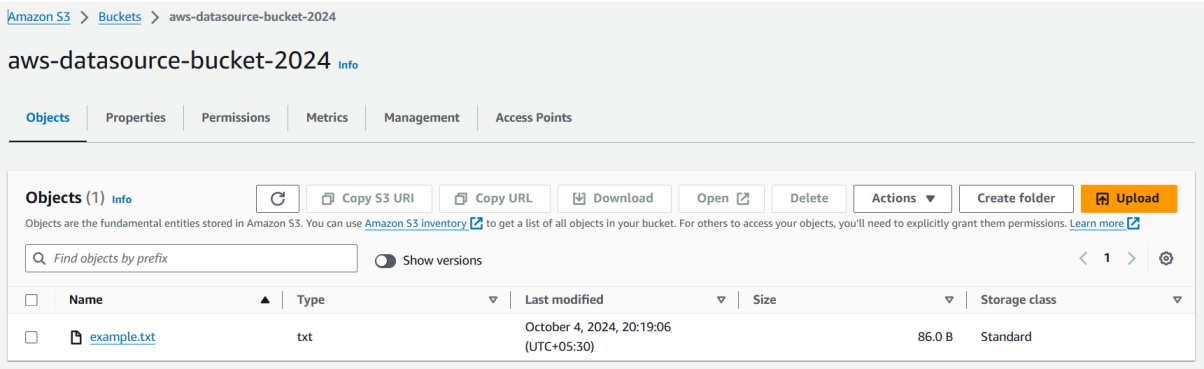
- (•) x86_64
- ( ) arm64



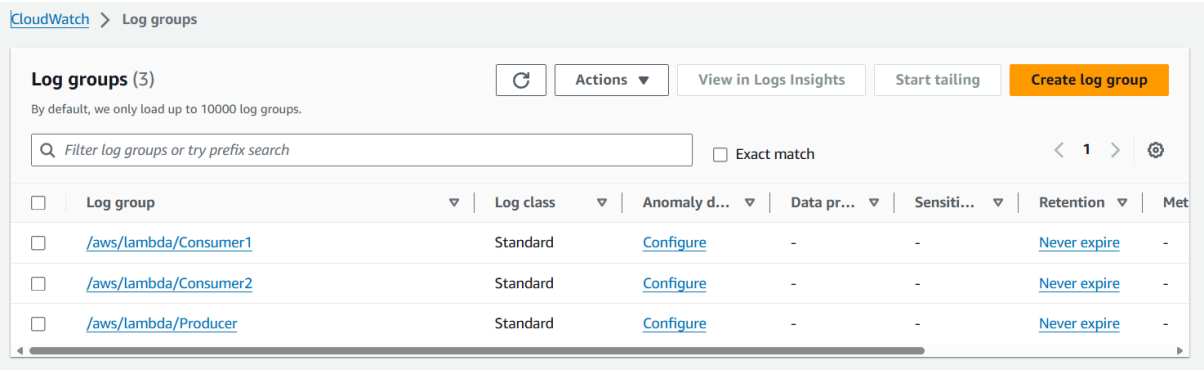
23. Once your function is created then you need to add a trigger for Kinesis like you did before.



24. Now we are going to upload an object in our S3 bucket. This object should have a .txt type so that it can trigger our lambda function.
25. Below you can see that we have uploaded a text file in our S3 bucket.

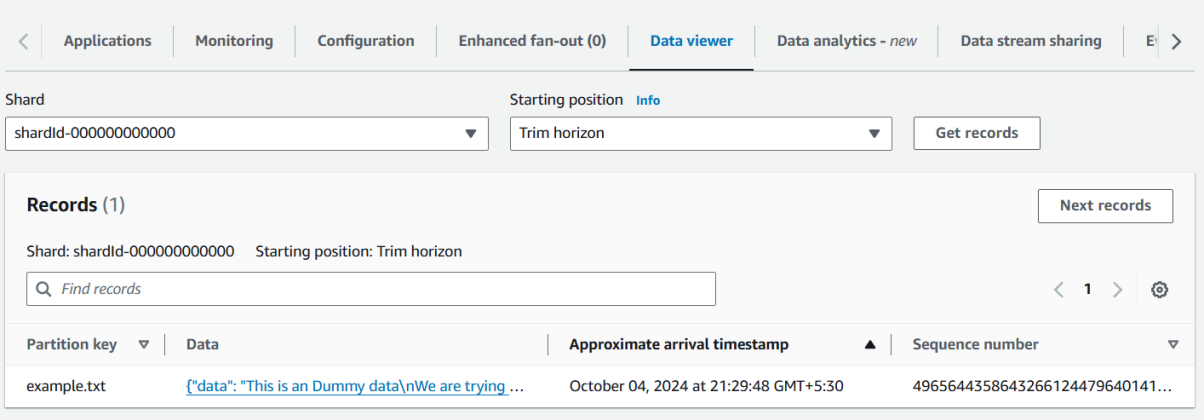


26. Now go to Cloud Watch and look for the logs, here you can see that the producer lambda function was triggered along with both of the consumer lambda functions when we uploaded a text file in our S3 bucket.



27. If you go inside your log stream for the consumers and expand the logs you will be able to see the record as shown below.

28. Navigate to Kinesis, go to Data viewer and choose your shard ID then the starting position and click on get records and you will see your text file and the data here.



29. Once you are done just delete all the resources one by one.