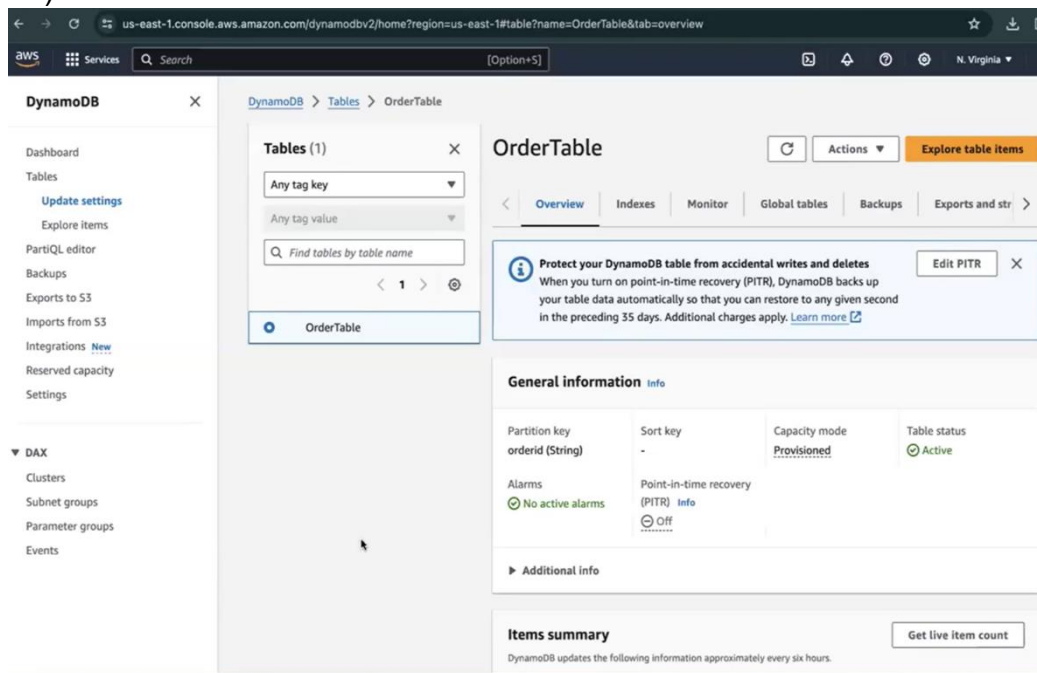
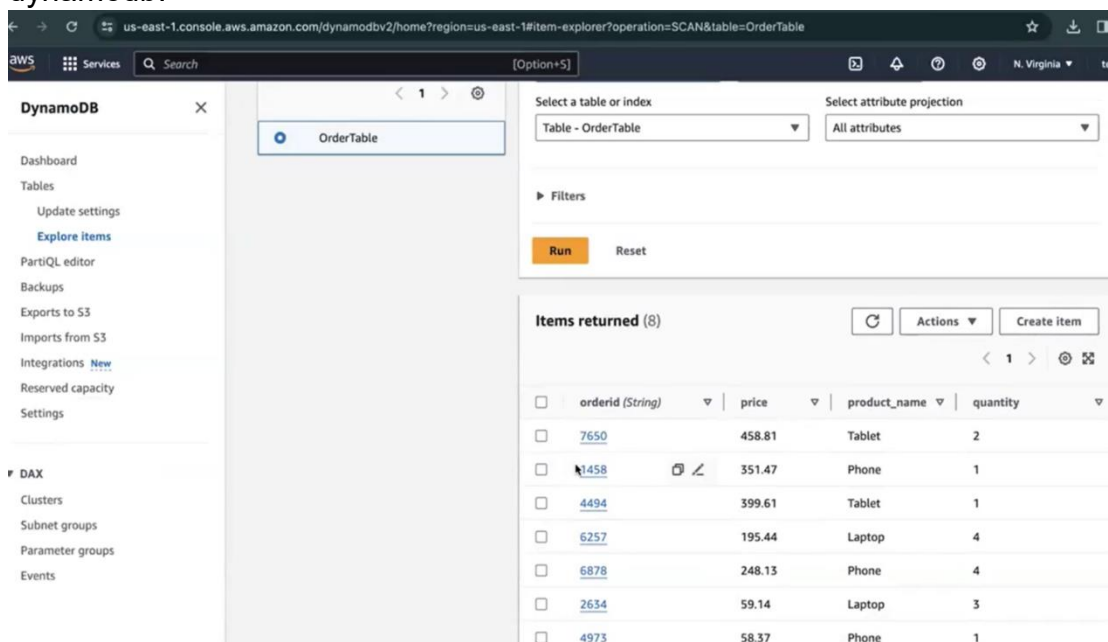


# Sales Data Projection

1. this was a sales data streaming project for processing the data from Dynamo db to destination S3 target path.
2. there was some source from where records(orderid, product\_name, quantity, price) were being written into the nosql dynamodb(which is a key value based db).



3. we created a dynamodb table where we targeted particular partition key(orderid) and the entire data(orderid, product\_name, quantity, price) used to get ingested as a value there. That's how we were publishing a record in dynamodb.



4. next our task was to capture changes happening in table. Records were getting inserted, updated/deleted to table in real time. CDC is whatever changes are happening in real time we capture those changes. That's what CDC is. So we needed to enable dynamodb streams from 'Exports & Streams' option to perform CDC. As we were able to consume the CDC changes, hence created CDC pipeline.
5. then we set up a kinesis stream which is a queuing mechanism to hold our data. Kinesis stream has a concept of shards just like publishing data to partitions. Data here arrives at kinesis stream in different shard.

The screenshot shows the Amazon Kinesis console interface. The left sidebar contains navigation links for Dashboard, Data streams, Data Firehose, Managed Apache Flink, and Resources. The main content area displays the 'Data stream summary' for the 'kinesis-sales-order' stream. The summary includes the following details:

Status	Capacity mode	ARN	Creation time
Active	On-demand	arn:aws:kinesis:us-east-1:1851725469799:stream/kinesis-sales-order	January 20, 2024 at 11:53 GMT+5:30

Below the summary, the 'Data viewer' tab is selected, showing the 'Shard' dropdown set to 'shardid-000000000002'. The 'Starting position' is set to 'At timestamp', and the 'Start date' is '2024/01/20'. The 'Start time' is '00:00:00'. A 'Get records' button is visible. Below this, the 'Records (18)' section shows a list of records. The first record is expanded, showing the following JSON data:

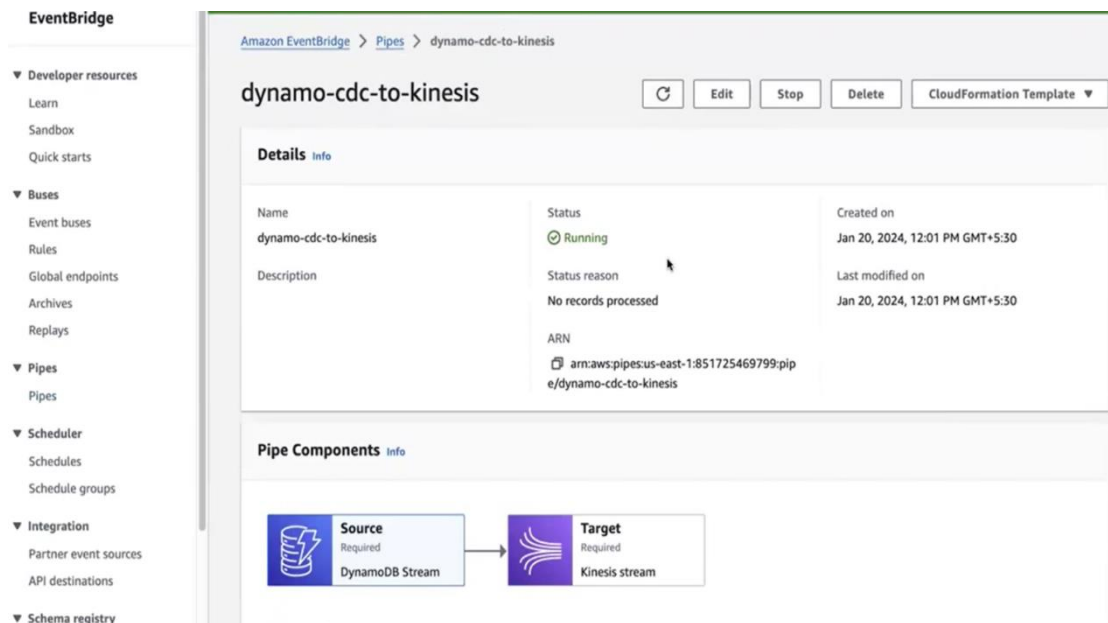
```
{
  "eventID": "f3d2479e0c6ed4762e99a377f4a143b6",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-1",
  "dynamodb": {
    "ApproximateCreationDateTime": 1705732562,
    "Keys": {
      "orderid": {
        "S": "3714"
      }
    },
    "NewImage": {
      "quantity": {
        "N": "5"
      },
      "orderid": {
        "S": "3714"
      },
      "price": {
        "N": "190.07"
      },
      "product_name": {
        "S": "Charger"
      },
      "SequenceNumber": "3510000000077909614008",
      "SizeBytes": 60,
      "StreamViewType": "NEW_IMAGE"
    },
    "eventSourceARN": "arn:aws:dynamodb:us-east-1:1851725469799:table/OrderTable/stream/2024-01-20T06:21:09.489"
  }
}
```

The screenshot shows the 'Record data' dialog box. It displays the 'Sequence number' as '49648482746025350141701660598947998844101090210306064418' and the 'Shard ID' as 'shardid-000000000002'. Below this, there are two tabs: 'Raw data' (selected) and 'JSON'. A 'Copy' button is visible. The 'Raw data' tab shows the following JSON data:

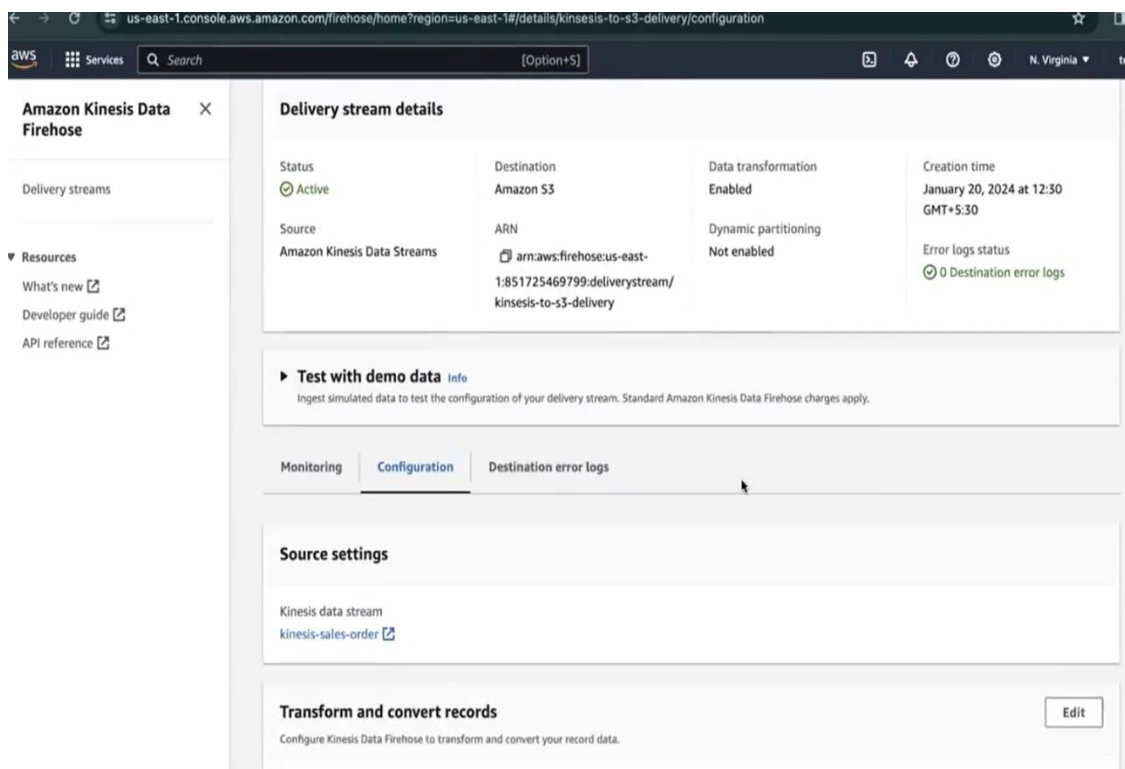
```
{
  "eventID": "f3d2479e0c6ed4762e99a377f4a143b6",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-1",
  "dynamodb": {
    "ApproximateCreationDateTime": 1705732562,
    "Keys": {
      "orderid": {
        "S": "3714"
      }
    },
    "NewImage": {
      "quantity": {
        "N": "5"
      },
      "orderid": {
        "S": "3714"
      },
      "price": {
        "N": "190.07"
      },
      "product_name": {
        "S": "Charger"
      },
      "SequenceNumber": "3510000000077909614008",
      "SizeBytes": 60,
      "StreamViewType": "NEW_IMAGE"
    },
    "eventSourceARN": "arn:aws:dynamodb:us-east-1:1851725469799:table/OrderTable/stream/2024-01-20T06:21:09.489"
  }
}
```

A 'Close' button is located at the bottom right of the dialog box.

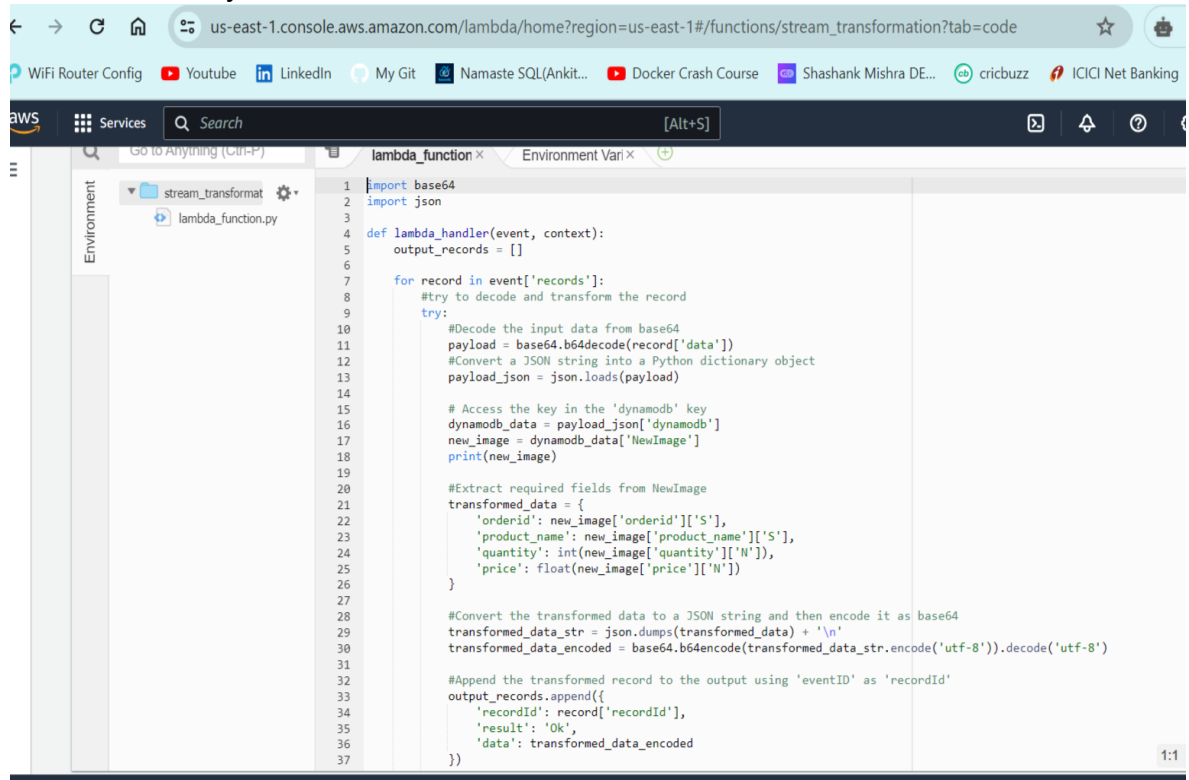
6. then we had a event bridge pipe in between dynamodb stream and kinesis stream. We set a batch size consisting of no of message to be sent to any kinesis stream shard. Set up required IAM role to eventbridge pipe to access dynamo stream & kinesis stream. Using event bridge we used to flow data to kinesis stream.



7. next we created a firehose stream which is a delivery stream. It processed data(like ingest, transform) and delivered streaming data to destinations(like data lakes, data warehouses) in batches based on buffer size or buffer time factor. Here source is kinesis stream and destination is s3.

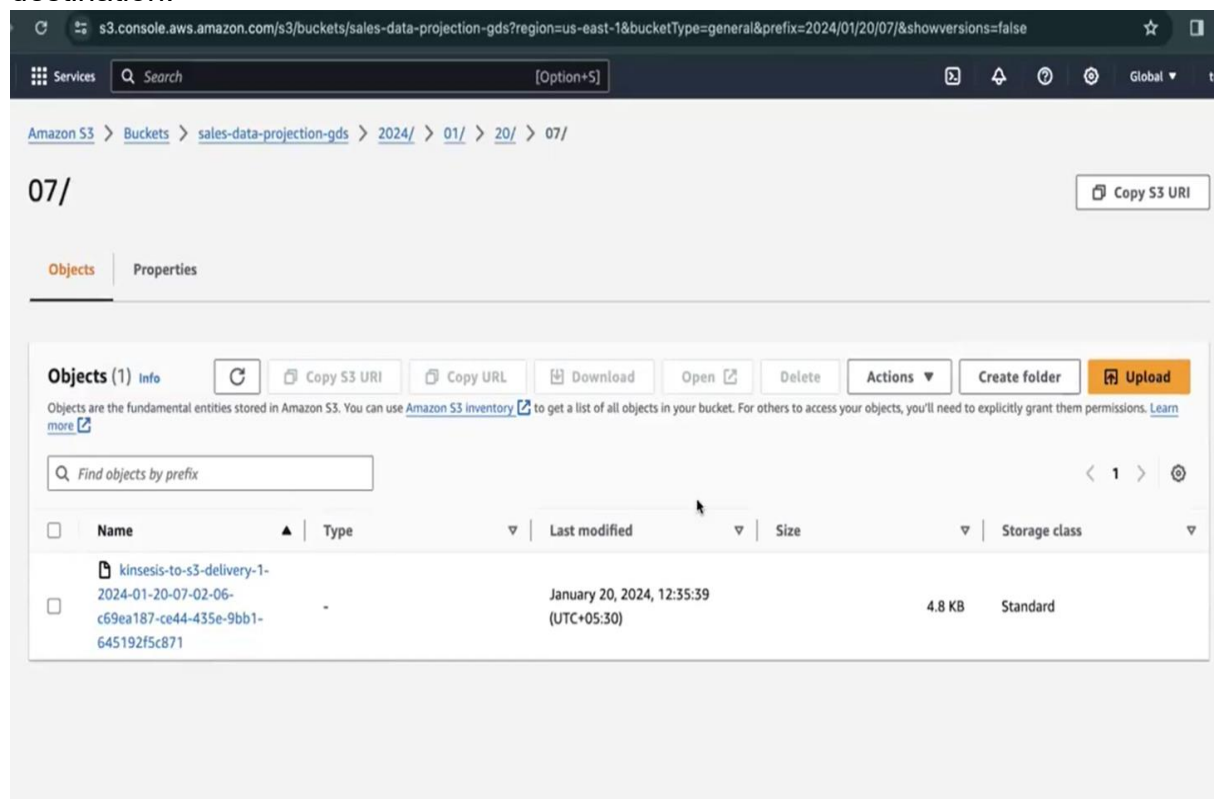


8. So Kinesis firehose delivery stream was loading data from source Kinesis stream, holding that streaming data until the buffer size/buffer time is reached & then data is sent to lambda function in a batch. Following this set up **Firehose** to invoke our **Lambda function** to **transform** incoming batch records this way.



```
1 import base64
2 import json
3
4 def lambda_handler(event, context):
5     output_records = []
6
7     for record in event['records']:
8         #try to decode and transform the record
9         try:
10             #Decode the input data from base64
11             payload = base64.b64decode(record['data'])
12             #Convert a JSON string into a Python dictionary object
13             payload_json = json.loads(payload)
14
15             # Access the key in the 'dynamodb' key
16             dynamodb_data = payload_json['dynamodb']
17             new_image = dynamodb_data['NewImage']
18             print(new_image)
19
20             #Extract required fields from NewImage
21             transformed_data = {
22                 'orderId': new_image['orderId']['S'],
23                 'product_name': new_image['product_name']['S'],
24                 'quantity': int(new_image['quantity']['N']),
25                 'price': float(new_image['price']['N'])
26             }
27
28             #Convert the transformed data to a JSON string and then encode it as base64
29             transformed_data_str = json.dumps(transformed_data) + '\n'
30             transformed_data_encoded = base64.b64encode(transformed_data_str.encode('utf-8')).decode('utf-8')
31
32             #Append the transformed record to the output using 'eventId' as 'recordId'
33             output_records.append({
34                 'recordId': record['recordId'],
35                 'result': 'Ok',
36                 'data': transformed_data_encoded
37             })
```

9. Once this is done, then delivering the transformed batch records to S3 destination.



10. Lastly, we created crawler over s3 target path which resulting in catalog table creation. Also, needed to add classifier for json format(i.e. \$.columns).

The screenshot shows the AWS Glue console interface. A green banner at the top states "Crawler successfully starting" and "The following crawler is now starting: 'sales\_data\_crawler'". The main panel displays the configuration for the "sales\_data\_crawler".

**Crawler properties**

Property	Value
Name	sales_data_crawler
IAM role	glue-role
Database	sales-data-catalog
State	READY
Description	-
Security configuration	-
Lake Formation configuration	-
Table prefix	projection_
Maximum table threshold	-

**Crawler runs (1)**

The list of crawler runs for this crawler.

Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table count
January 20, 2024 at 07:13:30	-	-	Running	-	-

11. using athena query we then did analysis

The screenshot shows the AWS Athena console interface. The query editor displays the SQL query: "projection\_sales\_data\_projection\_gds". The query has been executed successfully, and the results are displayed in a table.

**Query results**

Completed Time in queue: 103 ms Run time: 1.306 sec Data scanned: 8.89 KB

**Results (10)**

#	orderid	product_name	quantity	price	partition_0	partition_1	partition_2
1	7864	Phone	4	175.66	2024	01	20
2	9317	Charger	2	458.18	2024	01	20
3	6468	Charger	3	243.42	2024	01	20
4	7341	Tablet	1	230.83	2024	01	20
5	3958	Charger	3	476.51	2024	01	20
6	6324	Laptop	1	31.55	2024	01	20
7	3238	Headphones	5	434.68	2024	01	20
8	8007	Phone	2	36.83	2024	01	20
9	8033	Phone	3	166.54	2024	01	20