Amazon Web Services
Data Engineering Immersion Day

Lab 1. Hydrating the Data Lake with Glue Streaming ETL

July 2021

Table of Contents

Introduction

This lab will guide you to understand AWS Glue Streaming ETL feature. You will start with hydrating your Data Lake from Amazon Kinesis Data Generator (KDG). The final outcome is to query the Data Lake in near real-time.

In this lab you will complete the following tasks:

1. Setup a Streaming Data Generator for Kinesis
2. Create Glue Streaming job
3. Query the data stream in Athena

   If you'd like to run the workshop on your own after the AWS hosted event, please follow the lab instruction here: https://github.com/aws-samples/data-engineering-for-aws-immersion-day

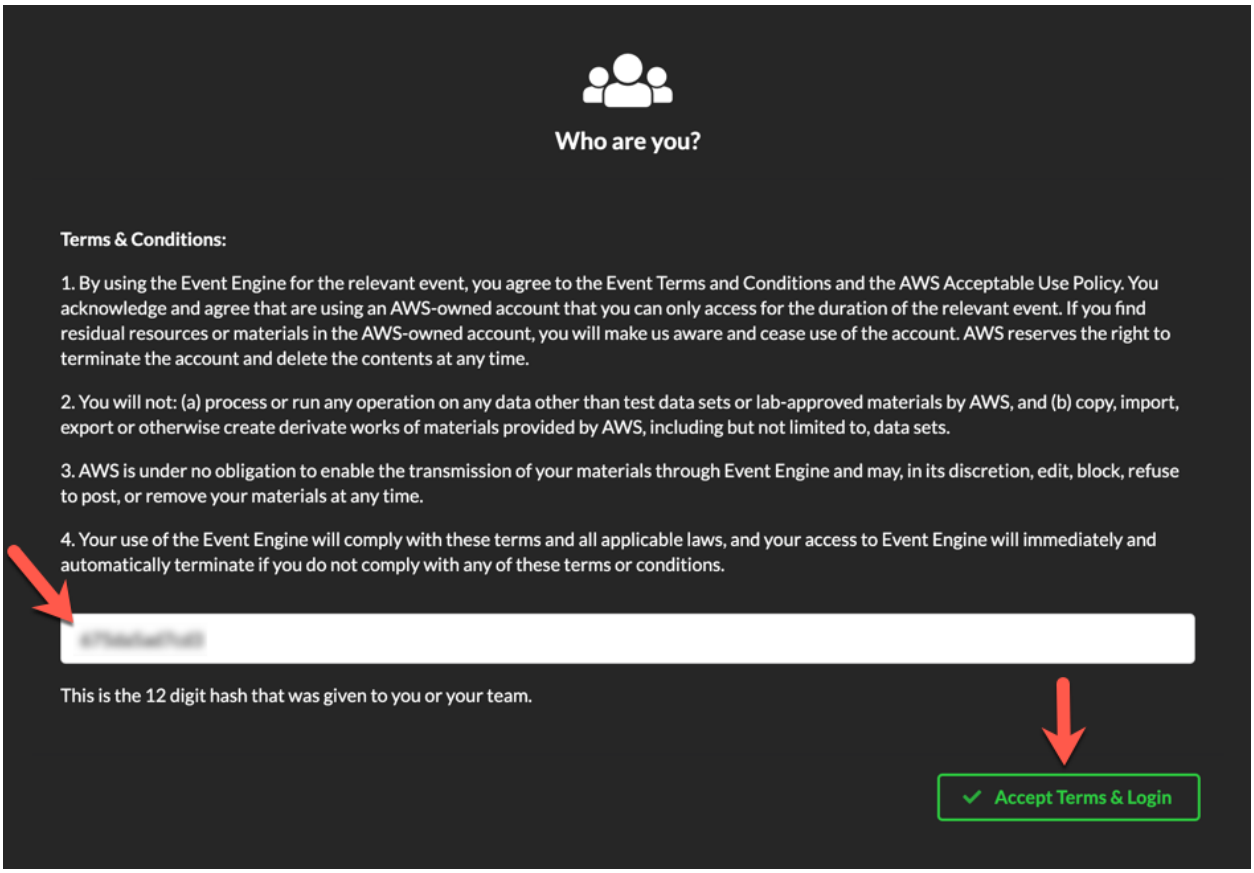
Go To Lab Environment

Please skip this section if you are running the lab on your own AWS account.

Today, you are attending a formal event and you will have been sent your access details beforehand. If in the future you might want to perform these labs in your own AWS environment by yourself, you can follow instructions on GitHub - https://github.com/aws-samples/data-engineering-for-aws-immersion-day.

A 12-character access code (or 'hash') is the access code that grants you permission to use a dedicated AWS account for the purposes of this workshop.

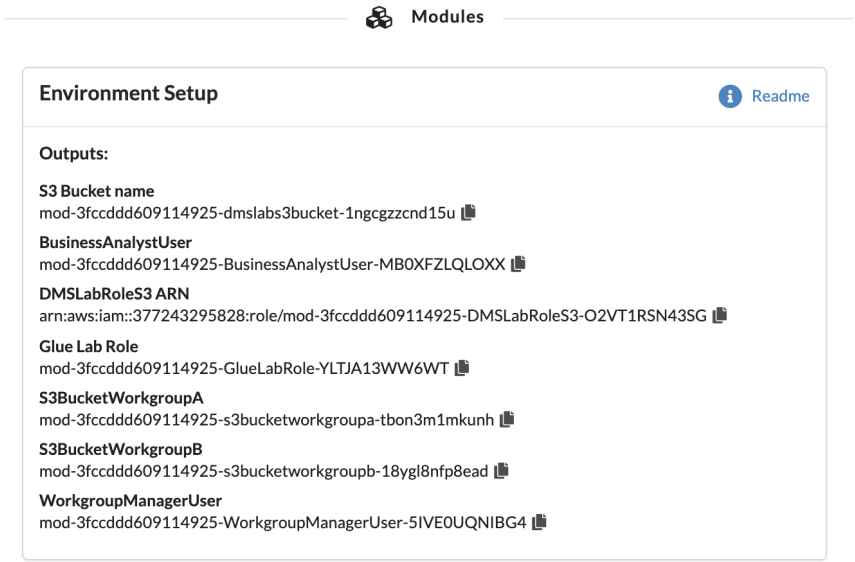
1. Go to https://dashboard.eventengine.run/, enter the access code and click Proceed:



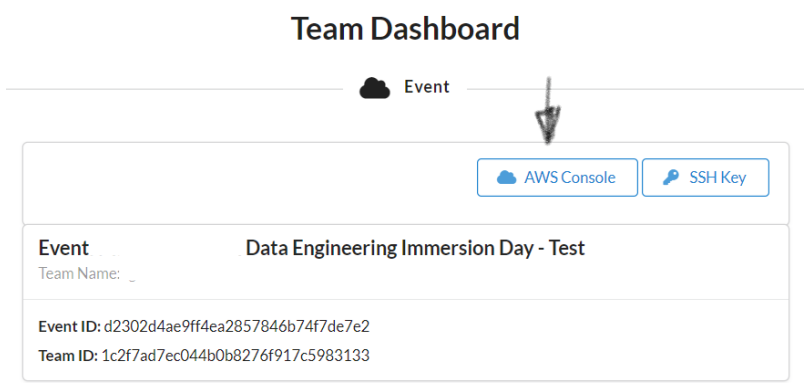
2. On the Team Dashboard web page, you will see a set of parameters that you will need during the labs. Best to save them to a text file locally, alternatively you can always go to this page to review them. Replace the parameters with the corresponding values from here where indicated in subsequent labs:

   Because you're at a formal event, some AWS resources have been pre-deployed for your convenience, for example:

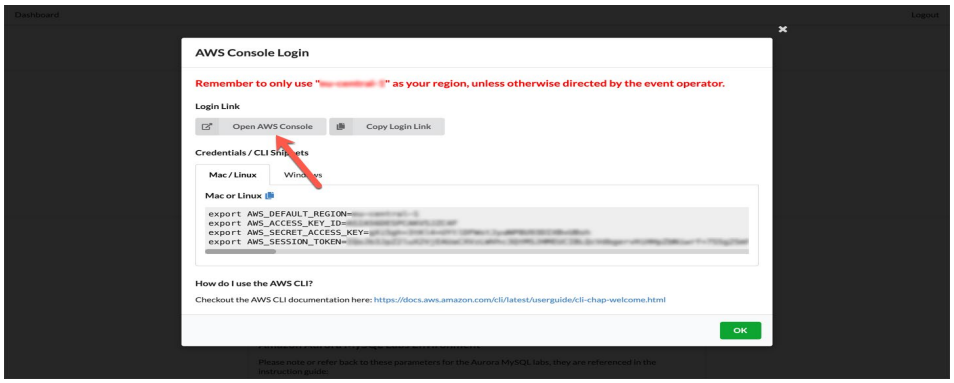
- S3 Bucket, IAM roles etc



3. On the Team Dashboard, please click AWS Console to log into the AWS Management Console:



4. Click Open AWS Console. For the purposes of this workshop, you will not need to use command line and API access credentials:



Once you have completed these steps, you can continue with the rest of this lab.

Setup Streaming Data Generator

We need an Amazon Kinesis Data Generator (Amazon KDG) to simulate the streaming data. If you have set up Amazon KDG with Kinesis Clickstream Lab, you should be able to reuse the tool. Otherwise, please follow the instruction in Streaming Data Prelab to launch the CloudFormation template, in order to set up your Amazon Kinesis Data Generator.

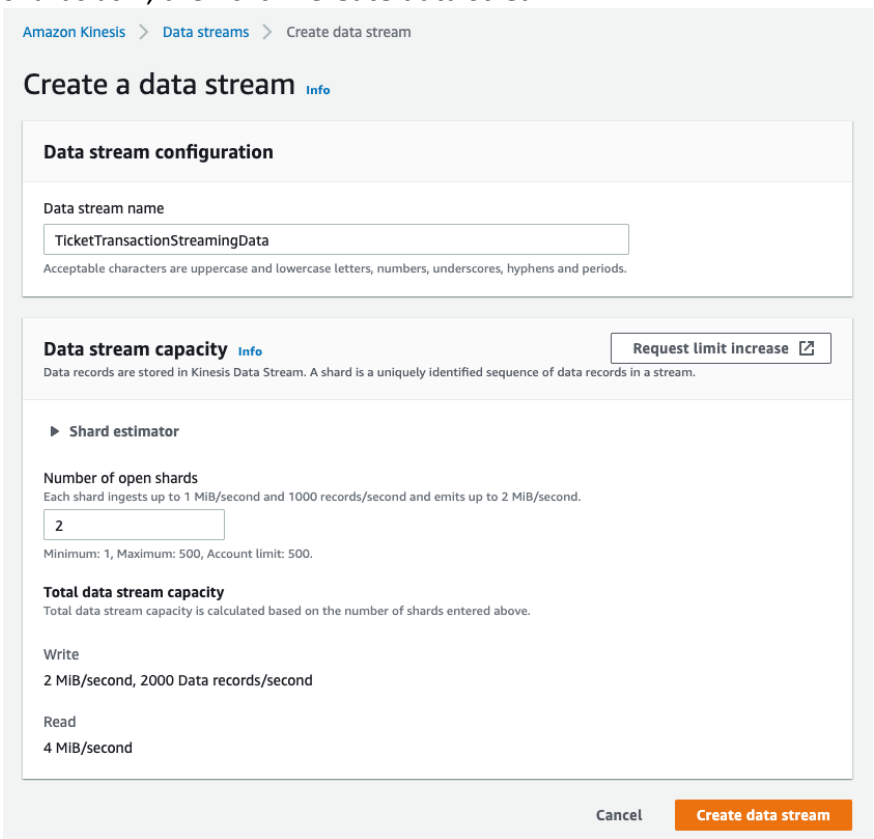
After the KDG setup is completed, you can find a URL from the output tab of the Streaming Data Prelab CloudFormation Stack, with the key name KinesisDataGeneratorUrl. Make sure you can login to the console using the username and password you provided when launching your prelab CloudFormation template. Bookmark the URL for further use.

Create Kinesis Data Stream

1. Navigate to **AWS Kinesis** console by using this link and make sure you are in the correct AWS region.

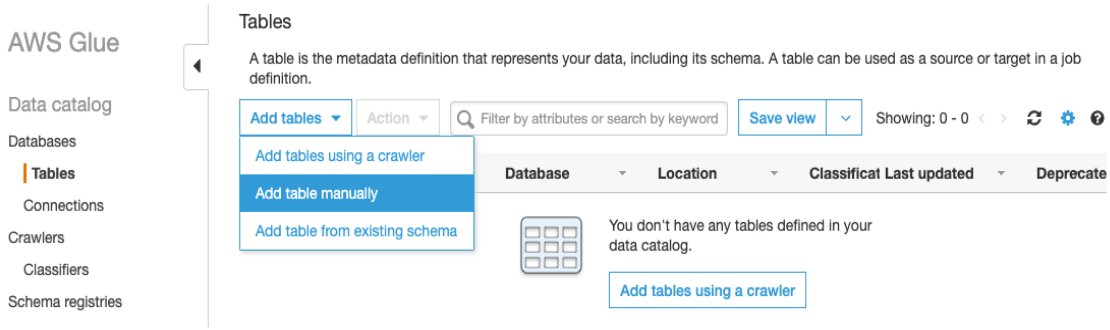
2. Click "**Create data stream**"

3. Put **TicketTransactionStreamingData** as data stream name and put number of open shards as 2, then click "**Create data stream**".



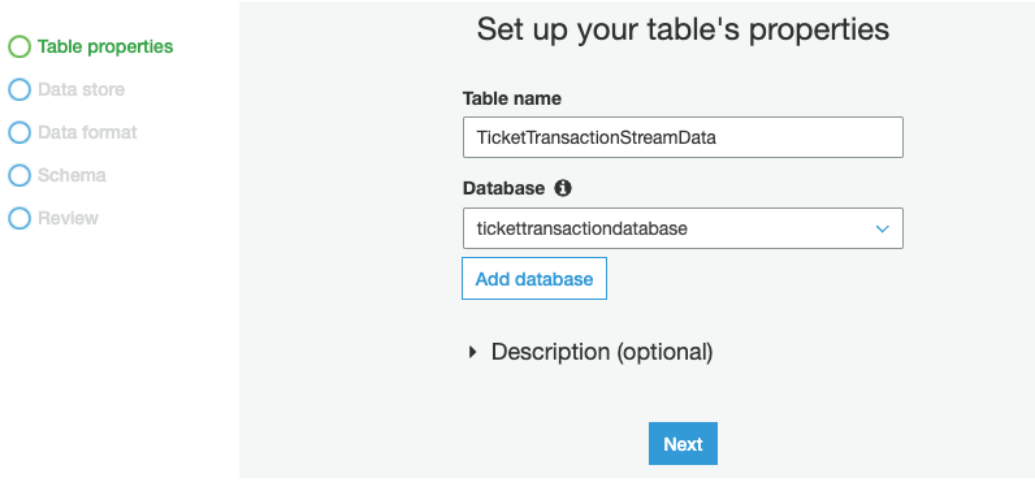
Create Table for Kinesis Stream Source in Glue Data Catalog

1. Navigate to **AWS Glue** console by using this link and make sure you are in the correct AWS region
2. On the AWS Glue menu, select Tables

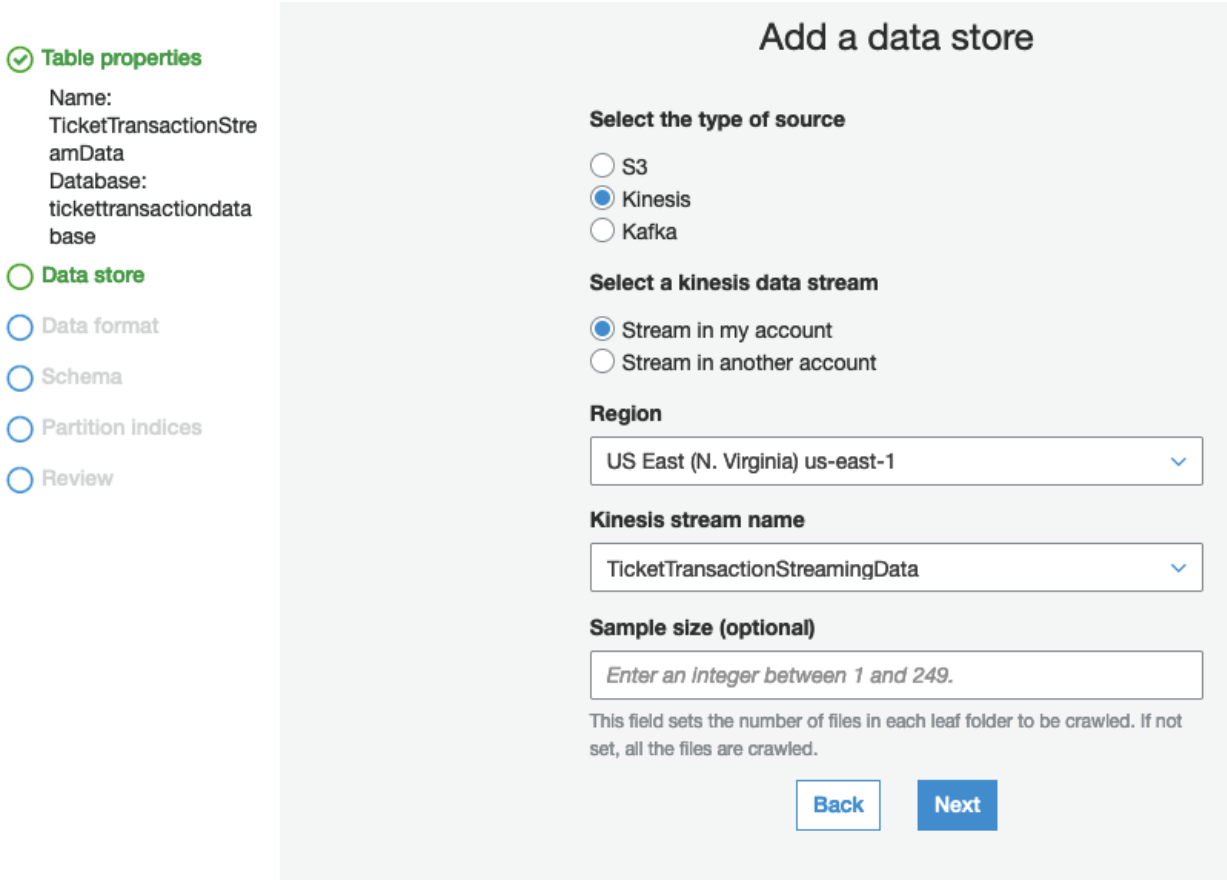


3. Put **TicketTransactionStreamData** as the table name
4. Click Add database and put **tickettransactiondatabase** as the database name, and click create.

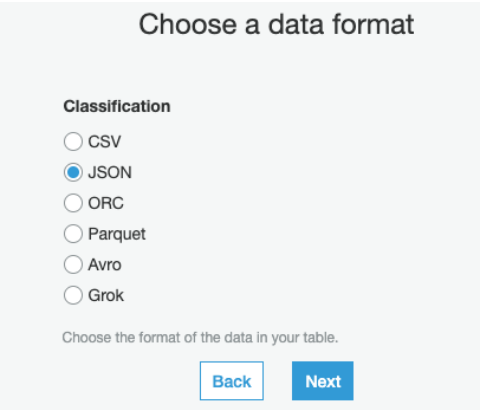
5. Using drop down to select the database we just created, and click Next



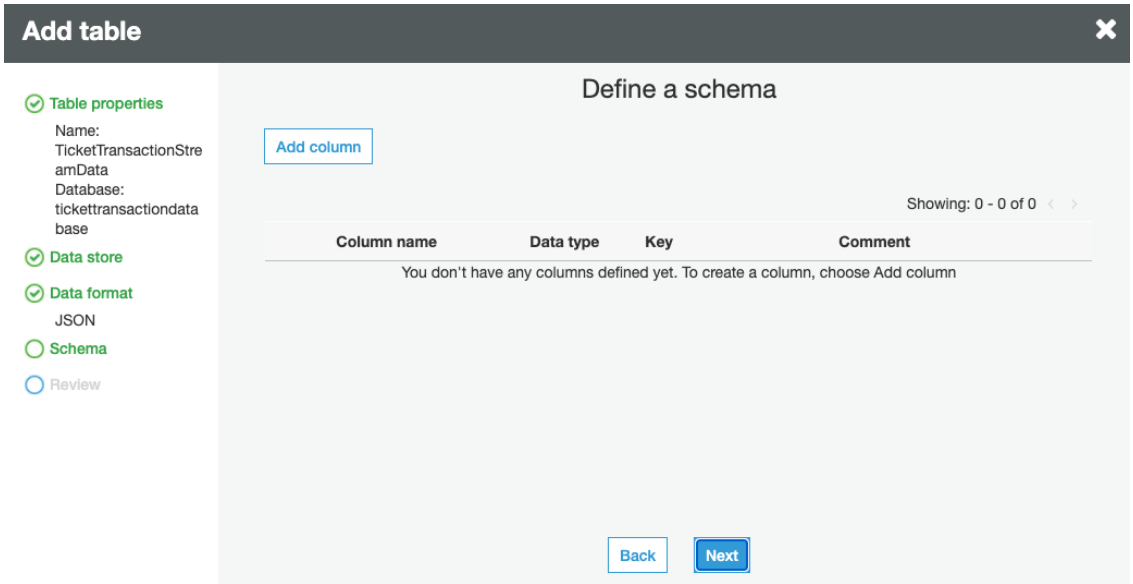
6. Select **Kinesis** as the source, select **Stream in my account** for Select a kinesis data stream, select the appropriate AWS region where you have created the stream, select the stream name as **TicketTransactionStreamingData** from the dropdown, and click **Next.**



7. Choose **JSON** as the incoming data format, as we will trigger JSON payload from Kinesis Data Generator in following steps. Click **Next**.



8. Leave the schema as empty, as we will enable schema detection feature when defining a Glue stream job. Click **Next**.

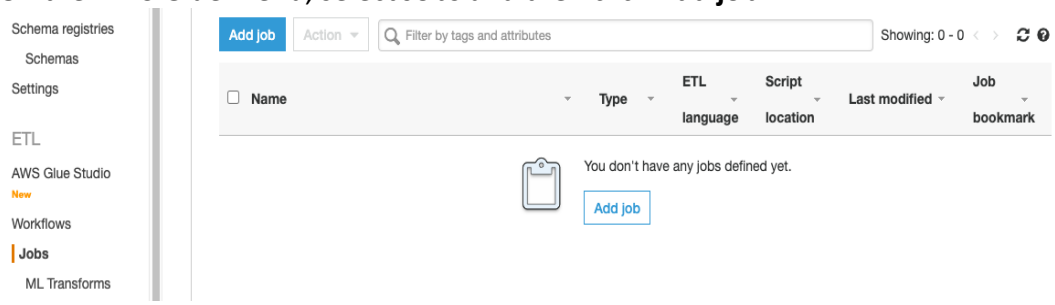


9. In the Add partition indices page, without adding anything, click **Next**. Review all the details and click **Finish**.

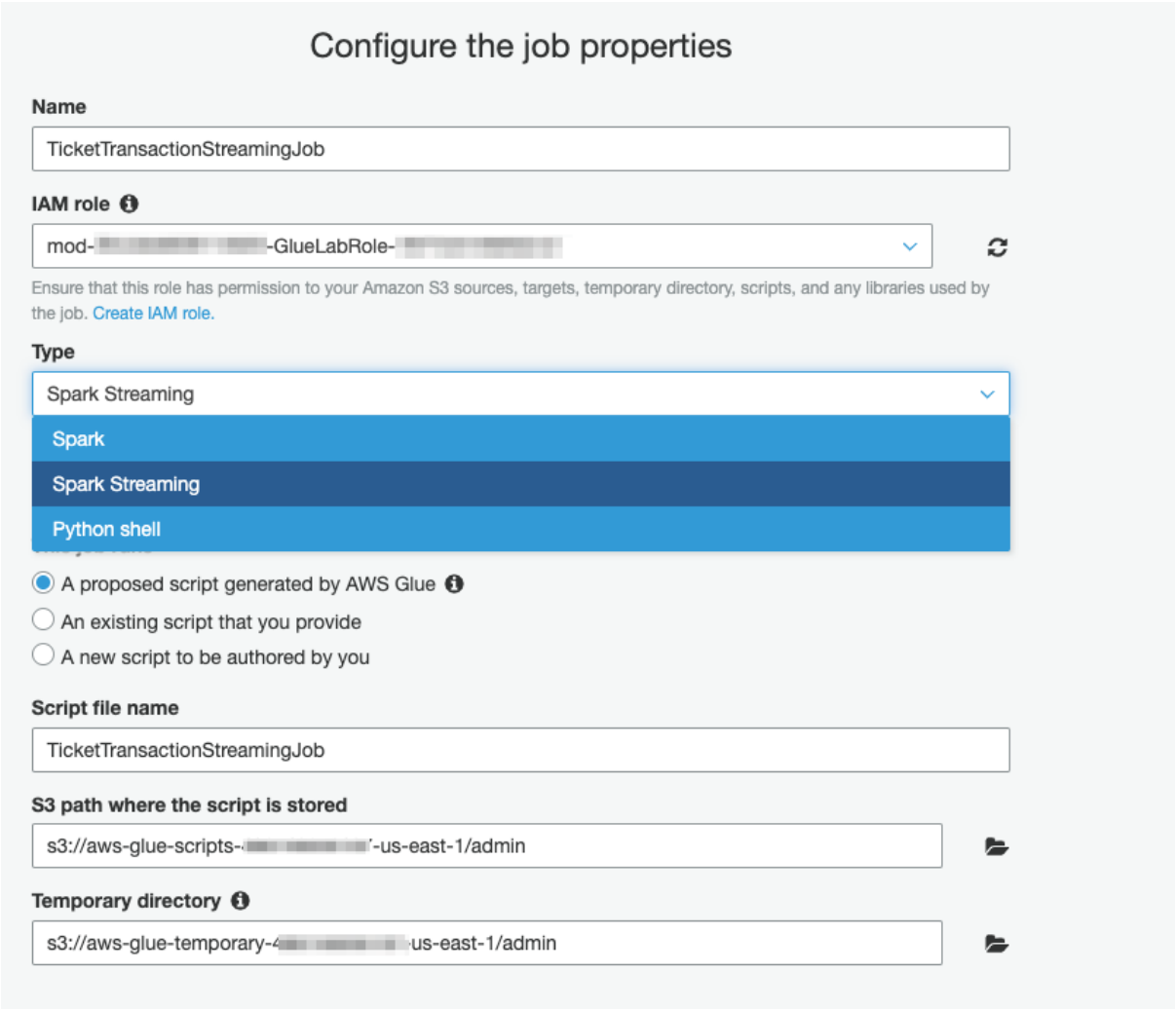
Create and trigger Glue Stream job

1. Navigate to **AWS Glue** console

2. On the AWS Glue menu, select **Jobs** and then click **Add job**

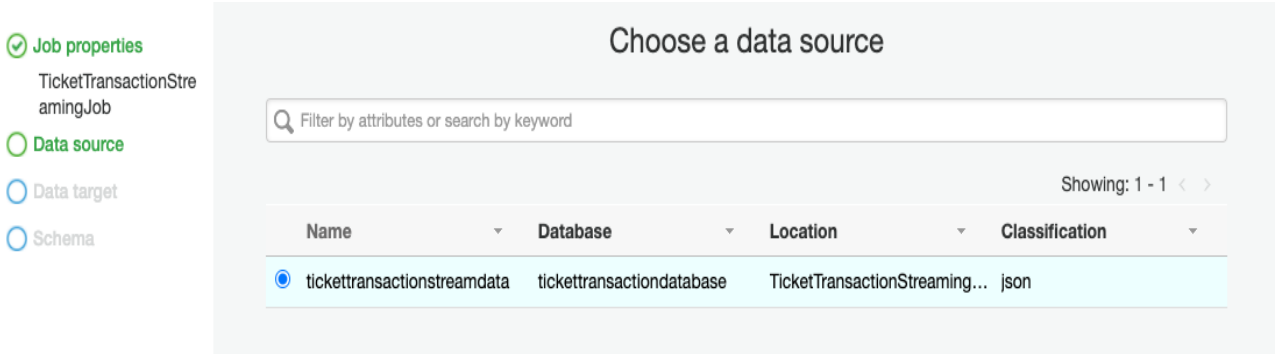


3. Put **TicketTransactionStreamingJob** as the job name, select the IAM role with "GlueLabRole" in the name. For job type, use dropdown list, select **Spark Streaming**;

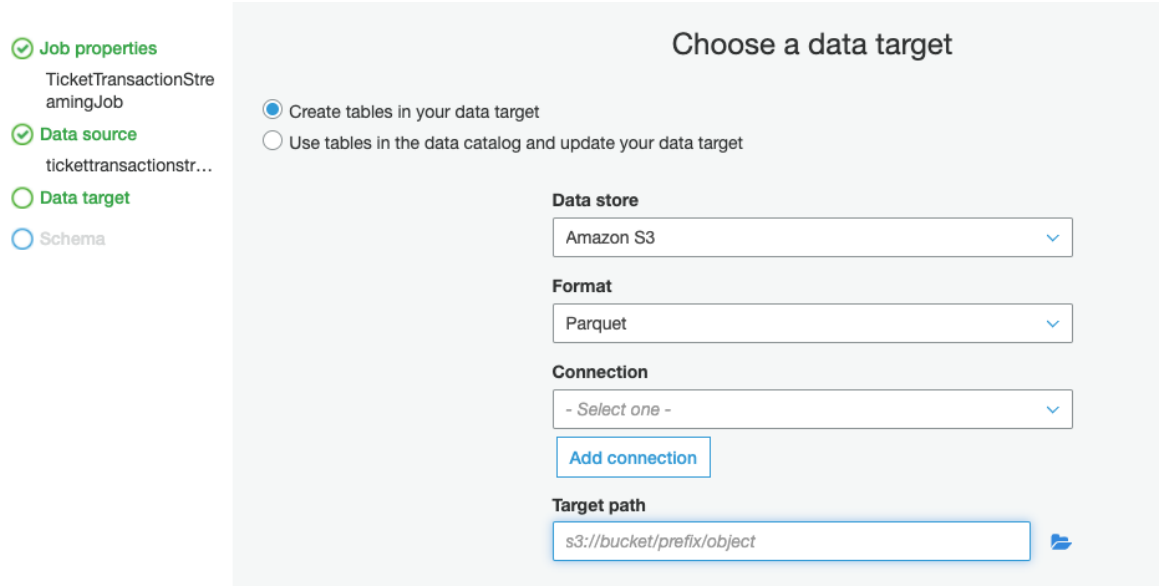


4. leave the rest configurations as is and click **Next**.

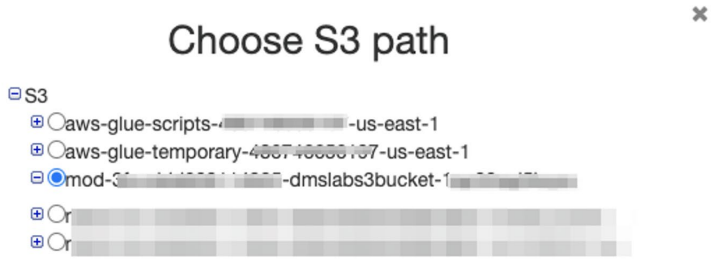
5. For Data source, select the data source **tickettransactionstreamdata**, then click **Next**.



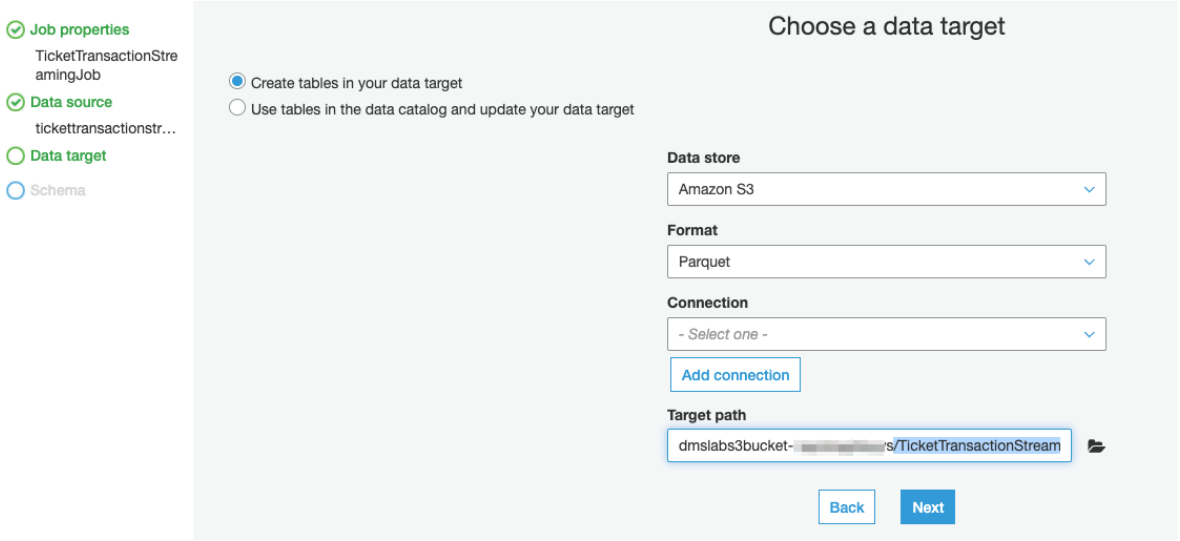
6. In Data target, select **Create tables in your data target**. In the Data store dropdown list, select **Amazon S3**. Select **Parquet** format from dropdown list.



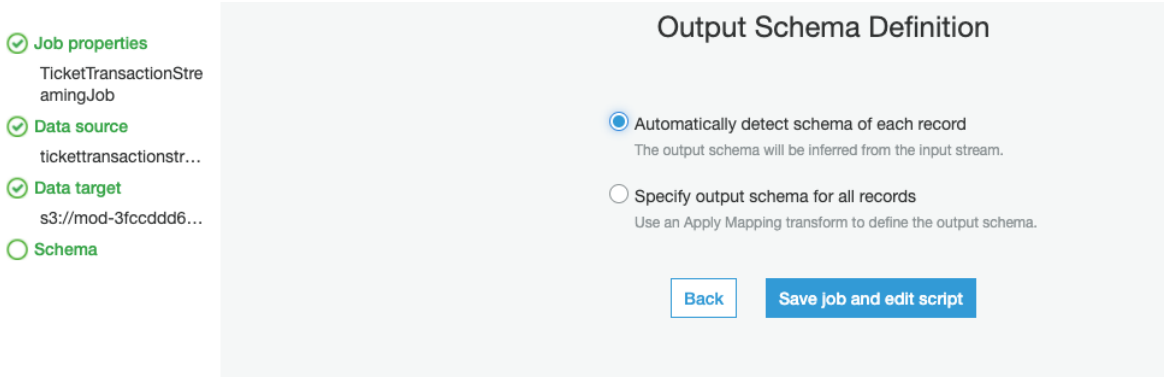
7. Click the **folder** button next to **Target path** to select a S3 bucket. From the pop-up window, select a S3 bucket name that contains "**dmslabs3bucket"**.



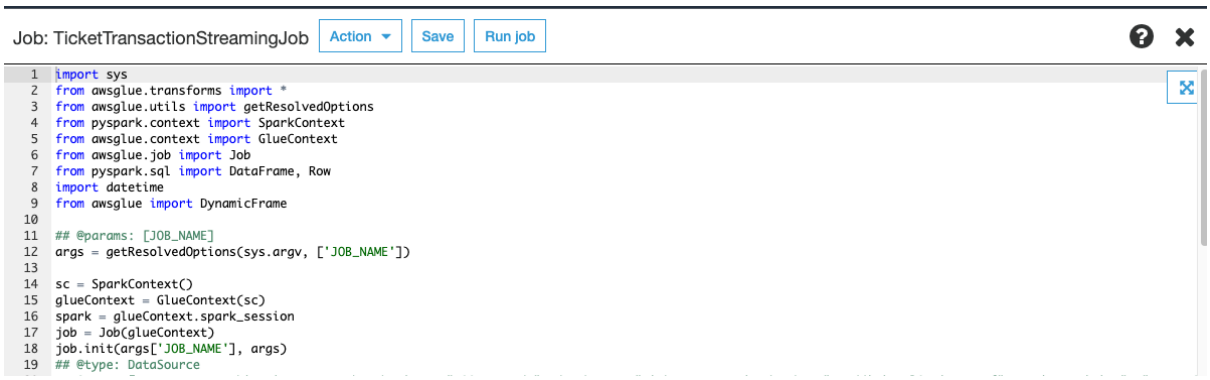
8. Make sure you add a path at the end **/TicketTransactionStreamingData**



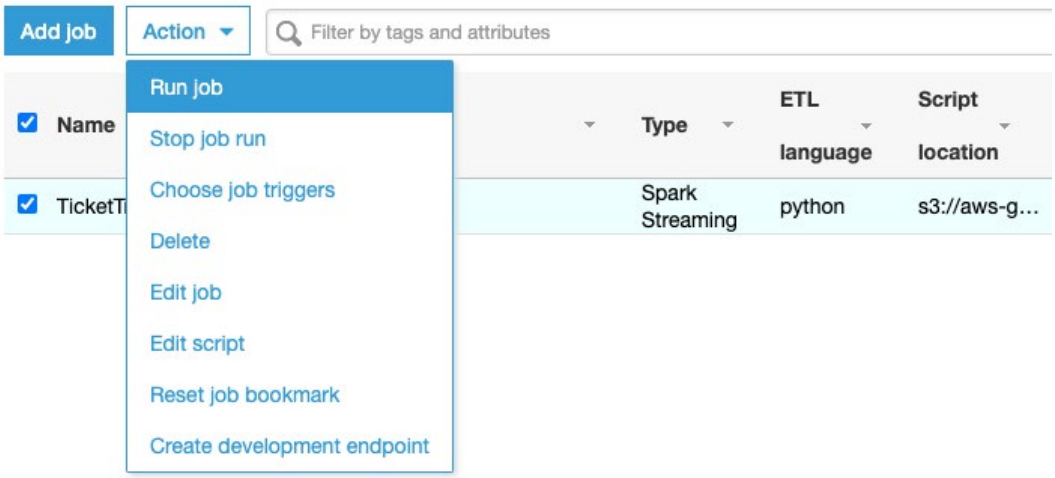
9. Make sure you select **Automatically detect schema of each record**, then click **Save job and edit script**.



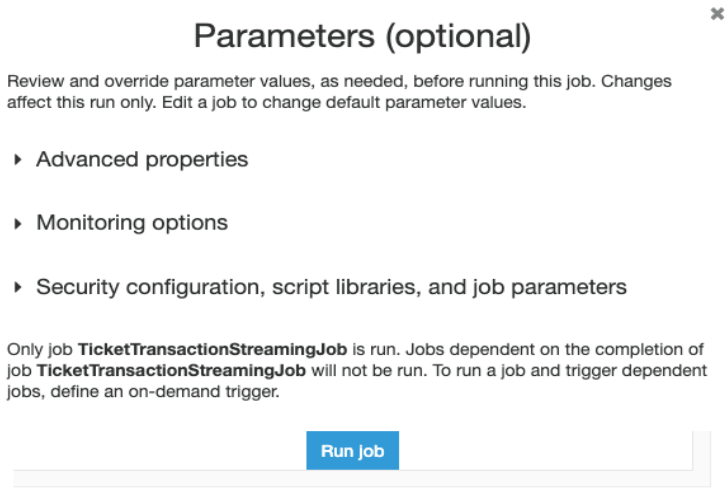
10. Review the generated script, click **Save** and then quit the editor.



11. Select the **TicketTransactionStreamingJob** we just created, from the Action dropdown list, select **Run job**.



12. Leave the optional parameters as default and click **Run job** to trigger the Glue Stream Job

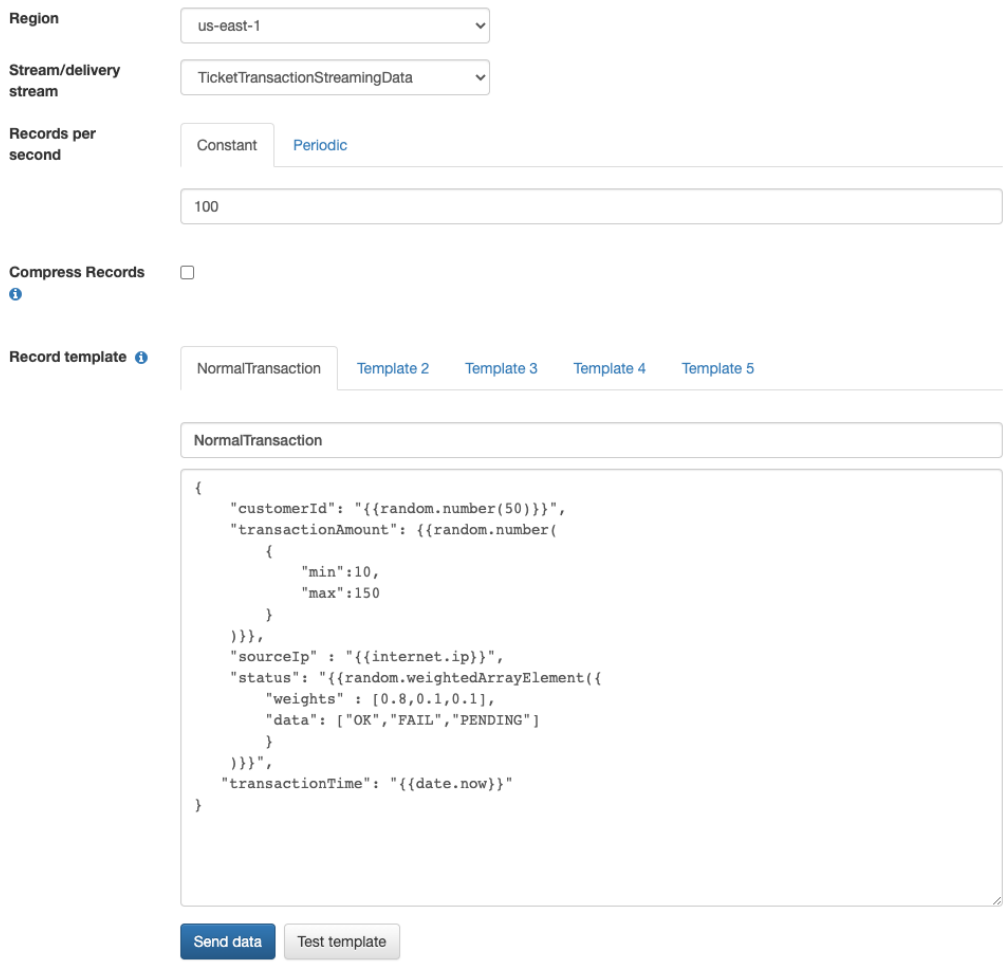


Trigger stream data from KDG

1. Launch KDG using the URL you bookmarked from the lab setup, login using the username and password you specified when deploying the CloudFormation stack.

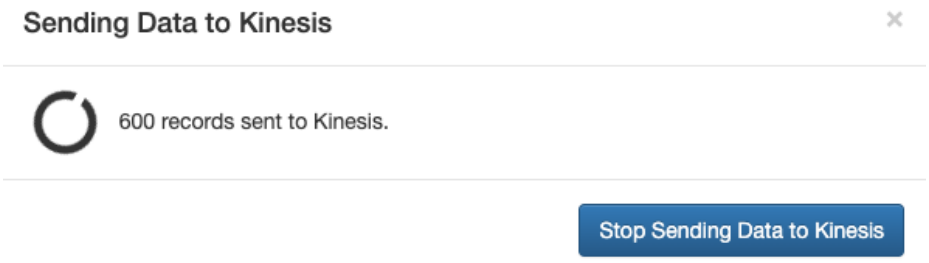
2. Make sure you select the appropriate region, from the dropdown list, select the **TicketTransactionStreamingData** as the target Kinesis stream, leave Records per second as default (**100** records per second); for the record template, type in **NormalTransaction** as the payload name, and copy the template payload as below:

```
{
    "customerId": "{{random.number(50)}}",
    "transactionAmount": {{random.number(
        {
            "min":10,
            "max":150
        }
    )}},
    "sourceIp" : "{{internet.ip}}",
    "status": "{{random.weightedArrayElement({
        "weights" : [0.8,0.1,0.1],
        "data": ["OK","FAIL","PENDING"]
        }
    )}}",
   "transactionTime": "{{date.now}}"
}
```



To learn more about what the payload will look like when sending from KDG simulator, refer to the document as this link, https://awslabs.github.io/amazon-kinesis-data-generator/web/help.html

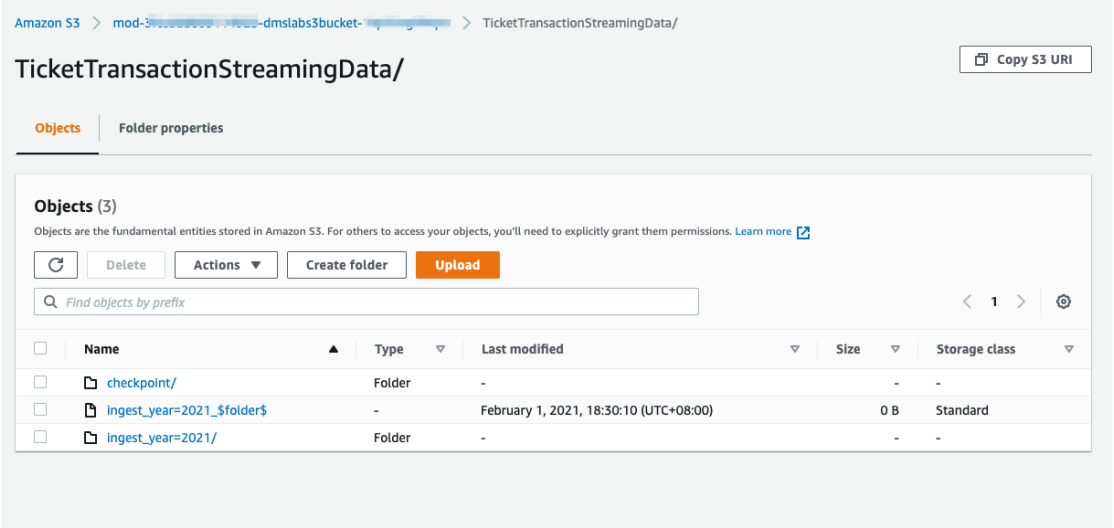
3. Click **Send data** to trigger the simulated ticket purchasing transaction streaming data.



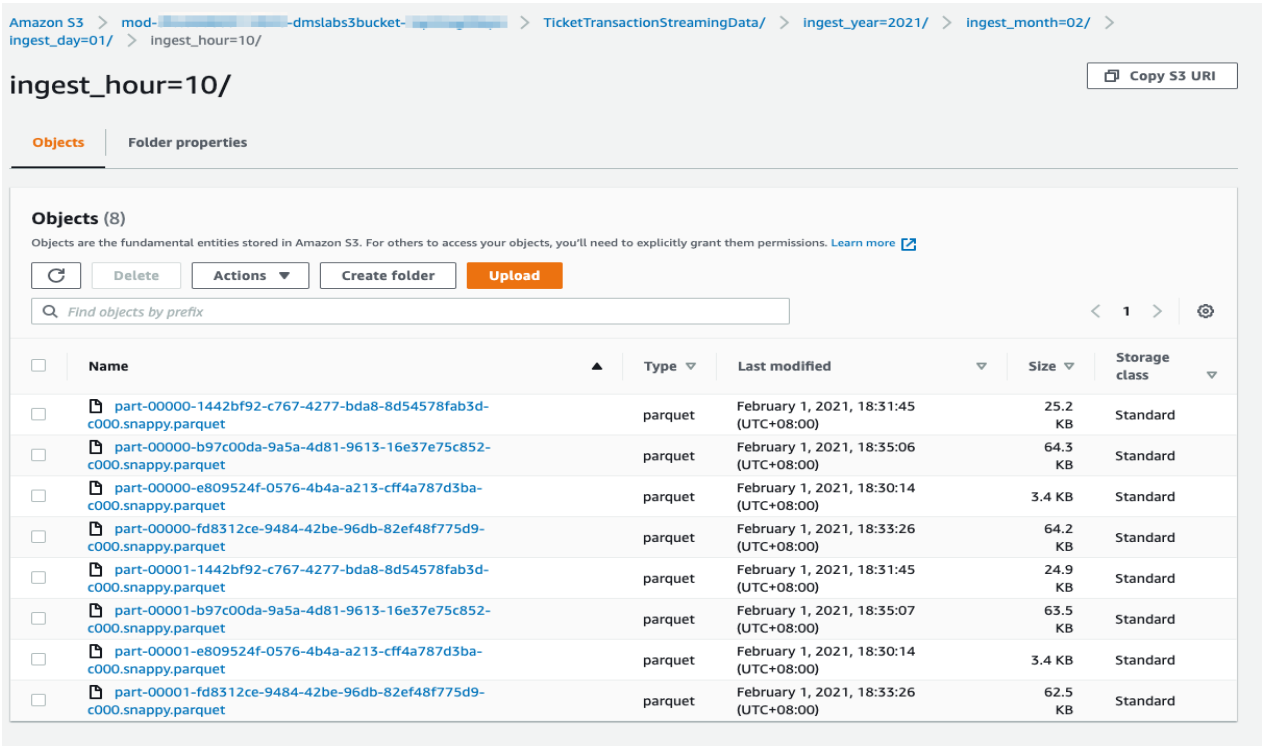
Verify the Glue stream job

1. Navigate to **Amazon S3** console by using this link https://s3.console.aws.amazon.com/s3/home

2. Navigate to the S3 bucket path we've set as Glue Stream Job target, note the folder structure of the processed data.



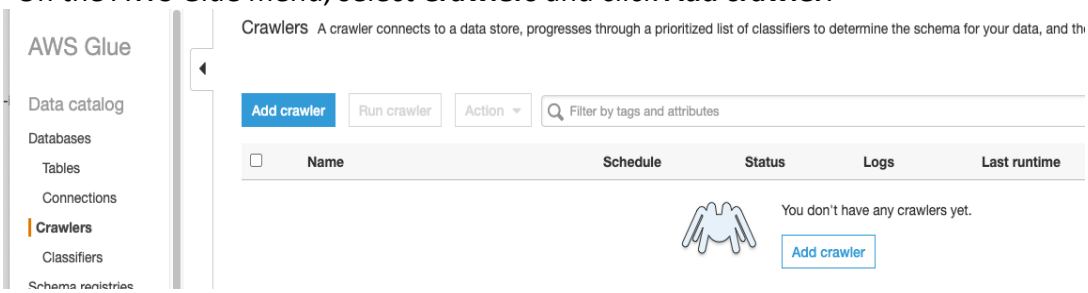
3. Check the folder content using current date and time as the folder name. Verify that the streaming data has been transformed into parquet format and persisted into corresponding folders.



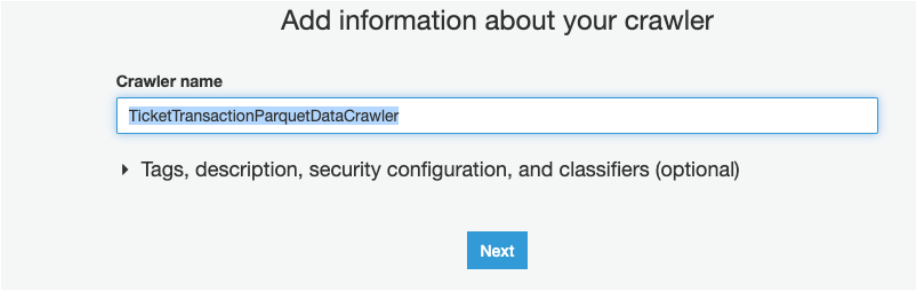
Create Glue Crawler for the transformed data

1. Navigate to AWS Glue console and make sure that you are in the correct AWS region

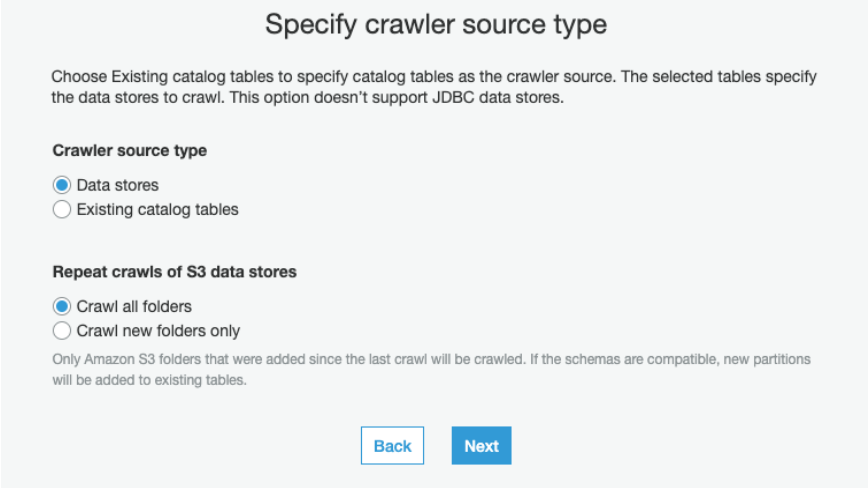
2. On the AWS Glue menu, select **Crawlers** and click **Add crawler**.



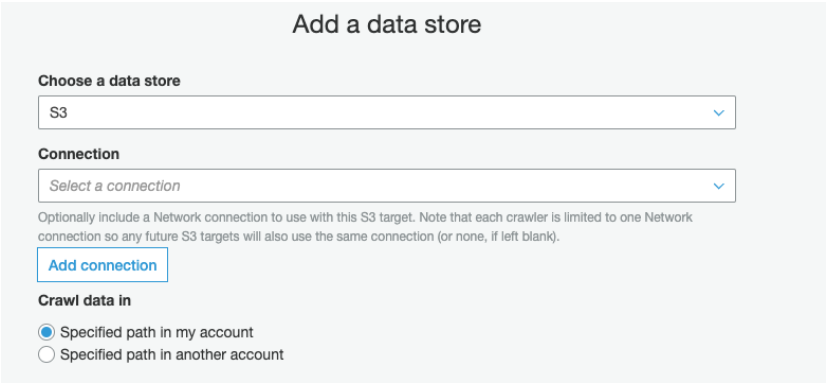
3. Put **TicketTransactionParquetDataCrawler** as the name of the crawler, click **Next**.



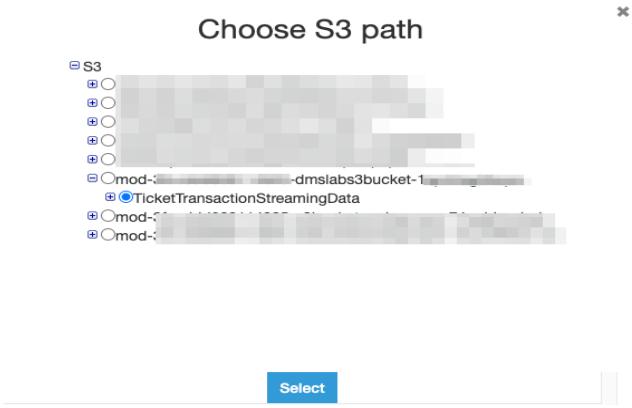
4. Leave the default to specify **Data stores** as Crawler source type and **Crawl all folders**, click **Next**.



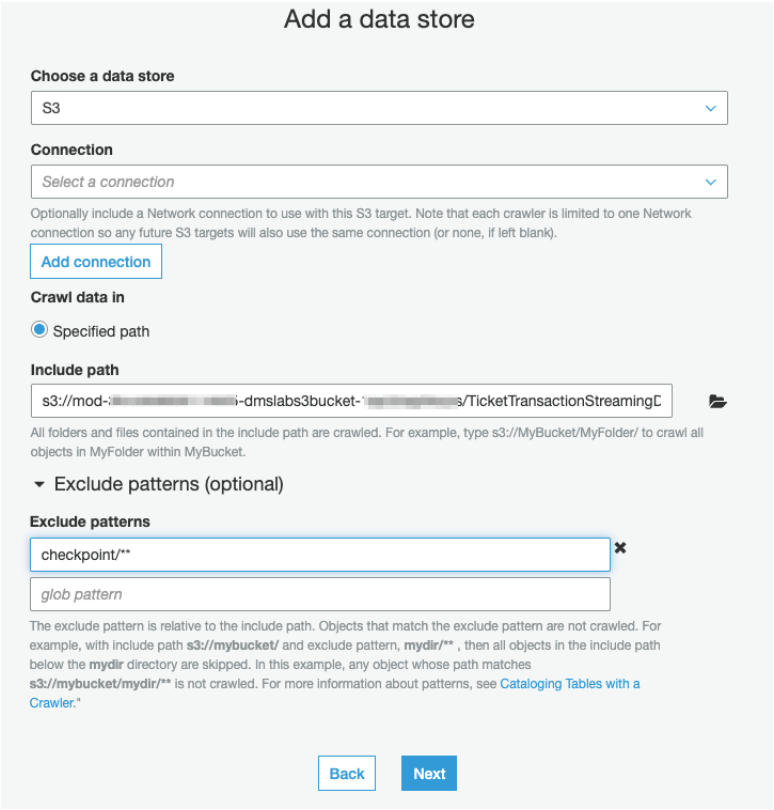
5. Choose S3 as data store and choose Specified path in my account.



6. Click the icon next to Include path input to select the S3 bucket. Make sure you select the folder **TicketTransactionStreamingData**. Click **Select**.

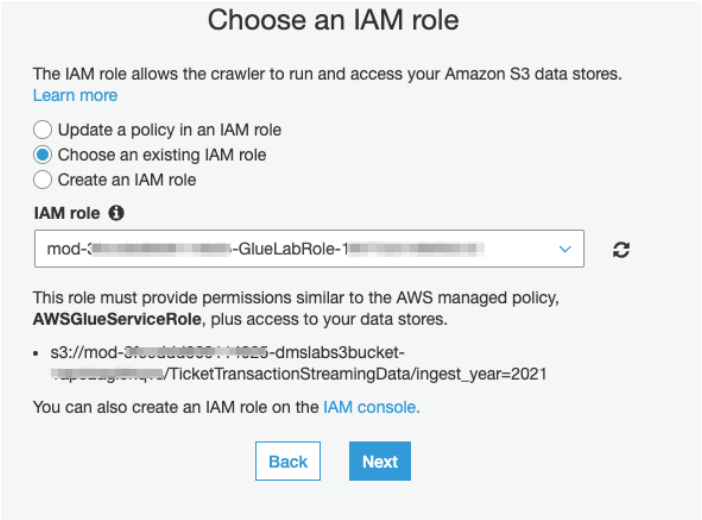


7. Expand the **Exclude patterns**, put **checkpoint/**** to exclude the data in checkpoint folder. Review the current input and click **Next**.

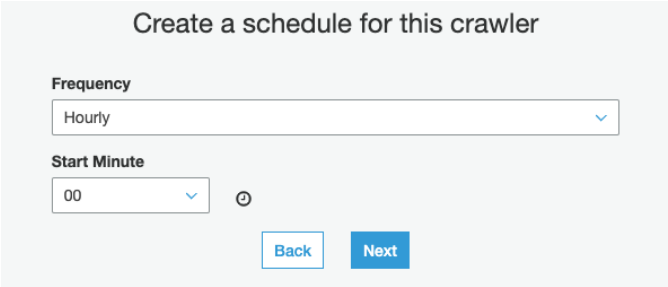


8. Select No to indicate no other data store needed, then click Next.

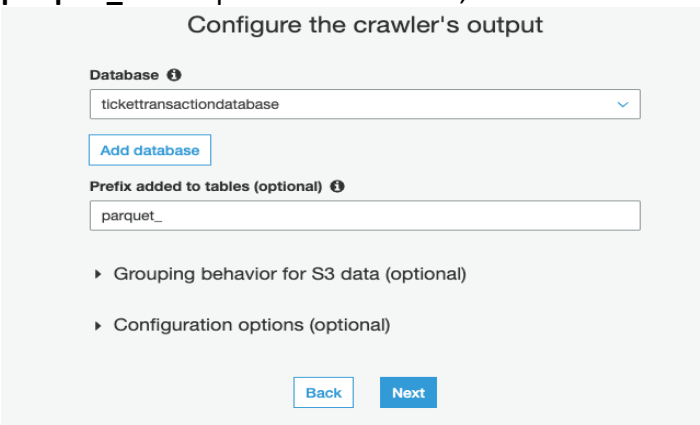
9. Choose an existing IAM role, using the dropdown list to select the role with **GlueLabRole** in the name, click **Next.**



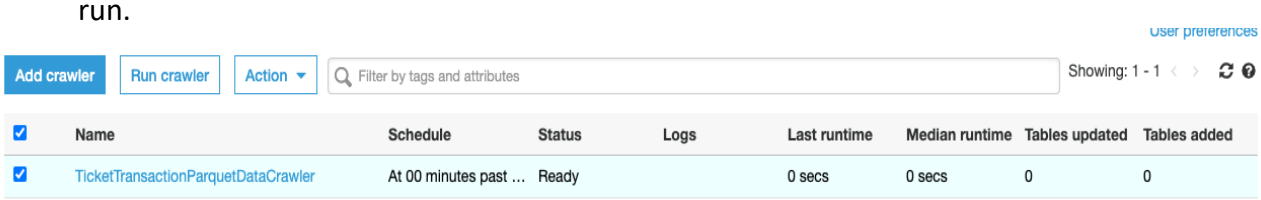
10. As the data is partitioned to hour, so we set the crawler to run every hour to make sure the newly added partition is added. Click **Next**.



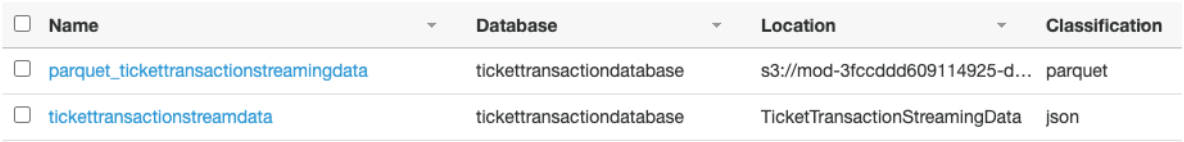
11. Using the dropdown list to select **tickettransactiondatabase** as the output database, use **parquet_** as the prefix for the table, click **Next**.



12. Review the crawler configuration and click **Finish** to create the crawler.
13. Once the crawler is created, select the crawler and click **Run crawler** to trigger the first run.



14. When the crawler job has stopped, go to Glue Data catalog, under Tables, verify that **parquet_tickettransactionstreamingdata** table is listed.



15. Click the **parquet_tickettransactionstreamingdata** table, verify that Glue has correctly identified the streaming data format while transforming source data from Json format to Parquet.

Tables > parquet_tickettransactionstreamingdata

Last updated 1 Feb 2021 07:39 PM **Table** Version (Current version)

Edit table | Delete table | View partitions | Compare versions | Edit schema

Name	parquet_tickettransactionstreamingdata
Description	
Database	tickettransactiondatabase
Classification	parquet
Location	s3://mod-[redacted]-dmslabs3bucket-[redacted]/TicketTransactionStreamingData/
Connection	
Deprecated	No
Last updated	Mon Feb 01 19:39:02 GMT+800 2021
Input format	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
Output format	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
Serde serialization lib	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
Serde parameters	serialization.format **1**
Table properties	sizeKey **1121888** objectCount **28** UPDATED_BY_CRAWLER **TicketTransactionParquetDataCrawler** CrawlerSchemaSerializerVersion **1.0**
	recordCount **85300** averageRecordSize **20**
	exclusions **["s3://mod-[redacted]-dmslabs3bucket-[redacted]/TicketTransactionStreamingData/checkpoint/**"]**
	CrawlerSchemaDeserializerVersion **1.0** compressionType **none** typeOfData **file**

Schema

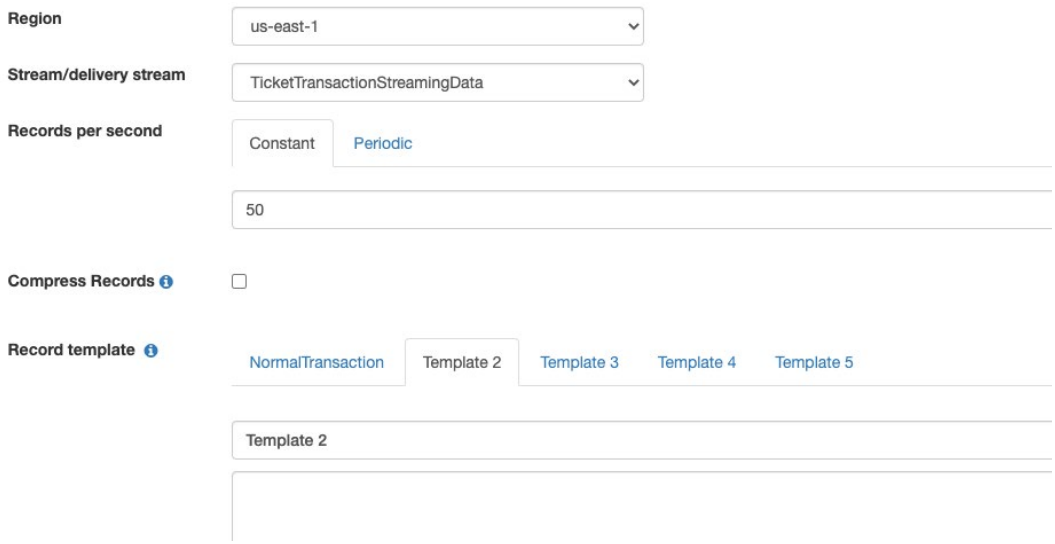
Showing: 1 - 9 of 9

	Column name	Data type	Partition key	Comment
1	customerid	string		
2	sourceip	string		
3	status	string		
4	transactionamount	bigint		
5	transactiontime	string		
6	ingest_year	string	Partition (0)	
7	ingest_month	string	Partition (1)	
8	ingest_day	string	Partition (2)	
9	ingest_hour	string	Partition (3)	

Trigger abnormal transaction data from KDG

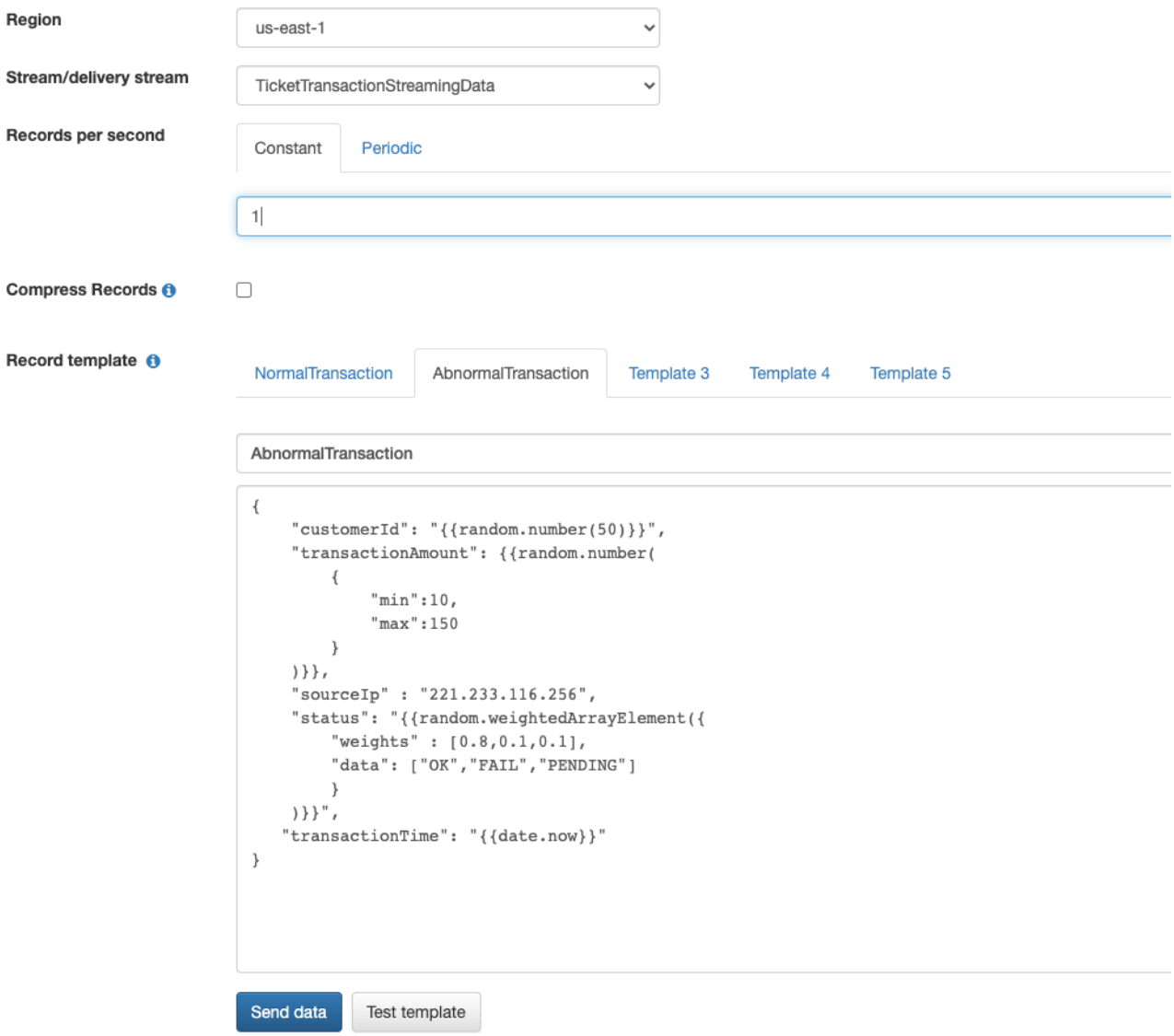
1. Keep the KDG streaming data running, open another browser and launch KDG using the URL you bookmarked from the lab setup, login using the username and password you provided when launching the CloudFormation template.

2. Make sure you select the appropriate region, from the dropdown list, select the **TicketTransactionStreamingData** as the target Kinesis stream, put Records per second as 1; click Template 2, and prepare to copy abnormal transaction data,



3. for the record template, type in **AbnormalTransaction** as the payload name, and copy the template payload as below:

```
{
   "customerId": "{{random.number(50)}}",
   "transactionAmount": {{random.number(
      {
         "min":10,
         "max":150
      }
   )}},
   "sourceIp" : "221.233.116.256",
   "status": "{{random.weightedArrayElement({
      "weights" : [0.8,0.1,0.1],
      "data": ["OK","FAIL","PENDING"]
      }
   )}}",
  "transactionTime": "{{date.now}}"
}
```

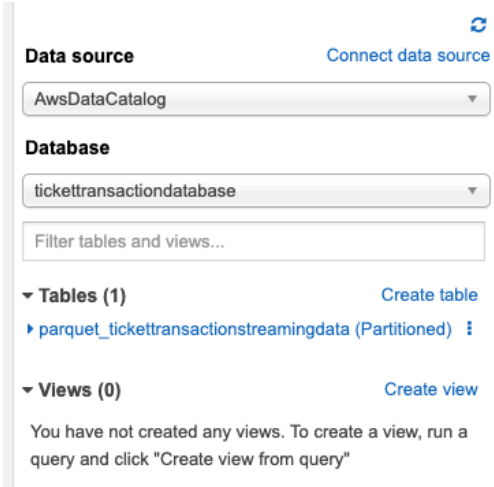


4. Click Send data to simulate abnormal transactions (1 transaction per second all from the same source IP address).

Detect Abnormal Transactions using Ad-Hoc query from Athena

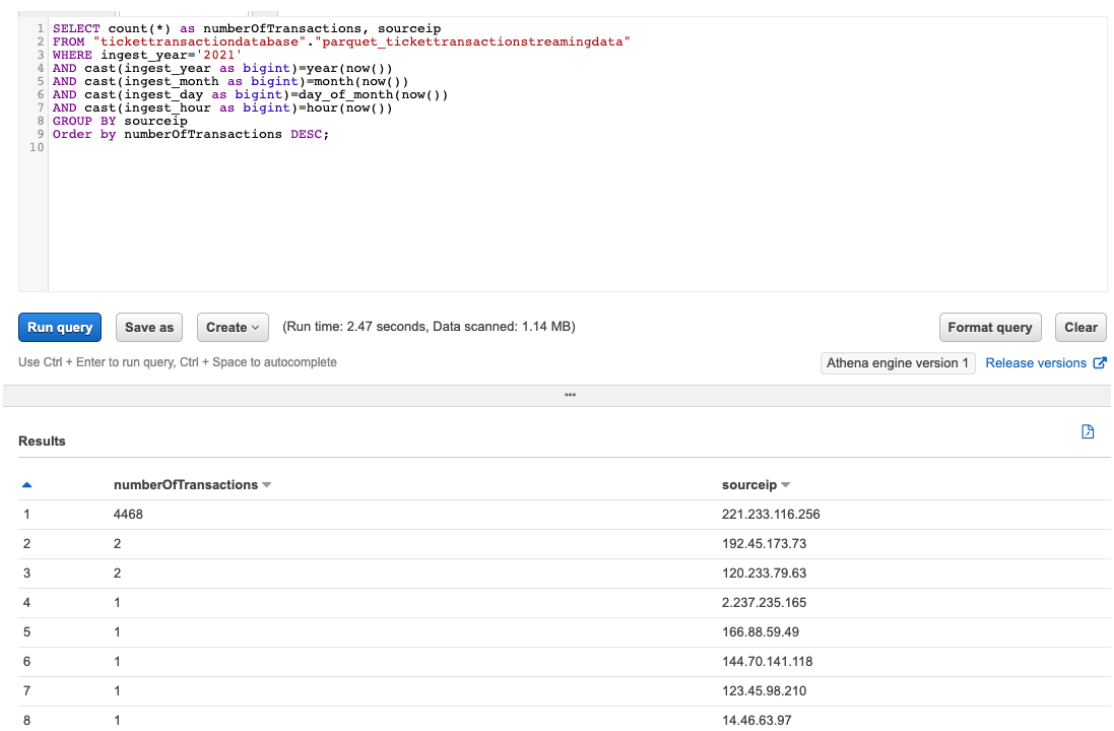
1. Navigate to **AWS Athena** console by using this link https://console.aws.amazon.com/athena/home

2. Make sure you select **AwsDataCatalog** as Data source and **tickettransactiondatabase** as the database, refresh to make sure the **parquet_tickettransactionstreamingdata** is showing in the table list.



3. Copy query as below, this is to query last hour the number of transactions by sourceip. You should see there's large number of transactions from the same sourceip.

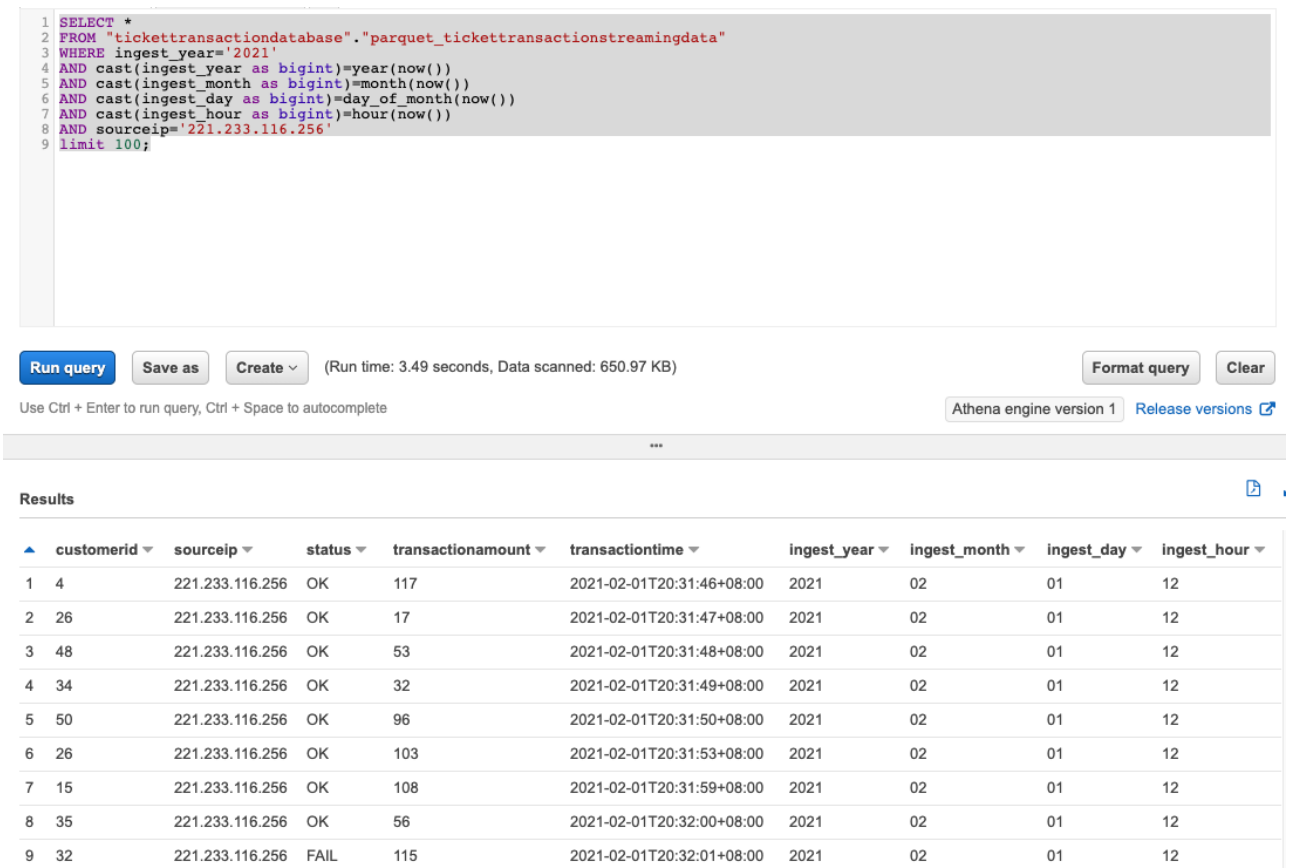
```
SELECT count(*) as numberOfTransactions, sourceip
FROM "tickettransactiondatabase"."parquet_tickettransactionstreamingdata"
WHERE ingest_year='2021'
AND cast(ingest_year as bigint)=year(now())
AND cast(ingest_month as bigint)=month(now())
AND cast(ingest_day as bigint)=day_of_month(now())
AND cast(ingest_hour as bigint)=hour(now())
GROUP BY sourceip
Order by numberOfTransactions DESC;
```



	numberOfTransactions	sourceip
1	4468	221.233.116.256
2	2	192.45.173.73
3	2	120.233.79.63
4	1	2.237.235.165
5	1	166.88.59.49
6	1	144.70.141.118
7	1	123.45.98.210
8	1	14.46.63.97

4. Copy query as below, this is to further check if the transaction details from the same source IP. The query verified that the request is coming from same IP but with different customer id, so it's verified as abnormal transactions.

```
SELECT *
FROM "tickettransactiondatabase"."parquet_tickettransactionstreamingdata"
WHERE ingest_year='2021'
AND cast(ingest_year as bigint)=year(now())
AND cast(ingest_month as bigint)=month(now())
AND cast(ingest_day as bigint)=day_of_month(now())
AND cast(ingest_hour as bigint)=hour(now())
AND sourceip='221.233.116.256'
limit 100;
```



	customerid	sourceip	status	transactionamount	transactiontime	ingest_year	ingest_month	ingest_day	ingest_hour
1	4	221.233.116.256	OK	117	2021-02-01T20:31:46+08:00	2021	02	01	12
2	26	221.233.116.256	OK	17	2021-02-01T20:31:47+08:00	2021	02	01	12
3	48	221.233.116.256	OK	53	2021-02-01T20:31:48+08:00	2021	02	01	12
4	34	221.233.116.256	OK	32	2021-02-01T20:31:49+08:00	2021	02	01	12
5	50	221.233.116.256	OK	96	2021-02-01T20:31:50+08:00	2021	02	01	12
6	26	221.233.116.256	OK	103	2021-02-01T20:31:53+08:00	2021	02	01	12
7	15	221.233.116.256	OK	108	2021-02-01T20:31:59+08:00	2021	02	01	12
8	35	221.233.116.256	OK	56	2021-02-01T20:32:00+08:00	2021	02	01	12
9	32	221.233.116.256	FAIL	115	2021-02-01T20:32:01+08:00	2021	02	01	12