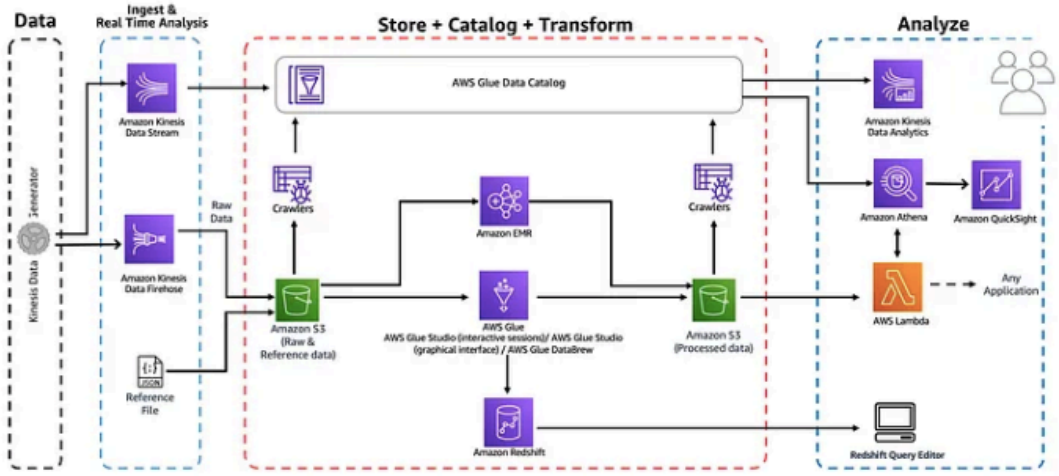
Analytics on AWS



Amazon Athena and AWS Glue have emerged as powerful tools for seamlessly querying and processing data stored in Amazon S3. Harness the potential of Athena and Glue to analyse data, create databases, and execute SQL queries effortlessly.

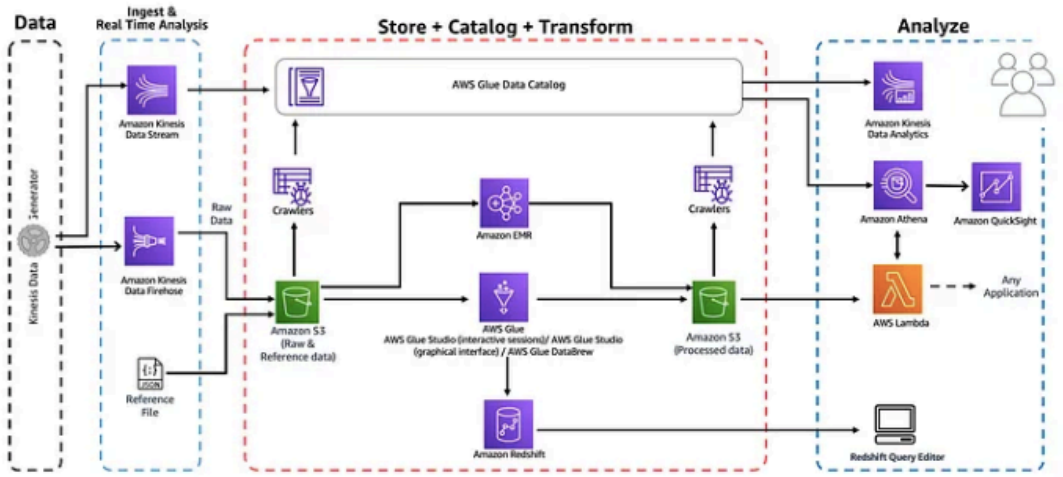
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Implement:

The workshop explores the capabilities of **Amazon Athena**, a serverless query service, and **AWS Glue**, a fully-managed ETL service.

In this workshop, we will go over a sequence of modules, covering various aspects of building an analytics platform on AWS. You will learn to **ingest**, **store**, **transform** and **consume** data using several analytics services such as **AWS Glue**, **Amazon Athena**, **Amazon Kinesis**, **Amazon QuickSight** as well as **AWS Lambda**.
Note: We need AWS account with **AdminstratorAccess** and this lab should be executed in **us-east-1** region

A more detailed representation of the design is presented in the architecture below -



XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Tasks:

1. Ingest and Store
   a) Navigate to S3 Console & Create a **new bucket** in us-east-1 region and add reference data.
   b) Create a **Kinesis Firehose delivery stream** to ingest data & store in S3
   c) **Generate Dummy data** — configure Kinesis Data Generator to produce fake data and ingest it into Kinesis Firehose.
   d) Validate that data has arrived in S3.

2. Catalog Data
   a) Create **IAM** Role
   b) Create **glue crawlers** to discovery the schema of the newly ingested data in S3.
   c) Verify **newly created tables in catalog**
   d) Query ingested data using **Amazon Athena**

3. Data transformation
   Transform Data with AWS Glue Studio (interactive sessions)
   a) Prepare **IAM Policies and Rules**.
   b) Use **Jupyter Notebook in AWS Glue** for interactive ETL development.

4. Analyze with Athena

5. Visualize in Quicksight

6. Serve with Lambda

7. Cleanup

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

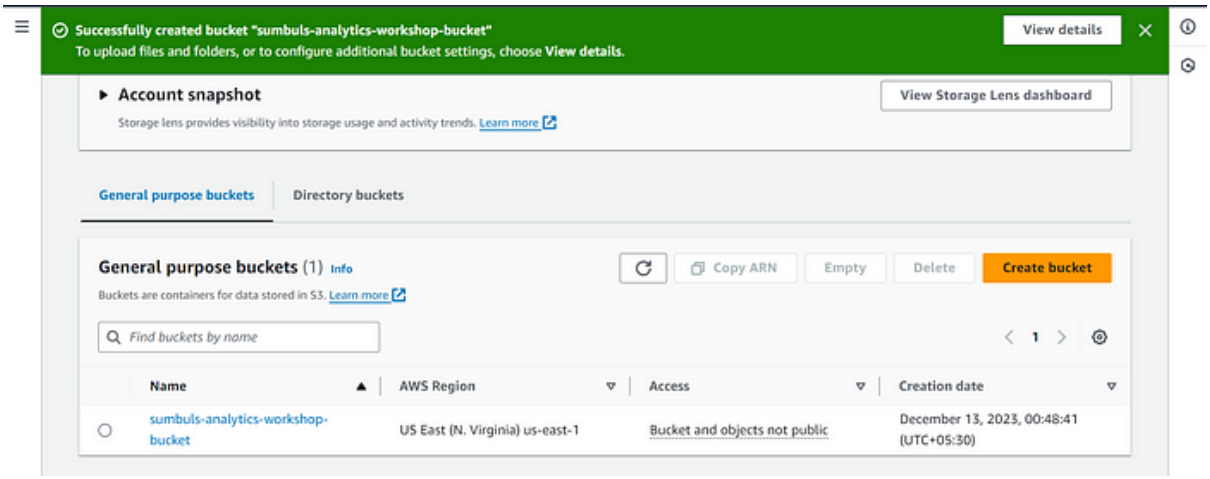
Solution:

Task 1 👉: Ingest and Store

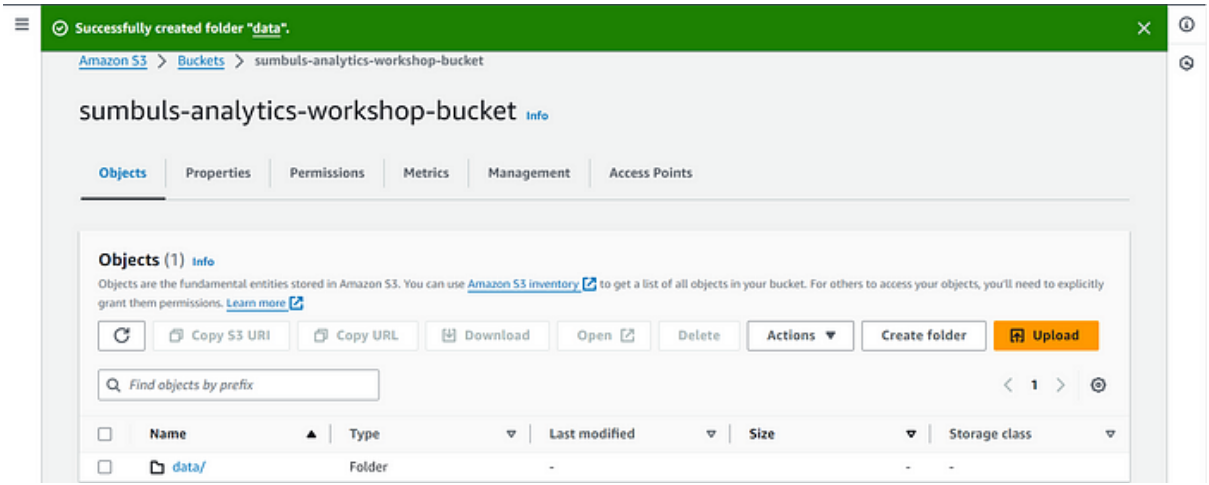
Generate some dummy data in near real-time using Kinesis data generator utility, and deliver the data to Amazon S3 with Kinesis Firehose delivery stream. We will also copy some reference data directly into Amazon S3 bucket.

a) Navigate to S3 Console & Create a new bucket in us-east-1 region and add reference data.

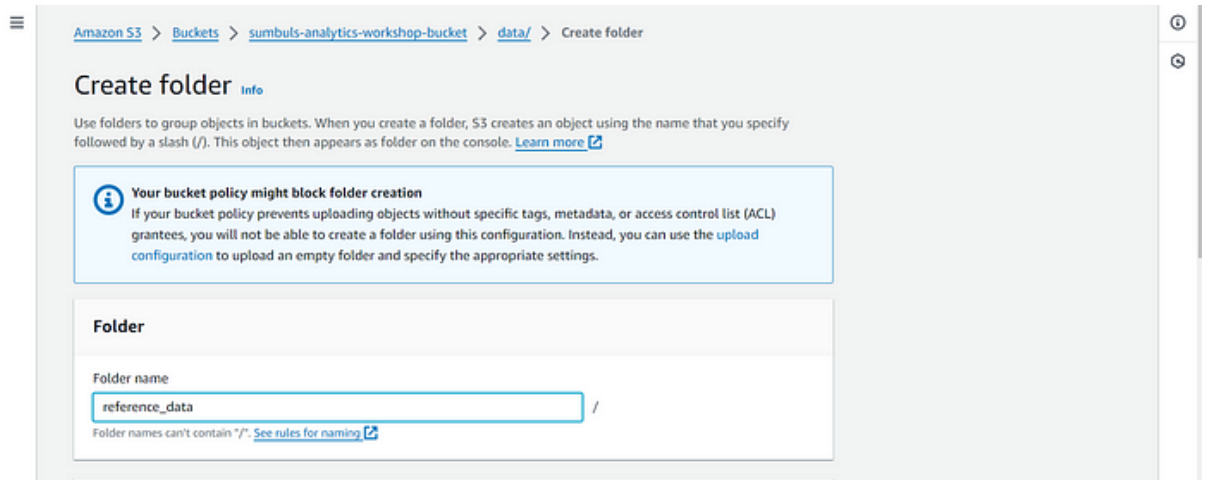
Goto : S3 Console Click me and click — **Create Bucket**
Bucket Name : `yourname-analytics-workshop-bucket`
Region : **US EAST (N. Virginia)**
Optionally add Tags, e.g.: workshop: AnalyticsOnAWS
Click **Create bucket**

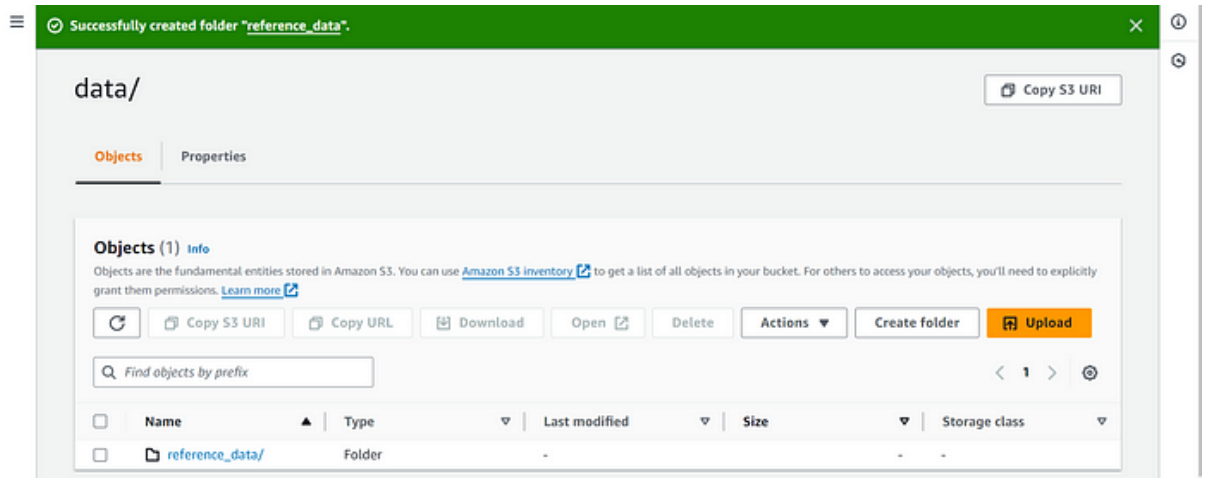


Adding reference data
Open — `yourname-analytics-workshop-bucket`
Click — **Create folder**
New folder : `data`
Click — **Save**

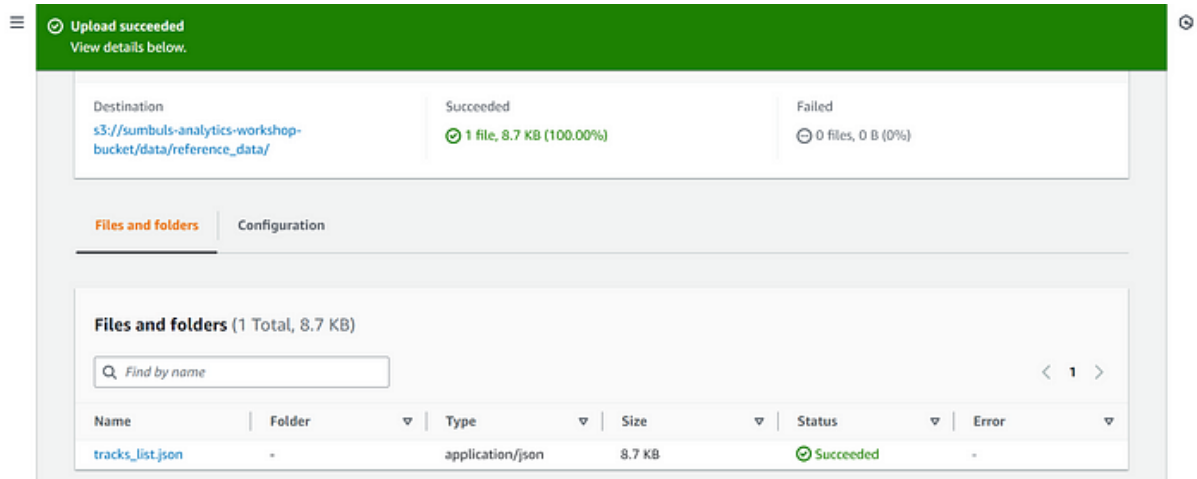


Open — **data**
Click — **Create folder** (From inside the data folder)
New folder : `reference_data`
Click — **Save**





Open — **reference_data**
download this file locally : [tracks_list.json]
In the S3 Console — Click — Upload
Click **Add files** & upload the **tracks_list.json** file here
Click **Upload** (bottom left)

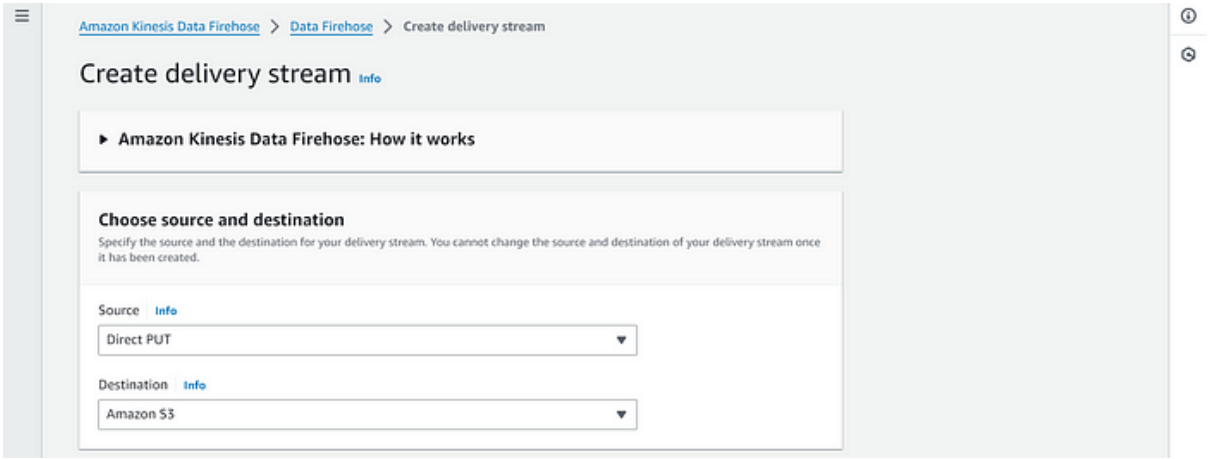


XXXXXXXXXXXXXXXXXXX

b) Create a Kinesis Firehose delivery stream to ingest data & store in S3

Goto: Kinesis Firehose Console Click me. Click **Create delivery stream**

Step 1: Choose source and destination
Source: **Direct PUT**
Destination: **Amazon S3**



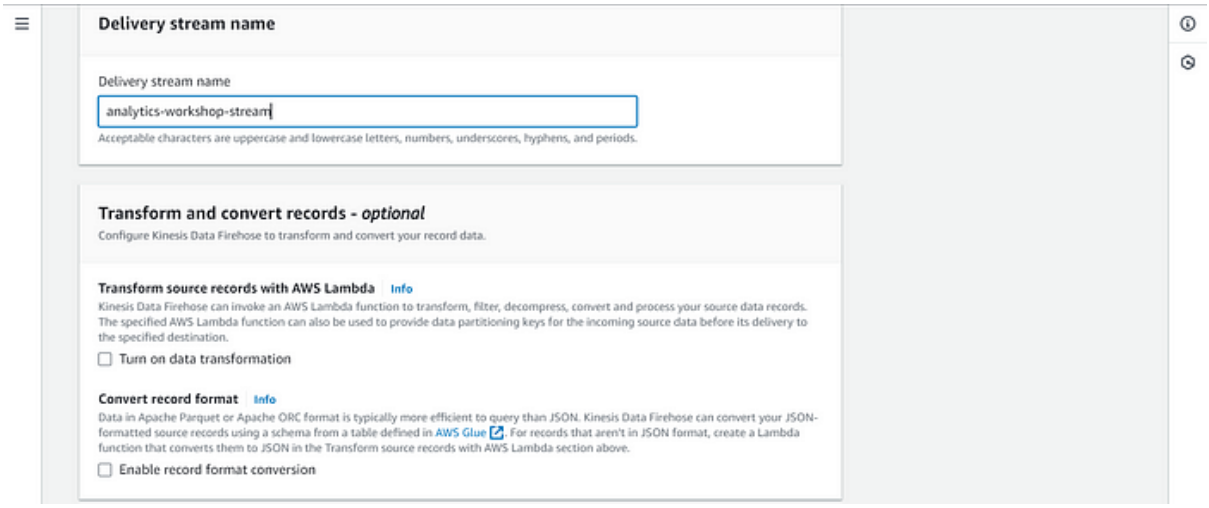
Step 2: Delivery stream name

Delivery stream name: `analytics-workshop-stream`

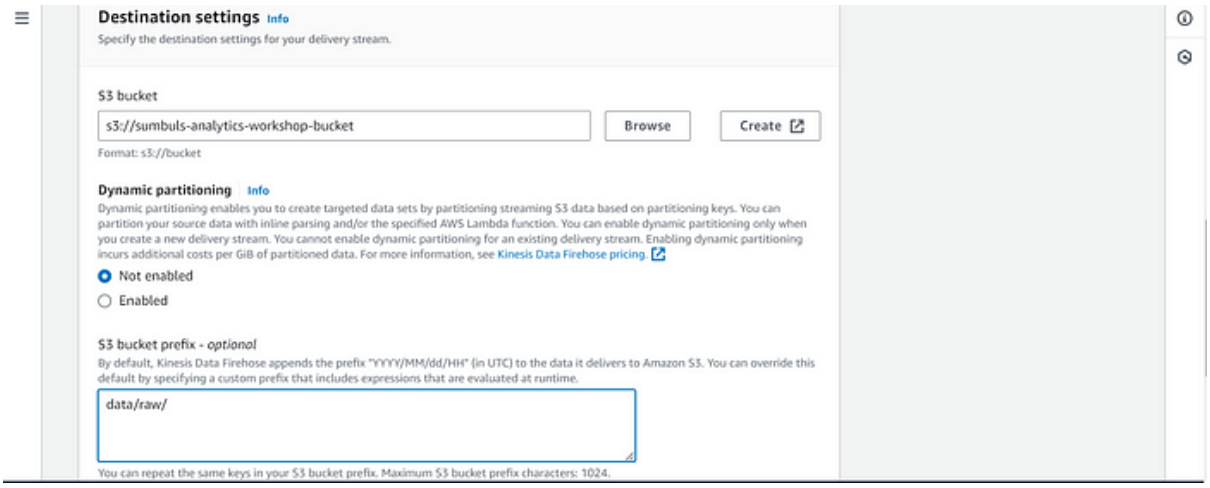
Step 3: Transform and convert records

Transform source records with AWS Lambda: **Disabled**

RecConvert record format: **Disabled**

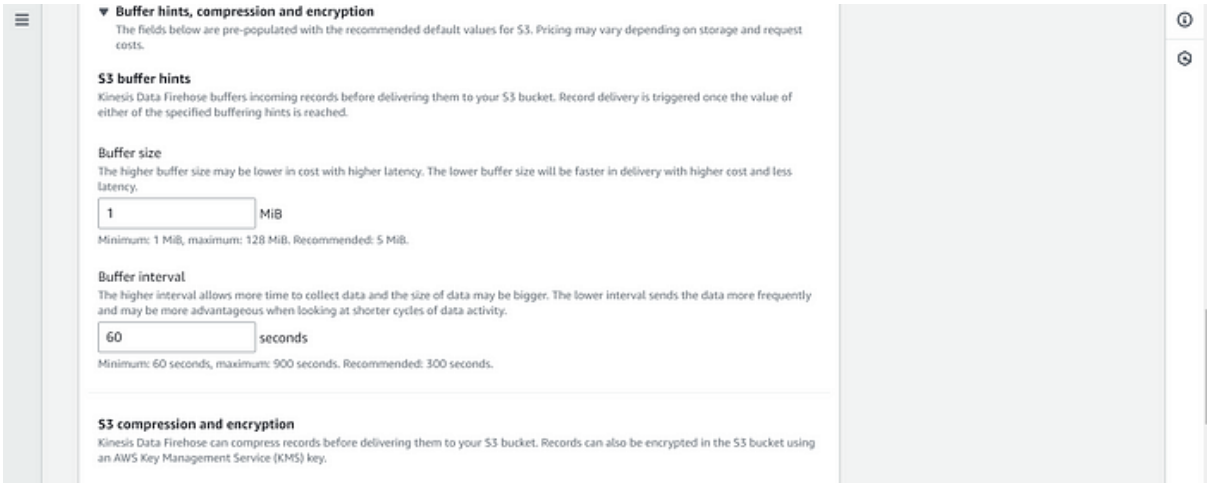


Step 4: Destination settings
S3 bucket: **yourname-analytics-workshop-bucket**
Dynamic partitioning: **Not Enabled**
S3 bucket prefix: `data/raw/`
(**Note:** the slash / after **raw** is important. If you miss it Firehose will copy the data into an undesired location)



S3 bucket error output prefix: *Leave Blank*
Expand **Buffer hints, compression and encryption:**
Buffer size: **1 MiB**
Buffer interval: **60 seconds**

Compression for data records: **Not Enabled**
Encryption for data records: **Not Enabled**



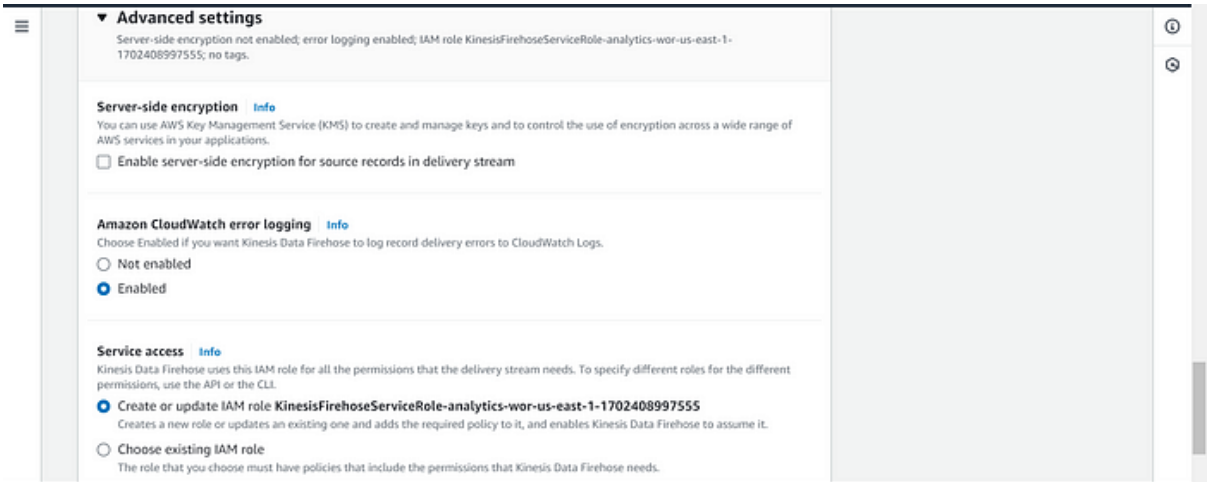
Step 5: Advanced settings

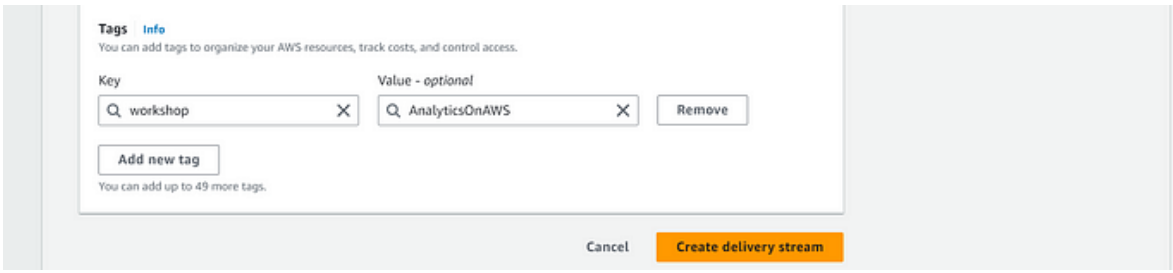
Server-side encryption: **unchecked**

Amazon Cloudwatch error logging: **Enabled**

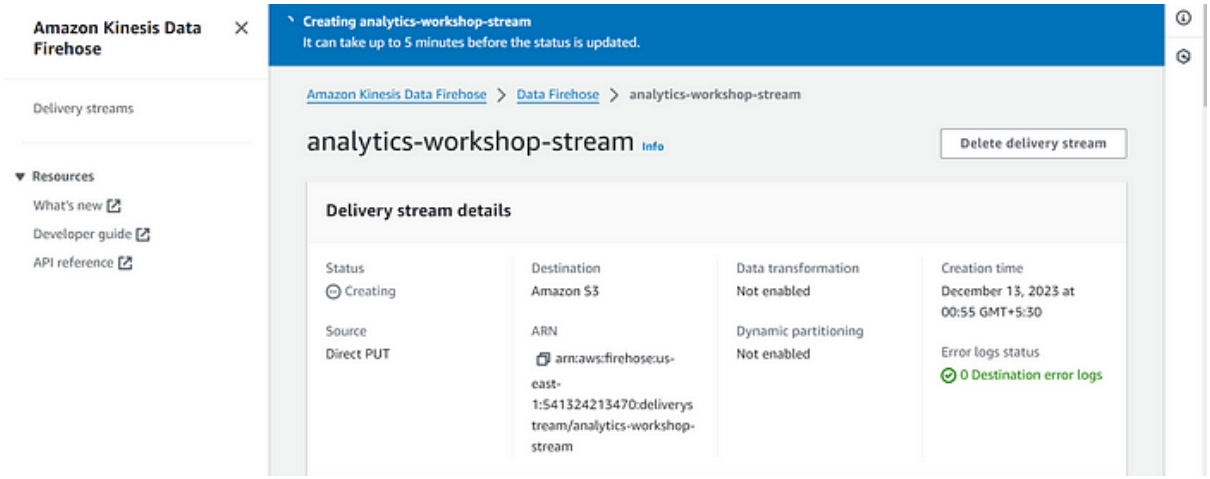
Permissions: **Create or update IAM role KinesisFirehoseServiceRole-xxxx**

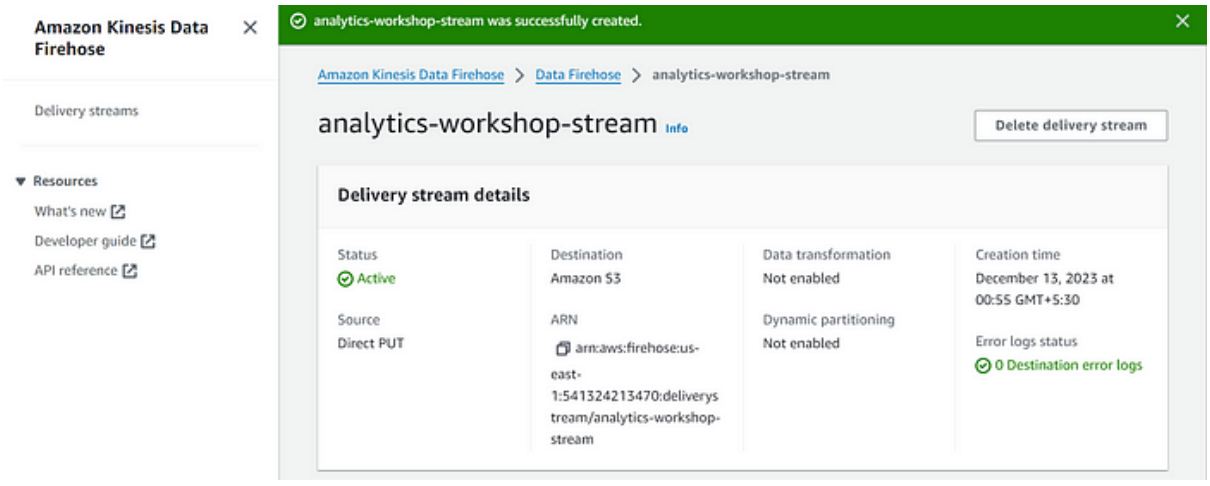
Optionally add Tags, e.g.: Key: `workshop` Value: `AnalyticsOnAWS`





Step 6: Review
Review the configuration & make sure its as mentioned above and click — **Create delivery stream.**



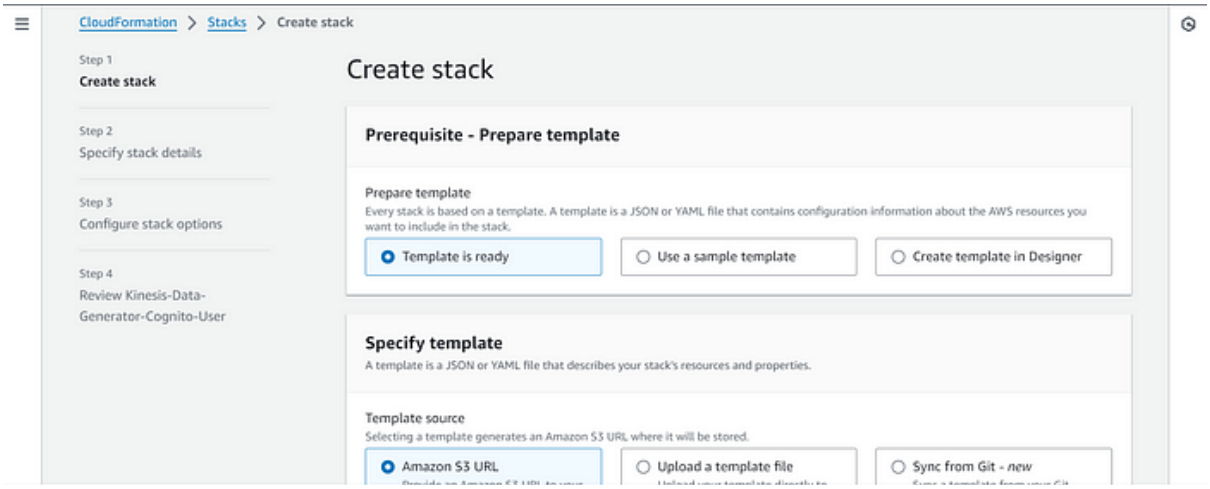


XXXXXXXXXXXXXXXXXXXXX

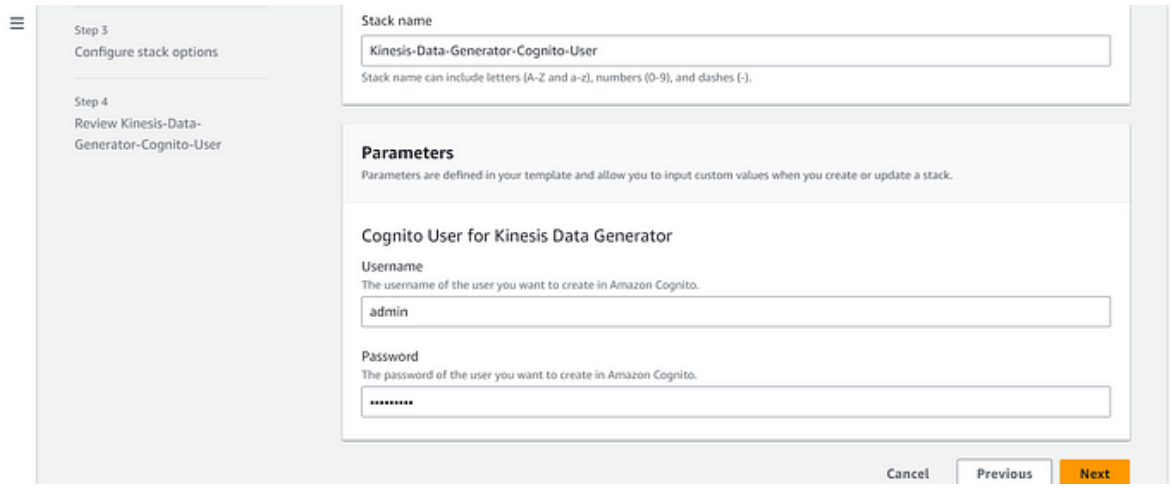
c) Generate Dummy data — configure Kinesis Data Generator to produce fake data and ingest it into Kinesis Firehose

Configure Amazon Cognito for Kinesis Data Generator — In this step we will launch a cloud formation stack that will configure Cognito. This cloudformation scripts launches in **N.Virginia region**

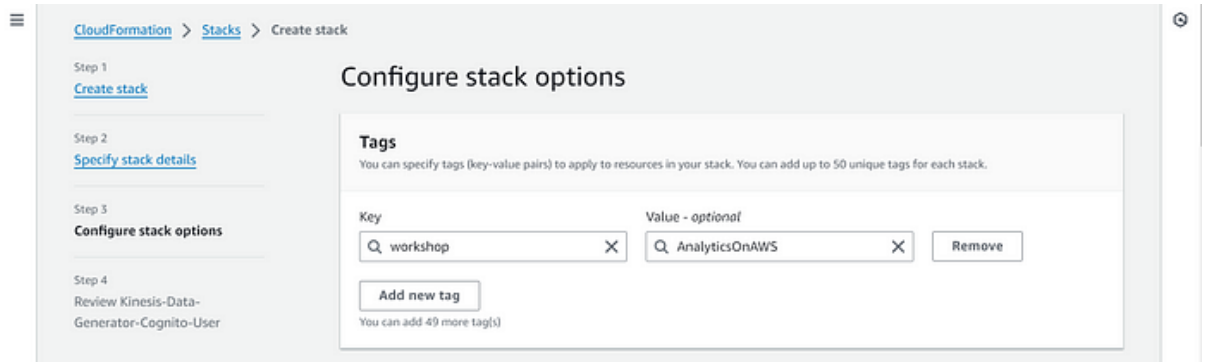
Goto: https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=Kinesis-Data-Generator-Cognito-User&templateURL=https://aws-kdg-tools-us-east-1.s3.amazonaws.com/cognito-setup.json

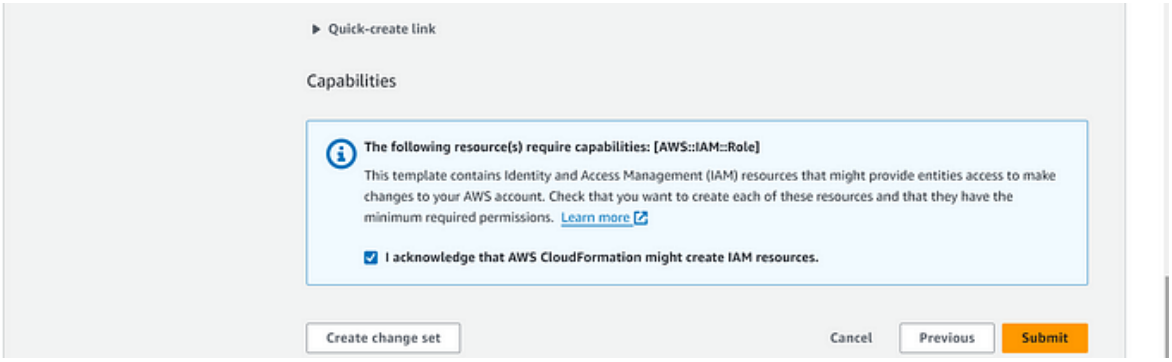


Click **Next.** Specify Details:
Username: **admin**
Password: **choose an alphanumeric password**
Click **Next**

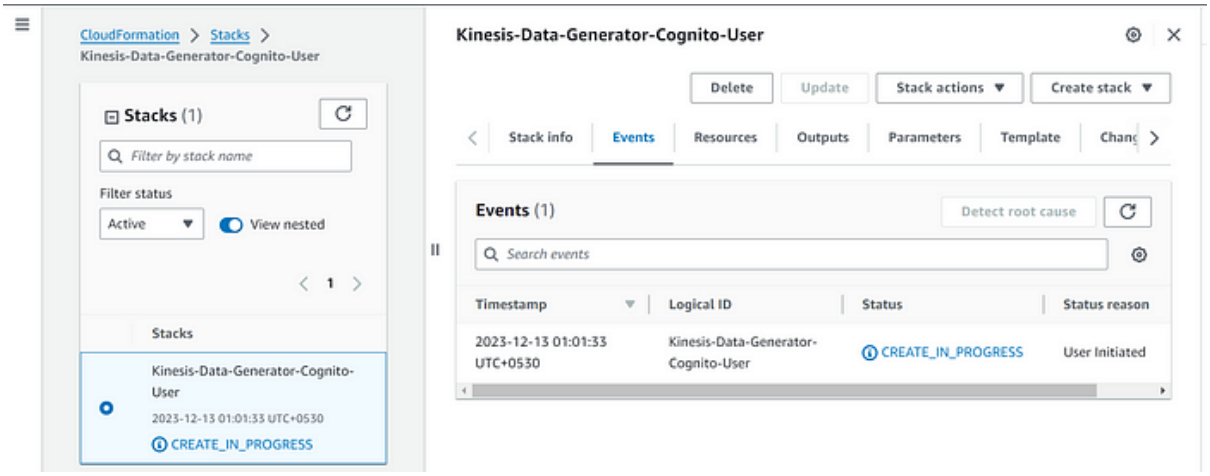


Options:
Optionally add Tags, e.g.: **workshop**: **AnalyticsOnAWS**
Click **Next**

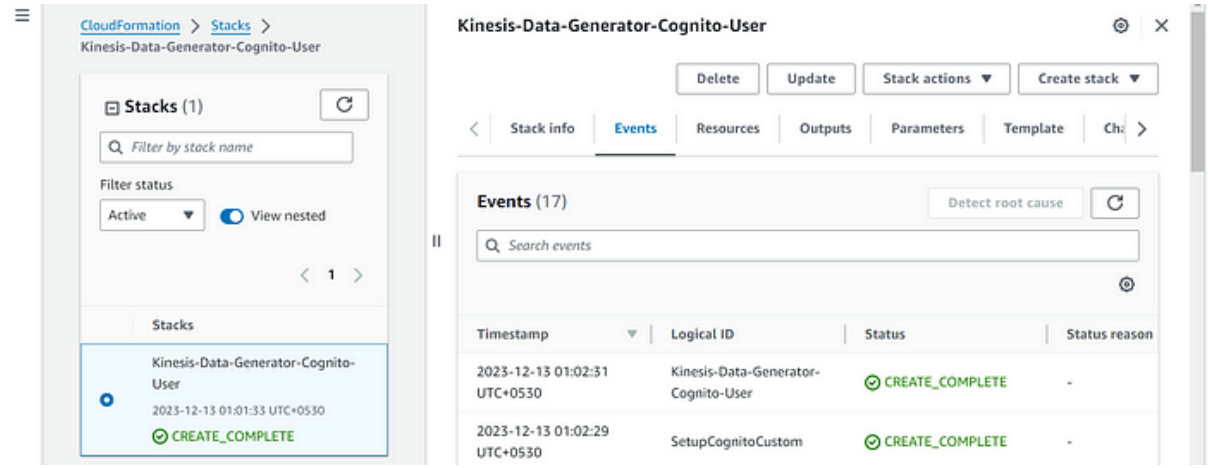




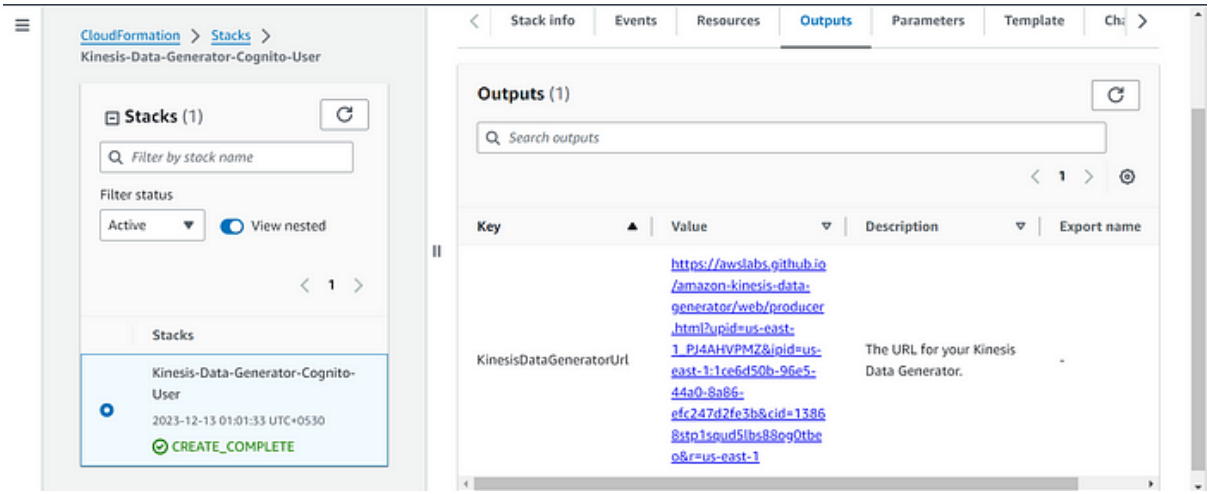
Review:
I acknowledge that AWS CloudFormation might create IAM resources: **Check.** Click **Create stack.**



Wait till the stack status changes to **Create_Complete**

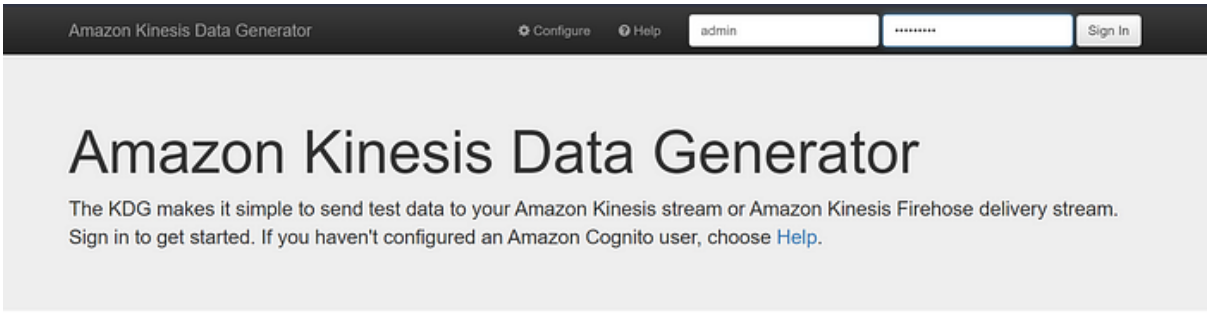


Select the **Kinesis-Data-Generator-Cognito-User** stack

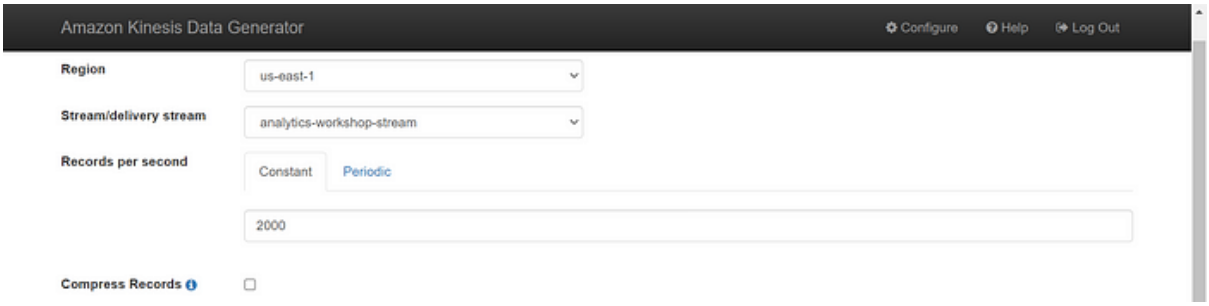


GoTo outputs tab: click on the link that says: **KinesisDataGeneratorUrl** — This will open your Kinesis Data Generator tool.

On Amazon Kinesis Data Generator homepage.
Login with your username & password from previous step.



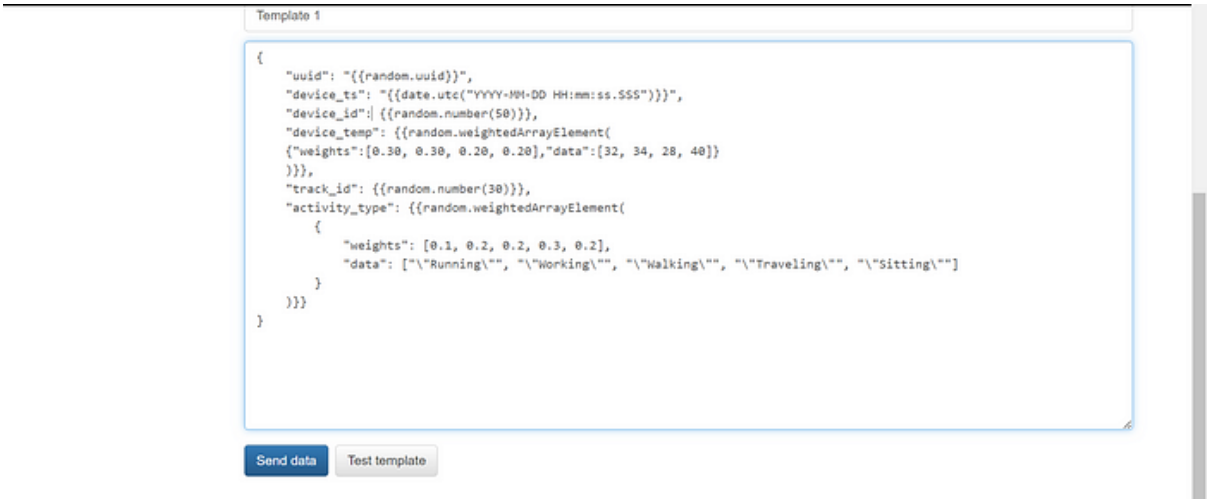
- Stream/delivery stream: **analytics-workshop-stream**
- Records per second: **2000**



- Record template (Template 1): In the **big text area**, insert the following json template:

```
{
    "uuid": "{{random.uuid}}",
    "device_ts": "{{date.utc("YYYY-MM-DD HH:mm:ss.SSS")}}",
    "device_id": {{random.number(50)}},
    "device_temp": {{random.weightedArrayElement(
    {"weights":[0.30, 0.30, 0.20, 0.20],"data":[32, 34, 28, 40]}
    )}},
    "track_id": {{random.number(30)}},
    "activity_type": {{random.weightedArrayElement(
        {
            "weights": [0.1, 0.2, 0.2, 0.3, 0.2],
```

```
            "data": ["\"Running\"", "\"Working\"", "\"Walking\"",
"\"Traveling\"", "\"Sitting\""]
        }
    )}}
}
```



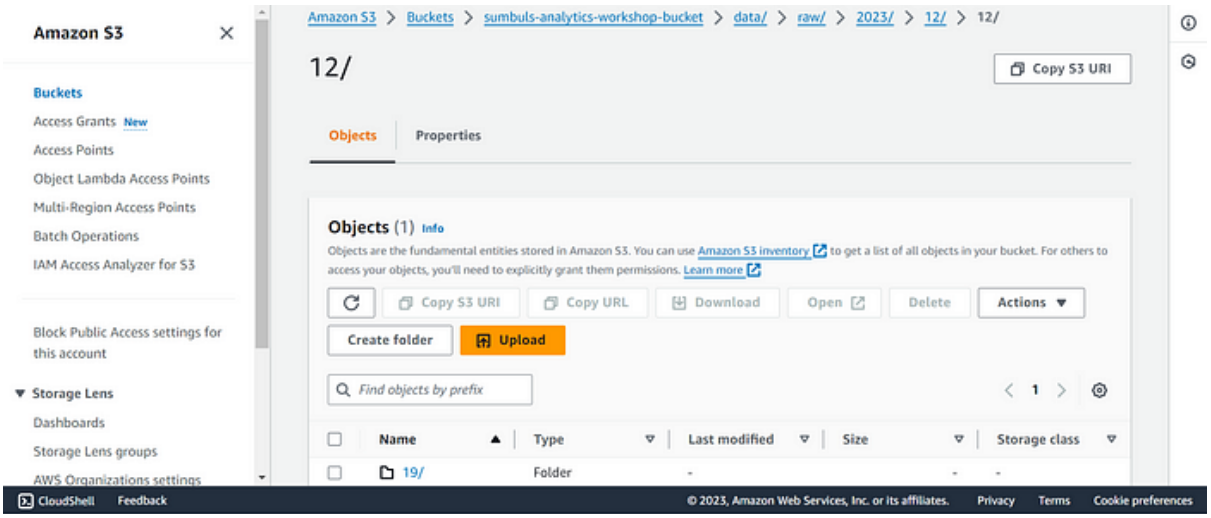
Click — **Send Data.** Once the tool sends ~10,000 messages, you can click on — **Stop sending data to Kinesis**

XXXXXXXXXXXXXXXXXXXX

d) Validate that data has arrived in S3

After few moments go to the S3 console Click me. Navigate to: **yourname-analytics-workshop-bucket > data**

There should be a folder called **raw** > Open it and keep navigating, you will notice that firehose has dumped the data in S3 using **yyyy/mm/dd/hh** partitioning.



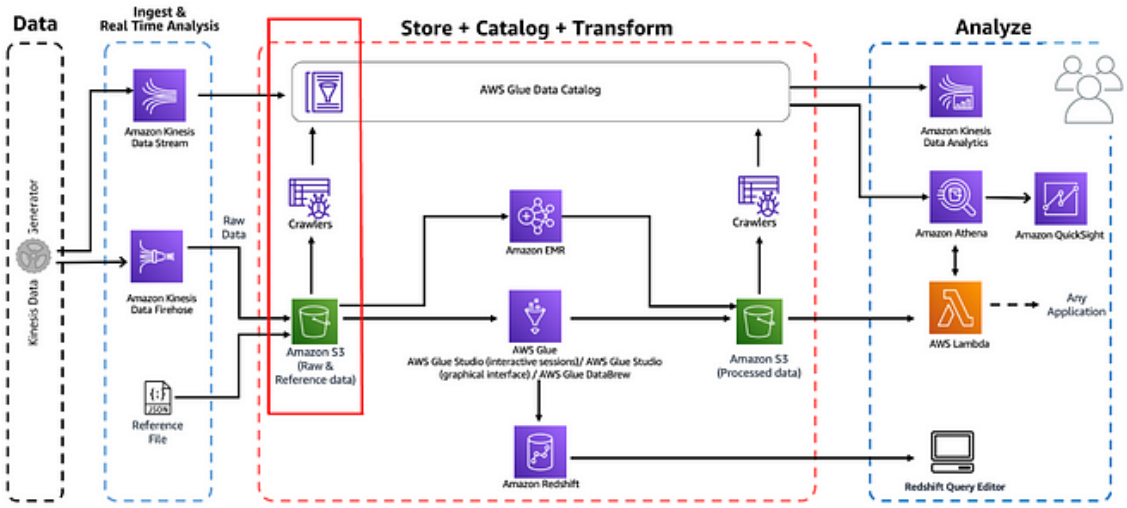
If you have received the dummy data in your S3 buckets, we are good to proceed to next step!

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Task 2 👉: Catalog Data

Here, we are going to register the datasets in the AWS Glue Data Catalog. We will automate the metadata capture with the help of Glue Crawlers.

Once the catalog entities are created, we will able to start querying the *raw* format of the data from Amazon Athena.

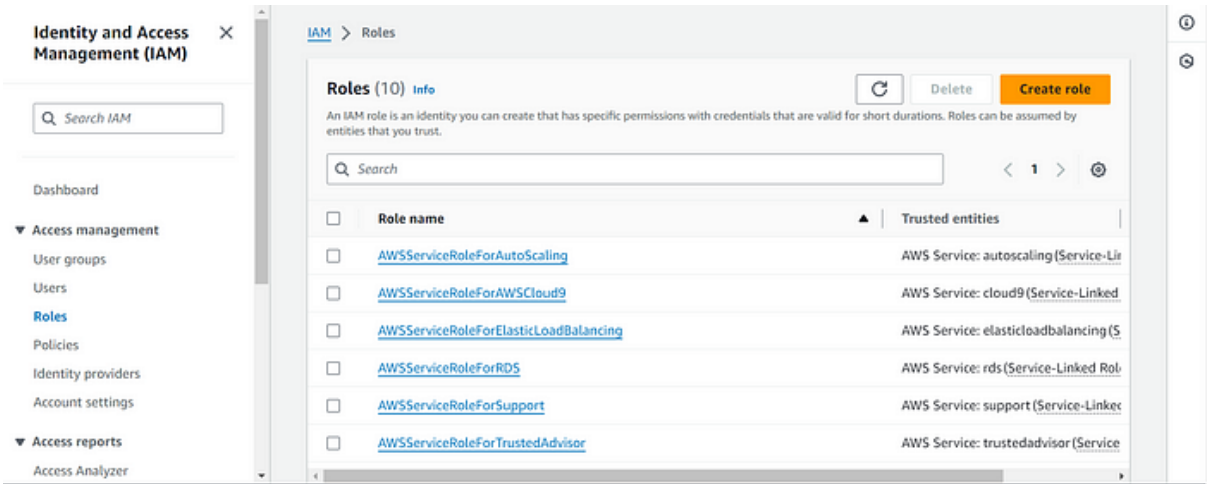


a) Create IAM Role

In this step we will navigate to the IAM Console and create a new AWS Glue service role. This allows AWS Glue to access the data stored in S3 and to create the necessary entities in the Glue Data Catalog. Go to: [Click me]

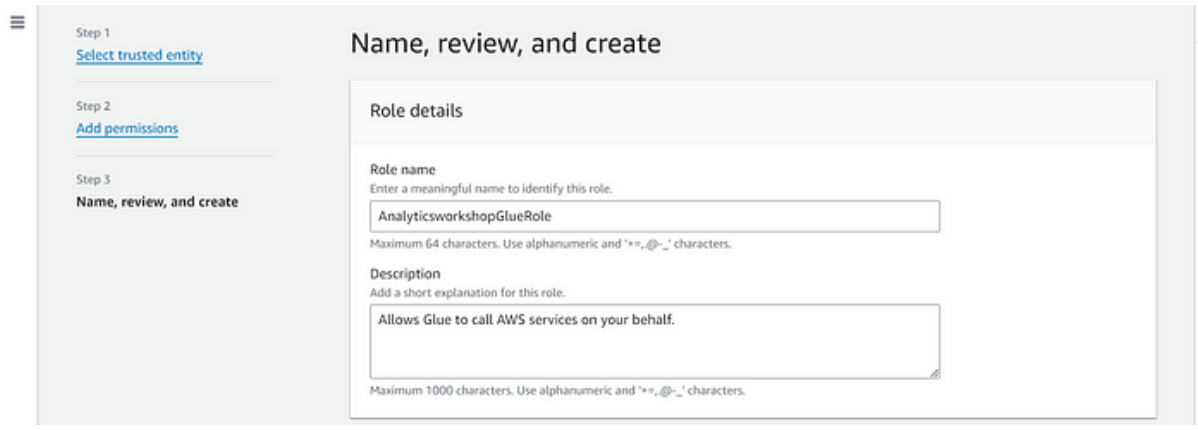
Click **Create role**
Choose the service that will use this role: **Glue and c**lick **Next**

- Search for **AmazonS3FullAccess**
  Select the entry's **checkbox**
- Search for **AWSGlueServiceRole**
  Select the entry's **checkbox**

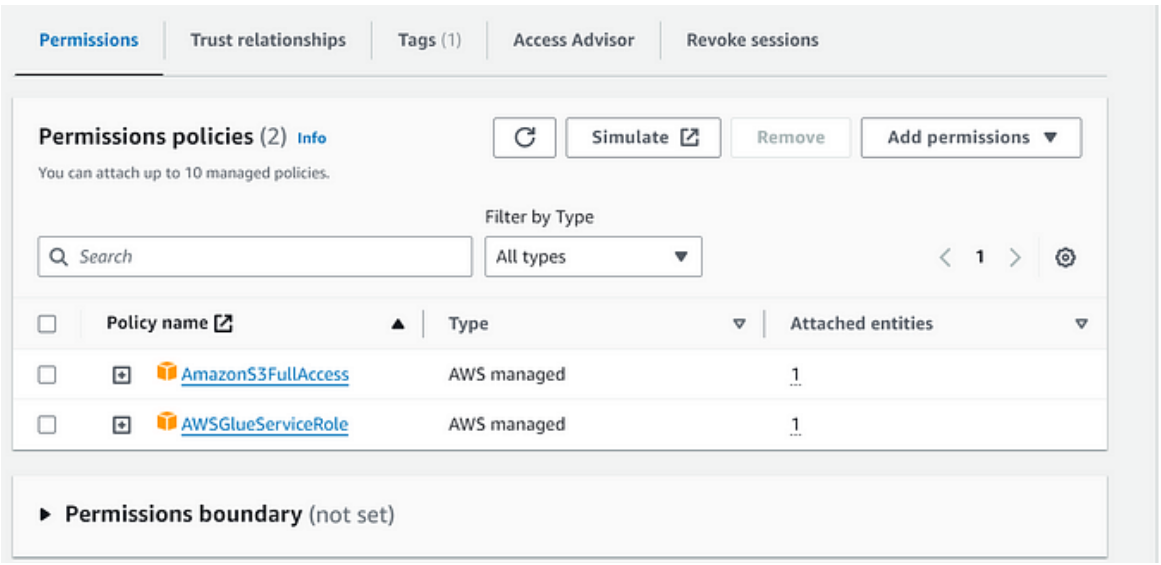


Click **Next**

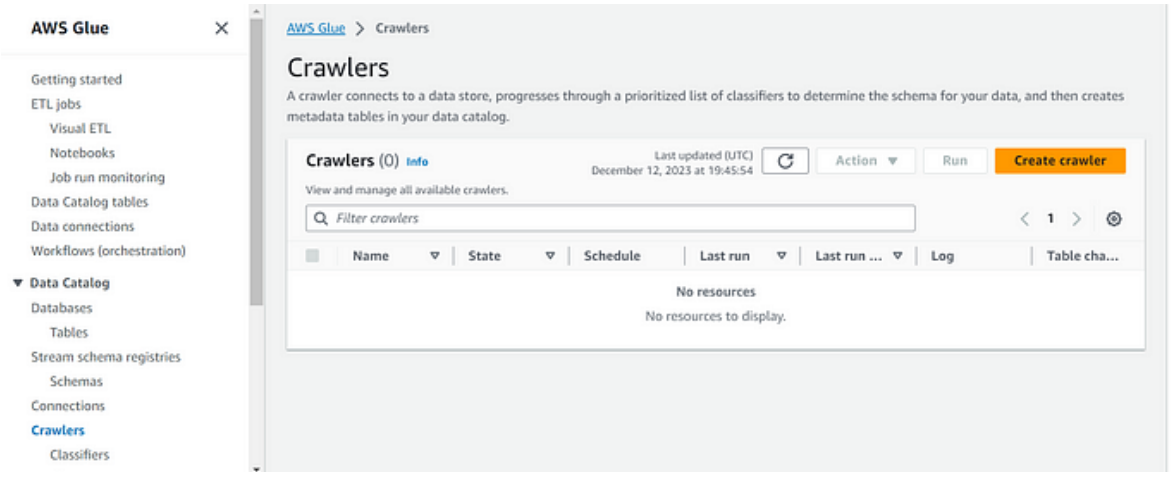
Role name: `AnalyticsworkshopGlueRole`
Make sure that only two policies attached to this role (**AmazonS3FullAccess**, **AWSGlueServiceRole**)
Optionally add Tags, e.g.: **workshop**: **AnalyticsOnAWS**



Click **Create role**

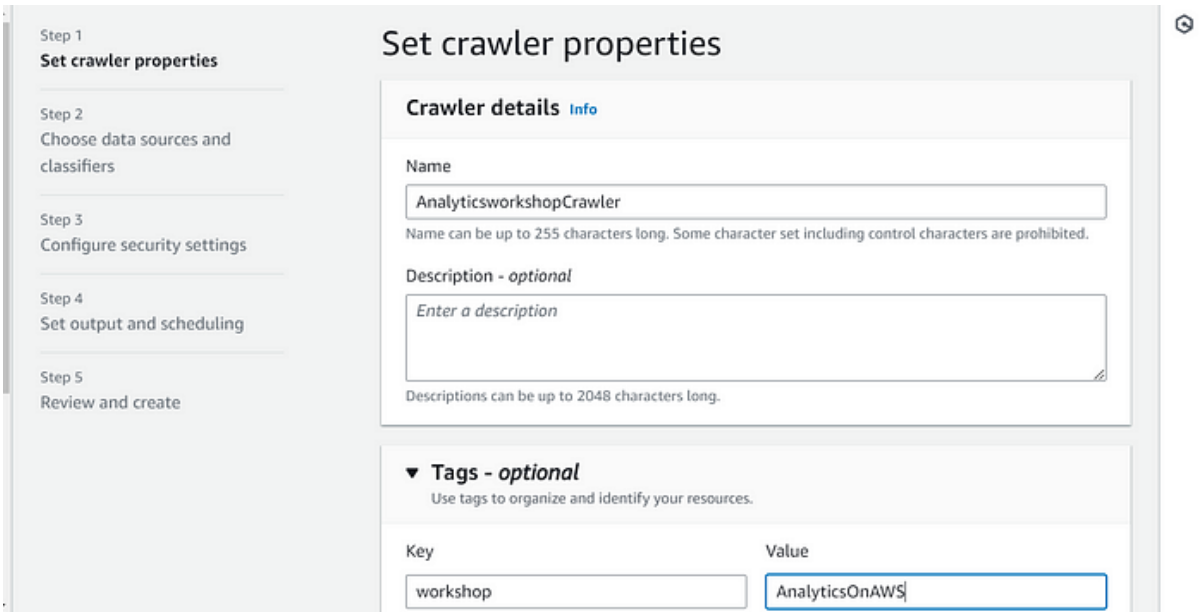


b) Create glue crawlers to discovery the schema of the newly ingested data in S3.
Goto: Click me. On the left panel, click on **Crawlers,** Click on **Create crawler.**



Enter Crawler info and Click **Next**

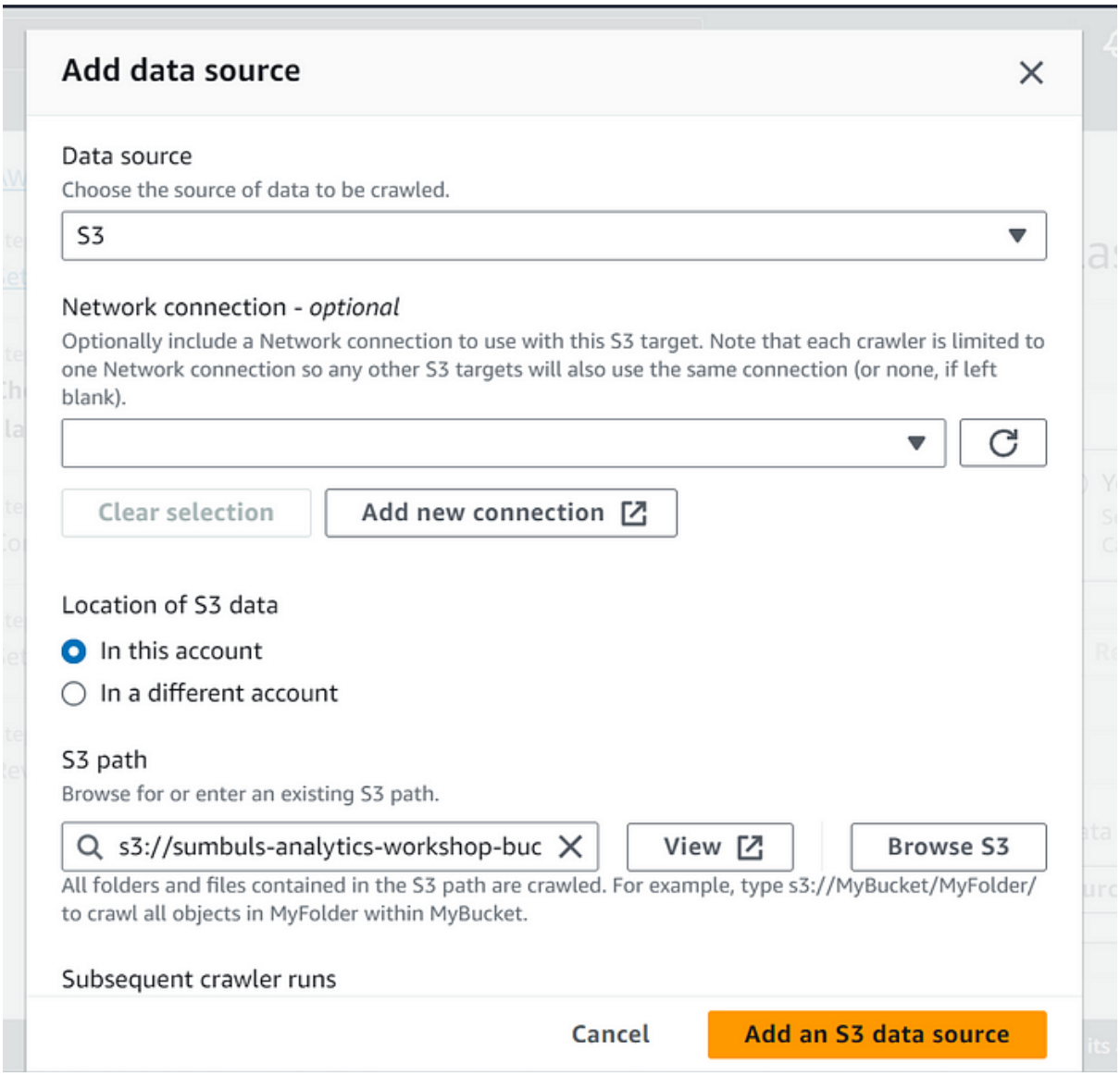
- Crawler name: `AnalyticsworkshopCrawler`
- Optionally add Tags, e.g.: **workshop**: **AnalyticsOnAWS**



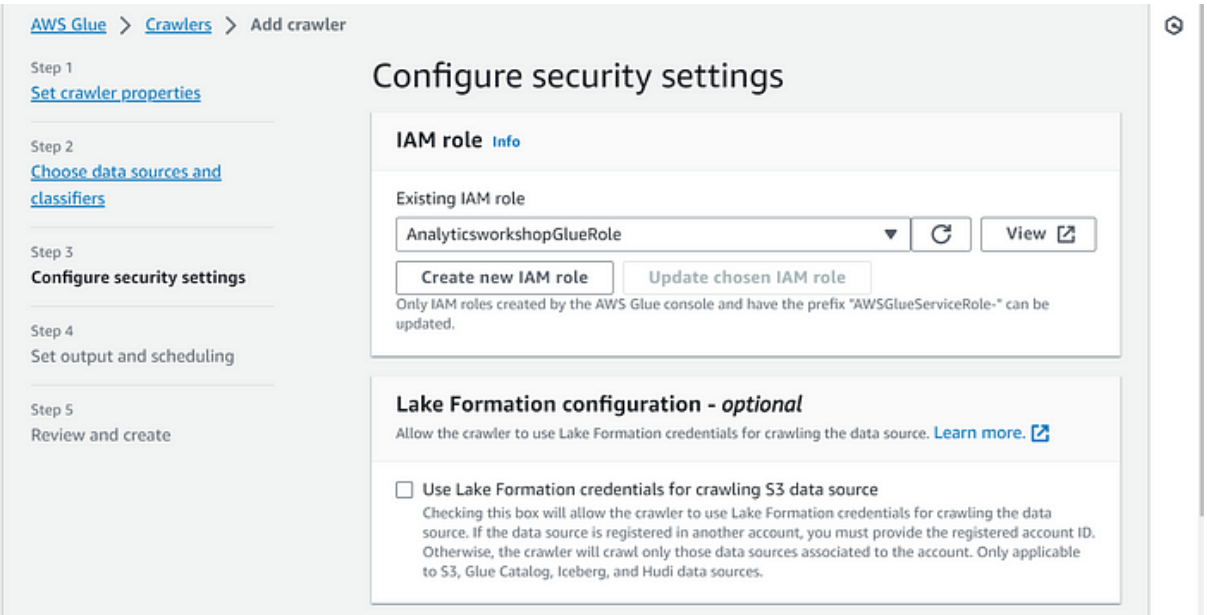
Click **Add a data source**

Choose a **Data source**: **S3**
Leave **Network connection — optional** as-is
Select **In this account** under **Location of S3 data**
Include S3 path: `s3://yourname-analytics-workshop-bucket/data/`

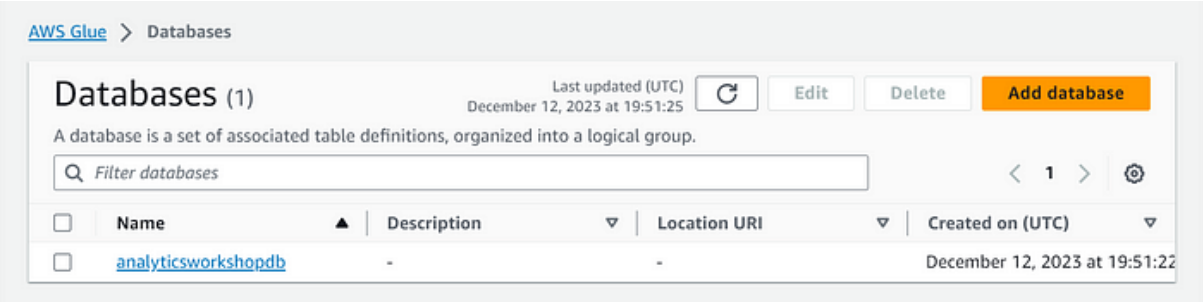
Leave **Subsequent crawler runs** to default selection of **Crawl all sub-folders**
Click **Add an S3 data source**
Select recently added S3 data source under **Data Sources.** Click **Next**



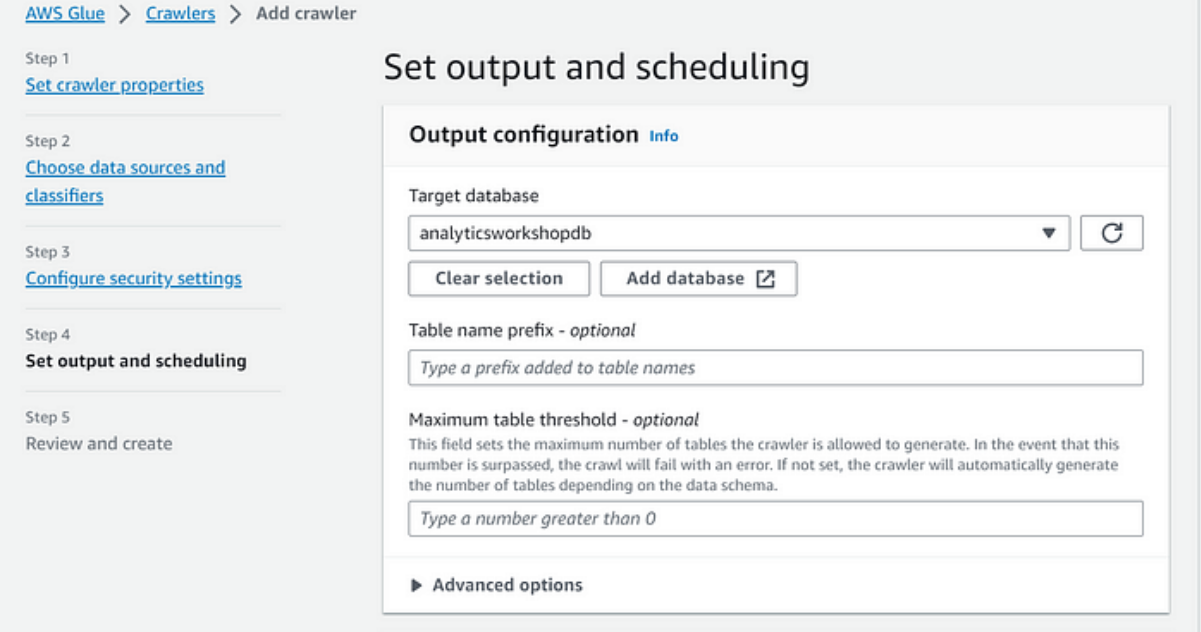
IAM Role. Under **Existing IAM role**, select `AnalyticsworkshopGlueRole`



Output configuration:
Click **Add database** to bring up a new window for creating a database.
Database details Name: `analyticsworkshopdb` Click **Create database**



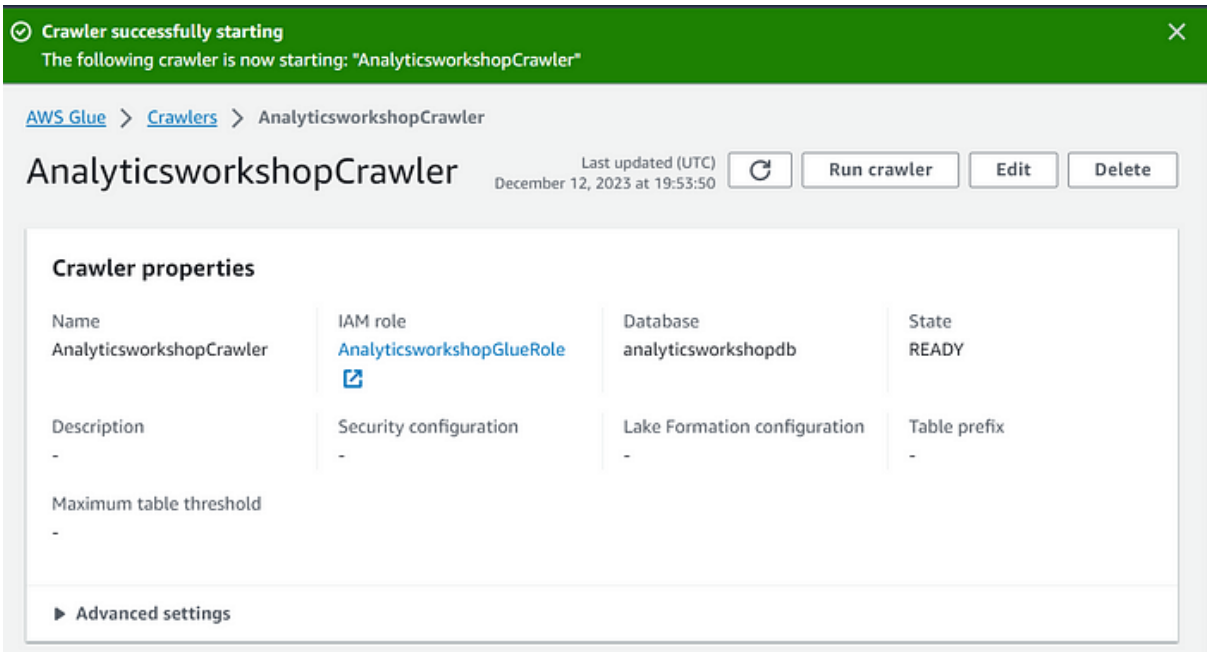
Closes the current window and returns to the previous window.
Refresh by clicking the refresh icon to the right of the **Target database**
Choose `analyticsworkshopdb` under **Target database**



Under **Crawler schedule.** Frequency: **On demand.** Click **Next**

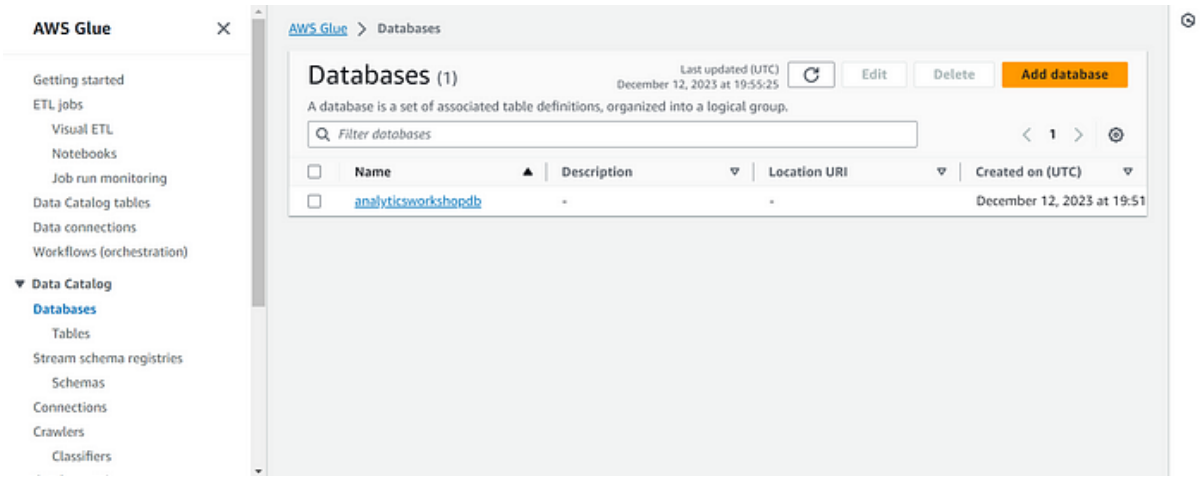
Review all settings under **Review and create.** Click **Create crawler**

You should see this message: **The following crawler is now created: “AnalyticsworkshopCrawler”.** Click **Run crawler** to run the crawler for the first time. Wait for few minutes

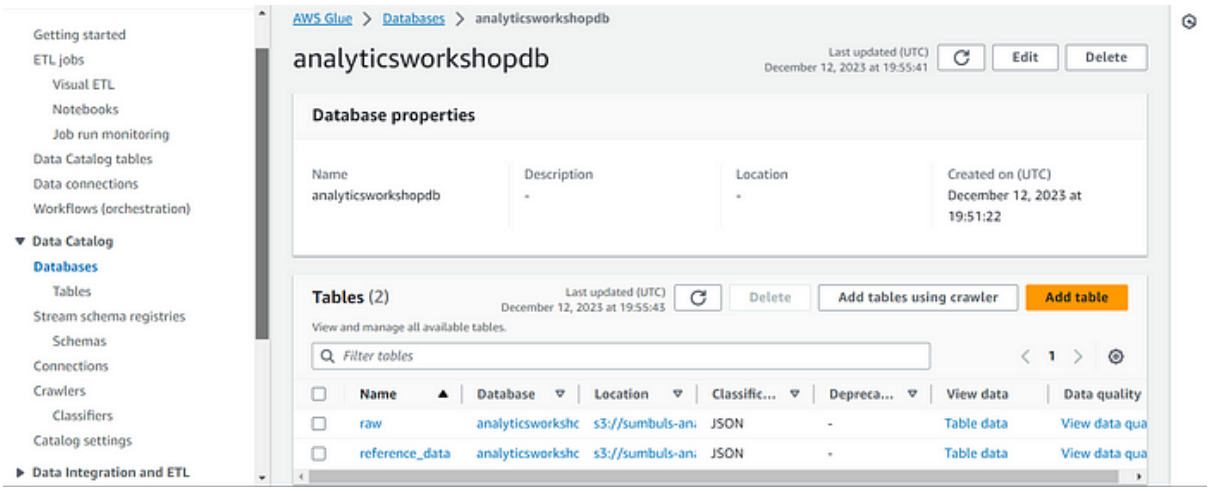


c) Verify newly created tables in catalog

Navigate to Glue Catalog Click me and explore the crawled data:
Click **analyticsworkshopdb**
Click **Tables in analyticsworkshopdb**
Click **raw**



Look around and explore the **schema** for your dataset
look for the `averageRecordSize`, `recordCount`, `compressionType`



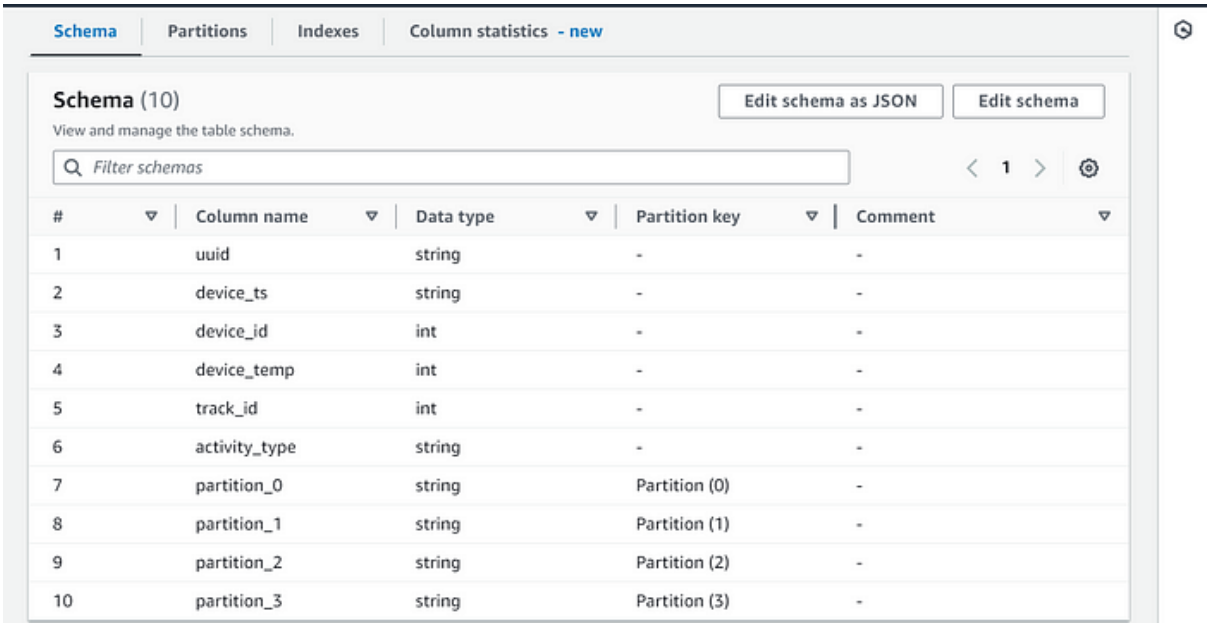


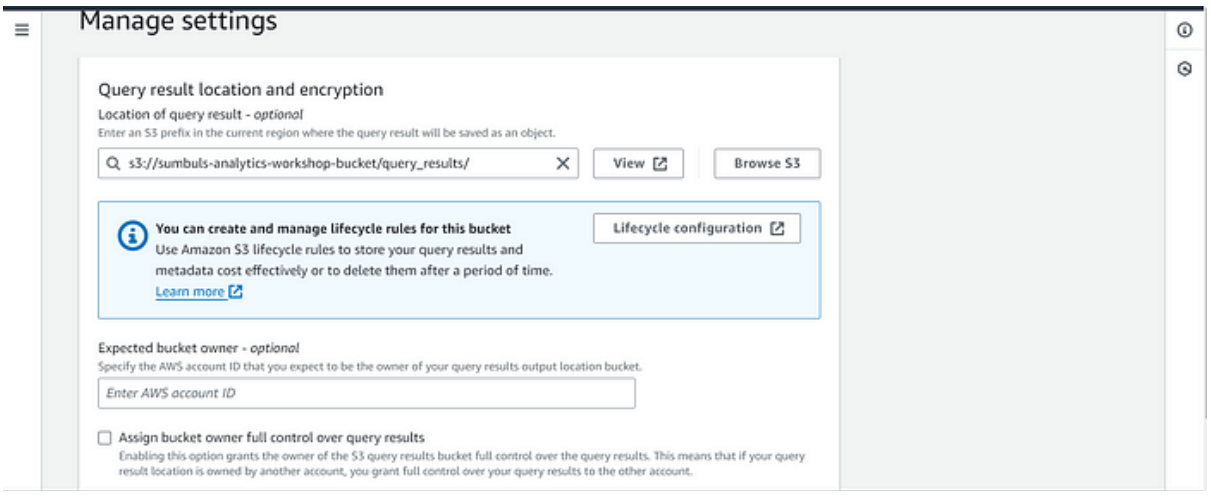
Table properties (11)

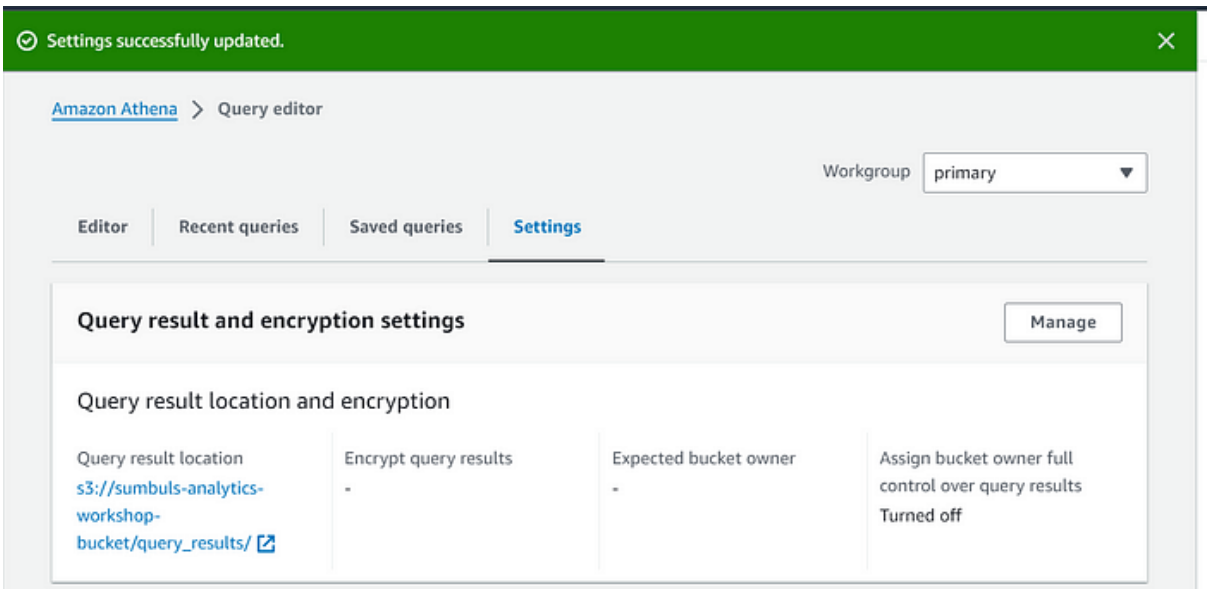
Key	Value
sizeKey	17487551
objectCount	18
UPDATED_BY_CRAWLER	AnalyticsworkshopCrawler
CrawlerSchemaSerializerVersion	1.0
recordCount	18000
averageRecordSize	967
partition_filtering.enabled	true
CrawlerSchemaDeserializerVersion	1.0
compressionType	none
classification	json
typeOfData	file

d) Query ingested data using Amazon Athena

Let's query the newly ingested data using Amazon Athena.
Goto: [Click me]

If necessary, click **Edit seetings** in the blue alert near the top of the Athena console.
Location of query result Under **Query result location and encryption**: `s3://yourname-analytics-workshop-bucket/query_results/`Click **Save**

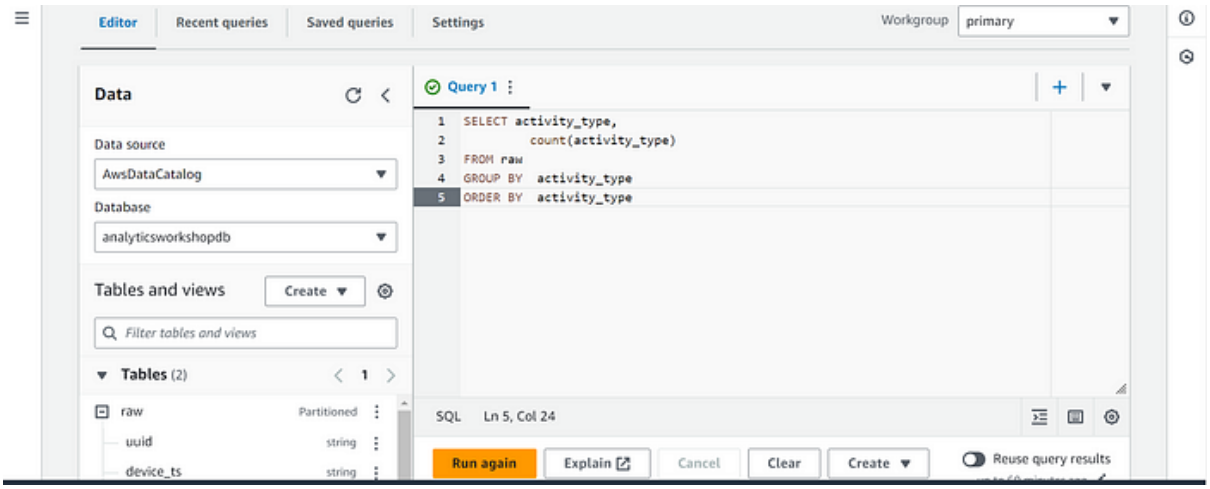


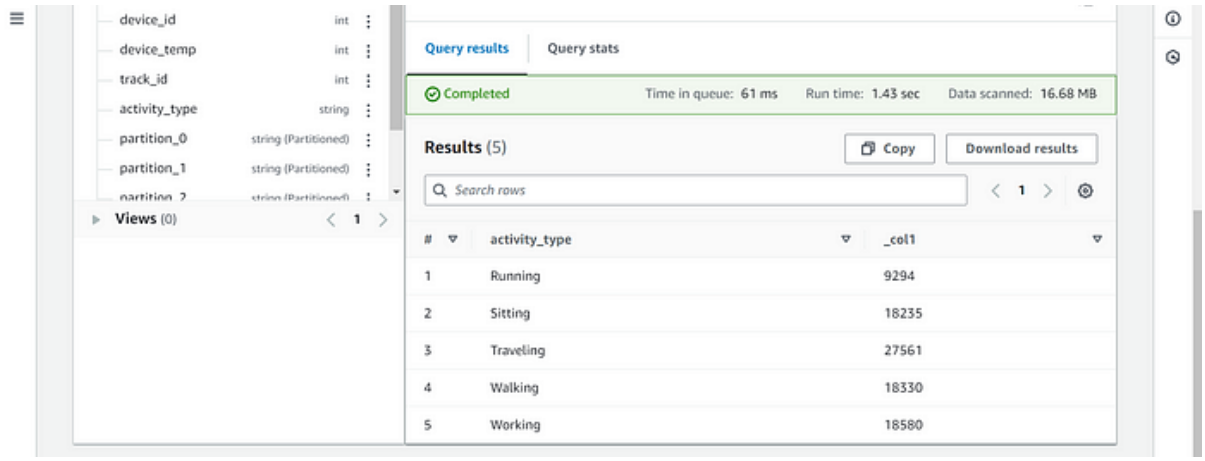


Click **Editor** tab
On the left panel (**Database**) drop down ,
select **analyticsworkshopdb** > select table **raw**
Click on **3 dots** (3 vertical dots) > Select **Preview Table.** Review the output.

In query editor, paste the following below query, and click **Run**.

```
SELECT activity_type,
         count(activity_type)
FROM raw
GROUP BY  activity_type
ORDER BY  activity_type
```





Now that we have cataloged the data, lets proceed to the next step of transforming the data using AWS Glue ETL!

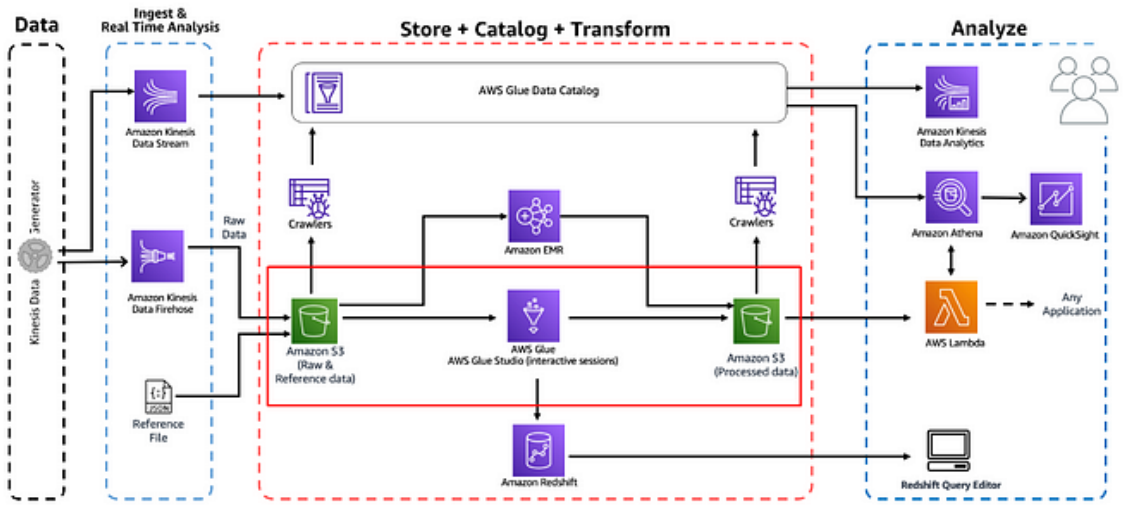
XXXXXXXXXXXXXXXXXXX

Task 3 👉: Data transformation

Transform Data with AWS Glue Studio (interactive sessions)

a) Prepare IAM Policies and Rules.
b) Use Jupyter Notebook in AWS Glue for interactive ETL development.

In this module, we are going to use AWS Glue interactive sessions to process the data and store back the results into a *transformed layer* back in S3. We will use Glue Studio and Jupyter notebooks powered by AWS Glue Interactive Sessions to work through the data transformation steps.



What is AWS Glue interactive sessions?

Interactive sessions allows you to interactively develop AWS Glue processes, run and test each step, and view the results. If you prefer a code-based experience and want to interactively author data integration jobs, interactive sessions is recommended.

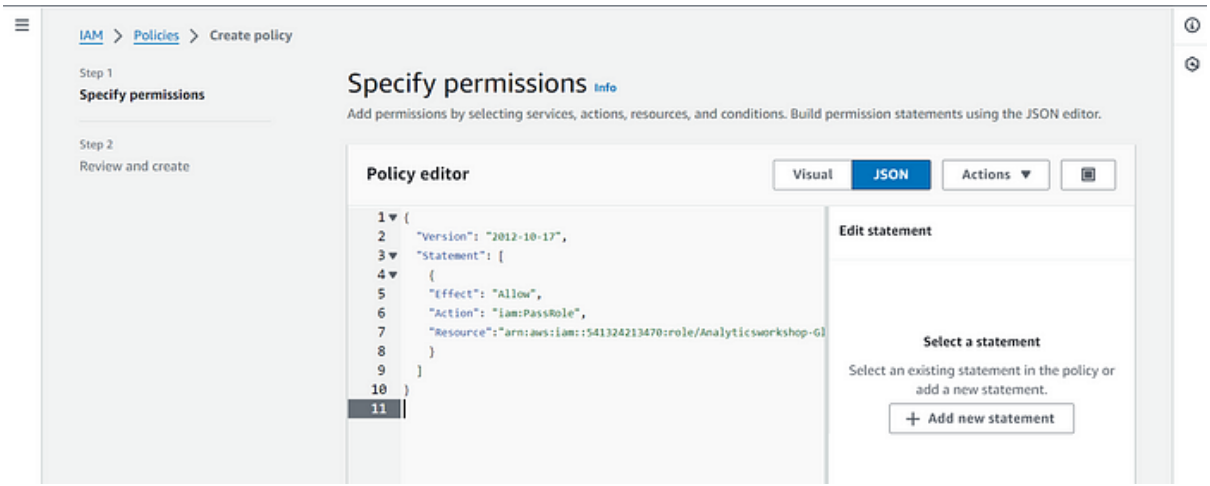
a) Prepare IAM Policies and Rules.
Navigate to IAM console and create the necessary IAM policies and role to work with **AWS Glue Studio Jupyter notebooks** and interactive sessions.

Let's start by creating an **IAM policy** for the AWS Glue notebook role.
Go to: Click me. Click **Policies** from menu panel on the left.
Click **Create policy.** Click on **JSON** tab.

Replace default text in policy editor window with the following policy statemenent.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource":"arn:aws:iam::<AWS account ID>:role/Analyticsworkshop-GlueISRole"
    }
  ]
}
```

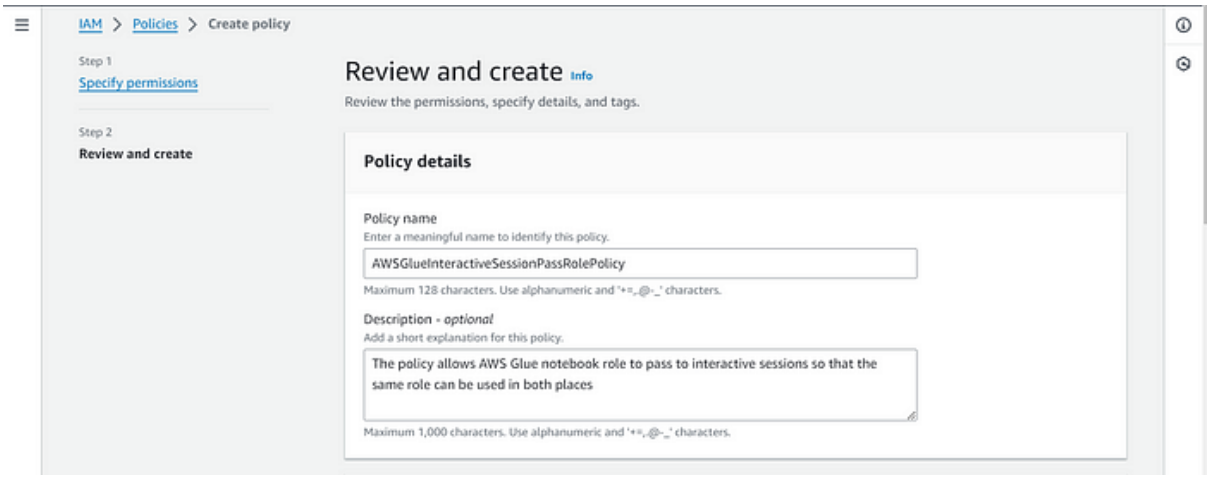
Note: `Analyticsworkshop-GlueISRole` is the role that we create for the **AWS Glue Studio Jupyter notebook** in next step.
Alert: Replace with your **AWS account ID** in the copied policy statement.

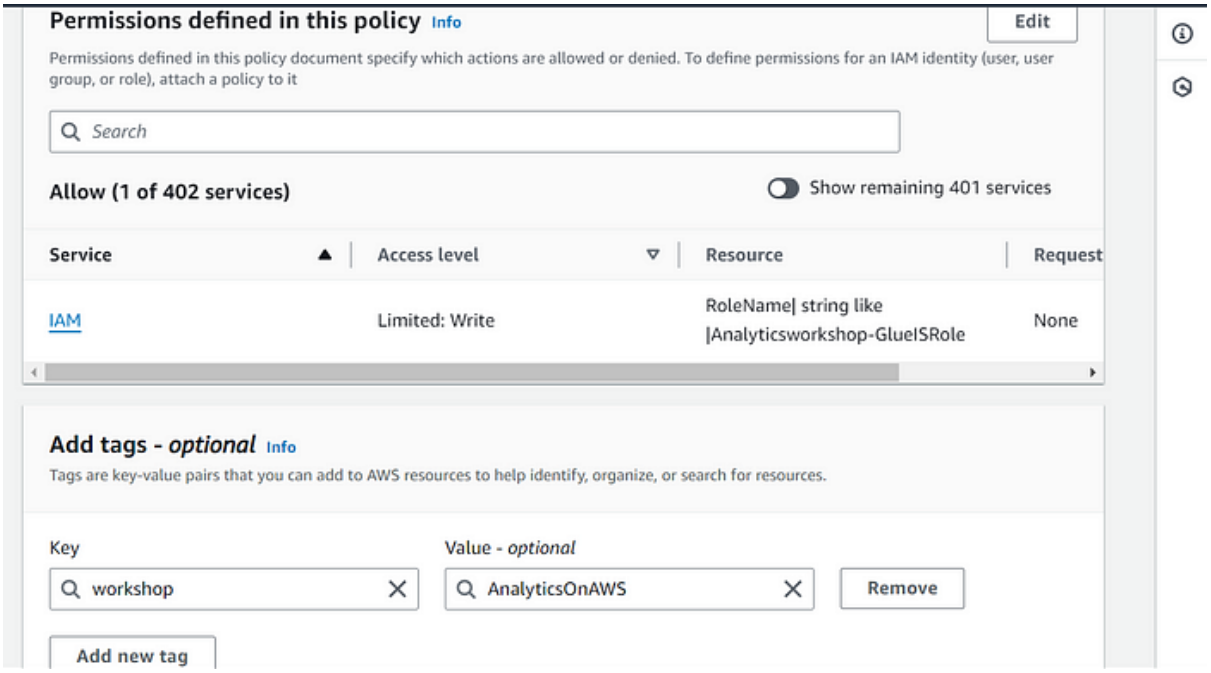


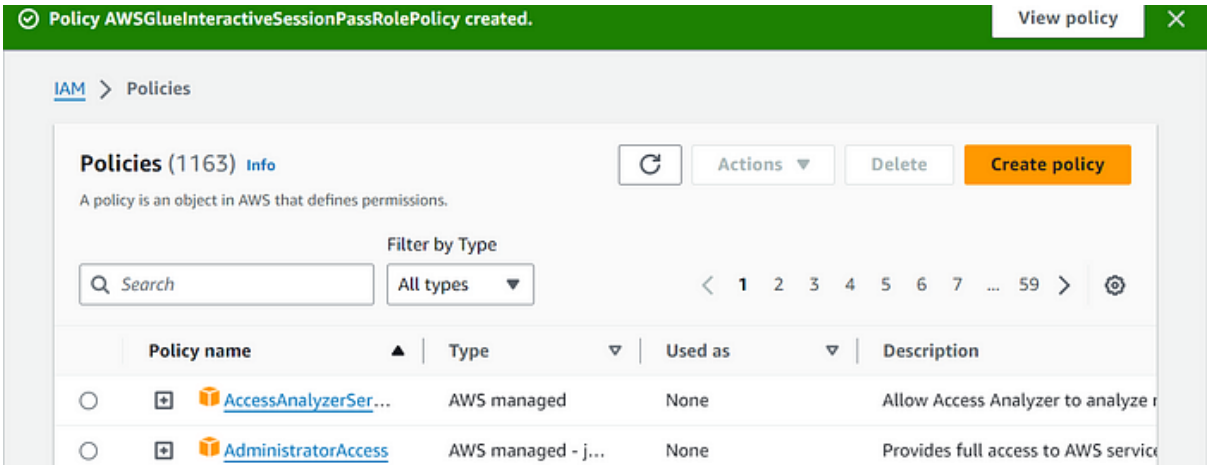
Click **Next: Tags** add Tags, e.g.: **workshop**: **AnalyticsOnAWS**
Click **Next: Review**

- Policy Name: `AWSGlueInteractiveSessionPassRolePolicy`

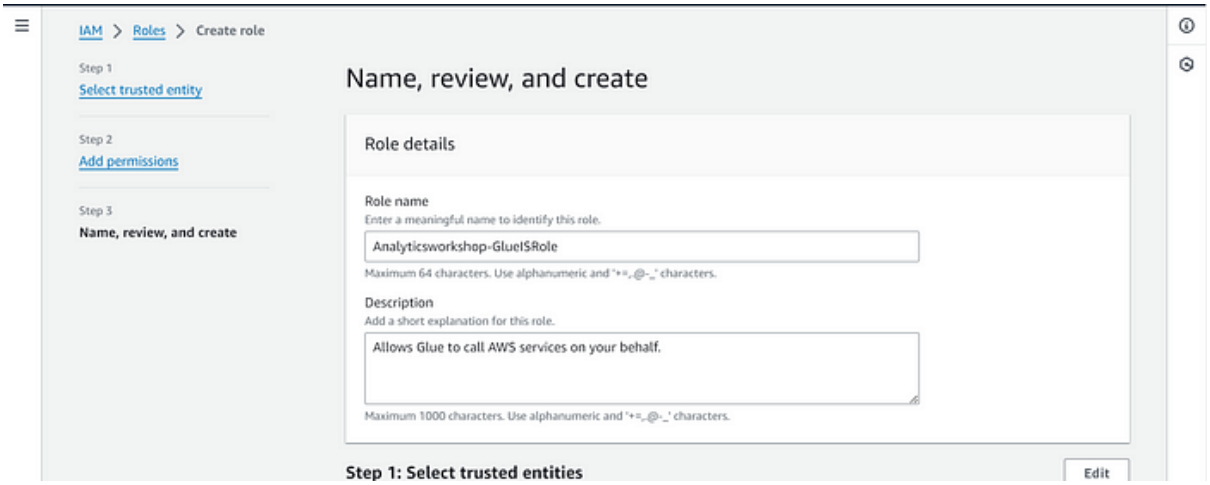
- Optionally write description for the policy: `The policy allows AWS Glue notebook role to pass to interactive sessions so that the same role can be used in both places`
- Click **Create policy**







Next, create an **IAM role for AWS Glue notebook.** Goto: [Click me]

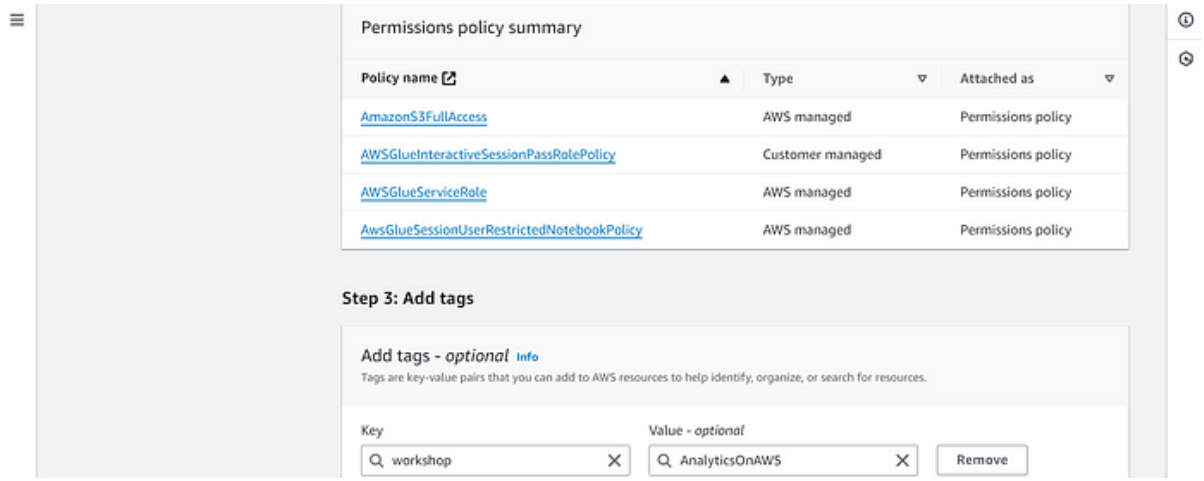


Click **Roles** from menu panel on the left. Click **Create role.** Choose the service that will use this role: **Glue** under **Use Case** and **Use cases for other AWS services** and Click **Next:**

Search for following policies and select the checkbox against them, then Click **Next**:

- `AWSGlueServiceRole`
- `AwsGlueSessionUserRestrictedNotebookPolicy`

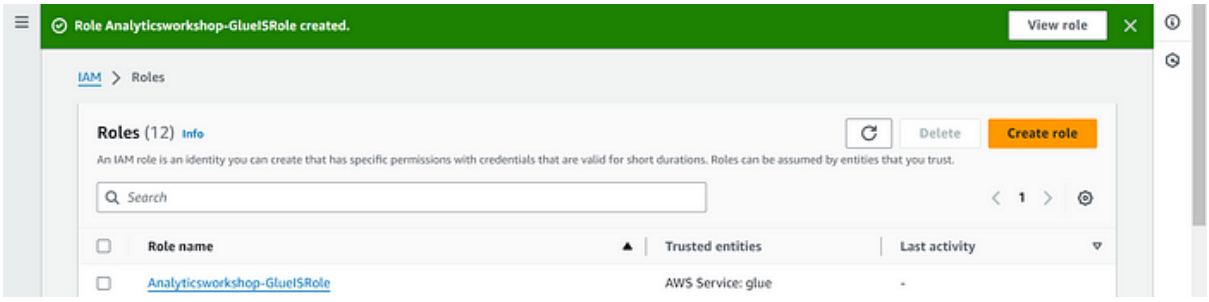
- `AWSGlueInteractiveSessionPassRolePolicy`
- `AmazonS3FullAccess`



Role name: `Analyticsworkshop-GlueISRole`

Make sure only four policies are attached to this role (**AWSGlueServiceRole**, **AwsGlueSessionUserRestrictedNotebookPolicy**, **AWSGlueInteractiveSessionPassRolePolicy**, **AmazonS3FullAccess**)

Optionally add Tags, e.g.: **workshop**: **AnalyticsOnAWS**
Click **Create role**



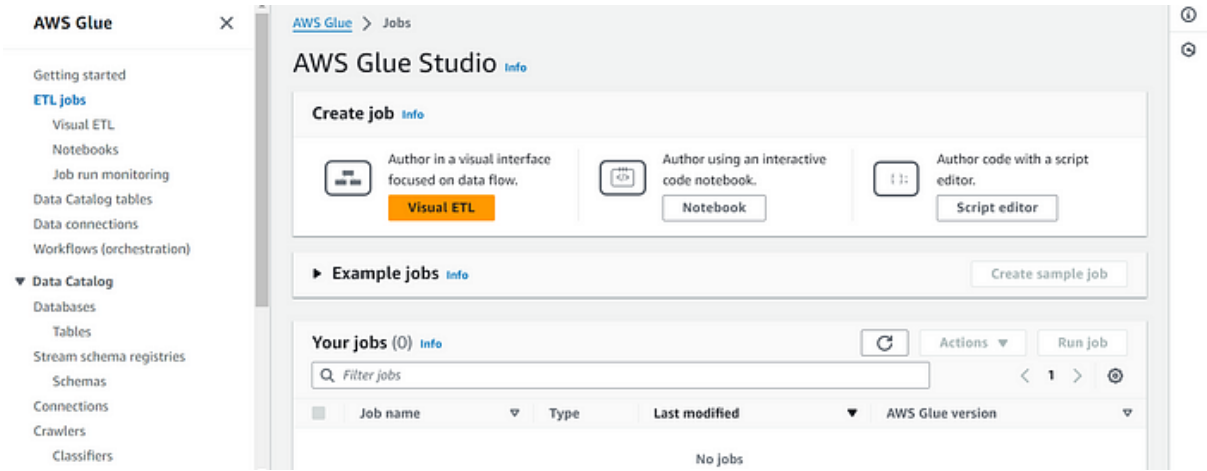
Note: For on getting started with notebooks in AWS Glue Studio, refer to [Getting started with notebooks in AWS Glue Studio](#) .

b) Use Jupyter Notebook in AWS Glue for interactive ETL development

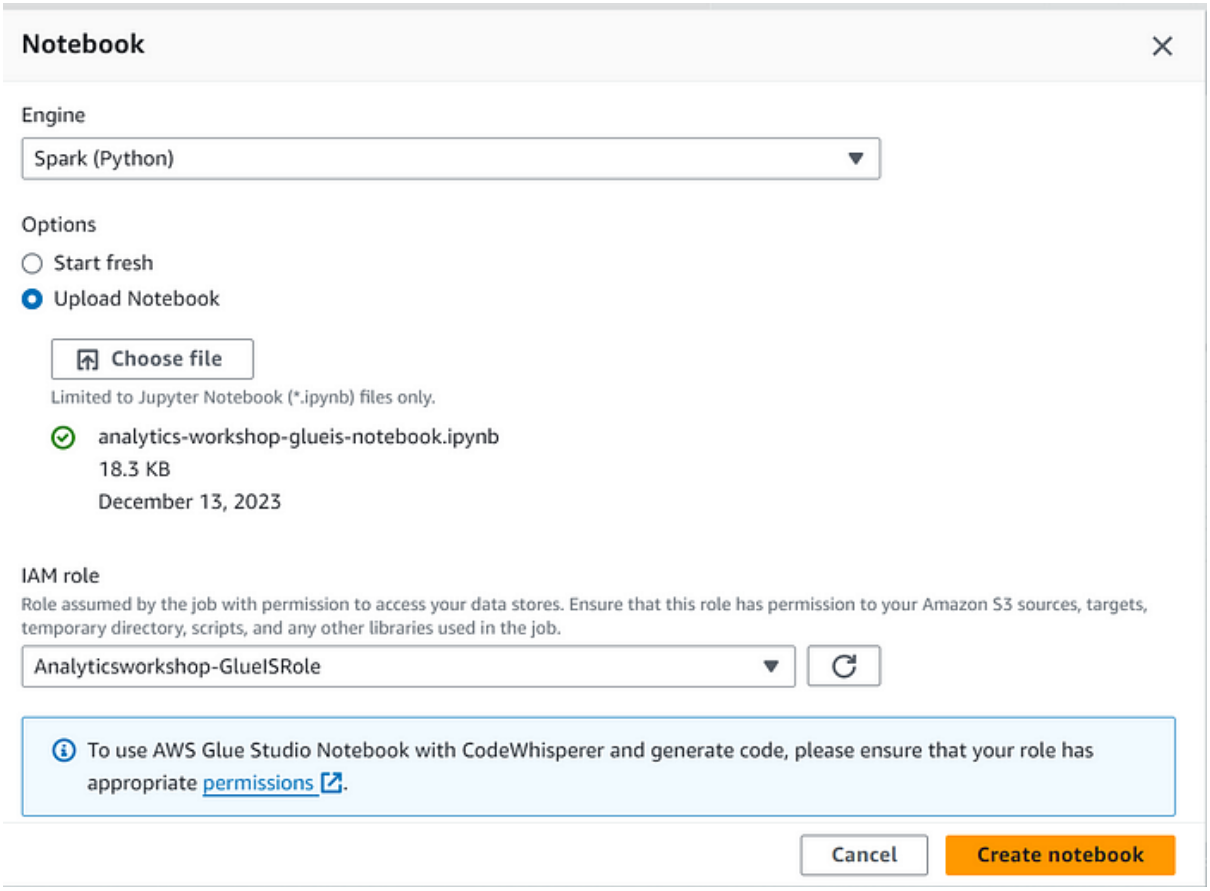
In this step you will be creating an **AWS Glue job with Jupyter Notebook** to interactively develop **Glue ETL scripts** using PySpark.

Download and save this file locally on your laptop : [analytics-workshop-glueis-notebook.ipynb](#). Go to: Glue Studio jobs [Click me](#)

- Select **Jupyter Notebook** option
- Select **Upload and edit an existing notebook.** Click **Choose file**
- Browse and upload **analytics-workshop-glueis-notebook.ipynb** which you downloaded earlier. Click **Create.**

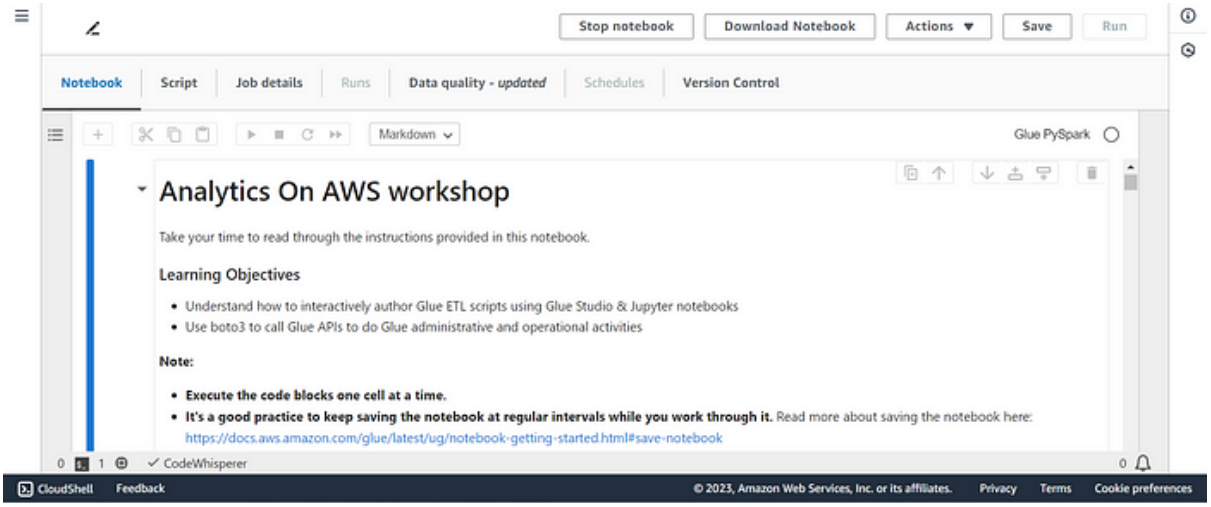


- Under **Notebook setup** and **Initial configuration**
- Job name: `AnalyticsOnAWS-GlueIS`
- IAM role `Analyticsworkshop-GlueISRole`
- Leave **Kernel** to default as `Spark`
- Click **Start notebook**



Once the notebook is initialized, follow the instructions in the notebook

Read and understand the instructions as they explain important Glue concepts.



[1]:
```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import boto3
import time
```

```
Trying to create a Glue session for the kernel.
Session Type: glueetl
Worker Type: G.1X
Number of Workers: 2
Session ID: 9972cd57-84a4-49c6-894b-1d9d3e59f3cb
Applying the following default arguments:
--glue_kernel_version 1.0.2
--enable-glue-datacatalog true
Waiting for session 9972cd57-84a4-49c6-894b-1d9d3e59f3cb to get into ready status...
```

Read More:https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-pyspark-extensions-glue-context.html

Execute Code »

[3]:
```
raw_data = glueContext.create_dynamic_frame.from_catalog(database="analyticsworkshopdb", table_name="raw")

reference_data = glueContext.create_dynamic_frame.from_catalog(database="analyticsworkshopdb", table_name="reference_data")
```

[4]:
```
raw_data.printSchema()
```

```
root
|-- uuid: string
|-- device_ts: string
|-- device_id: int
|-- device_temp: int
|-- track_id: int
|-- activity_type: string
|-- partition_0: string
|-- partition_1: string
|-- partition_2: string
|-- partition_3: string
```

[5]:
```
reference_data.printSchema()
```

```
root
|-- track_id: string
|-- track_name: string
|-- artist_name: string
```

Count records

- In this step we will count the number of records in the dataframe
- count( ): Returns the number of rows in the underlying DataFrame

Execute Code »

[6]:
```
print(f'raw_data (count) = {raw_data.count()}')
print(f'reference_data (count) = {reference_data.count()}')
```

```
raw_data (count) = 92000
reference_data (count) = 100
```

[7]:
```
raw_data.toDF().show(5)
```

```
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|                uuid|          device_ts|device_id|device_temp|track_id|activity_type|partition_0|partition_1|partition_2|partition_3|
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|342c251b-bb11-4d9...|2023-12-12 19:36:...|       17|         28|      16|    Traveling|       2023|         12|         12|         19|
|2ef4c5a2-ffef-407...|2023-12-12 19:36:...|       12|         32|      13|    Traveling|       2023|         12|         12|         19|
|7a0e72ea-f256-406...|2023-12-12 19:36:...|       14|         28|      20|    Traveling|       2023|         12|         12|         19|
|93471858-799e-498...|2023-12-12 19:36:...|       18|         34|      22|    Traveling|       2023|         12|         12|         19|
|e0e33b91-5aed-4e1...|2023-12-12 19:36:...|       38|         32|      26|    Traveling|       2023|         12|         12|         19|
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
only showing top 5 rows
```

[8]:
```
reference_data.toDF().show(5)
```

```
+--------+-----------+--------------------+
|track_id| track_name|         artist_name|
+--------+-----------+--------------------+
|       1| God's Plan|               Drake|
|       2|Meant To Be|Bebe Rexha & Flor...|
|       3|    Perfect|          Ed Sheeran|
|       4|    Finesse|Bruno Mars & Cardi B|
|       5|     Psycho|Post Malone Featu...|
+--------+-----------+--------------------+
only showing top 5 rows
```

[9]:
```
# Adding raw_data as a temporary table in sql context for spark

raw_data.toDF().createOrReplaceTempView("temp_raw_data")

# Running the SQL statement which
runningDF = spark.sql("select * from temp_raw_data where activity_type = 'Running'")
print(f'Running (count): {runningDF.count()}')

runningDF.show(5)
```

```
Running (count): 9294
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|                uuid|          device_ts|device_id|device_temp|track_id|activity_type|partition_0|partition_1|partition_2|partition_3|
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|0cee2f4f-bd4c-4c0...|2023-12-12 19:36:...|       33|         32|      17|      Running|       2023|         12|         12|         19|
|7564761f-f0fa-495...|2023-12-12 19:36:...|       44|         40|      23|      Running|       2023|         12|         12|         19|
|4a3d3cc0-22fc-4cf...|2023-12-12 19:36:...|       28|         32|      10|      Running|       2023|         12|         12|         19|
|67f84bbc-639f-4e9...|2023-12-12 19:36:...|       41|         40|      23|      Running|       2023|         12|         12|         19|
|ce0d0952-c15b-473...|2023-12-12 19:36:...|       47|         34|      17|      Running|       2023|         12|         12|         19|
```

[10]:
```
# Running the SQL statement which
workingDF = spark.sql("select * from temp_raw_data where activity_type = 'Working'")
print(f'Working (count): {workingDF.count()}')

workingDF.show(5)
```

```
Working (count): 18580
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|                uuid|          device_ts|device_id|device_temp|track_id|activity_type|partition_0|partition_1|partition_2|partition_3|
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
|ddb2c11f-5a3e-427...|2023-12-12 19:36:...|       34|         40|      13|      Working|       2023|         12|         12|         19|
|7943aae1-a418-49b...|2023-12-12 19:36:...|       40|         28|      28|      Working|       2023|         12|         12|         19|
|51946c2c-2967-437...|2023-12-12 19:36:...|       14|         32|      23|      Working|       2023|         12|         12|         19|
|8afb93f8-9120-487...|2023-12-12 19:36:...|       22|         28|      14|      Working|       2023|         12|         12|         19|
|6428b660-f5aa-469...|2023-12-12 19:36:...|       47|         28|      19|      Working|       2023|         12|         12|         19|
+--------------------+-------------------+---------+-----------+--------+-------------+-----------+-----------+-----------+-----------+
only showing top 5 rows
```

Glue Transforms - Filtering & Counting - activity_type = Running

- Now, lets perform the same operation using Glue inbuilt transforms
- We will use the **filter** transform
- Filter() - Selects records from a DynamicFrame and returns a filtered DynamicFrame.
- You specify a function, such as a function, which determines whether a record is output (function returns true) or not (function returns false).
- In this function, we are filtering on the condition activity_type == 'Running'

Read More: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-pyspark-transforms-filter.html#aws-glue-api-crawler-pyspark-transforms-filter-example

Execute Code »

```
[ ]: def filter_function(dynamic_record):
        if dynamic_record['activity_type'] == 'Running':
```

Execute Code »

```
[11]: def filter_function(dynamic_record):
        if dynamic_record['activity_type'] == 'Running':
            return True
        else:
            return False
      runningDF = Filter.apply(frame=raw_data, f=filter_function)

      print(f'Running (count): {runningDF.count()}')
```

```
Running (count): 9294
```

Glue Transforms - Filtering & Counting - activity_type = Working (Using python Lambda Expressions)

- Small anonymous functions can be created with the lambda keyword.
- Lambda functions can be used wherever function objects are required. They are syntactically restricted to a single expression.
- Example: This function returns the sum of its two arguments: lambda a, b: a+b.

Execute Code »

```
[12]: workingDF = Filter.apply(frame=raw_data, f=lambda x: x['activity_type'] == 'Working')

      print(f'Working (count): {workingDF.count()}')
```

```
Working (count): 18580
```

- Performs an equality join on two DynamicFrames.
- This transforms accepts the following arguments.
  - frame1: The first DynamicFrame to join
  - frame2: The second DynamicFrame to join
  - keys1: The keys to join on for the first frame
  - keys2: The keys to join on for the second frame
- In our case we will be joining the these two frames : **raw_data** & **reference_data**
- We will be joing these two frames on column **track_id**

Read More: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-pyspark-transforms-join.html

Execute Code »

```
[13]: joined_data = Join.apply(raw_data, reference_data, 'track_id', 'track_id')
```

```
[14]: joined_data.printSchema()
```

```
root
|-- track_id: string
|-- partition_2: string
|-- activity_type: string
|-- .track_id: int
|-- partition_1: string
|-- device_temp: int
|-- track_name: string
|-- artist_name: string
|-- partition_3: string
|-- device_ts: string
|-- device_id: int
|-- partition_0: string
|-- uuid: string
```

Cleaning up the joined_data dynamicframe

- Other than the columns we were interested in we have the partition columns
- These were generated by firehose for placing the files in yyyy/mm/dd/hh directory structure in S3
- We will use Glue's in-built **DropFields** transform to drop partition columns

Read more about AWS Glue transforms here : https://docs.aws.amazon.com/glue/latest/dg/built-in-transforms.html

Execute Code »

```
[15]: joined_data_clean = DropFields.apply(frame=joined_data, paths=['partition_0','partition_1','partition_2','partition_3'])
```

Execute Code »

```
[16]: joined_data_clean.printSchema()
```

```
root
|-- track_id: string
|-- activity_type: string
|-- .track_id: int
|-- device_temp: int
|-- track_name: string
|-- artist_name: string
|-- device_ts: string
|-- device_id: int
|-- uuid: string
```

sample data

```
[17]: joined_data_clean.toDF().show(5)
```

```
+--------+-------------+---------+-----------+----------+--------------------+--------------------+---------+--------------------+
|track_id|activity_type|.track_id|device_temp|track_name|         artist_name|           device_ts|device_id|                uuid|
+--------+-------------+---------+-----------+----------+--------------------+--------------------+---------+--------------------+
|       4|    Traveling|        4|         34|   Finesse|Bruno Mars & Cardi B|2023-12-12 19:36:...|       12|b7b81dc9-434f-40f...|
|       4|    Traveling|        4|         28|   Finesse|Bruno Mars & Cardi B|2023-12-12 19:36:...|       26|8e021de6-3701-431...|
|       4|    Traveling|        4|         28|   Finesse|Bruno Mars & Cardi B|2023-12-12 19:36:...|       13|f87cc281-364d-422...|
|       4|      Walking|        4|         32|   Finesse|Bruno Mars & Cardi B|2023-12-12 19:36:...|       47|a5f704fb-fb15-4f3...|
|       4|      Running|        4|         28|   Finesse|Bruno Mars & Cardi B|2023-12-12 19:36:...|       39|7dff8dd9-d681-4dd...|
+--------+-------------+---------+-----------+----------+--------------------+--------------------+---------+--------------------+
only showing top 5 rows
```

Final step of the transform - Writing transformed data to S3

- In this step we will be using Glue's write_dynamic_frame functionality to write transformed data to S3
- We will be storing the transformed data in a different directory & in parquet format
- make sure you change the S3 bucket name **yourname-analytics-workshop-bucket** to reflect your bucket name

---

- Why parquet format ?
  - Apache Parquet is a columnar storage formats that is optimized for fast retrieval of data and used in AWS analytical applications.
  - Columnar storage formats have the following characteristics that make them suitable for using with Athena: Compression by column, with compression algorithm selected for the column data type to save storage space in Amazon S3 and reduce disk space and I/O during query processing.
  - Predicate pushdown in Parquet and ORC enables queries to fetch only the blocks it needs, improving query performance.
  - When a query obtains specific column values from your data, it uses statistics from data block predicates, such as max/min values, to determine
  - Splitting of data in Parquet allows analytics tools to split the reading of data to multiple readers and increase parallelism during its query processing.

Execute Code »

```
[18]: try:
          datasink = glueContext.write_dynamic_frame.from_options(
              frame = joined_data_clean, connection_type="s3",
              connection_options = {"path": "s3://yourname-analytics-workshop-bucket/data/processed-data/"},
              format = "parquet")
          print('Transformed data written to S3')
      except Exception as ex:
          print('Something went wrong')
          print(ex)
```

Validate — Transformed / Processed data has arrived in S3

Once the ETL script has ran successfully, return to the console: Click me

Click — **yourname-analytics-workshop-bucket > data**

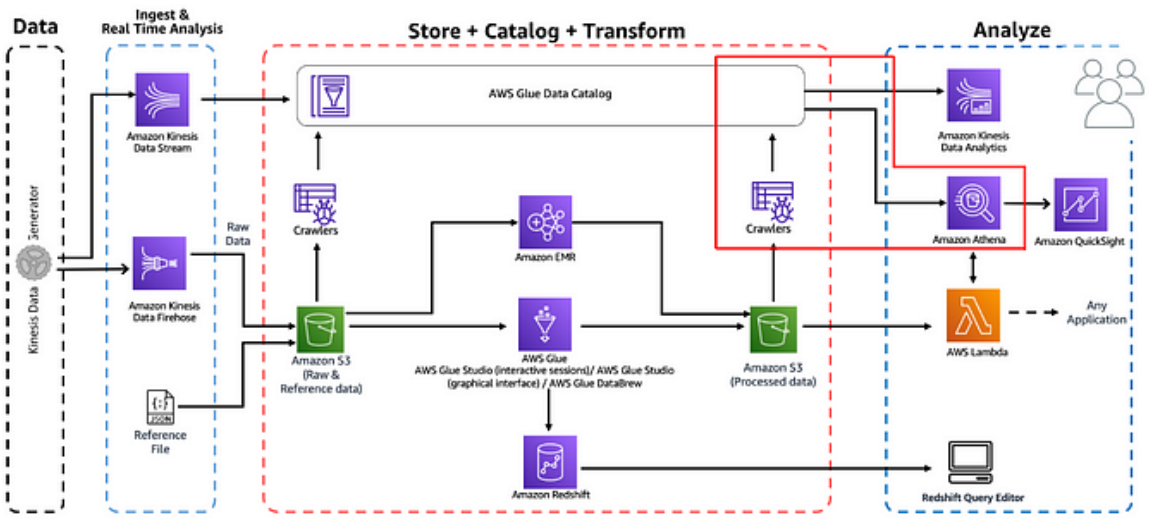
Open the **processed-data** folder: Ensure that **.parquet** files have been created in this folder.

Now that we have transformed the data, we can query the data using **Amazon Athena**. We could also further transform/aggregate the data with AWS Glue or Amazon EMR.

XXXXXXXXXXXX

Task 4 👉: Analyze with Athena

So far, we have stored a few data sets in Amazon S3, and cataloged them in **AWS Glue data catalog**. With **Amazon Athena** we will be able to explore the data using Standard **SQL** queries.



Login to the Amazon Athena Console. Goto: Athena Console [Click me]

If you see a notification requiring you to first create an S3 bucket to store the results of your queries, follow these steps:

- Go to the **S3 console** and create a bucket using your preferred name, e.g. **yourname-query-results**
- After creating the bucket, return to the Athena console and click **'Settings'** on the top-right of the console.
- Enter the name of the bucket you have just created, ensuring you include a trailing slash: **s3://yourname-query-results/**
- Hit **Save**

As **Athena** uses the **AWS Glue catalog** for keeping track of data source, any **S3** backed table in **Glue** will be visible to **Athena**. On

the left panel, select '**analyticsworkshopdb**' from the drop down. Run the following query:

```
SELECT artist_name,        count(artist_name) AS count FROM processed_data
GROUP BY artist_name ORDER BY count desc
```

Explore the Athena UI and try running some queries. Try querying the `emr_processed_data` table. This query returns the list of tracks repeatedly played by devices. Later, we will visualize this query using QuickSight:

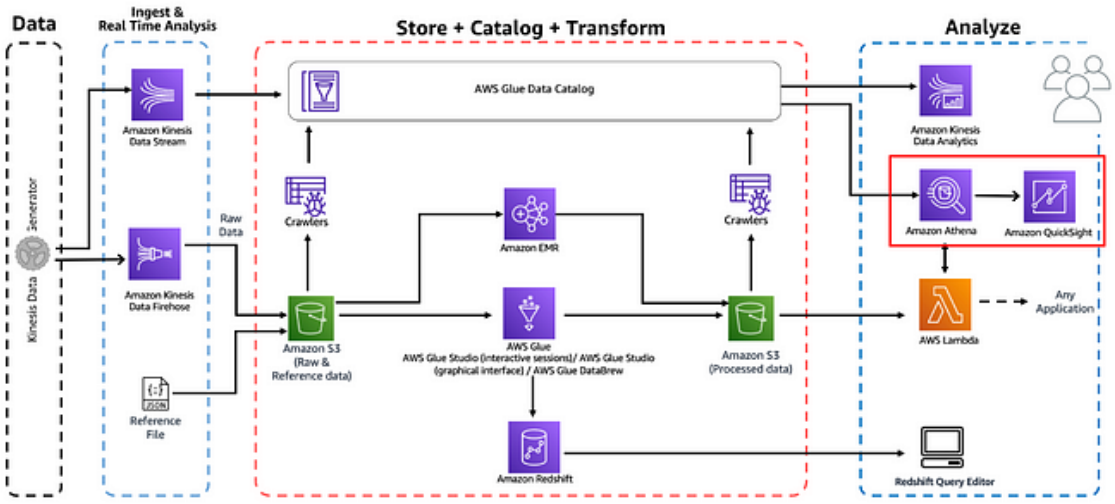
```
SELECT device_id,
       track_name,
       count(track_name) AS count
FROM processed_data
GROUP BY device_id, track_name
ORDER BY count desc
```

We could run any similar Athena queries and explore the data further.

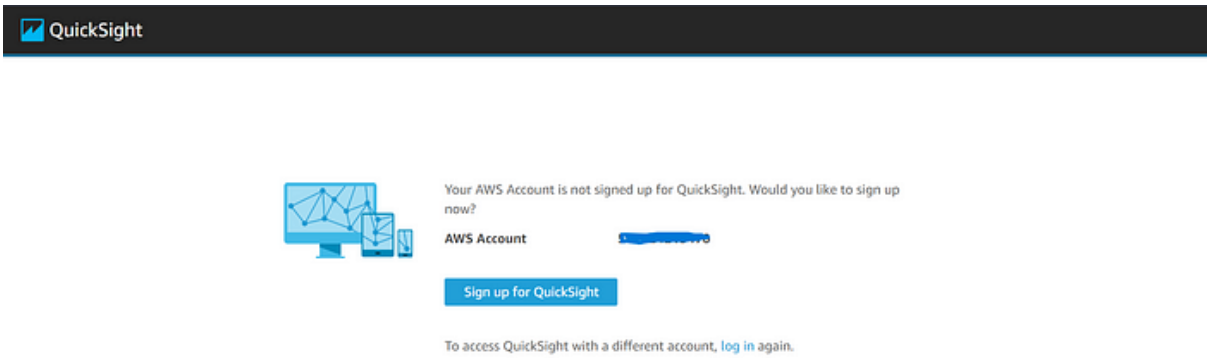
XXXXXXXXXXXXXXXXX

Task 5 👉: Visualize in Quicksight

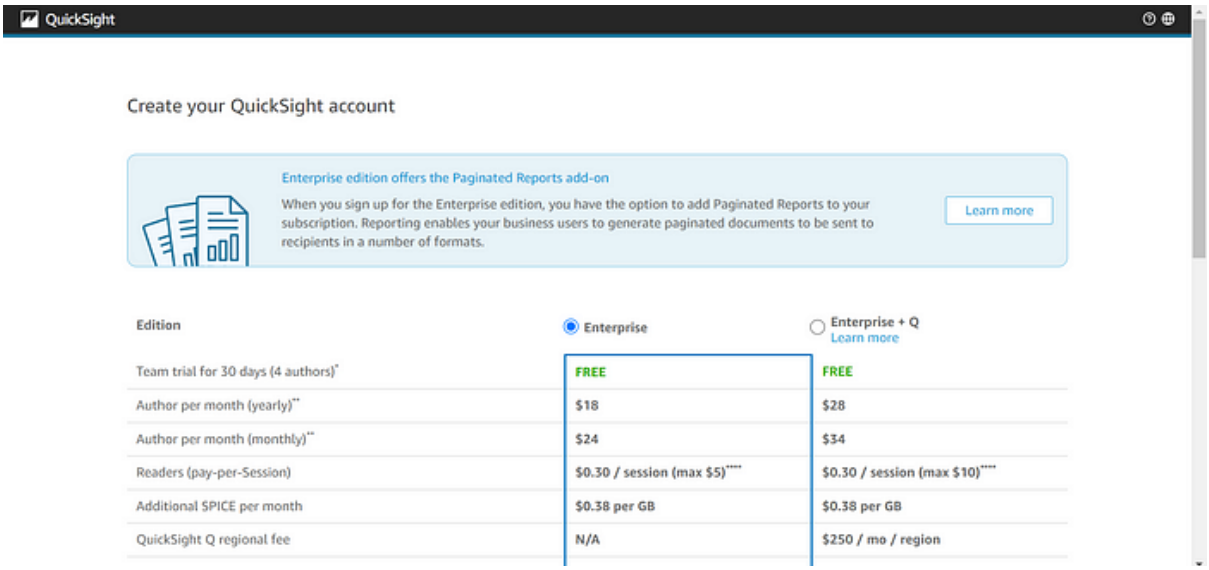
In this module, we are going use Amazon Quicksight to build a few visualizations over the data collected and stored in S3.



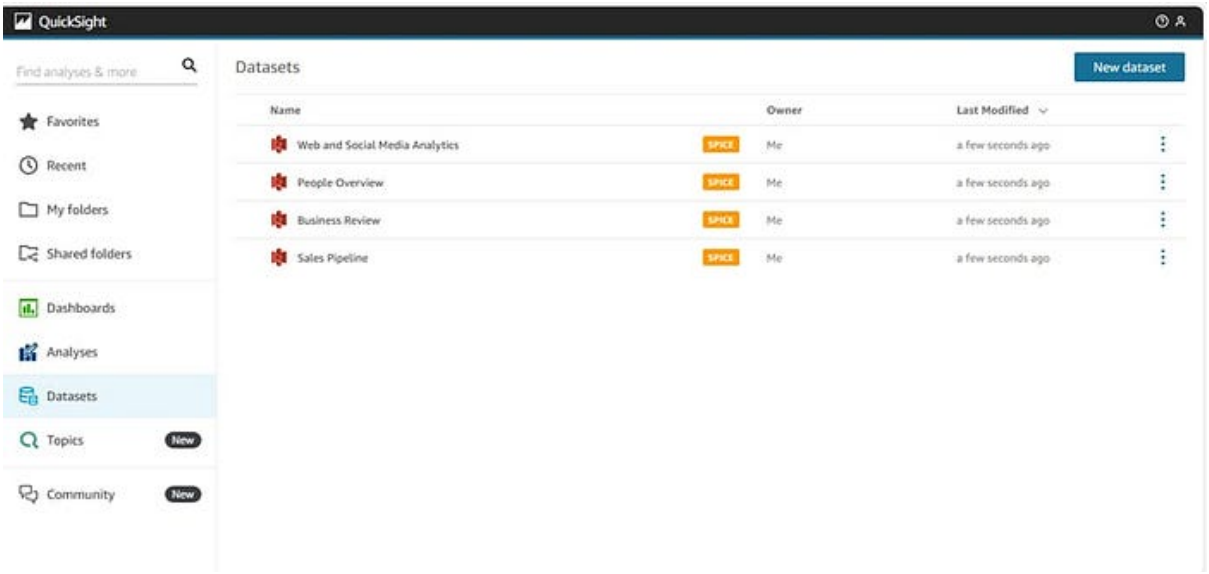
Please feel free to explore the different visualization options available in Quicksight!
In this step we will visualize our processed data using QuickSight.
Goto: Quicksight Console [Click me]

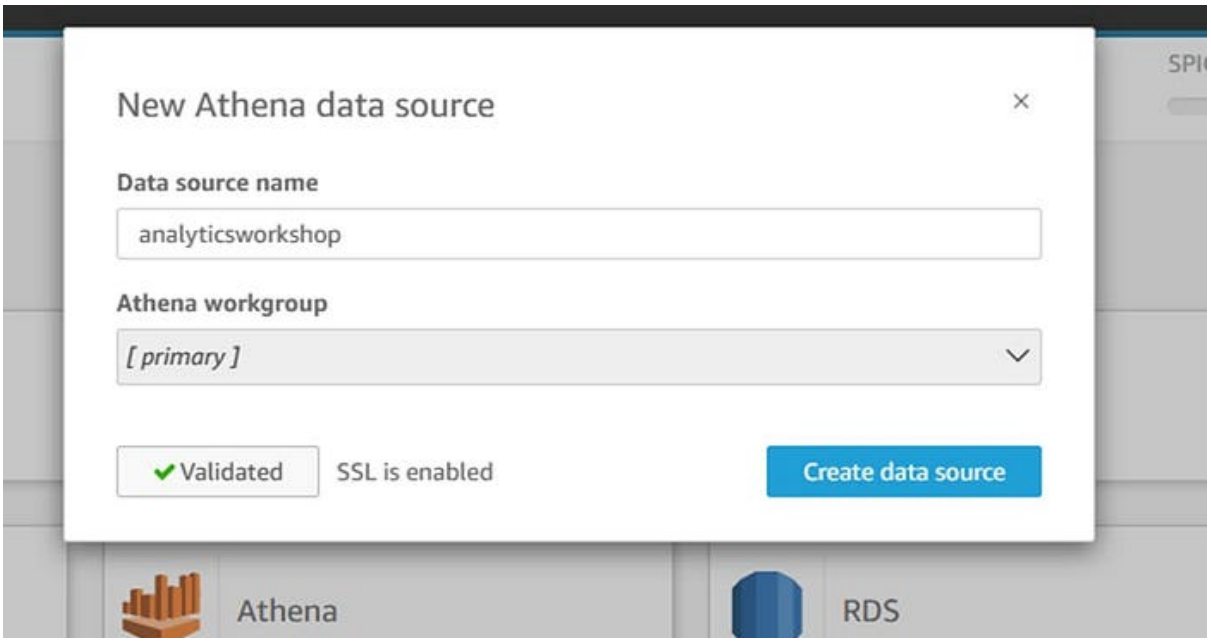


Create account. Click **Sign up for QuickSight**
Ensure **Enterprise** is selected and click **Continue**
QuickSight account name: `yournameanalyticsworkshop`
Notification email address: `you@youremail.com`
Select **Amazon Athena** — this enables QuickSight access to Amazon Athena databases.
Select **Amazon S3.** Select **yourname-analytics-workshop-bucket.**
Click **Finish.** Wait for your QuickSight account to be created.

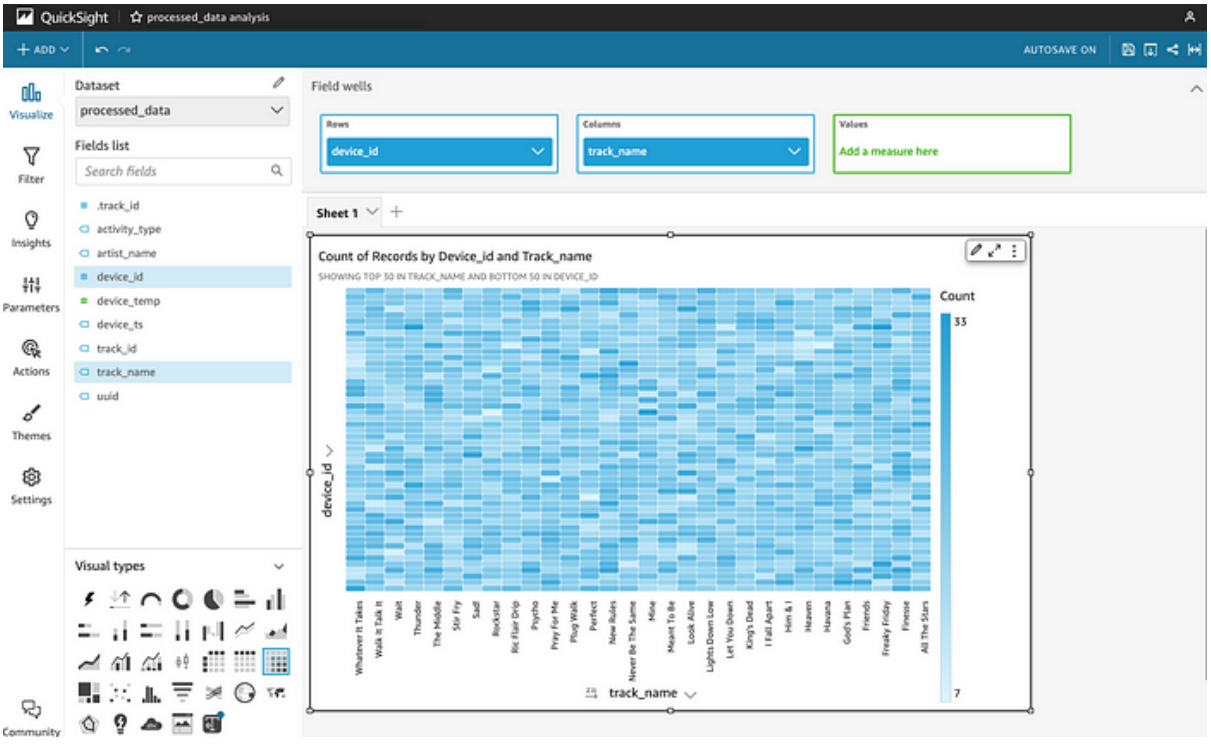


[Adding a New Dataset](#) Go to: Click me. On top right, click **New Dataset.**
Click **Athena.** New Athena data source.
Data source name: `analyticsworkshop`





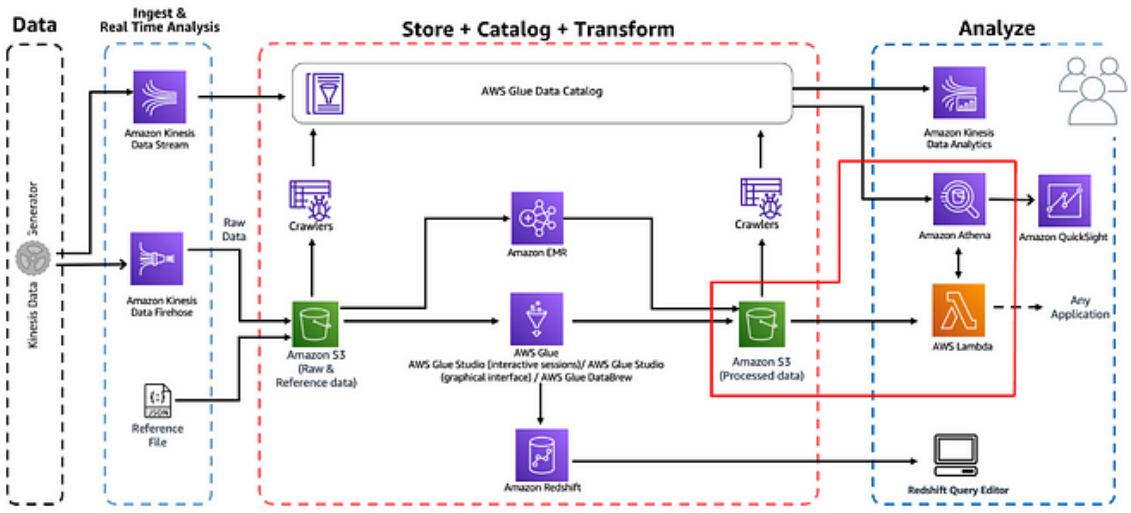
Click **Validate connection.** This will check if your QuickSight can access Athena
Click **Create data source.** Choose your table:
Database: contain sets of tables — select **analyticsworkshopdb**
Tables: contain the data you can visualize —
select **processed_data**
Click **Select.** Finish data set creation:
Select **Directly query your data.** Click **Visualize.**



XXXXXXXXXXXXXXXXXXXXX

Task 6👉: Serve with Lambda

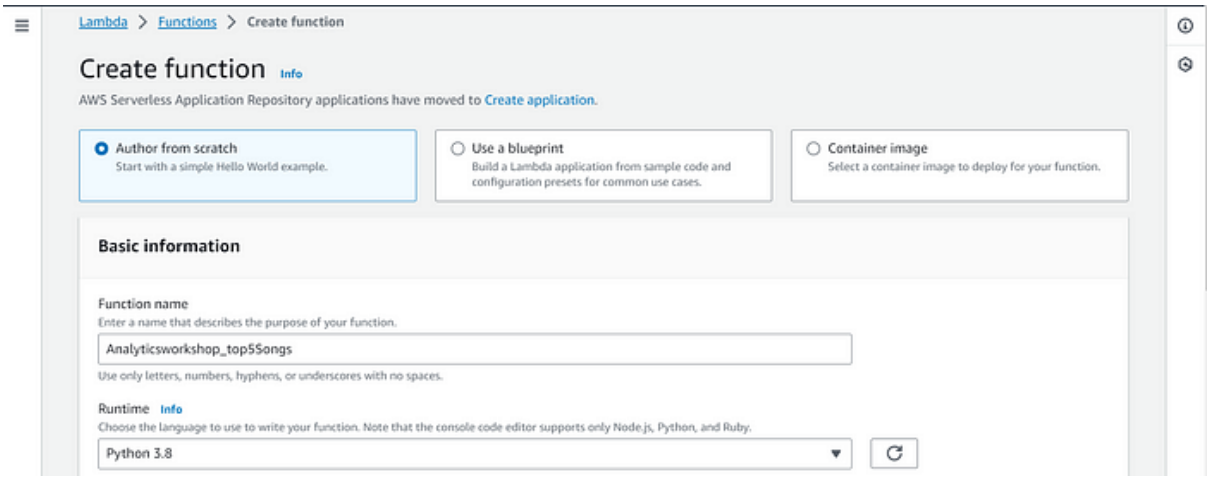
In this module we are going to create a Lambda Function with a very specific use case example. The lambda function we are going to write will host the code for Athena to query and fetch Top 5 Popular Songs by Hits from processed data in S3.

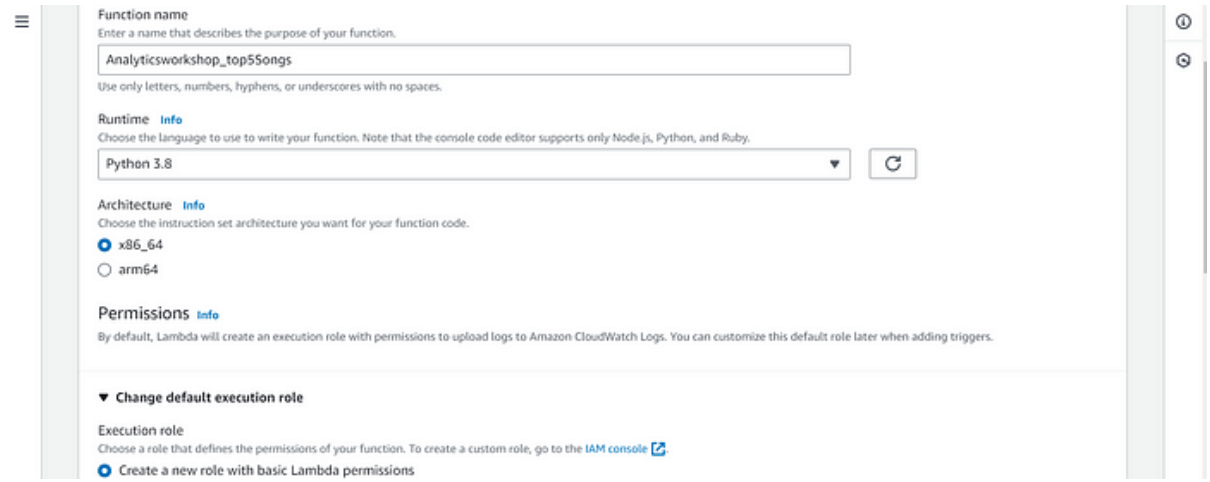


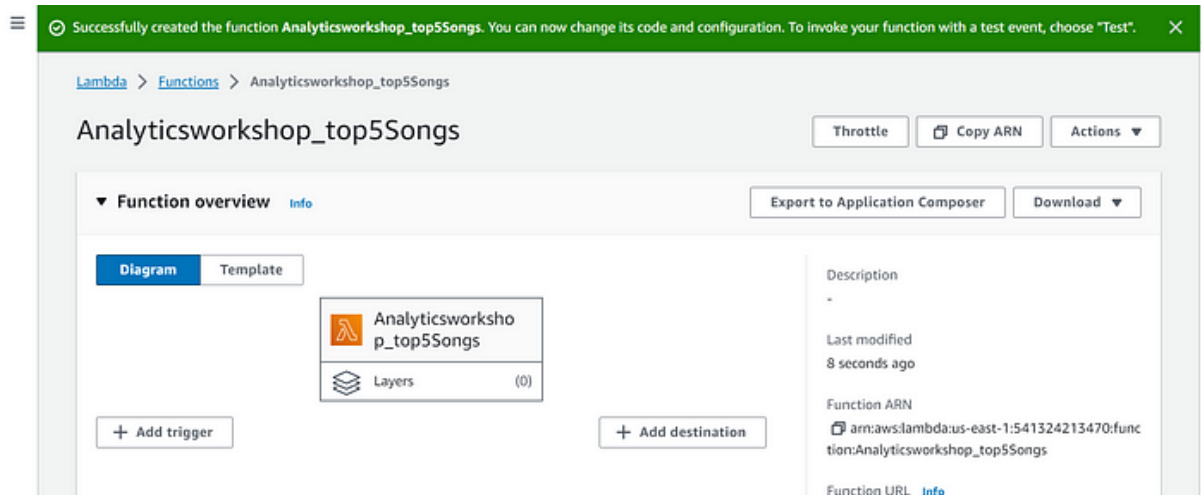
a) C[reating a Lambda Function]
Go to: Lambda Console [Click me]. Click **Create function.**
Select **Author from scratch.**
Under **Basic Information**:

- Give Function name as `Analyticsworkshop_top5Songs`
- Select Runtime as **Python 3.8**

- Expand **Choose or create an execution role** under Permissions, make sure **Create a new role with basic Lambda permissions** is selected. Click **Create Function**







Function Code

Scroll down to Function Code section and replace existing code under in **lambda_function.py** with the python code below:

```
import boto3
import time
import os

# Environment Variables
DATABASE = os.environ['DATABASE']
TABLE = os.environ['TABLE']
# Top X Constant
TOPX = 5
# S3 Constant
S3_OUTPUT = f's3://{os.environ["BUCKET_NAME"]}/query_results/'
# Number of Retries
RETRY_COUNT = 10

def lambda_handler(event, context):
    client = boto3.client('athena')
    # query variable with two environment variables and a constant
    query = f"""
        SELECT track_name as \"Track Name\",
                artist_name as \"Artist Name\",
                count(1) as \"Hits\"
        FROM {DATABASE}.{TABLE}
        GROUP BY 1,2
        ORDER BY 3 DESC
        LIMIT {TOPX};
    """
```

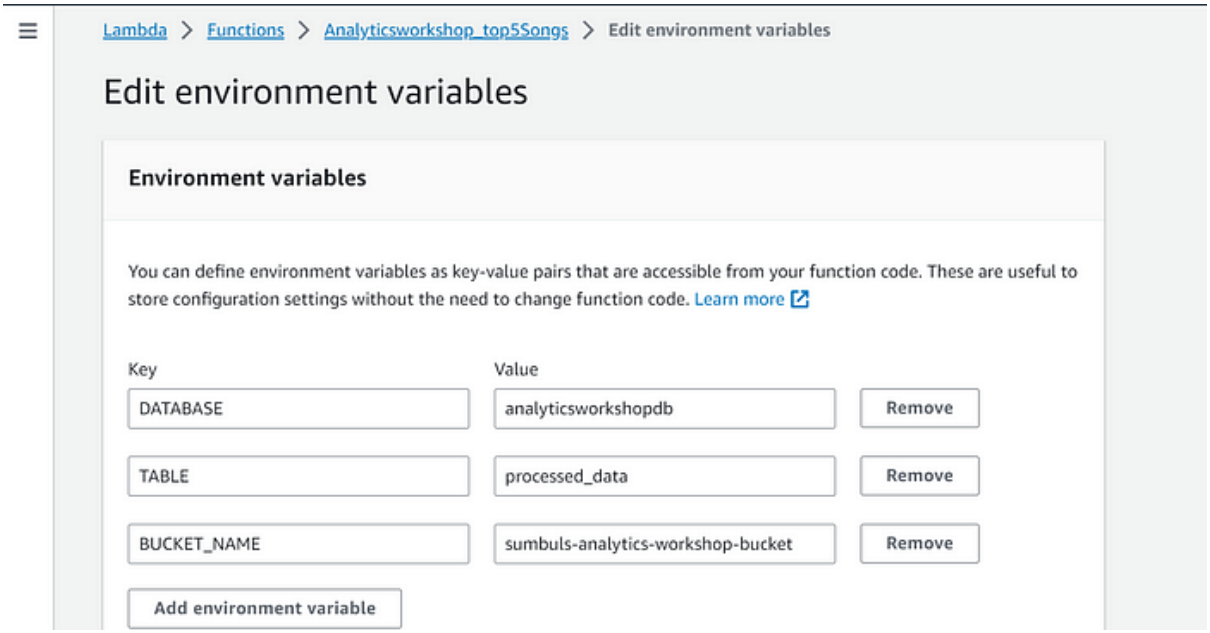
```
    response = client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={ 'Database': DATABASE },
        ResultConfiguration={'OutputLocation': S3_OUTPUT}
    )
    query_execution_id = response['QueryExecutionId']
    # Get Execution Status
    for i in range(0, RETRY_COUNT):
        # Get Query Execution
        query_status = client.get_query_execution(
            QueryExecutionId=query_execution_id
        )
        exec_status = query_status['QueryExecution']['Status']['State']
        if exec_status == 'SUCCEEDED':
            print(f'Status: {exec_status}')
            break
        elif exec_status == 'FAILED':
            raise Exception(f'STATUS: {exec_status}')
        else:
            print(f'STATUS: {exec_status}')
            time.sleep(i)
    else:
        client.stop_query_execution(QueryExecutionId=query_execution_id)
        raise Exception('TIME OVER')
    # Get Query Results
    result = client.get_query_results(QueryExecutionId=query_execution_id)
    print(result['ResultSet']['Rows'])
    # Function can return results to your application or service
    # return result['ResultSet']['Rows']
```

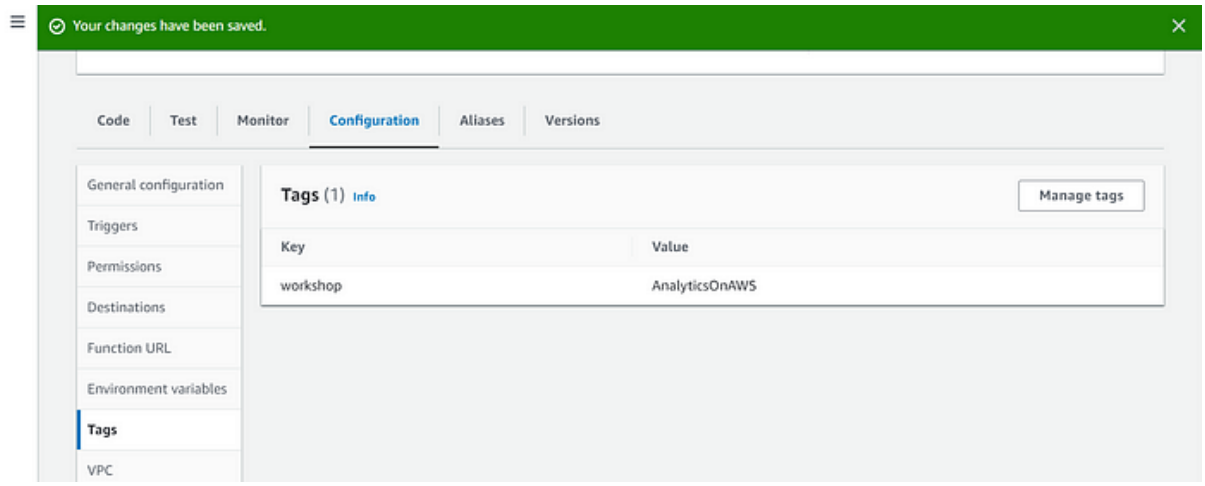
Environment Variables

Environment variables for Lambda functions enable you to dynamically pass settings to your function code and libraries, without making changes to your code. Read more about Lambda Environment Variables here — *https://docs.aws.amazon.com/lambda/latest/dg/env_variables.html*

Scroll down to **Environment variables** section and add below three Environment variables.

- Key: `DATABASE`, Value: `analyticsworkshopdb`
- Key: `TABLE`, Value: `processed_data`
- Key: `BUCKET_NAME`, Value: `yourname-analytics-workshop-bucket`
- Leave the **Memory (MB)** as default which is 128 MB
- Change **Timeout** to 10 seconds.
- Optionally add Tags, e.g.: **workshop**: **AnalyticsOnAWS**
- Click **Save**





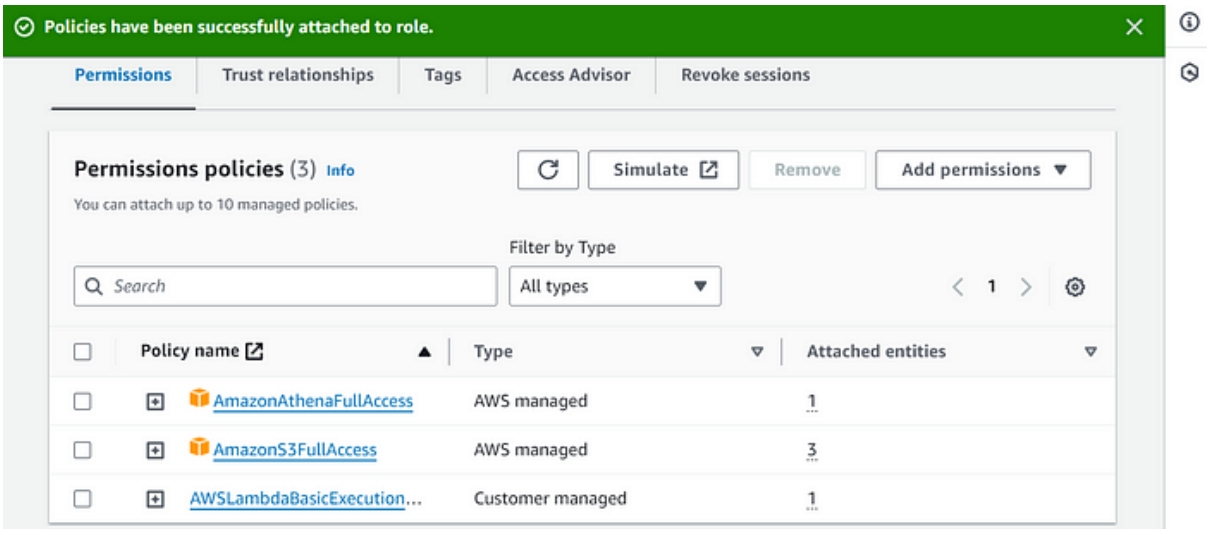
[Execution Role]

Select the **Permissions** Tab at the top: Click the Role Name link under **Execution Role** to open the IAM Console in a new tab. Click **Add permissions** and click **Attach policies**

Add the following two policies (search in filter box, check and hit Attach policy):`AmazonS3FullAccess`

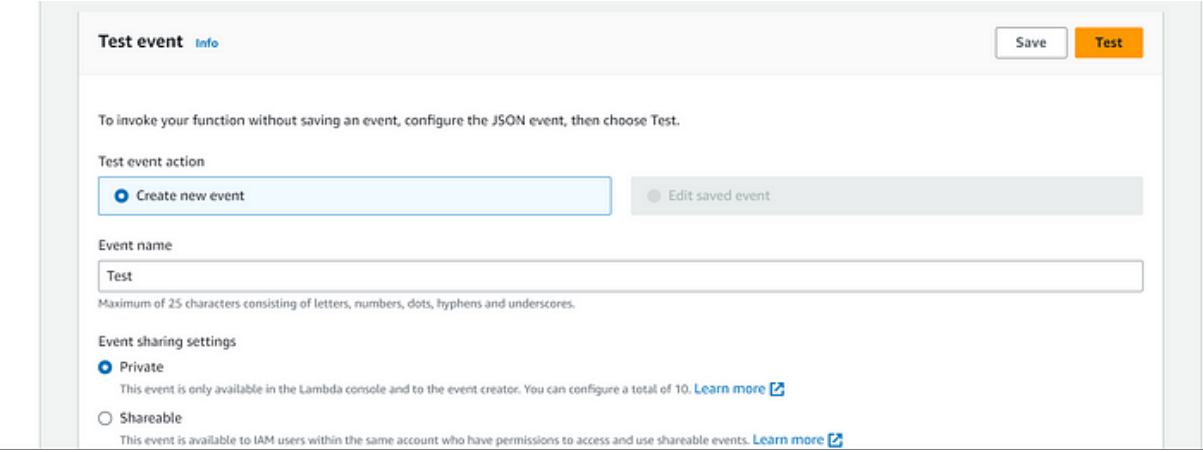
`AmazonAthenaFullAccess`

Once these policies are attached to the role, close this tab.



Configuring The Test Event

Our function is now ready to be tested. Deploy the function first by clicking on **Deploy** under the Function code section. Next, let's configure a dummy test event to see execution results of our newly created lambda function.



Click **Test** on right top hand corner of the lambda console. A new window will pop up for us to configure test event.
Create new test event is selected by default.
Event name: **Test**
Template: **Hello World.** Leave everything as is.
Click **Save.** Click **Test** again

You should be able to see the output in json format
under **Execution Result** section.

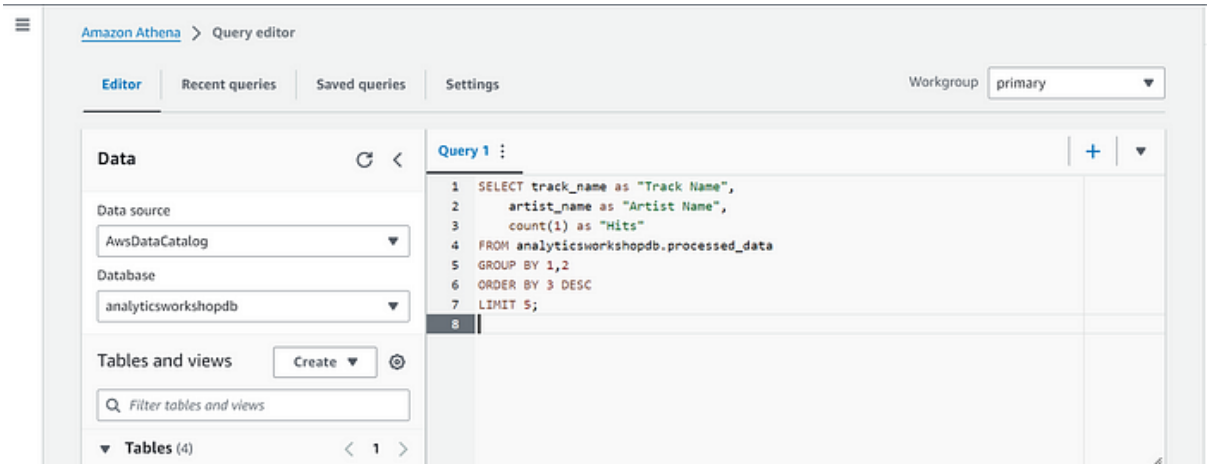
Verification through Athena

Let's verify the results through Athena. Goto: Athena Console Click me.
On the left panel, select **analyticsworkshopdb** from the

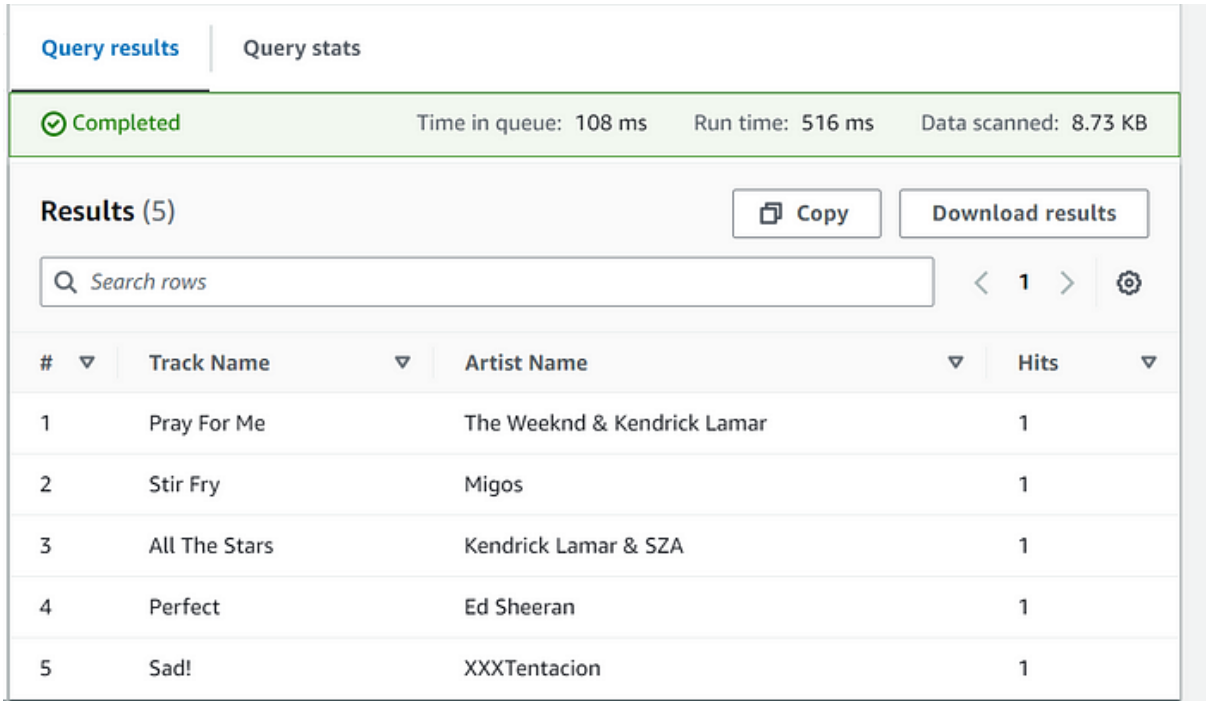
dropdown.

Run the following query:

```
SELECT track_name as "Track Name",:
    artist_name as "Artist Name",
    count(1) as "Hits"
FROM analyticsworkshopdb.processed_data
GROUP BY 1,2
ORDER BY 3 DESC
LIMIT 5;
```



Compare the results of this query with the results of **lambda** function; they should be identical.



You have now created a lambda function from scratch and tested it.

XXXXXXXXXXXXXXX

Task 7 👉: Clean up

Failing to clean up all the resources will result in incurring continued AWS usage charges. Make sure you delete all resources created as part of this lab by following all the steps below.
Resources to delete

- Kinesis Firehose Delivery Stream. Go to: Kinesis Console Click me. Delete Firehose: **analytics-workshop-stream**

- Kinesis Data Stream. Go to: Kinesis Console Click me. Delete Data Stream: **analytics-workshop-data-stream**
- Kinesis Data Analytics Studio Notebook. Go to: Kinesis Console Click me. Delete Notebook: **AnalyticsWorkshop-KDANotebook**
- Lambda. Goto: Lambda Console Click me. Navigate to list of functions and select **Analyticsworkshop_top5Songs**. Under **Actions** drop down menu, select **Delete**.
- **Glue Database.** Go to: Glue Console Click me. Delete Database: **analyticsworkshopdb**
- **Glue Crawler.** Go to: Glue Crawlers Click me Delete Crawler: **AnalyticsworkshopCrawler**
- **Glue Studio Job**. GoTo: https://us-east-1.console.aws.amazon.com/gluestudio/home?region=us-east-1#/jobs Check **AnalyticsOnAWS-GlueStudio** Check **AnalyticsOnAWS-GlueIS**
- Delete **IAM Role.** Go to: IAM Console Click me Search for following roles and delete: **Analyticsworkshop-GlueISRole** **Analyticsworkshop_RedshiftRole** **AnalyticsworkshopKinesisAnalyticsRole** **Analyticsworkshop_top5Songs-role-**
- Delete **IAM Policy** Go to: IAM Console Click me Search for following policies and delete: **AWSGlueInteractiveSessionPassRolePolicy**

- Delete **S3 bucket** Go to: S3 Console Click me Delete Bucket: **yourname-analytics-workshop-bucket.** You may need to first **Empty** the bucket as prompted. Once emptied, proceed to **Delete** the bucket
- Delete the **Cognito CloudFormation Stack.** Go to: CloudFormation Click me Click: **Kinesis-Data-Generator-Cognito-User** Click: **Actions** > **DeleteStack.** On confirmation screen:Click **Delete**
- Close **QuickSight** account. Go to: Quicksight Console Click me Click: **Unsubscribe**
- **Cognito Userpool.** Go to: Cognito Console Click me Click **Kinesis Data-Generator Users.** Click **Delete Pool**

That's it! Hope you found the workshop useful!

Conclusion

- We designed a serverless data lake architecture
- Build a data processing pipeline and Data Lake using Amazon S3 for storing data.
- Use Amazon Kinesis for real-time streaming data
- Use Amazon Kinesis Data Analytics for real-time data analysis

- Use AWS Glue to automatically catalog datasets
- Did Data Transformation
- Ran interactive ETL scripts in a Jupyter notebook on AWS Glue Studio using AWS Glue interactive sessions
- Used Glue Studio to run, and monitor ETL jobs in AWS Glue.
- Query data using Amazon Athena & visualize it using Amazon QuickSight.

Thanks,
Happy learning & sharing 😊
Rohit Manral

References:

- **https://000072.awsstudygroup.com/1-introduce/**
- **https://catalog.us-east-1.prod.workshops.aws/workshops/44c91c21-a6a4-4b56-bd95-56bd443aa449/en-US/lab-guide**
- **Medium**