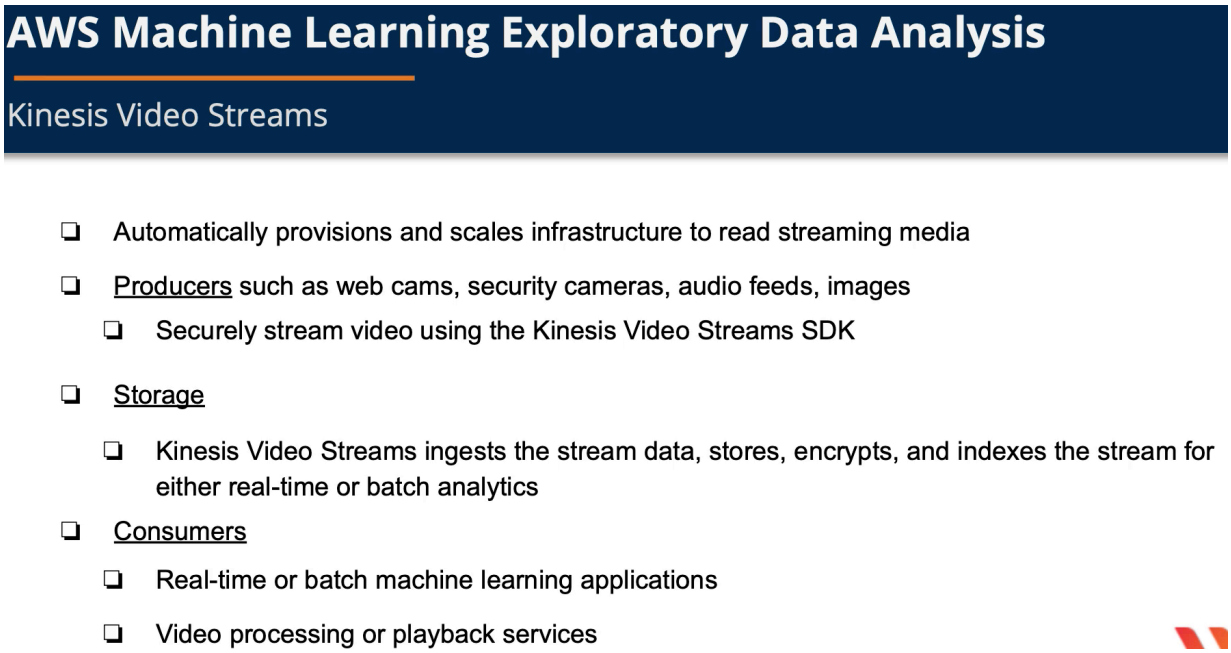
Whizlabs - ML Specialty Exam Course - Data Analysis - part 2

https://www.whizlabs.com/learn/course/aws-mls-practice-tests/video/3494

Kinesis Video Stream

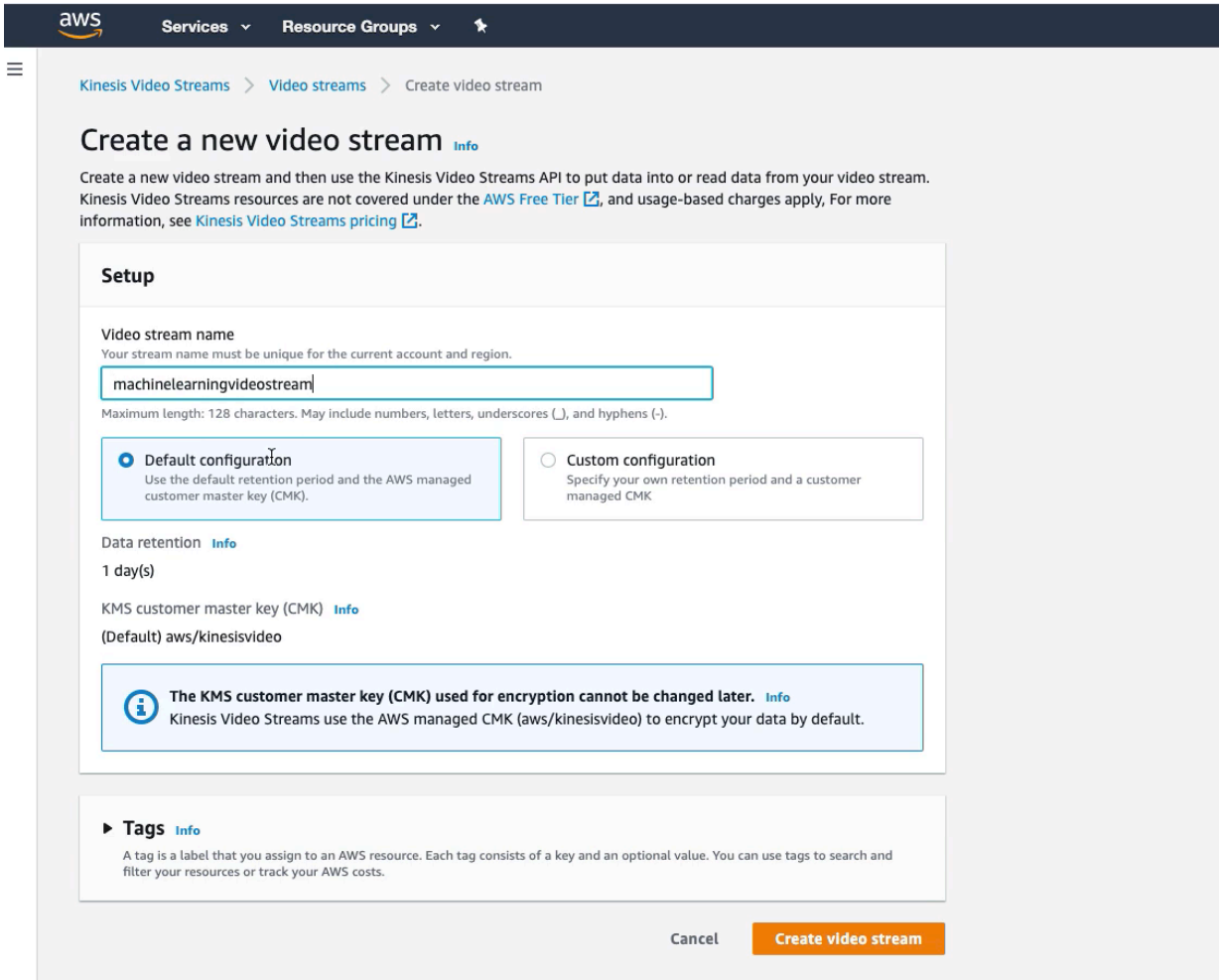


Use Cases

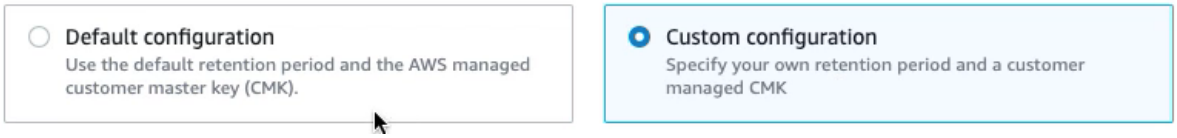
Kinesis Video Streams in Machine Learning

- Facial recognition
  - Facial analytics using machine learning services and algorithms such as the SageMaker built-in Image Classification algorithm or the Rekognition service
- Object detection
  - Object identification algorithm to detect when certain objects appear in the stream
- Computer vision and video analytics using machine learning frameworks such as Apache MxNet, TensorFlow, and OpenCV
- Smart home/city applications
  - Home surveillance, Red light cams
- Industrial automation
  - Predictive maintenance using MxNet, TensorFlow, and OpenCV

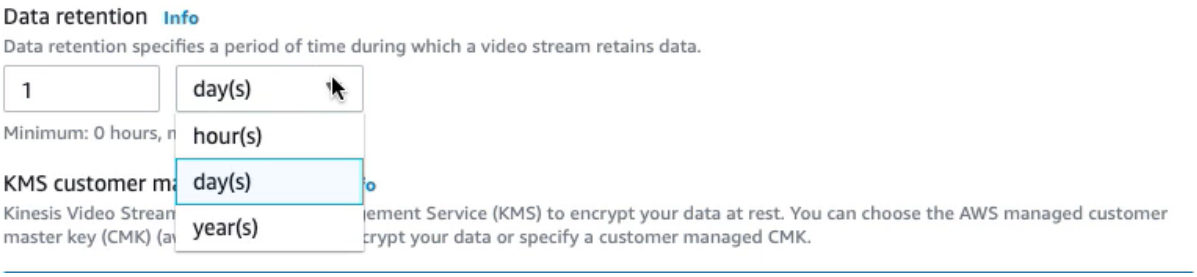
Quick Tour:



To note we can specify our own Customer Manged Key (CMK)

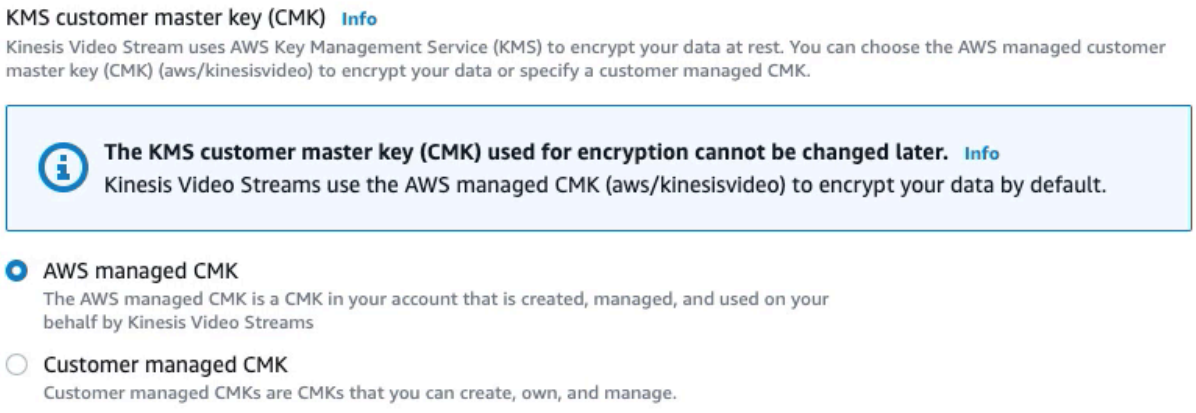


Retention period can be in hours, days and years

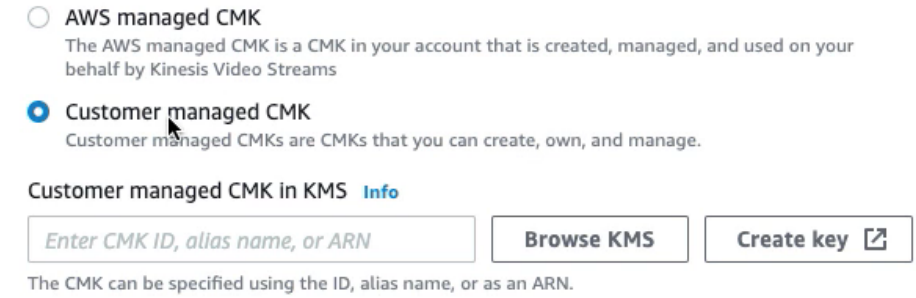


Default is 1 day. Max is 10 years.

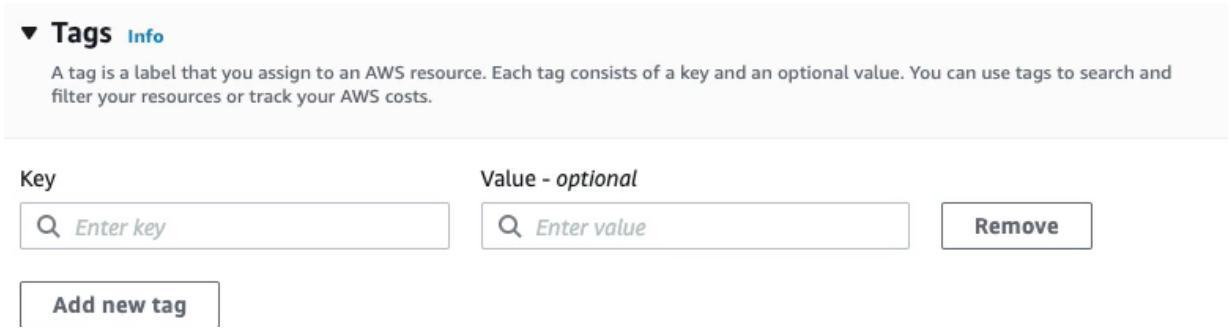
As for KMS - data is encrypted before it is stored to video stream storage layer (S3)
Data is always encrypted at REST



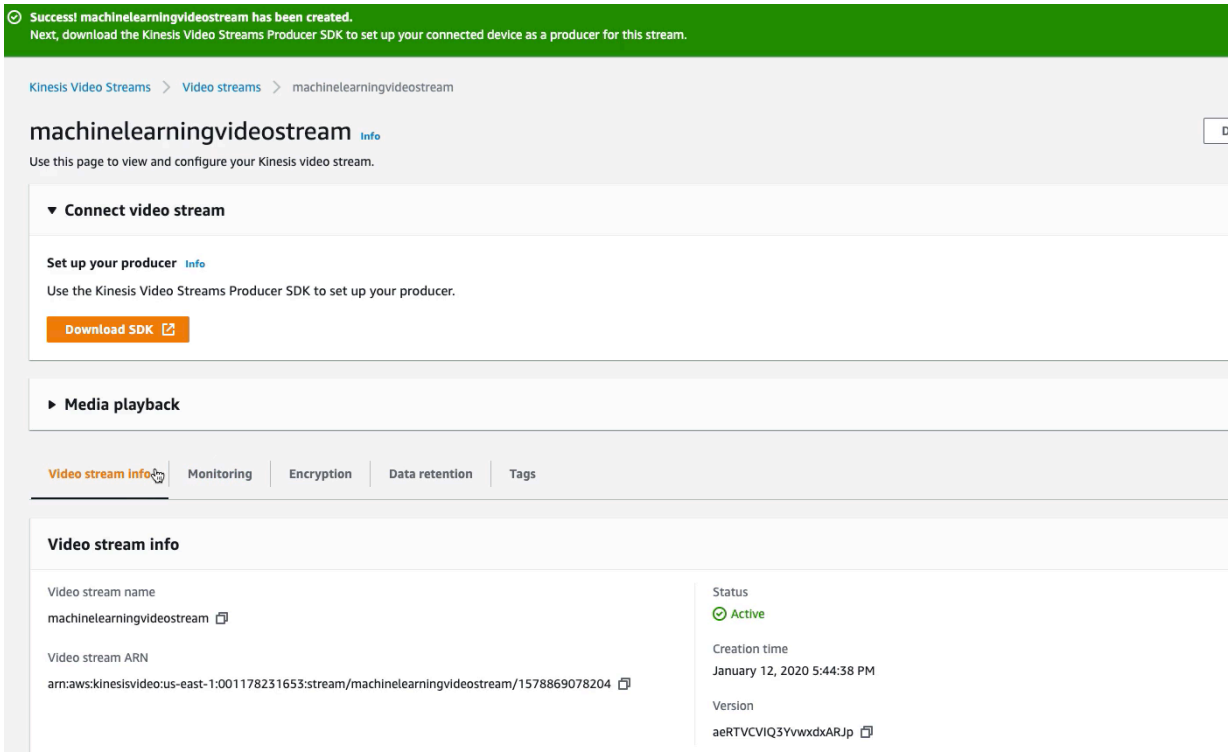
We can also set up our own key:

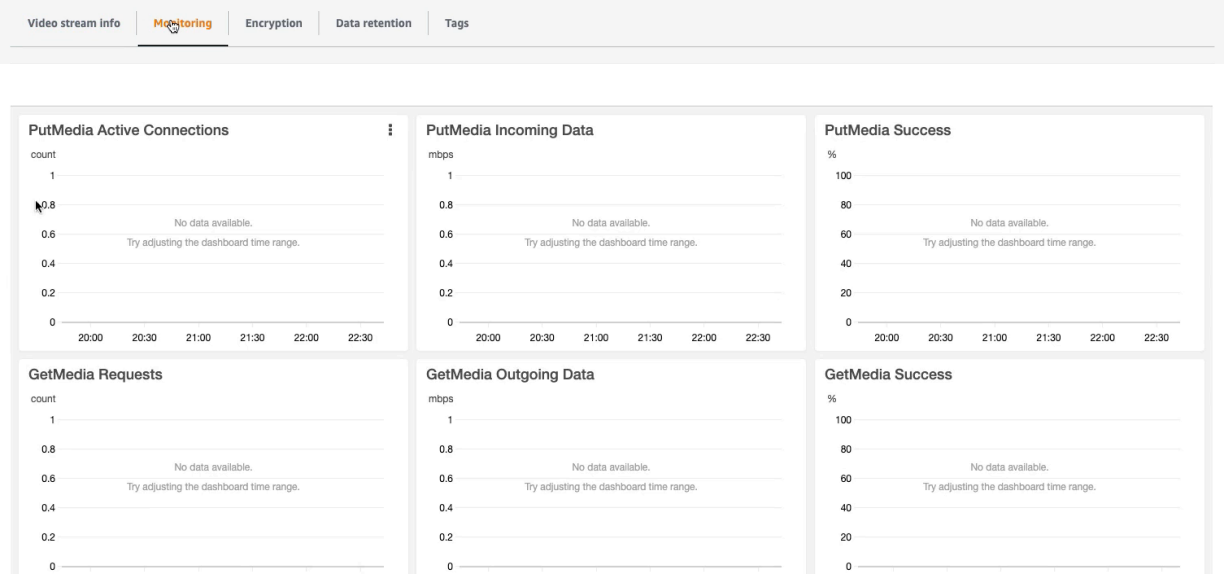


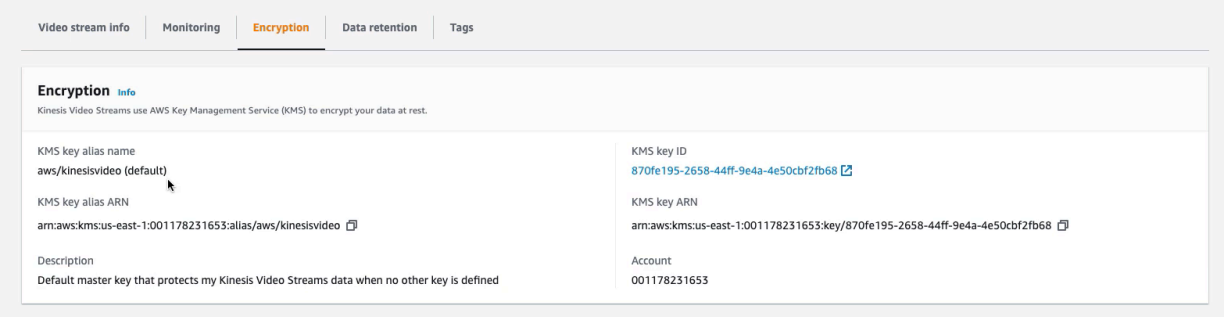
We can tag the video stream

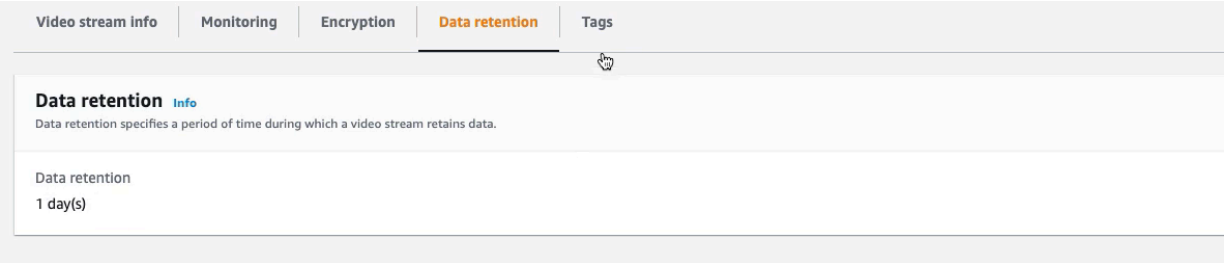


Now the video stream is created

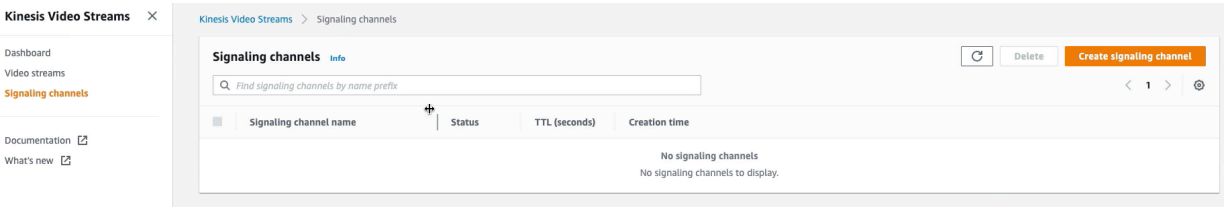






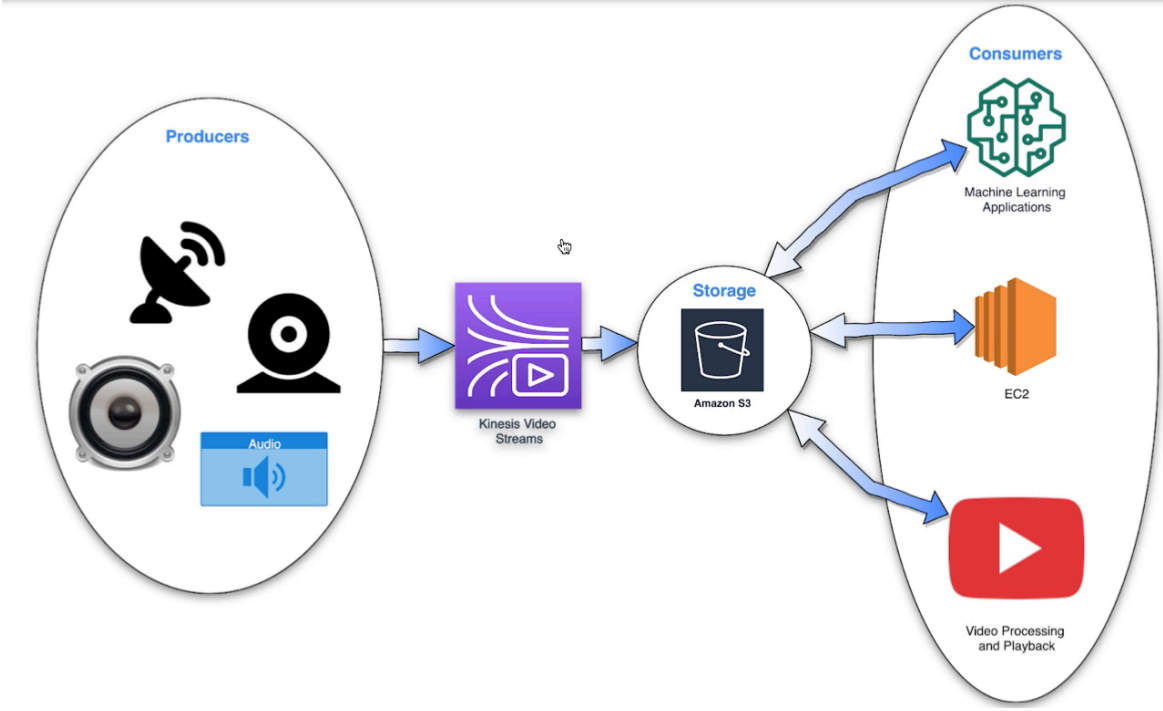


We can also look at Signaling channels

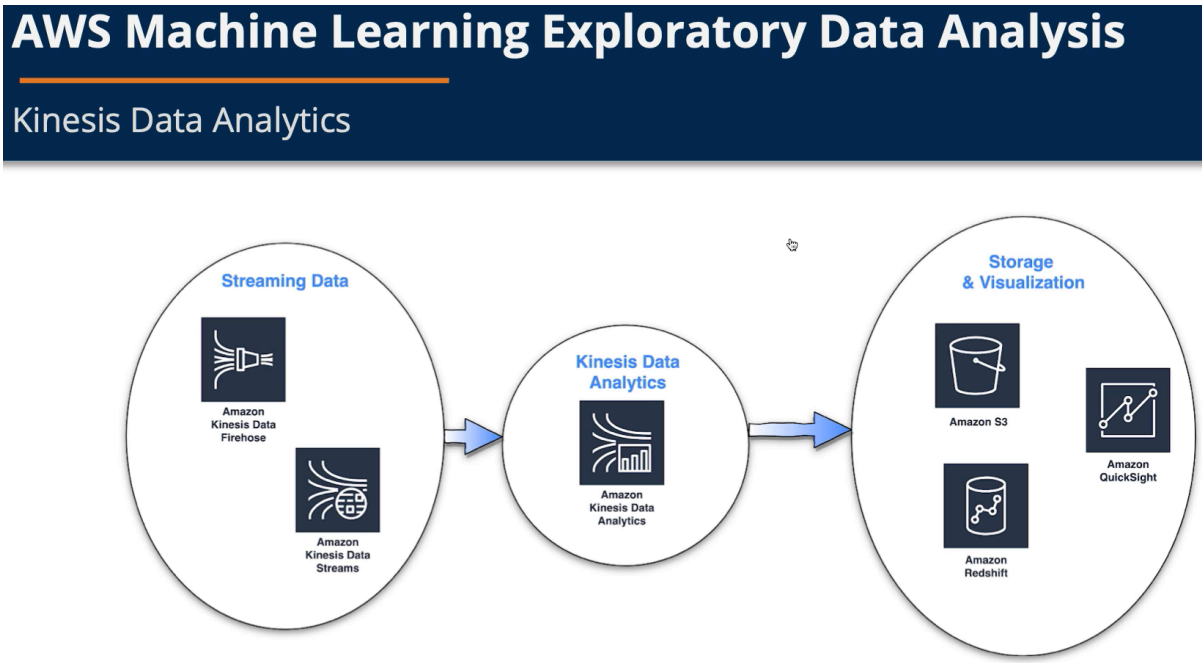


For example; doorbell camera - we can see it and talk to them

Typical architecture for Kinesis Video Streams



Kinesis Data Analytics



- ❏ Use SQL to process streaming data
- ❏ Sources: Kinesis Data Streams and Kinesis Data Firehose
- ❏ SQL queries put to S3, Redshift, or visualization and Business Intelligence tools
- ❏ A Kinesis Data Analytics streaming application consists of three parts
  - ❏ Streaming data sources
  - ❏ Analytics written in SQL
  - ❏ Destinations for the results
- ❏ The streaming application continuously reads data from a streaming source, generates analytics using SQL code, and emits results to 1 to 4 destinations

Kinesis Data Analytics Streaming Application

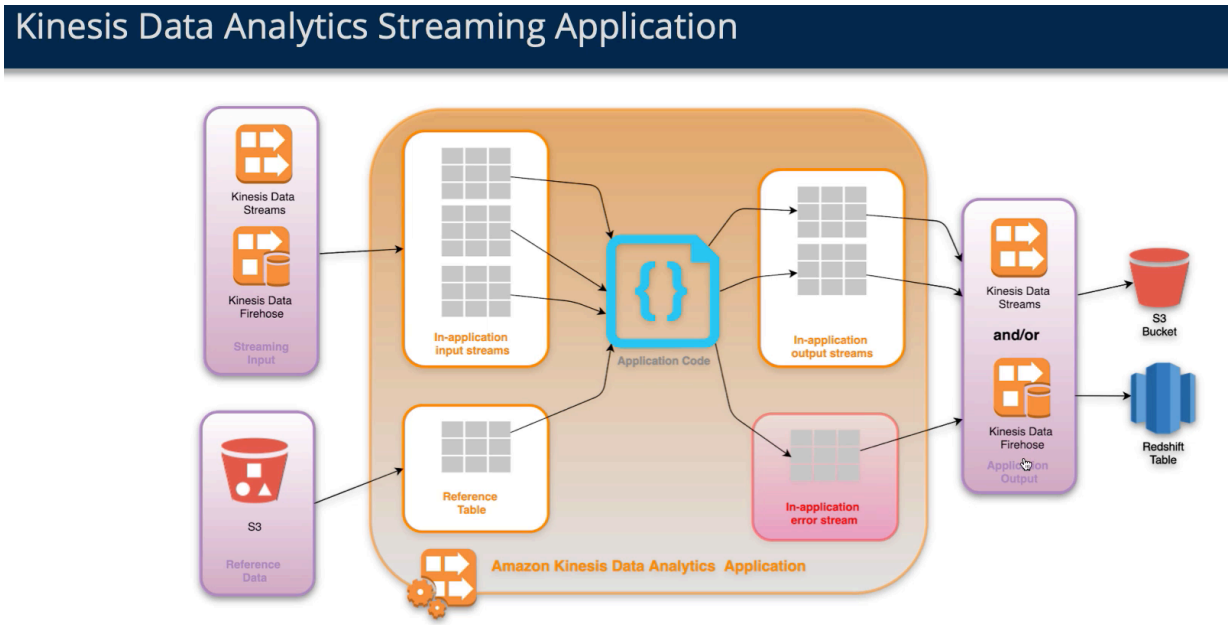
- ❏ Streaming application is the primary resource in Kinesis Data Analytics
- ❏ Create/manage streaming applications via the console or the Kinesis Data Analytics API
- ❏ The API has operations to manage your streaming applications
- ❏ A Kinesis Data Analytics streaming application continually read and process streaming data in real-time
- ❏ Application code in SQL processes the incoming streaming data and produces output
- ❏ Kinesis Data Analytics writes the output to configured destinations

- ❏ Streaming data input from Kinesis Data Streams or Kinesis Data Firehose
- ❏ Reference data from S3
- ❏ Data read continuously from the streaming data sources but one time from the reference source
- ❏ Reference data sources are used for joining against the incoming stream to enrich the data

Example:
In application input stream read continuously from Firehose or Stream.
Reference table is read just once

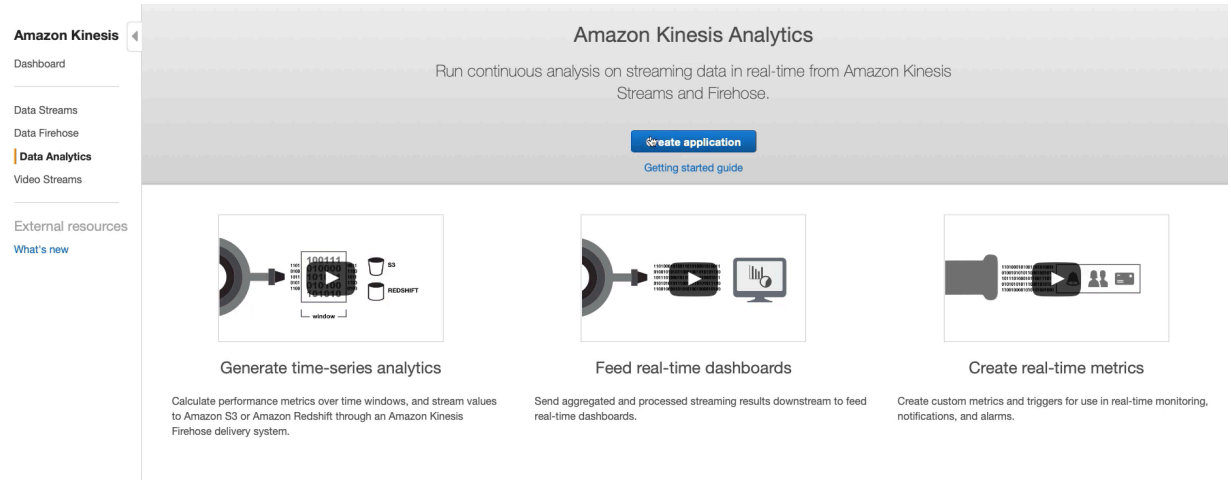
Application writes to: in-application output stream and errors to in-application error stream

The in-application output streams then writes the output to 1-4 destinations, that can be reached via a Kinesis Data Streams and/or Firehose => 1-4 S3/Redshift destinations



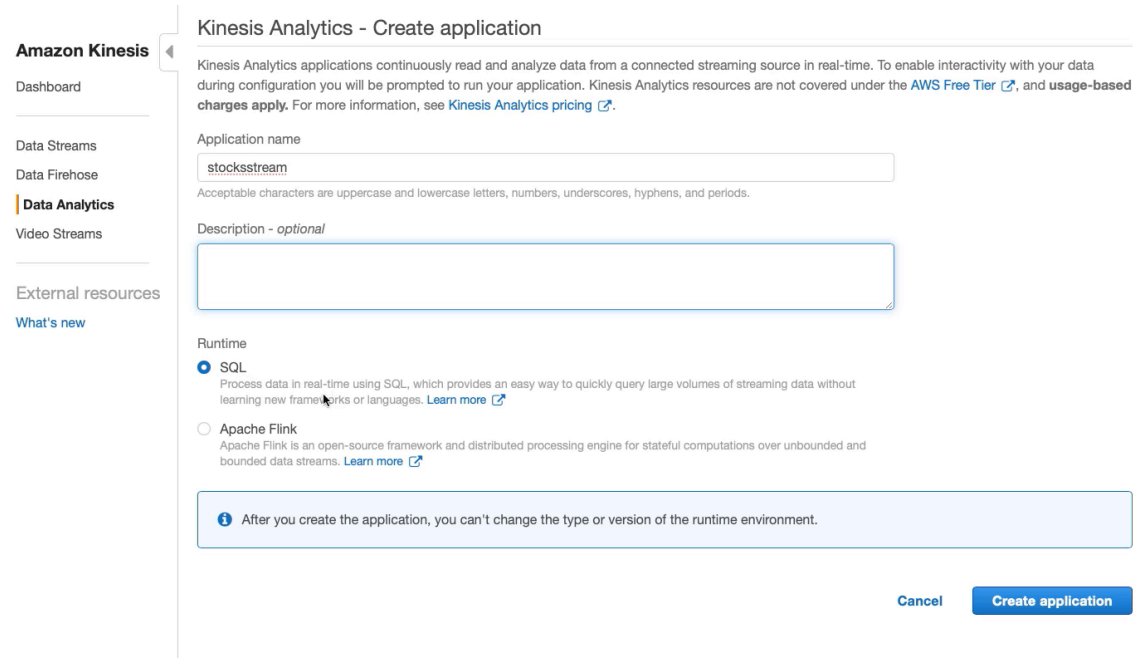
This is typically used to engineer data from sources on-the-fly, in real time.

Kinesis Data Analytics - Lab

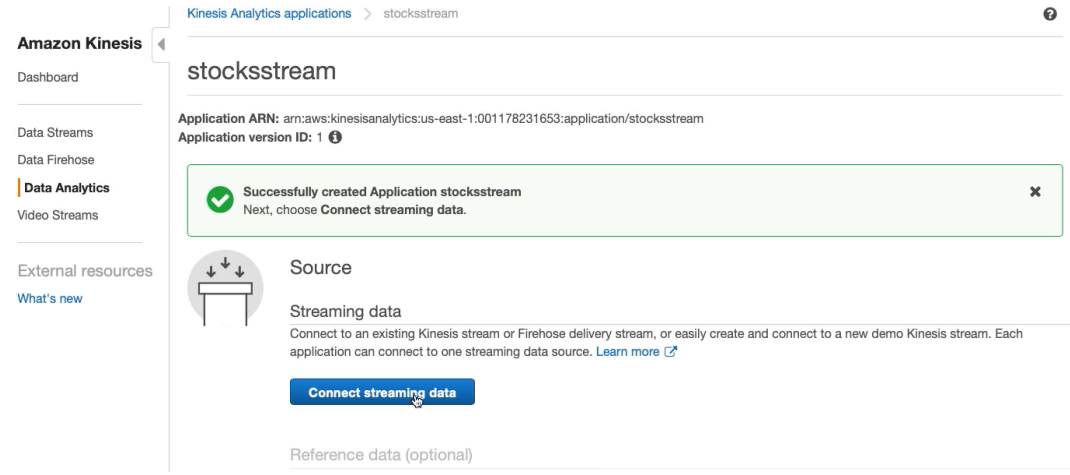


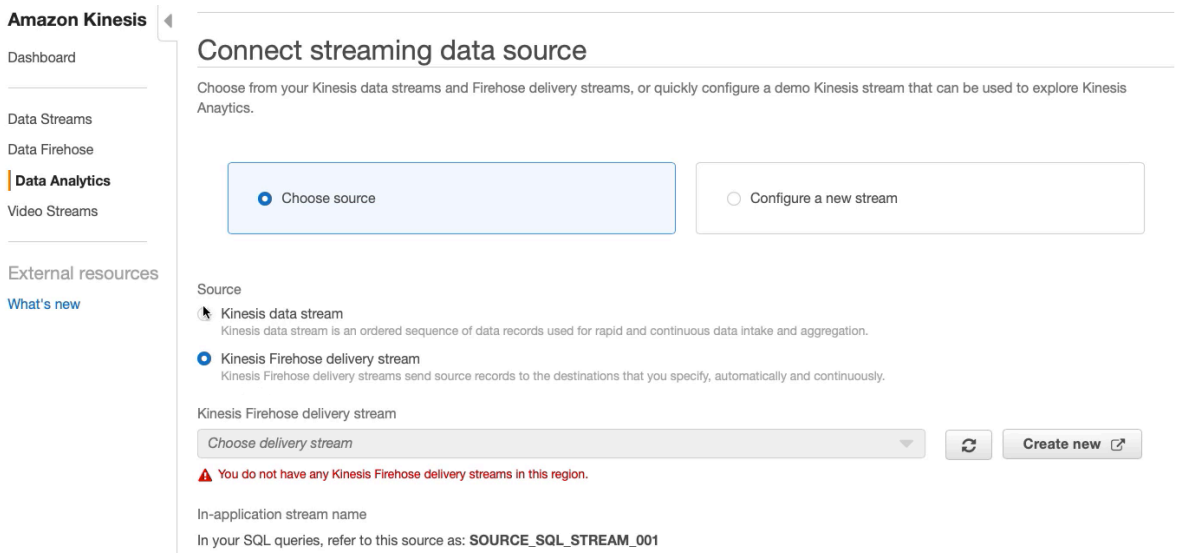
Crete Application
We will use the stock streaming data stream

We can run a **SQL** or **Apache Flink** to transform the data

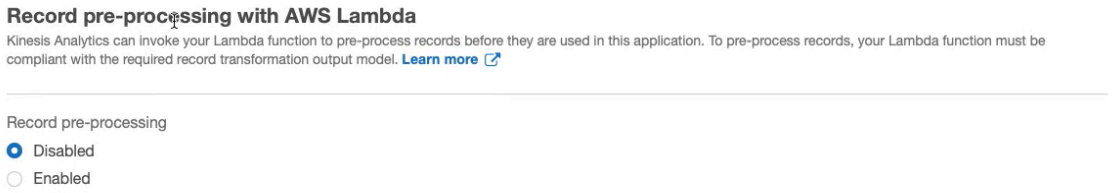


Connect streaming data:

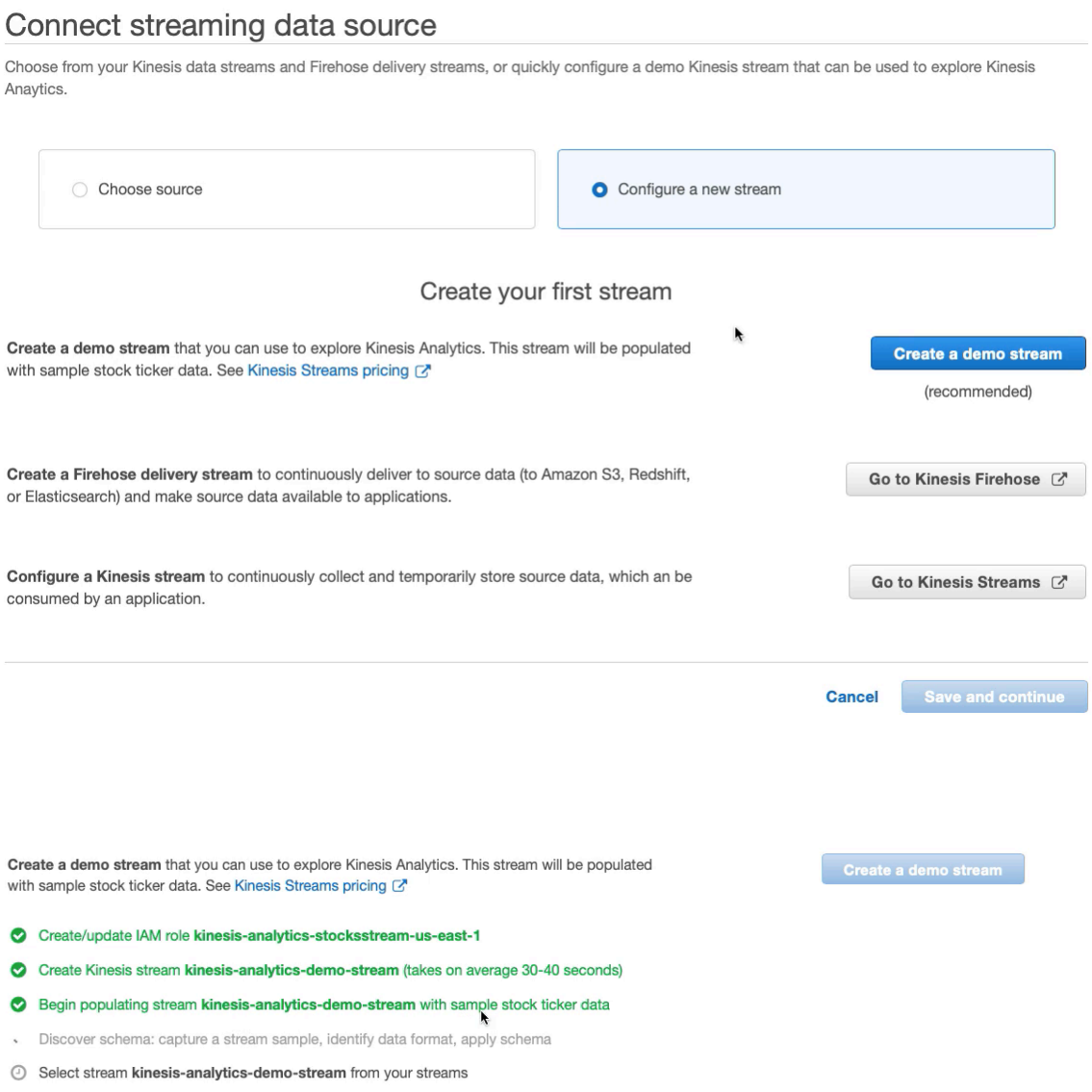




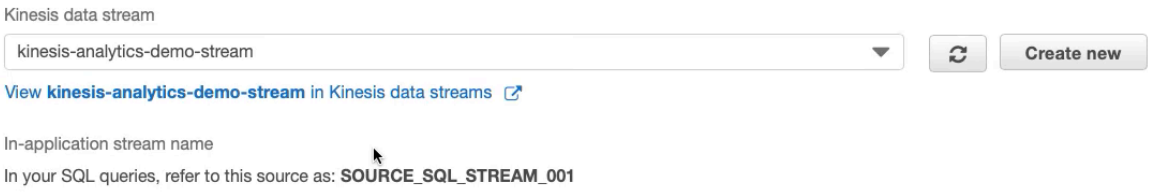
To note, We can do record pre-processing with Lambda



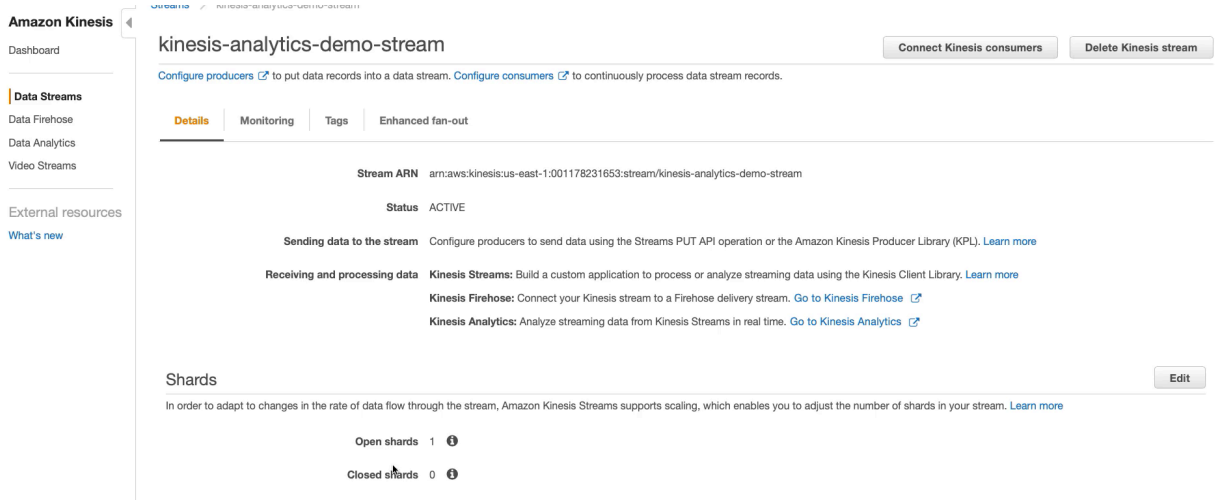
We are going to configure a new stream

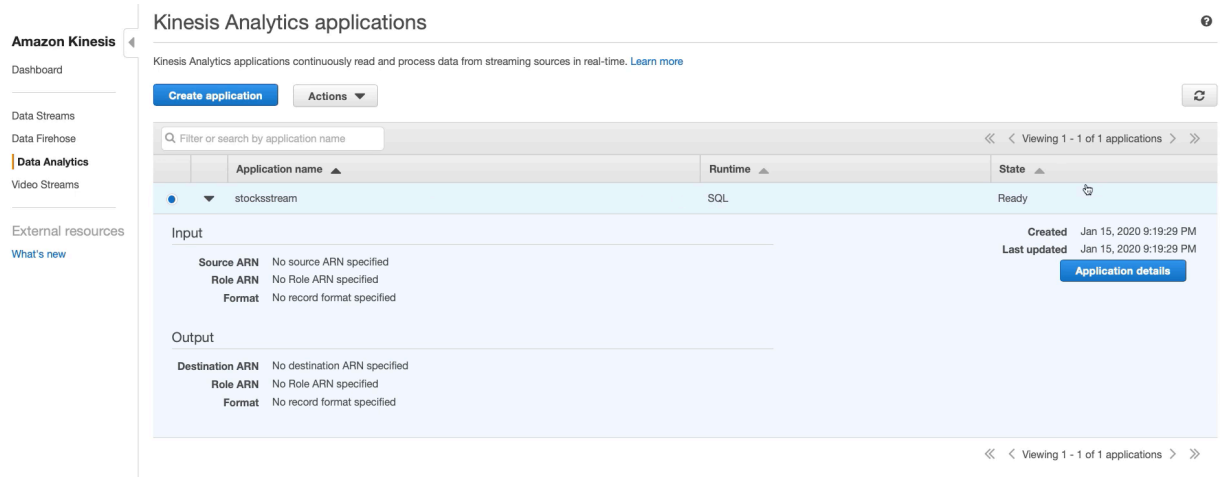


We can now select the newly created stock stream

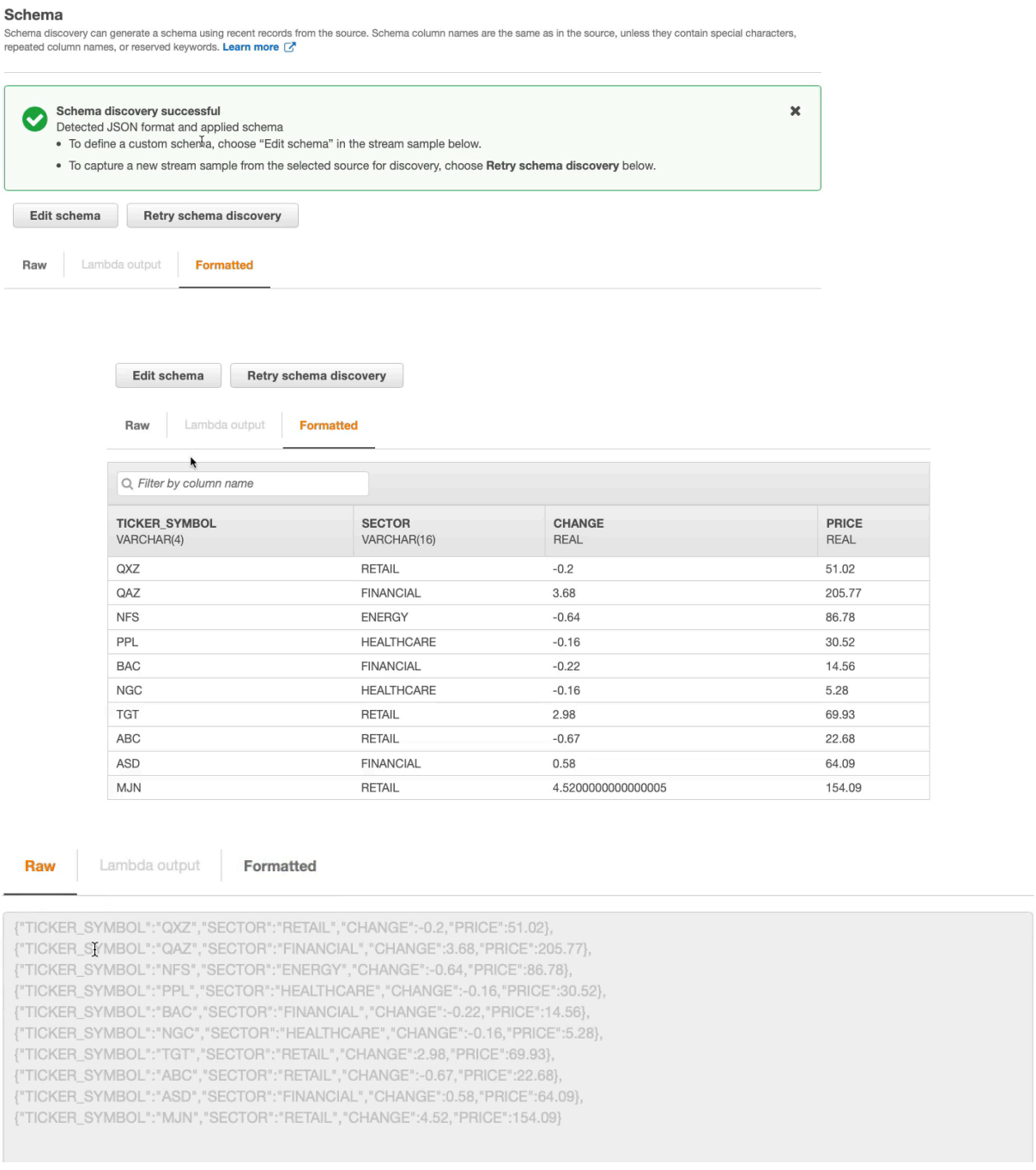


It has just 1 shard



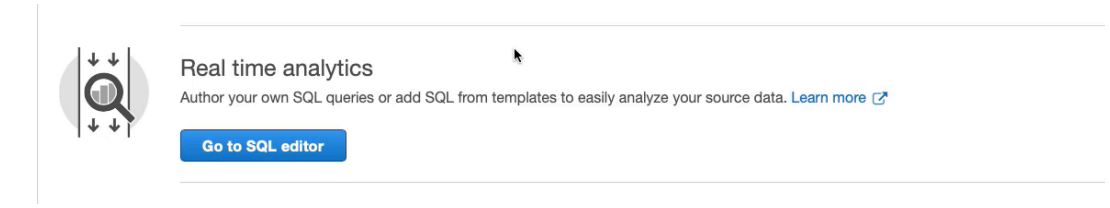


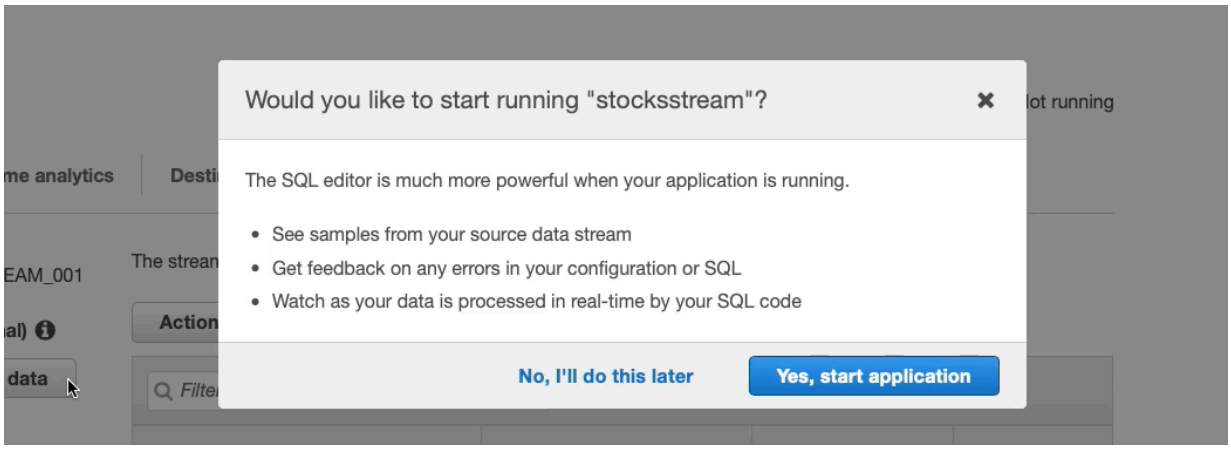
We now discover the schema

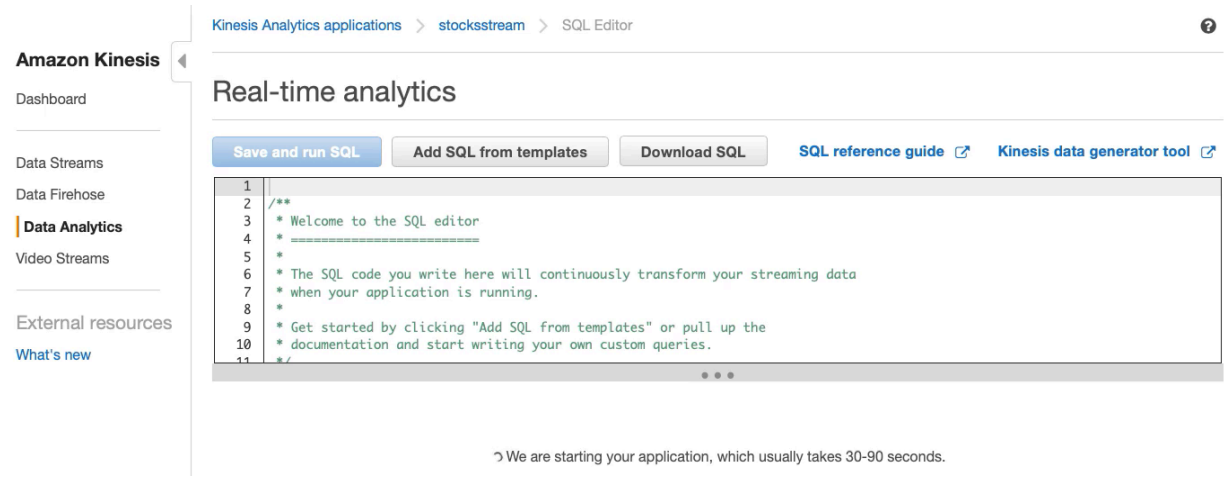


To note, we could also add a lambda to update that ticket symbol.

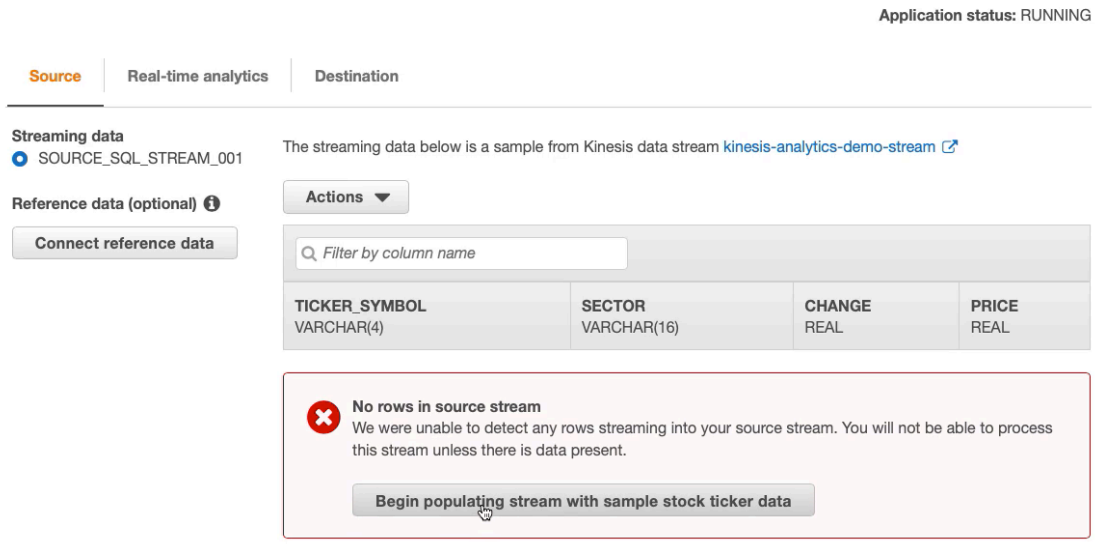
Now we go too the SQL Editor



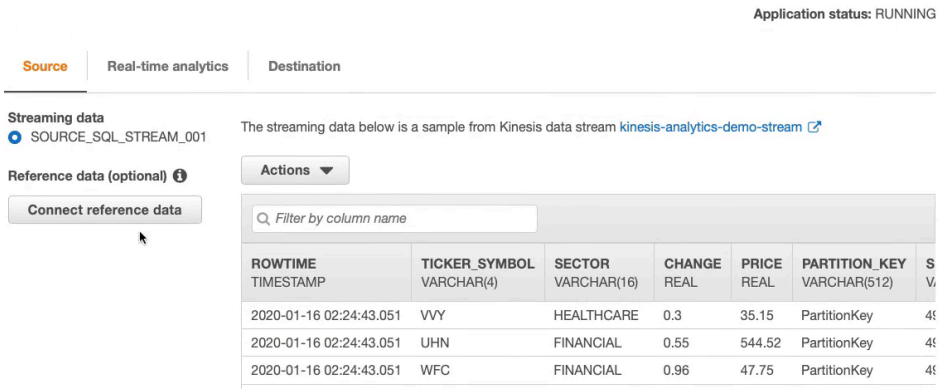




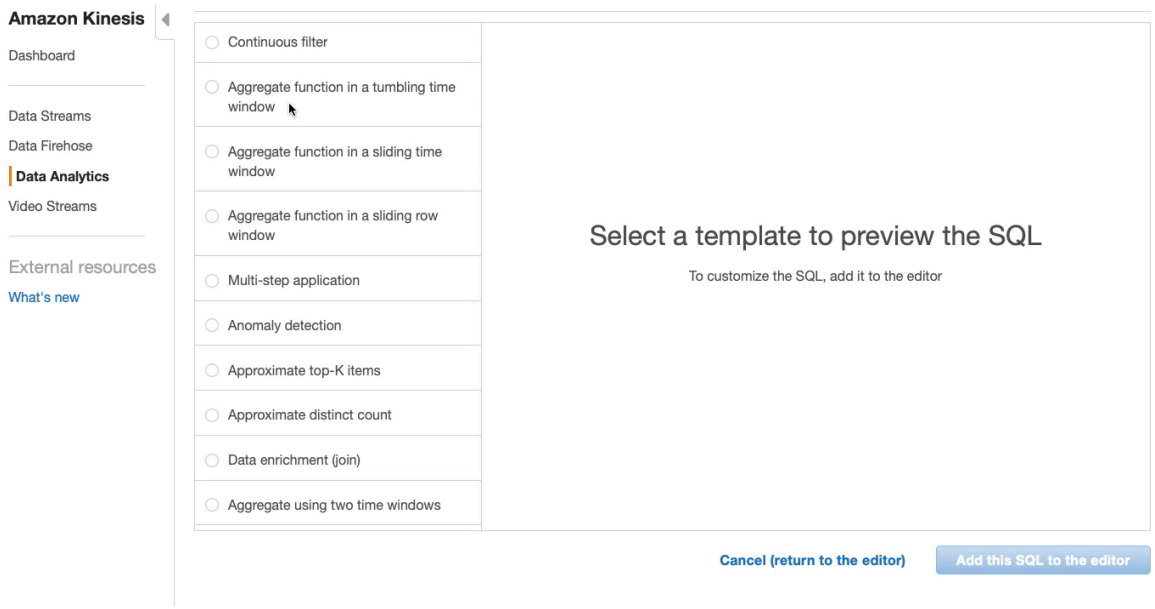
Application has now started

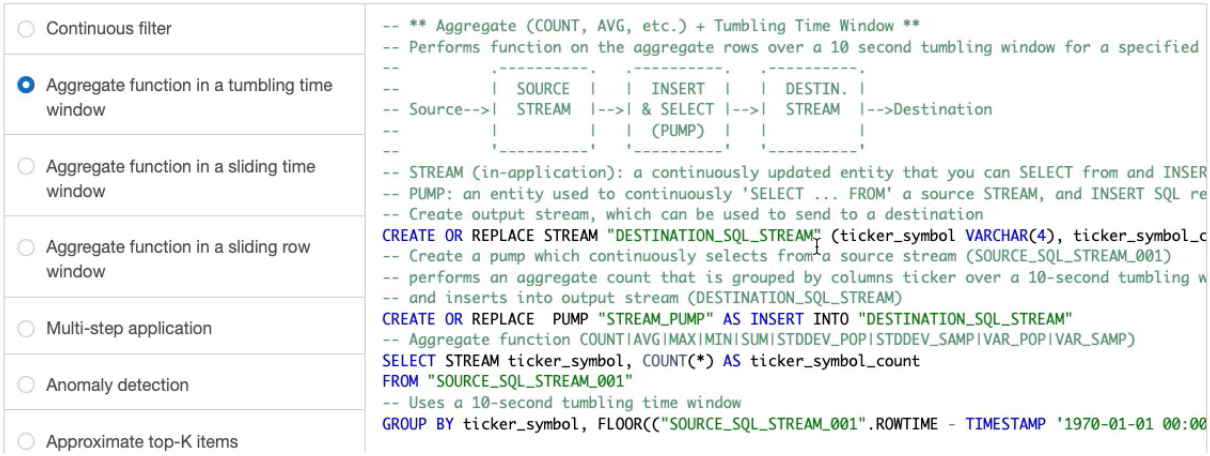


Populate it



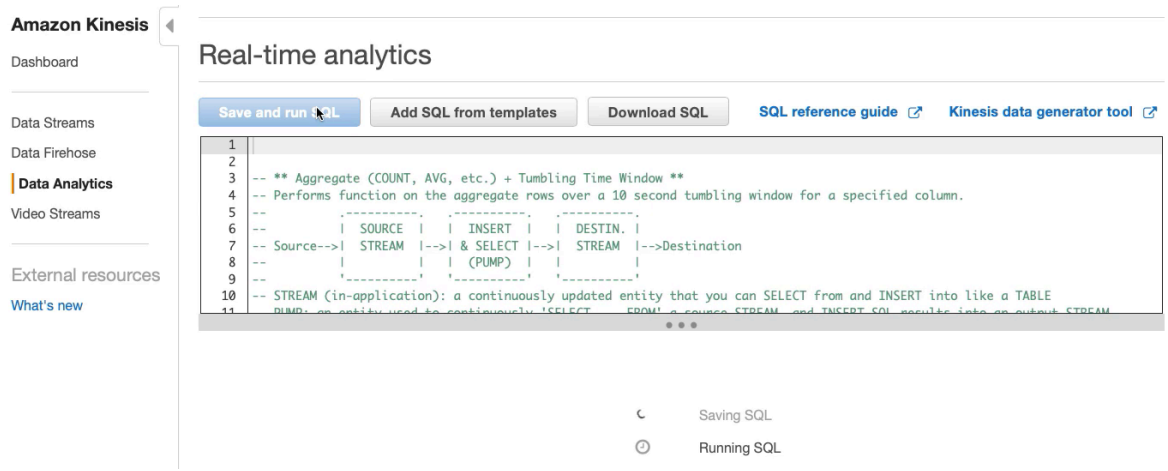
Add SQL from template



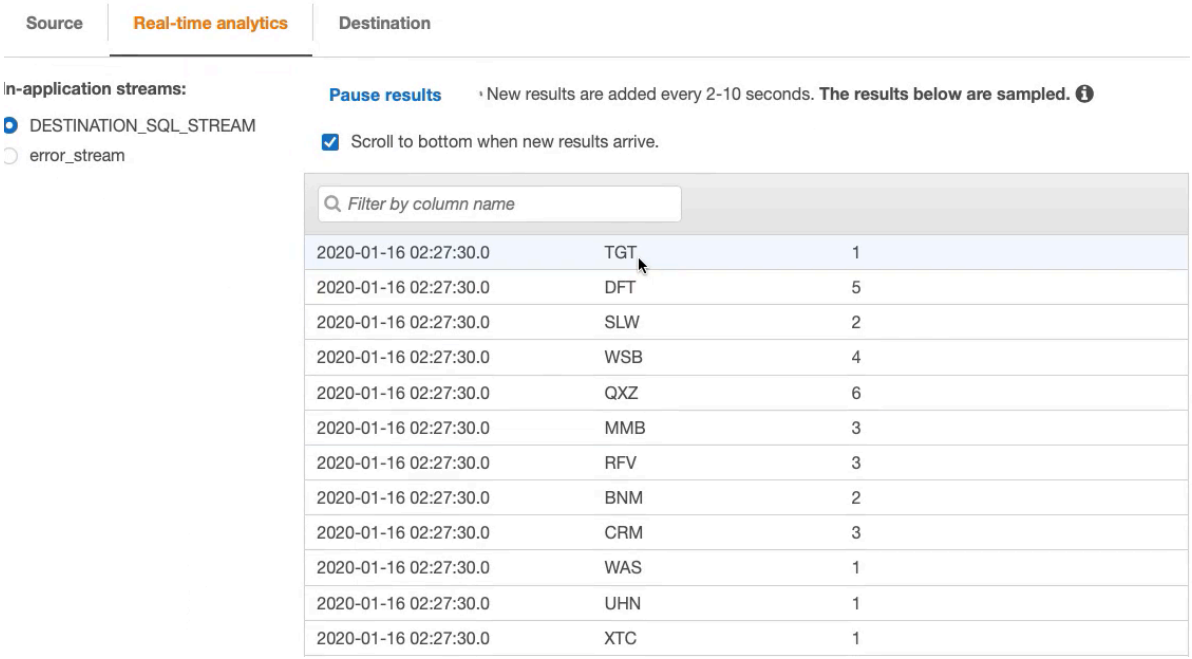


=> see how many ticket symbols come in a tumbling time window (10 seconds)

We can now run and save the SQL



We can now see real-time analytics



Finally, add a Destination to store this aggregated streaming data
=> Use Firehose to persist to S3 so SageMaker can consume

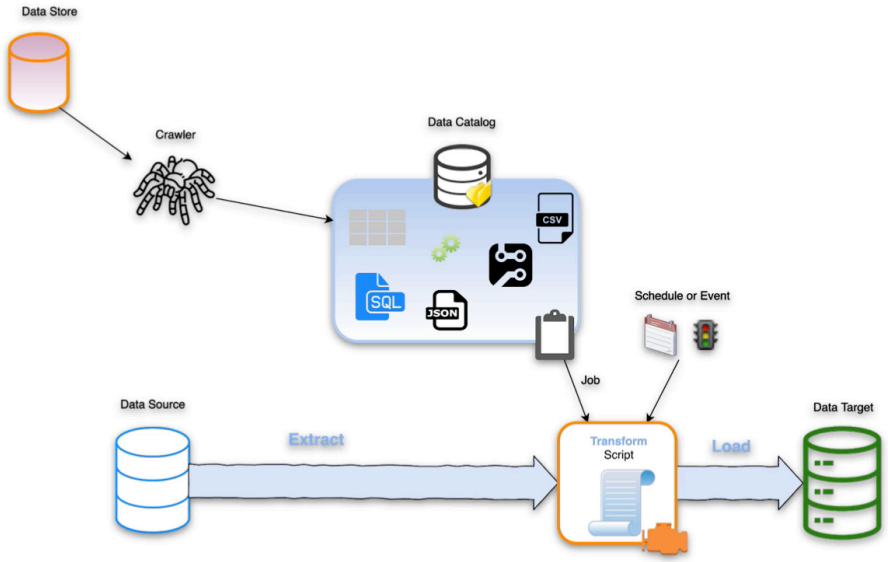
AWS Glue

To perform ETL work from a data source to a data target
- points crawler to data store
- crawler populates Glue data catalog with metadata table definition

- AWS Glue can generate a script to transform data or we can use the console.
- we can run the job on demand or on trigger (time based, schedule, trigger event)
- the script runs in an Apache Spark environment

AWS Machine Learning Exploratory Data Analysis

AWS Glue Concepts



AWS Glue Key Facts

- ❏ A fully managed ETL service for categorizing, cleaning, enriching, and moving your data
- ❏ Glue components
  - ❏ Central metadata repository: Glue Catalog
  - ❏ ETL engine that automatically generates python or scala code
  - ❏ Flexible scheduler for dependency resolution, job monitoring, and retries
- ❏ Serverless
- ❏ Can convert semi-structured schemas to relational-schemas on the fly

AWS Glue Terminology

- ❏ Data Catalog: persistent metadata store
- ❏ Classifier: determines the schema of your data
- ❏ Connection: the properties required to connect to data store
- ❏ Crawler: connects to a data store and steps through prioritized list of classifiers to determine schema
- ❏ Database: set of associated data catalog table definitions
- ❏ Data store: repository for persistently storing data
- ❏ Data source: data store used as input to transformation
- ❏ Data target: data store that a transformation writes to
- ❏ Job: ETL logic
- ❏ Table: metadata definition that represents your data
- ❏ Transform: code logic to change your data into a different format

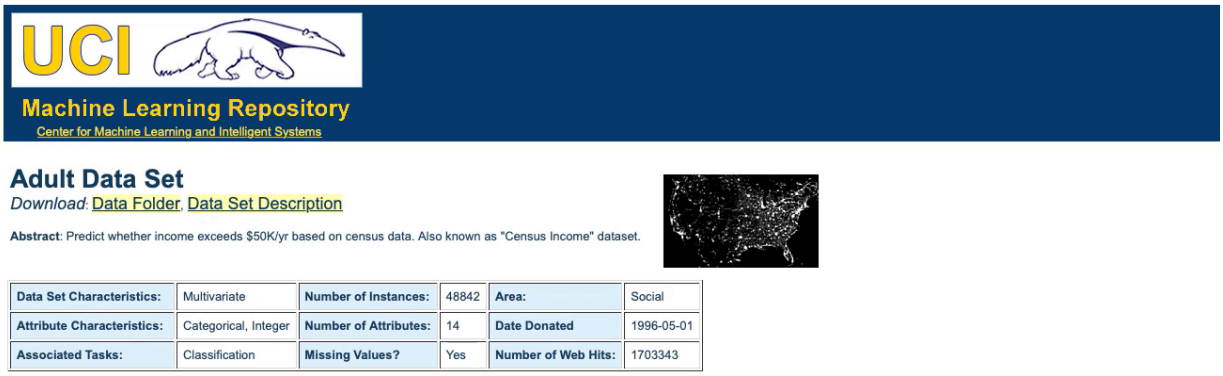
AWS Glue Components

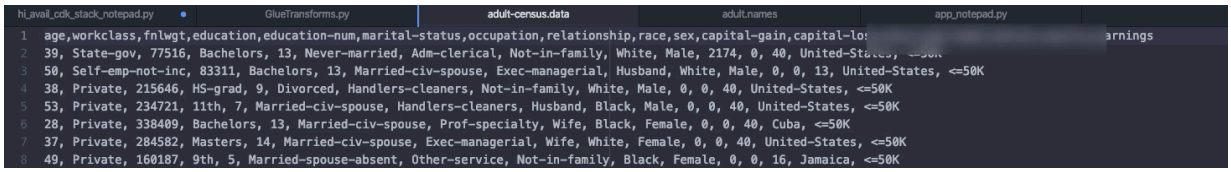
- ❏ Console: define and orchestrate ETL workflows
- ❏ Data Catalog: persistent metadata store
- ❏ Crawlers and Classifiers: crawlers scan data and classify it
- ❏ ETL Operations: using metadata in the data catalog, autogenerates python or scala code
- ❏ Jobs System: managed infrastructure to orchestrate your ETL workflow

AWS Glue - Lab

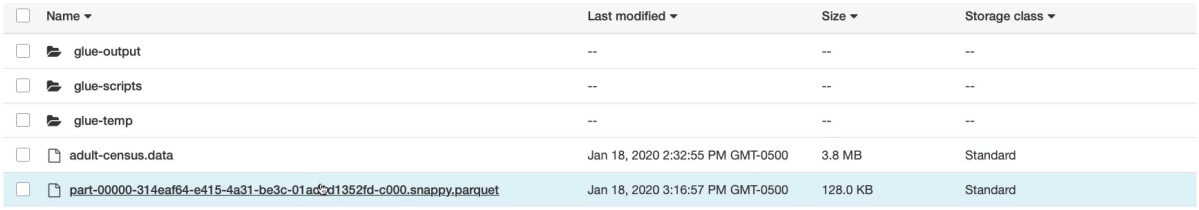
Take data from 1 format (parquet) and transform it to CSV

Using adult data set from census data

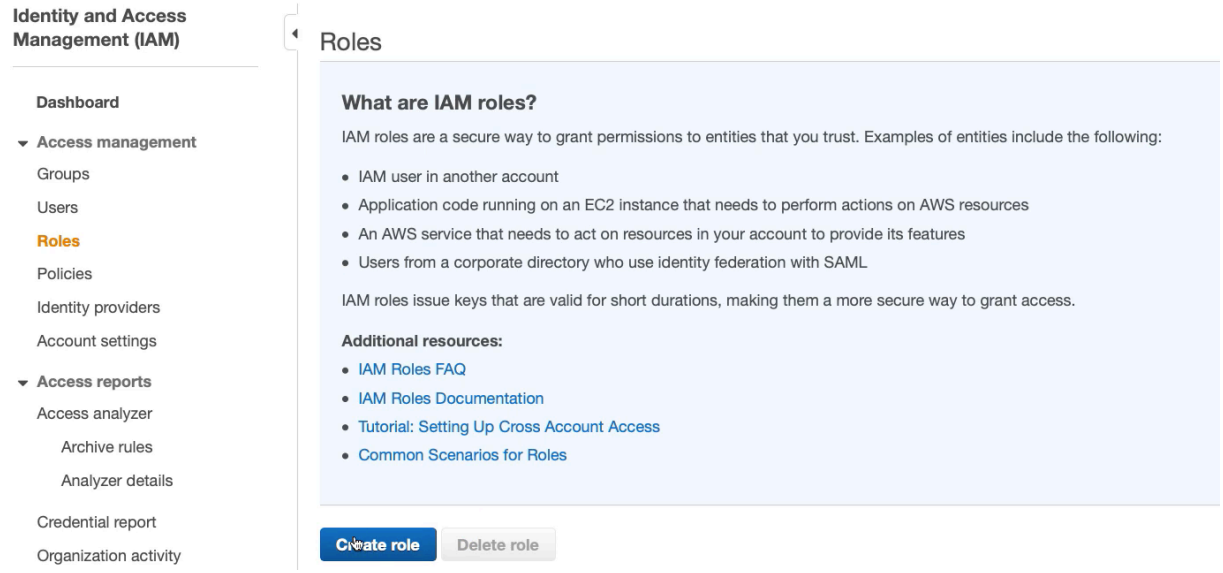




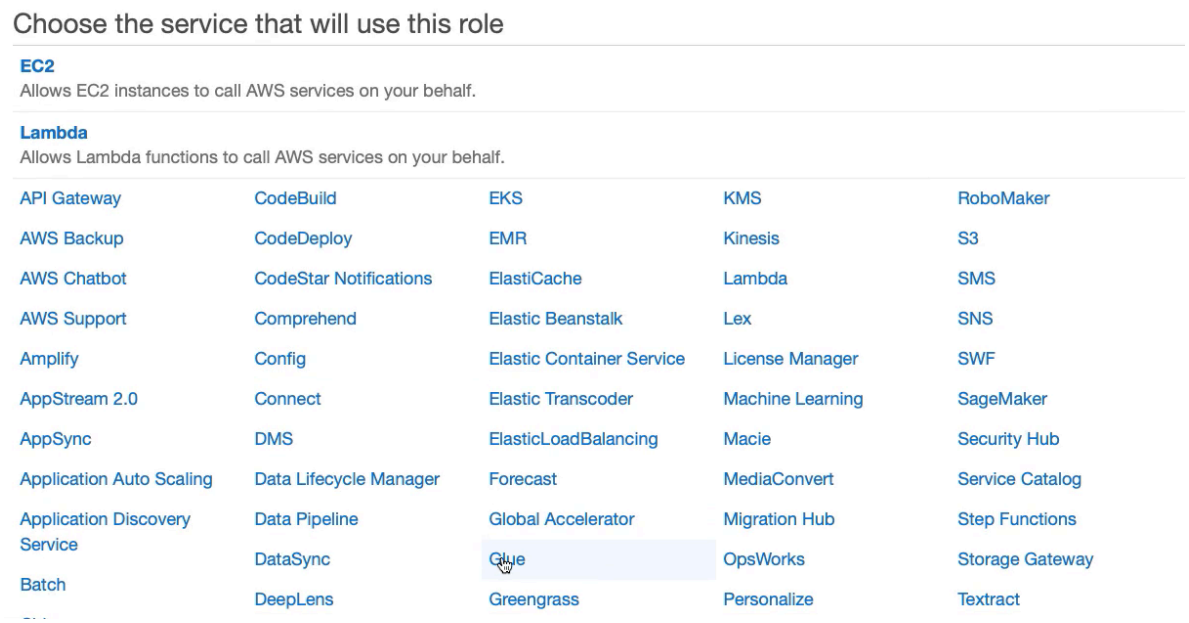
Data is available in S3 as parquet file



First create an IAM role or Glue

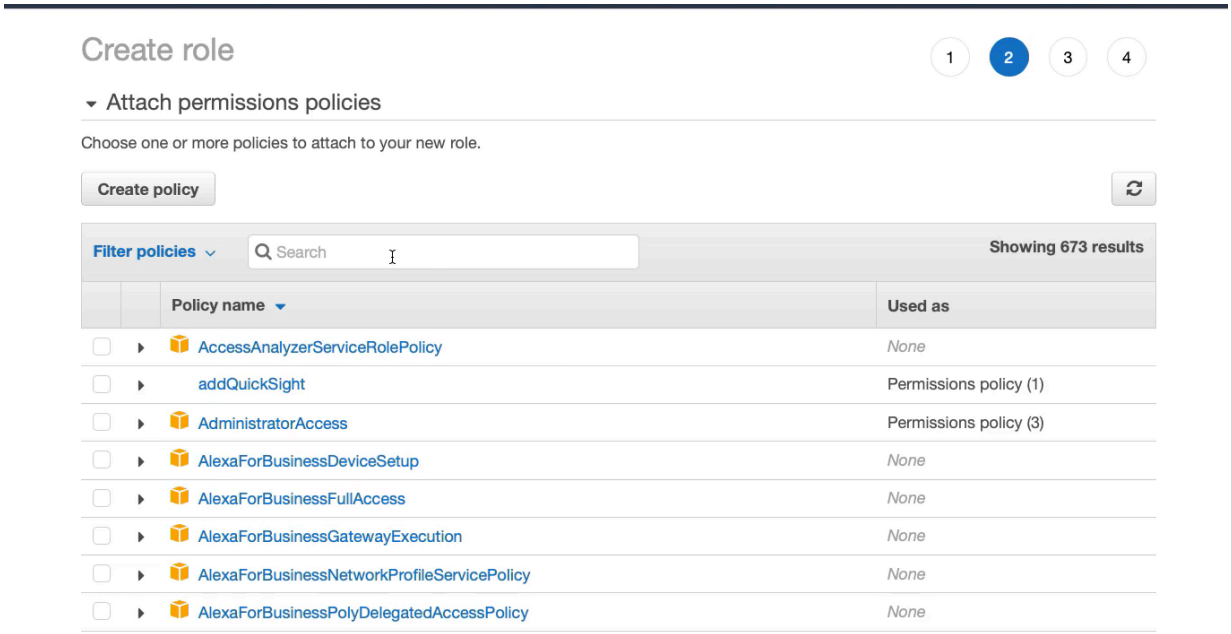


Use Glue Service

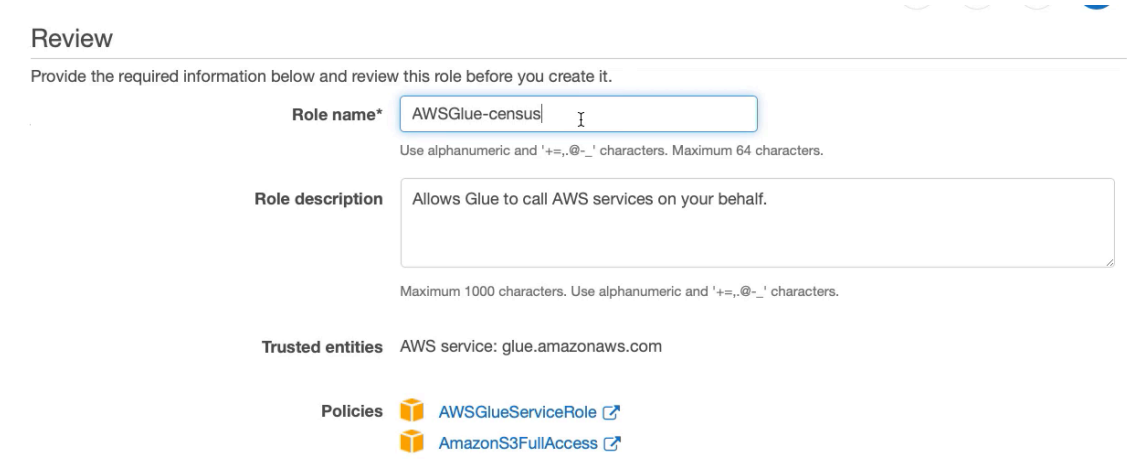


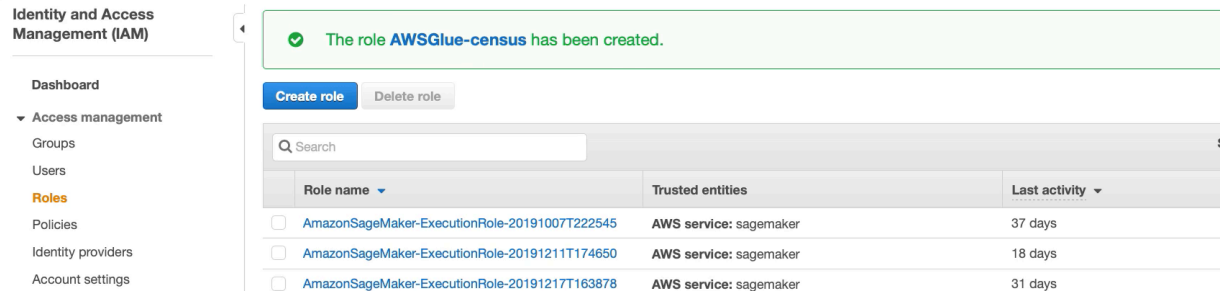
Then go to permissions

Attach 2 managed policies to the role:
- AWSGlueServiceRole
- AmazonS3FullAccess

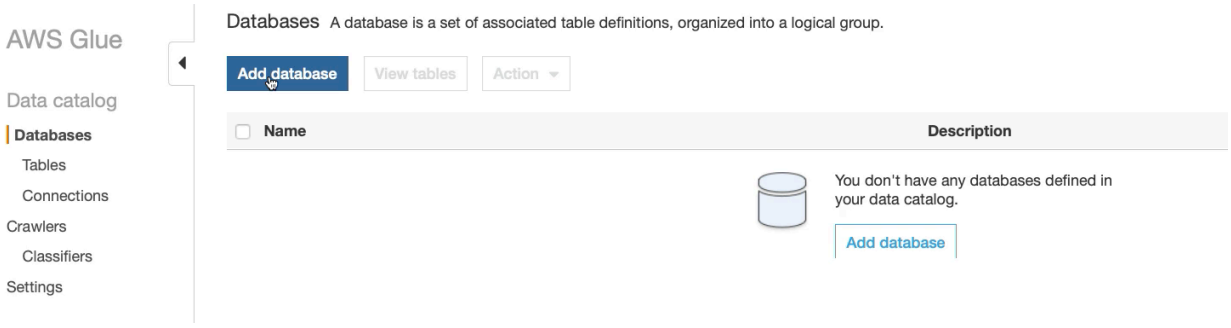


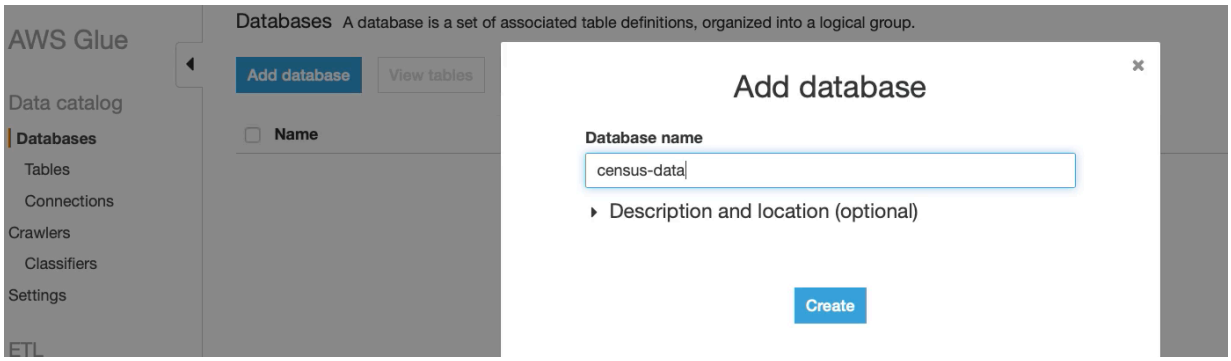
Let’s give it a name



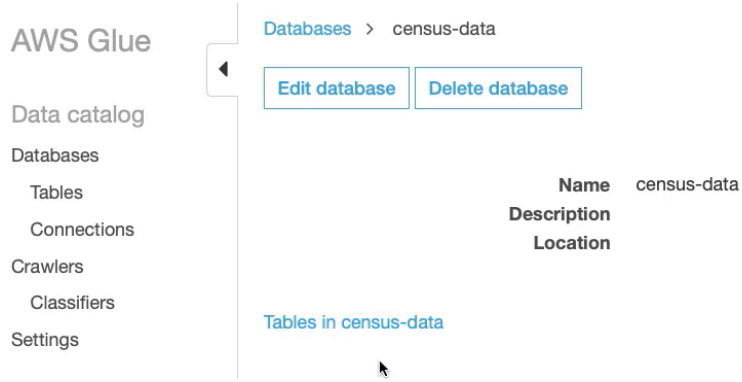


Now add a database - where all the metadata about the megastore will be housed

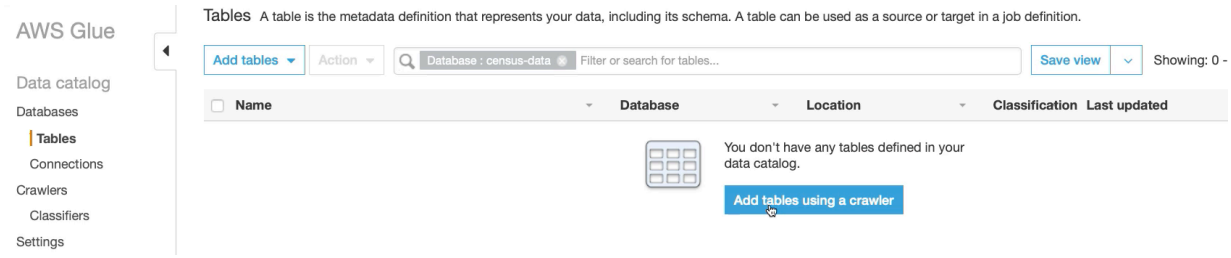


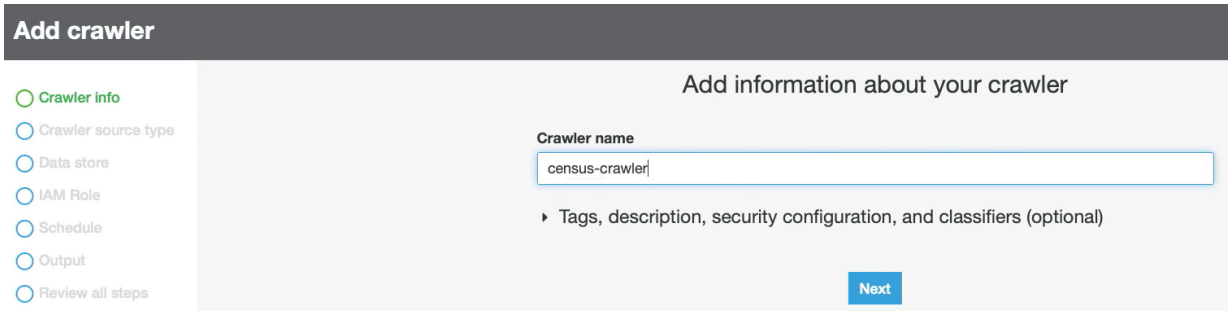


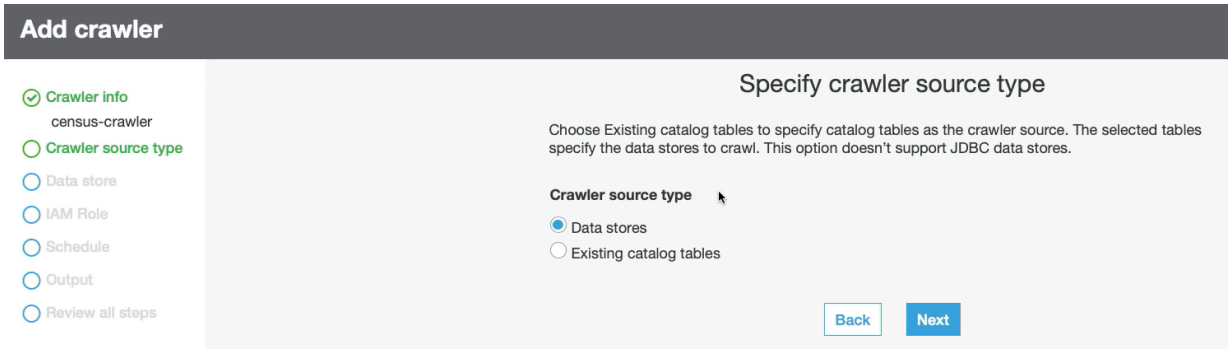
The census-data database has no table

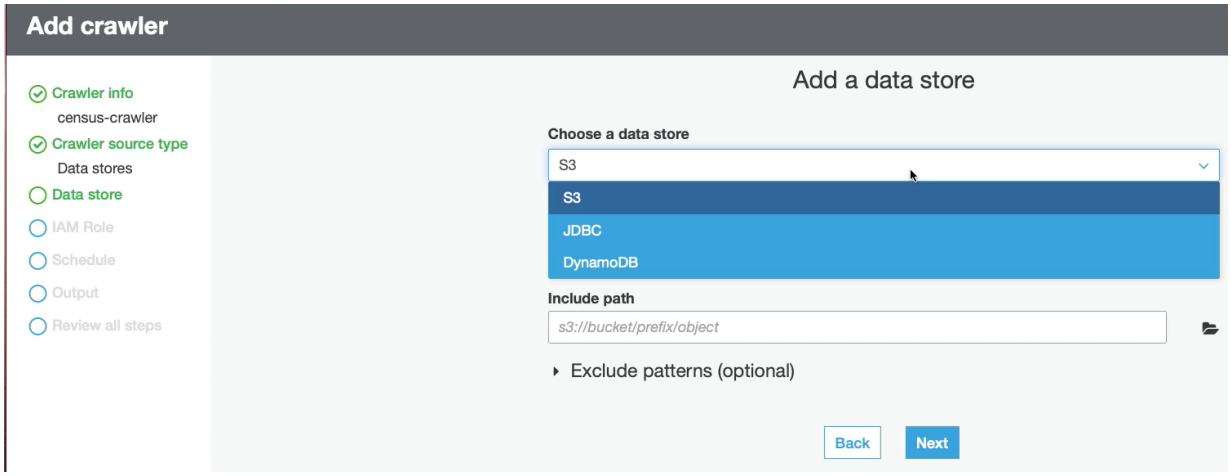


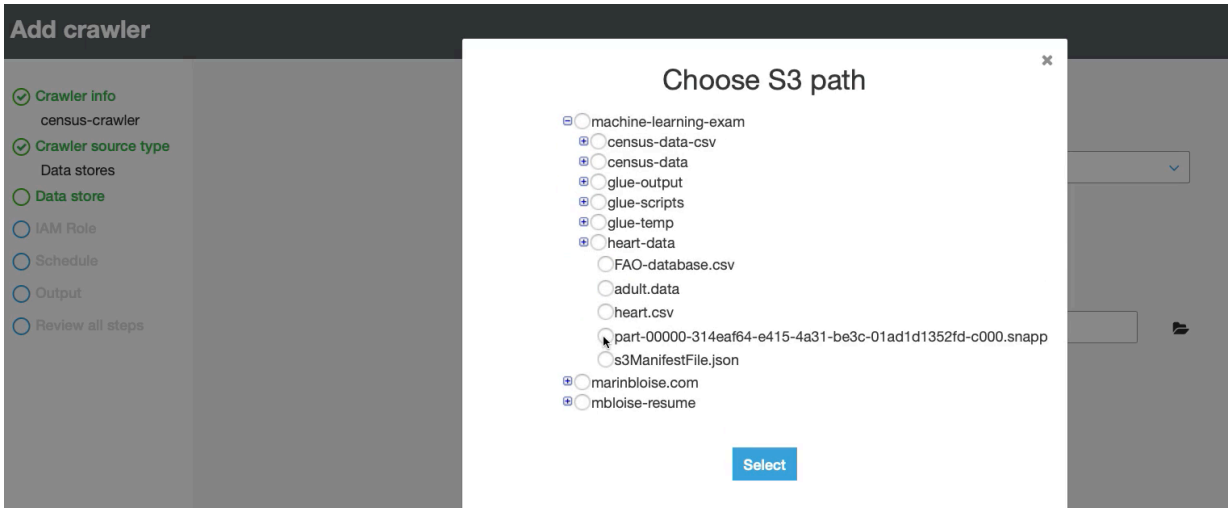
Let's create a table using a crawler



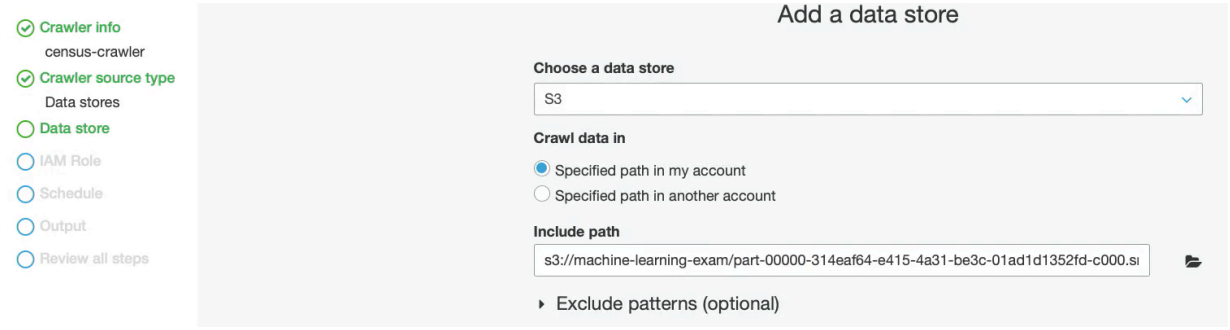




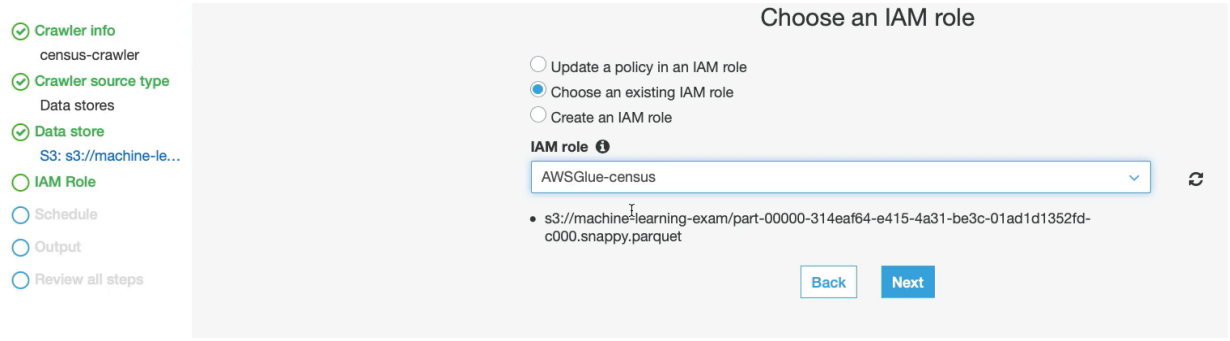




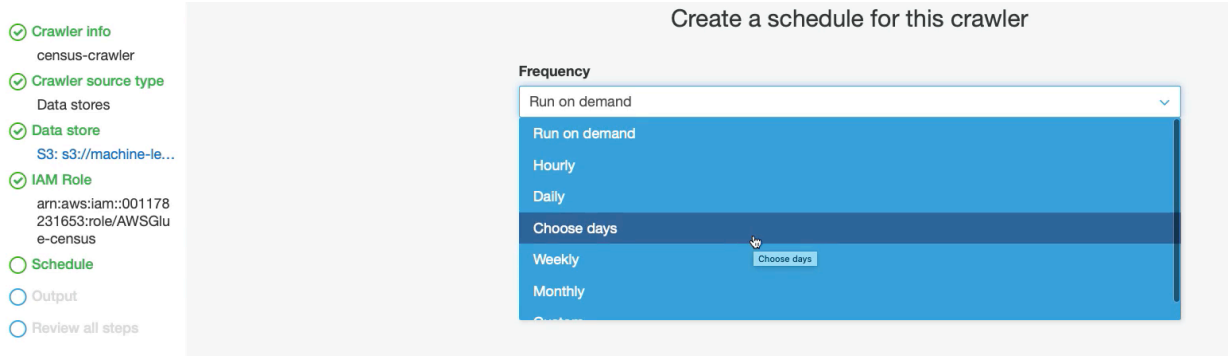
We now have the source set up



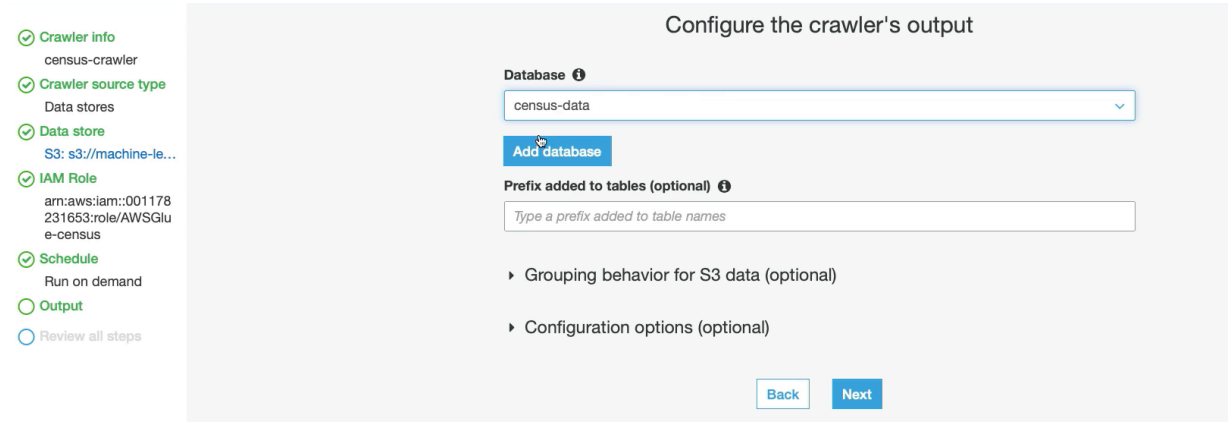
Select the role we created earlier to access S3 and other glue services



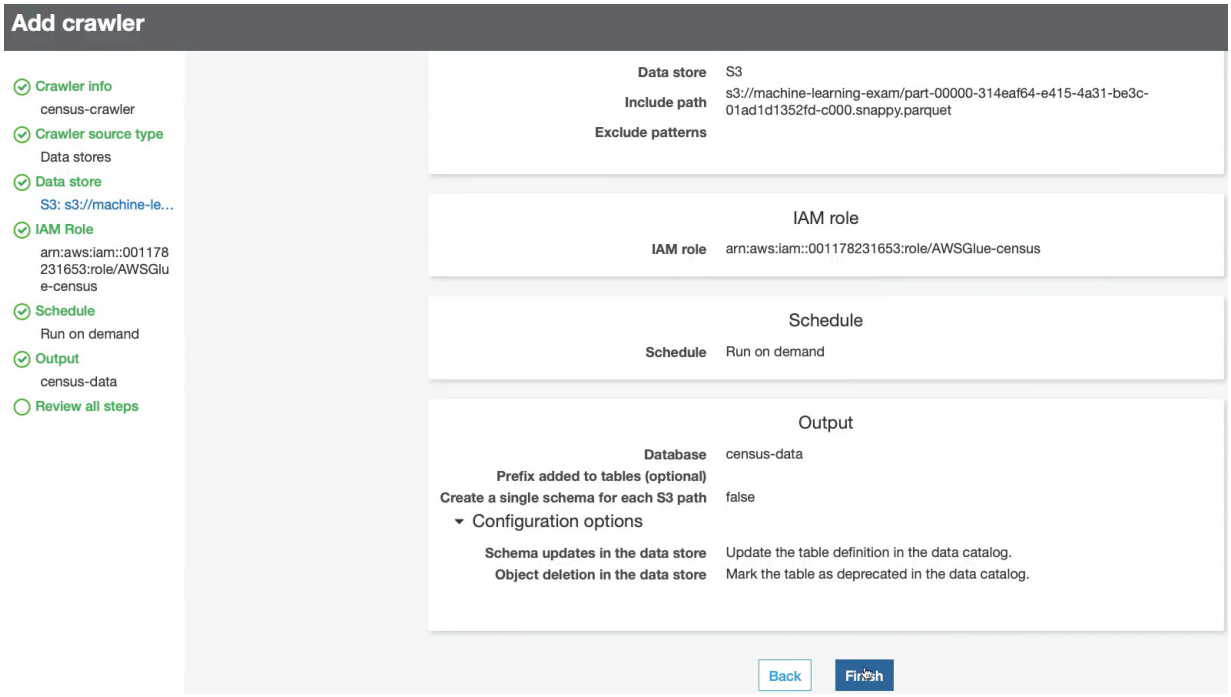
Schedule: run it on demand



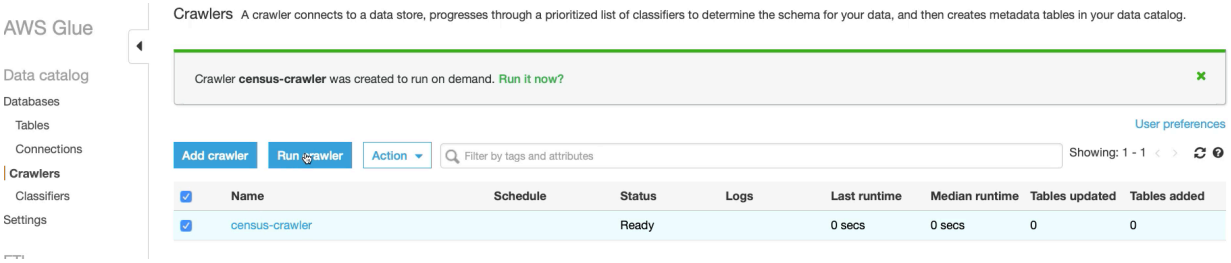
Select the database where to store the metadata



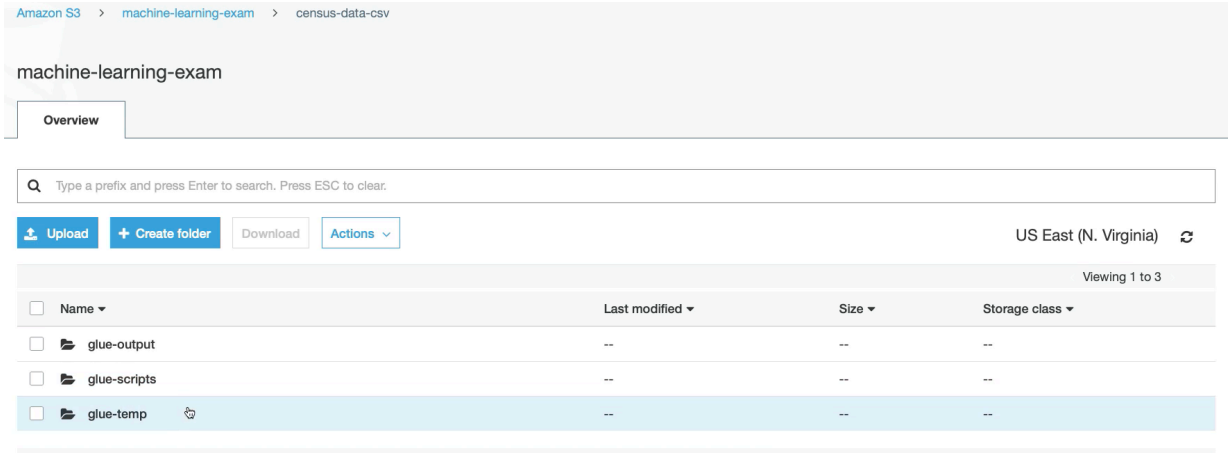
Finish



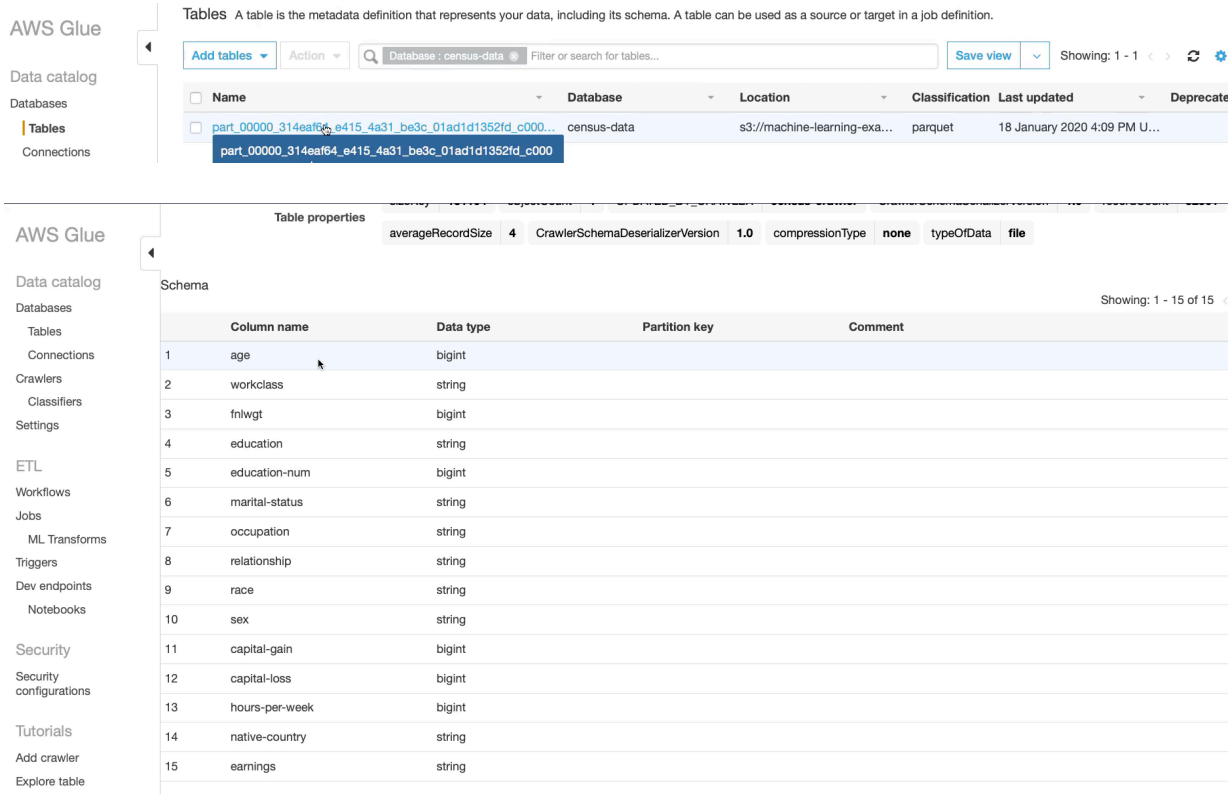
Now run the crawler to infer the schema



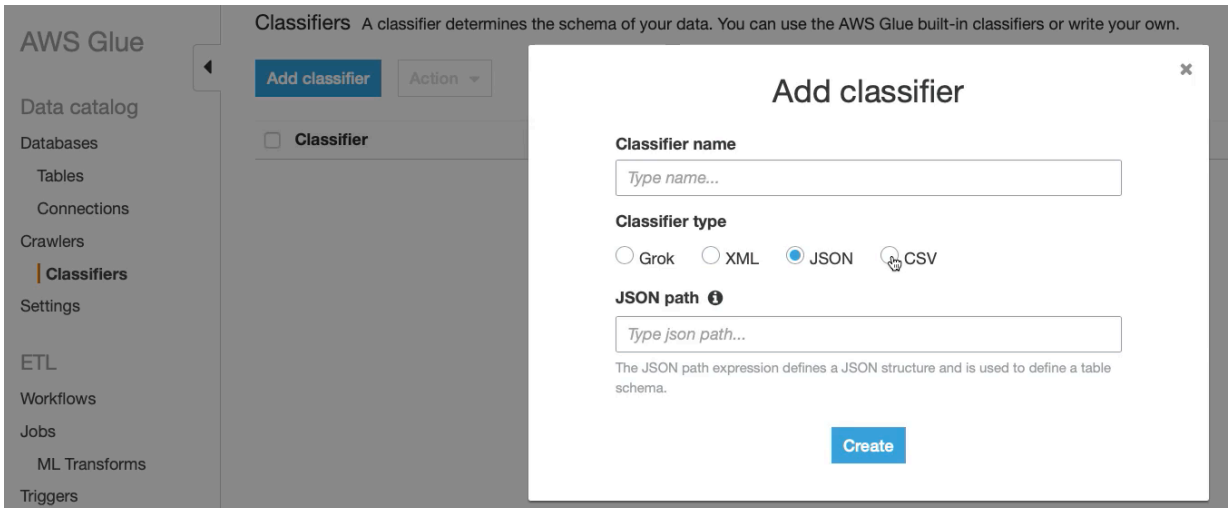
Create a crawler to crawl that
Best practices for S3 to store temporary files and output the results of glue job:



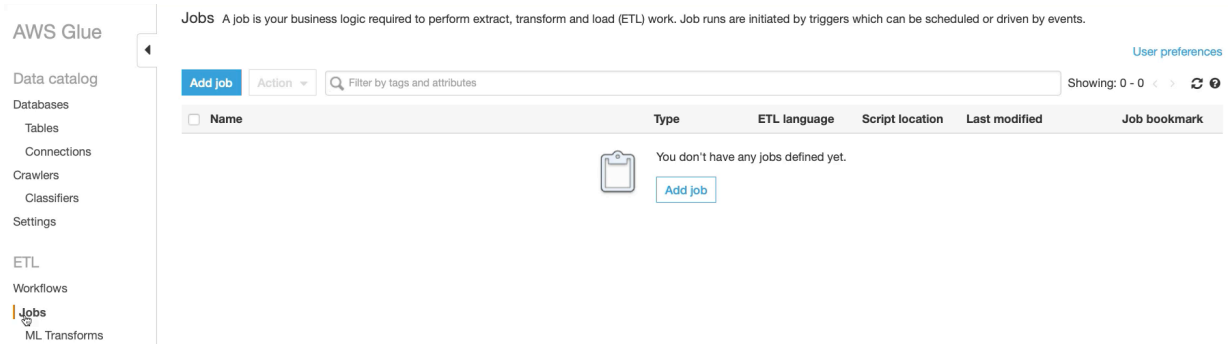
A table was created by our crawler

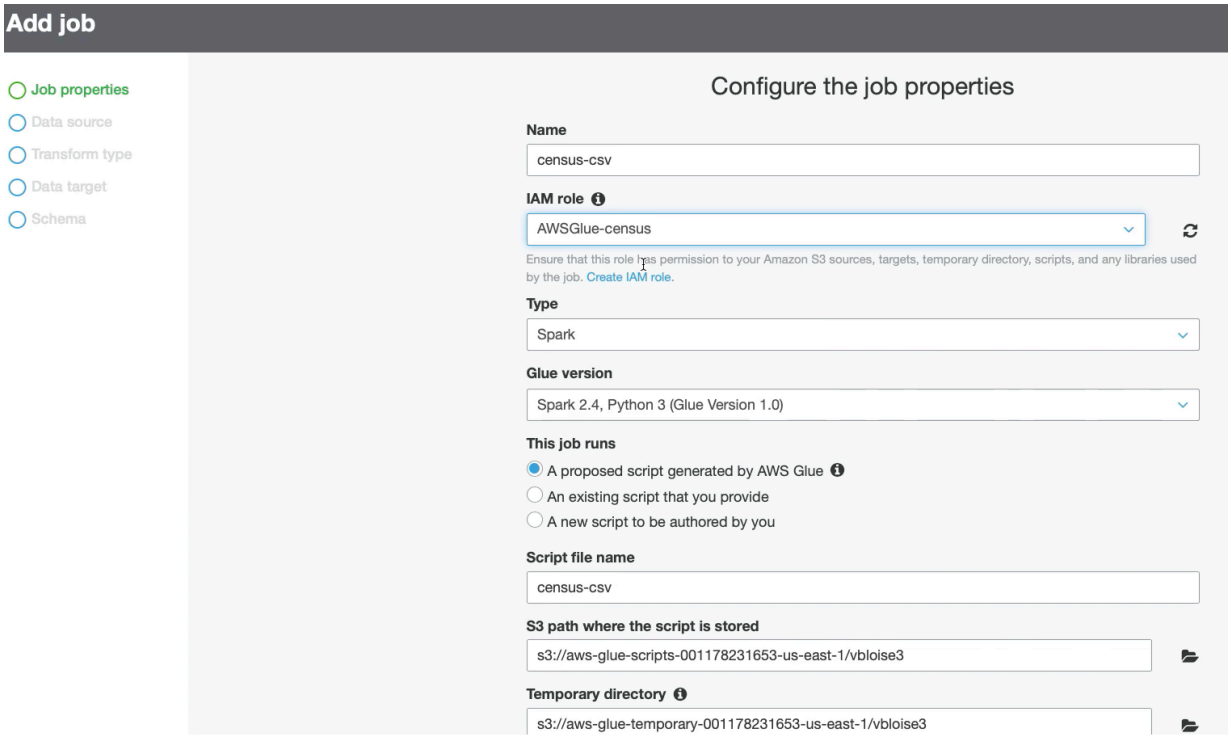


Classifiers:

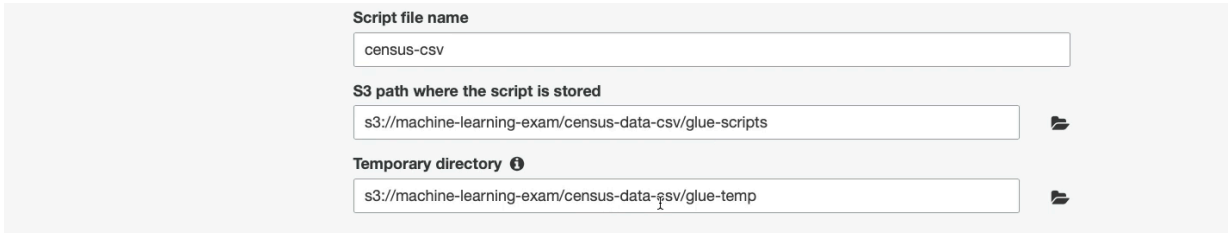


Now create a job to transform our data from parquet to CSV

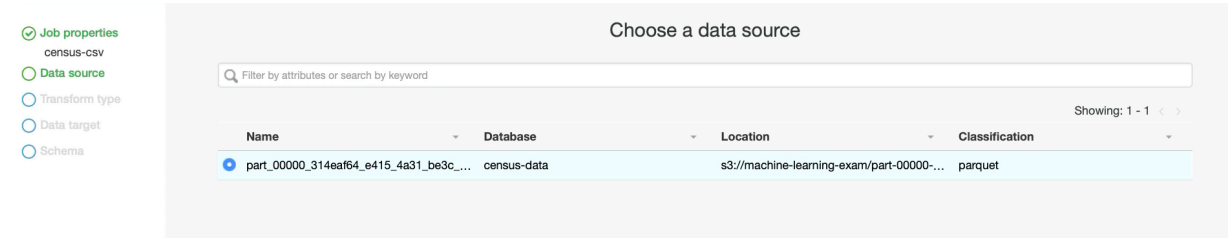




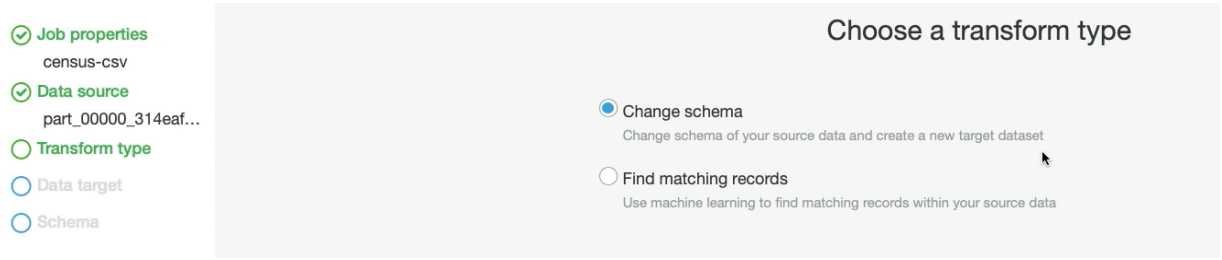
Update the 2 target directories



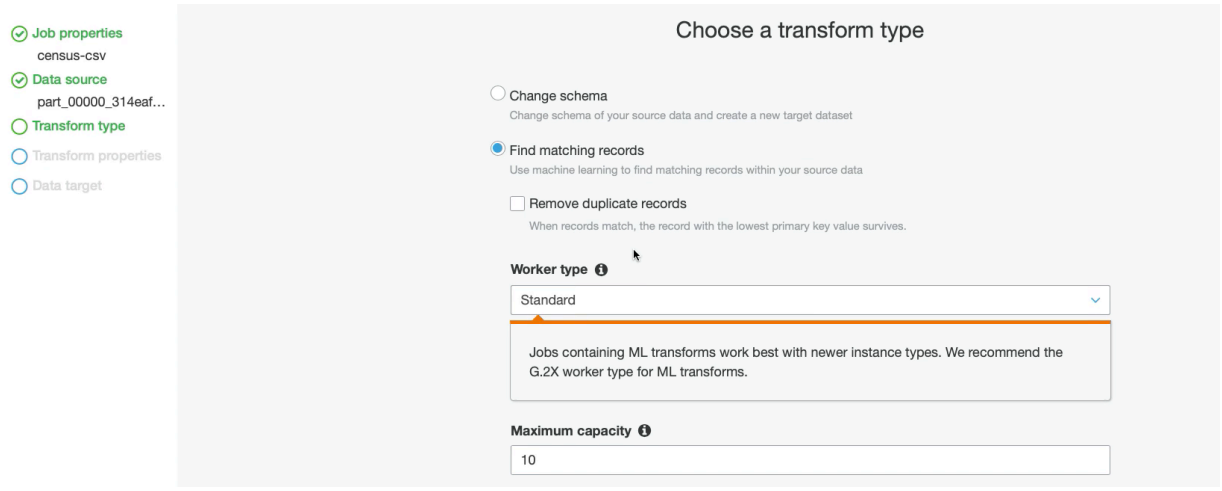
Pick data source: parquet file



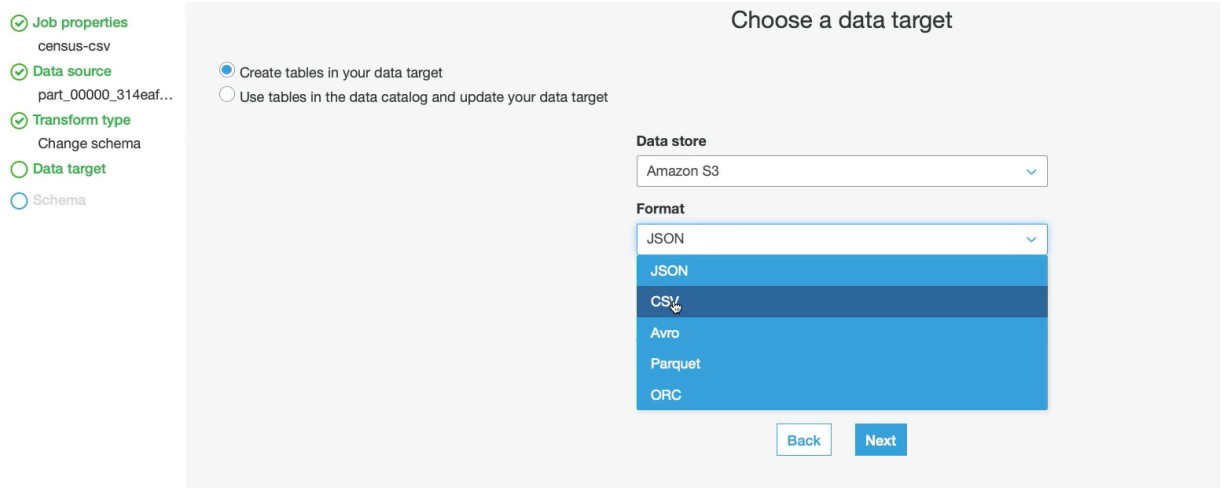
Transform

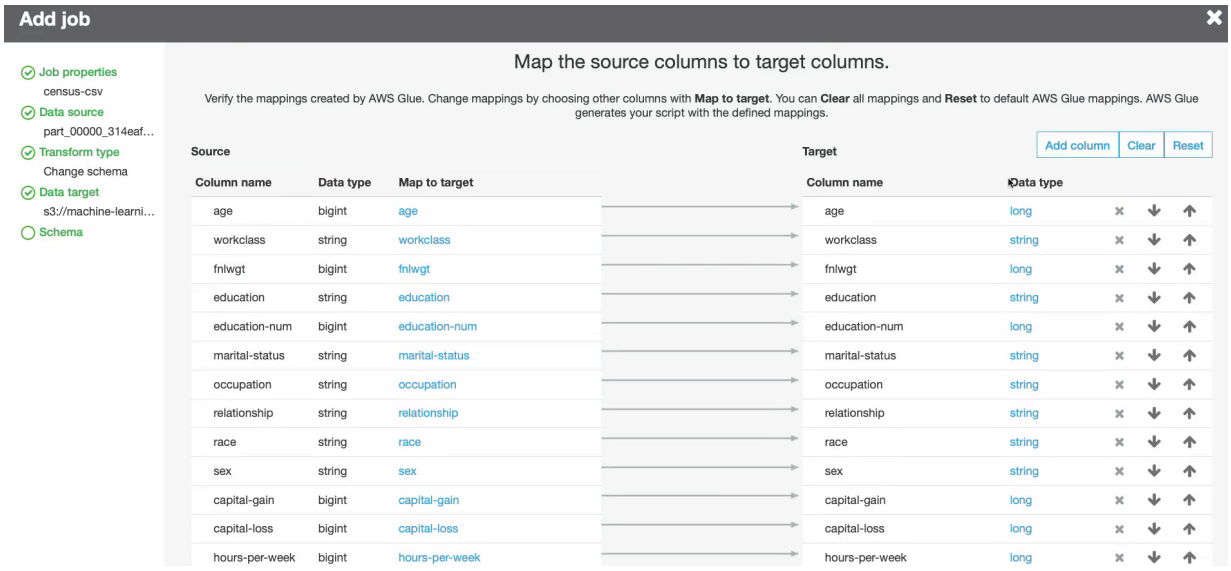


To note Glue can help find matching records using ML

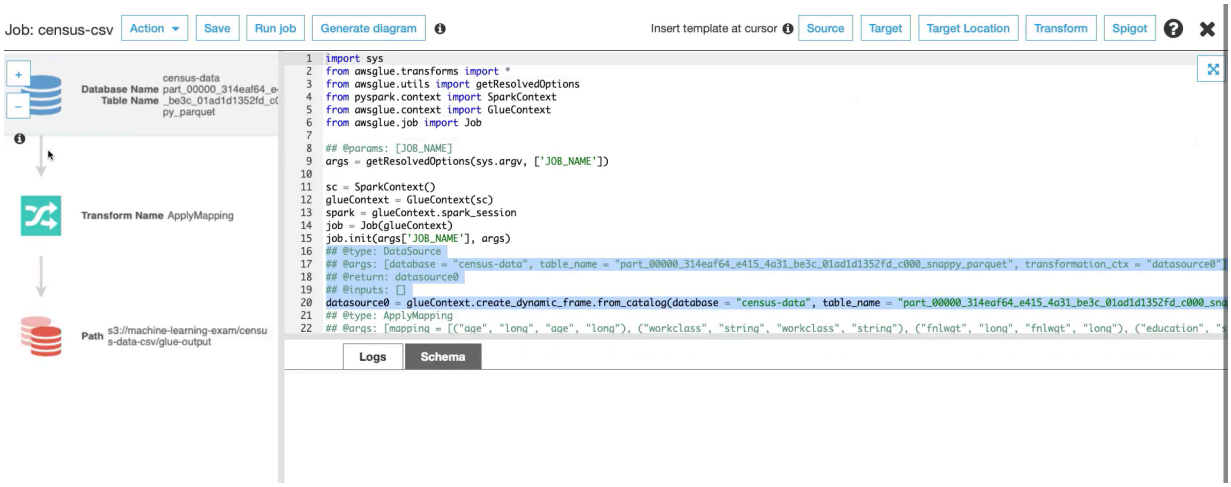


Target - we will use S3



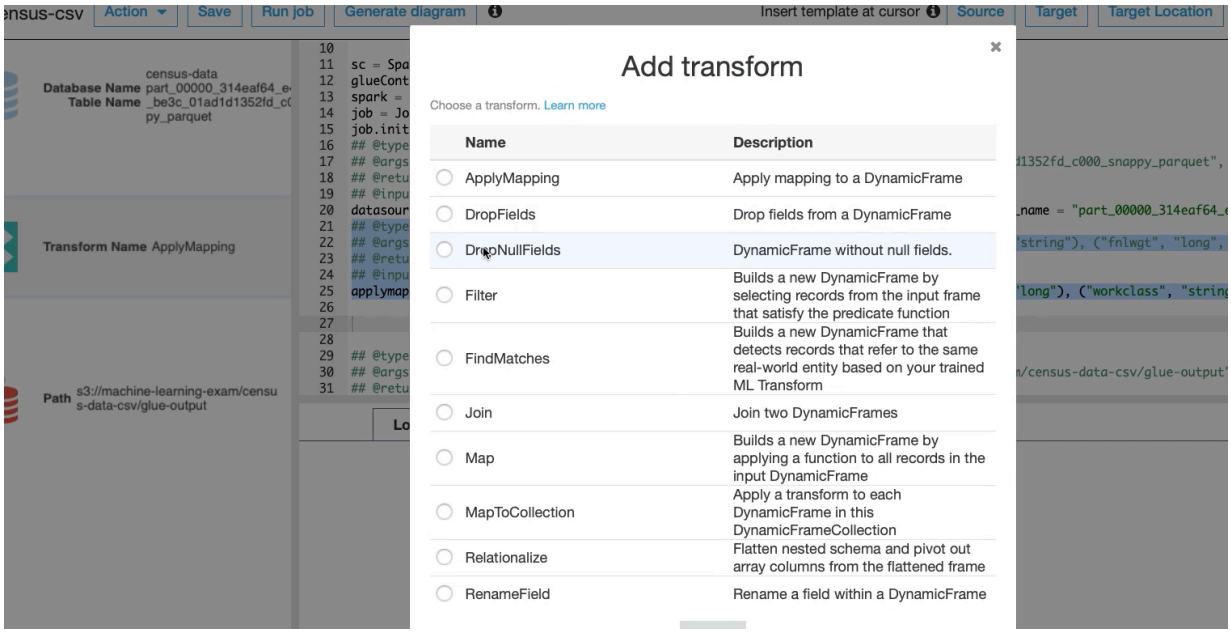


We now have a simple data pipeline

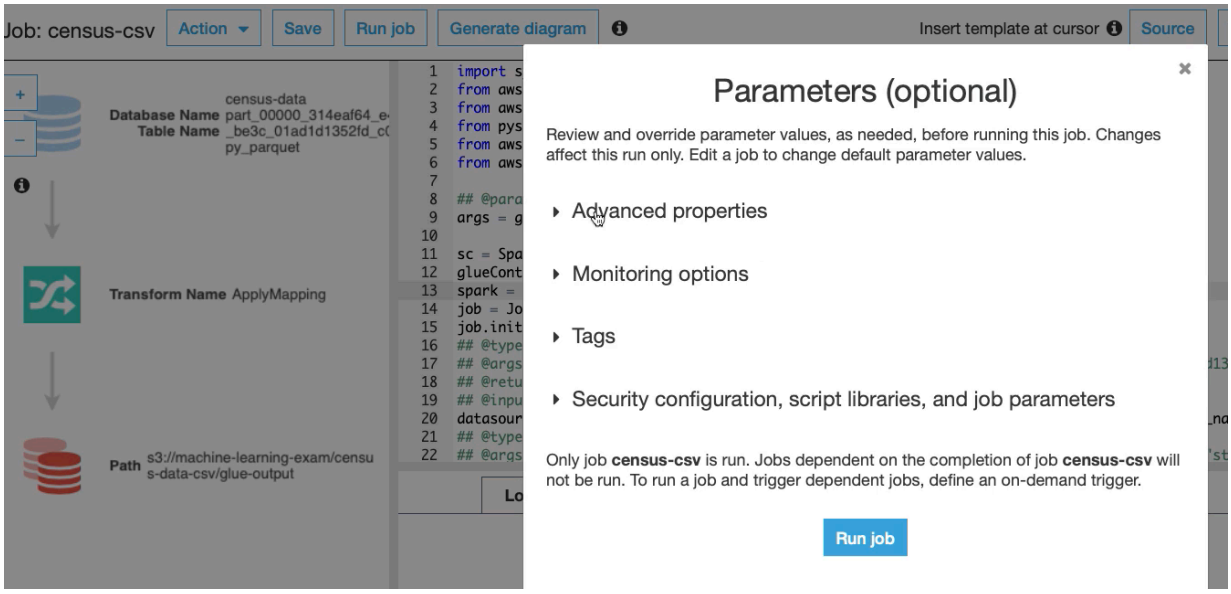



```
10
11  sc = SparkContext()
12  glueContext = GlueContext(sc)
13  spark = glueContext.spark_session
14  job = Job(glueContext)
15  job.init(args['JOB_NAME'], args)
16  ## @type: DataSource
17  ## @args: [database = "census-data", table_name = "part_00000_314eaf64_e415_4a31_be3c_01ad1d1352fd_c000_snappy_parquet", transformation_ctx = "dataso
18  ## @return: datasource0
19  ## @inputs: []
20  datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "census-data", table_name = "part_00000_314eaf64_e415_4a31_be3c_01ad1d1352fd_c
21  ## @type: ApplyMapping
22  ## @args: [mapping = [("age", "long", "age", "long"), ("workclass", "string", "workclass", "string"), ("fnlwgt", "long", "fnlwgt", "long"), ("educati
23  ## @return: applymapping1
24  ## @inputs: [frame = datasource0]
25  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("age", "long", "age", "long"), ("workclass", "string", "workclass", "string"), (
26  ## @type: DataSink
27  ## @args: [connection_type = "s3", connection_options = {"path": "s3://machine-learning-exam/census-data-csv/glue-output"}, format = "csv", transform
28  ## @return: datasink2
29  ## @inputs: [frame = applymapping1]
30  datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1, connection_type = "s3", connection_options = {"path": "s3://machine-l
31  job.commit()
```

To note we can add more transform steps in this code

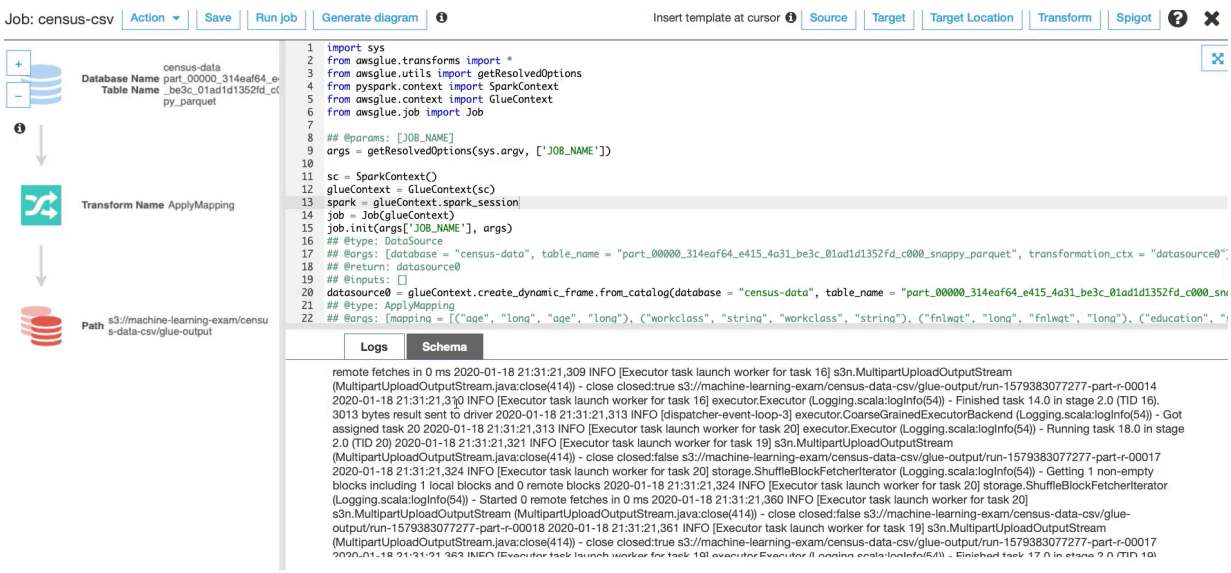


Now run job

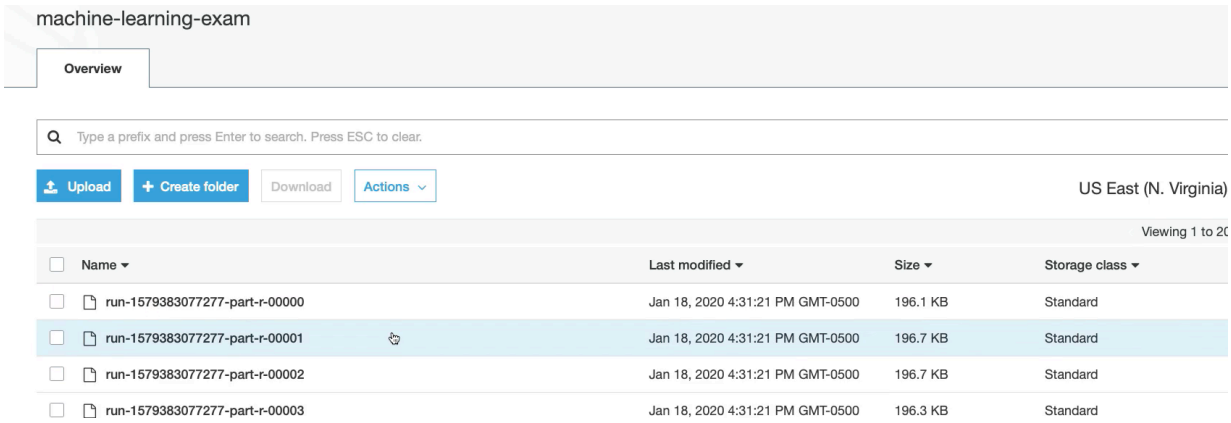


=> no resource to configure or manage.
Glue will take care of that for us (server less) - setting up in the background Apache Spark jobs

When it’s done, we can see the logs



In s3 we can now see the csv files that were created



It chunked up the info into 20 files.