

# AWS Glue and Athena

## Building a Serverless Data Lake

### Overview

In preparation for the workshop, we will create Amazon S3 buckets, copy sample data files to Amazon S3, and create an IAM service role.



### What is a data lake?

A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data, and run different types of analytics—from dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide better decisions.

### Introducing Amazon S3

Amazon Simple Storage Service (S3) is the largest and most performant object storage service for structured and unstructured data and the storage service of choice to build a data lake. With Amazon S3, you can cost-effectively build and scale a data lake of any size in a secure environment where data is protected by 99.999999999% (11 9s) of durability.

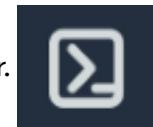
With data lakes built on Amazon S3, you can use native AWS services to run big data analytics, artificial intelligence (AI) and Machine Learning (ML) to gain insights from your unstructured data sets. Data can be collected from multiple sources and moved into the data lake in its original format. It can be accessed or processed with your choice of purpose-built AWS analytics tools and frameworks. Making it easy and quick to run analytics without the need to move your data to a separate analytics system.

## Start an AWS CloudShell environment

You will build the solution from ground up instead of using a CloudFormation template to provision the base resources. We will use the Command Line Interface (CLI) to accomplish this.

AWS CloudShell is a browser-based, pre-authenticated shell that you can launch directly from the AWS Management Console. You can run AWS CLI commands against AWS services using your preferred shell (Bash, PowerShell, or Z shell). And you can do this without needing to download or install command line tools.

1. Login to the AWS Management Console.
2. In the upper right menu section, select **N. Virginia (us-east-1)** region.



3. Start an AWS CloudShell session by clicking the CloudShell icon on the top menu bar.

4. A welcome dialog box will pop up, check **do not show this again**, and then click **Close**.

**Welcome to AWS CloudShell** X

AWS CloudShell is a browser-based shell that gives you command-line access to your AWS resources in the selected AWS region. AWS CloudShell comes pre-installed with popular tools for resource management and creation. You have the same credentials as you used to log in to the console. [Learn more](#)

<b>Pre-installed tools</b> AWS CLI, Python, Node.js and more	<b>Storage included</b> 1 GB of storage free per AWS region	<b>Saved files and settings</b> Files saved in your home directory are available in future sessions for the same AWS region
-----------------------------------------------------------------	----------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------

Do not show again **Close**

5. CloudShell will start preparing a Linux environment and you should see a terminal in your browser in a few moments.

## AWS CloudShell

us-east-1

```
Preparing your terminal...
[cldshell-user@ip-10-1-46-102 ~]$ Try these commands to get started:
aws help or aws <command> help or aws <command> --cli-auto-prompt
[cldshell-user@ip-10-1-46-102 ~]$ █
```

# Create S3 buckets and upload data

## Create an S3 bucket to store raw data and upload dataset

An **S3 bucket** is a storage resource that act as a container for objects stored in Amazon S3. You can review the [naming rules](#) before creating your buckets.

1. In the CloudShell terminal, run the following command to create a unique S3 bucket.

```
1 | aws s3api create-bucket --bucket [your-bucket-name]-raw --region us-east-1
```

## Upload the datasets to your Amazon S3 bucket

We will use the New York City Taxi and Limousine Commission (TLC) Trip Record Data. You can find more information about the dataset in the [Registry of Open Data on AWS](#) and by visiting the New York City Taxi and Limousine Commission (TLC) [data record page](#).

2. Run the following command to upload the yellow taxi trip data dataset to the S3 bucket you just created.

```
1 | aws s3 cp s3://nyc-tlc/csv_backup/ s3://[your-bucket-name]-raw/nyc-taxi/yellow-tripdata/
--exclude "*" --include "yellow_tripdata_2020*" --recursive
```

3. Run the following command to upload the taxi zone lookup table to the same S3 bucket.

```
1 | aws s3 cp "s3://nyc-tlc/misc/taxi_zone_lookup.csv" s3://[your-bucket-name]-raw/nyc-
taxi/taxi_zone_lookup/taxi_zone_lookup.csv
```

- Run the following command to list the files you uploaded.

```
1 | aws s3 ls s3://[your-bucket-name]-raw --recursive
```

## Create another Amazon S3 bucket to store transformed data

- Run the following command to create another unique S3 bucket. We will use this bucket to store transformed data.

```
1 | aws s3api create-bucket --bucket [your-bucket-name]-transformed --region us-east-1
```

## Create an IAM Service Role

### Create an IAM Service role for AWS Glue

**IAM Service Role** defines a set of permissions for making service requests that a trusted entity, an AWS service, can assume and perform.

- In the CloudShell terminal, run the following command to download and install Nano Text Editor. When prompted with "Is this ok", type `y`.

```
1 | sudo yum install nano
```

- Start nano to create a file.

```
1 | nano trust-policy.json
```

- In the editor, paste the following content. Type **Ctrl-X**, type **Y** and then **ENTER** to save and exit.

```
1 | {
2 |     "Version": "2012-10-17",
3 |     "Statement": [
4 |         {
5 |             "Effect": "Allow",
6 |             "Principal": {
7 |                 "Service": "glue.amazonaws.com"
8 |             },
9 |             "Action": "sts:AssumeRole"
10 |         }
11 |     ]
12 | }
```

4. Create another file using nano editor.

```
1 | nano s3access-policy.json
```

5. In the editor, paste the following content and update the bucket name. Type **Ctrl-X**, type **Y** and then **ENTER** to save and exit.

```
1 | {
2 |     "Version": "2012-10-17",
3 |     "Statement": [
4 |         {
5 |             "Effect": "Allow",
6 |             "Action": [
7 |                 "s3:PutObject",
8 |                 "s3:GetObject",
9 |                 "s3>ListBucket"
10 |             ],
11 |             "Resource": [
12 |                 "arn:aws:s3 :::: [your-bucket]-raw/*",
13 |                 "arn:aws:s3 :::: [your-bucket]-raw",
14 |                 "arn:aws:s3 :::: [your-bucket]-transformed/*",
15 |                 "arn:aws:s3 :::: [your-bucket]-transformed"
16 |             ]
17 |         }
18 |     ]
19 | }
```

6. In the CloudShell terminal, run the following commands to create an IAM role and attach permissions.

```
1 | aws iam create-role --role-name AWSGlueServiceRole-SDL-Jumpstart --assume-role-policy-document file://trust-policy.json
```

```
1 | aws iam put-role-policy --role-name AWSGlueServiceRole-SDL-Jumpstart --policy-name Glue-SDLS3Access --policy-document file://s3access-policy.json
```

```
1 | aws iam attach-role-policy --role-name AWSGlueServiceRole-SDL-Jumpstart --policy-arn arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole
```

7. To check if the IAM role was successfully created, run the following commands.

```
1 | aws iam get-role --role-name AWSGlueServiceRole-SDL-Jumpstart
```

```
1 | aws iam list-role-policies --role-name AWSGlueServiceRole-SDL-Jumpstart
```

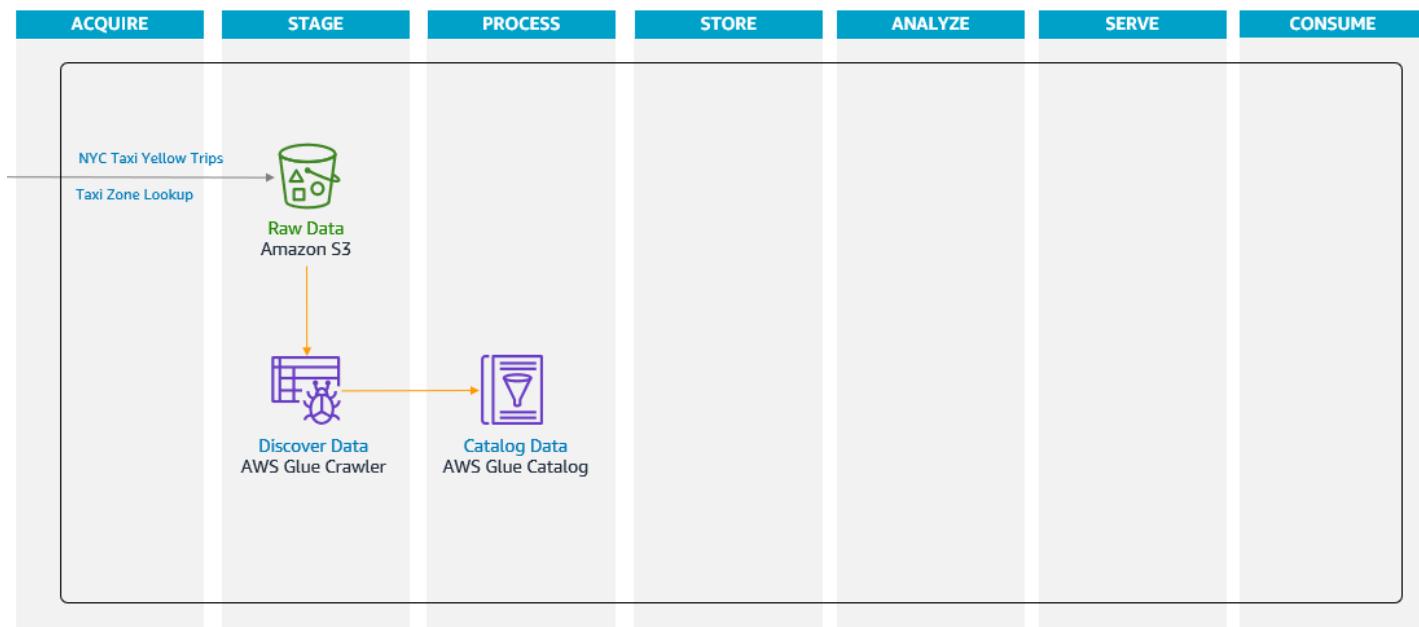
```
1 | aws iam list-attached-role-policies --role-name AWSGlueServiceRole-SDL-Jumpstart
```

It is a best practice to follow the **principle of least privilege**. Grant only permissions required to perform a task. Determine what users and/or roles need to do and then craft policies that allow them to perform only those tasks.

## Lab 1 - Discovering and Cataloging your Data

With an array of data sources and formats in your data lake, it's important to have the ability to discover and catalog it order to better understand the data that you have and at the same time enable integration with other purpose-built AWS analytics.

In this lab, we will create an AWS Glue Crawlers to auto discover the schema of the data stored in Amazon S3. The discovered information about the data stored in S3 will be registered in the AWS Glue Catalog. This allows AWS Glue to use the information stored in the catalog for ETL processing. It also allows other AWS services like Amazon Athena to run queries on the data stored in Amazon S3.



## Introducing AWS Glue

**AWS Glue** is a fully managed serverless extract, transform, and load (ETL) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores. AWS Glue consists of the following core components:

- **Data Catalog** – a central repository to store structural and operational metadata of your data assets.

- **ETL Engine** – automatically generate Scala or Python code.
- **Jobs System** – provides managed infrastructure to orchestrate your ETL workflow.

AWS Glue also includes additional components like:

- [AWS Glue DataBrew](#) - A visual data preparation tool that makes it easy for data analysts and data scientists to prepare data with an interactive, point-and-click visual interface without writing code.
- [AWS Glue Elastic Views](#) - Build materialized views that combine and replicate data across multiple data stores without you having to write custom code.

Together, these automate much of the undifferentiated heavy lifting involved with discovering, categorizing, cleaning, enriching, and moving data, so you can spend more time analyzing your data.

## Create a Glue Crawler

**Glue Crawler** is a feature that automatically infer database and table schema from your source data then stores the associated metadata in the AWS Glue Data Catalog.

1. Go to the AWS Glue Console.
2. In the left navigation menu, click **Crawlers**.
3. On the Crawlers page, click **Create a crawler**.

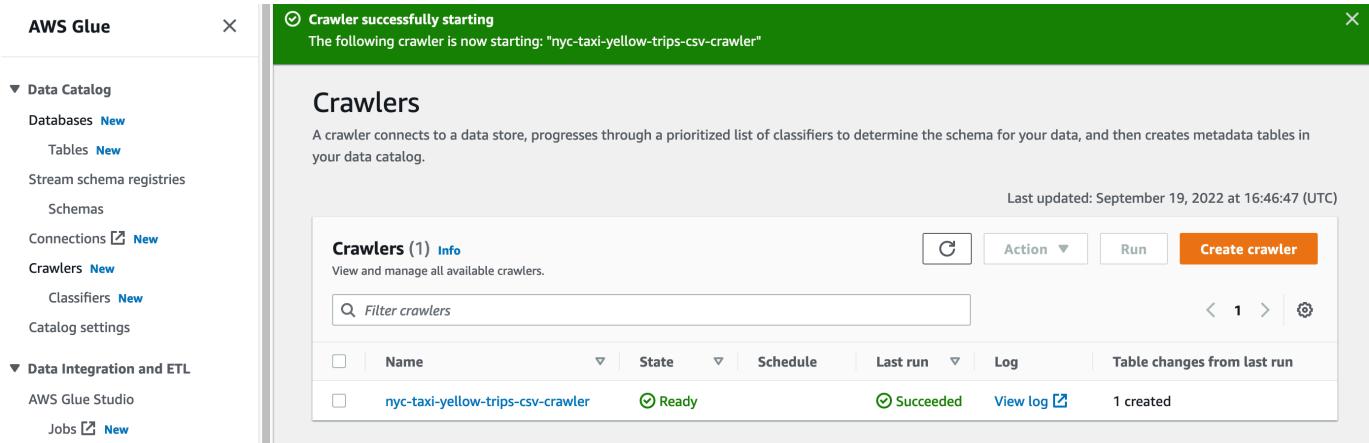
The screenshot shows the AWS Glue console interface. The left sidebar has a tree view with 'Data Catalog' expanded, showing 'Databases' (New), 'Tables' (New), 'Stream schema registries', 'Schemas', and 'Connections' (New). Below that is 'Crawlers' (New), 'Classifiers' (New), and 'Catalog settings'. Under 'Data Integration and ETL' is 'AWS Glue Studio' (New). The main content area is titled 'Crawlers' and contains a sub-section 'Crawlers (0) Info'. It has a 'Filter crawlers' search bar and buttons for 'Action', 'Run', and 'Create crawler'. A message at the bottom says 'You don't have any crawlers' and features a large 'Create crawler' button.

4. Specify `nyc-taxi-yellow-trips-csv-crawler` as the crawler name, click **Next**.
5. On the Choose data sources and classifiers screen, specify the following information, and then click **Next**.
  - Click Add a data source
  - Choose a Data source – **S3**
  - Select Location of S3 data – **In this account**
  - Include S3 path – `s3://[your-bucket-name]-raw/nyc-taxi/yellow-tripdata`
  - For Subsequent crawler runs, select to **Crawl all sub-folders**
  - Then click **Add an S3 data source**.

6. On Configure security settings, choose **AWSGlueServiceRole-SDL-Jumpstart** from the Existing IAM role, click **Next**.
7. On the Set output and scheduling screen, click **Add database**.
8. Specify `nyctaxi_db` as the unique database name, and then click **Create database**.
9. Go back to the previous tab (Set output and scheduling screen), refresh the selection for Target database and choose the newly created database `nyctaxi_db`.
10. Specify `raw_` in the Table name prefix - optional field.
11. On the Crawler schedule, leave the frequency **On demand**, click **Next**.
12. Review the crawler details, click **Create crawler**.

## Run the Glue Crawler

1. On the Crawlers page, select **nyc-taxi-yellow-trips-csv-crawler**, and then click **Run crawler**.
2. Upon successful completion of the crawler, you should see a value of **1** in the Tables added column.



The screenshot shows the AWS Glue interface. On the left, there's a navigation menu with sections like Data Catalog, Data Integration and ETL, and AWS Glue Studio. The main area is titled 'Crawlers' and contains a sub-section 'Crawlers (1) Info'. It shows a table with one row for 'nyc-taxi-yellow-trips-csv-crawler'. The table columns include Name, State, Last run, Log, and Table changes from last run. The 'State' column shows 'Ready' with a green circular icon. The 'Last run' column shows 'Succeeded' with a green circular icon. The 'Log' column has a 'View log' link. The 'Table changes from last run' column shows '1 created'.

## Review the metadata in Glue Data Catalog

**Glue Data Catalog** contain references to data that is used as sources and targets of your extract, transform, and load (ETL) jobs in AWS Glue. It also maintains a unified view of the data and enables various AWS services such as Athena, EMR, and Redshift Spectrum to access it.

1. In the left navigation menu, click **Tables**.
2. On the Tables page, click on the name **raw\_yellow\_tripdata** to review the table metadata and schema information.

The screenshot shows the AWS Glue Table details page for 'raw\_yellow\_tripdata'. The table is part of the 'nyctaxi\_db' database. The schema includes 18 columns: vendorid, tpep\_pickup\_datetime, tpep\_dropoff\_datetime, passenger\_count, trip\_distance, ratecodeid, store\_and\_fwd\_flag, and pulocationid. The table was last updated on June 19, 2024, at 15:11:37.

#	Column name	Data type	Partition key	Comment
1	vendorid	bigint	-	-
2	tpep_pickup_datetime	string	-	-
3	tpep_dropoff_datetime	string	-	-
4	passenger_count	bigint	-	-
5	trip_distance	double	-	-
6	ratecodeid	bigint	-	-
7	store_and_fwd_flag	string	-	-
8	pulocationid	bigint	-	-

## Create another Glue Crawler

Repeat the same steps to "crawl" the taxi zone lookup data stored in S3

1. In the left navigation menu, click **Crawlers**.
2. On the Crawlers page, click **add crawler**.
3. Specify **nyc-taxi-zone-lookup-csv-crawler** as the crawler name, click **Next**.
4. On the Choose data sources and classifiers screen, specify the following information, and then click Next.
  - o Click Add a data source
  - o Choose Data source - **S3**
  - o Select Location of S3 data - **In this account**
  - o Include S3 path - **s3://[your-bucket-name]-raw/nyc-taxi/taxi\_zone\_lookup**
  - o For Subsequent crawler runs, select to **Crawl all sub-folders**
  - o Then click **Add an S3 data source**.
5. On the Configure security settings, choose **AWSGlueServiceRole-SDL-Jumpstart** from the Existing IAM role, click **Next**.
6. On the Set output and scheduling screen, choose **nyctaxi\_db** as the database.
7. On the Crawler schedule, leave the frequency **On demand**, click **Next**.
8. Review the crawler details, click **Create crawler**.
9. On the Crawlers page, select **nyc-taxi-zone-lookup-csv-crawler**, and then click **Run**.
10. A new table, **taxi\_zone\_lookup**, will be created in the Glue Catalog after the crawler successfully completes. Review the metadata and schema information of the table.

**Crawlers**

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Name	State	Last run	Last run timestamp	Log	Table changes from...
nyc-taxi-yellow-trips-csv-cr...	Ready	Succeeded	June 19, 2024 at 15:10:18	<a href="#">View log</a>	1 created
nyc-taxi-zone-lookup-csv-c...	Ready	Succeeded	June 19, 2024 at 15:15:37	<a href="#">View log</a>	1 created

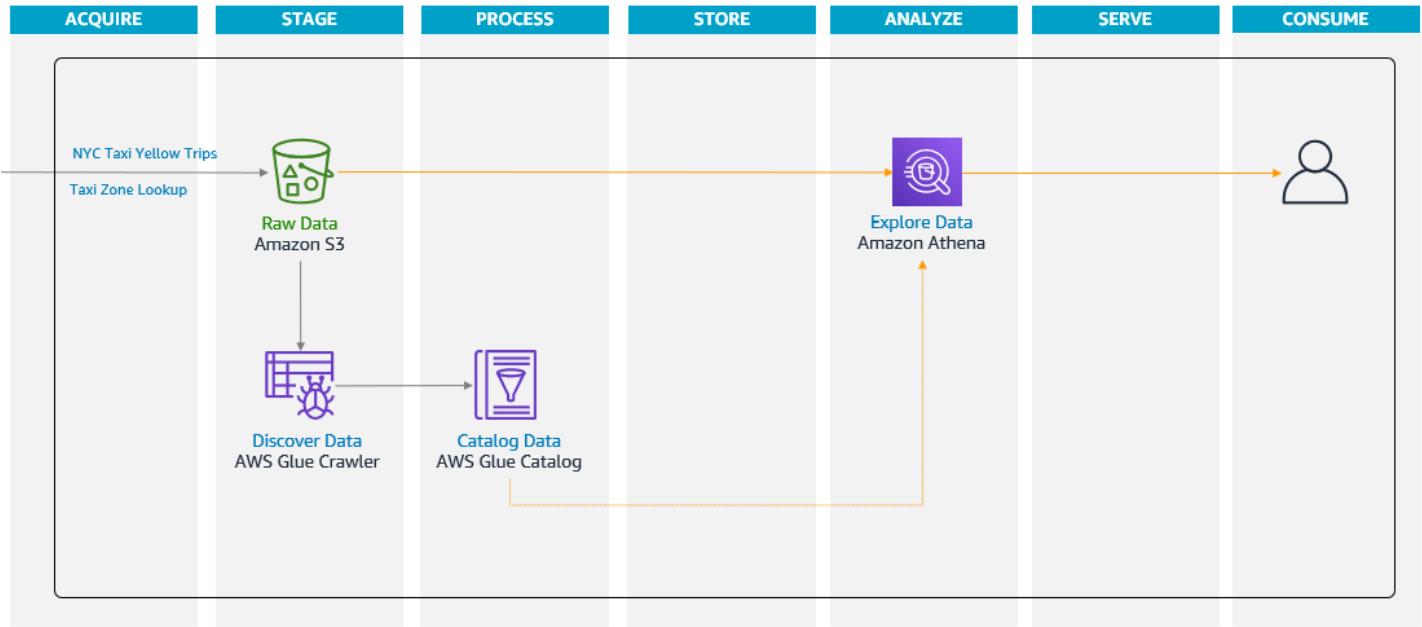
**Tables**

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Name	Database	Location	Classification	Deprecated	View data	Data quality
raw_yellow_tripdata	nyctaxi_db	s3://tarek-demo1-raw/nyc-taxi	CSV	-	<a href="#">Table data</a>	<a href="#">View data quality</a>
taxi_zone_lookup	nyctaxi_db	s3://tarek-demo1-raw/nyc-taxi	CSV	-	<a href="#">Table data</a>	<a href="#">View data quality</a>

## Lab 2 - Exploring your Data

In this lab, we will use Amazon Athena to explore our data and identify data quality issues. We will also learn how to update table properties in AWS Glue Catalog.



## Introducing Amazon Athena

**Amazon Athena** is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL. Athena is serverless, so there is no infrastructure to setup or manage, and you can start analyzing data immediately. You don't even need to load your data into Athena, it works directly with data stored in S3.

Amazon Athena automatically stores query results and metadata information for each query that runs in a query result location that you can specify in Amazon S3. If necessary, you can access the files in this location to work with them. You can also download query result files directly from the Athena console.

1. Launch a CloudShell terminal, and then run the following command to create an S3 bucket to store Athena's query results.

```
1 | aws s3api create-bucket --bucket aws-athena-query-results-us-east-1-[Account-ID] --region us-east-1
```

2. Go to Athena console. On the **Get Started** click **Launch Query Editor** or just select Query Editor from the left side panel/menu.

aws Services Search [Option+S] N. Virginia tarek.atwan @ developintelligence

Analytics

## Amazon Athena

### Start querying data instantly.

Amazon Athena is an interactive query service that makes it easy to analyze data in Amazon S3 and other federated data sources using standard SQL.

**Get started**

- Query your data with Trino SQL**  
Use Query editor to analyze data on S3, on premises, or on other clouds.
- Analyze your data using PySpark and Spark SQL**  
Use notebooks to build interactive Spark applications.

**Launch query editor**

**How it works**



1. Point to your data source  
Register and select any data source.

SQL queries per TB scanned  
\$5.00 per TB

SQL queries on Provisioned Capacity  
\$0.30 per DPU hour

Apache Spark calculations  
\$0.35 per DPU hour

**Pricing**  
Region: US East (N. Virginia)

**Cost calculator**

**Getting started**

Add the data source and identify the

Services Search [Option+S] N. Virginia tarek.atwan @ developintelligence

Amazon Athena > Query editor

**Editor** Recent queries Saved queries Settings **Workgroup primary**

Before you run your first query, you need to set up a query result location in Amazon S3. **Edit settings**

Athena now supports typeahead code suggestions to speed up SQL query development. Typeahead suggestions are turned on by default. You can change this setting in query editor preferences. **Edit preferences**

Data  **Query 1**  

3. Choose **Settings**, click on **Browse S3** and select **aws-athena-query-results-us-east-1-[Account-ID]** as the value for the Location of query result - optional field.

## Manage settings

### Query result location and encryption

#### Location of query result - *optional*

Enter an S3 prefix in the current region where the query result will be saved as an object.

[View](#) [Browse S3](#)

You can create and manage lifecycle rules for this bucket

Use Amazon S3 lifecycle rules to store your query results and metadata cost effectively or to delete them after a period of time.

[Lifecycle configuration](#) [Learn more](#) 

#### Expected bucket owner - *optional*

Specify the AWS account ID that you expect to be the owner of your query results output location bucket.

**Assign bucket owner full control over query results**

Enabling this option grants the owner of the S3 query results bucket full control over the query results. This means that if your query result location is owned by another account, you grant full control over your query results to the other account.

**Encrypt query results**

[Cancel](#)[Save](#)

4. Go to the bottom of the page, click **Save**.

5. Go to the top menu, click on **Editor** to return back to the Query editor page.

The screenshot shows the AWS Lambda interface with the Athena service selected. The main area is the 'Query editor'. On the left, the 'Data' pane shows a 'Data source' set to 'AwsDataCatalog' and a 'Database' set to 'nytaxi\_db'. Under 'Tables and views', there are two tables: 'raw\_yellow\_tripdata' and 'taxi\_zone\_lookup', and zero views. In the center, 'Query 1' is being run, with the SQL command 'SELECT \* FROM raw\_yellow\_tripdata'. Below the query, the 'Results' pane is visible, showing a preview of the data with columns like 'VendorID', 'tpep\_pickup\_datetime', 'tpep\_dropoff\_datetime', etc. A message at the top indicates that typeahead code suggestions are supported. The top right shows the workgroup is 'primary'.

## Preview Table Data

1. On the Query editor page, click on the actions menu icon : besides **raw\_yellow\_tripdata**, and then select **Preview table**.
2. Examine the data.

This screenshot shows the same Athena Query editor interface as above, but with a different focus. The 'Actions' menu is open over the 'raw\_yellow\_tripdata' table entry in the 'Tables' section of the Data pane. The 'Preview Table' option is highlighted. The rest of the interface remains the same, with the 'Query 1' editor showing the SELECT statement and the results pane below.

The screenshot shows the AWS Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog) and Database (nyctaxi\_db). The main area shows a query window with the following SQL:

```
1 SELECT * FROM "nyctaxi_db"."raw_yellow_tripdata" limit 10;
```

The results pane shows the output of the query, which includes 10 rows of trip data. The columns are: #, vendorid, tpep\_pickup\_datetime, tpep\_dropoff\_datetime, passenger\_count, trip\_distance, ratecodeid, store\_and\_fwd\_flag, pulocationid, dolocationid, and payment\_type.

#	vendorid	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	ratecodeid	store_and_fwd_flag	pulocationid	dolocationid	payment_type
1	1	2020-10-08 14:43:35	2020-10-08 14:56:26	1	2.4	1	N	234	229	1
2	2	2020-10-08 14:23:41	2020-10-08 14:40:26	1	4.77	1	N	75	224	2
3	2	2020-10-08 14:44:33	2020-10-08 15:00:39	1	3.99	1	N	107	45	1
4	2	2020-10-08 14:18:10	2020-10-08 14:33:17	1	3.43	1	N	263	170	1
5	2	2020-10-08 14:50:47	2020-10-08 15:00:49	1	1.25	1	N	140	237	1
6	1	2020-10-08 14:33:32	2020-10-08 14:38:38	2	0.8	1	N	236	141	1
7	1	2020-10-08 14:40:26	2020-10-08 15:03:28	1	2.6	1	N	236	237	1
8	2	2020-10-08 14:29:54	2020-10-08 15:13:54	1	20.36	1	N	132	14	2
9	1	2020-10-08 14:40:16	2020-10-08 14:48:24	1	1.4	1	N	140	236	2
10	1	2020-10-08 14:49:33	2020-10-08 14:53:19	1	0.5	1	N	236	75	1

3. Do the same for **taxi\_zone\_lookup** table. Observe that all string values are enclosed with “”.

The screenshot shows the AWS Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog) and Database (nyctaxi\_db). The main area shows a query window with the following SQL:

```
1 SELECT * FROM "nyctaxi_db"."taxi_zone_lookup" limit 10;
```

The results pane shows the output of the query, which includes 10 rows of zone data. The columns are: #, locationid, borough, zone, and service\_zone.

#	locationid	borough	zone	service_zone
1	1	"EWR"	"Newark Airport"	"EWR"
2	2	"Queens"	"Jamaica Bay"	"Boro Zone"
3	3	"Bronx"	"Allerton/Pelham Gardens"	"Boro Zone"
4	4	"Manhattan"	"Alphabet City"	"Yellow Zone"
5	5	"Staten Island"	"Arden Heights"	"Boro Zone"
6	6	"Staten Island"	"Arrochar/Fort Wadsworth"	"Boro Zone"
7	7	"Queens"	"Astoria"	"Boro Zone"
8	8	"Queens"	"Astoria Park"	"Boro Zone"
9	9	"Queens"	"Auburndale"	"Boro Zone"

## Working with CSV data enclosed in quotes

### Working with CSV files enclosed in Quotes

CSV files occasionally have quotes around the data values intended for each column which aren't part of the data to be analyzed. To read the CSV file properly, we can update the table properties in AWS Glue to use the OpenCSVSerDe.

1. Go to Glue console.
2. In the left navigation menu, click Tables under the Data Catalog section.
3. On the Table screen, click on the **taxi\_zone\_lookup table**.
4. Click **Actions**, then click on **Edit table**.
5. Update the **Serde serialization lib** with `org.apache.hadoop.hive.serde2.OpenCSVSerde`.
6. Remove existing Serde parameters and then add the following:
  - o `escapeChar`, enter a backslash \
  - o `quoteChar`, enter a double quote "
  - o `separatorChar`, enter a comma ,

Edit table

▼ Table details

Name	taxi_zone_lookup
Input format	<code>org.apache.hadoop.mapred.TextInputFormat</code>
Output format	<code>org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat</code>
Serde name	
Serialization lib	<code>org.apache.hadoop.hive.serde2.OpenCSVSerde</code>
Description	

## ▼ Table details

Name

Input format

Output format

Serde name

Serialization lib

Description



## ▼ Serde parameters

Key

Value

<input type="text" value="escapeChar"/>	<input type="text" value="\"/>	<input type="button" value="Remove"/>
-----------------------------------------	--------------------------------	---------------------------------------

<input type="text" value="quoteChar"/>	<input type="text" value="\"/>	<input type="button" value="Remove"/>
----------------------------------------	--------------------------------	---------------------------------------

<input type="text" value="separatorChar"/>	<input type="text" value=","/> <span style="border: 1px solid #0072bc; border-radius: 5px; padding: 2px;">,</span>	<input type="button" value="Remove"/>
--------------------------------------------	-----------------------------------------------------------------------------------------------------------------------	---------------------------------------

7. Click **Save**

8. Go to back to the Athena console.

9. On the Query editor page, click on the actions menu icon : besides **taxi\_zone\_lookup**, and then click **Preview table**.

The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a sidebar for 'Data' with sections for 'Data source' (set to 'AwsDataCatalog') and 'Database' (set to 'nyctaxi\_db'). Below that is a 'Tables and views' section with a 'Create' button and a search bar. It lists two tables: 'raw\_yellow\_tripdata' and 'taxi\_zone\_lookup'. Under 'Views (0)', there are no entries. The main area is titled 'Query 1 × | Query 2 × | Query 3 × | Query 4 ×' with 'Query 4' being the active tab. The query text is: 'SELECT \* FROM "nyctaxi\_db"."taxi\_zone\_lookup" limit 10;'. Below the query is a toolbar with buttons for 'Run again', 'Explain', 'Cancel', 'Save', 'Clear', and 'Create'. The 'Query results' tab is selected, showing a green status bar with 'Completed' and performance metrics: 'Time in queue: 244 ms', 'Run time: 468 ms', and 'Data scanned: 12.03 KB'. The 'Results (10)' table has columns: '#', 'locationid', 'borough', 'zone', and 'service\_zone'. The data is as follows:

#	locationid	borough	zone	service_zone
1	1	EWR	Newark Airport	EWR
2	2	Queens	Jamaica Bay	Boro Zone
3	3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	4	Manhattan	Alphabet City	Yellow Zone
5	5	Staten Island	Arden Heights	Boro Zone
6	6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
7	7	Queens	Astoria	Boro Zone
8	8	Queens	Astoria Park	Boro Zone
9	9	Queens	Auburndale	Boro Zone
10	10	Queens	Baisley Park	Boro Zone

## Run SQL queries to explore the data

1. Check number of yellow taxi trip records.

```
1 -- total records
2 -- 24648499
3 SELECT COUNT(*) "Count" FROM raw_yellow_tripdata;
```

2. Explore data categories.

```
1 -- observe NULL values
2 -- 809568
3 SELECT vendorid, COUNT(*) "Count"
4 FROM raw_yellow_tripdata
5 GROUP BY vendorid
6 ORDER BY 1;
```

```
1 -- observe other categories
2 SELECT pulocationid, COUNT(*) "Count"
3 FROM raw_yellow_tripdata
4 GROUP BY pulocationid
5 ORDER BY 1;
```

```
1 -- observe NULL values
2 -- 809568
3 SELECT payment_type, COUNT(*) "Count"
4 FROM raw_yellow_tripdata
5 GROUP BY payment_type
6 ORDER BY 1;
```

### 3. Explore records with NULL Vendor ID.

```
1 -- observe other columns with NULL values
2 -- passenger_count, ratecodeid, store_and_fwd_flag, payment_type
3 SELECT *
4 FROM raw_yellow_tripdata
5 WHERE vendorid IS NULL
6 LIMIT 100;
```

### 4. Explore records by time period.

```
1 -- tpep_pickup_datetime is defined as STRING
2 -- observe record counts that falls outside of the time period
3 SELECT SUBSTR(tpep_pickup_datetime, 1, 7) "Period", COUNT(*) "Total Records"
4 FROM raw_yellow_tripdata
5 GROUP BY SUBSTR(tpep_pickup_datetime, 1, 7)
6 ORDER BY 1;
```

### 5. Count records that falls outside of year 2020.

```
1 -- records with incorrect pickup datetime values
2 -- 280
3 SELECT COUNT(*) "Count"
4 FROM raw_yellow_tripdata
5 WHERE SUBSTR(tpep_pickup_datetime, 1, 7) NOT LIKE '2020%';
```

### 6. Count records with NULL values (based on Vendor ID) that falls within 2020.

```

1 | -- Records with NULL categories like Vendor ID
2 | -- 809568
3 | SELECT COUNT(*) "Count"
4 | FROM raw_yellow_tripdata
5 | WHERE vendorid IS NULL
6 | AND SUBSTR(tpep_pickup_datetime, 1, 7) LIKE '2020%';

```

7. Count records that falls in the last quarter of 2020, exclude records with missing Vendor ID.

```

1 | -- Total records in BER months, excluding columns with missing Vendor ID
2 | -- 4347658
3 | SELECT COUNT(*) "Count"
4 | FROM raw_yellow_tripdata
5 | WHERE vendorid IS NOT NULL
6 | AND SUBSTR(tpep_pickup_datetime, 1, 7) LIKE '2020-1%';

```

8. Join taxi trips data with taxi zone look up table.

```

1 | -- explore data with lookup information
2 | -- observe column names from lookup tables
3 | SELECT td.* , pu.* , do.*
4 | FROM raw_yellow_tripdata td,
5 |      taxi_zone_lookup pu,
6 |      taxi_zone_lookup do
7 | WHERE td.pulocationid = pu.locationid AND
8 |       td.pulocationid = do.locationid AND
9 |       vendorid IS NOT NULL AND
10 |      SUBSTR(tpep_pickup_datetime, 1, 7) LIKE '2020-1%'
11 | LIMIT 100;

```

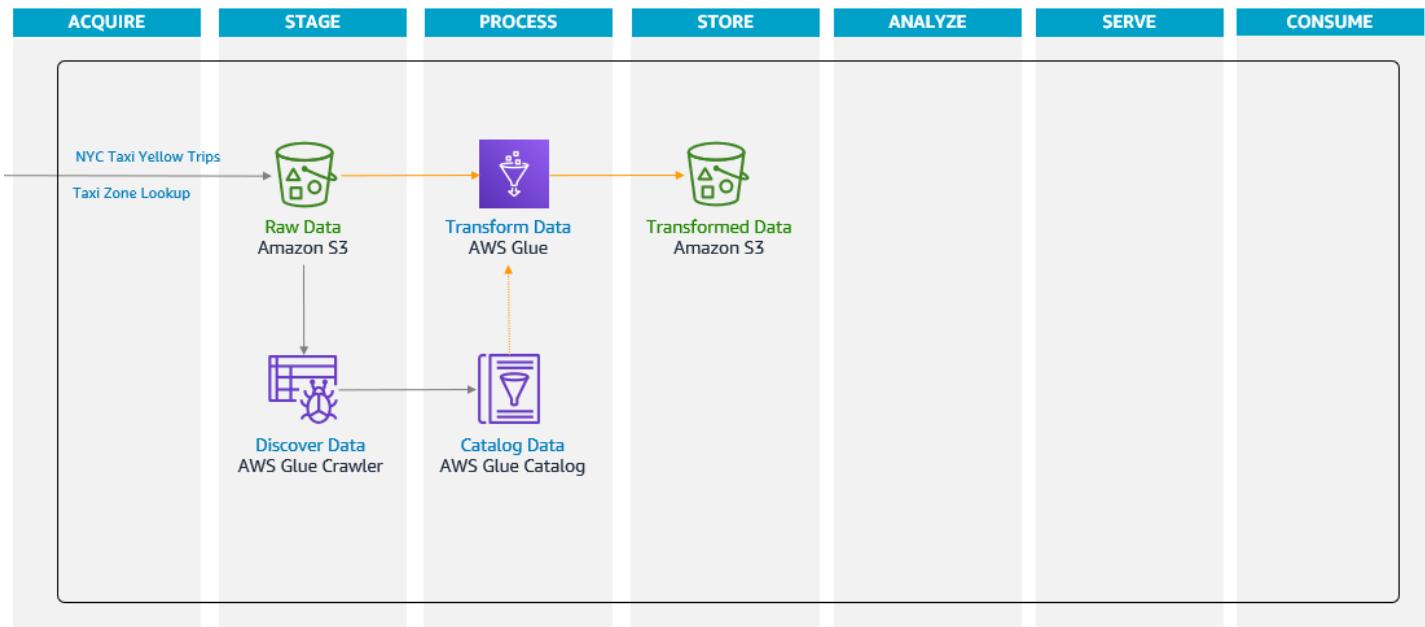
```

1 | -- Count total joined records for the last quarter of 2020.
2 | -- 4347658
3 | SELECT COUNT(*) "Count"
4 | FROM raw_yellow_tripdata td,
5 |      taxi_zone_lookup pu,
6 |      taxi_zone_lookup do
7 | WHERE td.pulocationid = pu.locationid AND
8 |       td.pulocationid = do.locationid AND
9 |       vendorid IS NOT NULL AND
10 |      SUBSTR(tpep_pickup_datetime, 1, 7) LIKE '2020-1%';

```

## Lab 3 - Transforming your Data

Data enrichment is the process of enhancing existing data with additional data from other sources to provide additional context or meaning, it makes data more useful and insightful during analysis.



## Introducing AWS Glue Studio

**AWS Glue Studio** is a graphical interface that makes it easy to create, run, and monitor extract, transform, and load (ETL) jobs in AWS Glue. You can visually compose data transformation workflows, AWS Glue studio will generate Apache Spark code on your behalf, and let it seamlessly run them on AWS Glue's Apache Spark-based serverless ETL engine.

## Create a job using Glue Studio

### Planning the Data Transformation Steps

It's a good practice to plan the steps to transform your data. Based on the information we captured during data exploration stage, we can come up with the following transformation step:

1. Read yellow trip data from S3, `raw_yellow_tripdata` table.
2. Clean yellow trip data data.
  - o Remove records with NULL values (vendorid, payment\_type, passenger count, ratecodeid).
  - o Filter records within a time period (remove records with invalid pickup datetime, narrow down data to be processed).
3. Join yellow trip data with taxi zone lookup to obtain pickup location information.
  - o Read lookup data from S3, `taxi_zone_lookup` table.
  - o Rename column names of lookup data to differentiate pickup locations from drop-off locations.
  - o Perform the join.

4. Join yellow trip data with taxi zone lookup to obtain drop-off location information.
  - Read lookup data from S3, `taxi_zone_lookup` table.
  - Rename column names of lookup data to differentiate drop-off locations from pickup locations.
  - Perform the join.
5. Perform data transformation on joined dataset.
  - Rename column names.
  - Set appropriate data types.
  - Remove redundant columns as a result of table joins.
6. Save processed dataset to S3 in a query optimized format.

## Create a job using Glue Studio

1. Go to Glue console.
2. In the left navigation menu, under the ETL section, click **AWS Glue Studio**.

3. On the AWS Glue Studio page, under **Create Job** click **Visual ETL**.
4. In the Glue Studio editor page, go to the Job details tab, set the following configuration:
  1. Name - `Transform NYC Taxi Trip Data`
  2. IAM Role - **AWSGlueServiceRole-SDL-Jumpstart**
  3. Job bookmark - **Disable**
  4. Number of retries - `0`

The screenshot shows the AWS Glue Data Catalog interface. At the top, there's a search bar and navigation tabs for 'Visual', 'Script', 'Job details', 'Runs', 'Data quality - updated', 'Schedules', and 'Version Control'. Below the search bar is a modal window titled '+ Add nodes' with a search input field 'Search sources, transforms and targets'. The main content area lists various data sources: AWS Glue Data Catalog, Amazon S3, Amazon Kinesis, Apache Kafka, Relational DB, Amazon Redshift, MySQL, PostgreSQL, Oracle SQL, Microsoft SQL Server, and Amazon DynamoDB. Each item has a small icon and a brief description. On the right side of the interface, there's a vertical toolbar with icons for actions like add, edit, delete, and search.

## Transform NYC Taxi Trip Data

Visual    Script    **Job details**    Runs    Data quality - *updated*    Schedules    Version Control

### Basic properties Info

#### Name

Transform NYC Taxi Trip Data

#### Description - optional

Descriptions can be up to 2048 characters long.

#### IAM Role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

AWSGlueServiceRole-SDL-Jumpstart



#### Type

The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

#### Glue version Info

Glue 4.0 - Supports spark 3.3, Scala 2, Python 3



#### Language

## Python 3

### Worker type

Set the type of predefined worker that is allowed when a job runs.

G 1X

(4vCPU and 16GB RAM)

### Automatically scale the number of workers

- AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

### Requested number of workers

The number of workers you want AWS Glue to allocate to this job.

10

### Generate job insights

- AWS Glue will analyze your job runs and provide insights on how to optimize your jobs and the reasons for job failures.

### Job bookmark | [Info](#)

Specifies how AWS Glue processes job bookmark when the job runs. It can remember previously processed data (Enable), update state information (Pause), or ignore state information (Disable).

Disable

### Flex execution | [Info](#)

- Reduce costs by running this job on spare capacity. Ideal for non-urgent workloads that don't require fast jobs start times or consistent execution times. See recommendations, limitations and pricing in the help panel by clicking on the Info link above.

### Number of retries

0

### Job timeout (minutes)

Set the execution time. The default is 2,880 minutes (48 hours) for a Glue ETL job. No job timeout is defaulted for a Glue Streaming job.

2880

### Usage profile

-

### ► Advanced properties

5. Click **Save**. Go back to the Visual tab.

AWS Glue **job bookmark** feature helps maintain state information and prevent the reprocessing of old data. It enables the job to process only new data when rerunning on a scheduled interval.

## Add a data source

# Adding Yellow Trip data from Amazon S3

1. Click on the **Source** icon, choose **S3**.

The screenshot shows the AWS Glue Data Catalog interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, a search bar containing 'Search', and a keyboard shortcut '[Option+S]'. Below the navigation bar, a green success message box displays: 'Successfully created job Transform NYC Taxi Trip Data. To run the job choose the Run Job button.' The main area is titled 'Transform NYC Taxi Trip Data' and features a tab bar with 'Visual' (selected), 'Script', 'Job details', 'Runs', 'Data quality - updated', 'Schedules', and 'Version Control'. A modal window titled '+ Add nodes' is open, showing a search bar 'Search sources, transforms and targets' and a 'Sources' tab selected. The 'Sources' tab lists the following data sources:

- AWS Glue Data Catalog: AWS Glue Data Catalog table as the data source.
- Amazon S3: JSON, CSV, or Parquet files stored in S3.
- Amazon Kinesis Data Stream: Read from an Amazon Kinesis Data Stream.
- Apache Kafka: Read from an Apache Kafka or Amazon MSK topic.
- Relational DB: AWS Glue Data Catalog table with a relational database as the data source.
- Amazon Redshift: Read your data from Amazon Redshift.
- MySQL: AWS Glue Data Catalog table with MySQL as the data source.
- PostgreSQL: AWS Glue Data Catalog table with PostgreSQL as the data source.
- Oracle SQL: AWS Glue Data Catalog table with Oracle SQL as the data source.
- Microsoft SQL Server: AWS Glue Data Catalog table with Microsoft SQL Server as the data source.

At the bottom of the modal window is a 'Manage Connections' button.

2. In the Data source – S3 bucket node, the specify the following information:

- Node properties tab, **Name** - `Yellow Trip Data`

- Data source properties tab, **Database** – `nyctaxi_db`
- Data source properties tab, **Table** – `raw_yellow_tripdata`

Last modified on 6/19/2024, 6:41:08 PM Actions ▾ Save Run

**Transform NYC Taxi Trip Data**

Visual Script Job details Runs Data quality - **updated** Schedules Version Control

**Data source properties - S3**

Name **Yellow Trip Data**

S3 source type **Info**  
 S3 location Choose a file or folder in an S3 bucket.  
 Data Catalog table

Database Choose a database.  
**nyctaxi\_db**

▶ Use runtime parameters

Table **raw\_yellow\_tripdata**

▶ Use runtime parameters

Partition predicate - optional  
Enter a boolean expression supported by Spark SQL, using only partition columns.  
Partition predicate syntax for Spark SQL is `year == year(date_sub(current_date, 7)) AND month == month(date_sub(current_date, 7)) AND day == day(date_sub(current_date, 7))`.

3. Click **Save**. Remember to save your work as you progress on building the transformation steps.

## Review resulting data schema and previewing data

4. Go to the Output schema tab to review the resulting data schema.
5. Go to Data preview tab to check sample data set while editing your job.
  - If asked for an IAM role select **AWSGlueServiceRole-SDL-Jumpstart**

## Transform NYC Taxi Trip Data

Visual    Script    Job details    Runs    Data quality - *updated*    Schedules    Version Control

+    Data source - S3 bucket  
Yellow Trip Data    ✓

Data preview    Output schema

Data preview (200) Info READY ?

Filter sample dataset

vendorid	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	ratecodeid
1	2020-01-01 00:28:15	2020-01-01 00:33:03	1	1.2	1
1	2020-01-01 00:35:39	2020-01-01 00:43:04	1	1.2	1
1	2020-01-01 00:47:41	2020-01-01 00:53:52	1	0.6	1
1	2020-01-01 00:55:23	2020-01-01 01:00:14	1	0.8	1
2	2020-01-01 00:01:58	2020-01-01 00:04:16	1	0	1
2	2020-01-01 00:00:11	2020-01-01 00:10:27	1	0.07	1

## Transform NYC Taxi Trip Data

The screenshot shows the AWS Glue Studio interface for transforming NYC Taxi Trip Data. At the top, there are tabs for Visual, Script, Job details, Runs, Data quality - *updated*, Schedules, and Version Control. The Visual tab is selected. Below the tabs, a data source is configured: "Data source - S3 bucket Yellow Trip Data" with a green checkmark indicating it is available. The "Output schema" tab is selected in the navigation bar. The schema table lists the following fields:

Key	Data type	Partition
vendorid	long	-
tpep_pickup_datetime	string	-
tpep_dropoff_datetime	string	-
passenger_count	long	-
trip_distance	double	-
ratecodeid	long	-
store_and_fwd_flag	string	-
pickuplocationid	long	-

At the bottom right of the schema table, there is a checkbox labeled "Infer schema from session".

AWS Glue Studio now allows you to preview your data at each step of the visual job authoring process so you can test and debug your transformations without having to save or run the job.

The first time you choose the Data preview tab, you are prompted to choose an IAM role to use. The IAM role you choose must have the necessary permissions to create the data previews. This can be the same role that you plan to use for your job, or it can be a different role.

After you choose an IAM role, it takes about 20 to 30 seconds before the data appears. You are charged for data preview usage as soon as you choose the IAM role.

## Remove records with NULL values

**Transform NYC Taxi Trip Data**

Visual    Script    Job details    Runs    Data qual

+ Add nodes X

cus X

Use: "cus"

**Custom Transform**  
Write **custom code to transform data.**  
Custom Transform  
Change field names, data types and drop fields. Formerly known as Apply Mapping.

**Join**  
Combine records from two datasets based on a set of conditions.

**SQL Query**  
Use a SQL query to transform data.

**Detect Sensitive Data**  
Detect PII and other sensitive information.

**Evaluate Data Quality**  
Evaluate the quality and completeness of your data.

**Fill Missing Values**  
Have AWS Glue infer reasonable values for missing data.

**Aggregate**  
Apply functions like sum or average to fields in the dataset.

**Custom Transform**  
Write custom code to transform data.

**Drop Duplicates**  
Remove duplicate records from your dataset.

**Drop Fields**

**Add Transforms**  

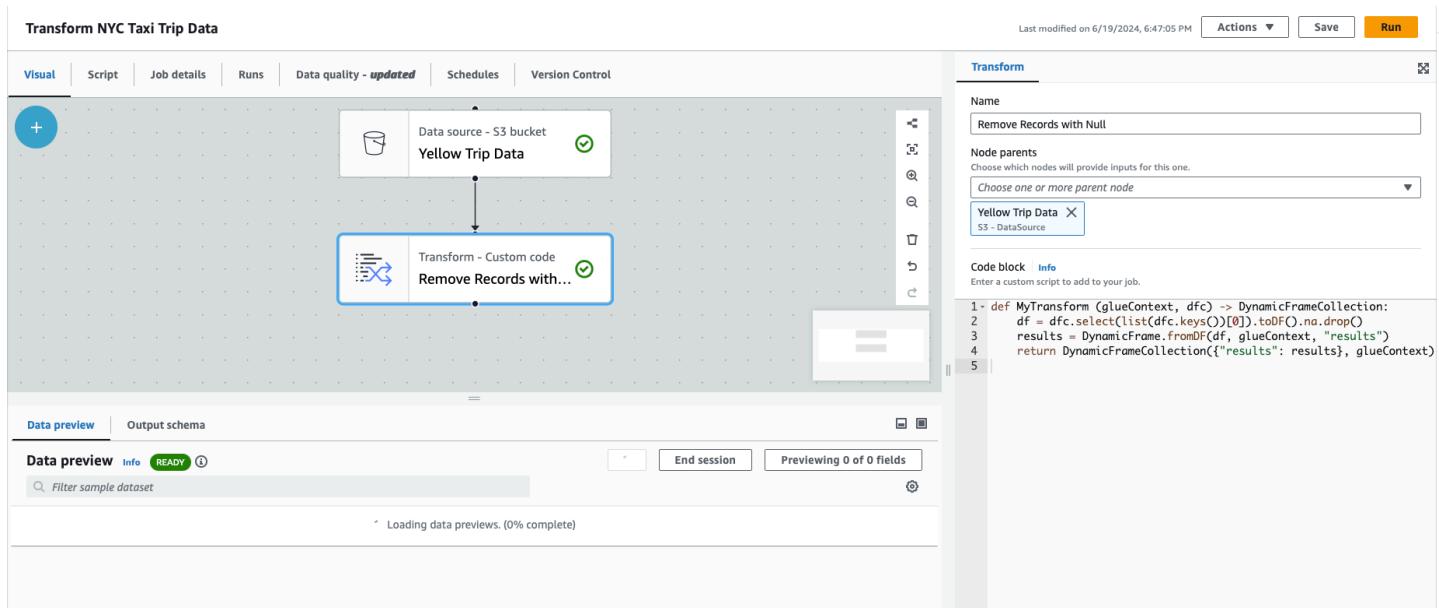
1. Click on the **Transforms** tab and choose **Custom Transform**. You can also search for the specific node.
2. In the Transform – Custom code node, specify the following information:
  - Node properties tab, **Name** - Remove Records with NULL

- Add the following in the Code block:

```

1 def MyTransform (glueContext, dfc) → DynamicFrameCollection:
2     df = dfc.select(list(dfc.keys())[0]).toDF().na.drop()
3     results = DynamicFrame.fromDF(df, glueContext, "results")
4     return DynamicFrameCollection({"results": results}, glueContext)

```



- Click on the **Transform** icon, choose **SelectFromCollection**.

## Transform NYC Taxi Trip Data

Visual

Script

Job details

Runs

Data qualit

+ Add nodes



select



Use: "select"

Drop Fields

Remove the selected fields from your data.

Select Fields

Choose which fields to keep from a dataset.

Select From Collection

Choose a single DynamicFrame from a collection of DynamicFrames.

Data Preparation Recipe

Select and execute an external DataBrew Recipe.

Detect PII and other sensitive information.

Evaluate Data Quality

Evaluate the quality and completeness of your data.

Fill Missing Values

Have AWS Glue infer reasonable values for missing data.

Aggregate

Apply functions like sum or average to fields in the dataset.

Custom Transform

Write custom code to transform data.

Drop Duplicates

Remove duplicate records from your dataset.

Drop Fields

Remove the selected fields from your data.

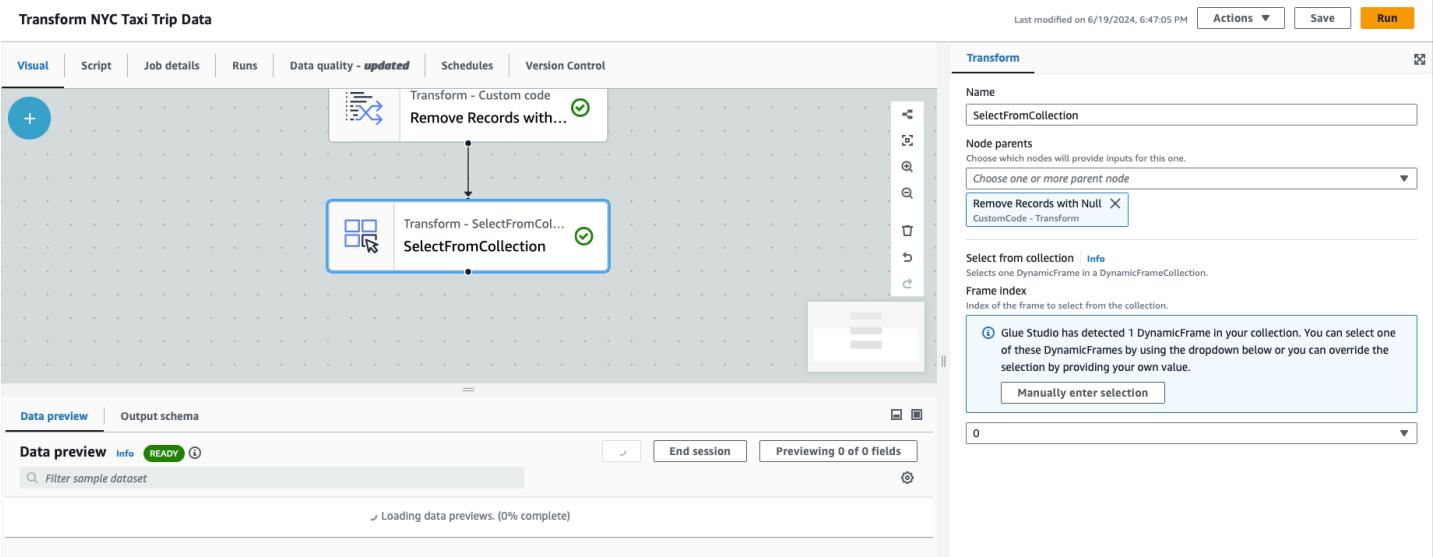
Drop Null Fields

Remove columns that have only empty/null values.

Add Transforms

4. Specify the following information:

- Node properties tab, **Name** - `SelectFromCollection`
- Transform tab, **Frame index** - `0`



5. Remember to save your work.

## Filter Records

### Filter records to remove records with invalid Pickup DateTime

In the interest of time, we will filter the records between *October 2020* and *December 2020* to reduce job processing time during the workshop.

1. Click on the **Transform** icon, choose **Filter**.

## Transform NYC Taxi Trip Data

Visual

Script

Job details

Runs

Data qual

+ Add nodes

X

filter

Use: "filter"



Filter

Filter data based on conditions.

Change field names, data types and drop fields. Formerly known as Apply Mapping.



Join

Combine records from two datasets based on a set of conditions.



SQL Query

Use a SQL query to transform data.



Detect Sensitive Data

Detect PII and other sensitive information.



Evaluate Data Quality

Evaluate the quality and completeness of your data.



Fill Missing Values

Have AWS Glue infer reasonable values for missing data.



Aggregate

Apply functions like sum or average to fields in the dataset.



Custom Transform

Write custom code to transform data.



Drop Duplicates

Remove duplicate records from your dataset.

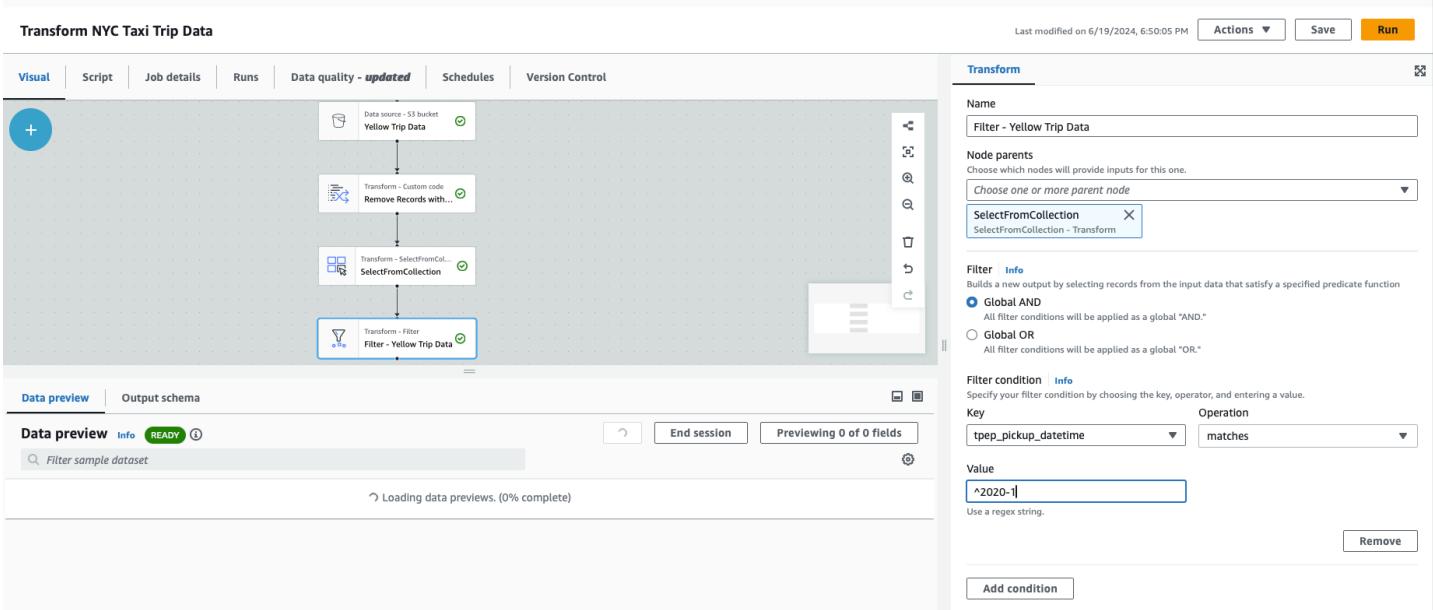


Drop Fields

Add Transforms

2. Specify the following information:

- Node properties tab, **Name** - Filter - Yellow Trip Data
- Transform tab, **Filter condition** - tpep\_pickup\_datetime matches ^2020-1



### 3. Remember to save your work

## Review the automatically generated script

1. Go to the Script details tab.
2. Observe that the code of the script are automatically generated by Glue Studio.

```

aws Services Search [Option+S] Last modified on 6/19/2024, 6:51:46 PM Actions Save Run
Transform NYC Taxi Trip Data N. Virginia tarek.katwan @ developintelligence
Visual Script Job details Runs Data quality - updated Schedules Version Control
Script (Locked) Info Download script Edit script
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7 from awsglue.dynamicframe import DynamicFrameCollection
8 from awsglue.dynamicframe import DynamicFrame
9 import re
10
11 # Script generated for node Remove Records with Null
12 def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
13     df = dfc.select(list(dfc.keys())[0], toDF().na.drop())
14     results = DynamicFrame.fromDF(df, glueContext, "results")
15     return DynamicFrameCollection([{"results": results}], glueContext)
16 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
17 sc = SparkContext()
18 glueContext = GlueContext(sc)
19 spark = glueContext.spark_session
20 job = Job(glueContext)
21 job.init(args['JOB_NAME'], args)
22
23 # Script generated for node Yellow Trip Data
24 YellowTripData_node1718811707400 = glueContext.create_dynamic_frame.from_catalog(database="nyctaxi_db", table_name="raw_yellow_tripdata", transformation_ctx="YellowTripData_node1718811707400")
25
26 # Script generated for node Remove Records with Null
27 RemoveRecordswithNull_node1718811977650 = MyTransform(glueContext, DynamicFrameCollection([{"YellowTripData_node1718811707400": YellowTripData_node1718811707400}], glueContext))
28
29 # Script generated for node SelectFromCollection
30 SelectFromCollection_node1718812149233 = SelectFromCollection.apply(dfc=RemoveRecordswithNull_node1718811977650, key=list(RemoveRecordswithNull_node1718811977650.keys())[0], transformation_ctx="SelectFromCollection")
31
32 # Script generated for node Filter - Yellow Trip Data
33 FilterYellowTripData_node1718812227608 = Filter.apply(frame=SelectFromCollection_node1718812149233, f=lambda row: (bool(re.match("^2020-1", row["tpep_pickup_datetime"]))), transformation_ctx="FilterYellowTripData")
34
35 job.commit()

```

## Add another data source

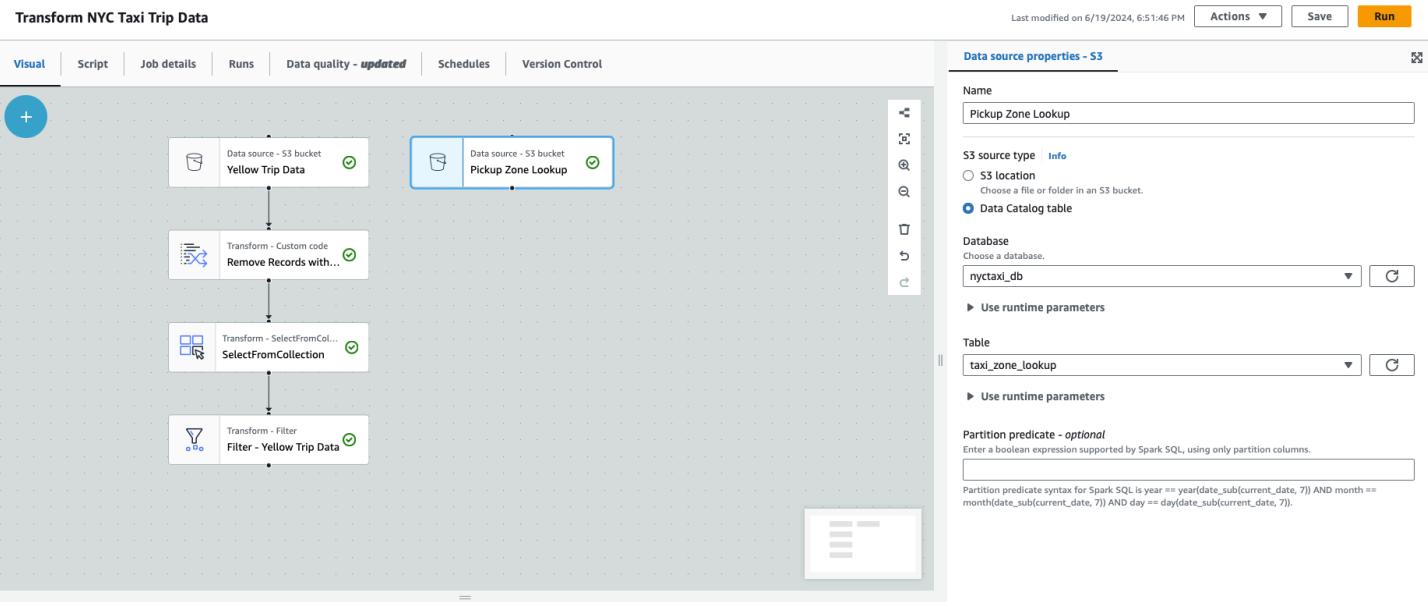
### Add Lookup table for Taxi Pickup Zone

1. Return to the Visual tab

2. Click on the **Source icon**, choose **S3**

3. In the **Data source - S3 bucket** node, the specify the following information:

- Node properties tab, **Name** - `Pickup Zone Lookup`
- Data source properties tab, **Database** - `nyctaxi_db`
- Data source properties tab, **Table** - `taxis_zone_lookup`



4. Remember to save your work.

## Modify column names of Pickup Taxi Zone Lookup table

1. Make sure the Amazon S3 - Pickup Zone Lookup node is selected.

2. Click on the Transform icon, choose **Choose Schema**. Formerly known as **Apply Mapping**

3. Specify the following information:

- Node properties tab, **Name** - `ApplyMapping - Pickup Zone Lookup`
- Transform tab, modify the target key
  - **locationid** to `pu_location_id`
  - **borough** to `pu_borough`
  - **zone** to `pu_zone`
  - **service\_zone** to `pu_service_zone`

## Transform NYC Taxi Trip Data

Visual

Script

Job details

Runs

Data quali

+ Add nodes



🔍 change



Use: "change"

Change Schema

Change field names, data types and drop fields. Formerly known as Apply Mapping.

as Apply Mapping.



Join

Combine records from two datasets based on a set of conditions.



SQL Query

Use a SQL query to transform data.



Detect Sensitive Data

Detect PII and other sensitive information.



Evaluate Data Quality

Evaluate the quality and completeness of your data.



Fill Missing Values

Have AWS Glue infer reasonable values for missing data.



Aggregate

Apply functions like sum or average to fields in the dataset.



Custom Transform

Write custom code to transform data.



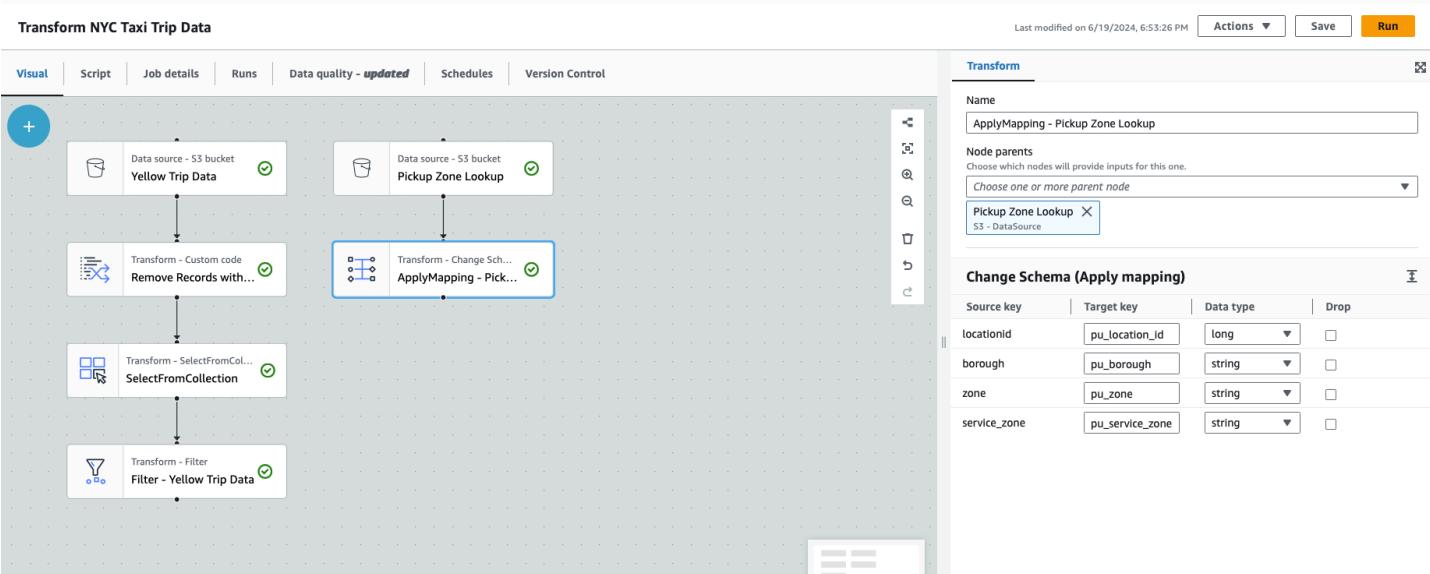
Drop Duplicates

Remove duplicate records from your dataset.



Drop Fields

Add Transforms



4. Remember to save your work.

## Join Data

### Join the Yellow Trips data with Pickup Taxi Zone Lookup data

1. Click on the **Transform** icon, choose **Join**.
2. Specify the following information:
  - o Node properties tab, **Name** - `Yellow Trips Data + Pickup Zone Lookup`
  - o Node properties tab, **Node Parents**
    - `ApplyMapping - Pickup Zone Lookup`
    - `Filter - Yellow Trip Data`

## Transform NYC Taxi Trip Data

Visual

Script

Job details

Runs

Data qualit

+ Add nodes

X



jo|



Use: "jo"



Join

Combine records from two datasets based on a set of conditions.



Change field names,

data types and drop fields. Formerly known as Apply Mapping.



Join

Combine records from two datasets based on a set of conditions.



SQL Query



Use a SQL query to transform data.



Detect Sensitive Data



Detect PII and other sensitive information.



Evaluate Data Quality



Evaluate the quality and completeness of your data.



Fill Missing Values



Have AWS Glue infer reasonable values for missing data.



Aggregate



Apply functions like sum or average to fields in the dataset.



Custom Transform



Write custom code to transform data.



Drop Duplicates



Remove duplicate records from your dataset.



Drop Fields

Add Transforms

## Transform 1

**Name**

Yellow Trips Data + Pickup Zone Lookup

**Node parents**  
Choose which nodes will provide inputs for this one.

Choose one or more parent node

Filter parent nodes

- CustomCode - Transform
- SelectFromCollection
  - SelectFromCollection - Transform
- Filter - Yellow Trip Data
  - Filter - Transform
- ApplyMapping - Pickup Zone Lookup
  - ApplyMapping - Transform

Unclassified nodes

Add condition

Last modified on 6/19/2024, 6:55:46 PM Actions ▾ Save Run

Transform NYC Taxi Trip Data

Visual 1 Script Job details Runs Data quality - updated Schedules Version Control

Transform 1

Name  
Yellow Trips Data + Pickup Zone Lookup

Node parents  
Choose which nodes will provide inputs for this one.  
Choose one or more parent node

ApplyMapping - Pickup Zone Lookup X Filter - Yellow Trip Data X

Join type  
Select the type of join to perform.  
Inner join  
Select all rows from both datasets that meet the join condition.

Join conditions  
Select a field from each parent node for the join condition.  
Add condition

Data preview Output schema

Transform properties tab, under the **Join conditions**, select the following keys:

- **ApplyMapping - Pickup Zone Lookup** - `pu_location_id`
- **Filter - Yellow Trip Data** - `pulocationid`

**Transform**

---

Name

Yellow Trips Data + Pickup Zone Lookup

Node parents

Choose which nodes will provide inputs for this one.

Choose one or more parent node

ApplyMapping - Pickup Zone Lookup X  
ApplyMapping - Transform

Filter - Yellow Trip Data X  
Filter - Transform

---

Join type

Select the type of join to perform.

Inner join  
Select all rows from both datasets that meet the join condition.

Join conditions

Select a field from each parent node for the join condition.

ApplyMapping - Pickup Zone Lookup      Filter - Yellow Trip Data

`pu_location_id` = `pulocationid` Delete

Add condition

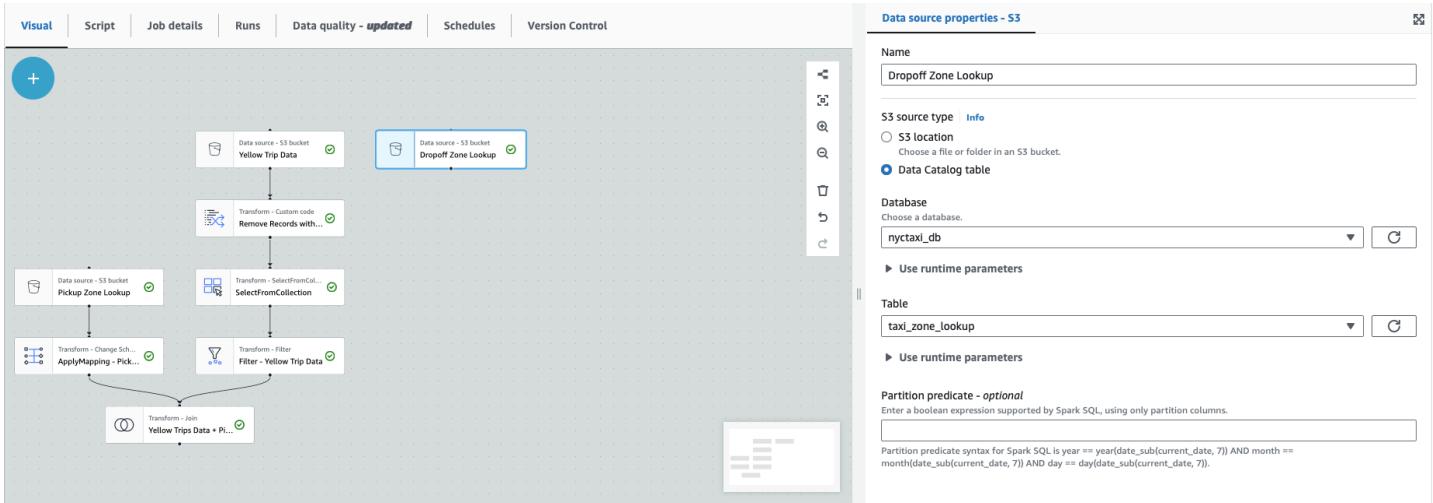
3. Remember to save your work.

## Add and Join another dataset

### Add Lookup table for Taxi Drop-off Zone

1. Click on the **Source** icon, choose **S3**
2. In the Data source - S3 bucket node, the specify the following information:
  - Node properties tab, **Name** - `Dropoff Zone Lookup`

- Data source properties tab, **Database** - `nyctaxi_db`
- Data source properties tab, **Table** - `taxis_zone_lookup`



3. Remember to save your work.

## Modify column names of Drop-off Taxi Zone Lookup Table

1. Make sure the Amazon S3 - Dropoff Zone Lookup node is selected.
2. Click on the **Transform** icon, choose **Change Schema**.
3. Specify the following information:
  - Node properties tab, **Name** - `ApplyMapping - Dropoff Zone Lookup`
  - Transform tab, modify the target key of the following:
    - **locationid** to `do_location_id`
    - **borough** to `do_borough`
    - **zone** to `do_zone`
    - **service\_zone** to `do_service_zone`

## Transform NYC Taxi Trip Data

Visual    Script    Job details    Runs    Data qual

+ Add nodes X

Search sources, transforms and targets

Sources    **Transforms**    Targets    Popular

 **Change Schema**  
Change field names, data types and drop fields. Formerly known as Apply Mapping.

 **Join** Change Schema  
Combine records from two datasets based on a set of conditions.

 **SQL Query**  
Use a SQL query to transform data.

 **Detect Sensitive Data**  
Detect PII and other sensitive information.

 **Evaluate Data Quality**  
Evaluate the quality and completeness of your data.

 **Fill Missing Values**  
Have AWS Glue infer reasonable values for missing data.

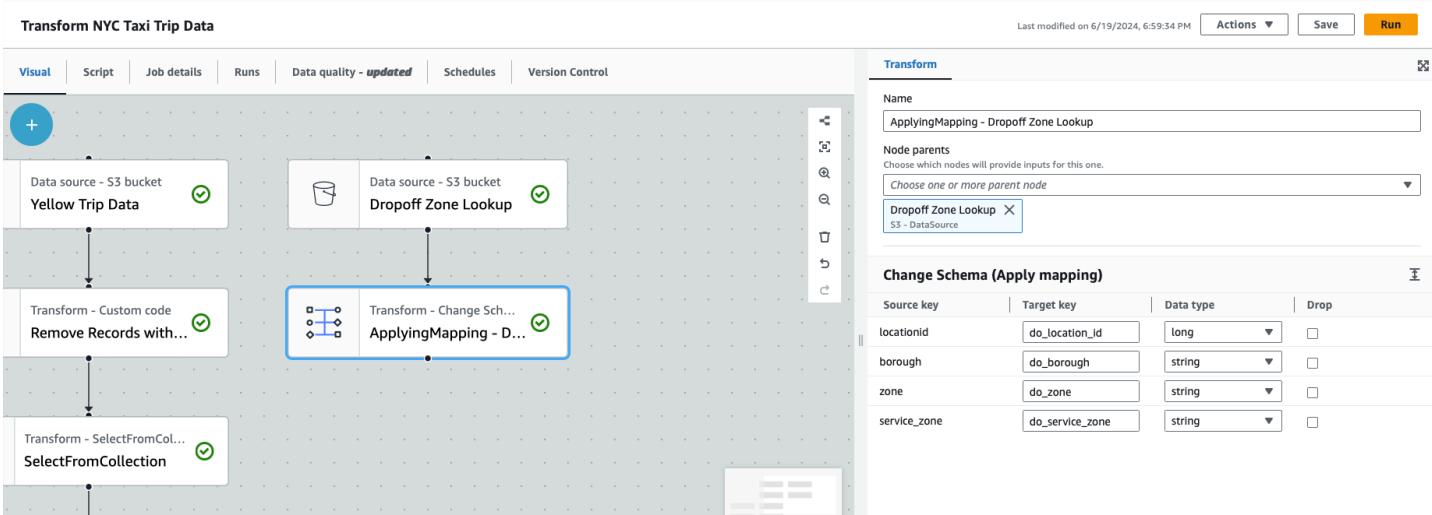
 **Aggregate**  
Apply functions like sum or average to fields in the dataset.

 **Custom Transform**  
Write custom code to transform data.

 **Drop Duplicates**  
Remove duplicate records from your dataset.

 **Drop Fields**

Add Transforms 



4. Remember to save your work.

## Join Yellow Trips data and Dropoff Taxi Zone Lookup data

1. Click on the **Transform** icon, choose **Join**.

2. Specify the following information:

- Node properties tab, **Name** - `Yellow Trips Data + Pickup Zone Lookup + Dropoff Zone Lookup`
- Node properties tab, **Node Parents**
  - `ApplyMapping - Dropoff Zone Lookup`
  - `Yellow Trips Data + Pickup Zone Lookup`

## Transform 1

### Name

Yellow Trips Data + Pickup Zone Lookup + Dropoff Zone Lookup

### Node parents

Choose which nodes will provide inputs for this one.

Choose one or more parent node

Filter parent nodes

SelectFromCollection

SelectFromCollection - Transform

Filter - Yellow Trip Data

Filter - Transform

ApplyMapping - Pickup Zone Lookup

ApplyMapping - Transform

Yellow Trips Data + Pickup Zone Lookup

Join - Transform

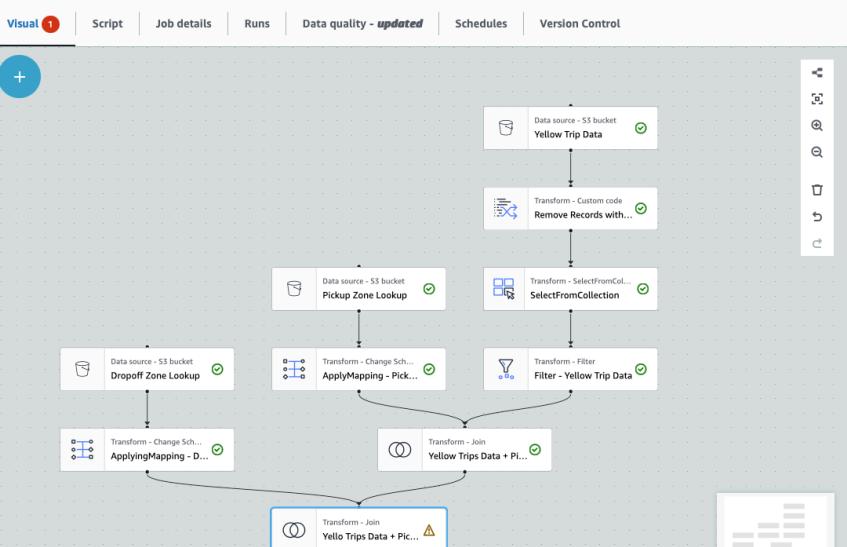
ApplyingMapping - Dropoff Zone Lookup

ApplyMapping - Transform

Unclassified nodes

## Transform NYC Taxi Trip Data

Last modified on 6/19/2024, 7:00:51 PM Actions Save Run



## Transform 1

### Name

Yellow Trips Data + Pickup Zone Lookup + Dropoff Zone Lookup

### Node parents

Choose which nodes will provide inputs for this one.

ApplyingMapping - Dropoff Zone Lookup X

ApplyMapping - Transform

### Join type

Select the type of join to perform.

Inner join

Select all rows from both datasets that meet the join condition.

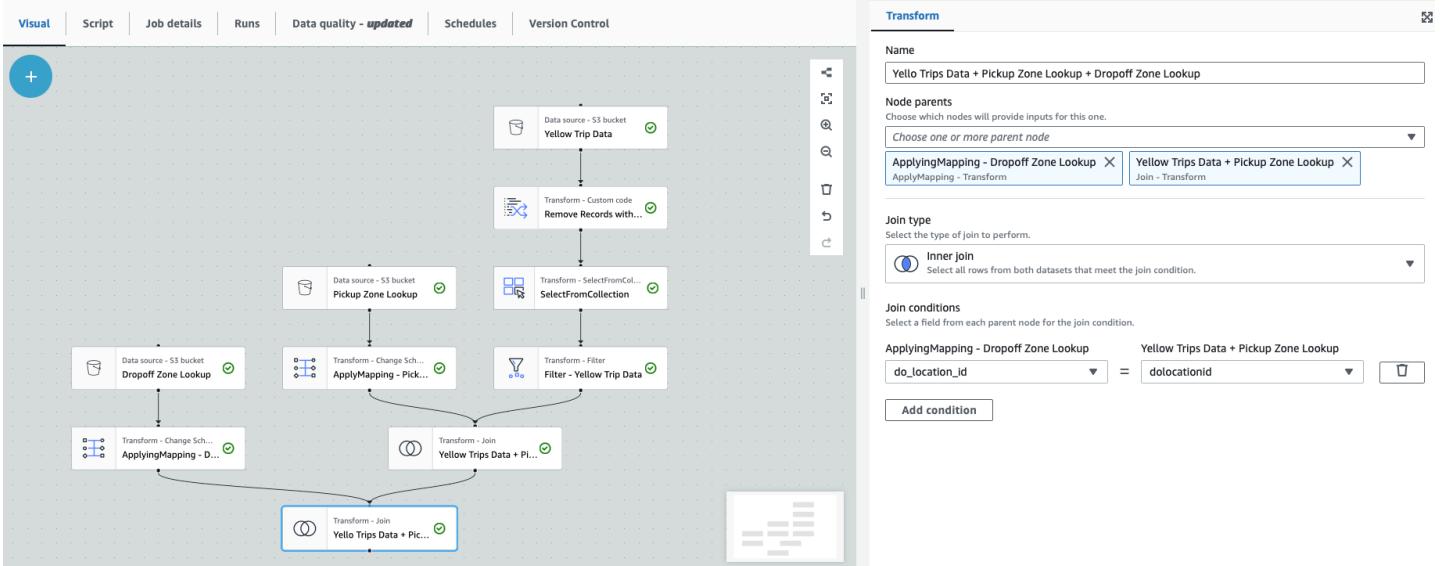
### Join conditions

Select a field from each parent node for the join condition.

Add condition

Transform properties tab, under the **Join conditions**, select the following keys:

- **ApplyMapping - Dropoff Zone Lookup** - `do_location_id`
- **Yellow Trips Data + Pickup Zone Lookup** - `dolocationid`

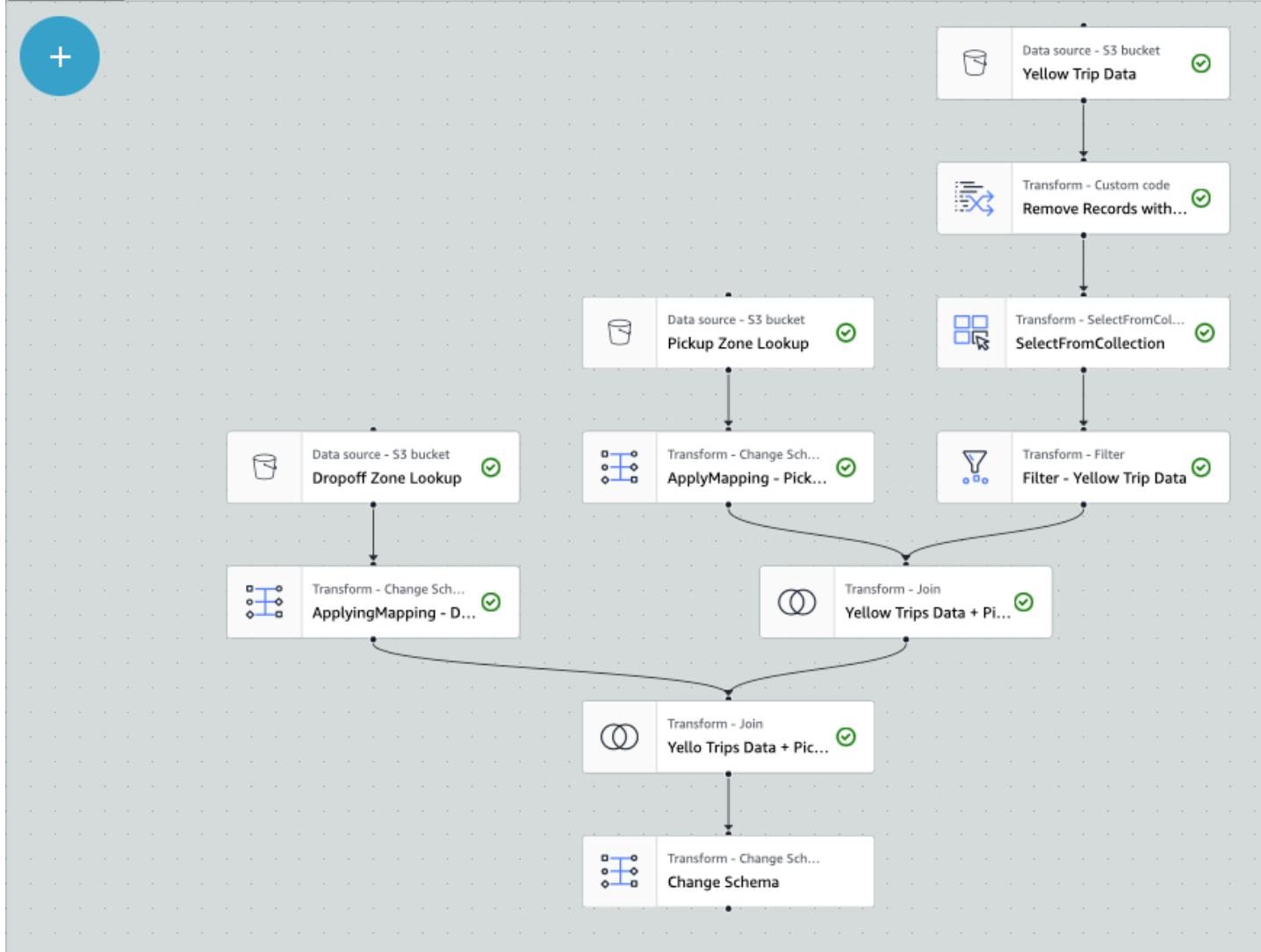


3. Remember to save your work.

## Transform data and save to target

### Modify column names and data types of the joined dataset

1. Click on the Transform icon, choose **Change Schema**.
2. Specify the following information:
  - Node properties tab, **Name** - `ApplyMapping - Joined Data`
  - Transform tab, modify the **Target key** and **Data type** of the following:
    - **vendorid** to `vendor_id`
    - **tpep\_pickup\_datetime** to `pickup_datetime`, **string** to `timestamp`
    - **tpep\_dropoff\_datetime** to `dropoff_datetime`, **string** to `timestamp`
  - Transform tab, drop the following **Source keys**:
    - `pulocationid`
    - `dolocationid`



**Transform**

### Change Schema (Apply mapping)

Source key	Target key	Data type	Drop
pu_borough	pu_borough	string	<input type="checkbox"/>
pu_zone	pu_zone	string	<input type="checkbox"/>
pu_service_zone	pu_service_zone	string	<input type="checkbox"/>
vendorid	vendor_id	long	<input type="checkbox"/>
tpep_pickup_datetime	pickup_datetime	timestamp	<input type="checkbox"/>
tpep_dropoff_datetime	dropoff_datetime	timestamp	<input type="checkbox"/>
passenger_count	passenger_count	long	<input type="checkbox"/>
trip_distance	trip_distance	double	<input type="checkbox"/>
ratecodeid	ratecodeid	long	<input type="checkbox"/>
store_and_fwd_flag	store_and_fwd_flag	string	<input type="checkbox"/>
pulocationid			<input checked="" type="checkbox"/>
dolocationid			<input checked="" type="checkbox"/>
payment_type	payment_type	long	<input type="checkbox"/>
fare_amount	fare_amount	double	<input type="checkbox"/>
extra	extra	double	<input type="checkbox"/>
mta_tax	mta_tax	double	<input type="checkbox"/>
tip_amount	tip_amount	double	<input type="checkbox"/>
tolls_amount	tolls_amount	double	<input type="checkbox"/>
improvement_surcharge	improvement_surchar!	double	<input type="checkbox"/>

3. Remember to save your work.

## Save transformed data to Amazon S3

## Transform NYC Taxi Trip Data

Visual    Script    Job details    Runs    Data quality

+ Add nodes    X

Search sources, transforms and targets

Sources    Transforms    **Targets**    Popular

 AWS Glue Data Catalog  
AWS Glue Data Catalog table as the data target.

 Amazon S3  
S3 bucket by specifying a bucket path as the data target.    Amazon S3

 Amazon Redshift  
Write your data to Amazon Redshift.

 MySQL  
AWS Glue Data Catalog table with MySQL as the data target.

 PostgreSQL  
AWS Glue Data Catalog table with PostgreSQL as the data target.

 Oracle SQL  
AWS Glue Data Catalog table with Oracle SQL as the data target.

 Microsoft SQL Server  
AWS Glue Data Catalog table with SQL Server as the data target.

 Snowflake  
Write your data to Snowflake.

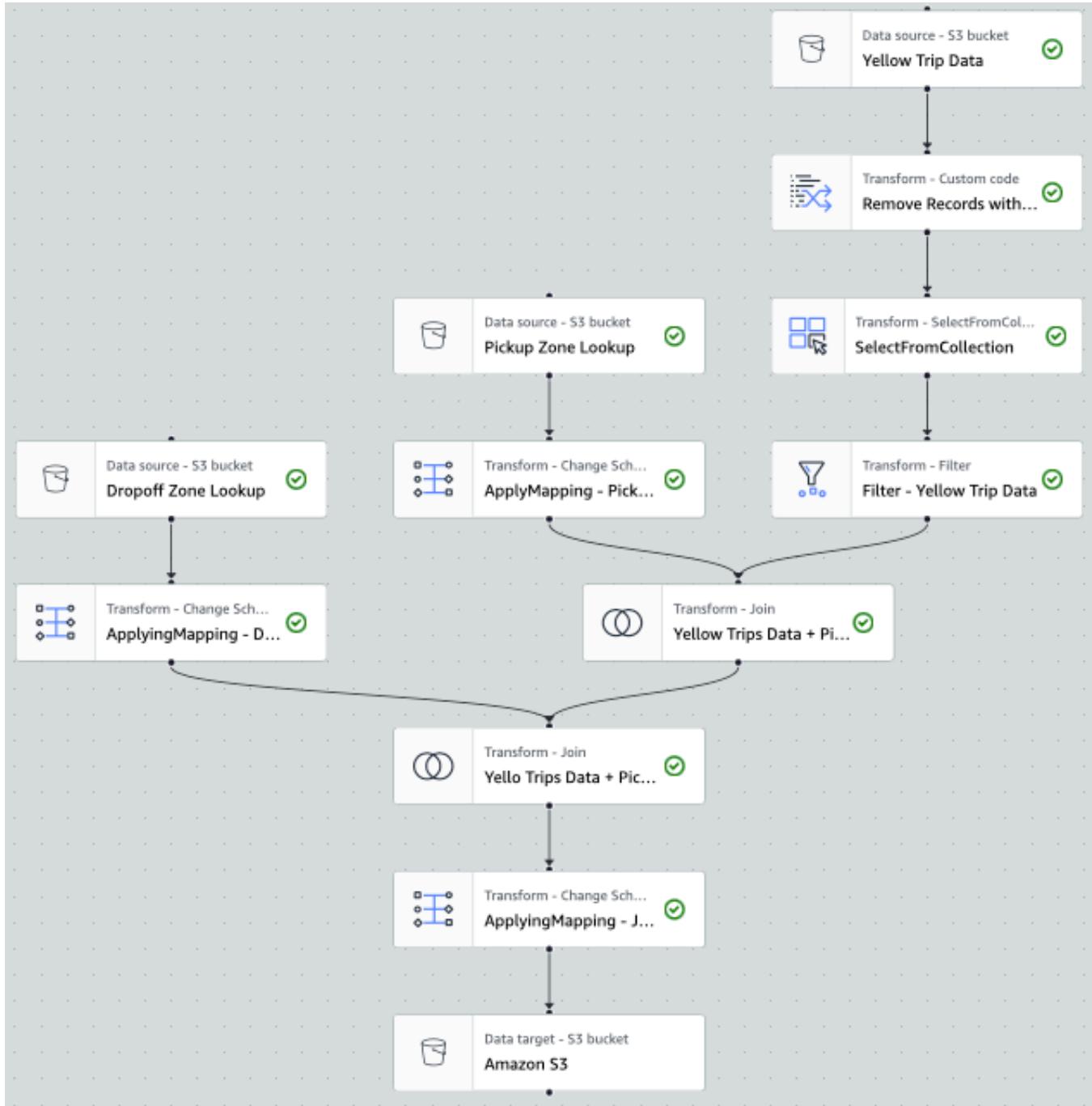
 Google BigQuery  
Write your data to Google BigQuery

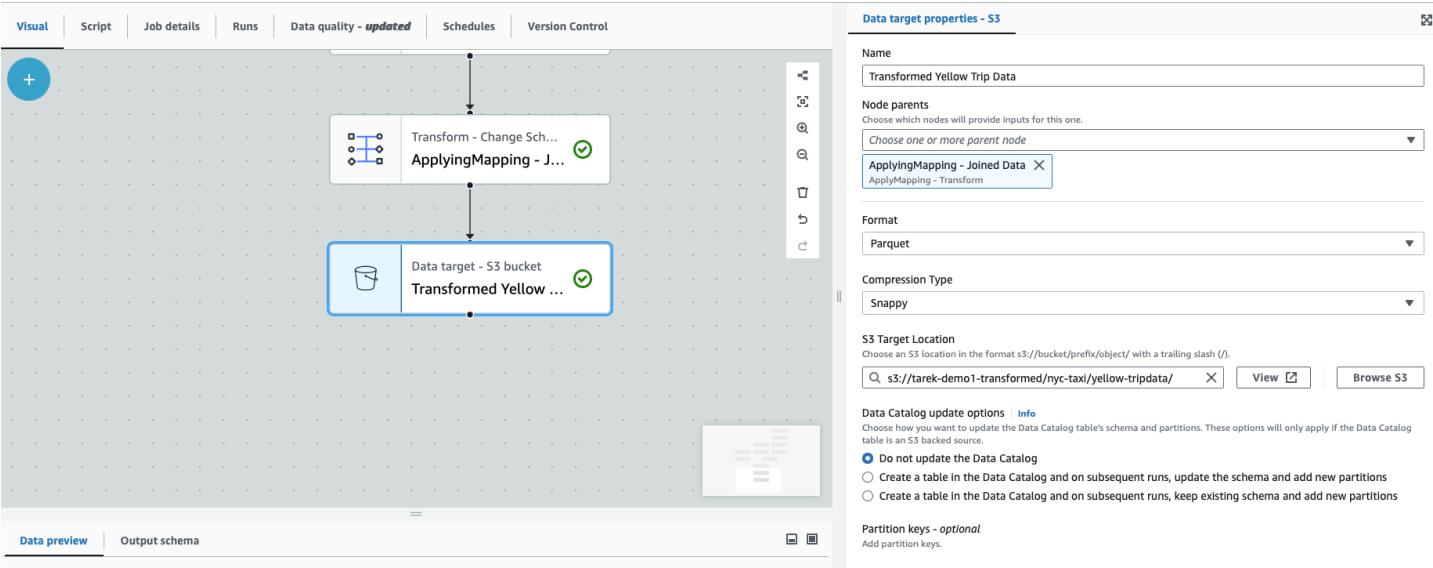
 Teradata Vantage  
Write your data to Teradata Vantage

Manage Connections 

1. Click on **Targets** tab and choose **Amazon S3**
2. Specify the following information:
  - Node properties tab, **Name** – Transformed Yellow Trip Data
  - Data target properties – S3 tab, specify the following:

- **Format** - Glue Parquet
- **Compression Type** - Snappy
- **S3 Target Location** - s3://[your-bucket-name]-transformed/nyc-taxi/yellow-tripdata/



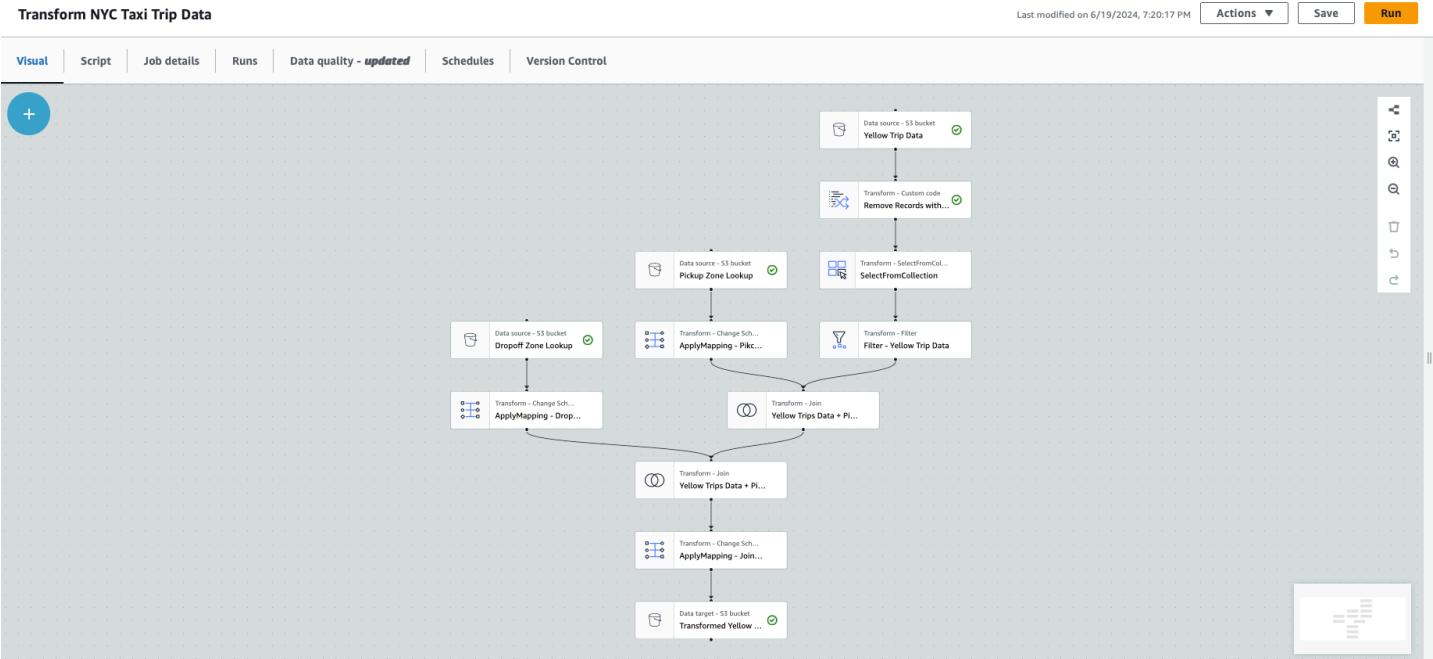


3. Remember to save your work.

## Run the Job

### Run the data transformation job

1. On the upper right section of the AWS Glue Studio page, click **Run**.



2. Go to the **Runs** tab to view the status on of the job.

3. Click on the **Job Id** to monitor the job run.

AWS Services Search [Option+S] Last modified on 6/19/2024, 7:20:17 PM Actions Save Run N. Virginia tarek.katwan @ developintelligence

### Transform NYC Taxi Trip Data

Job runs (1/1) info Last updated (UTC) June 19, 2024 at 16:23:04 View details Stop job run Table View Card View

Filter job runs by property

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
<span>Running</span>	0	06/19/2024 19:20:22	-	2 m 27 s	10 DPUs	G.1X	4.0

Run details Input arguments (10) Continuous logs Run insights Metrics Spark UI Stop job run

Job name <a href="#">Transform NYC Taxi Trip Data</a>	Start time (Local) 06/19/2024 19:20:22	Glue version 4.0	Last modified on (Local) 06/19/2024 19:22:25
Id <a href="#">jr_bf69a599e8c2cdb864fc7ed8b80151c933ad00a68961311a824f5fd - aa4350741</a>	End time (Local)	Worker type G.1X	Log group name /aws-glue/jobs
Run status <span>Running</span>	Start-up time 0	Max capacity 10 DPUs	Number of workers 10
Retry attempt number Initial run	Execution time 2 minutes 27 seconds	Execution class Standard	Timeout 2880 minutes
Trigger name	Security configuration -	Cloudwatch logs • All logs • Output logs • Error logs	Usage profile -

**Run details** **Input arguments (10)** **Continuous logs** **Run insights** **Metrics** **Spark UI**

Job name Start time (Local)  
[Transform NYC Taxi Trip Data](#) 06/19/2024 19:20:22

Id End time (Local)  
[jr\\_bf69a599e8c2cdb864fc7ed8b80151c933ad00a68961311a824f5fd - aa4350741](#)

Run status Start-up time  
Running 0

Retry attempt number Execution time  
Initial run 2 minutes 57 seconds

Trigger name Security configuration  
-

You can scroll down to see additional information about the job run and overall status

AWS Services Search [Option+S] N. Virginia tarekawtan@developintelligence.com

AWS Glue > Monitoring > Job run

## Job Run - jr\_bf69a599e8c2cdb864fc7ed8b80151c933ad00a68961311a824f5fd4a4350741

Run details [Info](#)
[Rewind job bookmark](#) [C](#) [Stop job run](#)

Job name	Id	Run status	Glue version
Transform NYC Taxi Trip Data	jr_bf69a599e8c2cdb864fc7ed8b80151c933ad00a68961311a824f5fd4a4350741	<span>Running</span>	4.0
Retry attempt number	Start time (Local)	End time (Local)	Start time (UTC)
Initial run	06/19/2024 19:20:22	-	2024/06/19 16:20:22
End time (UTC)	Start-up time	Execution time	Last modified on (Local)
-	0	3 minutes 15 seconds	06/19/2024 19:22:25
Last modified on (UTC)	Trigger name	Security configuration	Timeout
2024/06/19 16:22:25	-	-	2880 minutes
Max capacity	Number of workers	Worker type	Execution class
10 DPU	10	G.1X	Standard
Log group name	Cloudwatch logs	Performance and debugging recommendations	Usage profile
/aws-glue/jobs	<ul style="list-style-type: none"> <li>All logs <a href="#">C</a></li> <li>Output logs <a href="#">C</a></li> <li>Error logs <a href="#">C</a></li> </ul>	<a href="#">View in CloudWatch</a>	-

[Continuous logs \[Info\]\(#\)](#)

[Driver logs](#)

Driver and executor log streams [C](#)

```

24/06/19 16:23:40 INFO DAGScheduler: failed: Set()
24/06/19 16:23:40 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
24/06/19 16:23:40 INFO DAGScheduler: ShuffleMapStage 4 (map at DynamicFrame.scala:973) finished in 0.672 s
24/06/19 16:23:40 INFO TaskSetManager: Finished task 0.0 in stage 4.0 (TID 86) in 655 ms on 172.36.35.178 (executor 7) (1/1)
24/06/19 16:23:40 INFO TaskSetManager: Starting task 35.0 in stage 5.0 (TID 122) (172.36.35.178, executor 7, partition 35, ANY, 4694 bytes) taskResourceAssignments Map()
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.34.163.232:43915 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.34.105.73:34087 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on 172.36.35.178:38243 (size: 34.7 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.34.4.7:39015 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.38.32.85:33975 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.38.227.19:45577 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.36.35.178:38243 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.35.189.113:36755 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_9_piece0 in memory on 172.36.35.178:38243 (size: 9.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.35.83.109:34995 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 172.39.164.8:37993 (size: 25.4 KiB, free: 5.8 GiB)
24/06/19 16:23:39 INFO TaskSetManager: Starting task 34.0 in stage 5.0 (TID 121) (172.34.105.73, executor 3, partition 34, ANY, 4694 bytes) taskResourceAssignments Map()
24/06/19 16:23:39 INFO TaskSetManager: Starting task 32.0 in stage 5.0 (TID 119) (172.38.227.19, executor 6, partition 32, ANY, 4694 bytes) taskResourceAssignments Map()

```

[Input documents \(1\)](#)

[Metrics \[Info\]\(#\)](#)

1h 3h 12h 1d 3d 1w Custom UTC timezone C ▾

ETL data movement (bytes)

Data shuffle across executors (bytes)

Memory profile: drivers and executors (%)

CPU load (%)

Job execution: Active executors, completed stages & maximum needed executors

Worker utilization (%)

Once the job is completed you can further inspect the S3 bucket to see the newly populated data "transformed data"

The screenshot shows the AWS Glue Job History interface. At the top, there's a navigation bar with tabs: Visual, Script, Job details, Runs (which is selected), Data quality - *updated*, Schedules, and Version Control. On the right side of the header, there are buttons for Actions, Save, and Run.

Below the header, a section titled "Job runs (1/1) Info" displays a single run. The run was last updated on June 19, 2024 at 16:30:27. It has a status of "Succeeded" with 0 retries. The start time was 06/19/2024 19:20:22 and the end time was 06/19/2024 19:27:07, with a duration of 6 m 30 s. The capacity used was 10 DPUs, worker type was G.1X, and the glue version was 4.0.

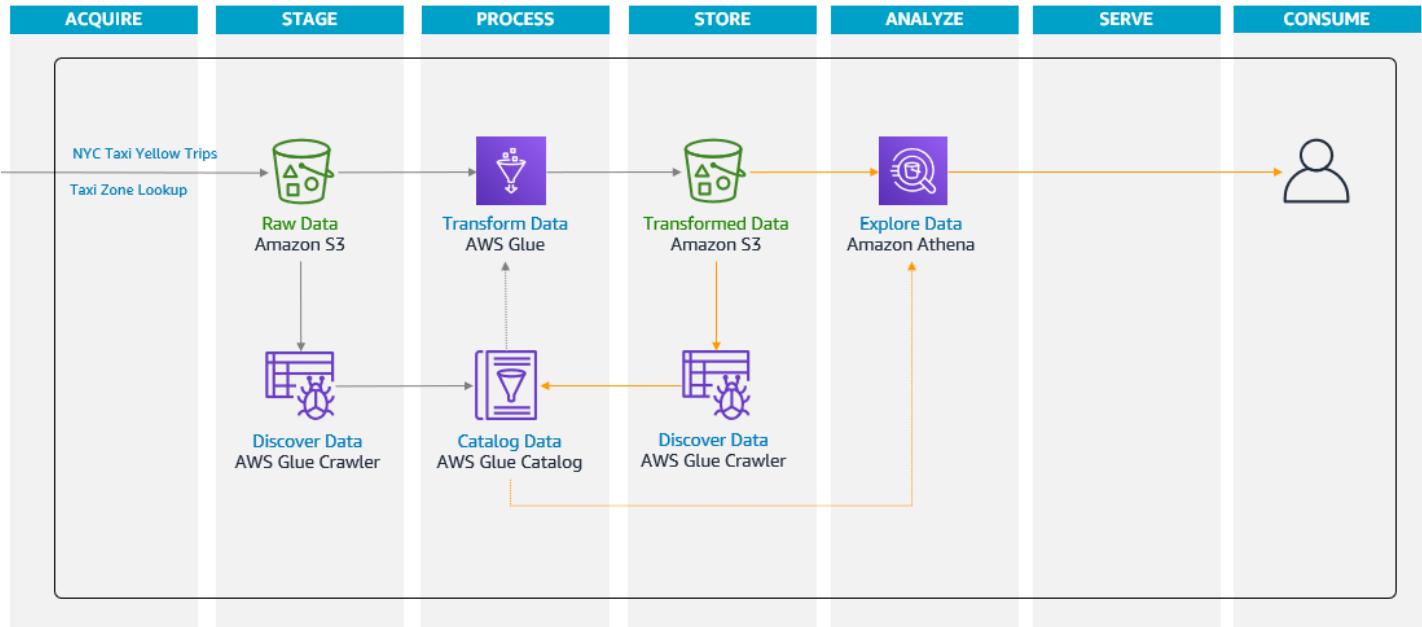
At the bottom of the page, there are tabs for Run details, Input arguments (10), Continuous logs, Run insights, Metrics, and Spark UI. The Run details tab is currently active.

Here is an example of the parquet files in the destination/target bucket for the transformed data

The screenshot shows the AWS S3 console. The left sidebar includes sections for Buckets, Storage Lens, Feature spotlight, and AWS Marketplace for S3. The main area shows the contents of the 'yellow-tripdata/' folder within the 'tarek-demo1-transformed' bucket. The folder contains 36 objects, all of which are parquet files. The files are named in a specific format: 'run-1718814343968-part-block-0-r-00000-snappy.parquet' through 'run-1718814343968-part-block-0-r-00006-snappy.parquet'. The objects were last modified on June 19, 2024, at various times between 19:26:36 and 19:26:55 UTC+03:00. They have sizes ranging from 2.7 MB to 8.0 MB and are stored in the Standard storage class.

## Lab 4 - Enriching your Data

A picture is worth 1000 words. Data visualization presents data in graphical format which makes data easier to digest and understand. It allows users to gain better insights from data to make informed decisions.



## Catalog Transformed Data

### Create a Glue Crawler

1. Go to the AWS Glue Console.
2. In the left navigation menu, click **Crawlers**.
3. On the Crawlers page, click **add crawler**.
4. Specify `nyc-yellow-tripdata-parquet-crawler` as the crawler name, click **Next**.
5. On the Choose data sources and classifiers screen, specify the following information, and then click **Next**.
  - o Click **Add a data source**
  - o Choose Data source – **S3**
  - o Select Location of S3 data – **In this account**
  - o For Subsequent crawler runs, select to **Crawl all sub-folders**
  - o Include S3 path – `s3://your-bucket-name-transformed/nyc-taxi/yellow-tripdata`
  - o For Subsequent crawler runs, select to **Crawl all sub-folders**
  - o Then click **Add an S3 data source**.
6. On the Configure security settings, choose **AWSGlueServiceRole-SDL-Jumpstart** from the Existing IAM role, click **Next**.
7. On the Set output and scheduling screen, choose **nyctaxi\_db** as the database.
8. On the Crawler schedule, leave the frequency **On demand**, click **Next**.
9. Review the crawler details, click **Create crawler**.
10. On the Crawlers page, select **nyc-yellow-tripdata-parquet-crawler**, and then click **Run crawler**.

## Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (3) Info			Last updated (UTC)	Action	Run	Create crawler
			June 19, 2024 at 16:35:58			
			Filter crawlers	< 1 >	Table changes fr...	
Name	State	Schedule	Last run	Last run timestamp	Log	Table changes fr...
<a href="#">nyc-taxi-yellow-trips-csv-crawler</a>	Ready		Succeeded	June 19, 2024 at 15:10:18	<a href="#">View log</a>	1 created
<a href="#">nyc-taxi-zone-lookup-csv-crawler</a>	Ready		Succeeded	June 19, 2024 at 15:15:37	<a href="#">View log</a>	1 created
<a href="#">nyc-yellow-tripdata-parquet-crawler</a>	Ready		Succeeded	June 19, 2024 at 16:32:44	<a href="#">View log</a>	1 created

## Validate transformed data

1. Go to Athena console.
2. Select **Preview table** to examine the data in **yellow\_tripdata** table.

The screenshot shows the AWS Athena console interface. On the left, there's a sidebar with 'Data source' set to 'AwsDataCatalog' and 'Database' set to 'nyctaxi\_db'. Below that is a 'Tables and views' section with a 'Create' button. Under 'Tables', there are three entries: 'raw\_yellow\_tripdata', 'taxi\_zone\_lookup', and 'yellow\_tripdata'. Under 'Views', there are zero. A context menu is open over the 'yellow\_tripdata' table, listing options like 'Run Query', 'Preview Table' (which is highlighted with a blue border), 'Generate table DDL', 'Insert', 'Insert into editor', 'Manage', 'Delete table', 'Generate statistics - new', 'View properties', 'View in Glue', and 'plain'. At the bottom right of the main area, there are tabs for 'Query results' and 'Query stat'.

3. On the Query editor page, run the following queries to examine the data.

```
1 -- total records
2 -- 4347658
3 SELECT COUNT(*) "Count"
4 FROM yellow_tripdata;
```

```
1 -- 2020-10-01 1575353
2 -- 2020-11-01 1409851
3 -- 2020-12-01 1362454
4 SELECT DATE_TRUNC('month', pickup_datetime) "Period",
5      COUNT(*) "Total Records"
6 FROM yellow_tripdata
7 GROUP BY DATE_TRUNC('month', pickup_datetime)
8 ORDER BY 1;
```

## Enrich Transformed Data

1. Run the following query to create a view to enrich the table with additional data.

```
1 CREATE OR REPLACE VIEW v_yellow_tripdata
2 AS
3 SELECT CASE vendor_id
4       WHEN 1 THEN 'Creative Mobile'
5       WHEN 2 THEN 'VeriFone'
6       ELSE 'No Data'
7     END "vendor_name",
8     pickup_datetime,
9     dropoff_datetime,
10    passenger_count,
11    trip_distance,
12    CASE ratecodeid
13      WHEN 1 THEN 'Standard Rate'
14      WHEN 2 THEN 'JFK'
15      WHEN 3 THEN 'Newark'
16      WHEN 4 THEN 'Nassau/Westchester'
17      WHEN 5 THEN 'Negotiated Fare'
18      WHEN 6 THEN 'Group Ride'
19      WHEN 99 THEN 'Special Rate'
20      ELSE 'No Data'
21    END "rate_type",
22    store_and_fwd_flag,
23    pu_borough,
24    pu_zone,
25    pu_service_zone,
26    do_borough,
```

```
27      do_zone,  
28      do_service_zone,  
29      CASE payment_type  
30          WHEN 1 THEN 'Credit Card'  
31          WHEN 2 THEN 'Cash'  
32          WHEN 3 THEN 'No Charge'  
33          WHEN 4 THEN 'Dispute'  
34          WHEN 5 THEN 'Unknown'  
35          WHEN 6 THEN 'Voided Trip'  
36          ELSE 'No Data'  
37      END "payment_type",  
38      fare_amount,  
39      extra,  
40      mta_tax,  
41      tip_amount,  
42      tolls_amount,  
43      improvement_surcharge,  
44      congestion_surcharge,  
45      total_amount  
46  FROM yellow_tripdata;  
47
```

2. Select **Preview table** to examine the enriched data in **v\_yellow\_tripdata** view.

**Data** C < Query 1 : X | Q

Data source: AwsDataCatalog

Database: nyctaxi\_db

Tables and views Create ▾ ⚙

Filter tables and views

▶ Tables (3) < 1 >

▼ Views (1) < 1 >

[+] v\_yellow\_tripdata ...

43  
44  
45  
46 FROM yel  
47

SQL Ln 47, Col 1

Run again

Run Query

Preview View

Insert

Insert into editor

Manage

Delete view

View properties

Show/edit query

Amazon Athena > Query editor

Editor | Recent queries | Saved queries | Settings | Workgroup primary

**Data** | [Query 1](#) | [Query 2](#) | [Query 3](#) | [Query 4](#) | [Query 5](#) | [Query 6](#) | [Query 7](#)

```
1 SELECT * FROM "nyctaxi_db"."v_yellow_tripdata" limit 10;
```

Data source: AwsDataCatalog | Database: nyctaxi\_db

Tables and views | Create | [Tables \(3\)](#) | [Views \(1\)](#) | SQL | Ln 1, Col 1 | Run again | Explain | Cancel | Clear | Create | Reuse query results up to 60 minutes ago

Query results | Query stats | Completed | Time in queue: 64 ms | Run time: 850 ms | Data scanned: 875.48 KB | Copy | Download results | < 1 >

#	vendor_name	pickup_datetime	dropoff_datetime	passenger_count	trip_distance	rate_type	store_and_fwd_flag	pu_borough	pu_zone	pu_service_zone	do_borough	do_zone
1	VeriFone	2020-12-27 13:08:51	2020-12-27 13:30:02	2	10.84	Standard Rate	N	Manhattan	TriBeCa/Civic Center	Yellow Zone	Staten Island	Arrochar/Fort Wadsworth
2	VeriFone	2020-12-08 20:55:49	2020-12-08 21:53:55	1	30.31	Standard Rate	N	Brooklyn	Sheepshead Bay	Boro Zone	Staten Island	Arrochar/Fort Wadsworth
3	VeriFone	2020-12-27 12:54:29	2020-12-27 13:02:19	1	1.15	Standard Rate	N	Staten Island	Arrochar/Fort Wadsworth	Boro Zone	Staten Island	Arrochar/Fort Wadsworth
4	VeriFone	2020-11-08 12:53:32	2020-11-08 13:00:42	1	0.97	Standard Rate	N	Staten Island	Arrochar/Fort Wadsworth	Boro Zone	Staten Island	Arrochar/Fort Wadsworth
5	VeriFone	2020-10-15 15:28:30	2020-10-15 16:36:18	1	20.21	Standard Rate	N	Queens	LaGuardia Airport	Airports	Staten Island	Arrochar/Fort Wadsworth
6	Creative Mobile	2020-12-23 17:53:31	2020-12-23 18:52:29	1	14.7	Standard Rate	N	Manhattan	Penn Station/Madison Sq West	Yellow Zone	Staten Island	Arrochar/Fort Wadsworth
7	VeriFone	2020-10-30 02:14:59	2020-10-30 02:48:03	1	15.75	Standard Rate	N	Manhattan	Penn Station/Madison Sq West	Yellow Zone	Staten Island	Arrochar/Fort Wadsworth
8	VeriFone	2020-10-31 02:16:19	2020-10-31 02:49:58	1	15.75	Standard Rate	N	Manhattan	Penn Station/Madison Sq West	Yellow Zone	Staten Island	Arrochar/Fort Wadsworth

Feedback | © 2024, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookies

### 3. Run the following query to get insights.

```

1  SELECT vendor_name "Vendor",
2      rate_type "Rate Type",
3      payment_type "Payment Type",
4      ROUND(AVG(fare_amount), 2) "Fare",
5      ROUND(AVG(extra), 2) "Extra",
6      ROUND(AVG(mta_tax), 2) "MTA",
7      ROUND(AVG(tip_amount), 2) "Tip",
8      ROUND(AVG(tolls_amount), 2) "Toll",
9      ROUND(AVG(improvement_surcharge), 2) "Improvement",
10     ROUND(AVG(congestion_surcharge), 2) "Congestion",
11     ROUND(AVG(total_amount), 2) "Total"
12   FROM v_yellow_tripdata
13 GROUP BY vendor_name,
14          rate_type,
15          payment_type
16 ORDER BY 1, 2, 3;
17

```