Table of Contents

[Landing Zone 5](#_Toc208223053)

[Use Glue Studio to ingest data from an S3 bucket 5](#_Toc208223054)

[customer\_landing\_to\_trusted.py 5](#_Toc208223055)

[Visual ETL 6](#_Toc208223056)

[accelerometer\_landing\_to\_trusted.py 7](#_Toc208223057)

[Visual ETL 8](#_Toc208223058)

[step\_trainer\_trusted.py 9](#_Toc208223059)

[Visual ETL 10](#_Toc208223060)

[Manually create a Glue Table using Glue Console from JSON data 12](#_Toc208223061)

[customer\_landing.sql 12](#_Toc208223062)

[accelerometer\_landing.sql 13](#_Toc208223063)

[step\_trainer\_landing.sql 14](#_Toc208223064)

[Use Athena to query the Landing Zone. 14](#_Toc208223065)

[Count of customer\_landing 15](#_Toc208223066)

[Count of accelerometer\_landing 15](#_Toc208223067)

[Count of step\_trainer\_landing 16](#_Toc208223068)

[Trusted Zone 16](#_Toc208223069)

[Configure Glue Studio to dynamically update a Glue Table schema from JSON data 16](#_Toc208223070)

[query Trusted 16](#_Toc208223071)

[Customer Trusted 17](#_Toc208223072)

[Count of customer\_trusted 18](#_Toc208223073)

[Accelerometer Trusted 19](#_Toc208223074)

[Count of accelerometer\_trusted 20](#_Toc208223075)

[Count of step\_trainer\_trusted 21](#_Toc208223076)

[Curated Zone 22](#_Toc208223077)

[Write a Glue Job to join trusted data 22](#_Toc208223078)

[customer\_trusted\_to\_curated.py 22](#_Toc208223079)

[Visual ETL 24](#_Toc208223080)

[Customer Trusted To Curated 24](#_Toc208223081)

[Write a Glue Job to create curated data 25](#_Toc208223082)

[step\_trainer\_trusted.py 25](#_Toc208223083)

[Visual ETL 27](#_Toc208223084)

[Step Trainer Trusted 27](#_Toc208223085)

[Use Athena to query Curated Glue Tables 28](#_Toc208223086)

[Count of customer\_curated 28](#_Toc208223087)

[Machine Learning Curated – Visual ETL 29](#_Toc208223088)

[Count of machine\_learning\_curated 29](#_Toc208223089)

[machine\_learning\_curated.py 30](#_Toc208223090)

[1. Introduction 32](#_Toc208223091)

[2. Code Walkthrough 32](#_Toc208223092)

[2.1 Imports and Setup 32](#_Toc208223093)

[2.2 Job Initialization 32](#_Toc208223094)

[2.3 Data Quality Rules 33](#_Toc208223095)

[2.4 Load Customer Landing Data 33](#_Toc208223096)

[2.5 Privacy Filter 33](#_Toc208223097)

[2.6 Data Quality Evaluation 34](#_Toc208223098)

[2.7 Write Customer Trusted Data 34](#_Toc208223099)

[2.8 Job Commit 34](#_Toc208223100)

[AWS Glue ETL Job – Accelerometer Trusted Zone 35](#_Toc208223101)

[accelerometer\_landing\_to\_trusted.py 35](#_Toc208223102)

[1. Introduction 35](#_Toc208223103)

[2. Code Walkthrough 35](#_Toc208223104)

[2.1 Imports and Setup 35](#_Toc208223105)

[2.2 Spark SQL Helper Function 35](#_Toc208223106)

[2.3 Job Initialization 36](#_Toc208223107)

[2.4 Data Quality Rules 36](#_Toc208223108)

[2.5 Load Source Data 36](#_Toc208223109)

[2.6 Join Accelerometer with Customer 37](#_Toc208223110)

[2.7 SQL Transformation 37](#_Toc208223111)

[2.8 Data Quality Evaluation 37](#_Toc208223112)

[2.9 Write Accelerometer Trusted Data 38](#_Toc208223113)

[2.10 Job Commit 38](#_Toc208223114)

[4. Key Takeaways 38](#_Toc208223115)

[AWS Glue ETL Job – Step Trainer Trusted Zone 39](#_Toc208223116)

[1. Introduction 39](#_Toc208223117)

[2. Code Walkthrough 39](#_Toc208223118)

[2.1 Imports and Setup 39](#_Toc208223119)

[2.2 Spark SQL Helper 39](#_Toc208223120)

[2.3 Job Initialization 40](#_Toc208223121)

[2.4 Data Quality Rules 40](#_Toc208223122)

[2.5 Load Source Data 40](#_Toc208223123)

[2.6 SQL Transformation 41](#_Toc208223124)

[2.7 Data Quality Check 41](#_Toc208223125)

[2.8 Write Final Data to S3 41](#_Toc208223126)

[2.9 Commit Job 42](#_Toc208223127)

[4. Key Takeaways 42](#_Toc208223128)

[AWS Glue ETL Job – Customers Curated Zone 43](#_Toc208223129)

[1. Introduction 43](#_Toc208223130)

[2. Code Walkthrough 43](#_Toc208223131)

[2.1 Imports and Setup 43](#_Toc208223132)

[2.2 Spark SQL Helper 43](#_Toc208223133)

[2.3 Job Initialization 43](#_Toc208223134)

[2.4 Data Quality Rules 44](#_Toc208223135)

[2.5 Load Source Data 44](#_Toc208223136)

[2.6 Join 45](#_Toc208223137)

[2.7 SQL Transformation 45](#_Toc208223138)

[2.8 Data Quality Check 45](#_Toc208223139)

[2.9 Write Final Data to S3 46](#_Toc208223140)

[4. Key Takeaways 46](#_Toc208223141)

[Machine Learning Curated Glue Job Documentation 47](#_Toc208223142)

[1. Purpose 47](#_Toc208223143)

[2. Code Walkthrough 47](#_Toc208223144)

[1. Import Required Libraries 47](#_Toc208223145)

[2. Initialize Job 47](#_Toc208223146)

[3. Define Data Quality Rules 47](#_Toc208223147)

[4. Load Source Datasets (Trusted Zone) 48](#_Toc208223148)

[5. Rename Customer Keys Before Join 48](#_Toc208223149)

[6. Join Step Trainer with Accelerometer 48](#_Toc208223150)

[7. Join with Customer Data 49](#_Toc208223151)

[8. Evaluate Data Quality 49](#_Toc208223152)

[9. Write Curated Dataset to S3 + Catalog 49](#_Toc208223153)

[10. Commit Job 50](#_Toc208223154)

[3. Key Takeaways 50](#_Toc208223155)

# Landing Zone

## Use Glue Studio to ingest data from an S3 bucket

### Submission Requirements

**customer\_landing\_to\_trusted.py**, **accelerometer\_landing\_to\_trusted.py**, and **step\_trainer\_trusted.py** Glue jobs have a node that connects to S3 bucket for customer, accelerometer, and step trainer landing zones.

### customer\_landing\_to\_trusted.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

import re

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node Amazon S3

AmazonS3\_node1757133904216 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/landing/"], "recurse": True}, transformation\_ctx="AmazonS3\_node1757133904216")

# Script generated for node Privacy Filter

PrivacyFilter\_node1757133969699 = Filter.apply(frame=AmazonS3\_node1757133904216, f=lambda row: (not(row["shareWithResearchAsOfDate"] == 0)), transformation\_ctx="PrivacyFilter\_node1757133969699")

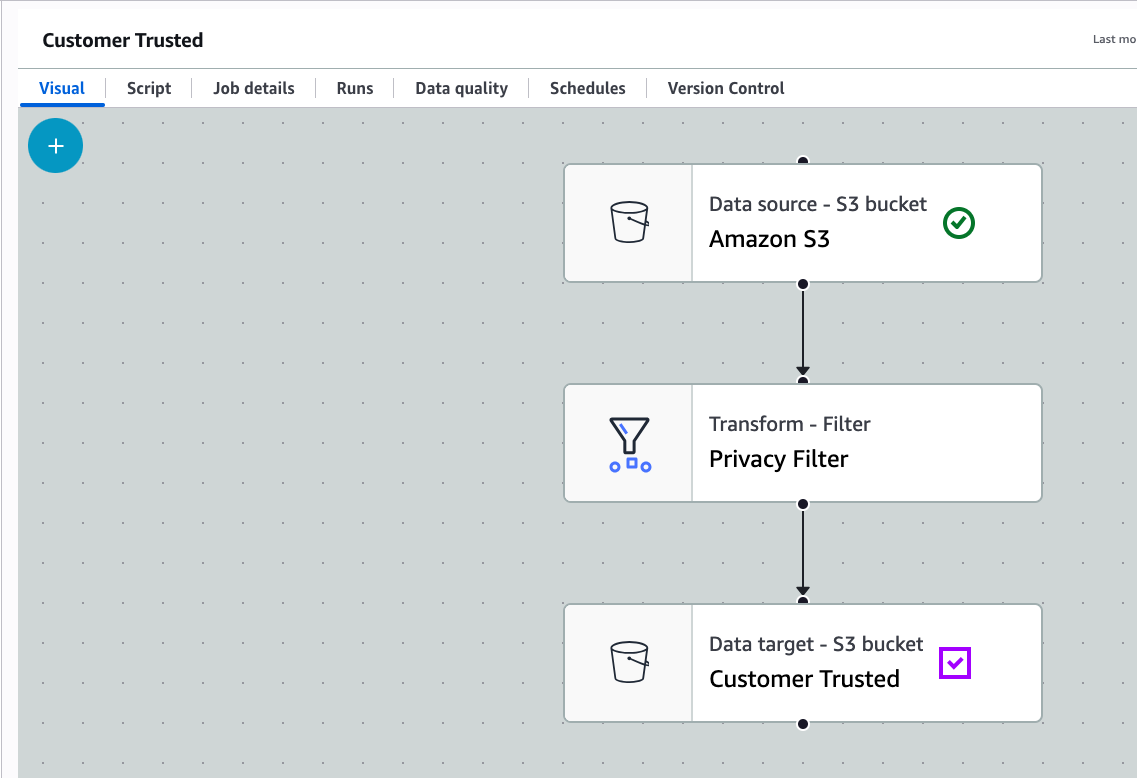
# Script generated for node Customer Trusted

EvaluateDataQuality().process\_rows(frame=PrivacyFilter\_node1757133969699, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757128621835", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

CustomerTrusted\_node1757134020703 = glueContext.write\_dynamic\_frame.from\_options(frame=PrivacyFilter\_node1757133969699, connection\_type="s3", format="json", connection\_options={"path": "s3://stedi-lake-house-qk/customer/trusted/", "partitionKeys": []}, transformation\_ctx="CustomerTrusted\_node1757134020703")

job.commit()

### Visual ETL



### accelerometer\_landing\_to\_trusted.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node Amazon S3

AmazonS3\_node1757128664531 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="accelorometer\_landing", transformation\_ctx="AmazonS3\_node1757128664531")

# Script generated for node Amazon S3

AmazonS3\_node1757128643733 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/trusted/"], "recurse": True}, transformation\_ctx="AmazonS3\_node1757128643733")

# Script generated for node Customer Privacy Filter

CustomerPrivacyFilter\_node1757128680082 = Join.apply(frame1=AmazonS3\_node1757128664531, frame2=AmazonS3\_node1757128643733, keys1=["user"], keys2=["email"], transformation\_ctx="CustomerPrivacyFilter\_node1757128680082")

# Script generated for node SQL Query

SqlQuery914 = '''

select user, x, y, z, timestamp from myDataSource

'''

SQLQuery\_node1757135173003 = sparkSqlQuery(glueContext, query = SqlQuery914, mapping = {"myDataSource":CustomerPrivacyFilter\_node1757128680082}, transformation\_ctx = "SQLQuery\_node1757135173003")

# Script generated for node Accelerometer Trusted

EvaluateDataQuality().process\_rows(frame=SQLQuery\_node1757135173003, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757128621835", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

AccelerometerTrusted\_node1757128772635 = glueContext.write\_dynamic\_frame.from\_options(frame=SQLQuery\_node1757135173003, connection\_type="s3", format="json", connection\_options={"path": "s3://stedi-lake-house-qk/accelerometer/trusted/", "partitionKeys": []}, transformation\_ctx="AccelerometerTrusted\_node1757128772635")

job.commit()

### Visual ETL

A screenshot of a computer

Description automatically generated

### step\_trainer\_trusted.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node AWS Glue Data Catalog

AWSGlueDataCatalog\_node1757143330187 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="customer\_curated", transformation\_ctx="AWSGlueDataCatalog\_node1757143330187")

# Script generated for node Step Trainer Landing

StepTrainerLanding\_node1757143347320 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/step\_trainer/landing/"], "recurse": True}, transformation\_ctx="StepTrainerLanding\_node1757143347320")

# Script generated for node SQL Query

SqlQuery919 = '''

SELECT st.serialnumber,

sensorreadingtime,

FIRST\_VALUE(distancefromobject)

OVER (PARTITION BY st.serialnumber, st.sensorreadingtime ORDER BY st.sensorreadingtime) AS distancefromobject

FROM stedi.step\_trainer\_landing st

JOIN customer\_curated c

ON c.serialnumber = st.serialnumber;

'''

SQLQuery\_node1757143772990 = sparkSqlQuery(glueContext, query = SqlQuery919, mapping = {"customer\_curated":AWSGlueDataCatalog\_node1757143330187, "step\_trainer\_landing":StepTrainerLanding\_node1757143347320}, transformation\_ctx = "SQLQuery\_node1757143772990")

# Script generated for node Amazon S3

EvaluateDataQuality().process\_rows(frame=SQLQuery\_node1757143772990, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757139860297", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

AmazonS3\_node1757143463027 = glueContext.getSink(path="s3://stedi-lake-house-qk/step\_trainer/final\_output/", connection\_type="s3", updateBehavior="UPDATE\_IN\_DATABASE", partitionKeys=[], enableUpdateCatalog=True, transformation\_ctx="AmazonS3\_node1757143463027")

AmazonS3\_node1757143463027.setCatalogInfo(catalogDatabase="stedi",catalogTableName="step\_trainer\_trusted")

AmazonS3\_node1757143463027.setFormat("json")

AmazonS3\_node1757143463027.writeFrame(SQLQuery\_node1757143772990)

job.commit()

### Visual ETL

### A screenshot of a computer Description automatically generated

## Manually create a Glue Table using Glue Console from JSON data

SQL DDL scripts **customer\_landing.sql**, **accelerometer\_landing.sql**, and **step\_trainer\_landing.sql** include all of the JSON fields in the data input files and are appropriately typed (not everything is a string).

### customer\_landing.sql

CREATE EXTERNAL TABLE `customer\_landing`(

`customername` string COMMENT 'from deserializer',

`email` string COMMENT 'from deserializer',

`phone` string COMMENT 'from deserializer',

`birthday` string COMMENT 'from deserializer',

`serialnumber` string COMMENT 'from deserializer',

`registrationdate` bigint COMMENT 'from deserializer',

`lastupdatedate` bigint COMMENT 'from deserializer',

`sharewithresearchasofdate` bigint COMMENT 'from deserializer',

`sharewithpublicasofdate` bigint COMMENT 'from deserializer',

`sharewithfriendsasofdate` bigint COMMENT 'from deserializer')

ROW FORMAT SERDE

'org.openx.data.jsonserde.JsonSerDe'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

's3://stedi-lake-house-qk/customer/landing/'

TBLPROPERTIES (

'classification'='json')

### accelerometer\_landing.sql

CREATE EXTERNAL TABLE `accelorometer\_landing`(

`user` string COMMENT 'from deserializer',

`timestamp` bigint COMMENT 'from deserializer',

`x` float COMMENT 'from deserializer',

`y` float COMMENT 'from deserializer',

`z` float COMMENT 'from deserializer')

ROW FORMAT SERDE

'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES (

'case.insensitive'='TRUE',

'dots.in.keys'='FALSE',

'ignore.malformed.json'='FALSE',

'mapping'='TRUE')

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

's3://stedi-lake-house-qk/accelerometer/landing'

TBLPROPERTIES (

'classification'='json',

'transient\_lastDdlTime'='1757062449')

### step\_trainer\_landing.sql

CREATE EXTERNAL TABLE `step\_trainer\_landing`(

`sensorreadingtime` bigint COMMENT 'from deserializer',

`serialnumber` string COMMENT 'from deserializer',

`distancefromobject` int COMMENT 'from deserializer')

ROW FORMAT SERDE

'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES (

'case.insensitive'='TRUE',

'dots.in.keys'='FALSE',

'ignore.malformed.json'='FALSE',

'mapping'='TRUE')

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

's3://stedi-lake-house-qk/step\_trainer/landing'

TBLPROPERTIES (

'classification'='json',

'transient\_lastDdlTime'='1757132555')

## Use Athena to query the Landing Zone.

Include screenshots showing various queries run on Athena, along with their results:

* Count of customer\_landing: 956 rows
* The customer\_landing data contains multiple rows with a blank shareWithResearchAsOfDate.
* Count of accelerometer\_landing: 81273 rows
* Count of step\_trainer\_landing: 28680 rows

### Count of customer\_landing

A screenshot of a computer

Description automatically generated

### Count of accelerometer\_landing

A screenshot of a computer

Description automatically generated

### Count of step\_trainer\_landing

A screenshot of a computer

Description automatically generated

# Trusted Zone

## Configure Glue Studio to dynamically update a Glue Table schema from JSON data

Glue Job Python code shows that the option to dynamically infer and update schema is enabled.

To do this, set the *Create a table in the Data Catalog and, on subsequent runs, update the schema and add new partitions* option to True.

## query Trusted

Include screenshots showing various queries run on Athena, along with their results:

* Count of customer\_trusted: 482 rows
  + The resulting customer\_trusted data has no rows where shareWithResearchAsOfDate is blank.
* Count of accelerometer\_trusted: 40981 rows
* Count of step\_trainer\_trusted: 14460 rows

However, **if you are following the stand-out suggestions**, your row counts should be as follows:

* Count of customer\_trusted: 482 rows
* The resulting customer\_trusted data has no rows where shareWithResearchAsOfDate is blank.
* Count of accelerometer\_trusted: 32025 rows
* Count of step\_trainer\_trusted: 14460 rows

### Customer Trusted

Sanitize the Customer data from the Website (Landing Zone) and only store the Customer Records who agreed to share their data for research purposes (Trusted Zone) - creating a Glue Table called **customer\_trusted**.

A screenshot of a computer

Description automatically generated

### Count of customer\_trusted

A screenshot of a computer

Description automatically generated

### Accelerometer Trusted

A screenshot of a computer

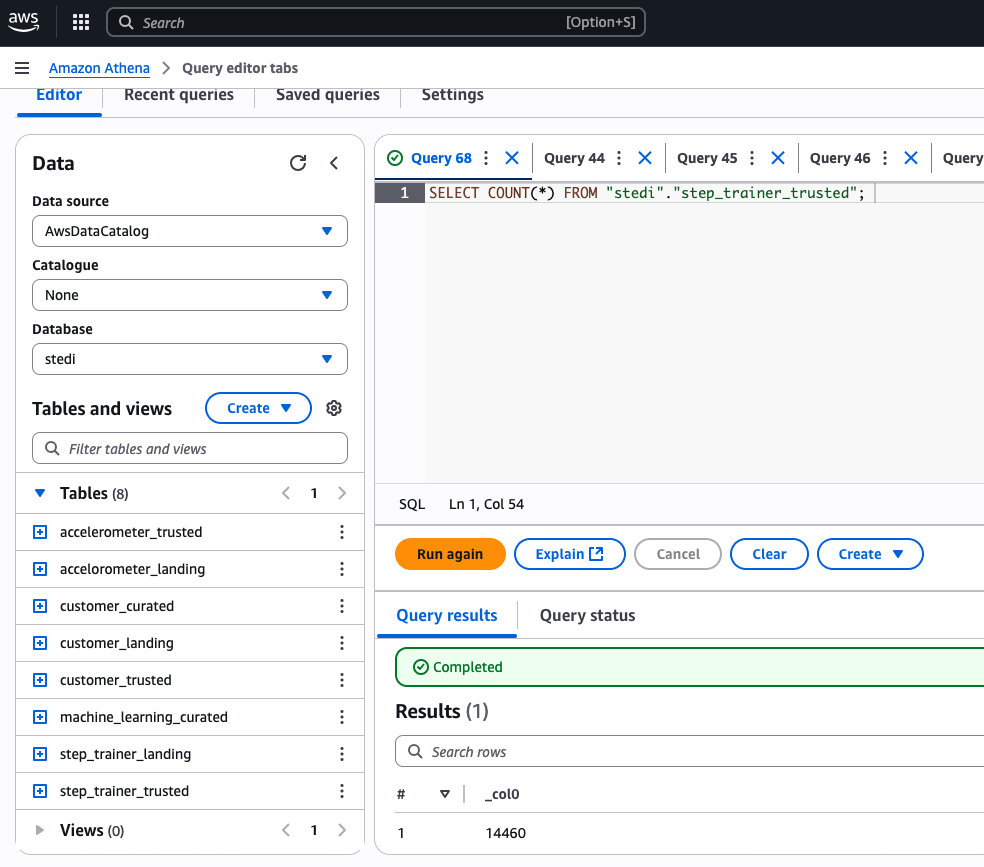
Description automatically generated

### Count of accelerometer\_trusted

A screenshot of a computer

Description automatically generated

### Count of step\_trainer\_trusted



# Curated Zone

## Write a Glue Job to join trusted data

**customer\_trusted\_to\_curated.py** has a node that inner joins the customer\_trusted data with the accelerometer\_trusted data by emails. The produced table should have only columns from the customer table.

Sanitize the Customer data (Trusted Zone) and create a Glue Table (Curated Zone) that only includes customers who have accelerometer data *and* have agreed to share their data for research called **customers\_curated**.

### customer\_trusted\_to\_curated.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node Accelerometer Landing

AccelerometerLanding\_node1757139949188 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/accelerometer/landing/"], "recurse": True}, transformation\_ctx="AccelerometerLanding\_node1757139949188")

# Script generated for node Customer Trusted

CustomerTrusted\_node1757139909252 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/trusted/"], "recurse": True}, transformation\_ctx="CustomerTrusted\_node1757139909252")

# Script generated for node Join

Join\_node1757139980937 = Join.apply(frame1=AccelerometerLanding\_node1757139949188, frame2=CustomerTrusted\_node1757139909252, keys1=["user"], keys2=["email"], transformation\_ctx="Join\_node1757139980937")

# Script generated for node SQL Query

SqlQuery923 = '''

select distinct customerName, email, phone, birthday, serialNumber,

registrationDate, lastUpdateDate, shareWithResearchAsOfDate,

shareWithPublicAsOfDate, shareWithFriendsAsOfDate from myDataSource

'''

SQLQuery\_node1757140002420 = sparkSqlQuery(glueContext, query = SqlQuery923, mapping = {"myDataSource":Join\_node1757139980937}, transformation\_ctx = "SQLQuery\_node1757140002420")

# Script generated for node Customers Curated

EvaluateDataQuality().process\_rows(frame=SQLQuery\_node1757140002420, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757139860297", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

CustomersCurated\_node1757140019180 = glueContext.write\_dynamic\_frame.from\_options(frame=SQLQuery\_node1757140002420, connection\_type="s3", format="json", connection\_options={"path": "s3://stedi-lake-house-qk/customer/curated/", "partitionKeys": []}, transformation\_ctx="CustomersCurated\_node1757140019180")

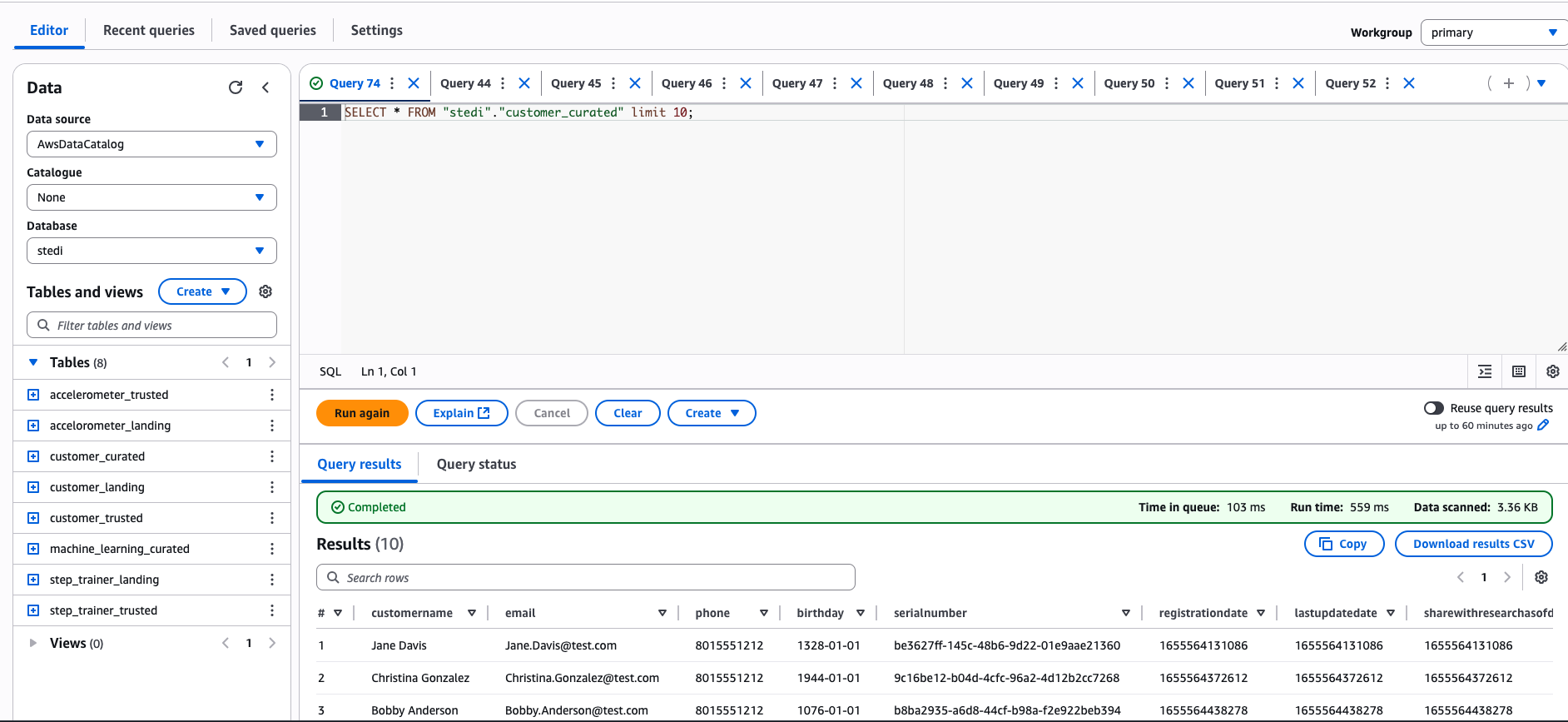
job.commit()

### Visual ETL

A screenshot of a computer

Description automatically generated

### Customer Trusted To Curated



## Write a Glue Job to create curated data

**step\_trainer\_trusted.py** has a node that inner joins the step\_trainer\_landing data with the customer\_curated data by serial numbers

**machine\_learning\_curated.py** has a node that inner joins the step\_trainer\_trusted data with the accelerometer\_trusted data by sensor reading time and timestamps

Hints:

* **Data Source - S3 bucket** node sometimes extracted incomplete data. Use the **Data Source - Data Catalog** node when that's the case.
* Use the Data Preview feature with at least 500 rows to ensure the number of customer-curated rows is correct. Click "Start data preview session", then click the gear next to the "Filter" text box to update the number of rows
* As before, the **Transform - SQL Query** node often gives more consistent outputs than any other node type. Tip - replace the JOIN node with it.
* The step\_trainer\_trusted may take about 8 minutes to run.

### step\_trainer\_trusted.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node AWS Glue Data Catalog

AWSGlueDataCatalog\_node1757143330187 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="customer\_curated", transformation\_ctx="AWSGlueDataCatalog\_node1757143330187")

# Script generated for node Step Trainer Landing

StepTrainerLanding\_node1757143347320 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"multiLine": "false"}, connection\_type="s3", format="json", connection\_options={"paths": ["s3://stedi-lake-house-qk/step\_trainer/landing/"], "recurse": True}, transformation\_ctx="StepTrainerLanding\_node1757143347320")

# Script generated for node SQL Query

SqlQuery919 = '''

SELECT st.serialnumber,

sensorreadingtime,

FIRST\_VALUE(distancefromobject)

OVER (PARTITION BY st.serialnumber, st.sensorreadingtime ORDER BY st.sensorreadingtime) AS distancefromobject

FROM stedi.step\_trainer\_landing st

JOIN customer\_curated c

ON c.serialnumber = st.serialnumber;

'''

SQLQuery\_node1757143772990 = sparkSqlQuery(glueContext, query = SqlQuery919, mapping = {"customer\_curated":AWSGlueDataCatalog\_node1757143330187, "step\_trainer\_landing":StepTrainerLanding\_node1757143347320}, transformation\_ctx = "SQLQuery\_node1757143772990")

# Script generated for node Amazon S3

EvaluateDataQuality().process\_rows(frame=SQLQuery\_node1757143772990, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757139860297", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

AmazonS3\_node1757143463027 = glueContext.getSink(path="s3://stedi-lake-house-qk/step\_trainer/final\_output/", connection\_type="s3", updateBehavior="UPDATE\_IN\_DATABASE", partitionKeys=[], enableUpdateCatalog=True, transformation\_ctx="AmazonS3\_node1757143463027")

AmazonS3\_node1757143463027.setCatalogInfo(catalogDatabase="stedi",catalogTableName="step\_trainer\_trusted")

AmazonS3\_node1757143463027.setFormat("json")

AmazonS3\_node1757143463027.writeFrame(SQLQuery\_node1757143772990)

job.commit()

### Visual ETL

A screenshot of a computer

Description automatically generated

### Step Trainer Trusted

A screenshot of a computer

Description automatically generated

## Use Athena to query Curated Glue Tables

Include screenshots showing various queries run on Athena, along with their results:

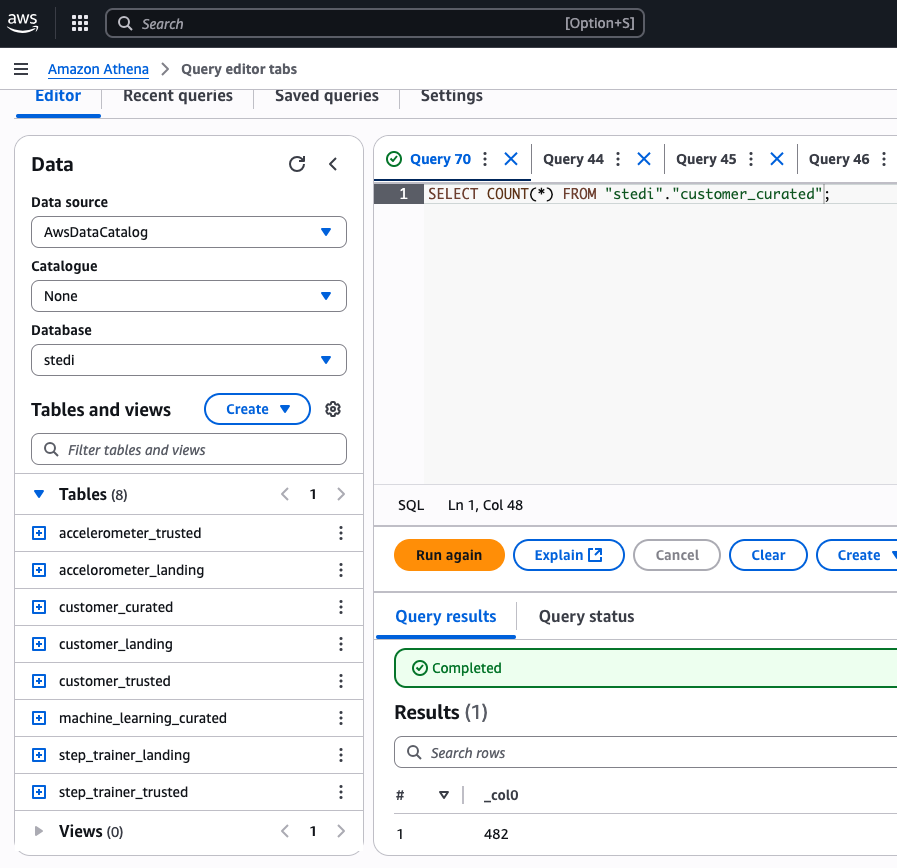
* Count of customer\_curated: 482 rows
* Count of machine\_learning\_curated: 43681 rows

However, **if you are following the stand-out suggestions**, your row counts should be as follows:

* Count of customer\_curated: 464 rows
* Count of machine\_learning\_curated: 34437 rows

Hint: If you get unexpected results, consider using the **Transform - SQL Query** node rather than Glue-provided nodes.

### Count of customer\_curated



### Machine Learning Curated – Visual ETL

A screenshot of a computer

Description automatically generated

### Count of machine\_learning\_curated

A screenshot of a computer

Description automatically generated

### machine\_learning\_curated.py

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Default ruleset used by all target nodes with data quality enabled

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Script generated for node Customer Trusted

CustomerTrusted\_node1757155489473 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="customer\_trusted", transformation\_ctx="CustomerTrusted\_node1757155489473")

# Script generated for node Step Trainer Trusted

StepTrainerTrusted\_node1757155397491 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="step\_trainer\_trusted", transformation\_ctx="StepTrainerTrusted\_node1757155397491")

# Script generated for node Accelerometer Trusted

AccelerometerTrusted\_node1757155386493 = glueContext.create\_dynamic\_frame.from\_catalog(database="stedi", table\_name="accelerometer\_trusted", transformation\_ctx="AccelerometerTrusted\_node1757155386493")

# Script generated for node Renamed keys for Join

RenamedkeysforJoin\_node1757155584336 = ApplyMapping.apply(frame=CustomerTrusted\_node1757155489473, mappings=[("serialnumber", "string", "right\_serialnumber", "string"), ("sharewithpublicasofdate", "long", "right\_sharewithpublicasofdate", "long"), ("birthday", "string", "right\_birthday", "string"), ("registrationdate", "long", "right\_registrationdate", "long"), ("sharewithresearchasofdate", "long", "right\_sharewithresearchasofdate", "long"), ("customername", "string", "right\_customername", "string"), ("sharewithfriendsasofdate", "long", "right\_sharewithfriendsasofdate", "long"), ("email", "string", "right\_email", "string"), ("lastupdatedate", "long", "right\_lastupdatedate", "long"), ("phone", "string", "right\_phone", "string")], transformation\_ctx="RenamedkeysforJoin\_node1757155584336")

# Script generated for node Step Trainer Join Accelerometer

StepTrainerJoinAccelerometer\_node1757155423358 = Join.apply(frame1=StepTrainerTrusted\_node1757155397491, frame2=AccelerometerTrusted\_node1757155386493, keys1=["sensorreadingtime"], keys2=["timestamp"], transformation\_ctx="StepTrainerJoinAccelerometer\_node1757155423358")

# Script generated for node Step Trainer Accelerometer JOIN Customer

StepTrainerAccelerometerJOINCustomer\_node1757155472373 = Join.apply(frame1=StepTrainerJoinAccelerometer\_node1757155423358, frame2=RenamedkeysforJoin\_node1757155584336, keys1=["serialnumber"], keys2=["right\_serialnumber"], transformation\_ctx="StepTrainerAccelerometerJOINCustomer\_node1757155472373")

# Script generated for node Machine Learning Curated.

EvaluateDataQuality().process\_rows(frame=StepTrainerAccelerometerJOINCustomer\_node1757155472373, ruleset=DEFAULT\_DATA\_QUALITY\_RULESET, publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757152980933", "enableDataQualityResultsPublishing": True}, additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"})

MachineLearningCurated\_node1757155615028 = glueContext.getSink(path="s3://stedi-lake-house-qk/machine\_learning/", connection\_type="s3", updateBehavior="UPDATE\_IN\_DATABASE", partitionKeys=[], enableUpdateCatalog=True, transformation\_ctx="MachineLearningCurated\_node1757155615028")

MachineLearningCurated\_node1757155615028.setCatalogInfo(catalogDatabase="stedi",catalogTableName="machine\_learning\_curated")

MachineLearningCurated\_node1757155615028.setFormat("json")

MachineLearningCurated\_node1757155615028.writeFrame(StepTrainerAccelerometerJOINCustomer\_node1757155472373)

job.commit()

AWS Glue ETL Job – Customer Trusted Zone

## 1. Introduction

This document explains the AWS Glue ETL job that processes **Customer Landing data** and produces a **Customer Trusted dataset**.

The purpose of this job is to:

* Load raw customer records from the **Landing Zone** in Amazon S3.
* Apply a **privacy filter** so that only customers who agreed to share their data are included.
* Validate the filtered dataset with **data quality rules**.
* Write the cleaned and trusted data to the **Trusted Zone** in Amazon S3.

This ensures that downstream analytics and machine learning processes only use customer data that meets privacy and quality requirements.

## 2. Code Walkthrough

### 2.1 Imports and Setup

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

import re

* **AWS Glue Libraries**: Provide transformations, job management, and data quality evaluation.
* **SparkContext & GlueContext**: Initialize Spark and Glue for distributed data processing.
* **Job**: Allows Glue to track and manage the ETL job lifecycle.
* **EvaluateDataQuality**: Enables applying rules to validate data consistency.
* **re**: Python regular expression library (not used in this script but imported by default).

### 2.2 Job Initialization

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

* Reads job arguments (JOB\_NAME) at runtime.
* Creates a **Spark session** (spark) and a **Glue session** (glueContext).
* Initializes the Glue job so AWS can monitor execution and log progress.

### 2.3 Data Quality Rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

* Defines the **ruleset** for evaluating data quality.
* Current rule: Dataset must have at least one column (ColumnCount > 0).
* This is a minimal check. Stronger rules could include:
  + RowCount > 0 (dataset not empty).
  + IsComplete "customerName" (important columns must not be null).
  + IsUnique "email" (no duplicate customers).

### 2.4 Load Customer Landing Data

AmazonS3\_node1757133904216 = glueContext.create\_dynamic\_frame.from\_options(

format\_options={"multiLine": "false"},

connection\_type="s3",

format="json",

connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/landing/"], "recurse": True},

transformation\_ctx="AmazonS3\_node1757133904216"

)

* Reads raw **customer landing data** from S3 (/customer/landing/).
* Data format = JSON.
* recurse=True → includes all nested folders under /landing/.
* Creates a **DynamicFrame**, a Glue data structure optimized for semi-structured data.

### 2.5 Privacy Filter

PrivacyFilter\_node1757133969699 = Filter.apply(

frame=AmazonS3\_node1757133904216,

f=lambda row: (not(row["shareWithResearchAsOfDate"] == 0)),

transformation\_ctx="PrivacyFilter\_node1757133969699"

)

* Applies a filter transformation on customer data.
* Keeps only rows where the field **shareWithResearchAsOfDate is not 0**.
* This ensures only customers who **agreed to share their data** are moved to the trusted zone.

### 2.6 Data Quality Evaluation

EvaluateDataQuality().process\_rows(

frame=PrivacyFilter\_node1757133969699,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757128621835", "enableDataQualityResultsPublishing": True},

additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"}

)

* Runs the **data quality rules** on the filtered dataset.
* Publishes results to AWS Glue Data Quality metrics.
* BEST\_EFFORT → logs issues but does not stop the pipeline.

### 2.7 Write Customer Trusted Data

CustomerTrusted\_node1757134020703 = glueContext.write\_dynamic\_frame.from\_options(

frame=PrivacyFilter\_node1757133969699,

connection\_type="s3",

format="json",

connection\_options={"path": "s3://stedi-lake-house-qk/customer/trusted/", "partitionKeys": []},

transformation\_ctx="CustomerTrusted\_node1757134020703"

)

* Writes the curated data into **S3 → /customer/trusted/**.
* Format = JSON.
* No partitioning (partitionKeys=[]).
* The **Customer Trusted Zone** is now available for downstream analytics and ML jobs.

### 2.8 Job Commit

job.commit()

* Marks the job as complete.
* Ensures Glue updates logs, job bookmarks, and metrics.

**4. Key Takeaways**

* This ETL job ensures that **customer privacy preferences** are respected.
* Data is checked for **basic quality** before landing in the trusted zone.
* Trusted data is written back to S3 for secure use in **analytics and machine learning workflows**.
* The pipeline is modular and can be extended with additional quality checks and partitioning strategies.

# AWS Glue ETL Job – Accelerometer Trusted Zone

### accelerometer\_landing\_to\_trusted.py

## 1. Introduction

This document explains the AWS Glue ETL job that processes **accelerometer landing data** and creates a **trusted accelerometer dataset**.

The main goal is to:

* Load accelerometer landing data and customer trusted data.
* Join the datasets to ensure only accelerometer readings from customers who agreed to share are included.
* Use Spark SQL to select relevant fields.
* Apply data quality validation.
* Write the curated accelerometer data into the **Trusted Zone** in Amazon S3.

## 2. Code Walkthrough

### 2.1 Imports and Setup

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

* **AWS Glue Libraries**: Provide transformations, data quality checks, and job utilities.
* **SparkContext / GlueContext**: Start the Spark processing engine and Glue wrapper.
* **Job**: Tracks ETL job execution lifecycle.
* **DynamicFrame**: Glue’s data abstraction for handling semi-structured data.

### 2.2 Spark SQL Helper Function

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

* Registers DynamicFrames as temporary Spark SQL views.
* Runs SQL queries against them.
* Converts the result back into a Glue DynamicFrame.

This allows you to run SQL like:

select user, x, y, z, timestamp from myDataSource

### 2.3 Job Initialization

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

* Retrieves JOB\_NAME at runtime.
* Initializes Spark and Glue contexts.
* Prepares the Glue job for execution.

### 2.4 Data Quality Rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

* Ensures that the dataset contains at least one column.
* This is a minimal check. Additional rules could be added (e.g., RowCount > 0, IsComplete "timestamp").

### 2.5 Load Source Data

AmazonS3\_node1757128664531 = glueContext.create\_dynamic\_frame.from\_catalog(

database="stedi",

table\_name="accelorometer\_landing",

transformation\_ctx="AmazonS3\_node1757128664531"

)

* Reads **Accelerometer Landing** data from the AWS Glue Catalog.

AmazonS3\_node1757128643733 = glueContext.create\_dynamic\_frame.from\_options(

format\_options={"multiLine": "false"},

connection\_type="s3",

format="json",

connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/trusted/"], "recurse": True},

transformation\_ctx="AmazonS3\_node1757128643733"

)

* Reads **Customer Trusted** data directly from S3 in JSON format.
* Includes all subfolders under /customer/trusted/.

### 2.6 Join Accelerometer with Customer

CustomerPrivacyFilter\_node1757128680082 = Join.apply(

frame1=AmazonS3\_node1757128664531,

frame2=AmazonS3\_node1757128643733,

keys1=["user"],

keys2=["email"],

transformation\_ctx="CustomerPrivacyFilter\_node1757128680082"

)

* Joins accelerometer records with customer trusted data.
* Ensures only records from **customers who agreed to share** are kept.
* Uses user from accelerometer and email from customer trusted as join keys.

### 2.7 SQL Transformation

SqlQuery914 = '''

select user, x, y, z, timestamp from myDataSource

'''

SQLQuery\_node1757135173003 = sparkSqlQuery(

glueContext,

query=SqlQuery914,

mapping={"myDataSource":CustomerPrivacyFilter\_node1757128680082},

transformation\_ctx="SQLQuery\_node1757135173003"

)

* Runs SQL on the joined dataset.
* Selects only relevant fields: user, x, y, z, and timestamp.
* Drops all other unnecessary attributes.

### 2.8 Data Quality Evaluation

EvaluateDataQuality().process\_rows(

frame=SQLQuery\_node1757135173003,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757128621835", "enableDataQualityResultsPublishing": True},

additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"}

)

* Applies data quality validation to the curated dataset.
* Publishes results to Glue metrics.
* Does not block data writing (logs only).

### 2.9 Write Accelerometer Trusted Data

AccelerometerTrusted\_node1757128772635 = glueContext.write\_dynamic\_frame.from\_options(

frame=SQLQuery\_node1757135173003,

connection\_type="s3",

format="json",

connection\_options={"path": "s3://stedi-lake-house-qk/accelerometer/trusted/", "partitionKeys": []},

transformation\_ctx="AccelerometerTrusted\_node1757128772635"

)

* Writes the filtered and curated accelerometer dataset to:  
  s3://stedi-lake-house-qk/accelerometer/trusted/
* Format = JSON.
* No partitioning applied.

### 2.10 Job Commit

job.commit()

* Finalizes the Glue job.
* Ensures metadata, logs, and metrics are stored.

### 4. Key Takeaways

* Ensures accelerometer data is only included if linked to **trusted customers**.
* Filters and reduces data to the necessary columns.
* Validates data with **basic quality rules**.
* Writes results into the **Accelerometer Trusted Zone** in S3.
* Provides a foundation for analytics and machine learning pipelines.

# AWS Glue ETL Job – Step Trainer Trusted Zone

## 1. Introduction

This ETL job processes **Step Trainer landing data** and enriches it with **customer curated data**.

The objectives are to:

* Load Step Trainer landing data from Amazon S3.
* Join it with curated customer data from the Glue Catalog.
* Select relevant fields including **distance from object** readings.
* Apply data quality checks.
* Write curated results into the **Trusted Zone** in Amazon S3 and update the Glue Catalog.

## 2. Code Walkthrough

### 2.1 Imports and Setup

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

* **Glue Libraries**: Support ETL transformations, utilities, and data quality.
* **SparkContext & GlueContext**: Provide distributed data processing.
* **Job**: Handles job lifecycle.
* **DynamicFrame**: AWS Glue abstraction for schema-flexible datasets.

### 2.2 Spark SQL Helper

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

* Registers Glue DynamicFrames as SQL tables.
* Executes SQL queries on them.
* Converts SQL results back to DynamicFrame for downstream processing.

### 2.3 Job Initialization

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

* Retrieves JOB\_NAME parameter.
* Creates Spark and Glue contexts.
* Starts the Glue job lifecycle.

### 2.4 Data Quality Rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

* Ensures datasets contain columns.
* Prevents empty or invalid schema propagation.

### 2.5 Load Source Data

AWSGlueDataCatalog\_node1757143330187 = glueContext.create\_dynamic\_frame.from\_catalog(

database="stedi",

table\_name="customer\_curated",

transformation\_ctx="AWSGlueDataCatalog\_node1757143330187"

)

* Loads **Customer Curated** table from the Glue Data Catalog.

StepTrainerLanding\_node1757143347320 = glueContext.create\_dynamic\_frame.from\_options(

format\_options={"multiLine": "false"},

connection\_type="s3",

format="json",

connection\_options={"paths": ["s3://stedi-lake-house-qk/step\_trainer/landing/"], "recurse": True},

transformation\_ctx="StepTrainerLanding\_node1757143347320"

)

* Reads **Step Trainer Landing** data directly from Amazon S3 (JSON format).

### 2.6 SQL Transformation

SqlQuery919 = '''

SELECT st.serialnumber,

sensorreadingtime,

FIRST\_VALUE(distancefromobject)

OVER (PARTITION BY st.serialnumber, st.sensorreadingtime ORDER BY st.sensorreadingtime) AS distancefromobject

FROM stedi.step\_trainer\_landing st

JOIN customer\_curated c

ON c.serialnumber = st.serialnumber;

'''

* Joins Step Trainer landing data with customer curated data.
* Filters out records for customers who did not consent.
* Selects:
  + serialnumber
  + sensorreadingtime
  + distancefromobject (first value per serial number & timestamp).

### 2.7 Data Quality Check

EvaluateDataQuality().process\_rows(

frame=SQLQuery\_node1757143772990,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757139860297", "enableDataQualityResultsPublishing": True},

additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"}

)

* Runs data quality checks on curated output.
* Publishes metrics (non-blocking).

### 2.8 Write Final Data to S3

AmazonS3\_node1757143463027 = glueContext.getSink(

path="s3://stedi-lake-house-qk/step\_trainer/final\_output/",

connection\_type="s3",

updateBehavior="UPDATE\_IN\_DATABASE",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="AmazonS3\_node1757143463027"

)

AmazonS3\_node1757143463027.setCatalogInfo(

catalogDatabase="stedi",

catalogTableName="step\_trainer\_trusted"

)

AmazonS3\_node1757143463027.setFormat("json")

AmazonS3\_node1757143463027.writeFrame(SQLQuery\_node1757143772990)

* Writes curated Step Trainer data to:  
  s3://stedi-lake-house-qk/step\_trainer/final\_output/
* Format = JSON.
* Updates AWS Glue Data Catalog with table **step\_trainer\_trusted**.

### 2.9 Commit Job

job.commit()

* Finalizes job execution and closes resources.

### 4. Key Takeaways

* Integrates Step Trainer landing data with customer curated data.
* Ensures only trusted customer data is included.
* Applies SQL transformations with **window functions** for distancefromobject.
* Performs basic data quality validation.
* Publishes trusted dataset to S3 **and registers it in the Glue Catalog**.

# AWS Glue ETL Job – Customers Curated Zone

## 1. Introduction

This AWS Glue ETL job creates the **Customers Curated dataset**.

The purpose of the job is to:

* Load **Accelerometer Landing** data from Amazon S3.
* Load **Customer Trusted** data from Amazon S3.
* Join both datasets on customer identity.
* Extract distinct customer records (removing duplicates).
* Apply data quality checks.
* Write the curated dataset into Amazon S3 in the **Customer Curated Zone**.

## 2. Code Walkthrough

### 2.1 Imports and Setup

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

from awsglue import DynamicFrame

* Glue and Spark libraries are imported.
* Provide functions for ETL operations, Spark execution, and data quality validation.

### 2.2 Spark SQL Helper

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

* Registers Glue DynamicFrames as temporary SQL tables.
* Executes SQL queries.
* Converts SQL result back to a DynamicFrame.

### 2.3 Job Initialization

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

* Retrieves JOB\_NAME.
* Creates Spark and Glue contexts.
* Initializes the Glue job lifecycle.

### 2.4 Data Quality Rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

* Ensures that the output has valid columns.
* Prevents schema errors in downstream datasets.

### 2.5 Load Source Data

AccelerometerLanding\_node1757139949188 = glueContext.create\_dynamic\_frame.from\_options(

format\_options={"multiLine": "false"},

connection\_type="s3",

format="json",

connection\_options={"paths": ["s3://stedi-lake-house-qk/accelerometer/landing/"], "recurse": True},

transformation\_ctx="AccelerometerLanding\_node1757139949188"

)

* Loads **Accelerometer Landing** data from S3 (JSON format).

CustomerTrusted\_node1757139909252 = glueContext.create\_dynamic\_frame.from\_options(

format\_options={"multiLine": "false"},

connection\_type="s3",

format="json",

connection\_options={"paths": ["s3://stedi-lake-house-qk/customer/trusted/"], "recurse": True},

transformation\_ctx="CustomerTrusted\_node1757139909252"

)

* Loads **Customer Trusted** data from S3 (JSON format).

### 2.6 Join

Join\_node1757139980937 = Join.apply(

frame1=AccelerometerLanding\_node1757139949188,

frame2=CustomerTrusted\_node1757139909252,

keys1=["user"],

keys2=["email"],

transformation\_ctx="Join\_node1757139980937"

)

* Joins Accelerometer Landing with Customer Trusted.
* Join condition:
  + Accelerometer’s user field
  + Customer’s email field

This ensures accelerometer readings are matched only with customers who have given consent.

### 2.7 SQL Transformation

SqlQuery923 = '''

select distinct customerName, email, phone, birthday, serialNumber,

registrationDate, lastUpdateDate, shareWithResearchAsOfDate,

shareWithPublicAsOfDate, shareWithFriendsAsOfDate from myDataSource

'''

* Removes duplicates using distinct.
* Selects curated customer fields:
  + **Identity fields**: customerName, email, phone, birthday
  + **Device field**: serialNumber
  + **Timestamps**: registrationDate, lastUpdateDate
  + **Privacy fields**: shareWithResearchAsOfDate, shareWithPublicAsOfDate, shareWithFriendsAsOfDate

### 2.8 Data Quality Check

EvaluateDataQuality().process\_rows(

frame=SQLQuery\_node1757140002420,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={"dataQualityEvaluationContext": "EvaluateDataQuality\_node1757139860297", "enableDataQualityResultsPublishing": True},

additional\_options={"dataQualityResultsPublishing.strategy": "BEST\_EFFORT", "observations.scope": "ALL"}

)

* Validates schema integrity.
* Publishes results but does not block job execution.

### 2.9 Write Final Data to S3

CustomersCurated\_node1757140019180 = glueContext.write\_dynamic\_frame.from\_options(

frame=SQLQuery\_node1757140002420,

connection\_type="s3",

format="json",

connection\_options={"path": "s3://stedi-lake-house-qk/customer/curated/", "partitionKeys": []},

transformation\_ctx="CustomersCurated\_node1757140019180"

)

* Writes curated customer data to:  
  s3://stedi-lake-house-qk/customer/curated/
* Format = JSON.
* This becomes the **Customer Curated Zone** dataset.

### 4. Key Takeaways

* Combines **Accelerometer Landing** with **Customer Trusted** datasets.
* Produces a **de-duplicated, curated list of customers** with device and privacy metadata.
* Ensures schema quality before publishing.
* Creates the **Customer Curated Zone** in Amazon S3.

# Machine Learning Curated Glue Job Documentation

## 1. Purpose

This AWS Glue job creates the **machine\_learning\_curated** dataset by integrating three trusted datasets:

* **Customer Trusted**
* **Step Trainer Trusted**
* **Accelerometer Trusted**

The result is a curated dataset suitable for **machine learning models**, such as predicting customer balance and stability.

## 2. Code Walkthrough

### 1. Import Required Libraries

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from awsgluedq.transforms import EvaluateDataQuality

* Imports Glue libraries, Spark, and Data Quality modules.

### 2. Initialize Job

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

* Retrieves the job name passed at runtime.
* Creates Spark and Glue contexts.
* Initializes the Glue job.

### 3. Define Data Quality Rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

* Ensures that the dataset contains at least one valid column.
* Provides basic schema validation.

### 4. Load Source Datasets (Trusted Zone)

CustomerTrusted\_node = glueContext.create\_dynamic\_frame.from\_catalog(

database="stedi", table\_name="customer\_trusted"

)

StepTrainerTrusted\_node = glueContext.create\_dynamic\_frame.from\_catalog(

database="stedi", table\_name="step\_trainer\_trusted"

)

AccelerometerTrusted\_node = glueContext.create\_dynamic\_frame.from\_catalog(

database="stedi", table\_name="accelerometer\_trusted"

)

* Loads three **trusted datasets** from the Glue Data Catalog.
* These datasets have already passed privacy and quality checks in earlier jobs.

### 5. Rename Customer Keys Before Join

RenamedkeysforJoin\_node = ApplyMapping.apply(

frame=CustomerTrusted\_node,

mappings=[

("serialnumber", "string", "right\_serialnumber", "string"),

("sharewithpublicasofdate", "long", "right\_sharewithpublicasofdate", "long"),

("birthday", "string", "right\_birthday", "string"),

("registrationdate", "long", "right\_registrationdate", "long"),

("sharewithresearchasofdate", "long", "right\_sharewithresearchasofdate", "long"),

("customername", "string", "right\_customername", "string"),

("sharewithfriendsasofdate", "long", "right\_sharewithfriendsasofdate", "long"),

("email", "string", "right\_email", "string"),

("lastupdatedate", "long", "right\_lastupdatedate", "long"),

("phone", "string", "right\_phone", "string")

]

)

* Renames columns in **Customer Trusted** by prefixing them with right\_.
* Prevents column name conflicts during join operations.

### 6. Join Step Trainer with Accelerometer

StepTrainerJoinAccelerometer\_node = Join.apply(

frame1=StepTrainerTrusted\_node,

frame2=AccelerometerTrusted\_node,

keys1=["sensorreadingtime"],

keys2=["timestamp"]

)

* Joins **Step Trainer Trusted** with **Accelerometer Trusted**.
* Matches records where sensorreadingtime = timestamp.
* Produces integrated sensor readings.

### 7. Join with Customer Data

StepTrainerAccelerometerJOINCustomer\_node = Join.apply(

frame1=StepTrainerJoinAccelerometer\_node,

frame2=RenamedkeysforJoin\_node,

keys1=["serialnumber"],

keys2=["right\_serialnumber"]

)

* Joins the combined **Step Trainer + Accelerometer** dataset with **Customer Trusted**.
* Matches on serialnumber.
* Adds customer profile information to the sensor data.

### 8. Evaluate Data Quality

EvaluateDataQuality().process\_rows(

frame=StepTrainerAccelerometerJOINCustomer\_node,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={

"dataQualityEvaluationContext": "EvaluateDataQuality\_node",

"enableDataQualityResultsPublishing": True

},

additional\_options={

"dataQualityResultsPublishing.strategy": "BEST\_EFFORT",

"observations.scope": "ALL"

}

)

* Runs data quality validation to ensure schema completeness.
* Publishes results to the Glue Data Quality service.

### 9. Write Curated Dataset to S3 + Catalog

MachineLearningCurated\_node = glueContext.getSink(

path="s3://stedi-lake-house-qk/machine\_learning/",

connection\_type="s3",

updateBehavior="UPDATE\_IN\_DATABASE",

partitionKeys=[],

enableUpdateCatalog=True

)

MachineLearningCurated\_node.setCatalogInfo(

catalogDatabase="stedi",

catalogTableName="machine\_learning\_curated"

)

MachineLearningCurated\_node.setFormat("json")

MachineLearningCurated\_node.writeFrame(StepTrainerAccelerometerJOINCustomer\_node)

* Writes the final curated dataset to **S3**.
* Registers it in the **Glue Data Catalog** as machine\_learning\_curated.
* Stored in **JSON format**.

### 10. Commit Job

job.commit()

* Finalizes the Glue job and releases resources.

## 3. Key Takeaways

* Integrates **sensor data (step trainer + accelerometer)** with **customer profiles**.
* Resolves **column conflicts** by renaming customer attributes before joins.
* Performs **data quality validation** to ensure the curated dataset is reliable.
* Outputs a **machine learning–ready dataset** for downstream AI/ML pipelines.
* Saves to **S3** and registers in the **Glue Data Catalog** for discoverability.