Caremetrix

Data Pipeline Architecture Documentation

Project Name: Caremetrix

Team: Synthea001

Introduction

This document provides a comprehensive overview of a robust, end-to-end data pipeline designed and implemented using Amazon Web Services (AWS). The pipeline is engineered to ingest, process, transform, catalogue, and analyse synthetic healthcare data generated by the **Synthea™** simulation platform.

About Synthea™

Synthea™ is an open-source patient population simulator that produces realistic synthetic health records, including demographics, medical conditions, medications, procedures, and healthcare encounters. Unlike real patient data, Synthea's synthetic data maintains the statistical properties and distributions of real populations while ensuring privacy and compliance by avoiding the use of any personal identifiable information (PII). This makes it ideal for testing, research, and developing healthcare analytics platforms without privacy concerns.

Pipeline Purpose and Goals

The primary purpose of this data pipeline is to transform raw synthetic healthcare data into clean, structured, and query-optimized datasets ready for business intelligence and analytics applications. The pipeline achieves this by leveraging AWS's serverless and managed services to minimize operational overhead while maximizing scalability, reliability, and performance.

Key Architectural Components

- **Data Ingestion and Storage**: Raw synthetic data from Synthea is ingested and stored in Amazon S3, serving as a cost-effective, durable, and highly available data lake.
- **Data Pre-processing**: An AWS Glue job performs lightweight cleaning tasks such as removing extra commas, adding ingestion timestamps, and normalizing file formatting immediately after data ingestion, preparing the data for further processing.

- **Data Transformation and ETL**: AWS Glue jobs handle comprehensive cleansing, deduplication, data type standardization, and convert data to Parquet format for efficient storage and faster querying.
- **Metadata Cataloguing and Schema Enforcement using Glue**: AWS Glue Crawlers automatically infer schemas and update the Glue Data Catalog, while pre-defined schemas ensure consistency and prevent schema drift is not efficient. So, we decided to use Glue Job instead.
- **Data Querying and Analytics**: Amazon Athena enables serverless, interactive SQL queries on Parquet data.
- **Visualization and Reporting**: By using Power BI, we are facilitating the reports and stats.

Architecture Components

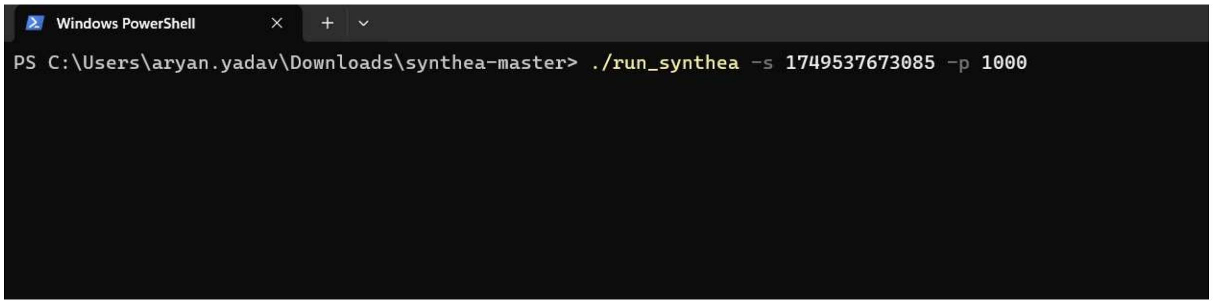
Component	Service Used	Purpose
Raw Storage	Amazon S3	Store unprocessed Synthea data
Pre-processing	AWS Glue	Basic cleanup before deeper ETL
ETL & Transformation	AWS Glue	Data cleansing, deduplication, conversion
Metadata Catalog	Glue Crawler	Automatic schema inference
Schema Management	AWS Glue	Schema enforcement & validation
Query Layer	Amazon Athena & Glue	SQL-based exploration & data marts
BI/Visualization	Power BI	Dashboards, reports, insights

3. Data Import Workflow

Local Synthea Data Generation

- Clone the Synthea repository to your local environment.
- Compile and run the simulation with parameters specifying the number of patients and a random seed for reproducibility. For example, to generate data for 1,000 patients with a seed of 1749537673085, run:

```
. /run_synthea -p 1000 -s 1749537673085
```



```
PS C:\Users\aryan.yadav\Downloads\synthea-master> ./run_synthea -s 1749537673085 -p 1000
```

- The simulation outputs CSV files into the `output/csv/` directory, including multiple tables such as patients, encounters, conditions, and more.

#NOTE: For more information on the table, you can refer About Tables.xlsx

Upload to Amazon S3

- Upload the generated CSV files to the designated **raw data** folder in your Amazon S3 bucket using AWS CLI or SDKs.
- Recommended S3 folder structure:

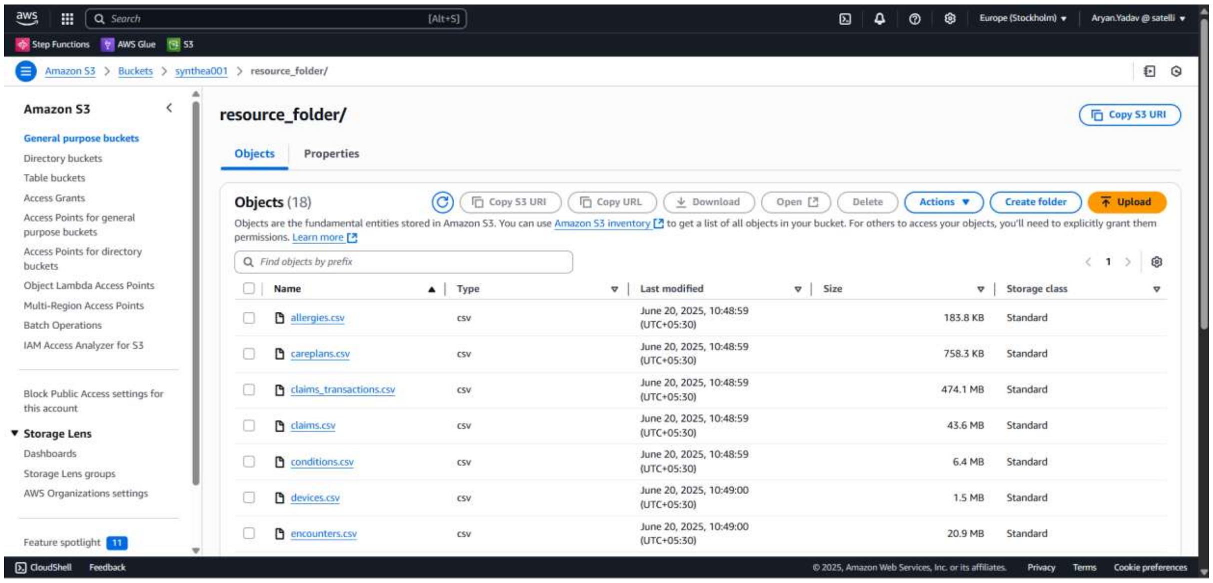
```
s3://synthea001/resource_folder/
    └── patients.csv
    └── encounters.csv
    └── conditions.csv
    └── ...
```

Pipeline Steps in Detail

Step 1: Store Raw Data in Amazon S3

- **Service**: Amazon S3
- **Bucket Structure**:

```
s3://synthea001/resource_folder/
s3://synthea001/cleaning_stage_1/
s3://synthea001/cleaning_stage_2/
```



resource_folder/

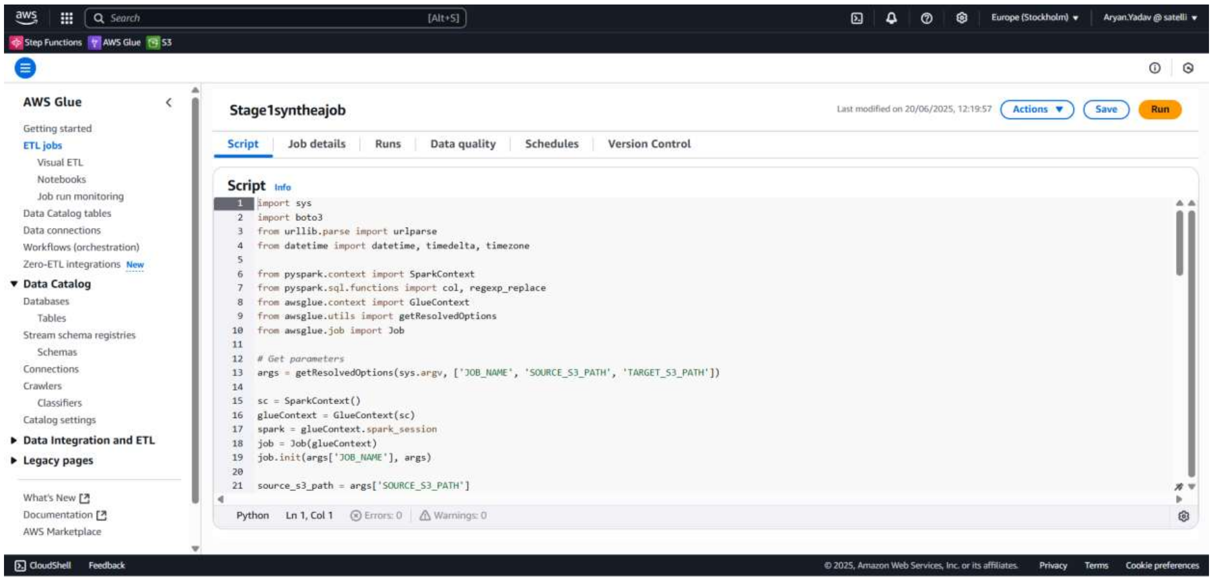
Objects (18)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
allergies.csv	csv	June 20, 2025, 10:48:59 (UTC+05:30)	183.8 KB	Standard
careplans.csv	csv	June 20, 2025, 10:48:59 (UTC+05:30)	758.3 KB	Standard
claims_transactions.csv	csv	June 20, 2025, 10:48:59 (UTC+05:30)	474.1 MB	Standard
claims.csv	csv	June 20, 2025, 10:48:59 (UTC+05:30)	43.6 MB	Standard
conditions.csv	csv	June 20, 2025, 10:48:59 (UTC+05:30)	6.4 MB	Standard
devices.csv	csv	June 20, 2025, 10:49:00 (UTC+05:30)	1.5 MB	Standard
encounters.csv	csv	June 20, 2025, 10:49:00 (UTC+05:30)	20.9 MB	Standard

Step 2: Pre-Processing via AWS Glue PySpark Job

- **Glue Job Role:**
  Initial lightweight preprocessing using PySpark to prepare raw CSV files for downstream transformation
- **Actions Performed:**
  - Remove trailing commas from each line to fix malformed rows.
  - Removing the redundant columns.
  - Write cleaned data to `s3://synthea001/cleaning_stage_1/#timestamp/`
- **Output Path:**
  Writes cleaned data to `s3://synthea001/cleaning_stage_1/#timestamp/`
- **Execution Time:**
  Typically, under 15 seconds per file, depending on file size and job parallelism

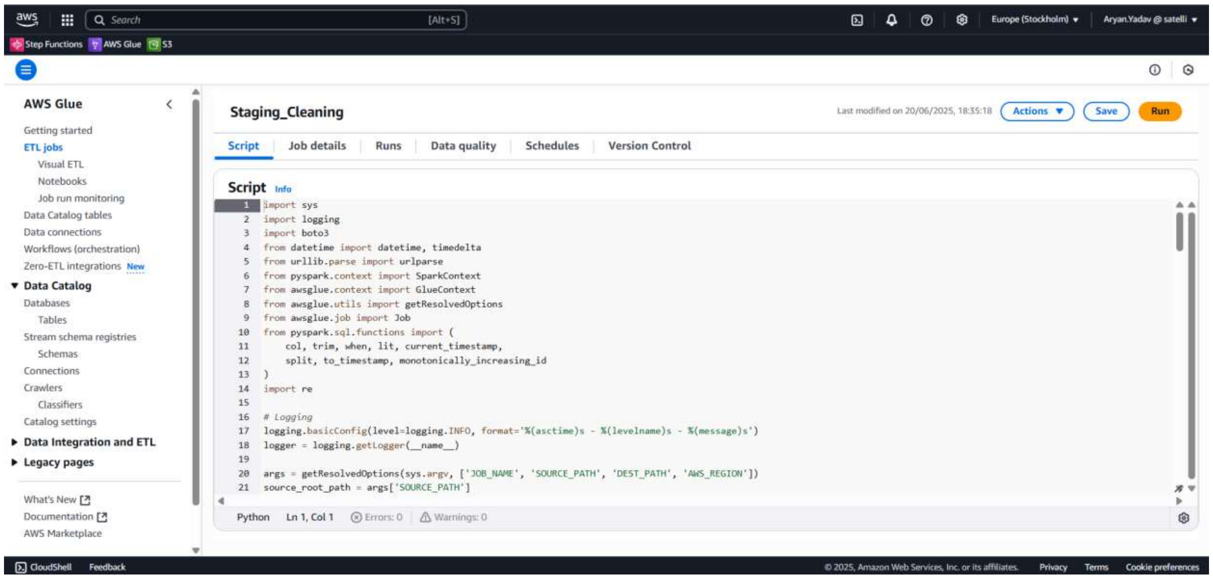


```python
import sys
import boto3
from urllib.parse import urlparse
from datetime import datetime, timedelta, timezone

from pyspark.context import SparkContext
from pyspark.sql.functions import col, regexp_replace
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from awsglue.job import Job

# Get parameters
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'SOURCE_S3_PATH', 'TARGET_S3_PATH'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

source_s3_path = args['SOURCE_S3_PATH']
```

Step 3: Data Cleansing & Parquet Conversion with AWS Glue

- **Service**: AWS Glue Jobs (PySpark)
- **Input**: Files in `s3://synthea001/cleaning_stage_1/#timestamp/`
- **Output**: Files in `s3://synthea001/cleaning_stage_2/#timestamp/` in **Parquet** format
- **Transformations**:
    - Remove duplicate rows
    - Convert date strings to timestamps
    - Apply consistent types across tables (e.g., age as integer)
    - Replace null values with None to ensure consistency
    - Standardize column names to lowercase
    - Handle Multivalued columns
    - Applied Surrogate keys and did the schema conversion handling
    - Maintained the SCD 2 in whole database
    - Convert data from CSV to optimized Parquet format for faster querying

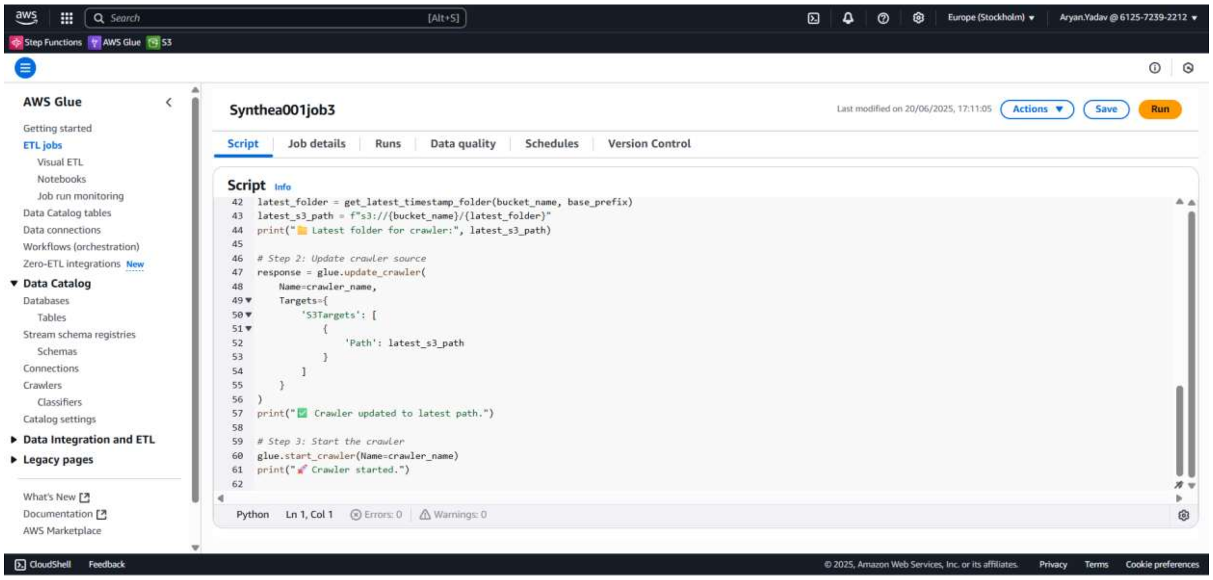
- **Performance**:
    - Parallelized execution using AWS Glue Dynamic Frame and Spark jobs
    - Optimized for cost with job bookmarks (incremental processing)



```python
import sys
import logging
import boto3
from datetime import datetime, timedelta
from urllib.parse import urlparse
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from awsglue.job import Job
from pyspark.sql.functions import (
    col, trim, when, lit, current_timestamp,
    split, to_timestamp, monotonically_increasing_id
)
import re

# Logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

args = getResolvedOptions(sys.argv, ['JOB_NAME', 'SOURCE_PATH', 'DEST_PATH', 'AWS_REGION'])
source_root_path = args['SOURCE_PATH']
```

Step 4: Crawler Trigger and Catalog Update via AWS Glue Job

- **Action:**
  Trigger the AWS Glue Crawler to scan and update the catalog with new data from timestamped Parquet folders
- **Target Location:**
  `s3://synthea001/cleaning_stage_2/#timestamp`
- **Purpose:**
  - Updates the Glue Data Catalog with latest schema from cleaned and partitioned data
  - Enables Athena to discover new partitions and tables without manual intervention
  - Ensures metadata reflects structure of transformed production-ready datasets
- **Automation:**
  Executed as part of the orchestrated pipeline after second-level cleaning is complete



Synthea001job3

Last modified on 20/06/2025, 17:11:05

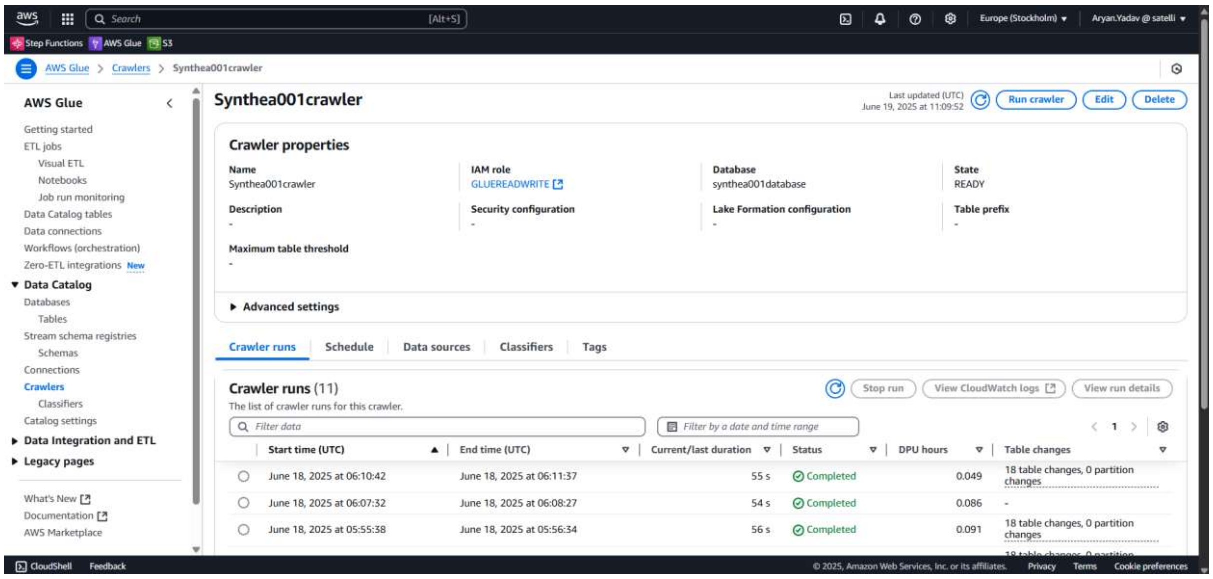
Script

```python
latest_folder = get_latest_timestamp_folder(bucket_name, base_prefix)
latest_s3_path = f"s3://{bucket_name}/{latest_folder}"
print("📁 Latest folder for crawler:", latest_s3_path)

# Step 2: Update crawler source
response = glue.update_crawler(
    Name=crawler_name,
    Targets={
        'S3Targets': [
            {
                'Path': latest_s3_path
            }
        ]
    }
)
print("✅ Crawler updated to latest path.")

# Step 3: Start the crawler
glue.start_crawler(Name=crawler_name)
print("🚀 Crawler started.")
```

Python Ln 1, Col 1 Errors: 0 Warnings: 0

Step 5: Data Cataloguing with AWS Glue Crawlers

- **Service**: AWS Glue Crawler
- **Source:** `s3://synthea001/Cleaning_Stage2/#timestamp`
- **Purpose**:
  - Detect new tables/partitions in Parquet files
  - Create or update tables in Glue Data Catalog
- **Output**: Structured metadata in Glue Catalog Database
- **Schedule**: Triggered after Glue job completion



Synthea001crawler

Last updated (UTC) June 19, 2025 at 11:09:52

Crawler properties

Name	IAM role	Database	State
Synthea001crawler	GLUEREADWRITE	synthea001database	READY
Description	**Security configuration**	**Lake Formation configuration**	**Table prefix**
-	-	-	-
Maximum table threshold			
-			

Crawler runs (11)

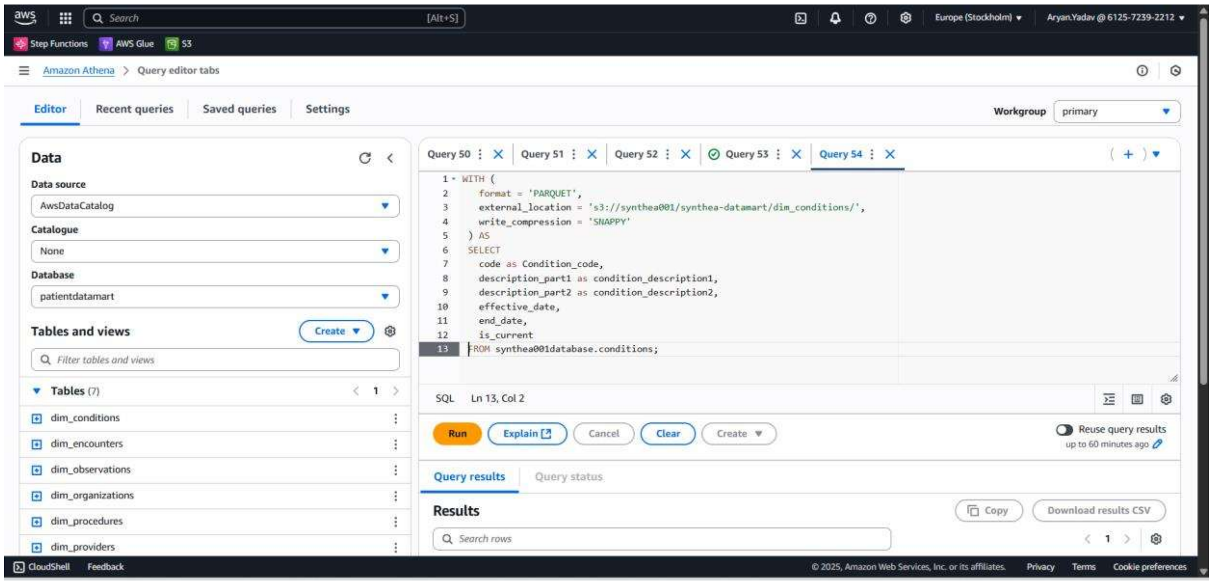
The list of crawler runs for this crawler.

Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table changes
June 18, 2025 at 06:10:42	June 18, 2025 at 06:11:37	55 s	Completed	0.049	18 table changes, 0 partition changes
June 18, 2025 at 06:07:32	June 18, 2025 at 06:08:27	54 s	Completed	0.086	-
June 18, 2025 at 05:55:38	June 18, 2025 at 05:56:34	56 s	Completed	0.091	18 table changes, 0 partition changes

Step 6: Creation of dimensions and fact table using Glue script

- Initialize Spark & Glue Context and fetch job arguments (like job name and paths).

- Create Dimension Tables (dim_*): Reads source tables (patients, encounters, etc.), adds SCD2 columns (effective_date, end_date, is_current), and saves them as Parquet tables in S3.
- Build dim_providers by joining providers and organizations.
- Create Fact Table (fact_patient_e): Joins cleaned dimension tables and adds a surrogate key (fact_patient_id) along with SCD2 columns. Write Fact Table to S3 in Parquet format and commit the Glue job.



```sql
WITH (
   format = 'PARQUET',
   external_location = 's3://synthea001/synthea-datamart/dim_conditions/',
   write_compression = 'SNAPPY'
 ) AS
 SELECT
   code as Condition_code,
   description_part1 as condition_description1,
   description_part2 as condition_description2,
   effective_date,
   end_date,
   is_current
 FROM synthea001database.conditions;
```

Step 7: Pipeline Orchestration using AWS Step Functions

Service: AWS Step Functions

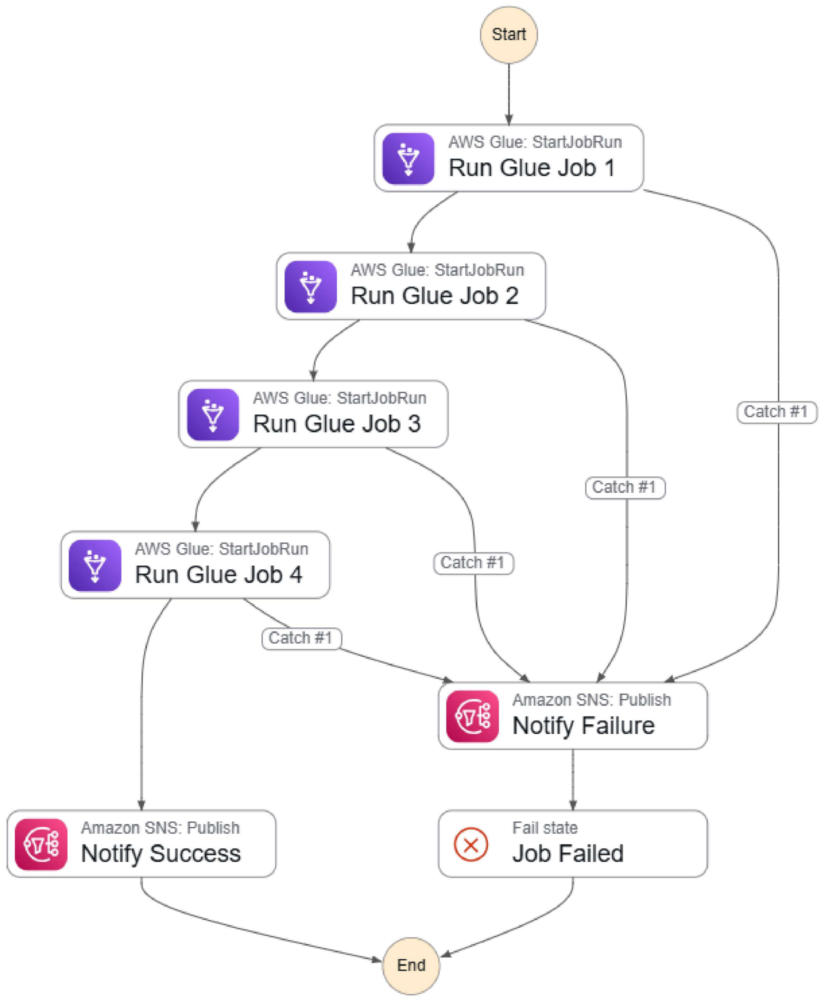
Input: Triggers for AWS Glue jobs and AWS Glue Crawler executions.

Operations:

- Define and execute a state machine that sequentially orchestrates:
  - AWS Glue job for initial cleaning (e.g., removing commas, adding timestamps)
  - AWS Glue jobs for cleansing and Parquet conversion
  - AWS Glue job to trigger the crawler and update the metadata catalog
- Incorporate error handling and retry logic for each step

Benefits:

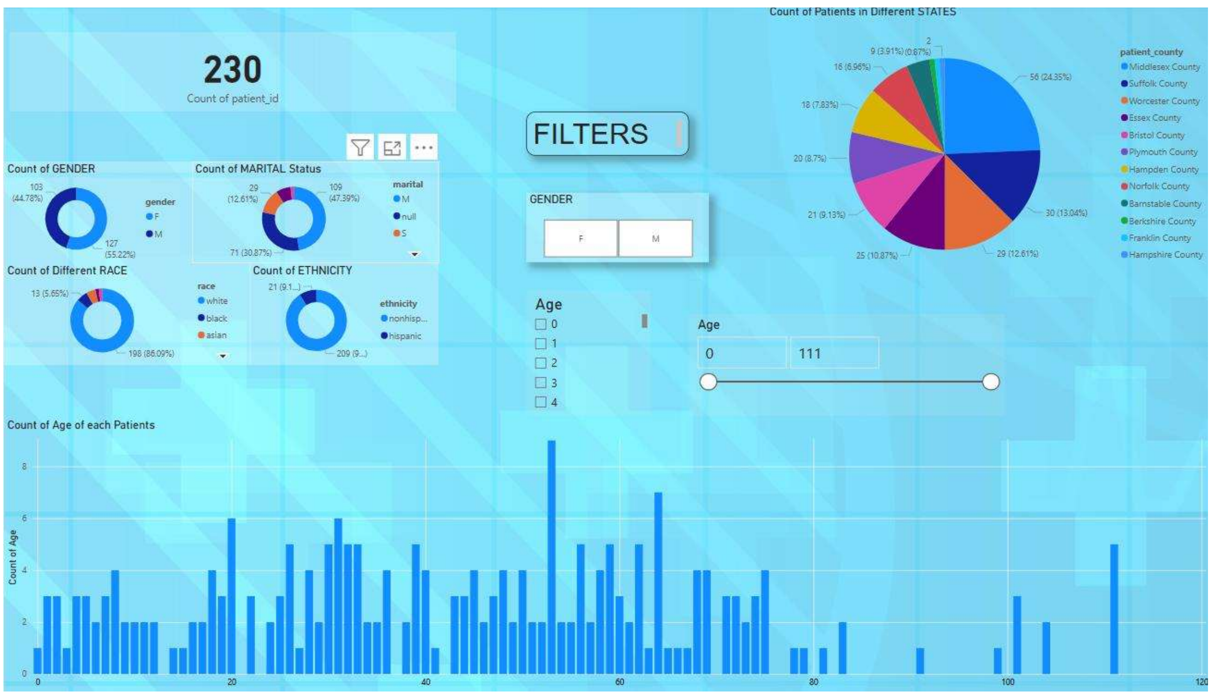
- Ensures reliable, sequential execution of all pipeline stages
- Automates retries and error recovery to minimize manual intervention
- Provides detailed execution visibility and audit trail
- Scales automatically to handle multiple or large pipeline runs



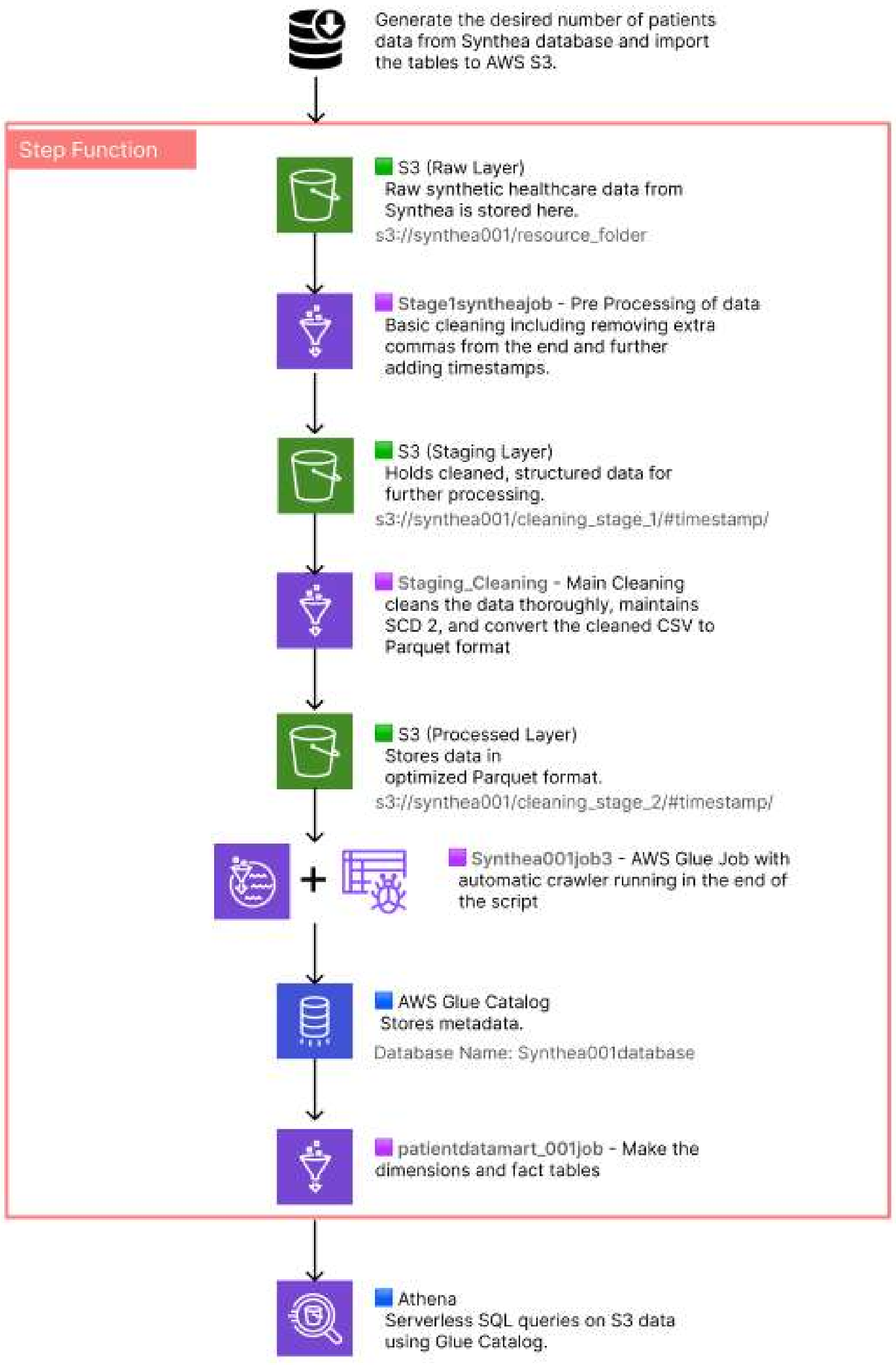
Step 8: Reporting and Visualization with Power BI

- **Action:**
  Connect Power BI to Amazon Athena using the JDBC driver to visualize query able healthcare datasets
- **Connection Method:**
  Athena JDBC connector (configured with AWS credentials and region-specific endpoint)
- **Purpose:**
  - Build interactive dashboards and reports based on curated data marts
  - Enable non-technical stakeholders to explore insights through a familiar BI tool

- Support ad hoc analysis using direct SQL queries executed through Athena
- **Benefits:**
  - Leverages Power BI's advanced visualization capabilities
  - Serverless querying via Athena ensures scalability and cost-efficiency
  - No data movement required—connects directly to data in Amazon S3 through Athena



Final flow of the Pipeline:


Generate the desired number of patients data from Synthea database and import the tables to AWS S3.
Step Function
🟩 S3 (Raw Layer)
Raw synthetic healthcare data from Synthea is stored here.
s3://synthea001/resource_folder
🟪 Stage1syntheajob - Pre Processing of data
Basic cleaning including removing extra commas from the end and further adding timestamps.
🟩 S3 (Staging Layer)
Holds cleaned, structured data for further processing.
s3://synthea001/cleaning_stage_1/#timestamp/
🟪 Staging_Cleaning - Main Cleaning
cleans the data thoroughly, maintains SCD 2, and convert the cleaned CSV to Parquet format
🟩 S3 (Processed Layer)
Stores data in optimized Parquet format.
s3://synthea001/cleaning_stage_2/#timestamp/
🟪 Synthea001job3 - AWS Glue Job with automatic crawler running in the end of the script
🟦 AWS Glue Catalog
Stores metadata.
Database Name: Synthea001database
🟪 patientdatamart_001job - Make the dimensions and fact tables
🟦 Athena
Serverless SQL queries on S3 data using Glue Catalog.