Project Documentation: Scalable AWS Data Pipeline for Synthea Dataset

Team Name: Synthea002

Team Members:
1. Manav Gupta (I10204)
2. Tamanna (I10206)
3. Lakshya Gupta (I10217)
4. Devang Sethi (I10183)

Introduction

This documentation presents a scalable and automated data pipeline built using Amazon Web Services (AWS). The system enables ingestion, processing, transformation, cataloging, and visualization of synthetic healthcare data produced by the Synthea simulation platform.

What is Synthea?

Synthea is an open-source synthetic patient generator that creates detailed yet fictitious healthcare records, including demographics, diagnoses, medications, procedures, and clinical visits. Unlike real data, Synthea's output ensures privacy by containing no personally identifiable information (PII), while preserving realistic data distributions — making it ideal for testing, analysis, and academic research.

Project Goals

The primary objective is to convert raw synthetic data into clean, well-structured, and query-optimized datasets suitable for analytics and reporting. The architecture leverages AWS managed services to deliver:

- Minimal maintenance
- High performance
- Cost efficiency
- Scalability and reliability

Key Components of the Pipeline

1. Data Lake with Amazon S3

Raw Synthea datasets are ingested into **Amazon S3**, which acts as a central, secure, and cost-efficient data lake.

2. Pre-Processing via AWS Lambda

Event-driven **Lambda functions** trigger post-upload to perform quick format **normalization and filtering.**

3. ETL Using AWS Glue

Comprehensive data cleaning, deduplication, schema alignment, and conversion to **Parquet** format are handled using **AWS Glue**.

4. Query Engine with Amazon Athena

Athena allows serverless SQL querying on the Parquet files, enabling creation of flexible, business-focused **data marts**.

5. Reporting with Amazon Quick Sight

Quick Sight connects to Athena for interactive dashboards and real-time business intelligence insights.

Architecture Components

Component	Service Used	Purpose
Data Upload	Amazon S3	Upload and store raw Synthea data in a central data lake
Minor Cleaning	AWS Glue Job	Initial cleanup (remove unnamed columns, handle extra commas)
Major Cleaning	AWS Glue Job	Advanced transformation, deduplication, type formatting, Parquet conversion
Data Mart Creation	AWS Glue Job	Organize cleaned data into specific structures
Automation	AWS Glue Triggers	Schedule and automate ETL job execution
Schema Management	Glue Data Catalog	Maintain consistent schema and structure across datasets

Visualization	AWS Quick Sight	Build interactive dashboards and derive business insights

Data Import Workflow

Generating Synthea Data Locally

- Begin by **cloning the official Synthea repository** into your local system.

- To generate synthetic patient records, **compile and run the simulation** with desired parameters such as the patient count and a random seed (to ensure reproducibility).
- Example command to generate 1,000 synthetic patients with a specific seed
  This command will produce synthetic health records (in CSV/JSON format) that can be used as input for the AWS data pipeline.

Upload to Amazon S3

- Upload the generated CSV files to the designated **raw data** folder.
- Recommended S3 folder structure:

s3://bucp2final/source/

s3:// bucp2final/raw/

s3:// bucp2final /staging/

Pipeline Steps in Detail

Step 1: Upload Raw Data to Amazon S3

- **Service:** Amazon S3

- **Description:** Upload the generated Synthea data files (CSV) to the initial `source/` folder in your S3 bucket using AWS CLI or SDKs.
- **Bucket Structure:**

`s3:// bucp2final/source/`
`s3:// bucp2final/raw/`
`s3:// bucp2final/staging/`

Step 2: Minor Cleaning with AWS Glue

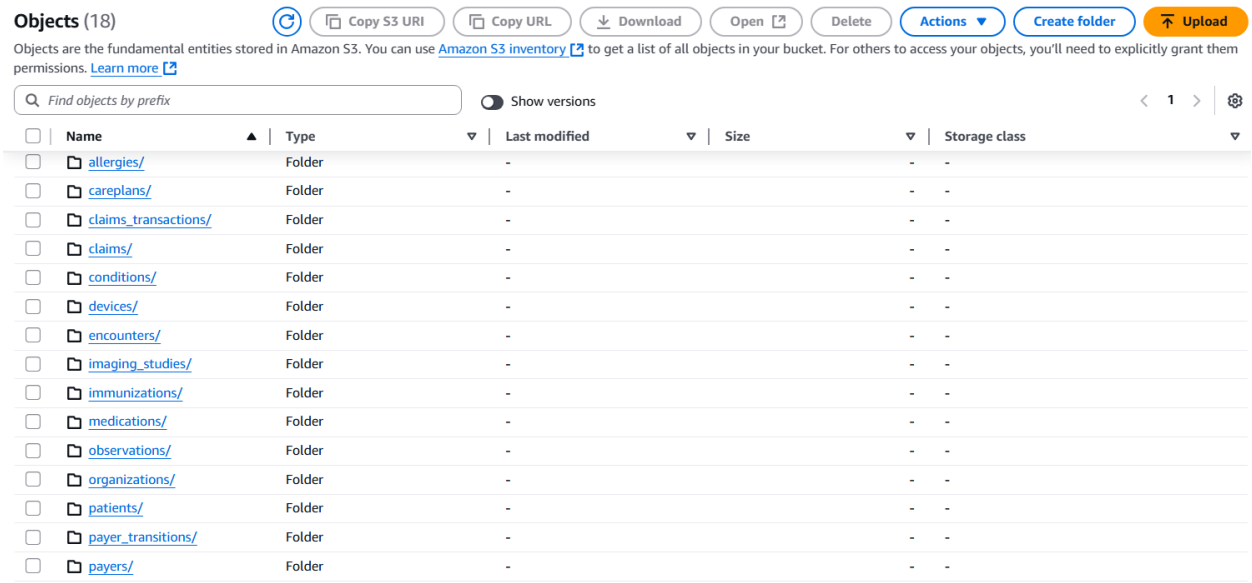
- **Service:** AWS Glue Job 1
- **Purpose:** Remove unnamed columns, fix extra delimiters, and prepare raw data for deeper transformation
- **Input:** `s3://bucp2final/source/`
- **Output:** `s3://bucp2final/raw/`

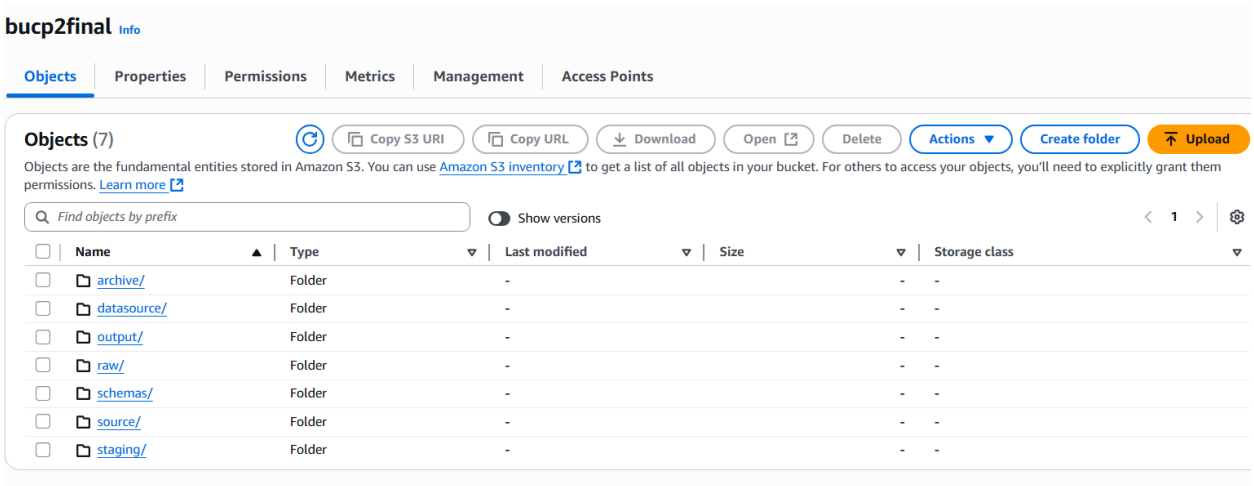
Step 3: Major Cleaning & Format Conversion

- **Service:** AWS Glue Job 2
- **Purpose:** Perform deduplication, type standardization, and convert data to Parquet format
- **Input:** `s3://bucp2final/raw/`
- **Output:** `s3://bucp2final/staging/`

Step 4: Data Mart Creation

- **Service:** AWS Glue Job 3
- **Purpose:** Aggregate and model data into business-ready structures





Data Cleaning (Minor)

1. remove unnamed columns
2. handle extra commas
3. move data from source to raw

Data Cleaning (Major)

1. replace null with none
2. standardize column names

3. trim columns
4. handle dash between phone numbers.
5. handle multi value columns
6. cast to datatype
7. csv to parquet format

tryingdatamart

Last modified on 22/06/2025, 17:50:26 | Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control

Script Info

```python
def get_latest_date_folder(bucket, prefix):
    """
    used to get the folder with latest date so that only
    the latest data is fetched
    """
    response = s3.list_objects_v2(Bucket=bucket, Prefix=prefix, Delimiter="/")
    date_folders = [content['Prefix'].split("/")[1] for content in response.get('CommonPrefixes', [])]
    return f"{prefix}{max(date_folders)}/"

latest_folder = get_latest_date_folder(bucket, prefix)

def scd2_merge(new_df, path, key_cols, hash_cols, sk_col, table_name):
    """
    compares new data with old data using hash comparison,
    expires old rows if there is change in any row and inserts new row with correct timestamps.
    """
    from pyspark.sql.types import StructType

    # Add hash to new data
```

raw_staging_data

Last modified on 20/06/2025, 16:49:34 | Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control

Script Info

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

#TIMESTAMP FOR METADATA
created_timestamp = datetime.now().strftime('%Y-%m-%dT%H-%M-%S')
s3 = boto3.client('s3')

#FUNCTION TO PROCESS FOLDERS
def list_date_folders(bucket, prefix):
    paginator = s3.get_paginator('list_objects_v2')
    prefixes = set()
    for page in paginator.paginate(Bucket=bucket, Prefix=prefix, Delimiter='/'):
        for common_prefix in page.get('CommonPrefixes', []):
            folder = common_prefix['Prefix'].rstrip('/').split('/')[-1]
            prefixes.add(folder)
```

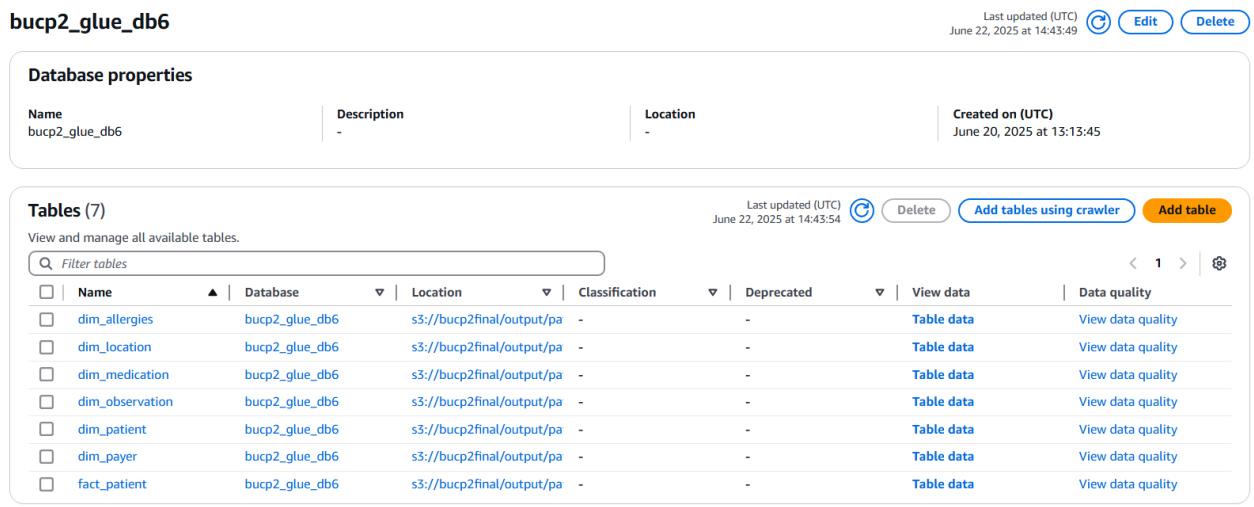
Data Marts Design:

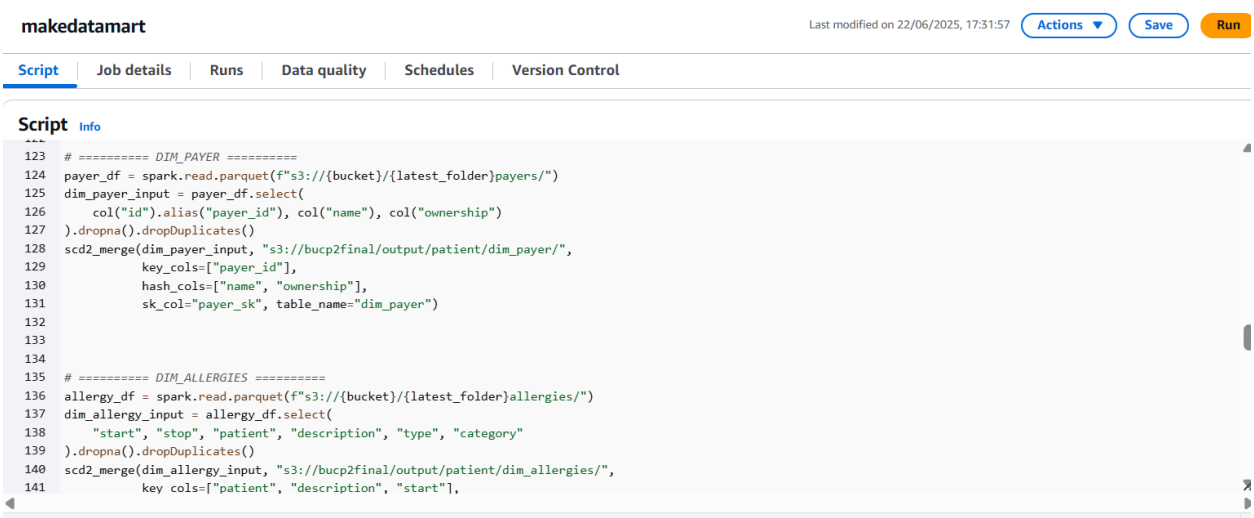
- As part of the final processing stage, we created a **Data Mart** using data from the `s3://synthea001/staging/` folder.

- The **database name** in **Amazon Athena** is:
  `bucp2_glue_db6`

- Within this Data Mart, we designed:
  1 Fact Table: `patient_fact`
  7 Dimension Tables: e.g., `encounter_dim`, `condition_dim`, `procedure_dim`, `medication_dim`, `provider_dim`, `organization_dim`, `location_dim`

- The schema design was planned and sketched **manually on paper** before implementation, following star schema best practices.





Script of Data Mart

Reporting and Analysis with Amazon Quick Sight

- We used **Amazon Quick Sight** to create **interactive dashboards and visual reports**.
- It connects directly with **Amazon Athena** to query the `bucp2_glue_db6` database.
- This allowed us to:
  - Visualize **patient demographics**, **encounter trends**, **condition distribution**, and more
  - Perform **data-driven analysis** without provisioning servers
  - Enable quick insights for stakeholders through shareable reports

Conclusion

- We successfully built a **serverless, end-to-end AWS data pipeline** for synthetic healthcare data using the Synthea platform.
- The pipeline automates data ingestion, cleaning, transformation, and analytics using:
  - **Amazon S3**, **AWS Glue**, **Athena**, and **Quick Sight**
- We designed a **data mart** with one fact and seven-dimension tables for structured analysis.
- The architecture ensures:
  - High **scalability**
  - Minimal **maintenance**
  - Fast and reliable **insight generation**