

**NSBM Green University**

**Faculty of Computing**

**BSc (Hons) Data Science**

**DS403.3- Big Data Programming**

**Final Report**

**Group - 2**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| 24490 | MRK Karunathilaka |
| 24572 | MKIM Rohana |
| 24614 | GAAS Ganegoda |

**Module Lecturer: Mr. Adhil Rushdy**

**Table of Contents**

[1. Introduction & Objectives 3](#_Toc197854226)

[2. Data Sources 3](#_Toc197854227)

[3. Architecture 3](#_Toc197854228)

[4. Batch Layer 4](#_Toc197854229)

[Step 01: Resource Group Creation 4](#_Toc197854230)

[Step 02: Data Lake Setup 4](#_Toc197854231)

[Step 03: Data Ingestion with Azure Data Factory (ADF) 5](#_Toc197854232)

[Challenges & Solutions 6](#_Toc197854233)

[Step 04: Data Ingestion with Azure Data Factory (ADF) 7](#_Toc197854234)

[Step 05: Data Warehousing with Azure Synapse Analytics 8](#_Toc197854235)

[5. Speed Layer 8](#_Toc197854236)

[6. Serving Layer 8](#_Toc197854237)

[7. Presentation Layer 8](#_Toc197854238)

[8. Key Technical Decisions & Justifications in Cold Path 9](#_Toc197854239)

[9. Challenges & Solutions in Cold Path 9](#_Toc197854240)

# Introduction & Objectives

# Data Sources

# Architecture

* Used architecture: - Lambda Architecture

A diagram of a software development process

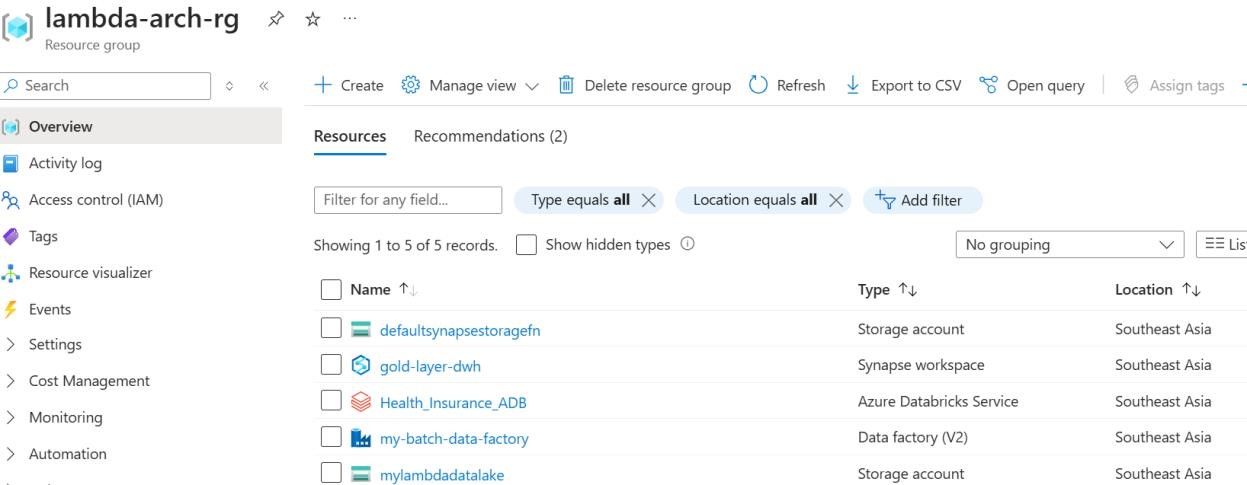
AI-generated content may be incorrect.

* Lambda Architecture Main Components
  + Batch Layer 🡪 Batch Processing
  + Speed Layer 🡪 Real-time/Near real-time Processing
  + Serving Layer 🡪 Combined both batch data and real-time data

# Batch Layer

## Step 01: Resource Group Creation

* **Action**: Created a resource group named **lambda-arch-rg** to centralize all project resources.
* **Purpose**: Ensures organized management and cost tracking.

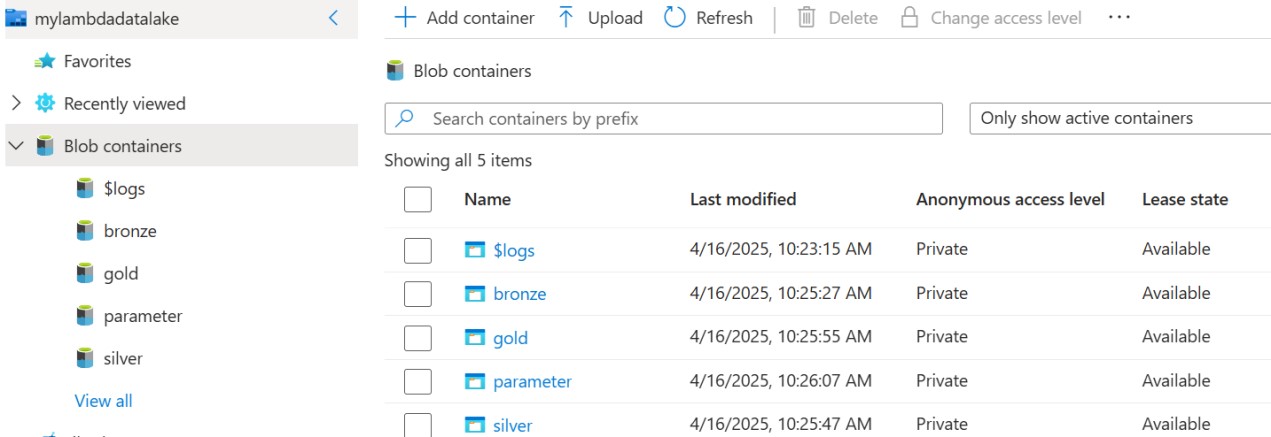


## Step 02: Data Lake Setup

* **Actions**:Created a **Storage Account** (ADLS Gen2) with four containers,

* + **bronze:** Stores raw data (e.g., CSV files from Google Drive).
  + **silver:** Holds transformed/cleaned data (Parquet format).
  + **gold:** Stores analysis-ready datasets (aggregated tables).
  + **parameter:** Contains JSON files for dynamic pipeline configurations.

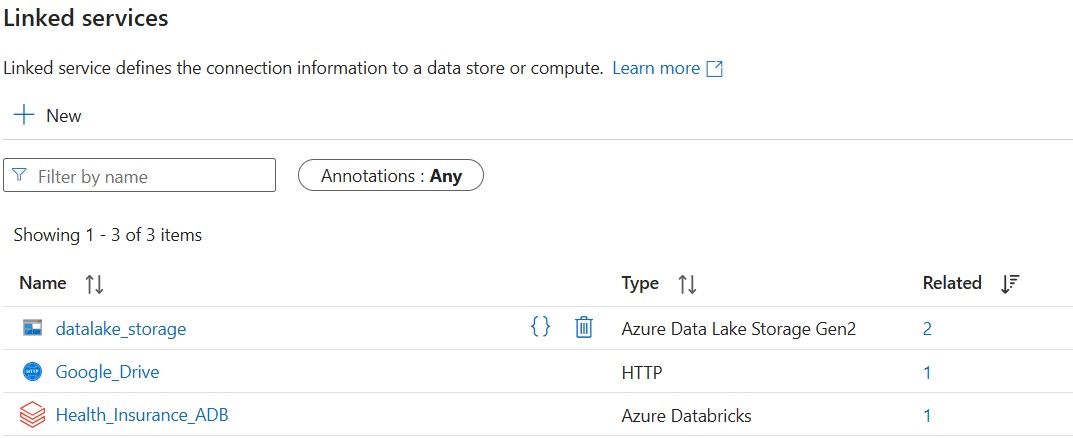
* **Purpose**: A Data Lake is very cost-effective; it can store both structured and unstructured data due to its object Storage.



## Step 03: Data Ingestion with Azure Data Factory (ADF)

• **Linked Services**

* **ADLS Gen2**: Connected to the data lake containers.
* **Google Drive**: Enabled CSV file ingestion (fallback after Git repo failed due to file size limits).



### Challenges & Solutions

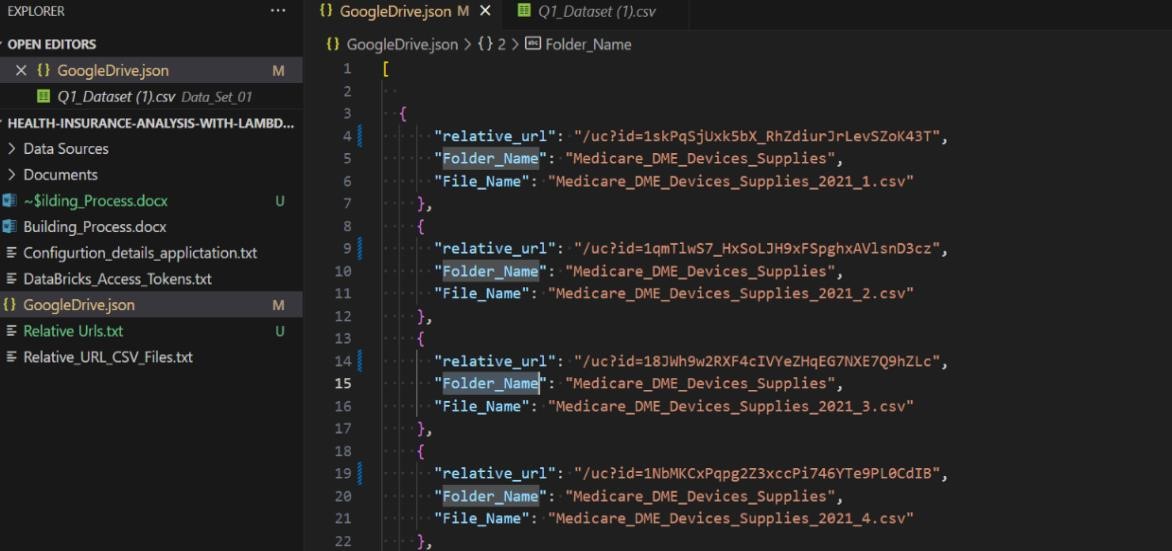
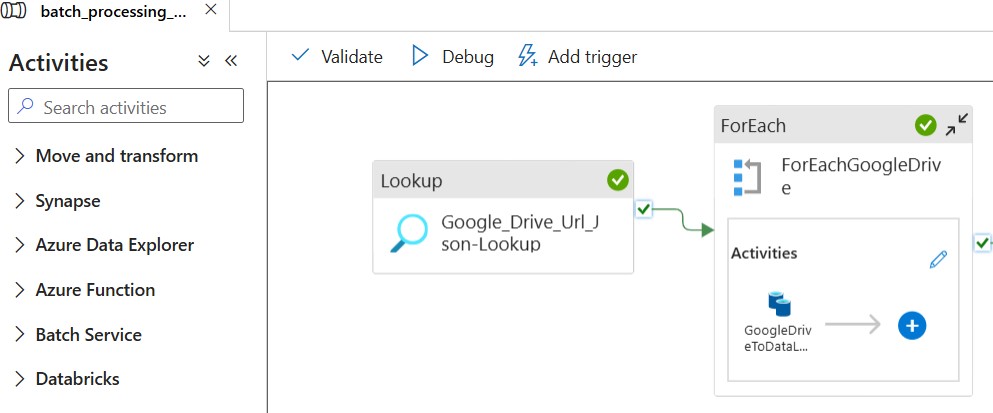
* **Google Drive 100MB Limit**

* + Split files into smaller chunks (<100MB) to avoid corruption.

* **Dynamic Pipeline**

* + Used **Lookup Activity** to fetch parameters from JSON files.

* + **ForEach Activity** + **Copy Activity** transferred files from Google Drive to bronze with,
    - Source parameter: relative\_url.
    - Sink parameters: Folder\_Name and File\_Name.



## Step 04: Data Ingestion with Azure Data Factory (ADF)

* **Cluster Configuration**

* + - **Single-Node Cluster** (Standard\_DS3\_v2, 14GB RAM) to minimize costs.
    - **Auto-termination**: Set to 10 minutes of inactivity.

* **Data Processing**

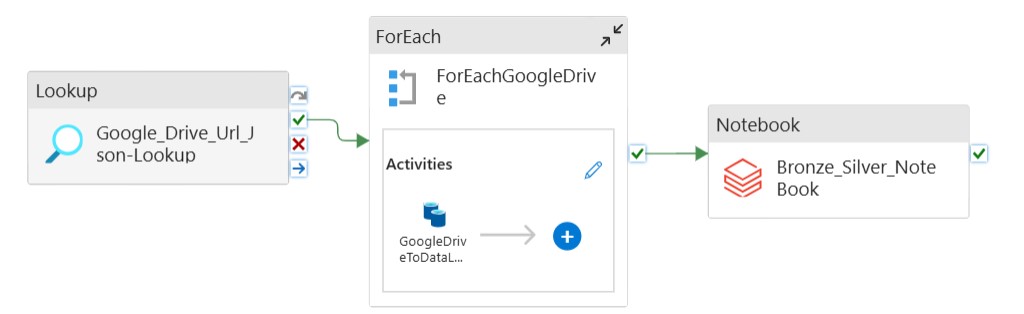
* 1. Mounted **bronze** and **silver** containers to Databricks.

* 1. Loaded data into Data Frames,
     + Claims\_df (claim details).
     + Drugs\_df (prescription data).
     + Medicare\_DME\_DS\_df (medical equipment records).

* 1. Transformed data (cleaning) → Saved to silver (Parquet).

* **Integration**

▪ Linked Databricks notebook to ADF’s batch\_processing\_pipeline via a Databricks-linked service.



## Step 05: Data Warehousing with Azure Synapse Analytics

• **Why Synapse?** Unified platform for,

* **Data Factory (ADF)**: Pipeline orchestration (redundant with standalone ADF but retained for learning).

* **Data Warehouse (DWH)**: Serverless SQL Pool chosen over Dedicated SQL Pool for: ▪ **Cost Efficiency**: Pay-per-query (~$5/TB scanned) vs. fixed hourly costs.

▪ **Data Virtualization**: Uses OPENROWSET() to query ADLS directly (no storage duplication).

# Speed Layer

# Serving Layer

# Presentation Layer

# Key Technical Decisions & Justifications in Cold Path

|  |  |  |
| --- | --- | --- |
| **Component** | **Choice** | **Reason** |
| **Cluster Type** | Single-Node  (Databricks) | Cost savings; sufficient for batch workloads. |
| **File Format** | Parquet | Columnar storage → 80% smaller scans vs. CSV. |
| **Synapse SQL Pool** | Serverless | No infrastructure costs; scales to zero. |
| **Data Ingestion** | Google Drive +  ADF | Workaround for Git’s file size limits. |

# Challenges & Solutions in Cold Path

|  |  |
| --- | --- |
| **Challenge** | **Solution** |
| Google Drive file corruption (>100MB) | Split files into sub-100MB chunks. |
| Databricks cluster startup delays | Auto-termination + single-node configuration. |
| Dynamic pipeline requirements | Parameterized JSON files + Lookup Activity. |