# Microsoft Fabric End to End Project

Welcome! This guide will walk you step-by-step through setting up a data pipeline using Microsoft Fabric. We’ll use an approach called the **medallion architecture** to organise earthquake data from raw to refined, making it easier for analysis. By the end of this guide, you’ll have an automated system that takes earthquake data from an API, processes it, and displays it in Power BI for easy insights—all updated daily!

### Business Case

Earthquake data is incredibly valuable for understanding seismic events and mitigating risks. Government agencies, research institutions, and insurance companies rely on up-to-date information to plan emergency responses and assess risks. With this automated pipeline, we ensure these stakeholders get the latest data in a way that’s easy to understand and ready to use, saving time and improving decision-making.



### Solution Overview

We're going to use **Microsoft Fabric** to build an end-to-end earthquake data pipeline using the **medallion architecture**. This involves:

* **Bronze Layer**: Collect raw earthquake data.
* **Silver Layer**: Transform it into a clean, structured format.
* **Gold Layer**: Enrich it with calculated fields for analysis.
* **Power BI**: Visualise the data interactively.
* **Data Factory**: Automate the entire process, ensuring up-to-date information daily.

### Medallion Architecture in Action

Let’s break down each layer:

1. **Bronze Layer (Raw Data Ingestion)**
   * **What It Does**: This layer is for storing raw data, exactly as we receive it.
   * **How We Do It**: Fetch earthquake data from an external API using a Python script and store it as **JSON files** in the Fabric Lakehouse.
   * **Why It’s Important**: This keeps the original data safe for audit or if you ever need to trace issues back to their source.
2. **Silver Layer (Data Transformation)**
   * **What It Does**: This layer takes the raw data and cleans it up for use.
   * **How We Do It**: Convert the **JSON files** into **Delta tables**. Here, we clean and filter the data—adding structured columns like event time, magnitude, and location.
   * **Why It’s Important**: This creates a clean, easy-to-use version of the data, ready for most reporting needs.
3. **Gold Layer (Data Enrichment)**
   * **What It Does**: This layer prepares data for high-quality analysis.
   * **How We Do It**: Use the Silver Layer data to create more meaningful information, like **country codes** and **significance classifications**. The final output is saved as a **Delta table**.
   * **Why It’s Important**: This gives you an enriched dataset that’s optimised for analysis, providing ready-to-use insights.
4. **Power BI Visualisation**
   * **What It Does**: Visualises the data to make it understandable at a glance.
   * **How We Do It**: Load the **Gold Layer** data into Power BI to create an interactive dashboard. Here, you can see trends and patterns in earthquake occurrences, like the number of events by country or the severity over time.
   * **Why It’s Important**: Visualisation is crucial for communicating complex data simply—making insights accessible to everyone.
5. **Orchestration with Data Factory**
   * **What It Does**: Automates the entire pipeline so that it runs every day.
   * **How We Do It**: Use **Data Factory** to set up a daily schedule, automating data collection, transformation, and enrichment.
   * **Why It’s Important**: Automation means you don’t need to manually update the data—ensuring stakeholders always have up-to-date information.

### Final Thoughts

By following this guide, you’ll implement a solid data pipeline from raw earthquake data to meaningful insights. This project uses the **medallion architecture** to make the data transformation process logical and organized, while **Power BI** makes it visually engaging and easy to understand. With **Data Factory**, you’ll automate everything, ensuring it keeps working for you on autopilot.

### Step 1: Setup

1. **Create Microsoft Work Email**
   * If you don't have one, create a Microsoft Work account. Here's a helpful guide: [YouTube Video](https://www.youtube.com/watch?v=9RB5xic9BiY).
2. **Sign In to Fabric**
   * Go to [Microsoft Fabric](https://app.fabric.microsoft.com/) and sign in using your new work email.
3. **Create a Fabric Workspace**
   * Select any experience.
   * On the left panel, click on "Workspaces" and then "New Workspace".
4. **Create a Lakehouse**
   * On the bottom left, navigate to **Data Engineering**.
   * Select **Lakehouse** under Recommended or go to your workspace and click **+ New**.
   * Name your Lakehouse. This will store the data ingested from the API.
   * Note: Delta format is the default table format, optimized for version control and efficient querying.
5. **Lakehouse Overview**
   * You can switch between the **Lakehouse** and **SQL Analytics Endpoint** using the top right options.
   * The Lakehouse is your central data repository, the SQL endpoint provides SQL access, and the semantic model layer helps define business logic.
     + A semantic model is a structured representation of data that defines relationships, meanings, and rules to provide context and support better analysis and understanding.

### Step 2: Understanding the API

* The earthquake API will return detailed earthquake data for a specified start and end date.
* You will need to pass a **start date** to define the range of the data. This start date will be dynamically set via Data Factory for daily data ingestion.
* API URL: <https://earthquake.usgs.gov/fdsnws/event/1/>

### Step 3: Fetch API Data with Python (Bronze Layer)

### ****Create a Notebook in the Lakehouse****

* + Click **+ New** → **Notebook**. Ensure the language is set to **PySpark**.
  + Attach your Lakehouse to the notebook.

1. **Python Code to Fetch Data**
   * Use the new code provided:

import requests

import json

from datetime import date, timedelta

# Remove this before running Data Factory Pipeline

start\_date = date.today() - timedelta(7) # 7 days

end\_date = date.today() - timedelta(1)

# Construct the API URL with start and end dates provided by Data Factory, formatted for geojson output.

url = f"<https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime={start\_date}&endtime={end\_date}>"

try:

# Make the GET request to fetch data

response = requests.get(url)

# Check if the request was successful

response.raise\_for\_status() # Raise HTTPError for bad responses (4xx or 5xx)

data = response.json().get('features', [])

if not data:

print("No data returned for the specified date range.")

else:

# Specify the file name (and path if needed)

file\_path = f'/lakehouse/default/Files/{start\_date}\_earthquake\_data.json'

# Save the JSON data

with open(file\_path, 'w') as file:

json.dump(data, file, indent=4)

print(f"Data successfully saved to {file\_path}")

except requests.exceptions.RequestException as e:

print(f"Error fetching data from API: {e}")

1. **Save the Data**
   * The file path can be obtained by clicking the three dots next to the **Files** folder in the Lakehouse and selecting **Copy Relative Path** for Spark.
   * Refresh the Lakehouse to see the newly created file.
2. **Load Data into a Dataframe**
   * Click on the file and load it into a DataFrame to verify the data.

### Step 4: Silver Layer Transformation (JSON to Table)

1. **Create a New PySpark Notebook**
2. **Transform Data into a Table**
   * Use the new code provided:

from pyspark.sql.functions import col, isnull, when

from pyspark.sql.types import TimestampType

from datetime import date, timedelta

# Remove this before running Data Factory Pipeline

start\_date = date. today() - timedelta(7)

# Load the JSON data into a Spark DataFrame

df = spark.read.option("multiline", "true").json(f"Files/{start\_date}\_earthquake\_data.json")

# Reshape earthquake data

df = (

df

.select(

'id',

col('geometry.coordinates').getItem(0).alias('longitude'),

col('geometry.coordinates').getItem(1).alias('latitude'),

col('geometry.coordinates').getItem(2).alias('elevation'),

col('properties.title').alias('title'),

col('properties.place').alias('place\_description'),

col('properties.sig').alias('sig'),

col('properties.mag').alias('mag'),

col('properties.magType').alias('magType'),

col('properties.time').alias('time'),

col('properties.updated').alias('updated')

)

)

# Validate data: Check for missing or null values

df = (

df

.withColumn('longitude', when(isnull(col('longitude')), 0).otherwise(col('longitude')))

.withColumn('latitude', when(isnull(col('latitude')), 0).otherwise(col('latitude')))

.withColumn('time', when(isnull(col('time')), 0).otherwise(col('time')))

)

# Convert 'time' and 'updated' to timestamp

df = (

df

.withColumn('time', (col('time') / 1000).cast(TimestampType()))

.withColumn('updated', (col('updated') / 1000).cast(TimestampType()))

)

# Append to the silver table

df.write.mode('append').saveAsTable('earthquake\_events\_silver')

1. **Write the Data to the Silver Table**
   * Use **append** mode to update the data daily.
   * Refresh the tables to see the new Silver layer.

### Step 5: Gold Layer Transformation

1. Create a new fabric environment and install reverse\_geocoder
   * On the top ribbon select the environment drop-down → New environment
   * Under Public Libraries type in reverse\_geocoder and click Save → Publish.
   * Note that this will take up to 15 minutes to publish and a further 5 to connect.
2. **Add Country Codes and Classifications**
   * Use the new code provided:

from pyspark.sql.functions import when, col, udf

from pyspark.sql.types import StringType

# Ensure the below library is installed on your fabric environment

import reverse\_geocoder as rg

from datetime import date, timedelta

# Remove this before running Data Factory Pipeline

start\_date = date. today() - timedelta(7)

df = spark.read.table("earthquake\_events\_silver").filter(col('time') > start\_date)

def get\_country\_code(lat, lon):

"""

Retrieve the country code for a given latitude and longitude.

Parameters:

lat (float or str): Latitude of the location.

lon (float or str): Longitude of the location.

Returns:

str: Country code of the location, retrieved using the reverse geocoding API.

Example:

>>> get\_country\_details(48.8588443, 2.2943506)

'FR'

"""

coordinates = (float(lat), float(lon))

return rg.search(coordinates)[0].get('cc')

# registering the udfs so they can be used on spark dataframes

get\_country\_code\_udf = udf(get\_country\_code, StringType())

# adding country\_code and city attributes

df\_with\_location = \\

df.\\

withColumn("country\_code", get\_country\_code\_udf(col("latitude"), col("longitude")))

# adding significance classification

df\_with\_location\_sig\_class = \\

df\_with\_location.\\

withColumn('sig\_class',

when(col("sig") < 100, "Low").\\

when((col("sig") >= 100) & (col("sig") < 500), "Moderate").\\

otherwise("High")

)

# appending the data to the gold table

df\_with\_location\_sig\_class.write.mode('append').saveAsTable('earthquake\_events\_gold')

### Step 6: Update Default Semantic Model

* The lakehouse contains a semantic model:
  + Queried via SQL endpoint and updates with lakehouse changes.
  + Direct Lake allows Power BI to load data directly, avoiding duplication.
* Steps to update the default semantic model:
  + Switch to **SQL Endpoint** and go to **Reporting**.
  + Select **Model** > **Default Semantic Model Objects**.
  + Click **Manage Default Semantic Model**, choose the **Gold Table**, and confirm.
  + The **Default Semantic Model** now includes the **Gold Table**.

### Step 7: Create Power BI Report

1. **Switch to Power BI:**
   * Click on **Data Engineering** (bottom left).
   * Change to **Power BI**.
2. **Create a New Report:**
   * Click **New Report**.
   * Pick a **Published Semantic Model**.
   * Select **Gold**, then click **Create Blank Report**.
3. **Save the Report:**
   * Go to **File** > **Save As**.
   * Give the report a name and click **Save**.
4. **Enable Map Visuals:**
   * Go to **Settings** > **Admin Portal**.
   * Use **Ctrl + F** to search for **Map Visuals**.
   * Enable the setting.
5. **Navigate to the Report:**
   * Use the **Left Navigator** or go to the **Workspace** to find and open the report.
6. **Configure the Map Visualisation:**
   * **Select Map Visualisation**:
     + For **Location**, pass in **Country Code**. Rename it to **Country**.
     + For **Bubble Size**, use **Sum of Sig**. Change it to **Max** and rename it to **Maximum Significance**.
     + For **Legend**, select **Sig\_Class** and rename it to **Classification**.
     + For **Tooltip**, use **ID** and rename it to **Number of Events**.
7. **Add a Slicer Visual:**
   * **Add Time** and rename it to **Select a Date Range**.
   * This will dynamically filter the data.
8. **Add Another Slicer:**
   * Use **Significance**.
   * Format it to **Tile** via Slicer Settings.
   * Hide the **Header**.
9. **Add a Multi-Row Card:**
   * Add two fields:
     + **Count (ID)**, renamed to **Total**.
     + **Sig (Max)**, renamed to **Significance**.

This structure makes each step clear and easy to follow.

### Step 8: Set Up Automated Data Factory Pipeline

1. **Create a Data Pipeline:**
   * Click **Power BI** (bottom left).
   * Click **Data Factory** > **Data Pipeline**.
   * Enter a name and click **Create**.
2. **Configure Bronze Pipeline Activity:**
   * Click on **Pipeline Activity** > **Notebook**.
   * Name it **Bronze**.
   * Go to **Settings**:
     + Select the **Workspace** and the **Bronze Notebook**.
     + Click **+ New Base Parameters** to add **Start and End Date**:
       - Name: **start\_date** / **end\_date** (match the notebook).
       - Type: **String**.
       - Value: **Add Dynamic Content**:

**Start Date**: @formatDateTime(addDays(utcnow(), -1), 'yyyy-MM-dd').

* + - * Remember to remove the code segments with the comment:

# Remove this before running Data Factory Pipelineform

**End Date**: @formatDateTime(utcnow(), 'yyyy-MM-dd').

* + - * This sets the start date to yesterday and the end date to today (for daily runs).

1. **Link Activities:**
   * Go to **Activities** > **Notebook**.
   * Click the green tick to link it from **Bronze**.
2. **Create Silver Notebook:**
   * Name it **Silver**.
   * Go to **Settings** and change the notebook to **Silver**.
   * **Base Parameters**: Use the **Outputs** from **Bronze** to avoid duplicating the code:
     + **Start Date**: @activity('Bronze').output.start\_date.
     + **End Date**: @activity('Bronze').output.end\_date.
3. **Create Gold Notebook:**
   * Add a **Gold Notebook** that runs on the success of **Silver** (link using the tick).
   * Name it **Gold**.
   * Go to **Settings** and select the **Gold Notebook**.
   * **Base Parameters**: Use the same **Outputs** from **Bronze**:
     + **Start Date**: @activity('Bronze').output.start\_date.
     + **End Date**: @activity('Bronze').output.end\_date.
4. **Validate and Run Pipeline:**
   * Click **Validate** > **Run**.
   * This should add an extra date to the slider.
5. **Monitor the Output:**
   * Watch the **Output** for pipeline results.
6. **Schedule the Pipeline:**
   * Click **Schedule** and select a **Daily Frequency**.
7. **Monitor Runs:**
   * Check the **Run History** or go to the **Monitoring Hub** to view the pipeline runs.
8. **Update Power BI Report:**
   * Go to the **Power BI Report**.
   * Click **Refresh** (top right) to pull in the new data.

This version uses the outputs from the Bronze activity for the Silver and Gold notebooks, reducing redundancy and making the process more efficient.

### Summary

You have now successfully created a daily automated pipeline that ingests earthquake data, processes it, and visualises it in Power BI. This guide introduces key concepts like Lakehouse, PySpark transformations, and Data Factory orchestration, making Microsoft Fabric accessible for beginners interested in data engineering and analytics.