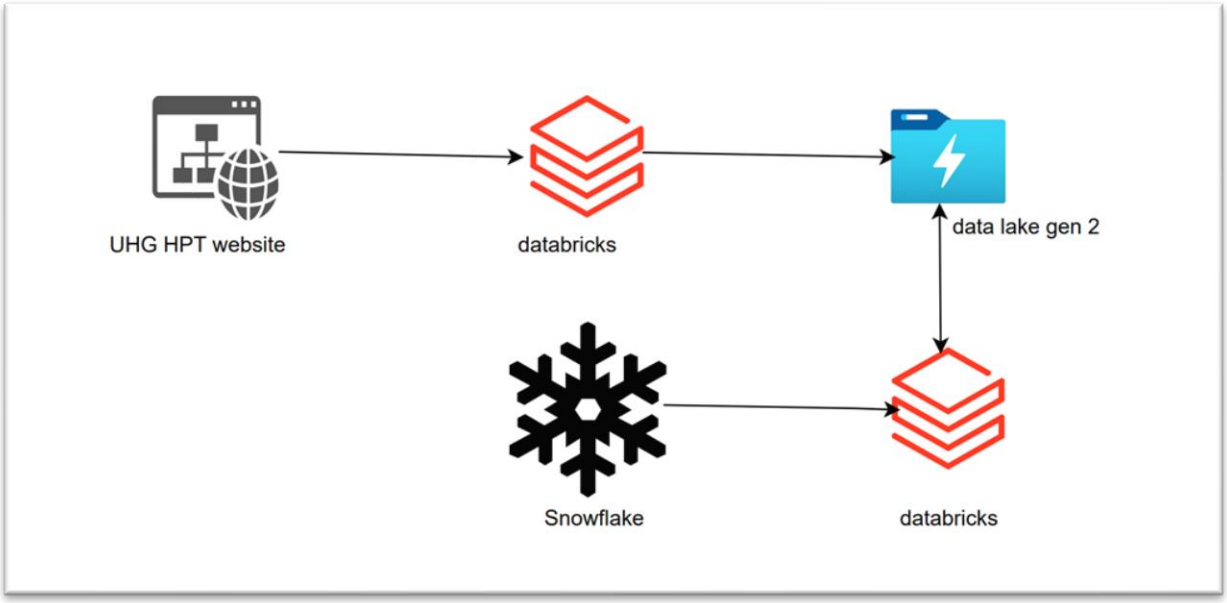
Project Architecture:



Downloading files from website:

1. Selenium Setup:

- A shell script (selenium-install.sh) is created to install the latest version of Chromium and ChromeDriver on the Databricks environment.
- The script checks for the latest version of Chromium from a Google API, downloads both Chromium and ChromeDriver, and installs them.
- Necessary libraries (like libgbm-dev) are installed via apt-get to avoid errors.

2. Databricks and Python Integration:

- Selenium, along with other Python libraries like requests, BeautifulSoup, and Azure SDK (azure.storage.blob), is imported for further tasks.
- After the installation script, %sh command is used to run the setup in the Databricks environment, ensuring that the environment is correctly prepared for Selenium to work.
- After setting up the Selenium environment, the Python interpreter is restarted to load the new libraries. Error handling and system restart using dbutils.library.restartPython() ensures smooth running of the process in the Databricks environment.

3. Azure Datalake Configuration:

- Azure Storage credentials (account name and key) are configured to enable access to an Azure Data Lake container.
- The BlobServiceClient from the Azure SDK is used to create a connection to the specified container (uhgdata).

4. Web Scraping and File Download:

- The code uses Selenium for web automation and BeautifulSoup for parsing HTML data.
- It automates the download of 1000 files from the UHG website, using dynamic scraping techniques to interact with the website.

5. Upload to Azure Data Lake:

- Once files are downloaded and stored temporarily, they are uploaded to the Azure Data Lake using the Blob Storage client.
- The process loops through all the files and uploads them to the specified container in Azure.

Datalake to snowflake:

1. Azure Storage Connection Setup:

- The code starts by configuring the credentials to connect to **Azure Data Lake Storage Gen2**. It uses the **storage account name** and **storage account key** to authenticate and access the ADLS Gen2 storage account where the JSON files are stored.
- Spark is configured to interact with the ADLS Gen2 account. This is done by setting the necessary configuration options, including the storage account's authentication type and the SAS token provider. These configurations enable seamless access to the data stored in the hierarchical file system of ADLS Gen2.

2. Creating a Spark Session:

- A Spark session is created to initialize the PySpark environment. This session acts as an entry point for interacting with Spark functionalities like reading, processing, and writing large-scale data.
- The session is necessary for loading and manipulating data within the Apache Spark framework.

3. Reading JSON Files from Azure Blob Storage:

- The path to the JSON files within Azure Blob Storage is specified. This path includes the container and the storage account details.
- The JSON files are read with the multiline option enabled. This ensures that the code can correctly process JSON files with multiple lines, which is often the case with complex, nested JSON structures.

4. Flattening the JSON Structure:

- JSON files, especially when they involve multiple reporting structures and plans, can be deeply nested. To make the data more usable for analysis, the code flattens the JSON structure:
    - **Exploding Nested Arrays:** The fields reporting_structure and reporting_plans contain arrays, each holding multiple items. Using the explode function, these arrays are expanded into separate rows, so that each reporting plan and its associated information are placed in individual rows.

- **Selecting and Renaming Columns:** After exploding the nested arrays, the relevant fields such as reporting_entity_name, plan_name, plan_id, and others are extracted. The columns are also renamed for clarity and easier handling in downstream processes.
- The purpose of this step is to transform the JSON structure into a flat table format for easier querying and analysis.

5. Displaying and Counting the Flattened Data:

- After flattening the data, the resulting DataFrame is displayed for verification. This helps to check whether the structure is correctly flattened and whether the data matches expectations.
- Additionally, the code counts the number of rows in the DataFrame to validate the size of the dataset being processed.

6. Saving the Flattened Data to Azure Blob Storage:

- The processed data is saved back into **Azure Data Lake Storage Gen2** in **Parquet format**. Parquet is a columnar storage format that is highly efficient in terms of storage space and query performance, making it ideal for large datasets.
- The data is written to a specific output location within the ADLS Gen2 container. The use of the coalesce function ensures that the output is consolidated into a single file, which is beneficial for managing storage and reducing fragmentation.

7. Integrating with Snowflake:

- The next step involves integrating with **Snowflake**, a high-performance cloud data warehouse, to store and analyze the processed data.
- The connection to Snowflake is established using:
  - **URL:** The endpoint for the specific Snowflake instance.
  - **User credentials:** Username and password for authentication.
  - **Database, Schema, and Warehouse:** The database, schema, and virtual warehouse in Snowflake where the data will be loaded.
- The processed DataFrame is then written to Snowflake. It is stored in a table called HPT, and the data is appended to any existing records in that table.

Loaded data in snowflake: