Azure Data Engineering Project Documentation

Table of Contents

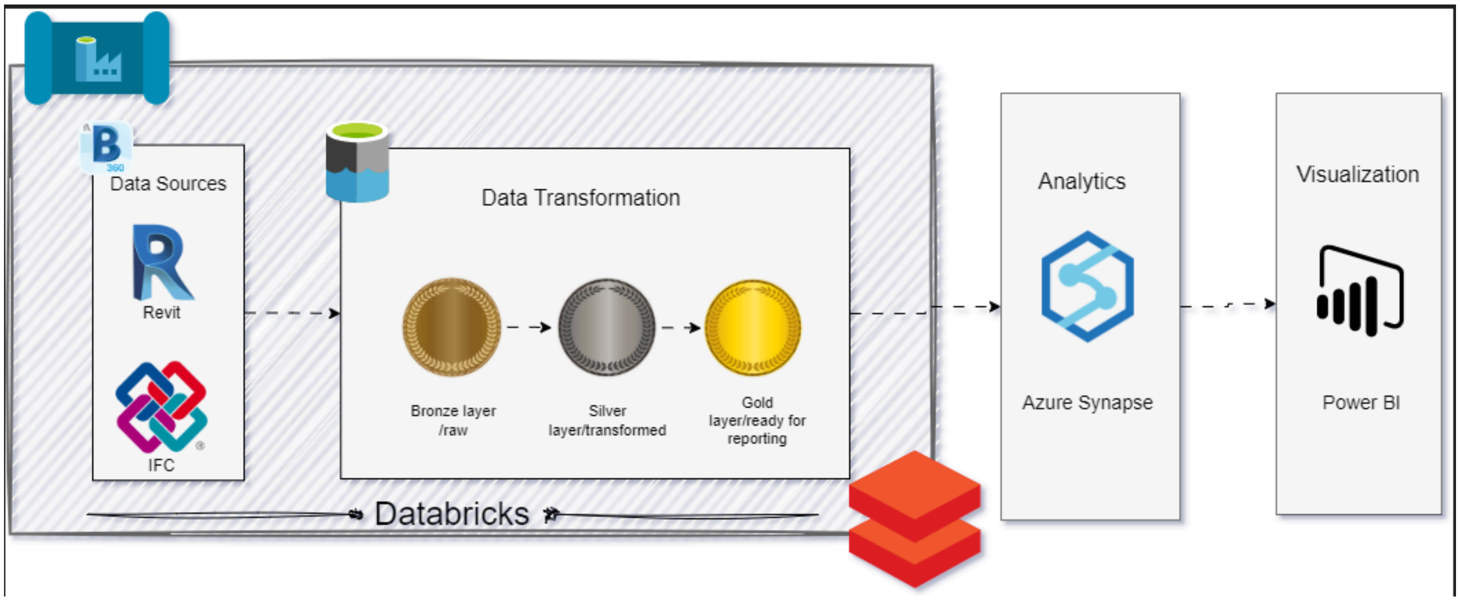
1. Repository Structure

```
BIM360-AzureCloud-Kevee
├── ADF Pipeline.png
├── AEC data architecture.svg
├── file_listing.md
├── Final.png
├── SA Pipeline.png
├── _databricks/
│   ├── 0_Mounting.ipynb
│   ├── 1_Modeldata.ipynb
│   ├── 2_Loadingdata.ipynb
│   ├── 3_Bronze to Silver.ipynb
│   └── 4_Silver to Gold.ipynb
├── _datafactory/
│   ├── publish_config.json
│   ├── readme.md
│   ├── factory/
│   │   └── kkd-learning-ADF.json
│   ├── linkedService/
│   │   ├── AzureDatabricks_kevee.json
│   │   ├── AzureSynapseAnalytics_Kevee.json
│   │   └── serverlesSQL_kevee_sa.json
│   └── pipeline/
│       └── kevee-Revit-PowerBI.json
└── _synapse/
    ├── publish_config.json
    ├── README.md
    ├── credential/
    │   └── WorkspaceSystemIdentity.json
    ├── dataset/
    │   ├── goldTables.json
    │   └── Projects.json
    ├── integrationRuntime/
    │   └── AutoResolveIntegrationRuntime.json
    ├── linkedService/
    │   ├── kevee-bim-revit-sa-WorkspaceDefaultSqlServer.json
    │   ├── kevee-bim-revit-sa-WorkspaceDefaultStorage.json
    │   └── serverlessSQL_kevee.json
    ├── pipeline/
    │   ├── child pipeline to create views.json
    │   └── Main Pipeline.json
    └── sqlscript/
        ├── CreatesQLserverlessView_gold.json
        └── list_of_databases.json
```

2. Pipeline Workflow

This Azure Data Engineering project implements a medallion architecture (Bronze, Silver, Gold) to process Autodesk 360 cloud data. The workflow consists of the following steps:

1. **Data Ingestion**: Raw data from Autodesk 360 cloud sources is ingested using Azure Data Factory pipelines.
2. **Bronze Layer**: The raw data is stored in its original form in the Bronze layer.
3. **Data Processing**: Azure Databricks is used for data transformation and processing through a series of notebooks.
4. **Silver Layer**: The processed and cleaned data is stored in the Silver layer.
5. **Gold Layer**: The business-ready aggregated data is stored in the Gold layer.
6. **Data Serving**: Azure Synapse Analytics is used to create views and make the data available for reporting through Power BI.



3. Databricks Notebooks Documentation

3.1. 0_Mounting.ipynb

Purpose: Sets up the storage mounts required for the Databricks workflows.

Overview:
This notebook is responsible for setting up storage mounts in Azure Databricks, enabling seamless access to Azure Data Lake Storage (ADLS) containers for the medallion architecture (Bronze, Silver, Gold layers). It uses the `dbutils.fs.mount` function to establish connections between Databricks and ADLS Gen2 storage, configures authentication using Azure Active Directory passthrough, and creates mount points for each layer of the architecture.

Flow:

1. **Configuration Setup**: A dictionary (`configs`) is created to define the authentication type and token provider class for accessing ADLS.
2. **Mounting Storage**: The `dbutils.fs.mount` function is called three times to mount the Bronze, Silver, and Gold containers to specific mount points in the Databricks workspace.
3. **Access Verification**: Once mounted, the storage containers can be accessed using the defined mount points (`/mnt/kevee/bronze` , `/mnt/kevee/silver` , `/mnt/kevee/gold`).

Important Functions:

`dbutils.fs.mount`

This function is used to mount an ADLS container to a Databricks workspace. Below is an example of how it is used in the notebook:

```
dbutils.fs.mount(
  source = "abfss://bronze@keveebimdata.dfs.core.windows.net/",
  mount_point = "/mnt/kevee/bronze",
  extra_configs = configs
)
```

Summary: This notebook establishes the structural foundation for data processing by defining consistent schemas and data models that will be used throughout the pipeline.

3.2. 1_Modeldata.ipynb

Purpose: Extracts and processes model data from BIM360 using Autodesk Forge APIs.

Overview:
This notebook defines a `ModelData` class that interacts with Autodesk Forge APIs to retrieve and process model data from BIM360. It authenticates using OAuth2, fetches model versions, metadata, and object properties, and processes the data into structured tables for further analysis.

Flow:

1. **Setup**: Required libraries are imported, including `PyForge` for API interactions and `flatten_dict` for data processing.
2. **Authentication**: The `OAuth2Negotiator` class is used to authenticate with Autodesk Forge and retrieve an access token.
3. **Model Retrieval**: The `ModelData` class fetches model versions, metadata, and object properties using Forge APIs.
4. **Data Processing**: The object tree and properties are processed into structured tables using helper methods like `process_object_tree` and `make_tables`.

Important Functions:

`get_model_version`

Fetches the version of a specific model from BIM360.

```
def get_model_version(self, token, project_id, folder_id, model_name):
    folders_api = FoldersApi(token)
    f_data, f_included = folders_api.get_folder_contents(project_id=project_id, folder_id=folde
    for thing in f_included:
        if thing['attributes']['name'] == model_name:
            return thing
    return None
```

`get_model_object_tree_properties`

Retrieves the object tree and properties for a specific model view.

```
def get_model_object_tree_properties(self, token, model_urn, view_name):
    model_derivative_api = ModelDerivativeApi(token)
    model_metadata_ids = model_derivative_api.get_metadata_ids(model_urn)
    for thing in model_metadata_ids['metadata']:
        if thing['name'].lower().replace(' ', '') == view_name.lower().replace(' ', ''):
            model_object_tree = model_derivative_api.get_object_tree(model_urn, thing['guid'])[
            model_object_properties = model_derivative_api.get_object_properties(model_urn, thi
            return model_object_tree, model_object_properties
    return None, None
```

`process_object_tree`

Processes the object tree and generates structured tables for model elements and types.

```
def process_object_tree(self, model_object_tree, model_object_properties):
    for category in model_object_tree:
        cat_name = category['name']
        type_frame, elem_frame = self.make_tables(cat_name, model_object_tree, model_object_prop
        yield type_frame, elem_frame
```

Summary:
This notebook provides a robust implementation for extracting and processing BIM360 model data using Autodesk Forge APIs. It organizes the data into structured tables, enabling further analysis and integration into downstream workflows

Processes the object tree and generates structured tables for model elements and types.

```
def process_object_tree(self, model_object_tree, model_object_properties):
    for category in model_object_tree:
        cat_name = category['name']
        type_frame, elem_frame = self.make_tables(cat_name, model_object_tree, model_object_prop
        yield type_frame, elem_frame
```

3.3. 2_Loadingdata.ipynb

Purpose: Handles the ingestion of raw data into the Bronze layer by processing BIM360 model data retrieved using the `ModelData` class.

Overview:
This notebook initializes the `ModelData` class to fetch model data from BIM360, processes the data into structured tables, and writes it to the Bronze layer in Azure Data Lake Storage (ADLS). It ensures data versioning and handles updates to existing datasets.

Flow:

1. **Import Dependencies**: Imports the `ModelData` class from `1_Modeldata.ipynb` and initializes a Spark session.
2. **Parameter Setup**: Defines and retrieves input parameters such as project ID, folder ID, model name, view name, and project name using `dbutils.widgets`.

3. **Initialize ModelData**: Creates an instance of the `ModelData` class to fetch model metadata, object tree, and properties.
4. **Data Processing**: Processes the object tree into structured tables (types and elements) using the `process_object_tree` method.
5. **Create Spark DataFrames**: Converts the processed tables into Spark DataFrames for further processing.
6. **Write to Bronze Layer**: Writes the data to the Bronze layer in ADLS, ensuring versioning and handling updates for existing datasets.

Important Functions:

`create_spark_dataframes`

Converts a list of Pandas DataFrames into Spark DataFrames.

```
def create_spark_dataframes(spark, frames):
    spark_frames = [spark.createDataFrame(df) for df in frames]
    return spark_frames
```

Data Writing Logic:
Handles the creation or update of Parquet files in the Bronze layer based on the model version.

- New Tables: Writes new tables to the Bronze layer if they do not already exist.
- Existing Tables: Updates existing tables by appending new data or overwriting rows with the same model version.

```
if v_project_name not in database_name:
    for table, df in dataframe_object.items():
        path = f'mnt/kevee/bronze/{v_project_name}/{table}.parquet'
        df.write.mode('overwrite').parquet(path)
else:
    db_path = f'mnt/kevee/bronze/{v_project_name}'
    dataframe_object_adls = get_dataframe_object(db_path)
    for table in dataframe_object.keys():
        if table not in dataframe_object_adls.keys():
            path = f'mnt/kevee/bronze/{v_project_name}/{table}.parquet'
            dataframe_object[table].write.mode('overwrite').parquet(path)
        else:
            # Handle updates for existing tables
            ...
```

Summary:
This notebook ingests raw BIM360 model data into the Bronze layer, ensuring proper versioning and handling updates. It leverages the ModelData class for data retrieval and processes the data into structured Spark DataFrames for storage in ADLS.

3.4. 3_Bronze to Silver.ipynb

Purpose: Transforms raw data from the Bronze layer to the cleansed Silver layer.

Key Functions:

- Reads data from the Bronze layer
- Applies data cleansing and transformation rules
- Handles data quality issues and anomalies
- Writes processed data to the Silver layer

Summary: This notebook implements the transformation logic that converts raw data into a cleaned, validated, and standardized format in the Silver layer.

3.5. 4_Silver to Gold.ipynb

Purpose: Transforms Silver layer data into business-ready datasets in the Gold layer.

Key Functions:

- Reads data from the Silver layer
- Applies business aggregations and transformations
- Creates dimensional models and fact tables
- Writes final datasets to the Gold layer

Summary: This notebook creates the final business-ready datasets that will be used for reporting and analytics in Power BI through Synapse Analytics.

4. Azure Data Factory Overview

The `_datafactory` directory contains the Azure Data Factory configuration for the orchestration of the data pipeline.

Key Components:

- **Factory Configuration**: Contains the main ADF configuration (`kkd-learning-ADF.json`)

- **Linked Services**: Connections to Databricks, Synapse Analytics, and Serverless SQL
- **Pipeline**: The `kevee-Revit-PowerBI.json` pipeline that orchestrates the overall data flow

Workflow:
The Data Factory pipeline triggers the Databricks notebooks in sequence, orchestrating the movement of data from source systems through the medallion architecture layers. It also handles scheduling, monitoring, and error handling for the overall pipeline execution.

5. Azure Synapse Analytics Overview

The `_synapse` directory contains the Azure Synapse Analytics configuration for data warehousing and serving.

Key Components:

- **Datasets**: Definitions for Gold tables and Projects
- **Linked Services**: Connections to various data sources and services
- **Pipelines**: The main pipeline and child pipeline for view creation
- **SQL Scripts**: SQL serverless scripts for creating Gold views and listing databases

Workflow:
Synapse Analytics consumes the Gold layer data and creates views that are optimized for reporting and analysis. The SQL serverless capabilities allow for flexible querying of the data, while the pipelines automate the creation and maintenance of these views.

6. Conclusion

This Azure Data Engineering project implements a robust data processing pipeline using the medallion architecture. The combination of Azure Data Factory for orchestration, Azure Databricks for processing, and Azure Synapse Analytics for serving creates a scalable and maintainable solution for Autodesk 360 cloud data analytics.

The pipeline follows modern data engineering best practices, including:

- Clear separation of concerns across the medallion layers
- Modular notebook design in Databricks
- Automated pipeline orchestration
- Scalable and performant data serving

This documentation provides an overview of the project structure and components. For detailed implementation details, please refer to the individual notebook files and configuration JSON files in the repository.