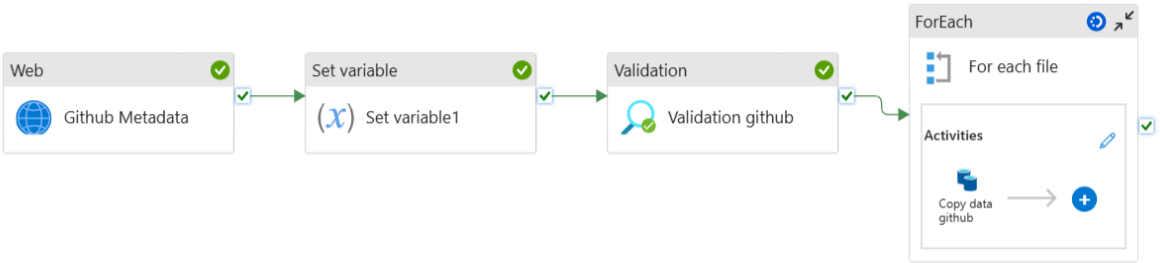
LLD

Low-Level Design (LLD)

1. Ingestion Layer (Azure Data Factory)



- **Pipeline**: Netflix_Ingestion_Pipeline
  - **Activities**:
    - **Copy Data**: Pulls CSVs (e.g., netflix_titles.csv, netflix_cast.csv) from GitHub URLs.
    - **Parameters**:
      - source_file: Dynamic CSV path (e.g., `https://github.com/.../netflix_titles.csv, netflix_cast.csv`).
      - sink_path: Data Lake bronze folder (bronze@medalli0n.dfs.core.windows.net/).
    - **Validation**: Ensures file existence and schema consistency.
  - **Output**: CSVs in bronze layer. (e.g., bronze/netflix_titles/).
  - **Trigger**: Manual or scheduled (e.g., daily, weekly).

2. Bronze Layer (Databricks Autoloader)

- **Notebook**: Bronze_Ingestion

```python
df = spark.readStream.format('cloudFiles')\
    .option("cloudFiles.format", "csv")\
    .option("cloudFiles.schemaLocation", checkpoint_path)\
    .load("abfss://raw@medalli0n.dfs.core.windows.net")
df.writeStream\
    .option("checkpointLocation", checkpoint_path)\
  .trigger(processingTime = '10 seconds')\
  .start('abfss://bronze@medalli0n.dfs.core.windows.net/netflix_titles')
```

- **Features**: Incremental loading, checkpointing, schema evolution (add new columns).
- **Output**: Delta tables (e.g., bronze_netflix_titles, bronze_netflix_cast).

3. Silver Layer (PySpark + Workflows)

- **Notebooks**:
  1. **Lookup Notebook**: Lookup_Silver

```python
files = [
    {
        "sourcefolder" : "netflix_directors",
        "targetfolder" : "netflix_directors"
    },
    {
        "sourcefolder" : "netflix_cast",
        "targetfolder" : "netflix_cast"
    },
    {
        "sourcefolder" : "netflix_countries",
        "targetfolder" : "netflix_countries"
    },
    {
        "sourcefolder" : "netflix_category",
        "targetfolder" : "netflix_category"
    },
]
dbutils.jobs.taskValues.set(key = "my_arr", value = files)
```

- **Purpose**: Defines source-target folder mappings.

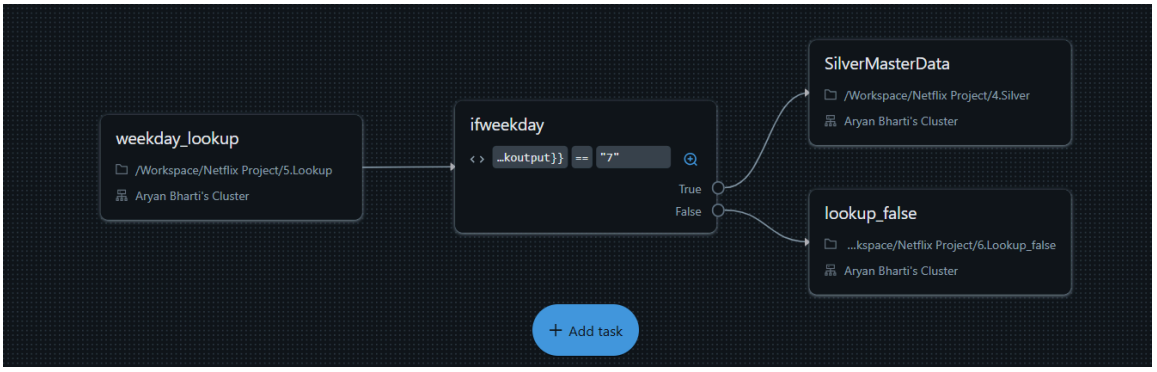
2. **Silver Transformation Notebook**: Silver_Transformation

```python
dbutils.widgets.text("sourcefolder", "netflix_directors")
dbutils.widgets.text("targetfolder", "netflix_directors")
var_src_folder = dbutils.widgets.get("sourcefolder")
var_tar_folder = dbutils.widgets.get("targetfolder")
```

- **Transformations**:

```python
df = df.fillna({'duration_minutes': 0, 'duration_seasons': 1})  # Null handling
df = df.withColumn('duration_minutes', col('duration_minutes').cast(IntegerType()))\
    .withColumn('duration_seasons', col('duration_seasons').cast(IntegerType()))
# Type cast
df = df.withColumn('Short_Titles', split(col('title'), ':')[0]) #Split column
df = df.withColumn('rating', split(col('rating'), '-')[0]) # Split column
df = df.withColumn('flag_check', when(col('type') == 'Movie', 1)\
    .when(col('type') == 'TV Show',1)\
        .otherwise(0))  # Conditional flag
df = df.withColumn("duration_rank", dense_rank().over(Window.orderBy(desc("
duration_minutes"))))  # Ranking
```

- **Workflow**: Silver_Workflow



  - **Tasks**:
    - **Lookup_Locations**: Runs Lookup_Silver, outputs my_array.

```python
dbutils.widgets.text('weekday','7')
var = int(dbutils.widgets.get('weekday'))
dbutils.jobs.taskValues.set(key='weekoutput', value = var)
```

    - **Silver_Notebook**: For-each loop over my_array, passes parameters to Silver_Transformation.
    - **Conditional Task**: Weekday_Lookup checks weekday(); if 7 (Sunday), runs Silver_Transformation, else prints day.

- **Output**: Transformed Delta tables (e.g., silver/netflix_titles).

4. Gold Layer (Delta Live Tables)

- **Notebook**: DLT_Gold

```python
import dlt
from pyspark.sql.functions import lit

# Lookup table rules for data quality
lookuptables_rules = {
    "rule1": "show_id IS NOT NULL"
}

# Master data rules for the gold layer
masterdata_rule = {
    "rule1": "newflag IS NOT NULL",
    "rule2": "show_id IS NOT NULL"
}

# Netflix Directors table
@dlt.table(name="netflixdirectors")
@dlt.expect_all_or_drop(lookuptables_rules)
def netflix_directors():
    df = spark.readStream.format("delta").load(
        "abfss://silver@medalli0n.dfs.core.windows.net/netflix_directors"
    )
    return df

# Netflix Cast table
@dlt.table(name="netflixcast")
@dlt.expect_all_or_drop(lookuptables_rules)
def netflix_cast():
    df = spark.readStream.format("delta").load(
        "abfss://silver@medalli0n.dfs.core.windows.net/netflix_cast"
    )
    return df

# Netflix Countries table
@dlt.table(name="netflixcountries")
@dlt.expect_all_or_drop(lookuptables_rules)
def netflix_countries():
    df = spark.readStream.format("delta").load(
        "abfss://silver@medalli0n.dfs.core.windows.net/netflix_countries"
    )
    return df

# Netflix Category table
@dlt.table(name="netflixcategory")
@dlt.expect_all_or_drop({"rule1": "show_id IS NOT NULL"})
def netflix_category():
    df = spark.readStream.format("delta").load(
        "abfss://silver@medalli0n.dfs.core.windows.net/netflix_category"
    )
    return df

# Gold Staging Netflix Titles table
@dlt.table
def gold_stg_netflixtitles():
    df = spark.readStream.format("delta").load(
        "abfss://silver@medalli0n.dfs.core.windows.net/netflix_titles"
    )
    return df

# Gold Transformed Netflix Titles view
@dlt.view
def gold_trans_netflixtitles():
    df = spark.readStream.table("LIVE.gold_stg_netflixtitles")
    df = df.withColumn("newflag", lit(1))
    return df

# Gold Netflix Titles table with master data rules
@dlt.table
@dlt.expect_all_or_drop(masterdata_rule)
# Corrected 'except_all_or_drop' to 'expect_all_or_drop'
def gold_netflixtitles():
    df = spark.readStream.table("LIVE.gold_trans_netflixtitles")
    return df
```

- **Pipeline**: DLT_Gold_Pipeline
    - **Output**: Gold Delta tables with dropped invalid records.
- **Features**: Streaming tables, views, and data quality enforcement.

Error Handling

- **ADF**: Retry on file fetch failure.
- **Databricks**: Debug via event logs (e.g., syntax errors in DLT).
- **DLT**: Drop invalid records; fail pipeline on critical errors.

Performance Considerations

- **Autoloader**: Checkpointing for efficient incremental loads.
- **Workflows**: Task parallelization via for-each loops.
- **DLT**: Minimizes recomputation with streaming updates.