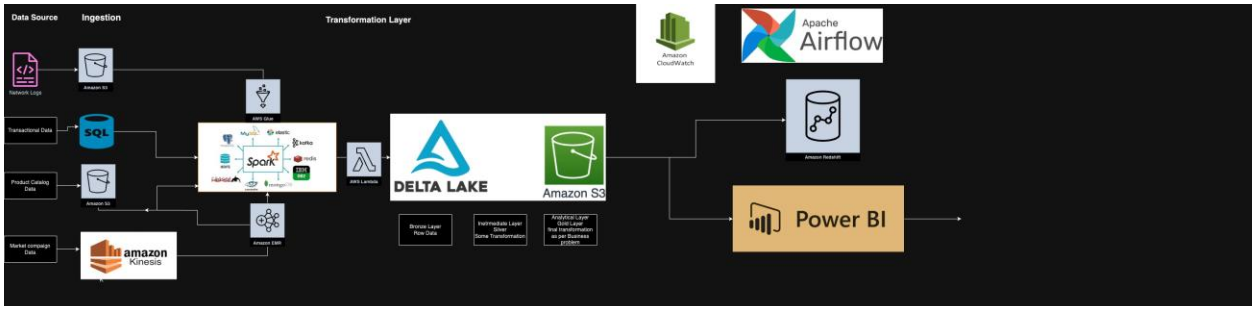

` Data flow diagrams

To implement an Apache Spark pipeline using AWS Glue to transform and integrate data from multiple sources, you can follow these steps:

Step 1: Create a Dynamic Frame

Create a Dynamic Frame from the data sources using AWS Glue. For example, to read data from an S3 bucket:

```
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext

glueContext = GlueContext(SparkContext.getOrCreate())

datasource_s3 = "s3://my-bucket/data/"
dynamic_frame = glueContext.create_dynamic_frame_from_options(
    connection_type="s3",
    connection_options={"paths": [datasource_s3]},
    format="json"
)
```

Step 2: Transform the Data

Transform the data using Apache Spark's DataFrame API. For example, to handle missing values and remove duplicates:

```
from pyspark.sql.functions import col
```

```
dynamic_frame.toDF().fillna("unknown").dropDuplicates(["column_name"])
```

Step 3: Integrate Data from Multiple Sources

Integrate data from multiple sources by joining the Dynamic Frames. For example, to join two Dynamic Frames:

```
dynamic_frame1 = glueContext.create_dynamic_frame_from_options(
   connection_type="s3",
   connection_options={"paths": ["s3://my-bucket/data1/"]},
   format="json"
)

dynamic_frame2 = glueContext.create_dynamic_frame_from_options(
   connection_type="s3",
   connection_options={"paths": ["s3://my-bucket/data2/"]},
   format="json"
)

joined_dynamic_frame = dynamic_frame1.toDF().join(dynamic_frame2.toDF(),
"common_column")
```

Step 4: Write the Transformed Data

Write the transformed data to a target location, such as an S3 bucket or a Redshift table. For example, to write to an S3 bucket:

```
joined_dynamic_frame.toDF().write.parquet("s3://my-bucket/output/")
```

Apache Airflow DAGs for Orchestration and Automation

The following Apache Airflow DAG is used for orchestration and automation:

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
```

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'tart_date': datetime(2024, 6, 15),
    'etries': 1,
    'etry_delay': timedelta(minutes=5),
}

dag = DAG(
    'telcocorp_pipeline',
    default_args=default_args,
    schedule_interval=timedelta(days=1),
)

def run_pipeline(**kwargs):
    # Run the pipeline code here
    pass

run_pipeline_task = PythonOperator(
    task_id='run_pipeline',
    python_callable=run_pipeline,
    dag=dag
)
```