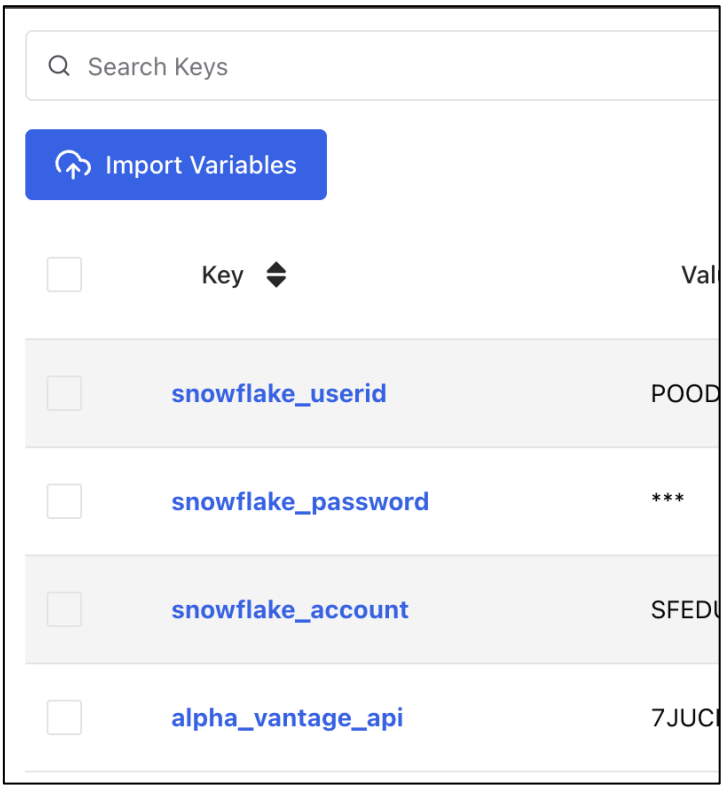
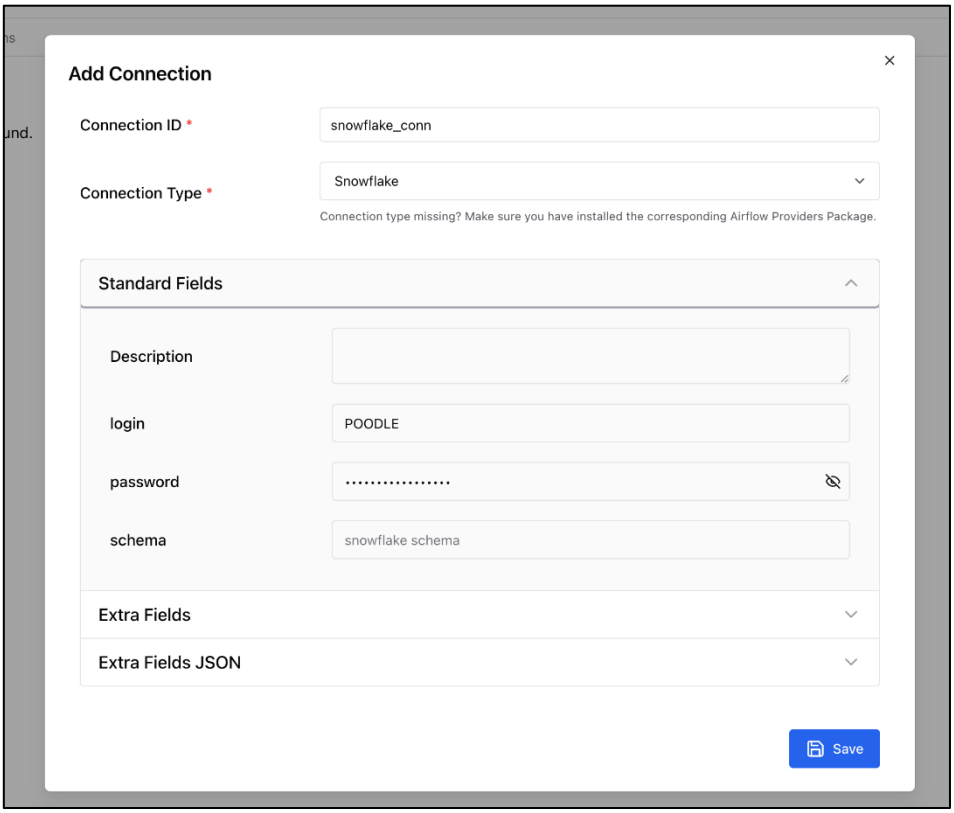
Airflow DAG for Full refresh on stock price using Alpha Vantage API

- **Set up required variables:**



- **Set up Snowflake connection:**



Extra Fields JSON

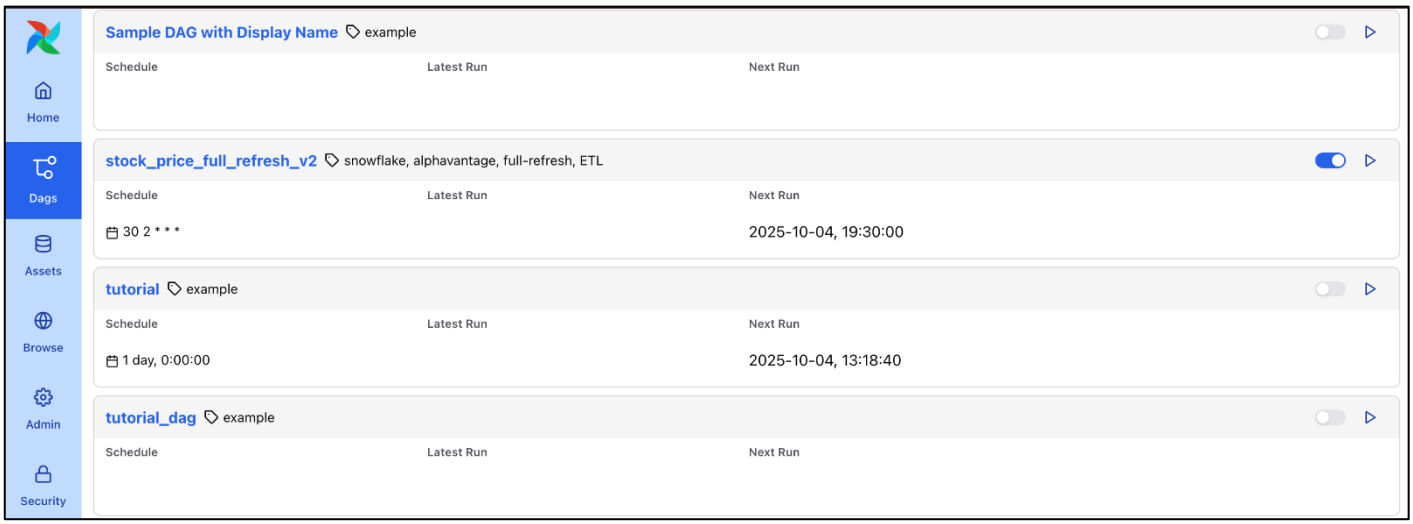
```
{
  "account": "SFEDU02-LVB17920",
  "warehouse": "POODLE_QUERY_WH",
  "database": "USER_DB_POODLE",
  "region": null,
  "role": null,
  "private_key_file": null,
  "private_key_content": null,
  "insecure_mode": false
}
```

Standard Fields

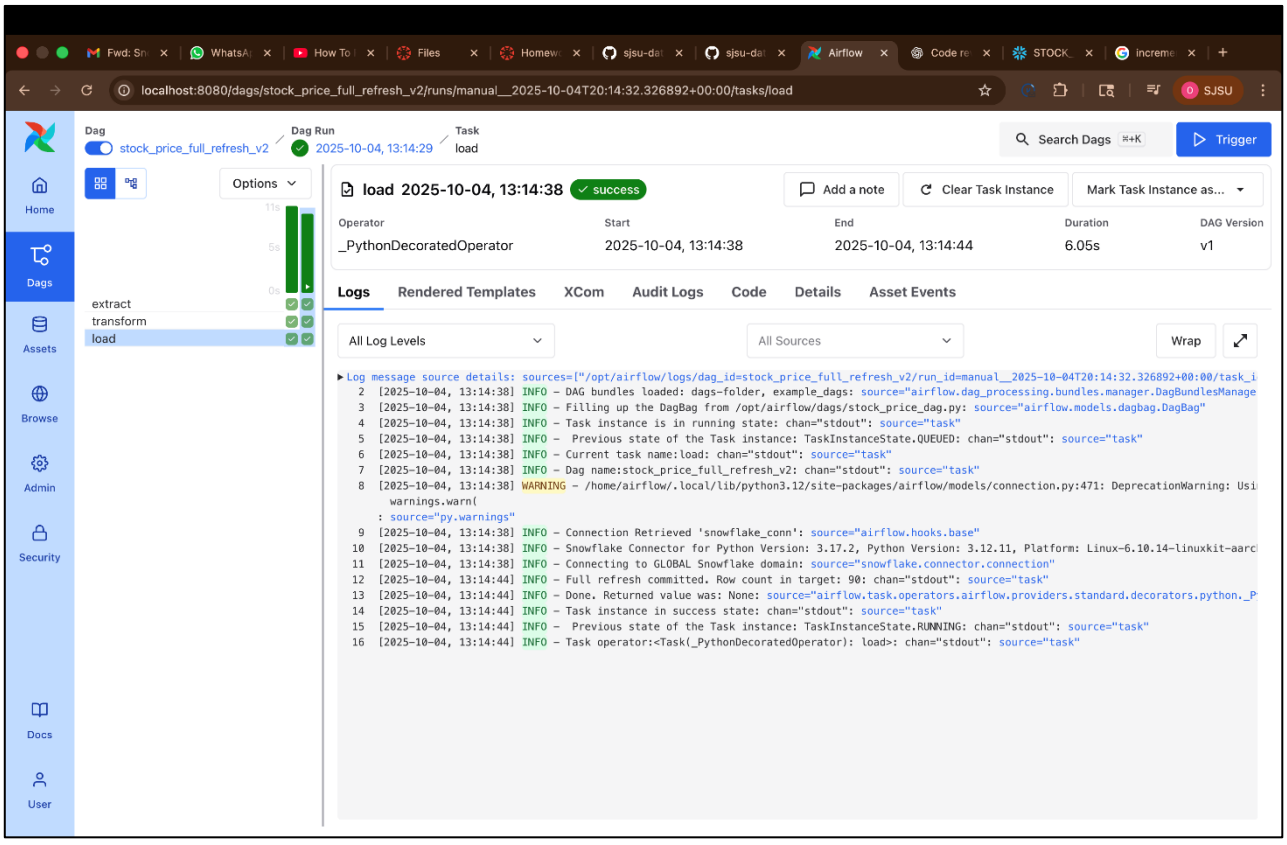
Extra Fields

Account	SFEDU02-LVB17920
Warehouse	POODLE_QUERY_WH
Database	USER_DB_POODLE
Region	
Role	
Private key (Path)	
Private key (Text)	
Insecure mode	Turns off OCSP certificate checks

- **Airflow Homepage showing my DAG “stock_price_full_refresh_v2”:**



- **Log screen of my DAG (Load Operation):**



- **Code Snippets:**

```python
SYMBOL = "AAPL"
TARGET_TABLE = "USER_DB_POODLE.RAW.STOCK_PRICE_DAG"

VANTAGE_API_KEY = Variable.get("alpha_vantage_api")

def return_snowflake_conn():
    hook = SnowflakeHook(snowflake_conn_id = "snowflake_conn")
    conn = hook.get_conn()
    return conn.cursor()

```

Establishing snowflake connection and accessing vantage api key

```python
@task
def extract(symbol: str) -> str:
    url = (
        "https://www.alphavantage.co/query"
        f"?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={VANTAGE_API_KEY}"
    )
    resp = requests.get(url, timeout = 60)
    resp.raise_for_status()
    return resp.text

```

Extract task

```python
@task
def transform(text: str):
    """
    Parse JSON and produce list of tuples:
    (OPEN, HIGH, LOW, CLOSE, TRADE_VOLUME, TRADE_DATE, SYMBOL)
    Limited to the last 90 calendar days, newest first. Casts numeric types.
    """
    payload = json.loads(text)
    if "Time Series (Daily)" not in payload:
        raise RuntimeError(f"Alpha Vantage response missing daily series: {payload}")

    series = payload["Time Series (Daily)"]
    sorted_dates = sorted(series.keys(), reverse=True)[:90]

    rows = []
    for d in sorted_dates:
        v = series[d]
        rows.append((
            float(v["1. open"]),
            float(v["2. high"]),
            float(v["3. low"]),
            float(v["4. close"]),
            int(v["5. volume"]),
            d,
            SYMBOL,
        ))
    return rows
```

Transform Task

```python
@task
def load(records):
    """
    Full refresh using a SQL transaction:
      1) Ensure table exists
      2) BEGIN
      3) TRUNCATE TABLE
      4) Bulk INSERT
      5) COMMIT  (or ROLLBACK on error)
    """
    if not records:
        raise ValueError("No rows to load; aborting to avoid truncating to empty table.")

    cur = return_snowflake_conn()
    conn = cur.connection
    try:
        cur.execute(f"""
            CREATE TABLE IF NOT EXISTS {TARGET_TABLE} (
                OPEN NUMBER,
                HIGH NUMBER,
                LOW NUMBER,
                CLOSE NUMBER,
                TRADE_VOLUME NUMBER,
                TRADE_DATE DATE,
                SYMBOL VARCHAR,
                PRIMARY KEY (TRADE_DATE, SYMBOL)
            )
        """)
```

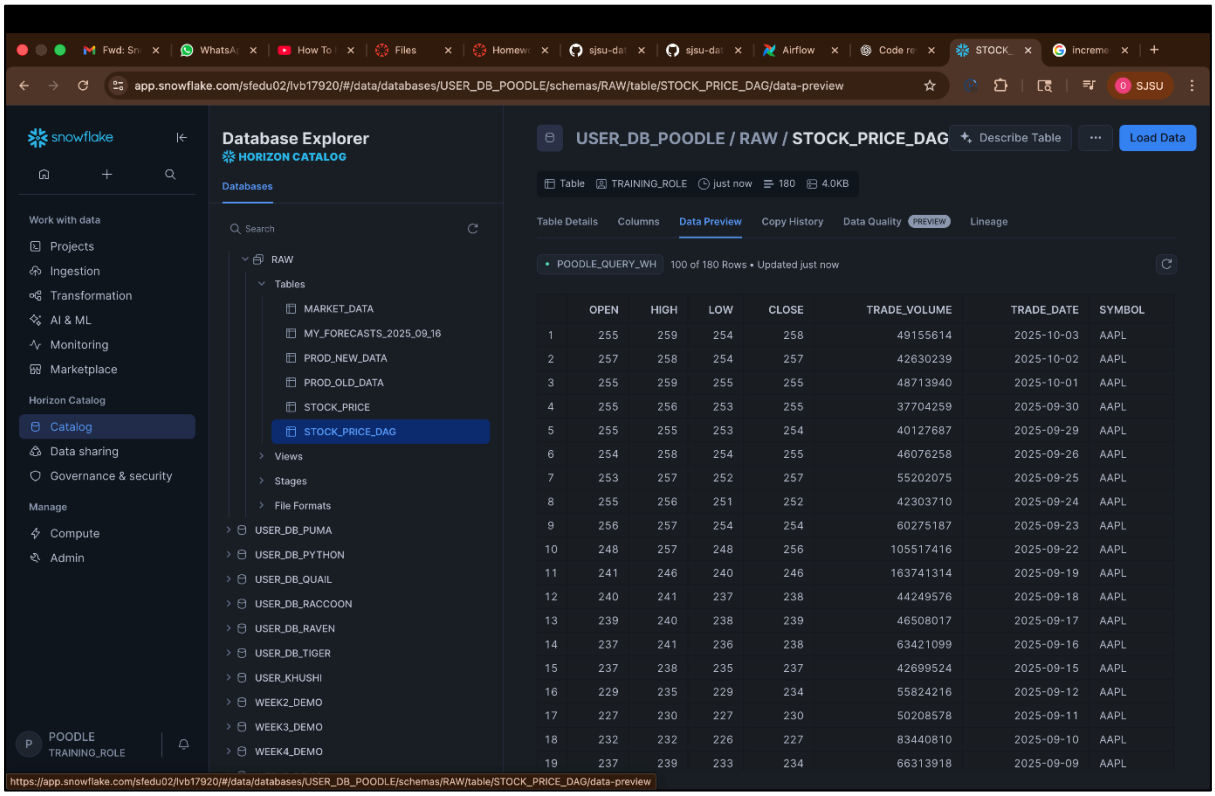
```python
        # TRUNCATE for a true full refresh
        cur.execute(f"TRUNCATE TABLE {TARGET_TABLE}")

        # Bulk insert
        insert_sql = f"""
            INSERT INTO {TARGET_TABLE}
            (OPEN, HIGH, LOW, CLOSE, TRADE_VOLUME, TRADE_DATE, SYMBOL)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        cur.executemany(insert_sql, records)

        cur.execute("COMMIT")

        cur.execute(f"SELECT COUNT(*) FROM {TARGET_TABLE}")
        count = cur.fetchone()[0]
        print(f"Full refresh committed. Row count in target: {count}")

    except Exception as e:
        try:
            cur.execute("ROLLBACK")
        except Exception:
            pass
        print("Error during full refresh:", e)
        raise
    finally:
        try:
            cur.close()
```

Load Task (Full Refresh using SQL transactions)

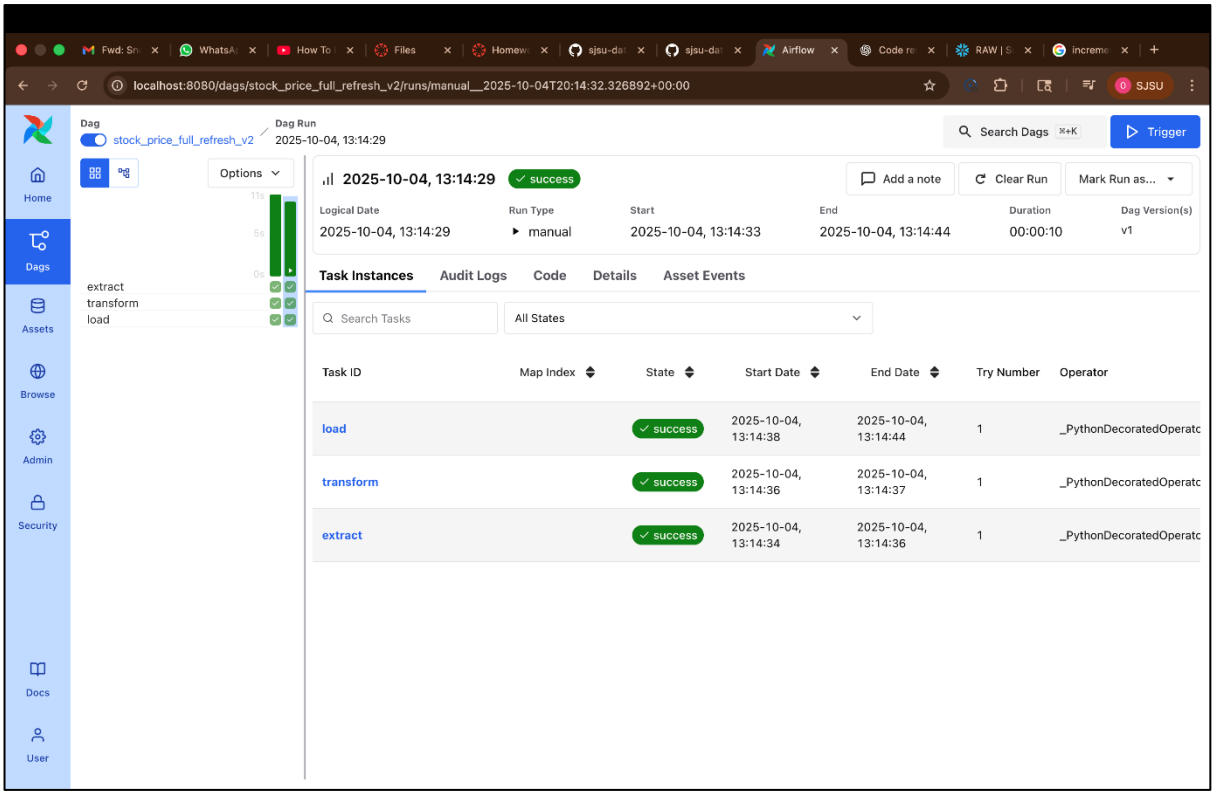
```python
with DAG(
    dag_id='stock_price_full_refresh_v2',
    start_date=datetime(2025, 10, 4),
    catchup=False,
    tags=['ETL', 'alphavantage', 'snowflake', 'full-refresh'],
    schedule='30 2 * * *',
    default_args={
        "owner": "data-eng",
        "retries": 2,
        "retry_delay": timedelta(minutes=5),
    },
    description="Fetch last 90d AAPL prices and full-refresh load into Snowflake using a SQL transaction",
) as dag:

    data = extract(SYMBOL)
    rows = transform(data)
    load(rows)

```

Creating “stock_price_full_refresh_v2” DAG

- **Extra Screenshots:**



Snowflake UI showing the output table “STOCK_PRICE_DAG” after successful execution of airflow DAG



Airflow UI showing successful execution of my ”stock_price_full_refresh_v2” DAG

- Omkar Rajale