ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Harnessing Apache Airflow Operators for Enhanced Workflow Automation

Pankaj Dureja

USA

ABSTRACT

This paper explores the utilization of Apache Airflow operators to enhance workflow automation, specifically focusing on the Python, MySQL, Oracle, Oracle Stored Procedure, Bash, and Rabbit MQ operators. Using these operators, companies can standardize and automate their data workflows, resulting in increased productivity and performance. The post emphasizes the implementation, advantages, and future possibilities to work with these operators in Apache Airflow. The paper concludes with an examination of how Apache Airflow operators can be employed as part of workflow automation, alongside the breadth they span and their adoption for the future.

***Corresponding author**

Pankaj Dureja, USA.

Received: October 16, 2023; **Accepted:** October 23, 2023; **Published:** October 30, 2023

Keywords: Apache Airflow, Workflow Automation, Python Operator, MySQL Operator, Oracle Operator, Oracle Stored Procedure Operator, Bash Operator, Rabbit MQ Operator, Data Pipelines, Task Scheduling, Workflow Management

Introduction

Today's data-driven organizations and the use of complex data processing require workflow automation. Apache Airflow is a popular open-source, flexible, and scalable automation platform for orchestrating these workflows. Why is Airflow strong - it has predefined tasks, called Operators, which can be ordered into Directed Acyclic Graphs (DAGs) of workflows. This paper focuses on each specific operators of Python, MySQL, Oracle, Oracle Stored Procedure, Bash, and Rabbit MQ and how to use them to automate your complex workflow.

Problem Statement

Working within an oil and gas company, managing and automating data workflows can be exceptionally challenging due to the diversity and complexity of the tasks involved. Despite the availability of various tools for task scheduling and workflow management, many organizations face significant difficulties in integrating and automating these diverse tasks. Traditional approaches often result in fragmented systems where different tools handle different parts of the workflow, leading to inefficiencies and difficulties in managing complex dependencies.

For instance, an organization might use a scheduler with robust capabilities for executing Unix commands, SQL queries, or stored procedures but lacks the ability to run Python scripts or manage message queuing. This limitation forces the organization to seek customized solutions from different vendors, which can be costly and inefficient. In such cases, organizations may end up using multiple scheduling tools to meet their varied requirements, which complicates workflow management and increases operational overhead.

Moreover, having multiple scheduling systems often leads to increased costs, as each tool might require separate licensing, maintenance, and support services. This not only strains the budget but also creates a fragmented environment where maintaining coherence and synchronization across different tools becomes a major challenge. The lack of a unified scheduling and automation platform can lead to significant delays, errors, and resource wastage.

Therefore, there is a pressing need for a comprehensive solution that can seamlessly integrate different types of tasks-whether they involve Unix commands, SQL queries, stored procedures, Python scripts, or message queuing-into a single, cohesive workflow. Such a solution would not only streamline operations and reduce costs but also enhance the overall efficiency and reliability of the data workflows within the organization.

Solution Implemented

To address these challenges, this paper implements a solution using Apache Airflow operators:

PythonOperator: Executes Python functions, providing flexibility for custom scripting and data processing tasks.

Following library needs to imported in the DAG:

from airflow.operators.python_operator import PythonOperator

Sample PythonOperator Code

```
    for params in parameters:
        push_task = PythonOperator(
            task_id=f'push_timestamp_{params["pv_schema_name"]}_{params["pv_table_name"]}',
            python_callable=push_timestamp_to_xcom,
            op_kwargs=params,
        )
```

```
def push_timestamp_to_xcom(ti, pv_schema_name, pv_table_name, **kwargs):
    hook = MySqlHook(mysql_conn_id='ms_database_conn')
    sql = f"""
    SELECT MAX(update_ts) AS update_ts
    FROM schema_dba.table_load_control
    WHERE schema_name = UPPER('{pv_schema_name}')
    AND table_name = UPPER('{pv_table_name}')
    AND office_nb = 0
    """
    result = hook.get_first(sql=sql)
    if result:
        update_ts = result[0]
        update_ts_str = update_ts.strftime('%Y-%m-%d %H:%M:%S') if isinstance(update_ts, datetime) else str(update_ts)
        ti.xcom_push(key=f'update_ts_{pv_schema_name}_{pv_table_name}', value=update_ts_str)
```

MySqlOperator: Facilitates interaction with MySQL databases, allowing for seamless execution of SQL queries.

Following library needs to imported in the DAG:
from airflow.providers.mysql.operators.mysql import MySqlOperator

Sample MySqlOperator Code

```
cst_now = datetime.now(tz=pytz.timezone('America/Chicago')).strftime('%Y-%m-%d %H:%M:%S')
upd_end_ts_qry = f"CALL upd_table_load_control_prc('{tbl['pv_source_schema_name'].upper()}','{tbl['pv_table_name'].upper()}', 0, '{cst_now}', {update_id})"
update_table_control = MySqlOperator(
    task_id=f'update_table_control_{tbl['pv_source_schema_name']}_{tbl['pv_table_name']}',
    sql=upd_end_ts_qry,
    mysql_conn_id='ms_database_dba_conn',
)
```

OracleOperator: Provides connectivity and operations for Oracle databases, enhancing data integration capabilities.

Following library needs to imported in the DAG:
from airflow.providers.oracle.operators.oracle import OracleOperator

Sample OracleOperator Code

```
oracle_task = OracleOperator(
    task_id='oracle_task',
    sql='SELECT * FROM my_oracle_table;',
    oracle_conn_id='my_oracle_connection',
)
```

OracleStoredProcedureOperator: Executes stored procedures in Oracle, enabling efficient database operations.

Following library needs to imported in the DAG:
from airflow.providers.oracle.operators.oracle import OracleStoredProcedureOperator

Sample OracleStoredProcedureOperator Code

```
oracle_generate_update_id_task = OracleStoredProcedureOperator(
    task_id="oracle_generate_update_id_task",
    oracle_conn_id="ora_connection_to_database_schema",
    procedure="schema_name.package_name.kp_get_update_id",
    parameters={
        "pv_desc": "ProductionLoad",
        "pv_type": "BATCH_JOB",
        "pn_office_nb": "0",
        "pn_user_id": "0",
        "pn_update_id": int
    }
)
```

BashOperator: Executes Bash scripts, providing a means to perform system-level operations.

Following library needs to imported in the DAG**:**
from airflow.operators.bash_operator import BashOperator

Sample PythonOperator Code

```
with TaskGroup(f"oracle_extract_{params['pv_schema_name']}_{params['pv_table_name']}") as oracle_extract:
    task_1 = EmptyOperator(task_id=f"start_oracle_extract_{params['pv_schema_name']}_{params['pv_table_name']}")
    task_2 = BashOperator(
        task_id=f"run_extract_{params['pv_schema_name']}_{params['pv_table_name']}",
        bash_command=f'scripts/{params["pv_schema_name"]}_{params["pv_table_name"]}_incr_extract.sh',
        env={'CDC_PASSWORD': oracle_cdc_password,
             'CDC_TIMESTAMP': f'{{{{ ti.xcom_pull(task_ids="push_timestamp_{params["pv_schema_name"]}_{params["pv_table_name"]}"
                              f', key="update_ts_{params["pv_schema_name"]}_{params["pv_table_name"]}") }}}}'}
    )
    task_1 >> task_2
```

RabbitMQOperator: Integrates with Rabbit MQ for message queuing, supporting asynchronous task execution and communication.

Following library needs to imported in the DAG**:**
from airflow.providers.rabbitmq.operators.rabbitmq import RabbitMQOperator

Sample MySqlOperator Code

```
def send_message_to_rabbitmq(**kwargs):
    try:
        print(message)
        task_publish_message = RabbitMQOperator(
            task_id='publish_message_to_queue',
            exchange='itc_es',  # Name of the exchange in RabbitMQ
            routing_key='ie_async_1008_10_stg',   # Name of the queue to which message will be published
            message=message,  # Message payload
            rabbitmq_conn_id='rabbitmq_default_stage'
        )
        task_publish_message.execute(context=kwargs)
    except AirflowException as e:
        # Handle Airflow-related exceptions
        print(f"Airflow exception occurred: {str(e)}")
    except Exception as e:
        # Handle other exceptions
        print(f"An unexpected exception occurred: {str(e)}")
```

Potential Extended Use Cases

ETL & Machine Learning Pipelines

Leveraging Python Operator to Automating the extraction, transformation, and loading (ETL) of data from various sources into a centralized database & orchestrate machine learning model training and deployment.

Data Integration

- Combining MySQL, Oracle Operator & Oracle Stored Procedure Operator to perform complex data integration tasks across different database systems.
- Scheduling and running regular data quality checks on MySQL tables to ensure data integrity.
- Integrating data from multiple Oracle databases into a data warehouse for comprehensive reporting.
- Running nightly batch jobs that process large volumes of transactional data using pre-defined Oracle stored procedures.

System Monitoring and Maintenance

Utilizing Bash Operator to automate system maintenance tasks such as log file cleanup and backups on Unix-based servers.

Real-Time Data Processing

Using Rabbit MQ Operator to handle real-time sensor data from oil rigs to processing systems, ensuring timely data availability for analysis.

Impact

Apache Airflow Operators play a big role in the performance and reliability of data workflows within an organization. Not only that, since Airflow serves as a common platform across those types of work and it meets most requirements, with Airflow, you no longer need a wide range of task scheduling system, you can achieve the same goal but with much lower cost, as well as a unified point of failure, and fewer system to be upgraded and maintained. It improves Automation with less error, rapid task performance, and proper Resource Consumption. This also leads to better workflow management and more pulse on your business - not only is Airflow more performant when it comes to scheduling and executing tasks, the feature of handling complex dependencies between tasks and having a good monitoring / alerting solution also help with the organization agility when it comes to meeting the business demands.

Scope

The scope of this paper is to provide a detailed examination of how specific Apache Airflow operators can be used to enhance workflow automation. It covers the implementation details, benefits, and potential use cases, but does not delve into other aspects of Apache Airflow, such as user interface customization or advanced security configurations. Future research can explore these additional dimensions to provide a more comprehensive understanding of Apache Airflow's capabilities.

Conclusion

Utilizing Apache Airflow operators for more advanced workflow automation will be a valuable way to work with complex data workflows. The operators covered in this paper deliver a meaningful value in terms of flexibility, scalability and efficiency. By unifying these operators together in a single workflow system, organizations can take advantage of powerful automation features, better task scheduling, and advanced workflow management capabilities. The possible use-cases to be extended make it very flexible and useful for specific domains, which means that a wide variety of use-cases can be implemented in Apache Airflow [1-7].

References

1. Maxime Beauchemin (2021) The Apache Airflow Book. O'Reilly Media 45-70.
2. Holden Karau, Rachel Warren (2017) High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark. O'Reilly Media 105-130.
3. Ian Pointer (2019) Programming PySpark. O'Reilly Media 78-95.
4. Anirudh Kala (2020) Apache Airflow: A Real-World Guide to Data Pipelines. Packt Publishing 115-140.
5. Wes McKinney (2017) Python for Data Analysis. O'Reilly Media 220-245.
6. Operators. Apache Airflow https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/operators.html.
7. Airflow Operators. Astronomer Docs https://www.astronomer.io/docs/learn/what-is-an-operator.