Introduction to Orchestration ETL Processes With Apache Airflow

Miroslav Tomić, Teaching Assistant

University of Novi Sad
Faculty of Technical Sciences
November 2023

Agenda

- Introduction
- Apache Airflow
- Tutorial Environment
- ETL Process Examples

Introduction

Introduction

- Apache Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows
- Workflows can be build with almost any technology, which is possible because of Airflow's extensible Python framework
- State of workflows are managed with help of web interface
- One of the main characteristic of Airflow workflows is that all workflows are defined in Python code
- "Workflows as code" serves several purposes:
  - Dynamic
  - Extensible
  - Flexible

Introduction

- Snippet of code

```
from datetime import datetime
from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:

    # Tasks are represented as operators
    hello = BashOperator( task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

    # Set dependencies between tasks
    hello >> airflow()
```

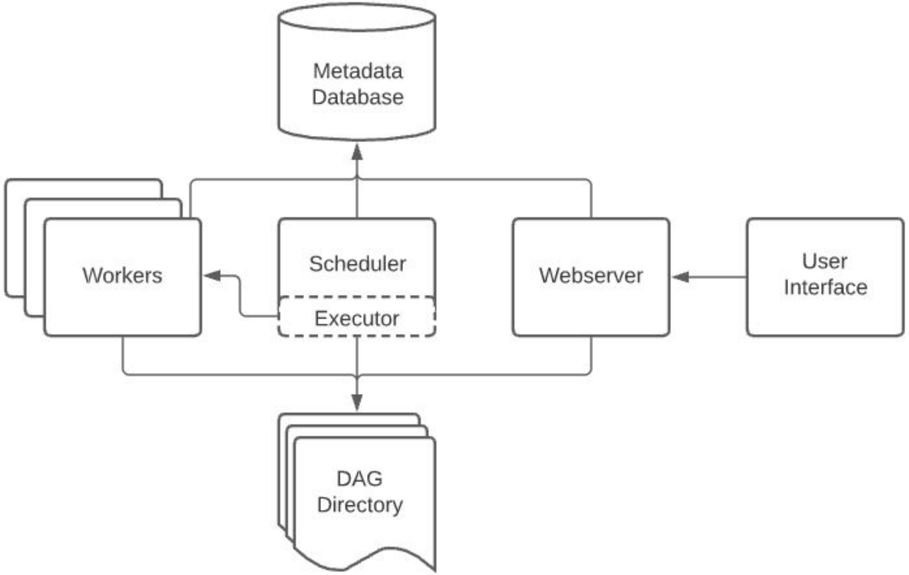
Apache Airflow

Apache Airflow – Installation

- It is possible to perform multiple types of installation
  - using released sources to run it on local machine as standalone solution
  - using PyPi
  - Using docker images
  - Using official Airflow Helm Chart
  - Using 3rd-party images, charts, deployments
- Prerequisites
  - Python: 3.8, 3.9, 3.10, 3.11
  - Databases
    - PostgreSQL: 11, 12, 13, 14, 15, 16
    - MySQL: 5.7, 8.0, 8.1
    - SQLite: 3.15.0+ - parallelization not possible
    - MSSQL(Experimental): 2017, 2019 - deprecated
  - Kubernetes: 1.23, 1.24, 1.25, 1.26, 1.27
    - If used with kubernetes

Apache Airflow – Architecture Overview

- A workflow is represented as a DAG (a Directed Acyclic Graph) and contains individual pieces of work called Tasks
- A DAG specifies the dependencies between Tasks and the order in which to execute then and run retries
- An Airflow installation consists of following:
  - scheduler
  - executor
  - webserver
  - dag files
  - metadata database



Apache Airflow – Architecture Overview

- Workloads
  - A DAG runs through a series of Tasks
- Control flow
  - DAGs are designed to be run many times, and multiple runs of them can happen in parallel
  - DAGs are parameterized
- User interface
  - Airflow comes with a user interface that let us see what DAGs and their tasks are doing, trigger runs of DAGs, view logs, and do some limited debugging and resolution of problems with DAGs

Apache Airflow – DAGs

- A DAG (Directed Acyclic Graph) is the core concept of Airflow, collecting Tasks together, organized with dependencies and relationships to say how they should run
- DAG declaration
  - context manager
  - standard constructor
  - dag decorator
- DAG arguments
  - https://airflow.apache.org/docs/apache-airflow/2.7.2/_api/airflow/models/dag/index.html#airflow.models.dag.DAG
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/dags.html#declaring-a-dag

Apache Airflow – Tasks

- A Task is the basic unit of execution in Airflow
- Tasks are arranged into DAGs, and then have upstream and downstream dependencies set between them into order to express the order they should run in
- There are three basic kinds of Task:
    - Operators
    - Sensors
    - TaskFlow-decorated task
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/tasks.html

Apache Airflow – Operators

- An Operator is conceptually a template for a predefined Task
- Operators can be defined declaratively inside DAG
- Airflow has a very extensive set of operators but most popular are
  - BashOperator
  - PythonOperator
  - EmailOperator
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/operators.html

Apache Airflow – Sensors

- Sensors are special type of Operator that are designed to wait for an event
- It can be time-based, or waiting for a file, or an external event
- Sensors are primarily idle they have two different modes of running
    - poke (default)
    - reschedule
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/sensors.html

Apache Airflow – TaskFlow

- If code is mostly written using Python instead of Operators then the TaskFlow API will make it much easier to author clean DAGs without extra boilerplate, all using the @task decorator
- TaskFlow takes care of moving inputs and outputs between Tasks using XComs
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/taskflow.html

Apache Airflow – Executor

- Executors are the mechanism by which task instances get run
- They have a common API and are "pluggable" this means they can be swapped based on installation needs
- To check which executor is currently set
  - airflow config get-value core executor
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/executor/index.html

Apache Airflow – XComs

- XComs (short for "cross-communications") are a mechanism that let Tasks talk to each other
- By default Tasks are entirely isolated and may be running on entirely different machines
- An XCom is identified by
  - key – its name
  - task_id
  - dag_id it
- XComs can have any (serializable) value, but they are only designed for small amounts of data
- The shouldn't be used to pass around large values like dataframes!
- XComs are explicitly "pushed" and "pulled" to/from their storage using the xcom_push and xcom_pull methods on Task Instances
- XComs can be used in templates
  - SELECT * FROM {{ task_instance.xcom_pull(task_ids='foo', key='table_name') }}
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/xcoms.html

Apache Airflow – Variables

- Variables are Airflow’s runtime configuration concept - a general key/value store that is global and can be queried from tasks
- They can be easily set with Airflow’s user interface or bulk-uploaded as a JSON file
- They can be used in
  - code – by importing Variable model and calling get on it
  - templates
- Variables are global and they shouldn't be used to pass data between tasks they should be used for configuration that covers the entire installation
- According to airflow documentation it is recommended to keep most of setting and configuration in DAG files so it can be versioned using source control. **Variables are only for values that are truly runtime-dependent**
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/variables.html

Apache Airflow – Params

- Params are used to provide runtime configuration to tasks
- Default Params can be configured in DAG code and also additional Params can be added
- It is possible to overwrite Param values at runtime when DAG is triggered
- Params are validated using JSON Schema that should be provided
- For scheduled DAG runs, default Param values are used
- https://airflow.apache.org/docs/apache-airflow/2.7.2/core-concepts/params.html

Tutorial environment

Tutorial Environment

- Docker is used to simulate real scenario
- Project is divided in different "units" that are containerized
    - airflow
        - used as orchestrator
        - contains docker config, airflow config and DAGs
    - mssql
        - used as source database
        - contains docker config, database backups and script for restoring backup
    - postgres
        - used as target database
        - contains docker config and script for creation of initial schemas and tables

ETL Process Examples

ETL Process Examples

- Idea is to demonstrate usage of Apache Airflow for creation and orchestration of ETL processes on examples that are easily understandable
- For that purpose examples should be downloaded from
  - https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms
- Two backups should be downloaded:
  - AdventureWorks2017.bak
  - AdventureWorksDW2017.bak
- After download they should be put inside mssql/backups directory

ETL Process Examples

- First example is DAG that is called
    - dsc_load_currency
    - Purpose of this DAG is to demonstrate initial filing of dimension table in OLAP database
- Second example is DAG that is called
    - dsc_load_currency_sensor
    - Purpose of this DAG is to demonstrate update of dimension table in OLAP database
- Third example is DAG that is called
    - dsc_load_currency_rate
    - Purpose of this DAG is to demonstrate filling of fact table in OLAP database and usage of params and variables in airflow
- Purpose of examples is to discuss about apache airflow code, concepts, role and other possibilities

References

- https://airflow.apache.org/docs/apache-airflow/2.7.2/index.html
- https://github.com/Microsoft/sql-server-samples/tree/master/samples/databases/adventure-works
- https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms

The end!

Thank you for your attention!

DSC EUROPE