Hands-On with Apache Airflow

Session 2

SESSION OVERVIEW

01 Airflow setup/installation on EC2 instance

02 Main motivations behind various operators in Airflow

03 Airflow CLI and UI walkthrough

04 Airflow operators and their demonstrations

IMPORTANT CONFIGRATIONS

- When you first setup airflow, it creates a file called airflow.cfg
- This file contains Airflow's configurations which can be easily edited to better suit ones requirements
- You can see these configurations from the CLI using:
  - **airflow config** command

IMPORTANT CONFIGRATIONS

Config Name	Description	Sample Values
dags_folder	• The folder where your airflow pipelines live • This path must be absolute	/home/ubuntu/airflow/dags
executor	• The type of executor that airflow should use. • Choices include SequentialExecutor, LocalExecutor, CeleryExecutor	CeleryExecutor
sql_alchemy_conn	• The SqlAlchemy connection string to the metadata database	postgresql+psycopg2://airflow:airflow@localhost/airflow
parallelism	• The amount of parallelism as a setting to the executor • This defines the max number of task instances that should run simultaneously for the airflow installation	32
dag_concurrency	• The number of task instances allowed to run concurrently by the scheduler	16
max_active_runs_per_dag	• The maximum number of active DAG runs per DAG	16
base_url	• The base URL of your website as airflow cannot guess what domain or cname you are using.	http://localhost:8080
broker_url	• The Celery broker URL. Celery supports RabbitMQ, Redis and experimentally a sqlalchemy database.	redis://127.0.0.1:6379/1

OPERATORS IN AIRFLOW

- Operators **determine what actually gets done** by a task.
- Operators describe **a single task in a workflow** which can be a shell script, a Hive query, a Python function etc.
- The DAG ensures that operators run in the correct order
- They may run on two completely different machines.
- Airflow has a feature of **operator cross communication** known as **Xcom**, which allows two operators to combine or share resources or files
- Operators are only loaded by Airflow if they are assigned to a DAG.

OPERATOR DEFINITION

```
# Import modules related to DAG and operators
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.operators.hive_operator import HiveOperator

# Create a DAG object with necessary configs
ml_dag = DAG('sample', description='Python ML',
          schedule_interval='0 12 * * *',
          start_date=datetime(2017, 3, 20))

# Create task objects with desired task operators
hive_task = HiveOperator(task_id='load_hive_tables', dag=ml_dag, ...)
python_task = PythonOperator(task_id='get_query_result', dag=ml_dag, ...)

# Create task dependencies
hive_task >> python_task
```

BASH OPERATOR

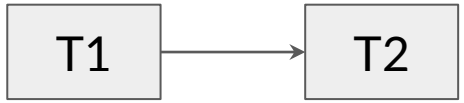
- The **BashOperator** is used for running any Shell command or script with your DAG.
- It is one of the most versatile and widely used operators in Airflow
- Arguments:
  - **task_id :** The ID for the task
  - **bash_command :** The shell command/script to be executed
  - **dag :** The DAG to which this task belongs

PYTHON OPERATOR

- The **PythonOperator** is used to execute Python callables.
- Arguments:
  - **python_callable :** the function name to be called
  - **op_args/op_kwargs :** to pass additional arguments to the Python callable

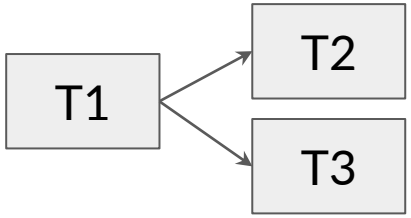
TASK DEPENDENCIES

- If task T1 should finish before T2 starts, you can do this using:
  - T1.set_downstream(T2)
  - T1 >> T2
  - T2.set_upstream(T1)
  - T2 << T1



- You can also define a chain of dependencies at once:
  - T1 >> T2 >> T3 instead of T1 >> T2 and T2 >> T3

- You can set multiple dependencies at once:
  - T1 >> [T2, T3]



PYTHON OPERATOR DAG

SQOOP OPERATOR

- The **SqoopOperator** is used to execute a Sqoop job
- It is usually used to transfer data between RDBMS and HDFS
- Arguments :
  - **conn_id :** connection ID
  - **table :** MySQL table name
  - **cmd_type :** Command type
  - **target_dir :** Target directory for HDFS

HIVE OPERATOR

- The **HiveOperator** is used to connect to Hive using hive_conn_id and execute Hive queries
- Arguments:
  - **hql :** Hive query to be executed
  - **hive_cli_conn_id :** Hive connection

SPARK OPERATOR

- The Spark operators are used to schedule spark jobs using Airflow
- **SparkSubmitOperator** launches Spark using the spark-submit CLI on the airflow machine
- It supports all the configurations and arguments needed by spark-submit
- To run Spark SQL, we can make use of **SparkSqlOperator**
- Arguments for SparkSubmitOperator :
  - **application :** The application to be submitted as a job, either jar or py file
  - **conn_id :** Connection ID
  - **application_args:** Arguments for the application being submitted

EMAIL OPERATOR

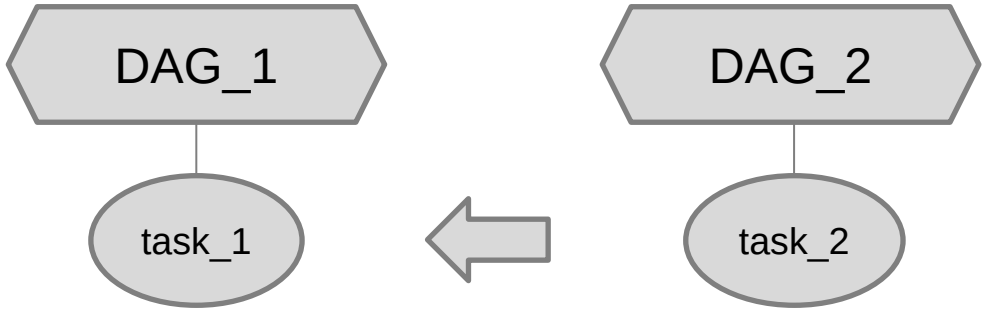
- The **EmailOperator** is used for alerting task events like task completion, task failure, task retry, task fail on retry
- It is also used to send attachments along with emails with error messages or logs
- Arguments :
  - **to :** list of receiver emails
  - **subject :** subject for the email
  - **html_content :** content of the email, html markup is used
  - **files :** files to attach in the email

BASE OPERATOR

- The **BaseOperator** is the abstract base class for all operators, so data members and methods declared in this class available to all the operators
- It contains methods such as set_downstream() and set_upstream()
- You can create a custom operator by extending the **airflow.models.baseoperator.BaseOperator** and then overriding the execute() method
- Some important arguments received by the BaseOperator include :
  - task_id, owner, email, email_on_retry, retries, depends_on_past, dag, params, default_args, queue, sla, execution_timeout, on_failure_callback, on_success_callback, on_retry_callback, trigger_rule, run_as_user, task_concurrency, do_xcom_push

SENSOR OPERATOR

- Sensors are useful to check for a certain condition before we run a task.
- Eg: we have a task in a DAG depending on a task in another DAG or we have a task which should run only after a file has arrived in HDFS
- Sensor tasks continue to execute at a time interval and succeed when a criteria is met or fail when they timeout
- Arguments:
  - **soft_fail :** if True marks the task as SKIPPED on failure
  - **poke_interval**: interval between each try (> 1 minute recommended)
  - **timeout :** time before the sensor fails



SENSOR OPERATOR EXAMPLES

- **WebHdfsSensor**
  - Waits for a file or folder to land in HDFS
- **ExternalTaskSensor**
  - Waits for a different DAG or a task in a different DAG to complete for a specific execution_date
- **TimeDeltaSensor**
  - Waits for a timedelta after the task's execution_date +schedule_interval
- **TimeSensor**
  - Waits until the specified time of the day

SESSION SUMMARY

01 Airflow Setup

02 Airflow Configurations

03 UI Walkthrough

04 Operators in Airflow with individual demonstrations

THANK YOU