

Big Data Technologies

Agenda

- Apache Airflow

Apache Airflow

Airflow Terminologies

- DAG: Directed Acyclic Graph
 - Vertices are tasks and Edges are dependencies
 - Can be parallelized.
- Operators
 - Single dedicated task in workflow/DAG.
 - Example: Bash command, Python Function, Database Operation, Email send, etc.
- Task
 - Instantiated operator and assigned to some worker.
 - Failed task can be retried.
- Workflow
 - Sequence of task arranged in control dependencies.
 - Workflow and DAG words are interchangeable.

Airflow Workflow

- Airflow workflows are written as Python code in form of DAG.
- DAG includes multiple tasks in form of operators.
- Operators are connected in complex sequence.
- DAG implemented in Python, but can execute variety of tasks.
- Programs written in any language.
 - Analytical/ML task
 - ETL task

- Big Data components/programs
- DAGs are more dynamic, manageable, testable and collaborative.
- Airflow is job scheduler that execute the tasks on defined time and/or followed by its dependencies.

Airflow Advantages

- Dynamic DAG
- Implemented in code - flexible
- Can be configured: parallelism, params, templates
- Extensible
- Plenty of operators/executors
 - Shell script, Big data task, OS command, etc.
 - Custom task/plugins
- Scalable
- Modular architecture
- Can handle any number of DAGs (with multi-node cluster)
- Configurable
 - airflow.cfg - admin settings
 - Centralized configurations
- Monitoring
 - Elegant Web UI
- Handle failures (n retries)
- Email alerts
- Open Source
- Active community
- New plugins

Airflow Installation

- Airflow is installed as Python package (Refer installation document).
- By default Airflow use AIRFLOW_HOME as "\$HOME/airflow/" directory.
- \$AIRFLOW_HOME

- airflow.cfg
 - dags folder
 - timezone
 - executor type
 - parallelism
- dags/
 - dag definition (python files)
- logs/
 - logs of airflow components
- airflow.db
 - using sqlite (default) database for airflow metastore (by default).
 - Recommended to use mysql/postgre-sql for production.
- webserver_config.py
 - Web server config like security, email, theme, etc.

Airflow Operators

- DAG only describe how to run the workflow, but do not perform actual computation.
- Operator describes single task in workflow.
- Operator characteristics
 - Atomic (usually) - standalone (not sharing resources with other operators)
 - Idempotent - produce same result for each run.
- Operator in execution (instantiated) is referred as task.
- Inherited from airflow's BaseOperator class.
- Operators can communicate using XCom.
- Operator categories
 - Sensor operators - Wait for certain criteria
 - HdfsSensor, FileSensor, ...
 - Transfer operators - Transfer the data
 - MySqlToHiveOperator, ...
 - Action operators - Do specified task
 - BashOperator, PythonOperator, HiveOperator, MySqlOperator, EmailOperator, ...

Airflow Architecture

- Metadata
 - RDBMS that stores historical and current DAGs and tasks details.
 - Also maintains information used resources.
 - Default RDBMS used is SQLite, but it is single-user database.
 - In production MySQL or Posgre-SQL is preferred option.
- Scheduler
 - Instructs and trigger task execution on worker node.
 - Implemented into Python.
 - Reads DAG file, task config and schedules the task on worker nodes in sequence.
 - It monitor tasks state from metadata and handle the task failure (as per config).
- Web Server
 - Flask-based UI for Airflow to monitor DAG.
 - Communicate with metadata to fetch the task state.
 - Task state is rendered in various formats including DAG, graphs, time/numbers, etc.
- Executor
 - Carry the task scheduled by the scheduler.
 - Runs on Worker node.
 - Various types:
 - Sequential (default) - For testing
 - Local - For single node cluster - Use python process.
 - Celery - Recommended for multi-node cluster.
 - Job queue written in Python.
 - Used in scalable distributed system.
 - Dask
 - Mesos
 - Kubernetes

Airflow - Single node cluster

- Single worker node.

- Not scalable (max all resources available on the system).
- Quite fast for limited number of DAGs and data size.
- Use LocalExecutor.
- Direct communication (via db) between scheduler and executor.

Airflow - Multi node installation

- Master node runs scheduler and web-server.
- Multiple worker nodes - running executors.
- Highly scalable for huge data and many nodes.
- Recommended executor is Celery.
- Scheduler and workers communicate with external queue system like Rabbit MQ / Redis.

Airflow Working

- Scheduler periodically pings for DAG folder and communicate with metastore.
- If any DAG is available for execution, scheduler starts a DAG run for it. DAG run is an object representing an instantiation of DAG.
- Scheduler update DAG state as "Running".
- For each task, task object is instantiated. The task state is updated as "Scheduled".
- Scheduler assigns priority to each task in DAG (as per config) and (Executor) push them into queuing system. The task state is updated to "Queued".
- Worker pull the task from queue, set its state as "Running" and start executing it.
- Upon completion of each task, task state is updated as succeed or failed.
- When all tasks in DAG run are executed, status of DAG run is updated as succeed or failed.
- Airflow web-server periodically get the data from meta-store and render it for users.
- If any new DAG is found in dags folder, scheduler begin its execution (as above).