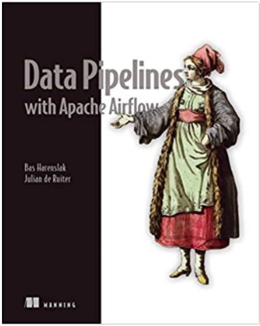
7.1: Orchestration with Airflow

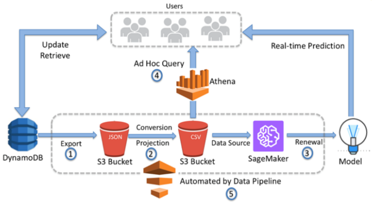
Instructor: Dr. GP Saggese - gsaggese@umd.edu

- Concepts in the slides
- Airflow tutorial
- Web resources
- Documentation
- Tutorial
- Mastery
- Data Pipelines with Apache Airflow



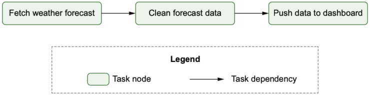
Workflow Managers

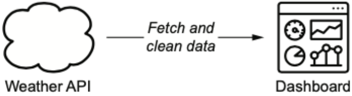
- **Data pipelines** move/transform data across stores
- **Orchestration problem** = coordinate jobs across systems
  - Run tasks on schedule
  - Run tasks in order (dependencies)
  - Monitor tasks
    - Notify devops on failure
    - Retry on failure
    - Track runtime
  - Meet real-time constraints
  - Scale performance



Workflow Managers

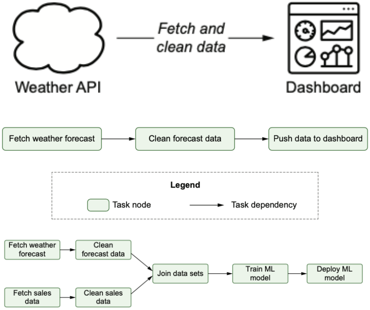
- **E.g., live weather dashboard**
  - Fetch weather data from API
  - Clean/transform data
  - Push data to dashboard/website
- Problems
  - Schedule tasks
  - Manage task dependencies
  - Monitor functionality and performance
  - Add machine learning quickly
  - Complexity increases quickly





Workflow Managers

- **Workflow managers address orchestration problem**
  - E.g., airflow, Luigi, Metaflow, make, cron
- **Represent data pipelines as DAGs**
  - Nodes are tasks
  - Direct edges are dependencies
  - Execute task when all ancestors executed
  - Execute independent tasks in parallel
  - Re-run failed tasks incrementally
- **Describe data pipelines**
  - Static files (e.g., XML, YAML)
  - Workflows-as-code (e.g., Python in Airflow)
- **Provide scheduling**
  - Describe what and when to run
- **Provide backfilling and**



Airflow

- Developed at AirBnB in 2015
  - Open-sourced as part of Apache project
- Batch oriented framework for building data pipelines (not streaming)
- Data pipelines
  - Represented as DAGs
  - Described as Python code
- Scheduler with rich semantics
- Web-interface for monitoring
- Large ecosystem
  - Support many DBs
  - Many actions (e.g., emails, pager notifications)
- Hosted and managed solution
  - Run Airflow on your laptop (e.g., in tutorial)
  - Managed solution (e.g., AWS)

Airflow: Execution Semantics

- **Scheduling semantic**
  - Define next scheduling interval
    - E.g., "every day at midnight", "every 5 minutes on the hour"
  - Similar to **cron**
- **Retry**
  - Re-run task after failure to recover from intermittent issues
- **Incremental processing**
  - Divide time into intervals per schedule
  - Execute DAG for data in that interval only
- **Catch-up**
  - Run all missing intervals up to now (e.g., after downtime)
- **Backfilling**
  - Execute DAG for past schedule intervals
  - E.g., re-process data after pipeline changes

Airflow: What Doesn't Do Well

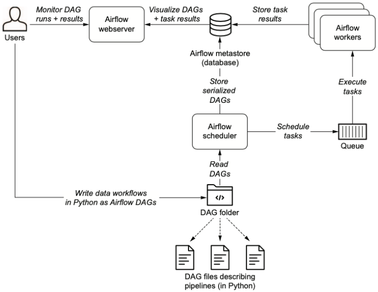
- **Not great for streaming pipelines**
  - Better for recurring batch tasks
  - Time is discrete, not continuous
    - E.g., schedule hourly, not process data continuously

- **Prefer static pipelines**
  - DAGs should remain consistent between runs
- **No data lineage**
  - No automatic tracking of data transformation
  - Implement manually
- **No data versioning**
  - No automatic tracking of data updates
  - Implement manually

SCIENCE ACADEMY

Airflow: Components

- **Users (DevOps)**
- **Web-server**
  - Visualize DAGs
  - Monitor DAG runs and results
- **Metastore**
  - Keep system state
  - Track executed DAG nodes



- **Scheduler**
  - Parse DAGs
  - Track completed dependencies
  - Add tasks to execution queue
  - Schedule tasks
- **Queue**
  - Tasks ready for execution
  - Tasks picked up by Workers
- **Workers**
  - Pick up tasks from Queue
  - Execute tasks

SCIENCE ACADEMY

Airflow: Concepts

- Each DAG run represents a data interval, i.e., an interval between two times
  - E.g., a DAG scheduled **@daily**
  - Each data interval starts at midnight for each day, ends at midnight of next day
- DAG scheduled after data interval has ended
- Logical date
  - Simulate the scheduler running DAG / task for a specific date
  - Even if it is physically run now

Airflow: Tutorial

- Follow Airflow Tutorial in README
- From the tutorial for Airflow

Airflow: Tutorial

- Script describes DAG structure as Python code
  - No computation inside DAG code
  - Defines DAG structure and metadata (e.g., scheduling)

airflow/example_dags/tutorial.py — view source

```python
from datetime import datetime, timedelta
from textwrap import dedent

# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG

# Operators; we need this to operate!
from airflow.operators.bash import BashOperator
```

- **Scheduler** executes code to build DAG
- `BashOperator` creates task wrapping Bash command

Airflow: Tutorial

- Dict with various default params to pass to the DAG constructor
  - E.g., different set-ups for dev vs prod
- Instantiate the DAG

airflow/example_dags/tutorial.py view source

```python
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}
```

airflow/example_dags/tutorial.py view source

```python
with DAG(
    'tutorial',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval=timedelta(days=1),
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=['example'],
) as dag:
```

Airflow: Tutorial

- DAG defines tasks by instantiating `Operator` objects
  - Default params passed to all tasks
  - Can be overridden explicitly
- Use a Jinja template
- Add tasks to the DAG with dependencies

airflow/example_dags/tutorial.py view source

```python
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    retries=3,
)
```

airflow/example_dags/tutorial.py view source

```python
templated_command = dedent(
    """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7)}}"
    echo "{{ params.my_param }}"
{% endfor %}
"""
)

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
)
```

```python
t1 >> [t2, t3]
```

SCIENCE ACADEMY