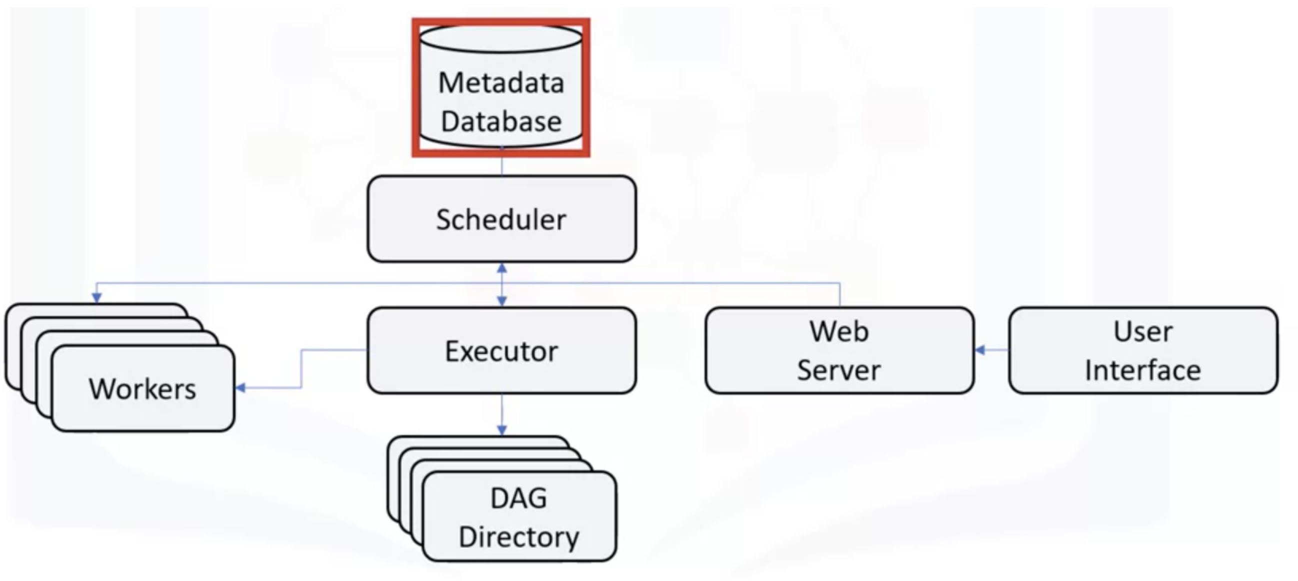
Week3_ApacheAirflow

- A workflow is represented as a DAG (Direct Acyclic Graph) and composed individual pieces of codes called tasks arranged in dependencies.
- Airflow is not a data streaming solution, it is a dataflow manager
- Airflow comes with a built-in Scheduler, which handles the triggering of all scheduled workflows
- The scheduler is reponsible for submitting individual tasks from each scheduled workflow to the executor
- The executor handles the running of these teasks by assigning them to workers
- The webserver serves airflow UI
- The DAG Directory hosts all of your dags
- Airflow hosts a Metadata Database which is usde by the Scheduler, Executor and the Web server to store the state of each state of each DAG and its tasks



Cons:
Pure python = flexibility

Useful UI = full insight of tasks

Integration = Plug and play

Easy to use = Unlimited Pipeline Scope

Open Source = Community of Developers

Scalable (Modular architecture)
Dynamic

Extensible = you can define your own operators
Lean

- **DAG**

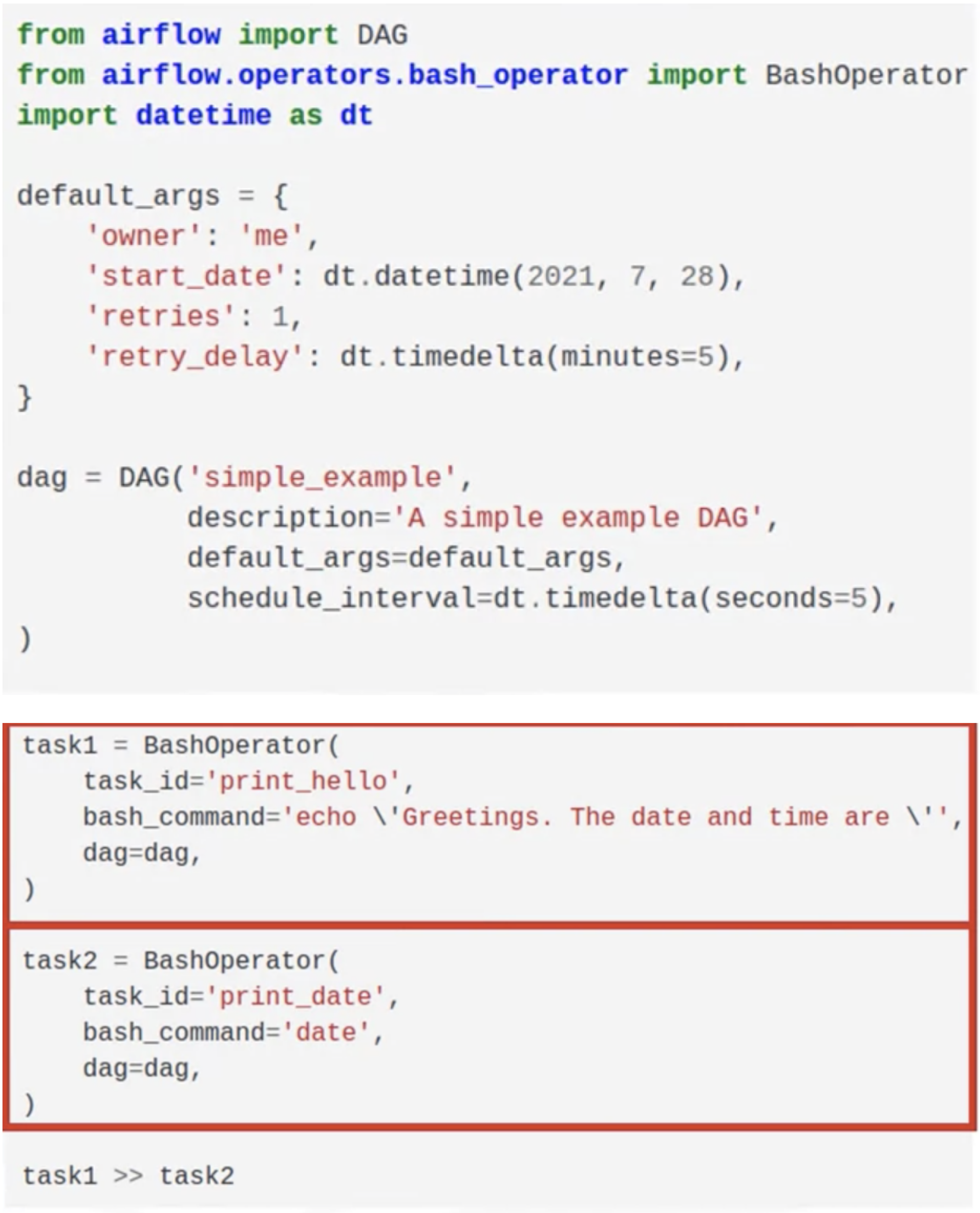
Directed acyclic graph
Consists of nodes and graphs, but each edge has a direction with no loops

DAGs are used to represent workflows:

- Nodes are tasks
- Edges are dependencies
- Tasks run in order
- Defined as Code (Python script)
- Scheduling instructions are specified in the DAG
- Tasks are written in Python
- Tasks implement operators, for example, Python, sql, or bash operators
- Operators determine what each task does
- Sensor operators poll for a certain time or condition
- Other operators include email and HTTP request operators

DAG Definition Components

Library imports
DAG arguments
DAG definition
Task definitions
Task pipeline



```python
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt

default_args = {
    'owner': 'me',
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5),
}

dag = DAG('simple_example',
          description='A simple example DAG',
          default_args=default_args,
          schedule_interval=dt.timedelta(seconds=5),
)

task1 = BashOperator(
    task_id='print_hello',
    bash_command='echo \'Greetings. The date and time are \'',
    dag=dag,
)

task2 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)

task1 >> task2
```

Airflow Scheduler

- Deploys on worker array
- Follows your dag
- First dag run
- Subsequent runs

List all dags

`airflow dags list`

List tasks in a DAG

`airflow tasks list dag_name`

Unpause a dag

`airflow dags unpause tutorial`

Pause a DAG

`airflow dags pause tutorial`

Building a DAG

`simple_example_DAG.py`

- Python library imports

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt
```

- **DAG arguments**

```
default_args = {

'owner': 'me',
'start_date': dt.datetime(2021, 7, 28),
'retries': 1,
'retry_delay': dt.timedelta(minutes=5)
}
```

- **DAG definition**

```
dag = DAG('simple_example',
description = 'A simple example DAG',
default_args = defaults_args,
schedule_interval = dt.timedelta(seconds=5)
)
```

- **Task Definition**

```
task1 = BashOperator(
task_id = 'print_hello',
bash_command = 'echo\'Greetings. The date time are\'',
dag=dag
)

task2 = BashOperator(
task_id = 'print_date',
bash_command = 'date',
dag=dag)
```

- **Task Pipeline**

`task1 >> task2`
print_hello runs and then print_date

Consider this DAG:

```

# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago

#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG

# define the DAG
dag = DAG(
    'my-first-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)

# define the tasks

# define the first task

extract = BashOperator(
    task_id='extract',
    bash_command='cut -d":" -f1,3,6 /etc/passwd >
/home/project/airflow/dags/extracted-data.txt',
    dag=dag,
)

# define the second task
transform_and_load = BashOperator(
    task_id='transform',
    bash_command='tr ":" "," < /home/project/airflow/dags/extracted-data.txt
> /home/project/airflow/dags/transformed-data.csv',
    dag=dag,
)

# task pipeline
extract >> transform_and_load
```

1. submit the DAG

`sudo cp my_first_dag.py $AIRFLOW_HOME/dags`

2. verify DAG
   `airflow dags list`
3. verify specific DAG
   `airflow dags list | grep "my-first-dag"`
4. list all tasts
   `airflow tasks list my-first-dag`

Log file location

`logs/dag_id/task_id/execution_date/try_number.log`

Monitoring metrics

- Counters: metrics that always increase
    - Total count of task instances failures
    - Total count of task instances successes
- Gauges: Metrics that may fluctuate
    - Number of running tasks
    - DAG bag size, or number of DAGs in production
- Timers: Metrics related to time duration
    - Milliseconds to finish a task
    - Milliseconds to reach a state

Airflow metrics --- collect-> StatsD --send--> prometheus --aggregate and visualize --> dashboard

---

Summary and Highlights

- Apache Airflow is scalable, dynamic, extensible, and lean
- The five main features of Apache Airflow are pure Python, useful UI, integration, easy to use, and open source
- A common use case is that Apache Airflow defines and organizes machine learning pipeline dependencies
- Tasks are created with Airflow operators
- Pipelines are specified as dependencies between tasks
- Pipeline DAGs defined as code are more maintainable, testable, and collaborative
- Apache Airflow has a rich UI that simplifies working with data pipelines
- You can visualize your DAG in graph or tree mode
- Key components of a DAG definition file include DAG arguments, DAG and task definitions, and the task pipeline
- The 'schedule_interval' parameter specifies how often to re-run your DAG
- You can save Airflow logs into local file systems and send them to cloud storage, search engines, and log analyzers
- Airflow recommends sending production deployment logs to be analyzed by Elasticsearch or Splunk
- With Airflow's UI, you can view DAGs and task events
- The three types of Airflow metrics are counters, gauges, and timers

- Airflow recommends that production deployment metrics be sent to and analyzed by Prometheus via StatsD