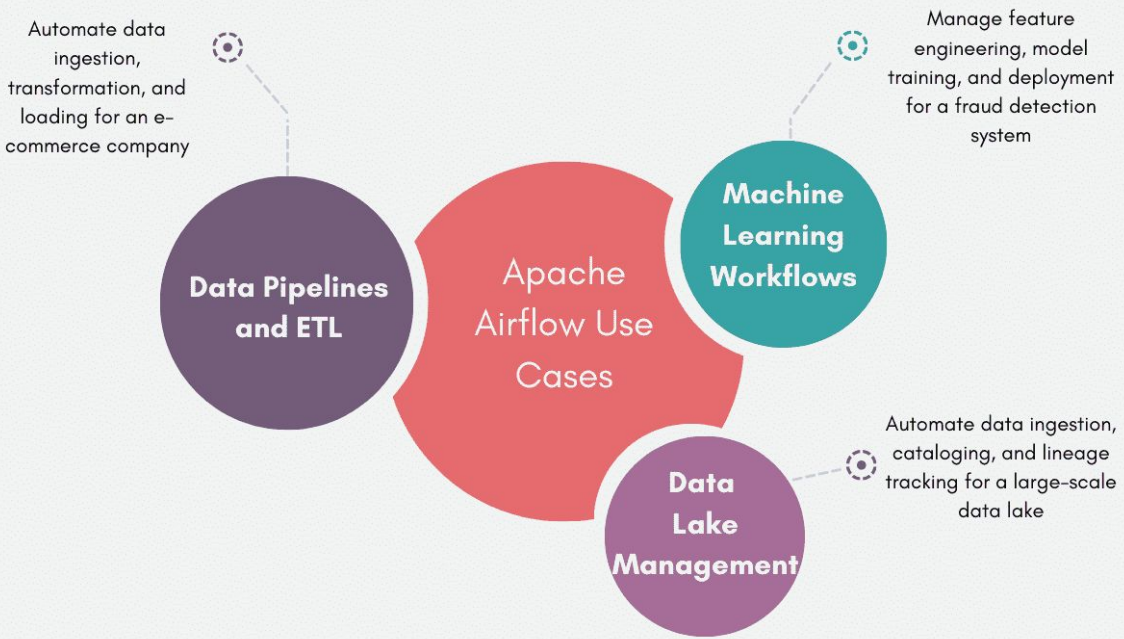
Apache Airflow

Shani Cohen

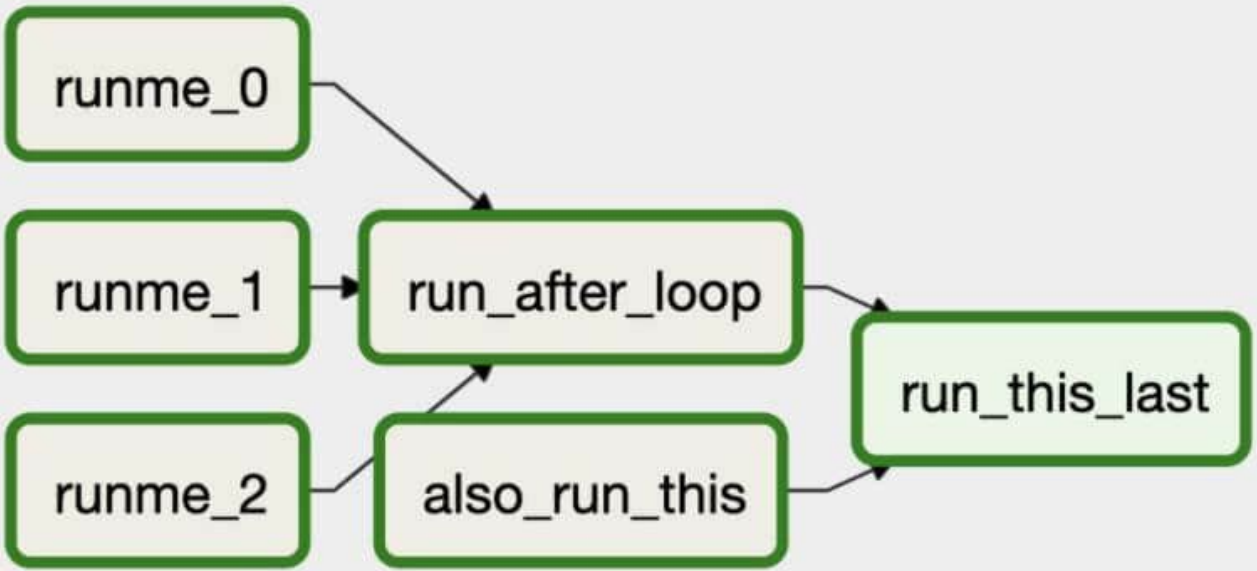
What is Apache Airflow?

Apache Airflow Use Cases

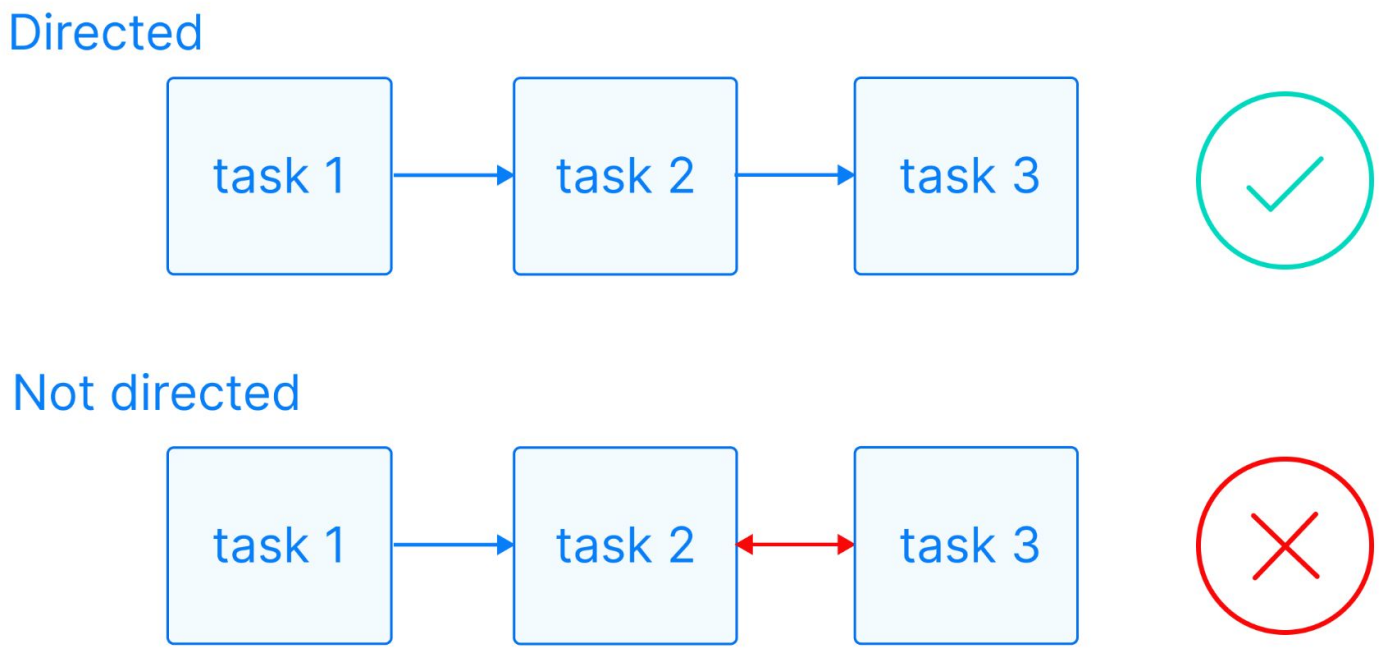


Integrations

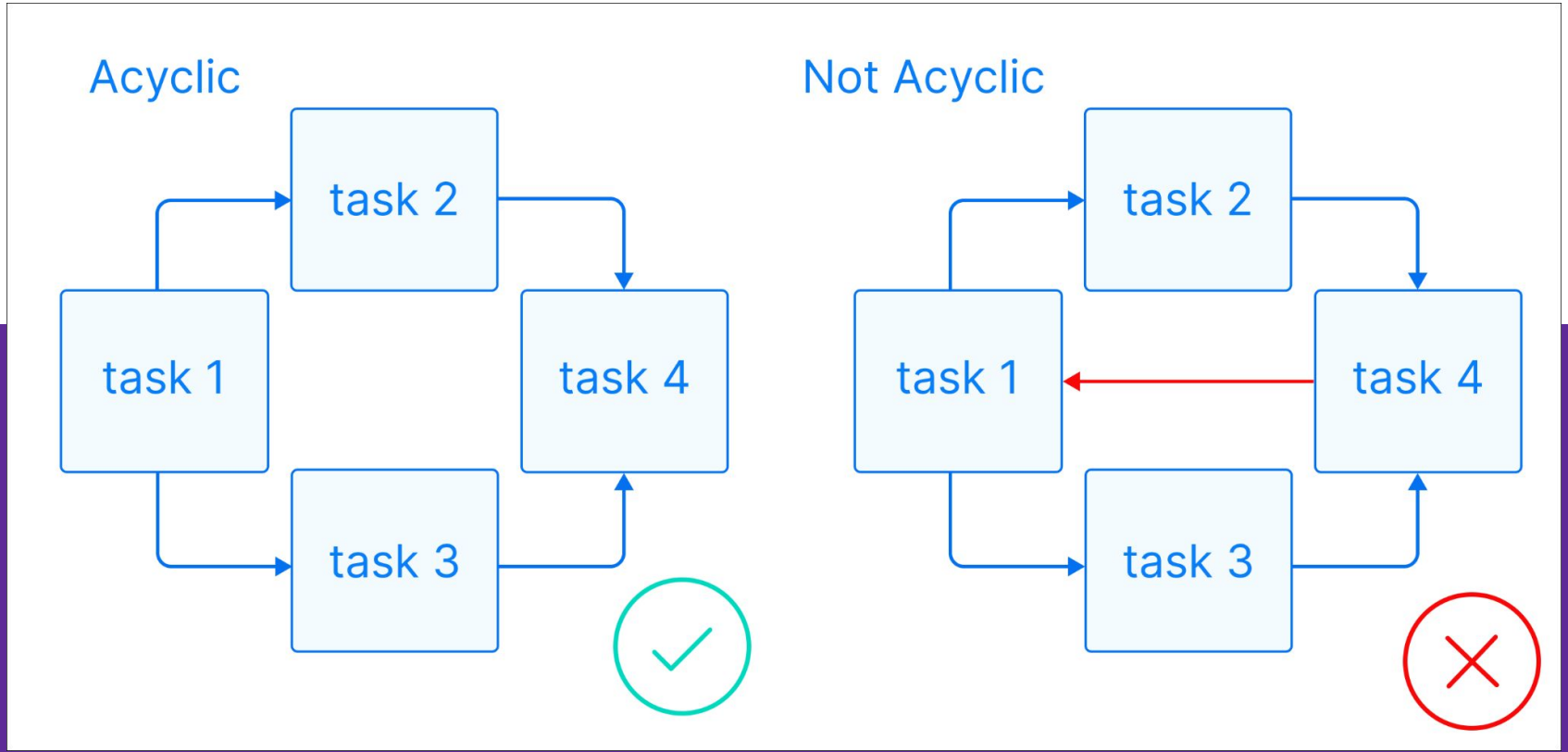
DAG


runme_0
runme_1
runme_2
run_after_loop
also_run_this
run_this_last

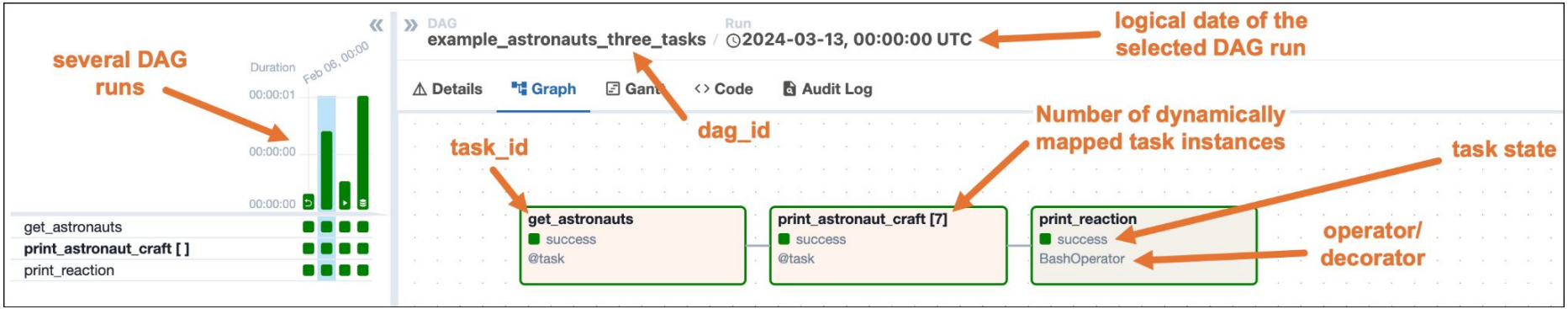
DAG

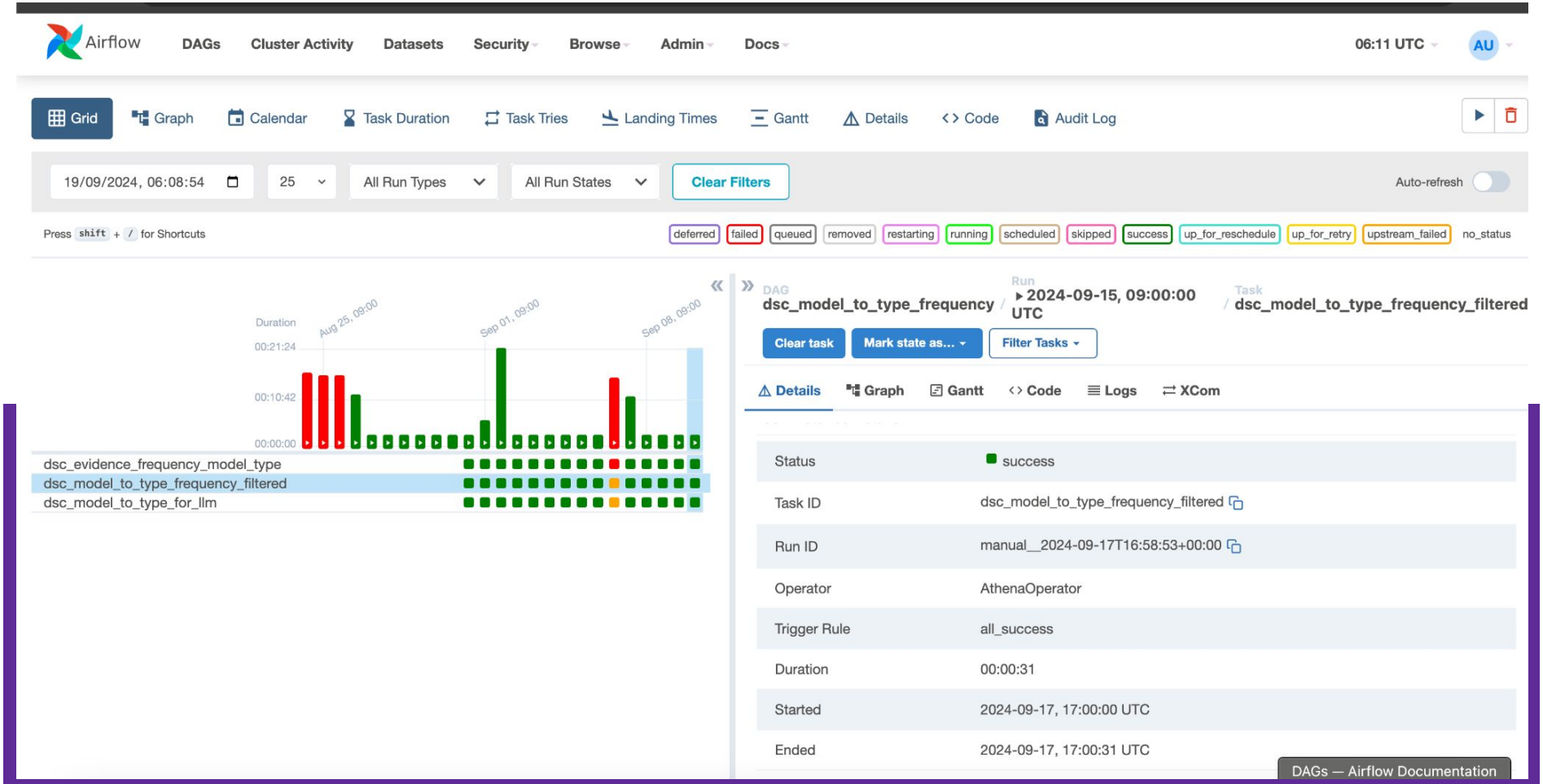


DAG


Acyclic
task 2
task 1
task 4
task 3
Not Acyclic
task 2
task 1
task 4
task 3

DAG run properties




Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
06:11 UTC
AU
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
19/09/2024, 06:08:54
25
All Run Types
All Run States
Clear Filters
Auto-refresh
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:21:24
00:10:42
00:00:00
Aug 25, 09:00
Sep 01, 09:00
Sep 08, 09:00
dsc_evidence_frequency_model_type
dsc_model_to_type_frequency_filtered
dsc_model_to_type_for_llm
DAG
dsc_model_to_type_frequency
Run
2024-09-15, 09:00:00 UTC
Task
dsc_model_to_type_frequency_filtered
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Logs
XCom
Status
success
Task ID
dsc_model_to_type_frequency_filtered
Run ID
manual__2024-09-17T16:58:53+00:00
Operator
AthenaOperator
Trigger Rule
all_success
Duration
00:00:31
Started
2024-09-17, 17:00:00 UTC
Ended
2024-09-17, 17:00:31 UTC
DAGs — Airflow Documentation

DAG Level Parameters

```
@dag(
    start_date=datetime(2024, 1, 1),
    schedule="@daily",
    catchup=False,
)
def taskflow_dag():
```

- dag_id
- start_date
- schedule
- catchup

Workflow as code

```python
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()
```

Task

```
t1 = BashOperator(
    task_id="print_date",
    bash_command="date",
)

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

Task Dependencies

```
t1.set_downstream(t2)

# This means that t2 will depend on t1
# running successfully to run.
# It is equivalent to:
t2.set_upstream(t1)

# The bit shift operator can also be
# used to chain operations:
t1 >> t2

# And the upstream dependency with the
# bit shift operator:
t2 << t1

# Chaining multiple dependencies becomes
# concise with the bit shift operator:
t1 >> t2 >> t3

# A list of tasks can also be set as
# dependencies. These operations
# all have the same effect:
t1.set_downstream([t2, t3])
t1 >> [t2, t3]
[t2, t3] << t1
```

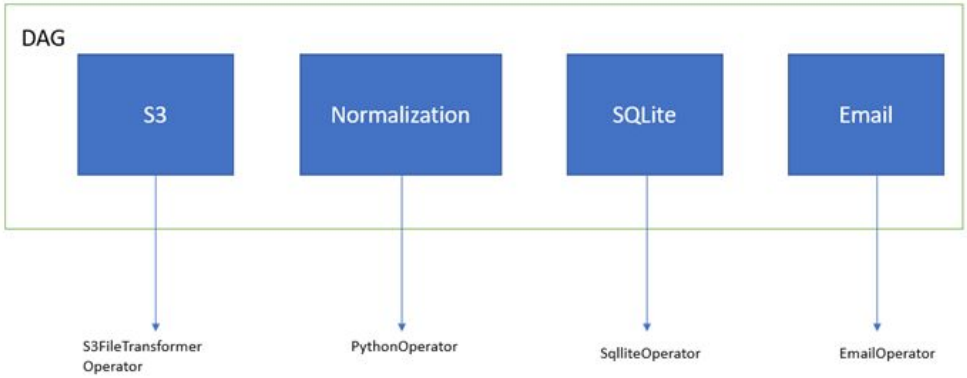
Taskflow

```python
from airflow.decorators import task
from airflow.operators.email import EmailOperator

@task
def get_ip():
    return my_ip_service.get_main_ip()

@task(multiple_outputs=True)
def compose_email(external_ip):
    return {
        'subject':f'Server connected from {external_ip}',
        'body': f'Your server executing Airflow is connect
    }

email_info = compose_email(get_ip())
```

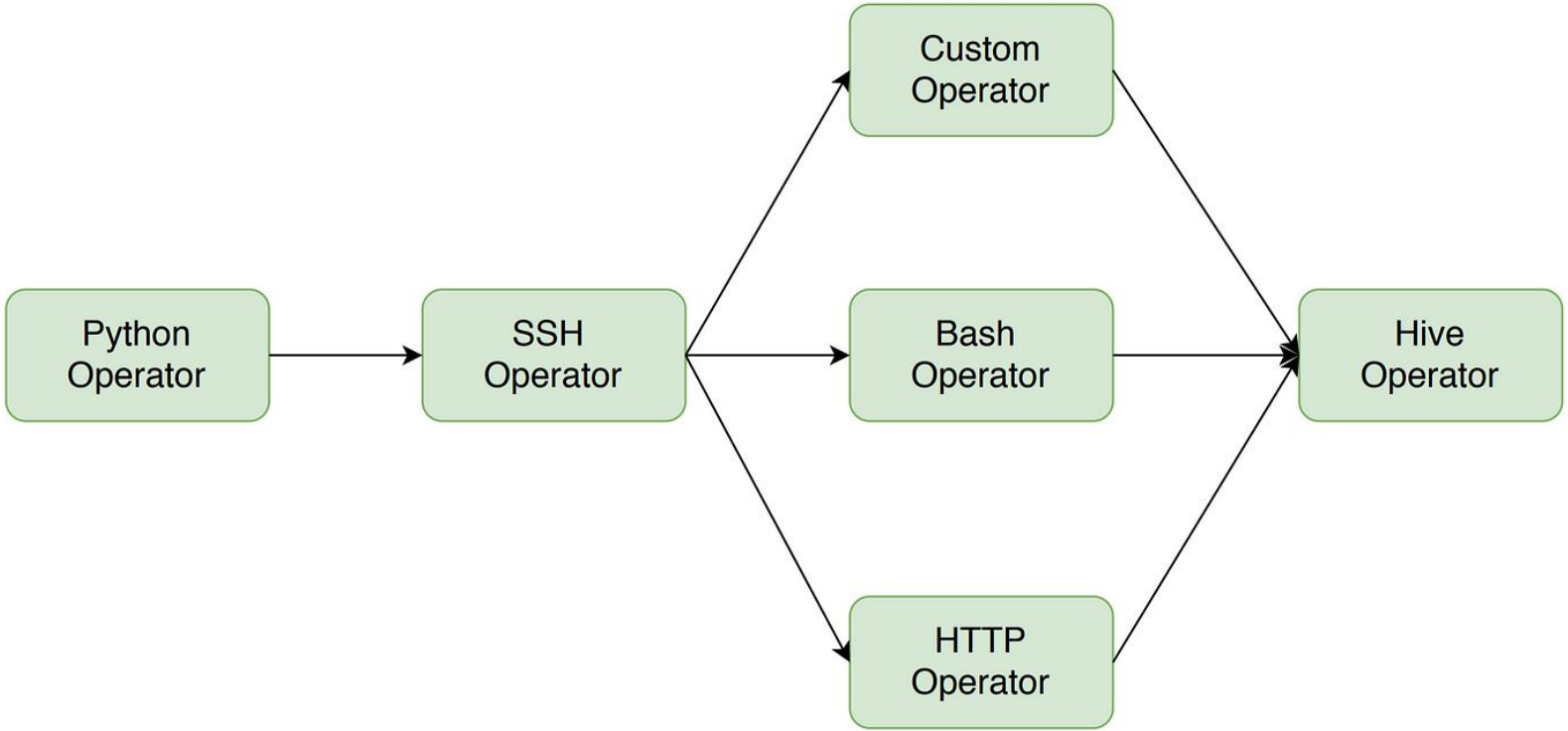
Task Dependencies - Taskflow

```
    order_data = extract()
    order_summary = transform(order_data)
    load(order_summary["total_order_value"])
```

operators


DAG
S3
Normalization
SQLite
Email
S3FileTransformer Operator
PythonOperator
SqlliteOperator
EmailOperator

Execution Example

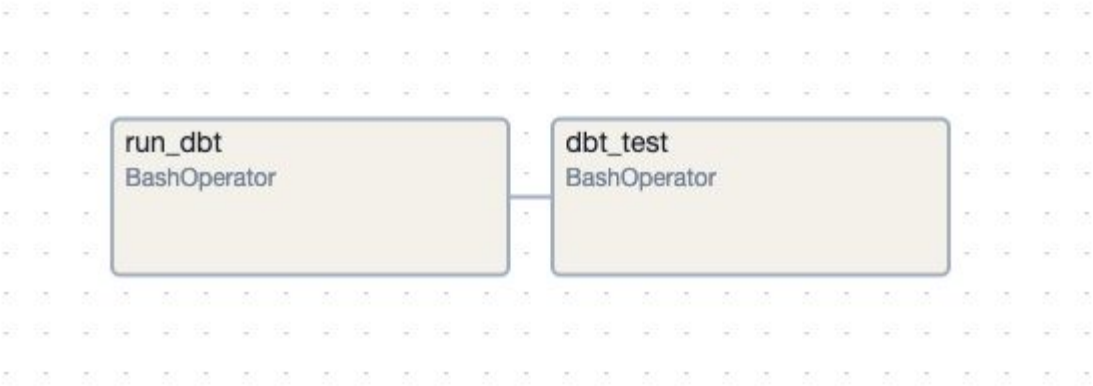


Workshop Resources:

https://github.com/shanicohen1902/dbt-cost-allocation-workshop

Exercise - Add dbt Test Execution to Your DAG

Modify ‘run_dbt’ DAG to add another task that executes `dbt test`.



Jinja

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

t3 = BashOperator(
    task_id="templated",
    depends_on_past=False,
    bash_command=templated_command,
)
```

Airflow Templates

```
BashOperator(
    task_id="print_day_of_week",
    bash_command="echo Today is {{ execution_date.format('dddd') }}",
)
```

Out Of The Box Templats

Variables


List Variable
Search
Actions
Key
Val
Description
Is Encrypted
env
dev
False

Variables

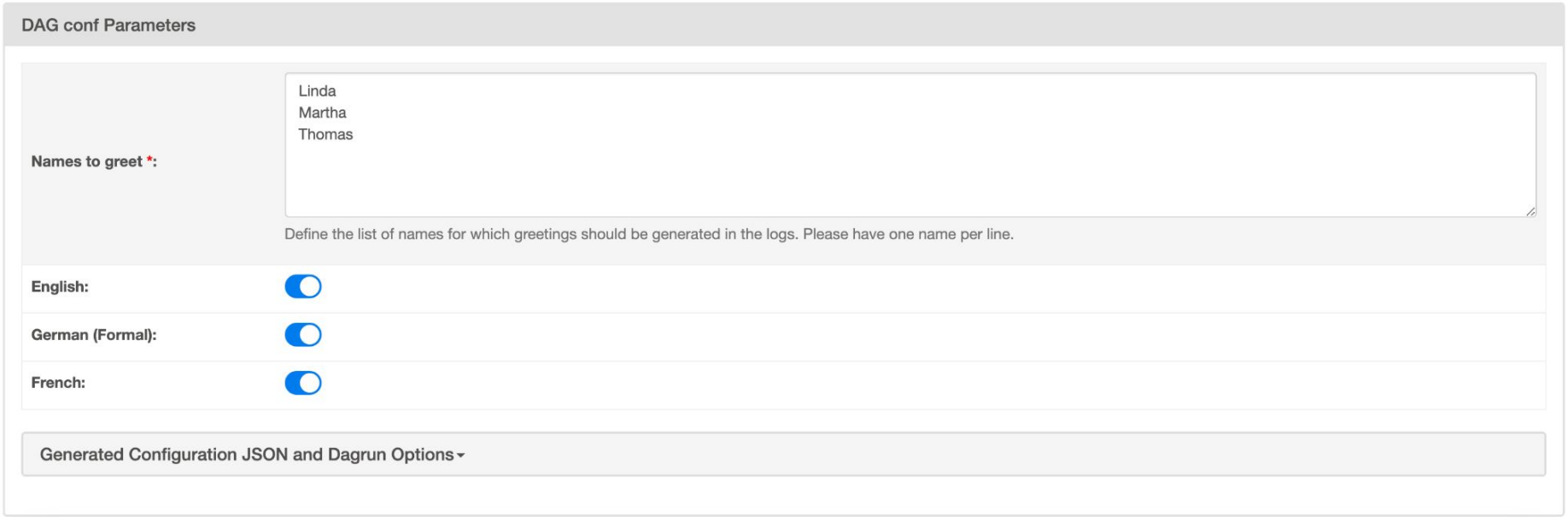
```python
from airflow.models import Variable

# Normal call style
foo = Variable.get("foo")

# Auto-deserializes a JSON value
bar = Variable.get("bar", deserialize_json=True)

# Returns the value of default_var (None) if the variable is not set
baz = Variable.get("baz", default_var=None)
```

Variables

```
# Raw value
echo {{ var.value.<variable_name> }}

# Auto-deserialize JSON value
echo {{ var.json.<variable_name> }}
```

Connections

Params


DAG conf Parameters
Names to greet *:
Linda
Martha
Thomas
Define the list of names for which greetings should be generated in the logs. Please have one name per line.
English:
German (Formal):
French:
Generated Configuration JSON and Dagrun Options

Params

```python
from pendulum import datetime
from airflow.decorators import dag, task


@dag(
    start_date=datetime(2023, 6, 1),
    schedule=None,
    catchup=False,
    params={"upstream_color": "Manual run, no upstream color available."},
)
def tdro_example_downstream():
    @task
    def print_color(**context):
        print(context["params"]["upstream_color"])

    print_color()


tdro_example_downstream()
```

Params

Params are also accessible as a Jinja template using the `{{ params.my_param }}` syntax.

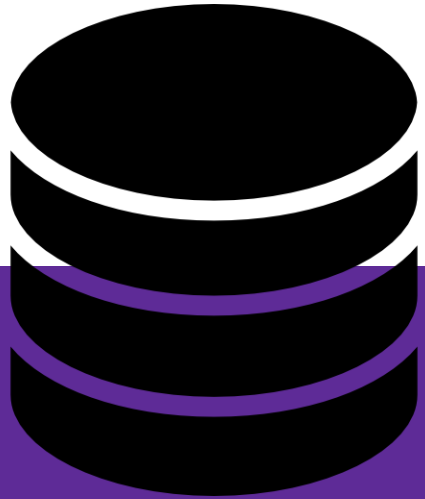
```
from airflow.utils.template import literal

BashOperator(
    task_id="use_literal_wrapper_to_ignore_jinja_template",
    bash_command=literal("echo {{ params.the_best_number }}"),
)
```

Params

Params are also accessible as a Jinja template using the `{{ params.my_param }}` syntax.

```
from airflow.utils.template import literal

BashOperator(
    task_id="use_literal_wrapper_to_ignore_jinja_template",
    bash_command=literal("echo {{ params.the_best_number }}"),
)
```

Xcom



Xcom

```
# pushes data in any_serializable_value into xcom with key "identifier as string"
task_instance.xcom_push(key="identifier as a string", value=any_serializable_value)
```

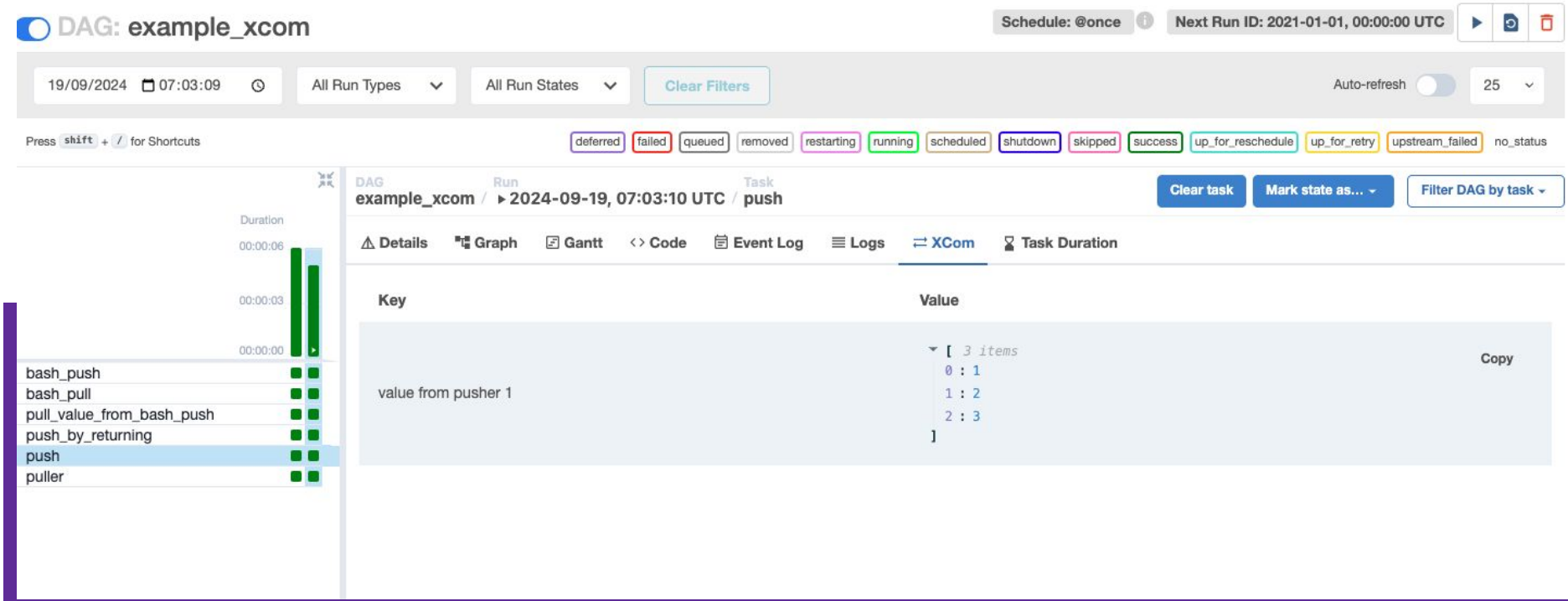
```
# pulls the xcom variable with key "identifier as string" that was pushed from within task-1
task_instance.xcom_pull(key="identifier as string", task_ids="task-1")
```

Xcom

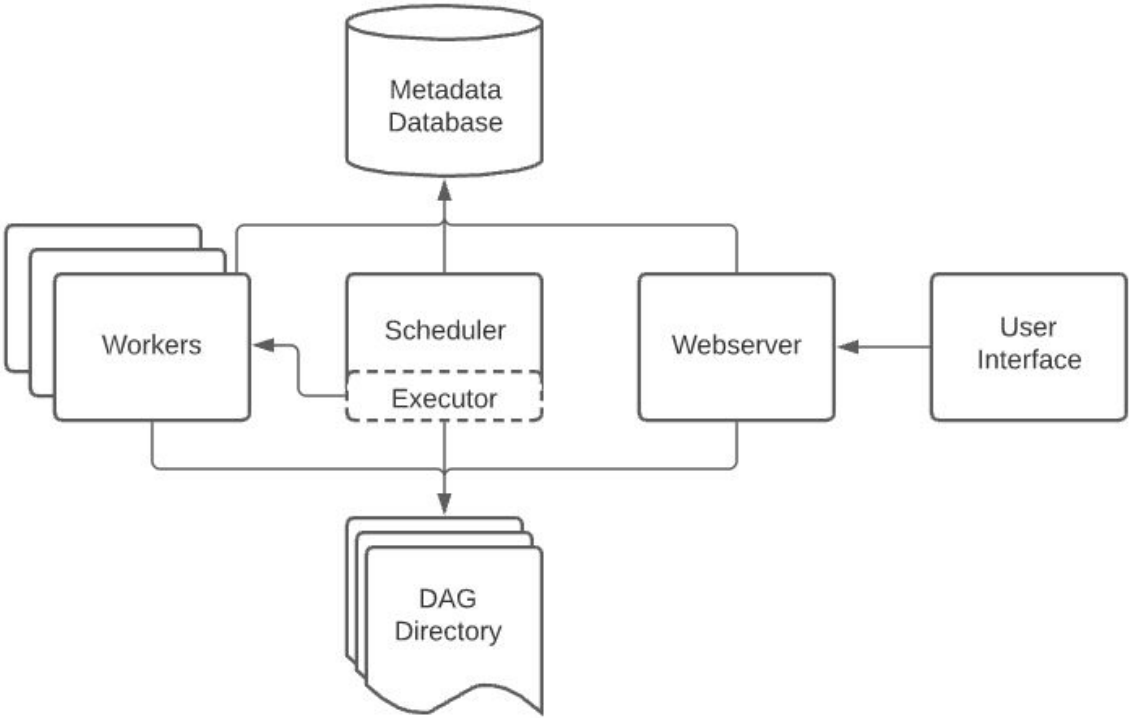
```
# Pulls the return_value XCOM from "pushing_task"
value = task_instance.xcom_pull(task_ids='pushing_task')
```

Xcom

```
SELECT * FROM {{ task_instance.xcom_pull(task_ids='foo', key='table_name') }}
```


DAG: example_xcom
Schedule: @once
Next Run ID: 2021-01-01, 00:00:00 UTC
19/09/2024 07:03:09
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:00:06
00:00:03
00:00:00
bash_push
bash_pull
pull_value_from_bash_push
push_by_returning
push
puller
DAG
example_xcom
Run
2024-09-19, 07:03:10 UTC
Task
push
Clear task
Mark state as...
Filter DAG by task
Details
Graph
Gantt
Code
Event Log
Logs
XCom
Task Duration
Key
Value
value from pusher 1
[3 items
0 : 1
1 : 2
2 : 3
]
Copy

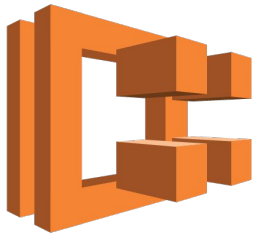
Airflow Architecture



Executors

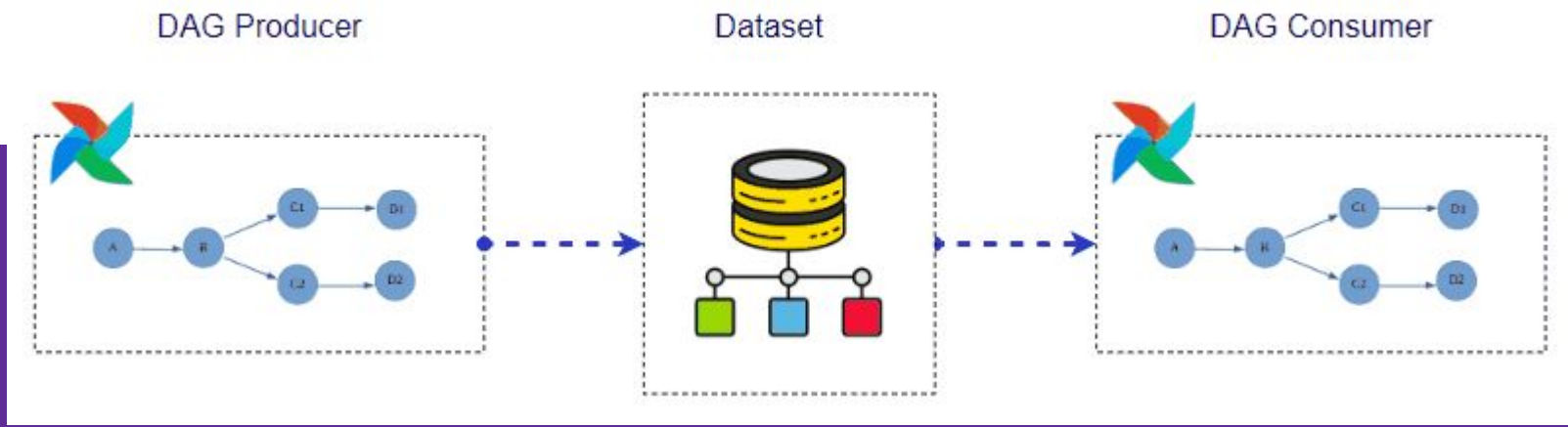
CELERY

kubernetes



AWS ECS

Data-aware scheduling



Data-aware scheduling

```python
from airflow.datasets import Dataset

with DAG(...):
    MyOperator(
        # this task updates example.csv
        outlets=[Dataset("s3://dataset-bucket/example.csv")],
        ...,
    )


with DAG(
    # this DAG should be run when example.csv is updated (by dag1)
    schedule=[Dataset("s3://dataset-bucket/example.csv")],
    ...,
):
    ...
```

Exercise - Xcom + Data-aware scheduling

1. Change ‘echo_airflow_home’ task to return airflow home string to xcom
2. Change ‘run_dbt’ task to take airflow_home sting string from xcom
3. Split run_dbt into two separate dags - run_dbt and run_test. Trigger run_test every time that run_dbt successfully executed

QUIZZZZZZZ!

What does DAG stand for in Apache Airflow?

A) Data Access Group

B) Directed Acyclic Graph

C) Data Aggregation Graph

D) Dynamic Action Group

Which component is responsible for executing tasks in Airflow?

A) Scheduler

B) Executor

C) Web Server

D) Metadata Database

How can tasks communicate with each other in Airflow?

A) Using global variables

B) Through direct function calls

C) By leveraging XComs

D) Data cannot be passed between tasks

What is the default execution model for Apache Airflow?

A) LocalExecutor

B) CeleryExecutor

C) KubernetesExecutor

D) SequentialExecutor

Which operator would you use to execute a Bash command in a task?

A) PythonOperator

B) BashOperator

C) DummyOperator

D) BranchPythonOperator

What parameter in the DAG definition controls whether past DAG runs should be executed?

A) catchup

B) retries

C) start_date

D) schedule_interval

___