Apache Airflow vs. Flyte — Comparison Cheat Sheet

Homepage | Documentation | Github | Slack | Twitter

Installation

Airflow

```
• export AIRFLOW_HOME=~/airflow
• AIRFLOW_VERSION=2.4.3
• PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"
• CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
• pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
• airflow standalone

With Astro:
• Docker
• Astro CLI
• astro dev init
```

Flyte

```
• Docker
• pip install flytekit
• brew install flyteorg/homebrew-tap/flytectl (OR) curl -L https://raw.githubusercontent.com/flyteorg/flytectl/HEAD/install.sh | bash
```

Task & DAG

Airflow

```python
from datetime import datetime, timedelta

from airflow import DAG
from airflow.operators.python_operator import PythonOperator


def concat_str(x):
    return x + " from Airflow"


def get_length(ti):
    return len(ti.xcom_pull(task_ids="t1"))


with DAG(
    "dag_example",
    description="...",
    schedule_interval=timedelta(days=1),
    start_date=datetime(2022, 1, 1),
) as dag:
    t1 = PythonOperator(
        task_id="t1",
        python_callable=concat_str,
        op_kwargs={"x": "Hola"},
    )
    t2 = PythonOperator(
        task_id="t2",
        python_callable=get_length,
        op_kwargs={},
    )
    t1 >> t2
```

Flyte

```python
from datetime import datetime

from flytekit import CronSchedule, LaunchPlan, task, workflow


@task
def concat_str(x: str) -> str:
    return x + " from Flyte"


@task
def get_length(x: str) -> int:
    return len(x)

@workflow
def wf(kickoff_time: datetime) -> int:
    return get_length(x=concat_str(x="Hola"))


LaunchPlan.get_or_create(
    name="my_cron",
    workflow=wf,
    schedule=CronSchedule(schedule="0 1 * * *",
    kickoff_time_input_arg="kickoff_time"),
)
```

Run code in bash

Airflow

```python
from datetime import datetime

from airflow import DAG
from airflow.operators.bash import BashOperator

with DAG(
    dag_id="example_bash_operator",
    schedule_interval=None,
    start_date=datetime(2022, 1, 1),
) as dag:
    BashOperator(task_id="bash_op", bash_command="echo foo")
```

Flyte

```python
from flytekit import workflow
from flytekit.extras.tasks.shell import ShellTask

shell_task = ShellTask(name="bash_op", script="echo foo")


@workflow
def example_bash_operator():
    shell_task()
```

Dynamic DAG

Airflow

```python
from datetime import datetime

from airflow import DAG
from airflow.operators.python_operator import PythonOperator


def hello_world_py(name):
    return f"{name} Hello, World!"


with DAG(
    "example_dynamic_dag",
    description="...",
    schedule_interval=None,
    start_date=datetime(2022, 1, 1),
) as dag:
    for task in ["task1", "task2"]:
        PythonOperator(
            task_id=task, python_callable=hello_world_py,
            op_kwargs={"name": task}
        )
```

Flyte

```python
from typing import List

from flytekit import dynamic, task, workflow


@task
def hello_world(index: str) -> str:
    return f"{index} Hello, World!"


@dynamic
def loop_hello_world(n: List[str]):
    for item in n:
        hello_world(index=item)


@workflow
def wf():
    loop_hello_world(n=["task1", "task2"])
```

Map Task

Airflow

```python
from datetime import datetime

from airflow import DAG
from airflow.decorators import task

with DAG(dag_id="simple_mapping", start_date=datetime(2022, 1, 1)) as
dag:

    @task
    def make_list():
        return [1, 2, 3]

    @task
    def add_one(x: int):
        return x + 1

    @task
    def sum_it(values):
        print(f"Total was {sum(values)}")

    added_values = add_one.expand(x=make_list())
    sum_it(added_values)
```

Flyte

```python
from typing import List

from flytekit import map_task, task, workflow


@task
def add_one(x: int) -> int:
    return x + 1


@task
def sum_it(values: List[int]):
    print(f"Total was {sum(values)}")


@workflow
def wf(x: List[int]):
    mapped_out = map_task(add_one)(x=x)
    sum_it(values=mapped_out)
```

Cross-DAG

Airflow

```python
from datetime import datetime

from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python import PythonOperator
from airflow.operators.trigger_dagrun import TriggerDagRunOperator

with DAG("dependent-dag", start_time=datetime(2021, 1, 1)) as dag:

    def greeting():
        print("Hello!")

    hello_python = python_operator.PythonOperator(task_id="hello",
python_callable=greeting)

    dummpy_op = DummyOperator(task_id="dummy_op")

    hello_python >> dummpy_op


def print_task_type(**kwargs):
    """
    Example function to call before and after dependent DAG.
    """
    print(f"The {kwargs['task_type']} task has completed.")

with DAG(
    "trigger-dagrun-dag",
    start_date=datetime(2021, 1, 1),
) as dag:
    start_task = PythonOperator(
        task_id="starting_task",
        python_callable=print_task_type,
        op_kwargs={"task_type": "starting"},
    )

    trigger_dependent_dag = TriggerDagRunOperator(
        task_id="trigger_dependent_dag",
        trigger_dag_id="dependent-dag",
        wait_for_completion=True,
    )

    end_task = PythonOperator(
        task_id="end_task",
        python_callable=print_task_type,
        op_kwargs={"task_type": "ending"},
    )

    start_task >> trigger_dependent_dag >> end_task
```

Flyte

```python
from typing import Dict

from flytekit import task, workflow


@task
def greeting() -> str:
    return "Hello"


@task
def dummy() -> str:
    return "Dummy task"


@workflow
def dependent_dag():
    task_1 = greeting()
    task_2 = dummy()
    task_1 >> task_2


@task
def print_task_type(kwargs: Dict[str, str]):
    print(f"The {kwargs['task_type']} task has completed.")


@workflow
def trigger_dagrun():
    task_1 = print_task_type(kwargs={"task_type": "starting"})
    sub_wf = dependent_dag()
    task_2 = print_task_type(kwargs={"task_type": "ending"})
    task_1 >> sub_wf
    sub_wf >> task_2
```

Branching

Airflow

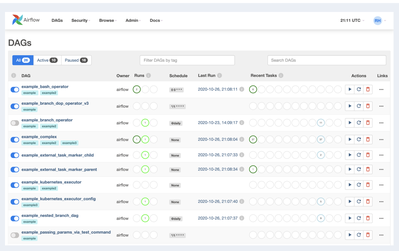
```python
from datetime import datetime

from airflow import DAG
from airflow.models.baseoperator import chain
from airflow.operators.dummy import DummyOperator
from airflow.operators.python import ShortCircuitOperator

with DAG(
    dag_id="example_short_circuit_operator",
    start_date=datetime(2021, 1, 1),
) as dag:
    cond_true = ShortCircuitOperator(
        task_id="condition_is_True",
        python_callable=lambda: True,
    )

    cond_false = ShortCircuitOperator(
        task_id="condition_is_False",
        python_callable=lambda: False,
    )

    ds_true = [DummyOperator(task_id="success_task")]
    ds_false = [DummyOperator(task_id="fail_task")]

    chain(cond_true, *ds_true)
    chain(cond_false, *ds_false)
```

Flyte

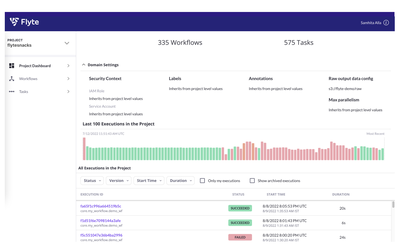
```python
from flytekit import conditional, task, workflow


@task
def condition_is_true() -> bool:
    return True


@task
def condition_is_false() -> bool:
    return False


@task
def success_task() -> str:
    return "success"


@task
def fail_task() -> str:
    return "fail"


@workflow
def wf_1() -> str:
    result = condition_is_true()
    return (
        conditional("example_flyte_conditional_success")
        .if_(result.is_true())
        .then(success_task())
        .else_()
        .fail("you hit else")
    )


@workflow
def wf_2() -> str:
    result = condition_is_false()
    return (
        conditional("example_flyte_conditional_fail")
        .if_(result.is_true())
        .then(fail_task())
        .else_()
        .fail("you hit else")
    )
```

UI

Airflow



Flyte



CLI Comparison

	Airflow	Flyte
Version	airflow version	flytectl version
View DAGs/Workflows	airflow dags list	flytectl get workflow -p flytesnacks -d development
Status of DAG/Execution	airflow dags state <dag_id>	flytectl get execution -p flytesnacks -d development <execution_id>
Trigger DAG/Workflow	airflow dags trigger <dag_id>	flytectl get launchplan --project flytesnacks --domain development flyte.workflows.example.my_wf --latest --execFile exec_spec.yaml flytectl create execution --project flytesnacks --domain development --execFile exec_spec.yaml
Delete DAG	airflow dags delete <dag_id>	flytectl delete execution <execution_id> -p flytesnacks -d development *^ Aborts the execution, but doesn't delete it from the DB.*
List DAG/Execution runs given a DAG ID/Workflow name	airflow dags list-runs -d <dag_id>	flytectl get execution -p flytesnacks -d development --filter.fieldSelector="launch_plan.name=core.control_flow.chain_entities.chain_tasks_wf" --filter.sortBy started_at
View Tasks	airflow tasks list <dag_id>	flytectl get execution -p flytesnacks -d development <workflow_execution_id> --details
Run Task	airflow tasks run <dag_id> <task_id> <execution_date_or_run_id>	flytectl get tasks -d development -p flytesnacks core.control_flow.merge_sort.merge --version latest --execFile exec_spec.yaml flytectl create execution --execFile exec_spec.yaml -p flytesnacks -d development
Status of Task	airflow tasks state <dag_id> <task_id> <execution_date_or_run_id>	flytectl get execution -p flytesnacks -d development <task_execution_id> (OR) flytectl get execution -p flytesnacks -d development <workflow_execution_id> --details
Status of Task in a DAG/ Workflow Execution	airflow tasks states-for-dag-run <dag_id> <execution_date_or_run_id>	flytectl get execution -p flytesnacks -d development <workflow_execution_id> --details
Clear/Re-run Task	airflow tasks clear <dag_id> *^ Clears the task instances and re-triggers them.*	flytectl create execution --project flytesnacks --domain development --execFile exec_spec.yaml *^ Flyte doesn't clear the state of an execution. You can, however, re-run a task/workflow.*

Terminology

Airflow	Flyte
Task & DAG	Task & Workflow
Provider	Integration
Operator	Task
DagRun/TaskInstance	LaunchPlan
Scheduler	Scheduler
Executor + Workers	WorkflowExecutor + NodeExecutor + FlytePropeller
Metadata Database	FlyteAdmin
Web UI	FlyteConsole
TaskGroup + Cross-DAG	SubWorkflow
Dynamic Task Mapping	Dynamic Workflows

*An association between two terminologies doesn't mean they perform the exact set of operations; it essentially means that they are similar to a certain extent.

Learn more about Flyte at flyte.org