Branch and Conditions in DAG

Here will look at an example of branching in Airflow as well as conditionally skipping tasks.

We'll use simple operators to demonstrate this.

`BranchPythonOperator`

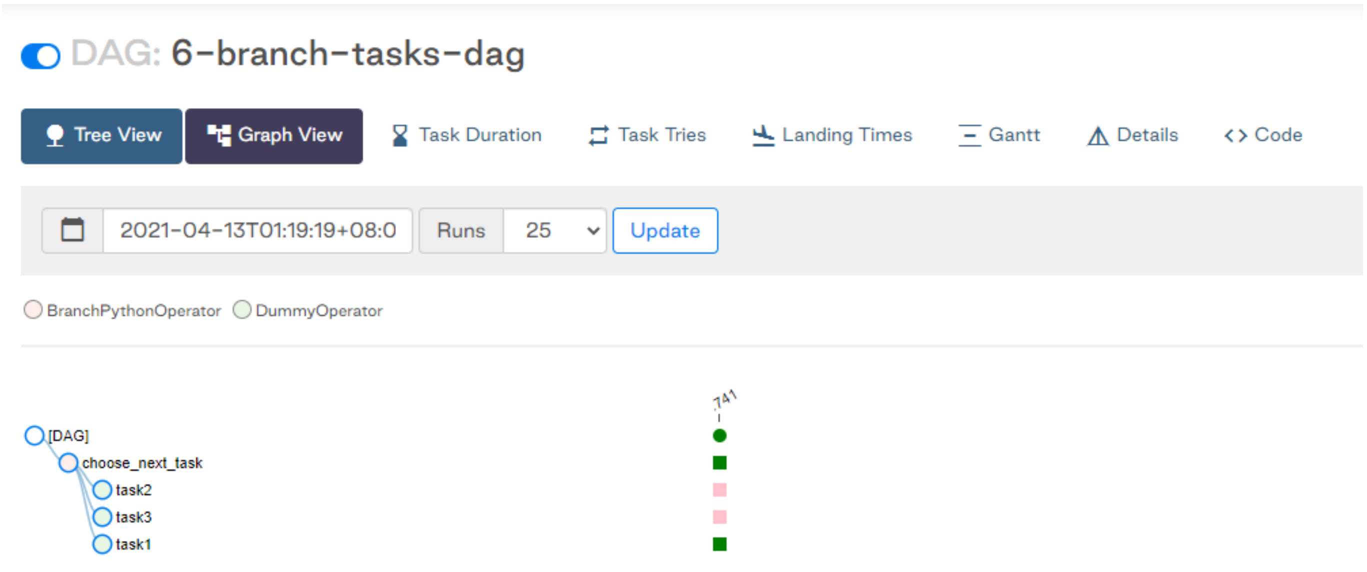
`BranchPythonOperator` is used by passing to it a python function that returns the `task_id` of the next task to execute. Possible tasks to be chosen are those that depend on the `BranchPythonOperator` task.

Example:

```
def _choose_next_task():
    return "task1"

with DAG(
    dag_id="6-branch-tasks-dag",
    start_date=airflow.utils.dates.days_ago(1),
    schedule_interval=None
) as dag:

    choose_next_task = BranchPythonOperator(task_id="choose_next_task",
python_callable=_choose_next_task)

    choice_1 = DummyOperator(task_id="task1")

    choice_2 = DummyOperator(task_id="task2")

    choice_3 = DummyOperator(task_id="task3")

    choose_next_task >> [choice_1, choice_2, choice_3]
```

Here, three tasks depend on the `BranchPythonOperator` and we've hardcoded what task to choose next on the `_choose_next_task` python function, which is `"task_1"`.

Here is what it looks like on the Tree View if we manually trigger this DAG.



Notice how the other tasks not chosen are in pink. That means they are in a "skipped" state.

Converging from branches

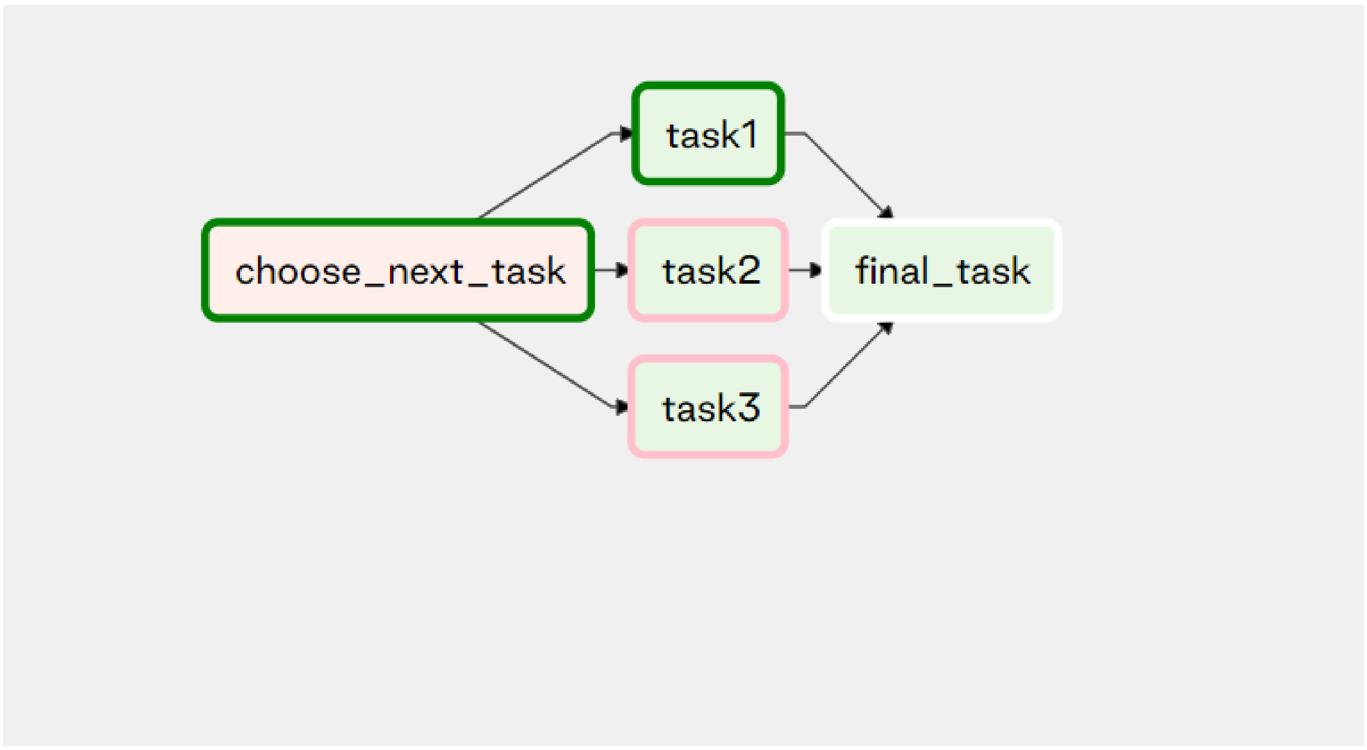
Now what if we want to converge all three choice/branch tasks into one task.

From what we know so far, you may try it out like this:

```
final_task = DummyOperator(task_id="final_task")

choose_next_task >> [choice_1, choice_2, choice_3]

[choice_1, choice_2, choice_3] >> final_task
```

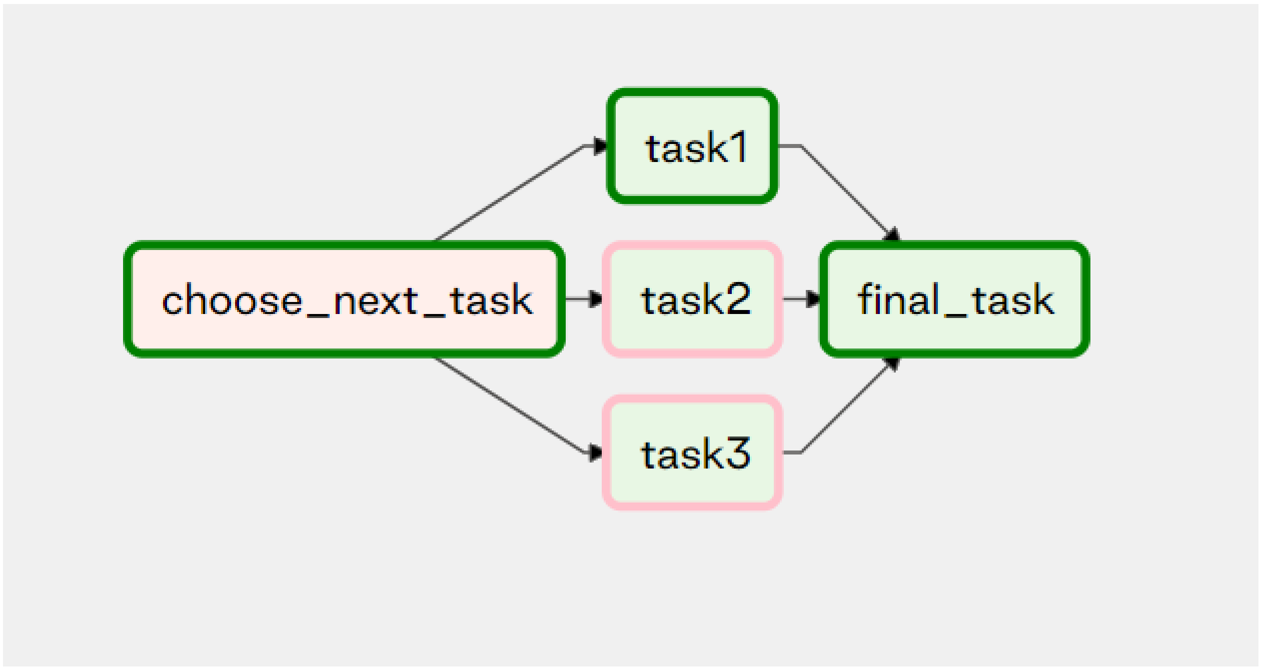
But you will see that the `final_task` won't execute. Why is that?

Well, remember that by default, a task will only execte if all of its dependencies are in a "success" state. A "skipped" state is not a sucess state so it won't execute.



What we can do is change the `"trigger_rule"` of the `final_task` so that it will execute as long as none of its dependencies failed (`"skipped"` state is different from `"failed"` state).

```
final_task = DummyOperator(task_id="final_task", trigger_rule="none_failed")

choose_next_task >> [choice_1, choice_2, choice_3]

[choice_1, choice_2, choice_3] >> final_task
```



Optional Task

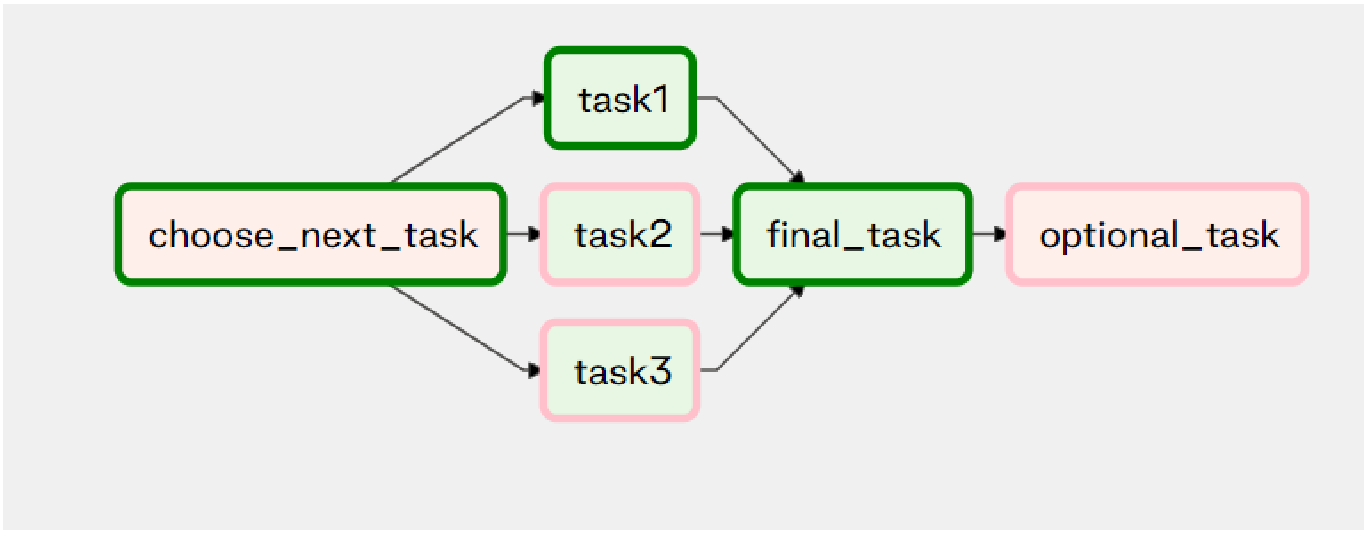
You can also create a task that can be optionally executed or skipped. You can raise the exception `AirflowSkipException` if you want a task to be skipped and not just failed.

Example:

```
def _optional_task():
    if random.choice([0, 1]):
        print("success")
    else:
        raise AirflowSkipException()

...

    optional_task = PythonOperator(task_id="optional_task",
python_callable=_optional_task)

    final_task >> optional_task
```

Here the `optional_task` has a 50% chance of being executed or skipped.

In this run, the `optional_task` was skipped.



Challenge Exercise

You have a normal ML pipeline of fetching from a data source, cleaning the data, training a model and then deploying the model. This DAG is executed on a daily basis. Create a DAG that is scheduled to start 5 days ago from the current date.

Now let's say that 3 days ago, the data source you connect to fetch data from has been migrated to another data source. You need a new set of fetch task and clean task for this data source.

You want to keep the old fetch / clean tasks though so you can backfill old data. Create a branch operator to switch between the two sets of clean and fetch tasks for the old and new data source.

Both sets of clean and fetch task should now converge to a single training model task. After training, there should be a final deploy task that should only be executed on the most recent scheduled interval. (You can use the `LatestOnlyOperator` which skips tasks if not in the latest interval).