DAG Runs

A DAG Run is an object representing an instantiation of the DAG in time.

Each DAG may or may not have a schedule, which informs how DAG Runs are created. `schedule_interval` is defined as a DAG argument, which can be passed a cron expression as a `str`, a `datetime.timedelta` object, or one of of the following cron "presets".

Tip

You can use an online editor for CRON expressions such as Crontab guru

Cron Presets

preset	meaning	cron
None	Don't schedule, use for exclusively "externally triggered" DAGs	
@once	Schedule once and only once	
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@daily	Run once a day at midnight	0 0 * * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@quarterly	Run once a quarter at midnight on the first day	0 0 1 */3 *
@yearly	Run once a year at midnight of January 1	0 0 1 1 *

Your DAG will be instantiated for each schedule along with a corresponding DAG Run entry in the database backend.

Note

If you run a DAG on a schedule_interval of one day, the run stamped 2020-01-01 will be triggered soon after 2020-01-01T23:59. In other words, the job instance is started once the period it covers has ended. The `execution_date` available in the context will also be 2020-01-01.

The first DAG Run is created based on the minimum `start_date` for the tasks in your DAG. Subsequent DAG Runs are created by the scheduler process, based on your DAG's `schedule_interval`, sequentially. If your start_date is 2020-01-01 and schedule_interval is @daily, the first run will be created on 2020-01-02 i.e., after your start date has passed.

Re-run DAG

There can be cases where you will want to execute your DAG again. One such case is when the scheduled DAG run fails.

Catchup

An Airflow DAG with a `start_date`, possibly an `end_date`, and a `schedule_interval` defines a series of intervals which the scheduler turns into individual DAG Runs and executes. The scheduler, by default, will kick off a DAG Run for any interval that has not been run since the last execution date (or has been cleared). This concept is called Catchup.

If your DAG is written to handle its catchup (i.e., not limited to the interval, but instead to `Now` for instance.), then you will want to turn catchup off. This can be done by setting `catchup = False` in DAG or `catchup_by_default = False` in the configuration file. When turned off, the scheduler creates a DAG run only for the latest interval.

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/airflow/blob/master/airflow/example_dags/tutorial.py
"""
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta


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)
}

dag = DAG(
    'tutorial',
    default_args=default_args,
    start_date=datetime(2015, 12, 1),
    description='A simple tutorial DAG',
    schedule_interval='@daily',
    catchup=False)
```

In the example above, if the DAG is picked up by the scheduler daemon on 2016-01-02 at 6 AM, (or from the command line), a single DAG Run will be created, with an *execution_date* of 2016-01-01, and the next one will be created just after midnight on the morning of 2016-01-03 with an execution date of 2016-01-02.

If the `dag.catchup` value had been `True` instead, the scheduler would have created a DAG Run for each completed interval between 2015-12-01 and 2016-01-02 (but not yet one for 2016-01-02, as that interval hasn't completed) and the scheduler will execute them sequentially.

Catchup is also triggered when you turn off a DAG for a specified period and then re-enable it.

This behavior is great for atomic datasets that can easily be split into periods. Turning catchup off is great if your DAG performs catchup internally.

Backfill

There can be the case when you may want to run the dag for a specified historical period e.g., A data filling DAG is created with `start_date` **2019-11-21**, but another user requires the output data from a month ago i.e., **2019-10-21**. This process is known as Backfill.

You may want to backfill the data even in the cases when catchup is disabled. This can be done through CLI. Run the below command

```
airflow dags backfill \
    --start-date START_DATE \
    --end-date END_DATE \
    dag_id
```

The backfill command will re-run all the instances of the dag_id for all the intervals within the start date and end date.

Re-run Tasks

Some of the tasks can fail during the scheduled run. Once you have fixed the errors after going through the logs, you can re-run the tasks by clearing them for the scheduled date. Clearing a task instance doesn't delete the task instance record. Instead, it updates `max_tries` to `0` and sets the current task instance state to `None` , which causes the task to re-run.

Click on the failed task in the Tree or Graph views and then click on **Clear**. The executor will re-run it.

There are multiple options you can select to re-run -

- **Past** - All the instances of the task in the runs before the current DAG's execution date
- **Future** - All the instances of the task in the runs after the current DAG's execution date
- **Upstream** - The upstream tasks in the current DAG
- **Downstream** - The downstream tasks in the current DAG
- **Recursive** - All the tasks in the child DAGs and parent DAGs
- **Failed** - Only the failed tasks in the current DAG

You can also clear the task through CLI using the command:

```
airflow tasks clear dag_id \
    --task-regex task_regex \
    --start-date START_DATE \
    --end-date END_DATE
```

For the specified `dag_id` and time interval, the command clears all instances of the tasks matching the regex. For more options, you can check the help of the clear command :

```
airflow tasks clear --help
```

External Triggers

Note that DAG Runs can also be created manually through the CLI. Just run the command -

```
airflow dags trigger --exec-date execution_date run_id
```

The DAG Runs created externally to the scheduler get associated with the trigger's timestamp and are displayed in the UI alongside scheduled DAG runs. The execution date passed inside the DAG can be specified using the `-e` argument. The default is the current date in the UTC timezone.

In addition, you can also manually trigger a DAG Run using the web UI (tab **DAGs** -> column **Links** -> button **Trigger Dag**)

Passing Parameters when triggering dags

When triggering a DAG from the CLI, the REST API or the UI, it is possible to pass configuration for a DAG Run as a JSON blob.

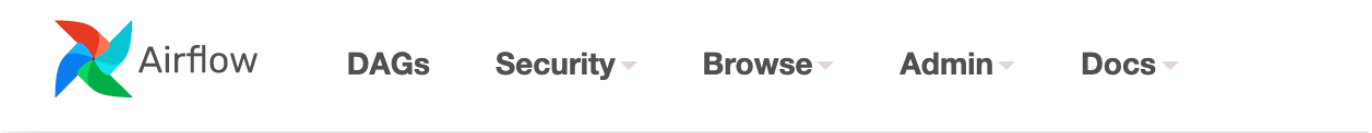
Example of a parameterized DAG:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

dag = DAG("example_parametrized_dag", schedule_interval=None, start_date=days_ago(2))

parameterized_task = BashOperator(
    task_id='parameterized_task',
    bash_command="echo value: {{ dag_run.conf['conf1'] }}",
    dag=dag,
)
```

Note: The parameters from `dag_run.conf` can only be used in a template field of an operator.

Using CLI

```
airflow dags trigger --conf '{"conf1": "value1"}' example_parametrized_dag
```



To Keep in Mind

- Marking task instances as failed can be done through the UI. This can be used to stop running task instances.
- Marking task instances as successful can be done through the UI. This is mostly to fix false negatives, or for instance, when the fix has been applied outside of Airflow.

Previous Next

Was this entry helpful?