# DAG Scheduling `start_date` and `schedule_interval`

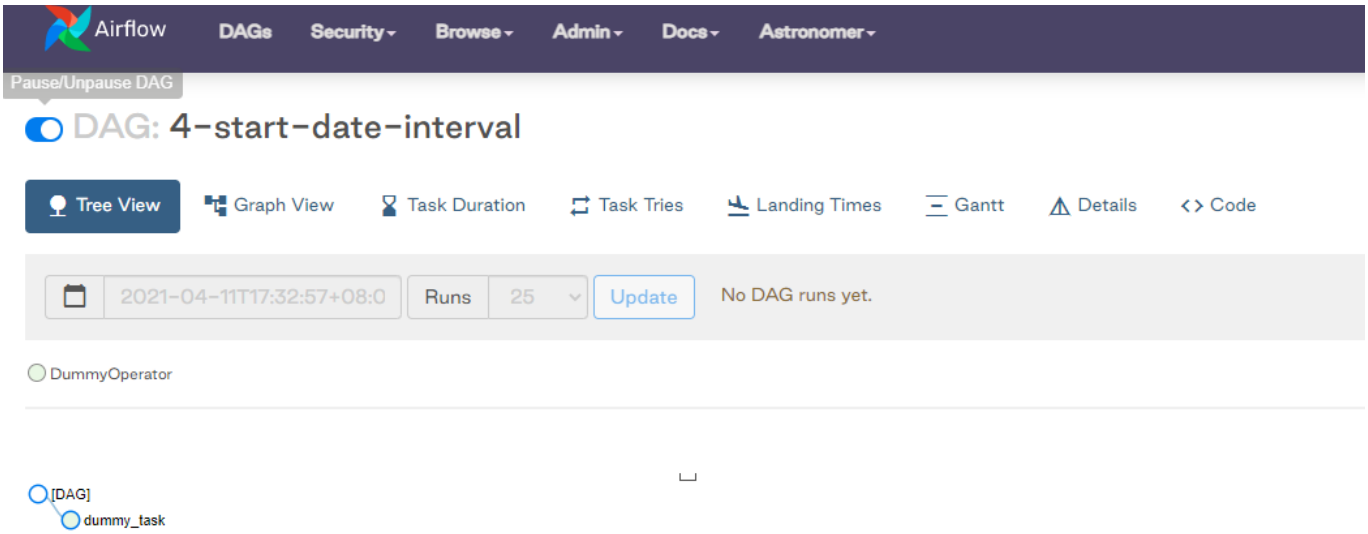
For this exercise, we want to practice the most confusing concept of DAG scheduling in airflow, which is when does the DAG actually run.

# Schedule a DAG for today

Let's start with a simple DAG with one task that is scheduled to start for today. In the context of this example, today is 4/11/2021. Adjust the dates in your code to reflect your current date.

```
local_tz = pendulum.timezone("Asia/Manila")

with DAG(
    dag_id="4-start-date-interval",
    start_date=datetime.datetime(2021, 4, 11, tzinfo=local_tz), # Adjust this to
your current date. You can also specify up to the hour and minute if you want.
    schedule_interval="@daily"
) as dag:

    DummyOperator(task_id="dummy_task")
```

But if you unpause this DAG in the UI it would never run, even when the `start_date` has already passed.




This is because the first execution of a DAG is actually the `start_date` plus the `schedule_interval`. So for the DAG we scheduled for today, the date time the DAG will run will actually be tomorrow (`start_date`(today) + `schedule_interval`(day) = tomorrow).


Modify the DAG `start_date` to the day before today and now it would run. Also, change the DAG id since it is best practice to change the `dag_id` if you change the `start_date` and the `schedule_interval`.

```
local_tz = pendulum.timezone("Asia/Manila")

with DAG(
    dag_id="4-start-date-interval-v2",
    start_date=datetime.datetime(2021, 4, 10, tzinfo=local_tz), # Adjust this to
the day before today in your case
    schedule_interval="@daily"
) as dag:

    DummyOperator(task_id="dummy_task")
```

# Incremental approach of airflow scheduling

Airflow is more oriented to batch/incremental approach to fetching data. It is optimized to fetching data according to your scheduled interval.

It wants to fetch data from your `start_date` to the next date according to your `schedule_interval` so it will want to wait until the full interval is finished before actually running. In our example it wants to wait until 1 day is over from the `start_date` of 4/11/2021 before running. So it would run on 4/12/2021.

When we modified the `start_date` to the day before , 4/10/2021 in our example, one day has already passed and it will now start fetching the data from 4/10/2021 to 4/11/2021.

# Getting the start and end date of your interval.

So now that we know why airflow scheduling works the way it does, how do we know in our code what is the start and end time of the interval we are running in so we can fetch data incrementally.

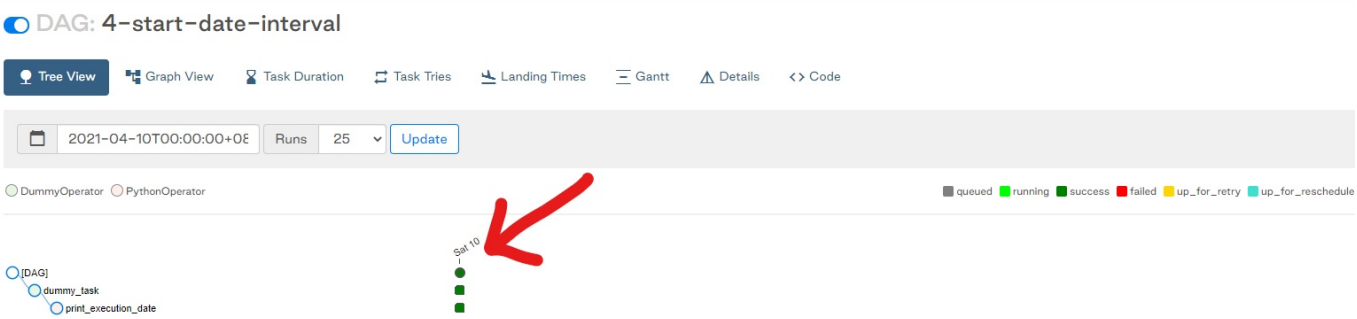
We will use template variables for this. Specifically `execution_date` and `next_execution_date`.

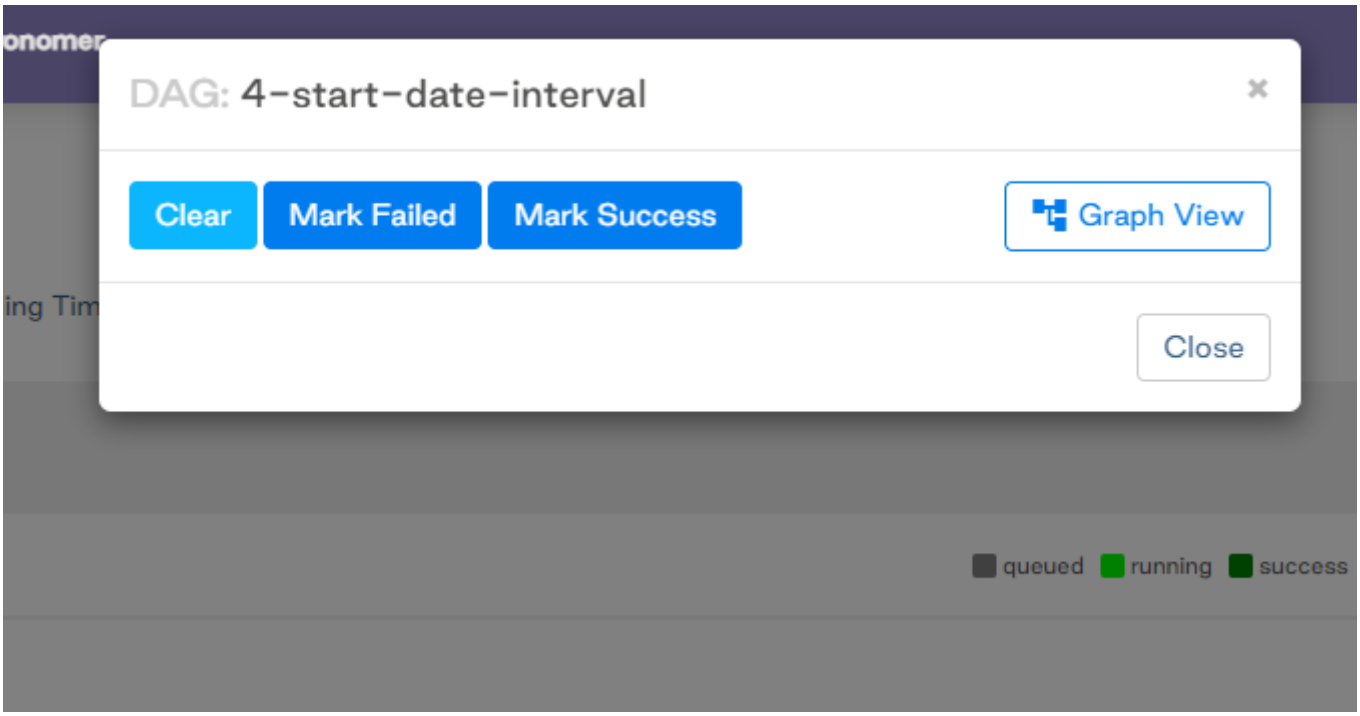
Note: Template variables are given to us in the UTC timezone or what the Airflow configuration timezone is. You need to convert it to local timezone so it will make sense in comparison to our `start_date`.

Create a function and python operator task like the following:

```
local_tz = pendulum.timezone("Asia/Manila")

def _print_dates(execution_date, next_execution_date):
    print(f"execution date: {local_tz.convert(execution_date)}")
    print(f"next execution date: {local_tz.convert(next_execution_date)}")

with DAG(
    dag_id="4-start-date-interval",
    start_date=datetime.datetime(2021, 4, 10, tzinfo=local_tz), # Adjust this to
your current date. You can also specify up to the hour and minute if you want.
    schedule_interval="@daily"
) as dag:

    dummy1 = DummyOperator(task_id="dummy_task")

    print_dates = PythonOperator(task_id="print_execution_date",
python_callable=_print_dates, provide_context=True)

    dummy1 >> print_dates
```

Then to re-run the DAG, you can click on the circle of the only DAG run we have so far in the tree view. Then click `Clear`



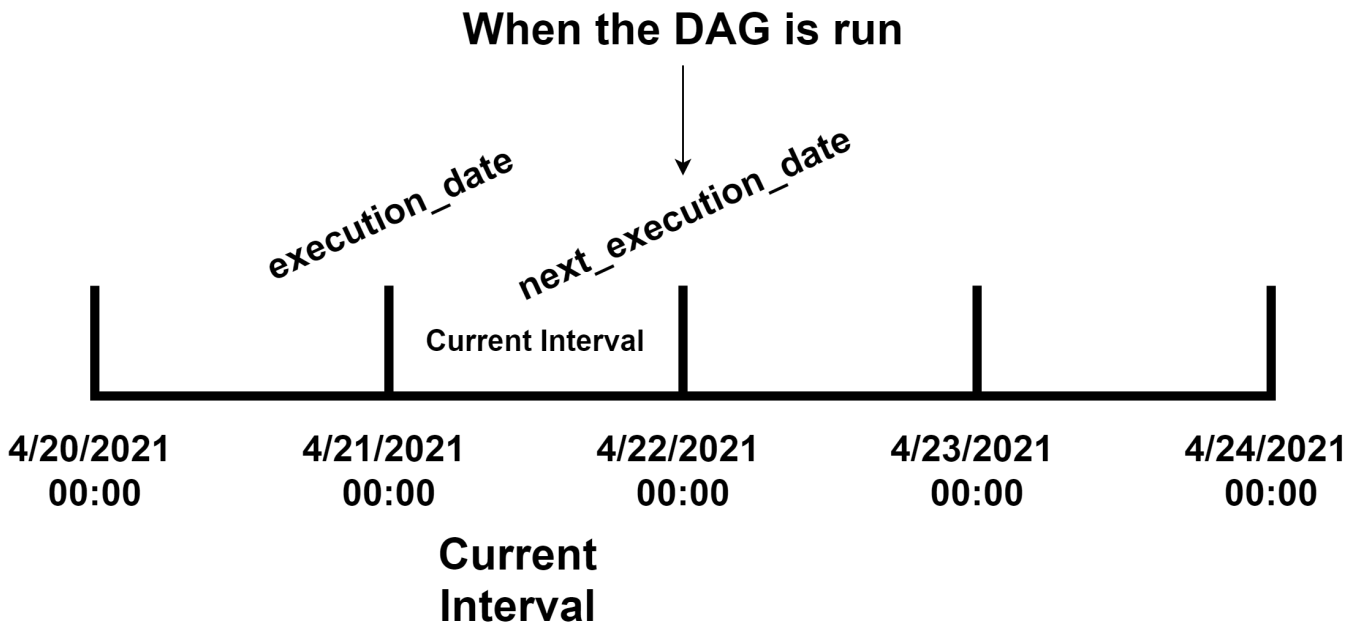

Go to the logs of the `print_dates` task and you will see that the `execution_date` is actually the day before and `next_execution_date` is actually today.

```
[2021-04-11 11:06:48,258] {logging_mixin.py:103} INFO - execution date: 2021-04-10T00:00:00+08:00
[2021-04-11 11:06:48,258] {logging_mixin.py:103} INFO - next execution date: 2021-04-11T00:00:00+08:00
```

In the above example, if today is 4/11, why is the `execution_date` 4/10.

In Airflow, the `execution_date` is actually the start of the scheduled interval and the `next_execution_date` is actually the end of that interval or when the DAG starts to run.

I know it's confusing.




You can use the `execution_date` and `next_execution_date` to pass as date parameters to our API requests or SQL queries.

# Backfilling

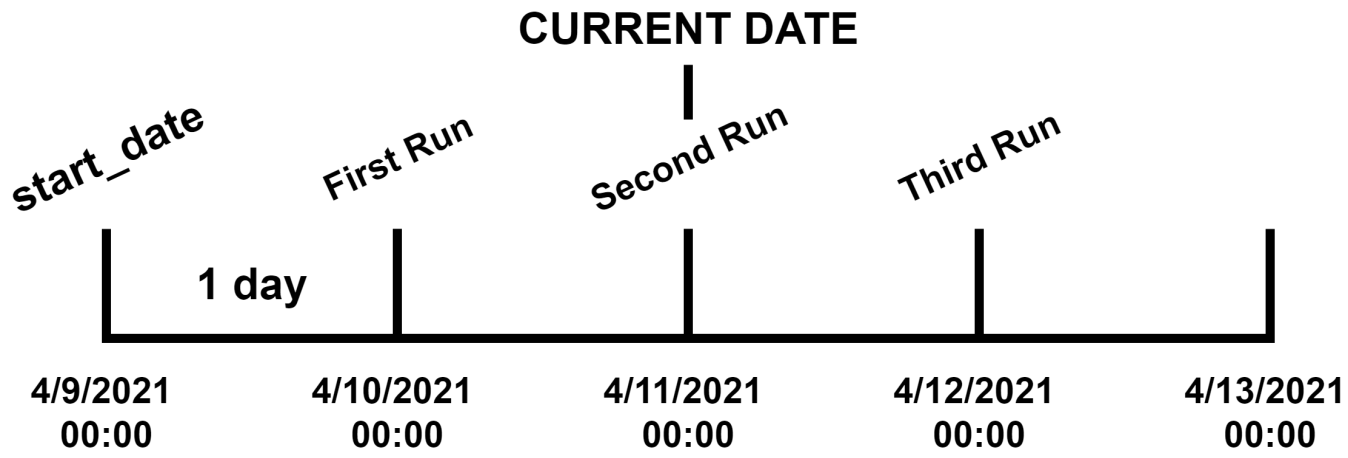
Since we now know that we can schedule DAGs to run during dates in the past, we can schedule it to do past runs and fill in missing data or re-process past data.

Backfilling can also be described as historical data collection.

Change the dag_id and increment the version. Also, change the `start_date` to 2 days before the `current_date` like so:

```
...

with DAG(
    dag_id="4-start-date-interval-v3",
    start_date=datetime.datetime(2021, 4, 9, tzinfo=local_tz), # Adjust this to 2
days before your current date
    schedule_interval="@daily"
) as dag:

...
```

Your `execution_date` and `next_execution_date` will again change to the start and end of each scheduled past interval.




And for each run of your `print_execution_date` tasks, they will have different `execution_date` and `start_date` for each past daily interval.

4/9/2021 - 4/10/2021

```
[2021-04-11 11:48:02,012] {logging_mixin.py:103} INFO - execution date: 2021-04-09T00:00:00+08:00
[2021-04-11 11:48:02,012] {logging_mixin.py:103} INFO - next execution date: 2021-04-10T00:00:00+08:00
```

4/10/2021 - 4/11/2021

```
[2021-04-11 11:48:02,049] {logging_mixin.py:103} INFO - execution date: 2021-04-10T00:00:00+08:00
[2021-04-11 11:48:02,049] {logging_mixin.py:103} INFO - next execution date: 2021-04-11T00:00:00+08:00
```

You can use these again to fetch/re-fetch past data using SQL queries (`WHERE rental_date between execution_date and next_execution_date` ) or API requests.