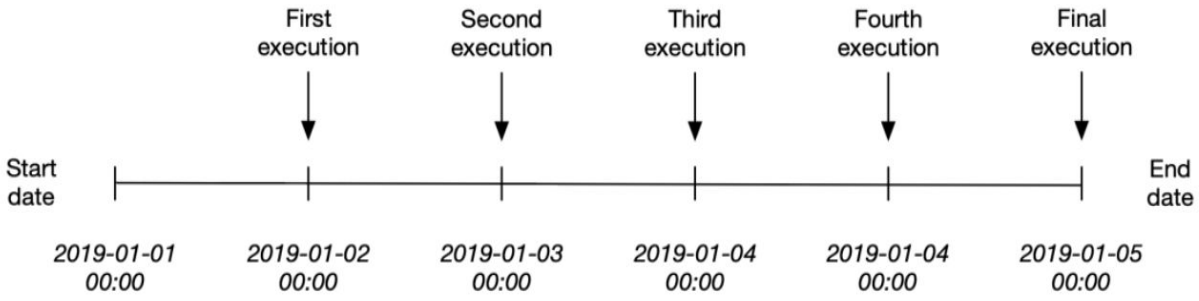
Start and End Time

Listing 3.3 Defining an end date for the DAG (03_with_end_date.py).

```
dag = DAG(
    dag_id="03_with_end_date",
    schedule_interval="@daily",
    start_date=dt.datetime(year=2019, month=1, day=1),
    end_date=dt.datetime(year=2019, month=1, day=5),
)
```



Cron-based intervals

```
# ┌─────── minute (0 - 59)
# │ ┌────── hour (0 - 23)
# │ │ ┌───── day of the month (1 - 31)
# │ │ │ ┌──── month (1 - 12)
# │ │ │ │ ┌─── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │       7 is also Sunday on some systems)
# * * * * *
```

- `0 * * * *` = hourly (running on the hour)
- `0 0 * * *` = daily (running at midnight)
- `0 0 * * 0` = weekly (running at midnight on Sunday)

Besides this, we can also define more complicated expressions such as the following:

- `0 0 1 * *` = midnight on the first of every month
- `45 23 * * SAT` = 23:45 every Saturday

- `0 0 * * MON,WED,FRI` = run every Monday, Wednesday, Friday at midnight
- `0 0 * * MON-FRI` = run every weekday at midnight
- `0 0,12 * * *` = run every day at 00:00 AM and 12:00 P.M.

Frequency-based intervals

To support this type of frequency-based schedule, Airflow also allows you to define scheduling intervals in terms of a relative time interval. To use such a frequency-based schedule, you can pass a *Timedelta* instance (from the *datetime* module in the standard library) as a schedule interval:

Listing 3.4 Defining a frequency-based schedule interval (04_time_delta.py)..

```
dag = DAG(
    dag_id="04_time_delta",
    schedule_interval=dt.timedelta(days=3), #A
    start_date=dt.datetime(year=2019, month=1, day=1),
          end_date=dt.datetime(year=2019, month=1, day=5),
)
```

#A Timedelta gives the ability to use frequency-based schedules.

This would result in our DAG being run every three days following the start date (on the 4th, 7th, 10th, etc. of January 2019). Of course, you can also use this approach to run your DAG every 10 minutes (using *timedelta(minutes=10)*) or every 2 hours (using *timedelta(hours=2)*).

Dynamic time references using execution dates

Listing 3.6 Using templating for specifying dates (06_templated_query.py).

```
fetch_events = BashOperator(
    task_id="fetch_events",
    bash_command=(
         "mkdir -p /data && "
                "curl -o /data/events.json "
         "http://localhost:5000/events?"
         "start_date={{execution_date.strftime('%Y-%m-%d')}}" #A
         "&end_date={{next_execution_date.strftime('%Y-%m-%d')}}" #B
    ),
    dag=dag,
)
```

#A Formatted *execution_date* inserted with Jinja templating.
#B *next_execution_date* holds the execution date of the next interval.

Listing 3.7 Using template shorthands (07_templated_query_ds.py).

```
fetch_events = BashOperator(
   task_id="fetch_events",
   bash_command=(
       "mkdir -p /data && "
       "curl -o /data/events.json "
      "http://localhost:5000/events?"
      "start_date={{ds}}&" #A
      "end_date={{next_ds}}" #B
   ),
   dag=dag,
)
```

#A *ds* provides YYYY-MM-DD formatted *execution_date*.
#B *next_ds* provides the same for *next_execution_date*.

Partitioning your data

Listing 3.8 Writing event data to separate files per date (08_templated_path.py).

```
fetch_events = BashOperator(
    task_id="fetch_events",
    bash_command=(
        "mkdir -p /data/events && "
        "curl -o /data/events/{{ds}}.json " #A
        "http://localhost:5000/events?"
        "start_date={{ds}}&"
        "end_date={{next_ds}}",
    dag=dag,
)
```

#A Response is written to templated filename.

Listing 3.10 Calculating statistics per execution interval (08_templated_path.py).. .....

```
def _calculate_stats(**context): #A
   """Calculates event statistics."""
   input_path = context["templates_dict"]["input_path"] #B
   output_path = context["templates_dict"]["output_path"]

   Path(output_path).parent.mkdir(exist_ok=True)

   events = pd.read_json(input_path)
   stats = events.groupby(["date", "user"]).size().reset_index()
   stats.to_csv(output_path, index=False)


calculate_stats = PythonOperator(
   task_id="calculate_stats",
   python_callable=_calculate_stats,
   templates_dict={
       "input_path": "/data/events/{{ds}}.json", #C
       "output_path": "/data/stats/{{ds}}.csv",
   },
   dag=dag,
)
```

#A Receive all context variables in this dict.
#B Retrieve the templated values from the templates_dict object.
#C Pass the values that we want to be templated.

Listing 3.13 Splitting jobs into multiple tasks to improve atomicity (11_atomic_send.py).

```
def _send_stats(email, **context):
    stats = pd.read_csv(context["templates_dict"]["stats_path"])
    email_stats(stats, email=email) #A


send_stats = PythonOperator(
    task_id="send_stats",
    python_callable=_send_stats,
    op_kwargs={"email": "user@example.com"},
    templates_dict={"stats_path": "/data/stats/{{ds}}.csv"},
    dag=dag,
)

calculate_stats >> send_stats
```

Template

Listing 4.13 Providing templated strings as input for the callable function

```
def _get_data(year, month, day, hour, output_path, **_):
   url = (
       "https://dumps.wikimedia.org/other/pageviews/"
       f"{year}/{year}-{month:0>2}/pageviews-{year}{month:0>2}{day:0>2}-{hour:0>2}0000.gz"
   )
   request.urlretrieve(url, output_path)


get_data = PythonOperator(
   task_id="get_data",
   python_callable=_get_data,
   op_kwargs={
       "year": "{{ execution_date.year }}", #A
       "month": "{{ execution_date.month }}",
       "day": "{{ execution_date.day }}",
       "hour": "{{ execution_date.hour }}",
       "output_path": "/tmp/wikipageviews.gz",
   },
   dag=dag,
)
```

Listing 4.15 Reading pageviews for given page names

```
extract_gz = BashOperator(
    task_id="extract_gz",
    bash_command="gunzip --force /tmp/wikipageviews.gz",
    dag=dag,
)


def _fetch_pageviews(pagenames):
   result = dict.fromkeys(pagenames, 0)
   with open(f"/tmp/wikipageviews", "r") as f: #A
       for line in f:
           domain_code, page_title, view_counts, _ = line.split(" ") #B
           if domain_code == "en" and page_title in pagenames: #C #D
               result[page_title] = view_counts

   print(result)
   # Prints e.g. "{'Facebook': '778', 'Apple': '20', 'Google': '451', 'Amazon': '9', 'Microsoft':
       '119'}"
```

```
fetch_pageviews = PythonOperator(
    task_id="fetch_pageviews",
    python_callable=_fetch_pageviews,
    op_kwargs={"pagenames": {"Google", "Amazon", "Apple", "Microsoft", "Facebook"}},
    dag=dag,
)
```

#A Open the file written in previous task
#B Extract the elements on a line
#C Filter only domain “en”
#D Check if page_title is in given pagenames