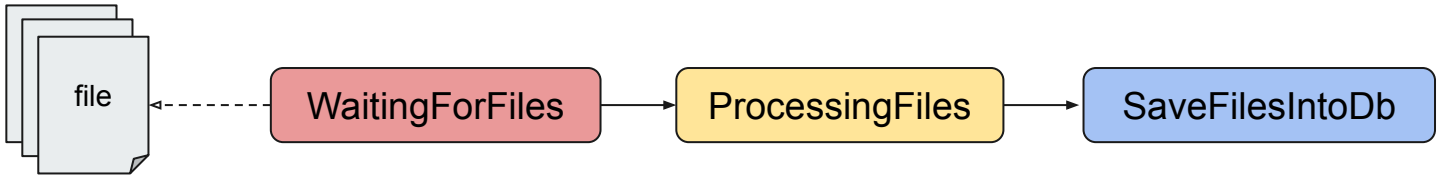
Control Your Tasks With SLAs

Service Level Agreement

Use Case

Let's suppose you are working at a company where you have a DAG waiting for files to come into a defined directory at 7am with a maximum of 5 min delay. Those files are required to be processed before 7:30am otherwise the dashboard used by your data analyst team will not work correctly for the current day. You may have this kind of DAG:



Use Case

According to the use case, 2 requirements must be met:

- The files should not arrive after 7:05am (5 min delay)
- The ProcessingFiles task should not exceed 15 mins of processing in order to have the data available in time (7:30 am max).

In order to fulfill those requirements we are going to use **SLAs**.

Definition

A Service Level Agreement (SLA) is a contract between a service provider and the end user that defines the level of service expected from the service provider.

SLAs do not define how the service itself is provided or delivered but rather define what the end user will received. The level of service definitions should be specific and measurable.

In Apache Airflow, Service Level Agreements are defined as the time by which a task or DAG should have succeeded.

Adding SLA to a Task

SLAs are set at a task level (Operator) as a **timedelta**. For example, if we want to add a SLA to a task using the BashOperator we will do the following:

```
BashOperator(task_id="t1", sla=timedelta(seconds=5), bash_command="echo 'test'")
```

In this example we are basically saying that the task t1 should not exceed 5 seconds to succeed. If it does, a SLA is recorded in the database and an email is automatically sent to the given email address of the DAG.

Adding SLA Callback to the DAG

When you instantiate a DAG, there is one parameter that you can use to call a function when a SLA is missed: `sla_miss_callback=func(dag, tl, btl, slas, btis)`

This parameter expects a function with the following parameters:

- `dag`: the dag object where the missed SLA(s) happened
- `task_list`: the task list having missed their SLA with their associated execution_time (full text)
- `blocking_task_list`: the task list being blocked by the tasks having missed their SLA with their associated execution_time (full text)
- `slas`: same as the task_list but in a list object (not in string format)
- `blocking_tis`: same as the blocking_task_list but in a list object (not in string format)

How the SLAs Work Internally

Here is exactly what is done in the source code:

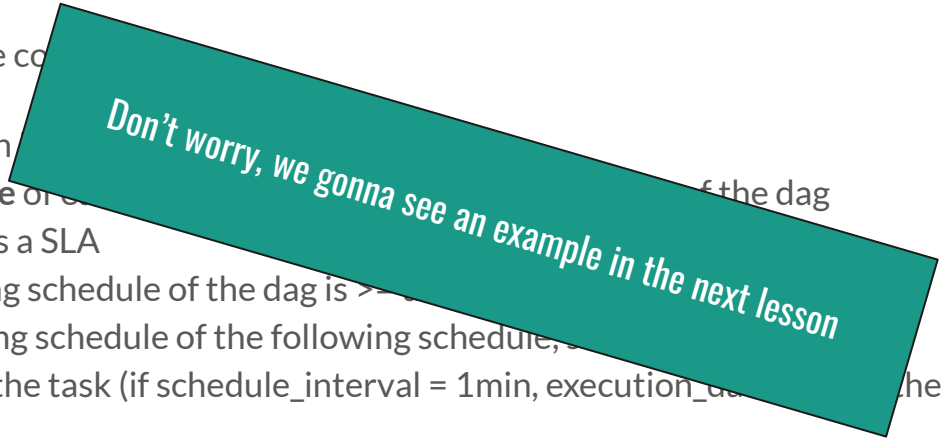
1. Save the `utcnow()` current date in UTC in a `ts` (timestamp) variable.
2. For each **maximum execution_date** of each **succeeded or skipped** task instance of the dag
    a. Check if the current task has a SLA
    b. If yes, loop until the following schedule of the dag is >= utcnow()
    c. In each loop, get the following schedule of the following schedule, so 2 schedule intervals from the execution_date of the task (if schedule_interval = 1min, execution_date = 9:00, the value will be 9:02)
    d. If the execution_date + 2 schedule intervals ahead + sla.time is less than utcnow(), a SLA is missing for the current task.

How the SLAs Work Internally

Here is exactly what is done in the source co

1. Save the `utcnow()` current date in
2. For each **maximum execution_date** of
   a. Check if the current task has a SLA
   b. If yes, loop until the following schedule of the dag is >=
   c. In each loop, get the following schedule of the following schedule, from the execution_date of the task (if schedule_interval = 1min, execution_d he value will be 9:02)
   d. If the execution_date + 2 schedule intervals ahead + sla.time is less than utcnow(), a SLA is missing for the current task.

f the dag



Scheduler Logs

- The `sla_miss_callback` function will print into the standard output of the process where the DagRun is executed. The logs are available at the path specified next to the `child_process_log_directory` configuration variable in airflow.cfg.
- In our case the logs are saved at this location `/home/airflow/airflow/logs/scheduler/latest`

Important Notes

- A SLA is **relative to the execution_date** of the task not the start time. You can only be alerted if the task runs “more than 30 minutes FROM the execution_date”. You won’t receive an alert if the task runs “more than 30 minutes”.
- You may think of using the execution_timeout parameter to express “more than 30 minutes” but it doesn’t serve the same purpose as a SLA. The execution_timeout parameter sets the maximum allowed time before a task is stopped and marked as being failed. With a SLA, your task will still run and you will be alerted that its processing time is longer than expected.
- Using SLAs with a DAG having a schedule_interval sets to None or @once has no effect. Why? Because to check if a missed SLA must be triggered or not, Airflow looks at the next schedule_interval which in this case, does not exist.

Important Notes

- If one or multiple tasks haven't succeed in time, an email alert is sent with the list of tasks that missed their SLA. This email alert is not sent from the callback and can't be turned off. The only way to avoid receiving email alerts is by setting the email parameter of your task to None.
- Be very careful when you use backfilling (process of replaying your DAG from the past) with tasks having a SLA. Indeed, you may end up with thousands of missed SLA in a very short period of time as it means for Apache Airflow that the tasks from the past didn't succeed in time.
- All tasks sharing the same missed SLA time are going be merged together in order to be sent by email. Also, each SLA is saved into the database.

Time to Code

I know it can be difficult to understand how SLAs work in Apache Airflow so let’s do some experiments!