Airflow sensors

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Mike Metzger
Data Engineer



Sensors

What is a *sensor*?

- An operator that waits for a certain condition to be true
 - Creation of a file
 - Upload of a database record
 - Certain response from a web request
- Can define how often to check for the condition to be true
- Are assigned to tasks

Sensor details

- Derived from airflow.sensors.base_sensor_operator
- Sensor arguments:
- mode How to check for the condition
 - mode='poke' The default, run repeatedly
 - mode='reschedule' Give up task slot and try again later
- poke_interval How often to wait between checks
- timeout How long to wait before failing task
- Also includes normal operator attributes

File sensor

- Is part of the airflow.sensors library
- Checks for the existence of a file at a certain location
- Can also check if any files exist within a directory

Other sensors

- ExternalTaskSensor wait for a task in another DAG to complete
- HttpSensor Request a web URL and check for content
- SqlSensor Runs a SQL query to check for content
- Many others in airflow.sensors and airflow.providers.*.sensors

Why sensors?

Use a sensor...

- Uncertain when it will be true
- If failure not immediately desired
- To add task repetition without loops

Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Airflow executors

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Mike Metzger
Data Engineer



What is an executor?

- Executors run tasks
- Different executors handle running the tasks differently
- Example executors:
 - SequentialExecutor
 - LocalExecutor
 - KubernetesExecutor

SequentialExecutor

- The default Airflow executor
- Runs one task at a time
- Useful for debugging
- While functional, not really recommended for production

LocalExecutor

- Runs on a single system
- Treats tasks as processes
- Parallelism defined by the user
- Can utilize all resources of a given host system

KubernetesExecutor

- Uses a Kubernetes as task manager
- Multiple worker systems can be defined
- Is significantly more difficult to setup & configure
- Extremely powerful method for organizations with extensive workflows

Determine your executor

- Via the airflow.cfg file
- Look for the executor= line

```
repl:~$ cat airflow/airflow.cfg | grep "executor = "
executor = SequentialExecutor
repl:~$
```

Determine your executor #2

- Via the first few lines of airflow info
- executor | SequentialExecutor

```
repl:~$ airflow info
Apache Airflow
version
                         2.7.1
executor
                         SequentialExecutor
task_logging_handler
                         airflow.utils.log.file_task_handler.FileTaskHandler
                         sqlite:///home/repl/airflow/airflow.db
sql_alchemy_conn
dags_folder
                         /home/repl/workspace/dags
plugins_folder
                         /home/repl/workspace
base_log_folder
                         /home/repl/airflow/logs
remote_base_log_folder
```



Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Debugging and troubleshooting in Airflow

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Mike Metzger
Data Engineer

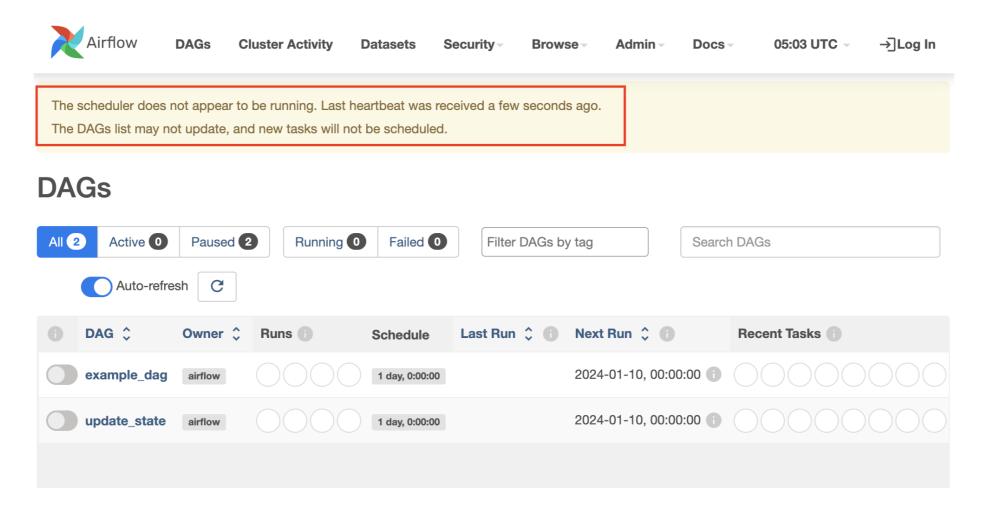


Typical issues...

- DAG won't run on schedule
- DAG won't load
- Syntax errors

DAG won't run on schedule

Check if scheduler is running



• Fix by running airflow scheduler from the command-line.

DAG won't run on schedule (part 2)

- At least one schedule_interval hasn't passed.
 - Modify the attributes to meet your requirements.
- Not enough tasks free within the executor to run.
 - Change executor type
 - Add system resources
 - Add more systems
 - Change DAG scheduling

DAG won't load

- DAG not in web UI
- DAG not in airflow dags list

Possible solutions

- Verify DAG file is in correct folder
- Determine the DAGs folder via airflow.cfg
- Note, the folder must be an absolute path

```
repl:~$ head airflow/airflow.cfg
[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository
# This path must be absolute
dags_folder = /home/repl/airflow/dags
```

Syntax errors

- The most common reason a DAG file won't appear
- Sometimes difficult to find errors in DAG
- Two quick methods:
 - Run airflow dags list-import-errors
 - Run python3 <dagfile.py>

airflow dags list-import-errors

Running the Python interpreter

```
python3 dagfile.py:
```

With errors

```
(af) mmetzger@hugo:~/airflow/dags$ python3 simple_dependency.py
Traceback (most recent call last):
   File "simple_dependency.py", line 13, in <module>
     task1 = BasOperator(task_id='first_task',
NameError: name 'BasOperator' is not defined
```

Without errors

```
(af) mmetzger@hugo:~/airflow/dags$ python3 simple_python_operator.py
(af) mmetzger@hugo:~/airflow/dags$
```

Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



SLAs and reporting in Airflow

INTRODUCTION TO APACHE AIRFLOW IN PYTHON



Mike Metzger
Data Engineer



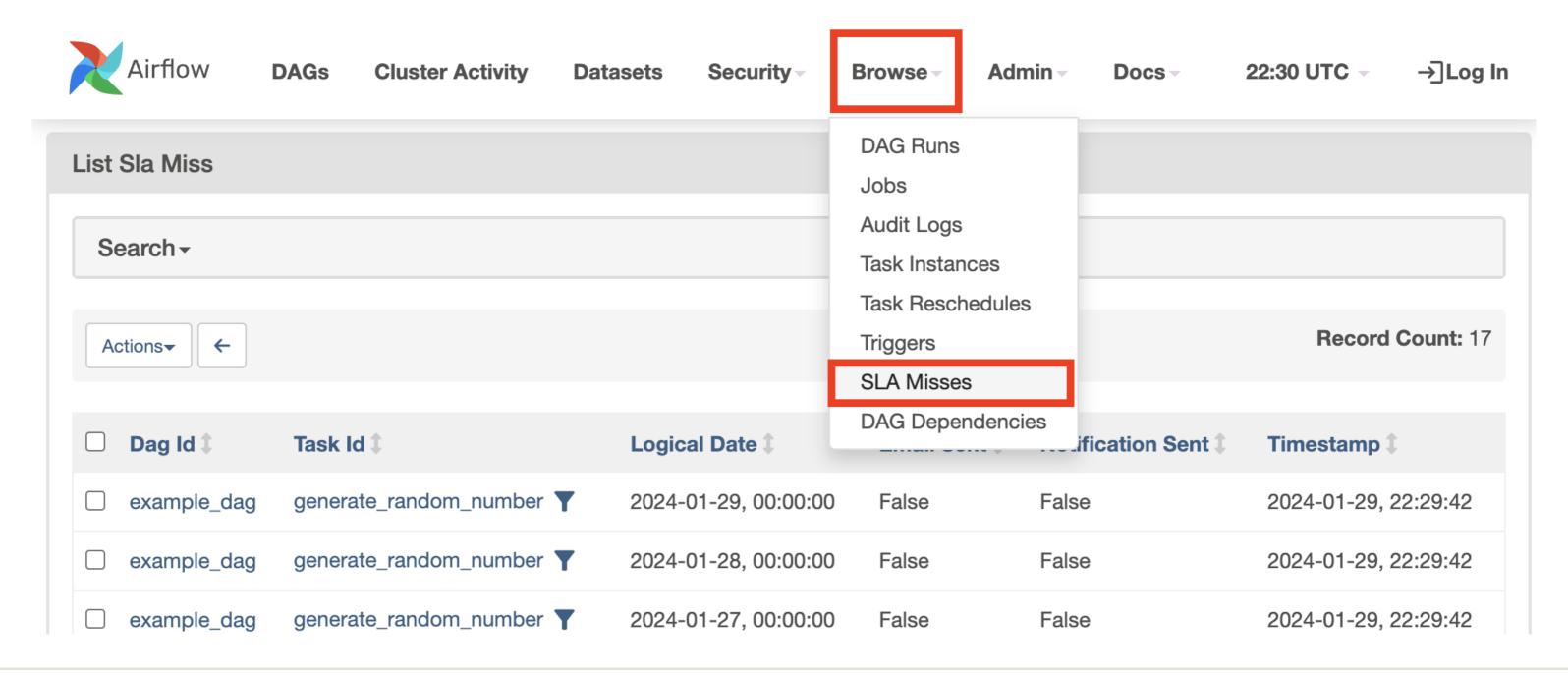
SLAs

What is an SLA?

- An SLA stands for Service Level Agreement
- Within Airflow, the amount of time a task or a DAG should require to run
- An SLA Miss is any time the task / DAG does not meet the expected timing
- If an SLA is missed, an email is sent out and a log is stored.
- You can view SLA misses in the web UI.

SLA Misses

Found under Browse: SLA Misses





Defining SLAs

Using the 'sla' argument on the task

On the default_args dictionary

```
default_args={
  'sla': timedelta(minutes=20),
  'start_date': datetime(2023,2,20)
}
dag = DAG('sla_dag', default_args=default_args)
```

timedelta object

- In the datetime library
- Accessed via from datetime import timedelta
- Takes arguments of days, seconds, minutes, hours, and weeks

```
timedelta(seconds=30)
timedelta(weeks=2)
timedelta(days=4, hours=10, minutes=20, seconds=30)
```

General reporting

- Options for success / failure / error
- Keys in the default_args dictionary

```
default_args={
   'email': ['airflowalerts@datacamp.com'],
   'email_on_failure': True,
   'email_on_retry': False,
   'email_on_success': True,
   ...
}
```

Within DAGs from the EmailOperator

Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

