Airflow Mini Project

Log Analyzer

Estimated Time: 3-5 hours

This mini-project will build on the work you did in your previous Airflow mini-project.

Once you have scheduled a DAG object in Airflow, you need to monitor the status of various jobs in your pipeline on a regular basis. One way to monitor the job status is to analyze the log messages generated from each run. In this project, you will create a log analyzer in Python to monitor the DAG Airflow you set up in the previous mini-project.

Your log analyzer should show the following information:

- The total count of error messages
- A detailed message regarding each error

Learning Objectives

With this mini-project, you will exercise the text processing techniques using Python. Parsing text files and getting useful information from logs is a common practice in real life projects. Also, you will learn where the logs are located in Airflow processes.

In this mini-project, you will:

- Use text processing techniques in Python to make sense of logs
- Learn where logs are located in Airflow
- Learn how to monitor automated Airflow DAGs to ensure they are working properly

Instructions

1. Locate the log files for your application

In Airflow, the location of the log file is configured at ~/airflow/airflow.cfg. Set the variable base_log_folder to the location of your log file.

```
# The folder where airflow should store its log files
# This path must be absolute
base_log_folder = /Users/wpengyu/airflow/logs
```

You will find the directory with the name of your DAG application, under which each task in the DAG will appear as a subdirectory.

```
sh-3.2$ ls -l ~/airflow/logs/
total 0
drwxr-xr-x   3 myuser  staff   96 Sep 17 20:26 dag_processor_manager
drwxrwxrwx   9 myuser  staff  288 Sep 25 20:11 marketvol
drwxr-xr-x  20 myuser  staff  640 Sep 30 20:02 scheduler
drwxrwxrwx   5 myuser  staff  160 Sep 17 20:27 tutorial
sh-3.2$ ls -l ~/airflow/logs/marketvol/
total 0
drwxrwxrwx  15 myuser  staff  480 Sep 30 20:03 copy_AAPL
drwxrwxrwx  15 myuser  staff  480 Sep 30 20:03 copy_TSLA
drwxrwxrwx  15 myuser  staff  480 Sep 30 20:03 init
drwxrwxrwx  15 myuser  staff  480 Sep 30 20:03 python_download_AAPL
drwxrwxrwx  15 myuser  staff  480 Sep 30 20:03 python_download_TSLA
drwxrwxrwx   8 myuser  staff  256 Sep 30 20:03 python_show_count
drwxrwxrwx   9 myuser  staff  288 Sep 24 20:00 python_show_vol
```

2. Create a log analyzer with Python

Log message follows a standard format. Each log entry starts with a datetime and the code line number, followed by the message types (INFO, WARNING, ERROR, etc).

```
[2020-10-03 20:00:25,474] {scheduler_job.py:963} INFO - 2 tasks up for execution:
       <TaskInstance: marketvol.python_download_AAPL 2020-10-03 00:00:00+00:00 [scheduled]>
       <TaskInstance: marketvol.python_download_TSLA 2020-10-03 00:00:00+00:00 [scheduled]>
[2020-10-03 20:00:25,478] {scheduler_job.py:997} INFO - Figuring out tasks to run in
Pool(name=default_pool) with 128 open slots and 2 task instances ready to be queued
[2020-10-03 20:00:25,478] {scheduler_job.py:1025} INFO - DAG marketvol has 0/16 running and
queued tasks
[2020-10-03 20:00:25,478] {scheduler_job.py:1025} INFO - DAG marketvol has 1/16 running and
queued tasks
```

```
[2020-10-03 20:00:25,481] {scheduler_job.py:1085} INFO - Setting the following tasks to queued
state:
	<TaskInstance: marketvol.python_download_AAPL 2020-10-03 00:00:00+00:00 [scheduled]>
	<TaskInstance: marketvol.python_download_TSLA 2020-10-03 00:00:00+00:00 [scheduled]>
[2020-10-03 20:00:25,494] {scheduler_job.py:1159} INFO - Setting the following 2 tasks to queued
state:
	<TaskInstance: marketvol.python_download_AAPL 2020-10-03 00:00:00+00:00 [queued]>
	<TaskInstance: marketvol.python_download_TSLA 2020-10-03 00:00:00+00:00 [queued]>
[2020-10-03 16:42:53,667] {scheduler_job.py:237} WARNING - Killing PID 41926
[2020-10-03 16:42:53,668] {scheduler_job.py:237} WARNING - Killing PID 41927
[2020-09-26 20:15:33,479] {taskinstance.py:1150} ERROR - No columns to parse from file
```

For this task, you need to find all the error messages to report. You will create a Python application to report the total number of errors and their associated error messages.

2.1. Iterate through the root directory and get all the log files
The Python application should target the root directory of the log, which will ensure all the log files will be analyzed. Python provides the "pathlib" module representing filesystem paths. To recursively list all files ending with extension ".log" follow this example:

```
from pathlib import Path
file_list = Path(log_dir).rglob('*.log')
```

2.2. Create a Python method to parse each file
Name the method "analyze_file" and use it to parse each log file. The method should return the following information:

- The total count of error entries from this file
- A list of error message details (the errors themself)

The method will be called like this:

```
count, cur_list = analyze_file(file)
```

The Python operators should call the analyze_file function. Name the operators t1 and t2 for the symbols AAPL and TSLA, respectively.

2.3. Print the cumulative information collected from all files
The application should print the cumulative error count and messages from all the log files that are analyzed

```
sh-3.2$ python3 log_analyzer.py /Users/myuser/airflow/logs/marketvol
Total number of errors: 6
Here are all the errors:
```

```
[2020-09-27 20:12:59,742] {taskinstance.py:1150} ERROR - No columns to parse from file
[2020-09-27 20:00:41,364] {taskinstance.py:1150} ERROR - No columns to parse from file
[2020-09-26 20:20:46,692] {taskinstance.py:1150} ERROR - No columns to parse from file
[2020-09-26 20:15:33,479] {taskinstance.py:1150} ERROR - No columns to parse from file
[2020-10-03 20:06:04,487] {taskinstance.py:1150} ERROR - No columns to parse from file
[2020-10-03 20:00:51,254] {taskinstance.py:1150} ERROR - No columns to parse from file
```

Instruction for Submission:

- Push your Python code and shell script to GitHub.
- Add a readme file, including the steps to run your code and verify the result. Your mentor should be able to run it by following your instructions.
    - Resources for creating a readme file: Example 1, Example 2
- Attach the command line execution log for the successful job run captured in a text file.