A look under the hood of the Airflow logging subsystem

Airflow Summit 2022
May 24 2022 @ New York Times Building
Philippe Gagnon

+Solutions Architect 🪐 @ Astronomer, Inc. 🪐
+Based in Montreal, Canada 🇨🇦
+Works on data platform architecture and implementation in heavily regulated industries (e.g. finance 🏦, healthcare 🩺) since 2017, mostly around stacks relying on open-source projects with top-tier communities
What is covered

- Logging in Airflow at a high level
- Default file-based logging process
- Remote logging to object storage
- Remote logging to dedicated services
- Roll your own task log handler
Airflow logging at a high level

logging.info("It's running...")
logging.error("It's broken...")

Task Handler

Logs mini-webserver
Airflow Worker

Airflow components log

Console

Storage

Task logs from storage
TaskLogReader
Task Handler

Airflow webserver

Task logs from worker webserver

Task logs from Kubernetes API server
Airflow logging core concepts

+ Leverages the stdlib `logging` module
+ Everything is really configured through `airflow_local_settings.py`
+ Defines three loggers: `airflow.processor`, `airflow.task`, `flask_appbuilder`, along with the root logger.
+ Logs retrieval is provided by implementing a `read(...)` method in task handlers (not part of the stdlib spec!)
+ Logs display in the webserver is implemented through the `TaskLogReader` class.
Airflow logging initialization

settings.py: initialize()

logging_config.py: configure_logging()

airflow.cfg: logging/logging_class_path

Set

Unset

logging_config = import_string(logging_class_path)

logging_config = DEFAULT_LOGGING_CONFIG
dictConfig schema details

```json
{
  version,  # must be 1
  formatters,
  filters,
  handlers,
  loggers,
  root,
  incremental,  # if False: replaces the existing configuration
  disable_existing_loggers,  # disables existing loggers
}
```
Out of the box

+ DEFAULT_LOGGING_CONFIG dictionary passed to logging.config.dictConfig

+ Handlers: RedirectStdHandler (root), FileTaskHandler (task logs), FileProcessorHandler (dag processor logs)
  + File...Handlers wrap NonCachingFileHandler which inherits from stdlib’s FileHandler
  + RedirectStdHandler outputs to sys.stderr/stdout
Writing logs using FileTaskHandler

+ Writes to local filesystem.
+ Delegates to FileHandler.emit(…)
+ Logs routed to proper file according to template defined in airflow.cfg log_filename_template (_render_filename)
+ Log directory and permissions created via _init_file
FileTaskHandler read(...) logic

User requests task logs

Logs on local filesystem

No

Kubernetes Executor?

Yes

Worker pod still exists?

Yes

Read worker pod logs using kube client

No

Unable to fetch logs

Read and display task logs from local fs

Yes

Fetch logs from worker logs webserver
Remote Logging

+ Feature enabled through airflow.cfg (set remote_logging = True)...

+ ... but actually configured in airflow_local_settings.py

```python
if REMOTE_LOGGING:
    if REMOTE_BASE_LOG_FOLDER.startswith('gs://'):
        ...
        DEFAULT_LOGGING_CONFIG['handlers'].update(GCS_REMOTE_HANDLERS)
    elif REMOTE_BASE_LOG_FOLDER.startswith('s3://'):
        ...
        DEFAULT_LOGGING_CONFIG['handlers'].update(S3_REMOTE_HANDLERS)
    elif REMOTE_BASE_LOG_FOLDER.startswith('cloudwatch://'):
        ...
        DEFAULT_LOGGING_CONFIG['handlers'].update(CLOUDWATCH_REMOTE_HANDLERS)
```
Remote Logging to Object Storage

+ Amazon S3, Google Cloud Storage, Azure Blob Storage mainly.
+ Very important to note is that this mechanism only uploads logs to object storage *when the logging handler is closed*, which in normal circumstances only happens when the application (i.e. task in this case) exits.
+ This is implemented by overloading the `close(...)` method in the log handler.
Example: Logging to S3

Task is running...

Task exits (success or failure*)

S3TaskHandler.close(): ...

S3TaskHandler.s3_write(log, remote_loc): ...

Local task log file

S3Hook(REMOTE_LOG_CONN_ID)

TaskLogReader.read_log_chunks(...): self.log_handler.read(...)

S3TaskHandler._read(...)

Log in S3

Not in S3

fallback to FileTaskHandler._read(...)
Remote Logging to external log services

+ **Elasticsearch, Cloudwatch Logs, Stackdriver** (Google Ops Suite)
+ ⚠ These log handlers only implement read functionality, and defer to FileTaskHandler for writing!
+ It’s necessary to rely upon an external application to ship logs to the remote logging service
+ In general, that ends up being **fluentd, fluentbit or logstash**
Example: Logging to Elasticsearch

Task is running...

Local task log file

watch

fluentd

ship

TaskLogReader.
read_log_chunks(...): self.log_handler.read(...)

Elasticsearch
TaskHandler._read(...)

query: log_id={dag_id}-{task_id}-{execution_date}-{try_number}
Primer on rolling your own

class MyTaskHandler(logging.Handler, LoggingMixin):
    def __init__(self):
        super(MyTaskHandler, self).__init__()

    def emit(self, record: logging.LogRecord):
        <Logic to "stream" logs goes here>

    def close(self):
        <Logic to ship logs in bulk goes here>

    def read(self, task_instance, try_number=None, metadata=None):
        <Logic to fetch logs goes here>
Or starting from FileTaskHandler

class MyTaskHandler(FileTaskHandler, LoggingMixin):
    def __init__(self):
        super(MyTaskHandler, self).__init__()

    def emit(self, record: logging.LogRecord):
        ...

    def close(self):
        ...

    def _read(self, task_instance, try_number=None, metadata=None):
        ...
Thank you! ❤

P.S. We brought swag! Come see me!

👕🌈✨