Airflow 3 Proposal: Synchronous DAG Execution

This document is intended to elaborate on the “Interactive DAG Execution” conversations in the Dev list and the initial Airflow 3 proposal shared a couple of weeks ago. Based on some of the feedback from the community, we decided to change the working name from “Interactive DAG Execution” to “Synchronous DAG Execution”. This is not yet a formal AIP, but is intended to facilitate a structured discussion, which will then be followed up with a formal AIP.

Synchronous DAG Execution use cases

Airflow historically has been used for “batch” orchestration and this has been both a strength and a limitation. But, even as part of Airflow’s early days, the concept of “triggering” a DAG run through the UI for debugging has been a core part of the value proposition.

This concept of triggering a DAG run was also added to the Airflow API, so that DAGs could be invoked programmatically through the API, instead of being only based on a schedule.

As part of Data Driven Scheduling, introduced in Airflow 2.4, DAG invocation progressed further to support triggering of DAGs based on Dataset update events. This was further enhanced in Airflow 2.9, with the introduction of API support for Dataset Update events, so that Dataset updates outside of Airflow could be used to trigger Dependent data pipelines.

More recently, there have been many requests for Synchronous DAG Execution with Airflow. Some of the use cases are detailed below:

Data-driven Applications

Many Enterprise applications have become data-driven applications, which provide a response to a customer based on personalized information. For example, a hotel reservation experience for a customer could include personalized options based on prior customer experience, which could be aggregated across multiple data systems and obtained during the customer interaction experience.

Data aggregation Bots

Agents being developed based on AI are commonly being used to aggregate a summary or a review based on prior trained knowledge of the domain.

Generative AI

With the advent of Generative AI, a common use case for DAG execution is for inference. Using the AskAstro LLM application as an example, the steps needed for responding to a question are on the lines of:

1. Rephrase the question using an LLM (multiple times)
2. Submit multiple versions of the question (original and re-phrased versions) to an LLM
3. De-duplicate the results
4. Optionally verify the results including the associated references for the results
5. Return the answer to the question

The above steps can easily be mapped to a Directed Acyclic Graph. With Airflow, this could be represented as a DAG with (2) and (4) as Dynamically Mapped Tasks.
However, to cleanly support the above DAG execution for Inference, the final step (5) needs to be supported in Airflow as a “return the result”.

Proposal

The proposal here has multiple parts as detailed below.

1. Enable a DAG to be run at the same time by one or many users, possibly with different parameters, without requiring a unique logical date for each DAGrun. This definitely builds on the separation between “logical date” and “data interval”, which has been incrementally been worked on since Airflow 2.2. Another way of saying this is to support “non-data-interval DAG runs”.
   This particular feature has been requested often including for Hyper parameter tuning as part of Machine Learning. (This is not proposing that the API server itself runs the DAG)

2. Add language support within the DAG to **return the result** in Pythonic form either as a value or a reference. In practice, this could mean the designation of a single **result task** in a DAG.
   Therefore, the API invocation for Synchronous DAG Execution would be similar to the “Trigger DAG” invocation, but would wait for a response from DAG execution, which could be returned to the user. The reference returned could be a dataset or to a blob in object storage.
   An initial approach for this could be:
   - Structure the existing Trigger API to return a "job id", and
   - Add a new “poll / wait for completion” API which can be invoked using the above “job id”.
   - Thereby enabling reliable handling of long running jobs.

3. Return handling
   It’s critical for a DAG to return a status, even if it is a failure. In the current DAG execution model, there is no completion event in the case of a task failure, with the exception of a “Teardown” event. With synchronous execution, a DAG must always:

- Return the result to the invoking API as soon as the result is available (from the **result task**), without having to wait for the teardown task (if any) to complete (this is the success case), and
- Return the failure status to the invoking API as a key task in the DAG has failed (inc upstream_failed etc.), rather than waiting for DagRun completion (i.e waiting for teardown task completion).