Airflow 3 Proposal: Task Context aka Task SDK

This document is intended to elaborate on the “Task Execution Context” conversations in the Dev list and the initial Airflow 3 proposal shared a couple of weeks ago. 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.

Motivation

Historically, Airflow’s task execution context has been oriented around local execution within a relatively trusted networking cluster.
This includes:

- the interaction between the Executor and the process of launching a task on Airflow Workers,
- the interaction between the Workers and the Airflow meta-database for connection and environment information as part of initial task startup,
- the interaction between the Airflow Workers and the rest of Airflow for heartbeat information, and so on.

This has been accomplished by colocating all of the Airflow task execution code with the user task code in the same container and process.

For Airflow users at scale i.e. supporting multiple data teams, this has posed many operational challenges:

- Dependency conflicts for administrators supporting data teams using different versions of providers, libraries, or python packages
- Security challenge in the running of customer-defined code (task code within the DAGs) for multiple customers within the same operating environment and service accounts
- Scalability of Airflow since one of the core Airflow scalability limitations has been the number of concurrent database connections supported by the underlying database instance. To alleviate this problem, we have consistently, as an Airflow community, recommended the use of PgBouncer for connection pooling, as part of an Airflow deployment.
- Operational issues caused by unintentional reliance on internal Airflow constructs within the DAG/Task code, which only and unexpectedly show up as part of Airflow production operations, coincidentally with, but not limited to upgrades and migrations.
- Operational management based on the above for Airflow platform teams at scale, because different data teams naturally operate at different velocities. Attempting to support these different teams with a common Airflow environment is unnecessarily challenging.

The internal API to reduce the need for interaction between the Airflow Workers and the metadatabase is a big and necessary step forward. However, it doesn't fully address the above challenges. The proposal below builds on the internal API proposal and goes significantly further to not only address these challenges above, but also enable the following key use cases:

1. Ensure that this interface reduces the interaction between the code running within the Task and the rest of Airflow. This is to address unintended ripple effects from core Airflow changes which has caused numerous Airflow upgrade issues, because Task (i.e. DAG) code relied on Core Airflow abstractions. This has been a common problem pointed out by numerous Airflow users including early adopters. This proposal would enable Airflow users to upgrade Airflow system components (Scheduler, et al), without impacting DAG user code.
2. Enable quick, performant execution of tasks on local, trusted networks, without requiring the Airflow workers / tasks to connect to the Airflow database to obtain all the information required for task startup,
3. Enable remote execution of Airflow tasks across network boundaries, by establishing a clean interface for Airflow workers on remote networks to be able to connect back to a central Airflow service to access all information needed for task execution. This is foundational work for remote execution.
4. Enable a clean language agnostic interface for task execution, with support for multiple language bindings, so that Airflow tasks can be written in languages beyond Python.

Proposal

The proposal here has multiple parts as detailed below.

1. Formally split out the Task Execution Interface as the Airflow Task SDK (possibly name it as the Airflow SDK), which would be the only interface to and from Airflow Task User code to the Airflow system components including the meta-database, Airflow Executor, etc.
2. Disable all direct database interaction from the Airflow Workers including Tasks being run on those Airflow Workers and the Airflow meta-database.
3. The Airflow Task SDK will include interfaces for:
    - Access to needed Airflow Connections, Variables, and XCom values
    - Report heartbeat
    - Record logs
    - Report metrics
4. The Airflow Task SDK will support a Push mechanism for speedy local execution in trusted environments.
5. The Airflow Task SDK will also support a Pull mechanism for the remote Task execution environments to access information from an Airflow instance over network boundaries.
6. The Airflow Task SDK will be designed to support multiple language bindings, with the first language binding of course being Python.

Assumption: The existing AIP for Internal API covers the interaction between the Airflow workers and Airflow metadatabase for heartbeat information, persisting XComs, and so on.