**Airflow Plugins**

Airflow was built with the intention of allowing its users to extend and customize its functionality through plugins. The most common types of user-created plugins for Airflow are Operators and Hooks. These plugins make DAGs reusable and simpler to maintain.

To create custom operator, follow the steps:

1. Identify Operators that perform similar functions and can be consolidated
2. Define a new Operator in the plugins folder
3. Replace the original Operators with your new custom one, re-parameterize, and instantiate them.

**Airflow Contrib**

Airflow has a rich and vibrant open source community. This community is constantly adding new functionality and extending the capabilities of Airflow. As an Airflow user, you should always check [**Airflow contrib**](https://github.com/apache/airflow/tree/master/airflow/contrib) before building your own airflow plugins, to see if what you need already exists.

Operators and hooks for common data tools like Apache Spark and Cassandra, as well as vendor specific integrations for Amazon Web Services, Azure, and Google Cloud Platform can be found in Airflow contrib. If the functionality exists and its not quite what you want, that’s a great opportunity to add that functionality through an open source contribution.

[**Check out Airflow Contrib**](https://github.com/apache/airflow/tree/master/airflow/contrib)

### Task Boundaries

DAG tasks should be designed such that they are:

* Atomic and have a single purpose
* Maximize parallelism
* Make failure states obvious

Every task in your dag should perform **only one job.**

“Write programs that do one thing and do it well.” - Ken Thompson’s Unix Philosophy

##### Benefits of Task Boundaries

* Re-visitable: Task boundaries are useful for you if you revisit a pipeline you wrote after a 6 month absence. You'll have a much easier time understanding how it works and the lineage of the data if the boundaries between tasks are clear and well defined. This is true in the code itself, and within the Airflow UI.
* Tasks that do just one thing are often more easily parallelized. This parallelization can offer a significant speedup in the execution of our DAGs.

NEXT