AIP-8 Split Providers into Separate Packages for Airflow 2.0

Status

State	Completed
Discussion Thread	AIP-8 Split Hooks/Operators into Separate Packages Voting discussion.
JIRA	AIRFLOW-3644 - Jira project doesn't exist or you don't have permission to view it.
Created	$action.dateFormatter.formatGivenString("yyyy-MM-dd", $content.getCreationDate())
In Release	2.0.0

Motivation

Apache Airflow integrates with many different software systems, with hooks and operators to Amazon Web Services, Google Cloud Platform, Microsoft Azure, and more. As cloud APIs evolve, they require changes to the Airflow operator and/or hook to support new features and bug fixes.

Cloud API launch schedules do not align with the Airflow release schedule. To use the latest cloud features or bug fixes, developers need to run Airflow at master or maintain a fork of Airflow and backport patches.

In addition, the Airflow issue tracker receives many questions/bugs for operator-specific issues. No single contributor has expertise across all hooks /operators. This produces overhead and delays as issues are routed to someone with appropriate expertise.

Context

Airflow 1.10 (and before) has been distributed as a single, monolithic package containing the “core” of Airflow as well as a number of operators, hooks, packages that implement various “providers” - cloud providers, databases, etc,. Airflow in 1.10* (and before) has always been distributed as a single “apache-airflow-X.Y” package in PyPI: https://pypi.org/project/apache-airflow/

. This however has proven to be problematic:

- with ~ every 2 months cadence of releases users have to wait few months for releasing bug-fixes
- there is “all-in-one” release which means that problems in specific areas (“google” or “aws” or “kubernetes” cannot be addressed independently from each other

In parallel to regular **airflow-1.10.*** releases we’ve also introduced backport package releases. Those are 56 packages that implement the current “master” version of “providers” packages (https://pypi.org/search/?q=%22apache-airflow-backport-providers%22&o=). Those “providers” packages were made possible by AIP-21 implementation. With an enormous effort from the community, we standardized and moved the “non-core” providers out of the regular “airflow” unstructured sources into very well structured and maintained “providers” packages.

While we currently released just one wave of those “provider” packages (and we are releasing the 2nd wave now), it has already been proven to be quite highly used and expected by the users. At least Google providers packages have proven to be used by the Google Composer team (including embedding Google providers in their images) “Cloud Composer 1.11.1+: Backport providers are installed by default for Airflow 1.10.6 and 1.10.9.” https://cloud.google.com/composer/docs/release-notes#August_03_2020

State for 1.10*

The current state of provider package automation

For now, the installation of airflow 1.10 can be augmented by adding the backport provider packages to make it possible to use the latest operators in 1.10. This can be done by manually installing appropriate ‘apache-airflow-backport-providers’ packages. We keep all the code in monorepo - where “airflow” and “providers’ code is shared in the same repo.

The following automation is implemented:

- We have automation in place to build changelog and release packages separately for each provider (or all at once, or only changed since the last release -> we need to make sure the last point is already fully operational).
- We have automation in place to create dependencies between [extras] and provider packages. We know which extras are linked with which providers and we automated this

- We have automation in place to detect dependencies between provider packages (for example we know -google provider might depend on -amazon packages for amazon->google transfer. Those are automatically added as “pip extras” when releasing packages. For example, installing “apache-airflow-backport-providers-google[amazon] - will install both “google” provider’s package and “amazon” one.
- We have automation in place (in CI) to make sure separate providers are installable one-by-one
- We can prepare release notes and release packages individually or in groups.

While we have not released single packages yet (only many at once), this model has proven to be working well. No major complaints or problems reported even after official incorporation by Google Composer, Astronomer and others.

The current state of constraint management

We currently have a special way of managing constraints as explained in https://github.com/apache/airflow/blob/master/CONTRIBUTING.rst#pinned-constraint-files

We currently keep the set of “latest working constraints” in a separate orphaned branch for each repository version. This is updated automatically when a new set of constraints passes all our checks. Constraints are automatically derived from the “setup.py” of apache airflow and tested automatically during the CI builds. This happens at every merge - even without the knowledge and actions of contributors. Committers are informed of the potential dependency issues by the failures in master CI builds, but it should not impact regular PRs.

Requirements / Constraints

- We would like Hook and Operator packages are releasable separately from the core Airflow package
- Hooks / operators that do not tie to a specific system or are part of the standard Python distribution (e.g. sqlite3) remain in the core package
- We want to keep simplicity of development of Apache Airflow, where provider-specific code can be developed using synchronized set of dependencies and can be tested to work together with other providers.
- It should be possible to identify people involved in developing each provider to be able to automatically determine who are the best people for each provider package.

The proposal for 2.0

The core of the proposal

We continue using the mono-repo approach. That has the added benefit that we can test everything together easily, that we keep dependencies configuration the same as we have now and we can have the single “constraint” file per python version which described a “known” working set of dependency versions for the whole “suite” of Core and Providers.

We introduce per-provider package separations similar to backport packages.

- For Airflow 2.0 those will be ‘`apache-airflow-providers*`’ rather than ‘`apache-airflow-backport-providers*`’.
- The ‘apache-airflow-providers*’ packages will have ‘apache-airflow>=2.0.0” requirement so that they are installable for Airflow 2.0+ only. Mechanism of building those will be very similar to the “backport” packages
- We will keep requirements in single `setup.py` for the whole suite o packages - and continue keeping it as open as possible
- The current way of managing dependencies (including constraint file) will be kept - providing one “source of truth” with the list of packages that are “proven to be working” by the CI process - and updated automatically as it is happening today.
- Versioning proposal is still to be decided closer to the release time (we want to come up with consistent process proposal to handle it). The options we considered so far:
    - CALVER with MAJOR AIRFLOW prefix (**2.YYYY.MM.DD**) for all 2.* Airflow releases. In the future **3.YYYY.MM.DD** for all 3.* releases. Backwards incompatibility is only allowed between MAJOR Airflow Versions
    - SEMVER separately for each package with a process that allows to mark PRs as introducing features/breaking changes to aid automation of release process. Dependency to Airflow version is maintained only by setup.py specification for each package.
    - SEMVER for each package but the package name contains major airflow version (**apache-airflow-2-provider-google**)
- We keep releasing backport packages for limited time (3 months) after 2.0 release

- We add “requirements” to current extras of Apache Airflow. Currently, Apache Airflow has a number of “extras” that very closely correspond to the “providers” we have (those need to be updated and names should be synchronized to have a perfect 1-1 match. We update the extras mechanism to both - correspond 1-1 with the “provider package names” but also we can add separate requirements for each extra (automatically) to add the corresponding “provider package” as the “extra” requirement. For example “google” extra will have the “apache-airflow-provider-google” package as a requirement. This is a powerful mechanism that allows us to keep backward-compatibility in the way of installing Apache Airflow.

Consequences

- the core “apache-airflow” package is stripped-off any providers for 2.0 release
- we have 57 new ‘apache-airflow-providers-*’ packages (for example ‘apache-airflow-providers-google’ or ‘apache-airflow-cncf-kubernetes’)
- installing the bare “apache-airflow” package makes it possible to run basic DAGs as the “fundamental” operators and hooks are included. All Example Dags from the core package should work.
- We keep backward compatibility with the current way of installing Apache Airflow. https://github.com/apache/airflow/blob/master/docs/installation.rst - will install not only the “google” and “postgres” basic PIP dependencies but also the corresponding “apache-airflow-providers-google” and “apache-airflow-providers-postgres” packages in their latest released versions:

```
pip install \
apache-airflow[postgres,google]==2.0.0 \
--constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.0.0/constraints-3.7.txt"
```

- Just to make it clear - “apache-airflow-providers-postgres” (same with mysql) brings its own dependencies for postgres DB that are not used by Airflow Core when Postgres/Mysql DB is used as Metadata. SQLAlchemy might use any of the available DBapis to communicate with the databases, so it is not related to those providers.
- While we have not done that so far - in the future, we can keep on releasing the “providers” packages separately and individually from the main repository. The release process follows the standard release procedure for Apache Software (voting, signing packages etc.). The process for that was prepared and tested during backport package release and needs just small adjustments. It will require a little different approach for the constraint installation where “master” should be used as constraint sources. We might also introduce “constraints-2.0” or even “constraint-2” moving tag (moving automatically) to serve the same purpose. This is all fully automatable including testing:

```
pip install \
apache-airflow[postgres,google]==2.0.0 \
--constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.0/constraints-3.7.txt"
```

Known Problems and solutions

- There is a known problem with installing a provider's package when you develop Airflow and install ‘airflow’ and ‘providers’ in separate folders. There is a POC implementing solution for that for the future 1.10 and 2.0 releases (https://github.com/apache/airflow/pull/10806)
- We have to make sure that we have no dependencies core -> providers. While we already check and generate cross-provider dependencies, we do not check that core classes depend on any of the provider’s classes. This should be very easy to automate.
- We have to get rid of the current core -> providers dependencies and use a “dependency injection” pattern. We have a working POC of dynamic package discovery: https://github.com/apache/airflow/pull/10822 that implements fast and easy to implement. When completed, it should implement the following:
    - Generic SQL operators: Currently. airflow.models.connection.Connection.get_hook always returns core classes only.
    - Connection form support for dynamic third-party packages.
        - airflow.www.forms._connection_types define a list of connection types, but the third package cannot extend it. This can make it difficult to release new packages regardless of the core. The minimum that needs to be done is to be able to define any value for this field similar to CLI/API. Closely related to https://github.com/apache/airflow/issues/9506
        - airflow.www.forms.ConnectionForm supports extra fields per connection, but the third package cannot add new fields.
        - airflow/www/static/js/connection_form.js defines additional forms behavior (hidden fields, relabelling, placeholders currently).
    - Extra links supported for operators coming from third-party packages- airflow.serialization.serialized_objects.BUILTIN_OPERATOR_EXTRA_LINKS
    - Logging via GCS/AWS/.... providers should work conditionally, based on the provider packages being installed.

Challenges

- In the long-term structure there are many more packages to maintain.
    - Since the packages are distributed separately, they needn't all be released at the same time.
    - Testing of each package will be faster. Someone contributing a fix to a BigQuery operator needn't be concerned with running the tests for the Hive operators. We have an issue that needs to be implemented for it: https://github.com/apache/airflow/issues/9318
    - It will be easier for people with relevant expertise, such as employees and customers of cloud providers to help with maintenance of these packages.
    - It will be easier to create new operators, with the only change needed to core being to create the relevant connection type.
- Airflow will be tightly coupled with many new packages.
    - Releases need not be tightly coupled. For example, pandas split out I/O modules into separate packages, such as pandas-gbq for BigQuery support. pandas-gbq supports several versions of pandas and vice versa. No special considerations are needed when making releases of pandas-gbq or pandas except to document the supported versions.
    - The open requirements approach of Airflow paired with the automated constraint management in monorepo will help to keep the cross-requirements under control.