Using extras in Airflow 2

This document summarizes the use cases for extras in Airflow and is a base for discussion on what should be approached for Airflow Extras in Airflow 3.

This document describes the state of Airflow two extras in Airflow 2 and state of the packaging standard that are related as of 17th of October 2024. The extras approach for Airflow 3 (currently in development in main) is the same as Airflow 2 - but we might discuss changing it and approaching extras differently.

This document does not describe any proposals on how to change it, it merely captures the current state as well as explains the possible future changes resulting from planned features in packaging and tooling that we use that might impact the future choices.

The actual proposals on changing the extras approach should be based on analyzing current use cases and approach, tooling availability and whether we will find suitable replacements for the use cases we have now.

The approach to extras depends very much on the packaging decisions we will make before releasing Airflow 3. Currently we only have one “apache-airflow” package and all extras are defined there. In Airflow 3 we will have more packages (at least 2) and some extras might or might not be available in some of those packages (and different packages might have different extras.

What are extras?

Extras are optional part of Python packaging distribution defined in PEP508:

> *An extra is an optional part of a distribution. Distributions can specify as many extras as they wish, and each extra results in the declaration of additional dependencies of the distribution when the extra is used in a dependency specification. For instance:*
>
> *requests[security]*
>
> *Extras union in the dependencies they define with the dependencies of the distribution they are attached to. The example above would result in requests being installed, and requests own dependencies, and also any dependencies that are listed in the “security” extra of requests.*
>
> *If multiple extras are listed, all the dependencies are unioned together.*

Summary of the current extra types

We are using the following types of extras in Airflow:

Core Airflow extras

These are core airflow extras that extend capabilities of core Airflow. They usually do not install provider packages (with the exception of celery and cncf.kubernetes extras), they just install necessary python dependencies for the provided package.

Example:

```
pip install 'apache-airflow[aiobotocore]'
```

Adds support for asynchronous (deferrable) operators for Amazon integration

Providers extras

These provider extras are simply convenience extras to install provider packages so that you can install the providers with simple command - including the provider package and necessary dependencies in single command, which allows PIP to resolve any conflicting dependencies. This is extremely useful for first time installation where you want to repeatedly install versions of dependencies which are ‘valid’ for both airflow and providers installed.

Example:

```
pip install apache-airflow[google,amazon,apache-spark]==2.10.2 \
  --constraint
"https://raw.githubusercontent.com/apache/airflow/constraints-2.10.2/constr
aints-3.8.txt"
```

Production Bundle extras

These are extras that install one or more extras as a bundle

- pip install 'apache-airflow[all]'
  All Airflow user facing features (no devel and doc requirements)

- pip install 'apache-airflow[all-core]'
  All core airflow features that do not require installing providers

- pip install 'apache-airflow[all-dbs]'

All database integrations

Bundle devel extras

Those are extras that bundle devel, editable and doc extras together to make it easy to install them together in a single installation. Some of the extras are more difficult to install on certain systems (such as ARM MacBooks) because they require system level dependencies to be installed.

```
pip install -e '.[devel]'
```

Minimum development dependencies - minimal, complete test environment

```
pip install -e '.[devel-hadoop]'
```

Adds Hadoop stack libraries devel dependencies

```
pip install -e '.[devel-all-dbs]'
```

Adds all libraries needed to test database providers

- devel-all

```
pip install -e '.[devel-all]'
```

Everything needed for development including Hadoop, all devel extras, all doc extras. Generally: all possible dependencies except providers

```
pip install -e '.[devel-ci]'
```

All dependencies required for CI tests (same as devel-all)

Doc extras

Those extras (“**doc**” and “**doc-gen**” are needed in order to add dependencies needed for documentation building and documentation generation (ERD diagrams mostly).

Editable vs. standard extras

In airflow, extras behave differently depending on whether they are installed in “standard” or “editable” mode. In Python packaging as of PEP 660 the packaging team standardized the approach of installing python packages in “--editable” mode that is intended to install packages locally in order to make it easy to develop those packages. The editable installation install all entry points of the package and “installs package” so that it is available for packaging tools, but the source code is linked to rather than copied, which allows to edit the files “in-place” while developing - this way it allows for faster iteration while developing the packages.

In Airflow the extras behave differently in “standard” installation (i.e. when you install a package from wheels, pypi or locally without –editable flag). This is done via build hooks implemented in hatch_build.py. The main differences come from different needs of installing packages in “production” (standard mode) and “development” (editable mode). The difference is in provider extras, devel bundle extras, doc extras, required dependencies..

Provider extras

Provider extras in standard mode install “provider package”. For example “google” provider extra installs “apache-airflow-providers-google” package. This allows for easy augmentation of the installation of airflow and specifying which optional providers should be installed together with airflow. For example:

```
pip install apache-airflow[google,mysql,amazon]
```

will install airflow with “apache-airflow-providers-google”, “apache-airflow-providers-mysql”, “apache-airflow-providers-amazon”. It is equivalent to

```
pip install apache-airflow apache-airflow-providers-google
apache-airflow-providers-mysql apache-airflow-providers-amazon
```

Editable mode is where providers extras are really useful for developers of providers, because those extras do not install providers, but they install “dependencies” of the providers - including development dependencies of those providers. Some of the providers have additional “devel-only” dependencies that are used in tests (for example one of amazon’s development dependency is “moto” that emulates AWS service for unit tests) so by running this:

```
pip install -e ".[amazon]"
```

Developers can install locally all dependencies (including development dependencies) of amazon provider. There is no need (and it would be actually a bad idea) to install the provider package itself. This makes it very easy to start developing and running tests for any of the 90+ providers airflow has.

This is likely to be mitigated in the future by splitting providers into subdirectories in airflow source code and extensive use of “workspaces” feature of uv where developers will be able to install and synchronize dependencies of airflow + selected providers using `uv` interface.

Bundle devel extras:

The bundle devel extras are only (generally) available in editable mode. They allow the installation of a bundle of dependencies that are likely to be installed together while developing airflow and its providers. For example “devel-hadoop” extra installs all providers and core airflow extras that are necessary for developing hadoop related providers.

Those devel extras are (generally) not available in the “standard” build with the exception of “devel-ci”. The “devel-ci” is used by the “main-tip-installation” caching mechanism that immensely speeds up building and rebuilding of the CI image of airflow - both for the CI purpose and local development with Breeze (thanks to do that in most cases it takes 20-30 seconds usually to rebuild the image - both in CI and local development). This way the following works:

```
pip install "apache-airflow[devel-all] @
git+ssh://git@github.com/apache/airflow.git"
```

We are using it to install airflow with all dependencies including all provider dependencies including all development dependencies of all providers.

Note: It should be possible to get rid of this extra in standard mode likely by slightly more complicating the installation of “main tip” cache - we already download the archive locally for installation, so we could install airflow in editable mode with “[devel-all]” and then uninstall it, to achieve similar effect most likely - then “devel-ci” could be removed from the “standard” mode (and from the wheel package). But it is still needed in editable mode.

Doc extras:

The doc extras are not available in “standard” mode of Airflow installation.

Required dependencies and pre-installed providers

The “required dependencies” for the Airflow core package are generally fixed and the same for “standard” and “editable” dependencies - with the exception of pre-installed providers. Some of the providers are pre=installed - which means that in “standard” installation they are installed as providers, but in “editable” mode they are not installed - instead dependencies of those providers are installed, so that you have the dependencies of those providers installed when you install airflow in editable mode.

```
"common.compat",
"common.io",
"common.sql",
"fab>=1.0.2",
"ftp",
"http",
"imap",
"smtp",
"sqlite",
"standard",
```

Future of development bundle extras (near future)

In the future, we might be able to switch to the new way of defining dependency groups: https://peps.python.org/pep-0735/

This PEP was accepted on October 13, 2024 (so few days before writing this document) and it describes a new accepted way of how dependency groups. The motivations for creating the PEP were very similar to the motivations for all the use cases described in this document, and with introduction of the PEP, we could eventually switch most of the use cases of ours to this new standard. Here is the motivation for the PEP (the PEP contains also use cases explaining how the PEP will address it).

There are two major use cases for which the Python community has no standardized answer:

- *How should development dependencies be defined for packages?*
- *How should dependencies be defined for projects which do not build distributions (non-package projects)?*

In support of these two needs, there are two common solutions which are similar to this proposal:

- *requirements.txt files*
- *package extras*

Both requirements.txt files and extras have limitations which this standard seeks to overcome.

The problem with this PEP is that it is not YET supported by any of the tools that are used for packaging. Pip does not support it yet, While most of the tools are eager to adopt it, some already have done so, others are slower to adopt it. Particularly there is currently no on-going work in `pip` to implement it and given quarterly release schedule of `pip` it will take at least 3-4 months (if not more) to get pip support it - and even then only the latest version of `pip` will support it and it will make airflow unusable for anyone who uses pip. However since Airflow relies more and more on `uv` - with workspaces, syncing of development environments, and given the fact that there is already merged PR to implement it in uv and given the (several-times-a-week) release cycle of uv, it's likely we will have support for it very quickly in uv.

Bundle extras are generally most useful for "editable" installation, so binding our development with uv makes it possible to switch to it in a matter of weeks not months (but then the use cases involving development bundles will not work with `pip` until they release the feature.

Work in progress/done for PEP735:

- open issue in pip https://github.com/pypa/pip/issues/12963
- **Merged(!) PR in uv: https://github.com/astral-sh/uv/pull/8266**
- **Merged(!) PR in tox: https://github.com/tox-dev/tox/issues/3408**
- open issue in pixi: https://github.com/prefix-dev/pixi/issues/2256
- open issue in poetry: https://github.com/python-poetry/poetry/issues/9751

Use cases for extras:

User use-cases

1. Easy installation of airflow + optional features (and providers) + pre-installed providers.

```
pip install 'apache-airflow[aiobotocore,google,amazon,sentry]'
```

Optionally with constraints:

```
pip install 'apache-airflow[aiobotocore,google,amazon,sentry]==2.10.2'
--constraint
"https://raw.githubusercontent.com/apache/airflow/constraints-2.10.2/constrai
nts-3.8.txt"
```

2. Installation of airflow with all production dependencies - this is useful for some of our users who want to test and prepare airflow with all dependencies for all community providers.

```
pip install 'apache-airflow[all]==2.10.2' --constraint
"https://raw.githubusercontent.com/apache/airflow/constraints-2.10.2/constrai
nts-3.8.txt"
```

Development use-cases

1. Installation of local development environment for local development of core airflow - without installing all provider dependencies and without pre-installed providers..

```
pip install -e '.[devel]'
```

2. Installation of local development environment with necessary provider dependencies:

```
pip install -e '.[aiobotocore,google,amazon,sentry]'
```

3. Installation of complete set of dependencies for caching the dependencies and preparing the CI image with ALL (production and development) dependencies

```
pip install -e '.[devel-ci]'
```

4. Installation of development bundles for developers who work on a group of dependencies:

```
pip install -e '.[devel-hadoop]'
```