# Installation



This page describes installations using the `apache-airflow` package published in PyPI, but some information may be useful during installation with other tools as well.

> **Note**
>
> Airflow is also distributed as a Docker image (OCI Image). For more information, see: Production Container Images

## Prerequisites

Airflow is tested with:

- Python: 3.6, 3.7, 3.8
- Databases:
  - PostgreSQL: 9.6, 10, 11, 12, 13
  - MySQL: 5.7, 8
  - SQLite: 3.15.0+
- Kubernetes: 1.16.9, 1.17.5, 1.18.6

**Note:** MySQL 5.x versions are unable to or have limitations with running multiple schedulers – please see: Scheduler. MariaDB is not tested/recommended.

**Note:** SQLite is used in Airflow tests. Do not use it in production. We recommend using the latest stable version of SQLite for local development.

Please note that with respect to Python 3 support, Airflow 2.0.0 has been tested with Python 3.6, 3.7, and 3.8, but does not yet support Python 3.9.

## Installation tools

The official way of installing Airflow is with the `pip` tool. There was a recent (November 2020) change in resolver, so currently only 20.2.4 version is officially supported, although you might have a success with 20.3.3+ version (to be confirmed if all initial issues from `pip` 20.3.0 release have been fixed in 20.3.3). In order to install Airflow you need to either downgrade pip to version 20.2.4 `pip install --upgrade pip==20.2.4` or, in case you use Pip 20.3, you need to add option `--use-deprecated legacy-resolver` to your pip install command.

While they are some successes with using other tools like poetry or pip-tools, they do not share the same workflow as `pip` - especially when it comes to constraint vs. requirements management. Installing via `Poetry` or `pip-tools` is not currently supported. If you wish to install airflow using those tools you should use the constraint files and convert them to appropriate format and workflow that your tool requires.

## Airflow extra dependencies

The `apache-airflow` PyPI basic package only installs what's needed to get started. Additional packages can be installed depending on what will be useful in your environment. For instance, if you don't need connectivity with Postgres, you won't have to go through the trouble of installing the `postgres-devel` yum package, or whatever equivalent applies on the distribution you are using.

Most of the extra dependencies are linked to a corresponding providers package. For example “amazon” extra has a corresponding `apache-airflow-providers-amazon` providers package to be installed. When you install Airflow with such extras, the necessary provider packages are installed automatically (latest versions from PyPI for those packages). However you can freely upgrade and install provider packages independently from the main Airflow installation.

For the list of the extras and what they enable, see: Reference for package extras.

## Provider packages

Unlike Apache Airflow 1.10, the Airflow 2.0 is delivered in multiple, separate, but connected packages. The core of Airflow scheduling system is delivered as `apache-airflow` package and there are around 60 providers packages which can be installed separately as so called `Airflow Provider packages`. The default Airflow installation doesn’t have many integrations and you have to install them yourself.

You can even develop and install your own providers for Airflow. For more information, see: Provider packages

For the list of the provider packages and what they enable, see: Providers packages reference.

## Differences between extras and providers

Just to prevent confusion of extras versus provider packages: Extras and providers are different things, though many extras are leading to installing providers.

Extras are standard Python setuptools feature that allows to add additional set of dependencies as optional features to “core” Apache Airflow. One of the type of such optional features are providers packages, but not all optional features of Apache Airflow have corresponding providers.

We are using the `extras` setuptools features to also install provider packages. Most of the extras are also linked (same name) with provider packages - for example adding `[google]` extra also adds `apache-airflow-providers-google` as dependency. However there are some extras that do not install providers (examples `github_enterprise`, `kerberos`, `async` - they add some extra dependencies which are needed for those `extra` features of Airflow mentioned. The three examples above add respectively github enterprise oauth authentication, kerberos integration or asynchronous workers for gunicorn. None of those have providers, they are just extending Apache Airflow “core” package with new functionalities.

## System dependencies

You need certain system level requirements in order to install Airflow. Those are requirements that are known to be needed for Linux system (Tested on Ubuntu Buster LTS) :

```
sudo apt-get install -y --no-install-recommends \
        freetds-bin \
        krb5-user \
        ldap-utils \
        libffi6 \
        libsasl2-2 \
        libsasl2-modules \
        libssl1.1 \
        locales  \
        lsb-release \
        sasl2-bin \
        sqlite3 \
        unixodbc
```

You also need database client packages (Postgres or MySQL) if you want to use those databases.

## Constraints files

Airflow installation might be sometimes tricky because Airflow is a bit of both a library and application. Libraries usually keep their dependencies open and applications usually pin them, but we should do neither and both at the same time. We decided to keep our dependencies as open as possible (in `setup.cfg` and `setup.py`) so users can install different version of libraries if needed. This means that from time to time plain `pip install apache-airflow` will not work or will produce unusable Airflow installation.

In order to have repeatable installation, starting from **Airflow 1.10.10** and updated in **Airflow 1.10.13** we also keep a set of “known-to-be-working” constraint files in the `constraints-master`, `constraints-2-0` and `constraints-1-10` orphan branches and then we create tag for each released version e.g.

`constraints-2.0.1` . This way, when we keep a tested and working set of dependencies.

Those “known-to-be-working” constraints are per major/minor python version. You can use them as constraint files when installing Airflow from PyPI. Note that you have to specify correct Airflow version and python versions in the URL.

You can create the URL to the file substituting the variables in the template below.

```
https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt
```

where:

- `AIRFLOW_VERSION` - Airflow version (e.g. `2.0.1` ) or `master` , `2-0` , `1-10` for latest development version
- `PYTHON_VERSION` Python version e.g. `3.8` , `3.7`

## Installation script

In order to simplify the installation, we have prepared a script that will select the constraints file compatible with your Python version

**Plain installation:**

If you don’t need to install any extra extra, you can use the command set below:

```
AIRFLOW_VERSION=2.0.1
PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"
# For example: 3.6
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
# For example: https://raw.githubusercontent.com/apache/airflow/constraints-2.0.1/constraints-3.6.txt
pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
```

**Installing with extras**

If you need to install extra dependencies of airflow, you can use the script below (the example below installs postgres and google extras.

```
AIRFLOW_VERSION=2.0.1
PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"
pip install "apache-airflow[postgres,google]==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
```

## Python versions support

As of Airflow 2.0 we agreed to certain rules we follow for Python support. They are based on the official release schedule of Python, nicely summarized in the Python Developer’s Guide

1. We end support for Python versions when they reach EOL (For Python 3.6 it means that we will stop supporting it on 23.12.2021).
2. The “oldest” supported version of Python is the default one. “Default” is only meaningful in terms of “smoke tests” in CI PRs which are run using this default version.
3. We support a new version of Python after it is officially released, as soon as we manage to make it works in our CI pipeline (which might not be immediate) and release a new version of Airflow (non-Patch version) based on this CI set-up.

## Set up a database

Airflow requires a database. If you’re just experimenting and learning Airflow, you can stick with the default SQLite option. If you don’t want to use SQLite, then take a look at Set up a Database Backend to setup a different database.

## Troubleshooting

This section describes how to troubleshoot installation issues.

### Airflow command is not recognized

If the `airflow` command is not getting recognized (can happen on Windows when using WSL), then ensure that `~/.local/bin` is in your `PATH` environment variable, and add it in if necessary:

```
PATH=$PATH:~/.local/bin
```

You can also start airflow with `python -m airflow`

### `Symbol not found: _Py_GetArgcArgv`

If you see `Symbol not found: _Py_GetArgcArgv` while starting or importing Airflow, this may mean that you are using an incompatible version of Python. For a homebrew installed version of Python, this is generally caused by using Python in `/usr/local/opt/bin` rather than the Frameworks installation (e.g. for `python 3.7`: `/usr/local/opt/python@3.7/Frameworks/Python.framework/Versions/3.7`).

The crux of the issue is that a library Airflow depends on, `setproctitle`, uses a non-public Python API which is not available from the standard installation `/usr/local/opt/` (which symlinks to a path under `/usr/local/Cellar`).

An easy fix is just to ensure you use a version of Python that has a dylib of the Python library available. For example:

```
# Note: these instructions are for python3.7 but can be loosely modified for other versions
brew install python@3.7
virtualenv -p /usr/local/opt/python@3.7/Frameworks/Python.framework/Versions/3.7/bin/python3 .toy-venv
source .toy-venv/bin/activate
pip install apache-airflow
python
>>> import setproctitle
# Success!
```

Alternatively, you can download and install Python directly from the Python website.

Previous Next