Content

Running Airflow locally

This quick start guide will help you bootstrap an Airflow standalone instance on your local machine.

! Note

Successful installation requires a Python 3 environment.

Only `pip` installation is currently officially supported.

While there have been 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.

The installation of Airflow is painless if you are following the instructions below. Airflow uses constraint files to enable reproducible installation, so using `pip` and constraint files is recommended.

```
# Airflow needs a home. `~/airflow` is the default, but you can put it
# somewhere else if you prefer (optional)
export AIRFLOW_HOME=~/airflow

# Install Airflow using the constraints file
AIRFLOW_VERSION=2.3.3
PYTHON_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"
# For example: 3.7
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW

# For example: https://raw.githubusercontent.com/apache/airflow/constraints-
2.3.3/constraints-3.7.txt
pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"

# The Standalone command will initialise the database, make a user,
# and start all components for you.
airflow standalone

# Visit localhost:8080 in the browser and use the admin account details
# shown on the terminal to login.
# Enable the example_bash_operator dag in the home page
```

Upon running these commands, Airflow will create the `$AIRFLOW_HOME` folder and create the "airflow.cfg" file with defaults that will get you going fast. You can override defaults using environment variables, see Configuration Reference. You can inspect the file either in `$AIRFLOW_HOME/airflow.cfg`, or through the UI in the `Admin->Configuration` menu. The PID file for the webserver will be stored in `$AIRFLOW_HOME/airflow-webserver.pid` or in `/run/airflow/webserver.pid` if started by systemd.

Out of the box, Airflow uses a SQLite database, which you should outgrow fairly quickly since no parallelization is possible using this database backend. It works in conjunction with the **`SequentialExecutor`** which will only run task instances sequentially. While this is very limiting, it allows you to get up and running quickly and take a tour of the UI and the command line utilities.

As you grow and deploy Airflow to production, you will also want to move away from the `standalone` command we use here to running the components separately. You can read more in Production Deployment.

Here are a few commands that will trigger a few task instances. You should be able to see the status of the jobs change in the `example_bash_operator` DAG as you run the commands below.

```
# run your first task instance
airflow tasks run example_bash_operator runme_0 2015-01-01
# run a backfill over 2 days
airflow dags backfill example_bash_operator \
    --start-date 2015-01-01 \
    --end-date 2015-01-02
```

If you want to run the individual parts of Airflow manually rather than using the all-in-one `standalone` command, you can instead run:

```
airflow db init

airflow users create \
    --username admin \
    --firstname Peter \
    --lastname Parker \
    --role Admin \
    --email spiderman@superhero.org

airflow webserver --port 8080

airflow scheduler
```

What’s Next?

From this point, you can head to the Tutorial section for further examples or the How-to Guides section if you’re ready to get your hands dirty.

Previous Next

Was this entry helpful?

☆ ☆ ☆ ☆ ☆

Want to be a part of Apache Airflow?

Join community

License
Donate
Thanks
Security