Modules Management

Airflow allows you to use your own Python modules in the DAG and in the Airflow configuration. The following article will describe how you can create your own module so that Airflow can load it correctly, as well as diagnose problems when modules are not loaded properly.

Packages Loading in Python

The list of directories from which Python tries to load the module is given by the variable `sys.path`. Python really tries to intelligently determine the contents of of this variable, including depending on the operating system and how Python is installed and which Python version is used.

You can check the contents of this variable for the current Python environment by running an interactive terminal as in the example below:

```
>>> import sys
>>> from pprint import pprint
>>> pprint(sys.path)
['',
 '/home/arch/.pyenv/versions/3.7.4/lib/python37.zip',
 '/home/arch/.pyenv/versions/3.7.4/lib/python3.7',
 '/home/arch/.pyenv/versions/3.7.4/lib/python3.7/lib-dynload',
 '/home/arch/venvs/airflow/lib/python3.7/site-packages']
```

`sys.path` is initialized during program startup. The first precedence is given to the current directory, i.e, `path[0]` is the directory containing the current script that was used to invoke or an empty string in case it was an interactive shell. Second precedence is given to the `PYTHONPATH` if provided, followed by installation-dependent default paths which is managed by site module.

`sys.path` can also be modified during a Python session by simply using append (for example, `sys.path.append("/path/to/custom/package")`). Python will start searching for packages in the newer paths once they're added. Airflow makes use of this feature as described in the section Additional modules in Airflow.

In the variable `sys.path` there is a directory `site-packages` which contains the installed **external packages**, which means you can install packages with `pip` or `anaconda` and you can use them in Airflow. In the next section, you will learn how to create your own simple installable package and how to specify additional directories to be added to `sys.path` using the environment variable `PYTHONPATH`.

Creating a package in Python

1. Before starting, install the following packages:

`setuptools` : setuptools is a package development process library designed for creating and distributing Python packages.

`wheel` : The wheel package provides a bdist_wheel command for setuptools. It creates .whl file which is directly installable through the `pip install` command. We can then upload the same file to PyPI.

```
pip install --upgrade pip setuptools wheel
```

2. Create the package directory - in our case, we will call it `airflow_operators`.

```
mkdir airflow_operators
```

3. Create the file `__init__.py` inside the package and add following code:

```
print("Hello from airflow_operators")
```

When we import this package, it should print the above message.

4. Create `setup.py` :

```
import setuptools

setuptools.setup(
    name='airflow_operators',
)
```

5. Build the wheel:

```
python setup.py bdist_wheel
```

This will create a few directories in the project and the overall structure will look like following:

```
.
├── airflow_operators
│   ├── __init__.py
├── airflow_operators.egg-info
│   ├── PKG-INFO
│   ├── SOURCES.txt
│   ├── dependency_links.txt
│   └── top_level.txt
├── build
│   └── bdist.macosx-10.15-x86_64
├── dist
│   └── airflow_operators-0.0.0-py3-none-any.whl
└── setup.py
```

6. Install the .whl file using pip:

```
pip install dist/airflow_operators-0.0.0-py3-none-any.whl
```

7. The package is now ready to use!

```
>>> import airflow_operators
Hello from airflow_operators
>>>
```

The package can be removed using pip command:

```
pip uninstall airflow_operators
```

For more details on how to create to create and publish python packages, see Packaging Python Projects.

Adding directories to the path

You can specify additional directories to be added to `sys.path` using the environment variable **`PYTHONPATH`** . Start the python shell by providing the path to root of your project using the following command:

```
PYTHONPATH=/home/arch/projects/airflow_operators python
```

The `sys.path` variable will look like below:

```
>>> import sys
>>> from pprint import pprint
>>> pprint(sys.path)
['',
 '/home/arch/projects/airflow_operators'
 '/home/arch/.pyenv/versions/3.7.4/lib/python37.zip',
 '/home/arch/.pyenv/versions/3.7.4/lib/python3.7',
 '/home/arch/.pyenv/versions/3.7.4/lib/python3.7/lib-dynload',
 '/home/arch/venvs/airflow/lib/python3.7/site-packages']
```

As we can see that our provided directory is now added to the path, let's try to import the package now:

```
>>> import airflow_operators
Hello from airflow_operators
>>>
```

We can also use **`PYTHONPATH`** variable with the airflow commands. For example, if we run the following airflow command:

```
PYTHONPATH=/home/arch/projects/airflow_operators airflow info
```

We'll see the `Python PATH` updated with our mentioned **`PYTHONPATH`** value as shown below:

```
Python PATH: [/home/arch/venv/bin:/home/arch/projects/airflow_operators:/usr/lib/python38.zip:/usr/lib/python3.8:/usr/lib/python3.8/lib
```

Additional modules in Airflow

Airflow adds three additional directories to the `sys.path` :

- DAGS folder: It is configured with option `dags_folder` in section `[core]` .
- Config folder: It is configured by setting `AIRFLOW_HOME` variable (`{AIRFLOW_HOME}/config`) by default.
- Plugins Folder: It is configured with option `plugins_folder` in section `[core]` .

You can also see the exact paths using the `airflow info` command, and use them similar to directories specified with the environment variable **`PYTHONPATH`** . An example of the contents of the sys.path variable specified by this command may be as follows:

Python PATH: [/home/rootcss/venvs/airflow/bin:/usr/lib/python38.zip:/usr/lib/python3.8:/usr/lib/python3.8/lib-dynload:/home/rootcss/venvs/airflow/lib/python3.8/site-packages:/home/rootcss/airflow/dags:/home/rootcss/airflow/config:/home/rootcss/airflow/plugins]

Below is the sample output of the `airflow info` command:

See also

When are plugins (re)loaded?

```
Apache Airflow: 2.0.0b3

System info
OS              | Linux
architecture    | x86_64
uname           | uname_result(system='Linux', node='85cd7ab7018e', release='4.19.76-linuxkit', version='#1 SMP Tue May 26 11:42:35 UTC
locale          | ('en_US', 'UTF-8')
python_version  | 3.8.6 (default, Nov 25 2020, 02:47:44)  [GCC 8.3.0]
python_location | /usr/local/bin/python

Tools info
git             | git version 2.20.1
ssh             | OpenSSH_7.9p1 Debian-10+deb10u2, OpenSSL 1.1.1d  10 Sep 2019
kubectl         | NOT AVAILABLE
gcloud          | NOT AVAILABLE
cloud_sql_proxy | NOT AVAILABLE
mysql           | mysql  Ver 8.0.22 for Linux on x86_64 (MySQL Community Server - GPL)
sqlite3         | 3.27.2 2019-02-25 16:06:06 bd49a8271d650fa89e446b42e513b595a717b9212c91dd384aab871fc1d0alt1
psql            | psql (PostgreSQL) 11.9 (Debian 11.9-0+deb10u1)

Paths info
airflow_home    | /root/airflow
system_path     | /opt/bats/bin:/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
python_path     | /usr/local/bin:/opt/airflow:/files/plugins:/usr/local/lib/python38.zip:/usr/local/lib/python3.8:/usr/
                | local/lib/python3.8/lib-dynload:/usr/local/lib/python3.8/site-packages:/files/dags:/root/airflow/conf
                | ig:/root/airflow/plugins
airflow_on_path | True

Config info
executor             | LocalExecutor
task_logging_handler | airflow.utils.log.file_task_handler.FileTaskHandler
sql_alchemy_conn     | postgresql+psycopg2://postgres:airflow@postgres/airflow
dags_folder          | /files/dags
plugins_folder       | /root/airflow/plugins
base_log_folder      | /root/airflow/logs

Providers info
apache-airflow-providers-amazon           | 1.0.0b2
apache-airflow-providers-apache-cassandra | 1.0.0b2
apache-airflow-providers-apache-druid     | 1.0.0b2
apache-airflow-providers-apache-hdfs      | 1.0.0b2
apache-airflow-providers-apache-hive      | 1.0.0b2
```

Previous Next

Was this entry helpful?

☆ ☆ ☆ ☆ ☆