Airflow Plugins Documentation

Release 0.1.0

Michael Kuty

Jan 18, 2018

Contents

Contents:

CHAPTER 1

Airflow Plugins

Airflow plugins.

- Free software: MIT license
- Documentation: https://airflow-plugins.readthedocs.io.

Features

- Database operations
- Slack operations
- ZIP operations
- Git operations
- File operations
- File sensors
- Cookiecutter operations
- Airflow variables utils

CHAPTER 2

Installation

Stable release

To install Airflow Plugins, run this command in your terminal:

```
$ pip install airflow-plugins
```

This is the preferred method to install Airflow Plugins, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Airflow Plugins can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/storiesbi/airflow-plugins
```

Or download the tarball:

```
$ curl  -OL https://github.com/storiesbi/airflow-plugins/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Airflow Plugins

Base operators

class airflow_plugins.operators.base.**BashOperator**(*bash_command=None*, **args*, ***kwargs*)

Bash Operator

class airflow_plugins.operators.base.**ExecutableOperator**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Simple wrapper around command line executable programs with helper functions to add options, flags and arguments.

add_flag(*flag_name*)

Add boolean flag option used as enabled or disabled state

add_option(*option_name*, *value*)
Add option to command

class airflow_plugins.operators.base.**PostgresOperator**(*sql=None*, **args*, ***kwargs*)
Run SQL on Postgresql based systems.

Database

class airflow_plugins.operators.db.**ChangeDatabaseName**(*sql=None*, **args*, ***kwargs*)
Rename database in operator.

class airflow_plugins.operators.db.**CreateDatabase**(*sql=None*, **args*, ***kwargs*)
Operator which creates database in PostgreSQL.

class airflow_plugins.operators.db.**CreateTableWithColumns**(**args*, ***kwargs*)
Create database with columns.

class airflow_plugins.operators.db.**DropDatabase**(*sql=None*, **args*, ***kwargs*)
Drop database operator.

class airflow_plugins.operators.db.**PostgresHook**(*database=None*, *fail_silently=False*, **args*, ***kwargs*)
Tuned PostgreSQL hook which support running SQL like create database. Supports silent fail.

run(*sql*, *autocommit=False*, *parameters=None*)
Runs a command or a list of commands. Pass a list of sql statements to the sql parameter to get them to execute sequentially

Parameters

- **sql** (*str or list*) – the sql statement to be executed (str) or a list of sql statements to execute
- **autocommit** (*bool*) – What to set the connection's autocommit setting to before executing the query.
- **parameters** (*mapping or iterable*) – The parameters to render the SQL query with.

class airflow_plugins.operators.db.**PostgresOperator**(*database=None*, *fail_silently=True*, **args*, ***kwargs*)
PostgreSQL operator which uses PostgresHook

Files

class airflow_plugins.operators.files.**DeleteFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Delete file operator.

class airflow_plugins.operators.files.**DownloadFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Download file operator.

class airflow_plugins.operators.files.**DynamicDeleteFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Dynamic delete file operator.

class airflow_plugins.operators.files.**DynamicDownloadFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Dynamic download file operator.

class airflow_plugins.operators.files.**DynamicTargetFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Dynamic target file operator

class airflow_plugins.operators.files.**DynamicUploadFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Dynamic upload file operator.

class airflow_plugins.operators.files.**UploadFile**(*task_id*, *owner='Airflow'*, *email=None*, *email_on_retry=True*, *email_on_failure=True*, *retries=0*, *retry_delay=datetime.timedelta(0, 300)*, *retry_exponential_backoff=False*, *max_retry_delay=None*, *start_date=None*, *end_date=None*, *schedule_interval=None*, *depends_on_past=False*, *wait_for_downstream=False*, *dag=None*, *params=None*, *default_args=None*, *adhoc=False*, *priority_weight=1*, *queue='default'*, *pool=None*, *sla=None*, *execution_timeout=None*, *on_failure_callback=None*, *on_success_callback=None*, *on_retry_callback=None*, *trigger_rule='all_success'*, *resources=None*, *run_as_user=None*, **args*, ***kwargs*)

Upload file operator.

CSV

class airflow_plugins.operators.csv.**CSVLook**(*bash_command=None*, **args*, ***kwargs*)
Get stats of the CSV file

class airflow_plugins.operators.csv.**CSVSQL**(*bash_command=None*, **args*, ***kwargs*)
Use csvsql tool for migration CSV to SQL. For more parameters check csvsql.

class airflow_plugins.operators.csv.**CSVStats**(*bash_command=None*, **args*, ***kwargs*)
Get stats of the CSV file Use csvstat.

class airflow_plugins.operators.csv.**CSVtoDB**(*bash_command=None*, **args*, ***kwargs*)
Use csvsql tool for migration csv to SQL database. For more parameters check csvsql.

class airflow_plugins.operators.csv.**SplitCSVtoDB**(*bash_command=None*, **args*, ***kwargs*)
Split CSV and upload to DB.

ZIP

class airflow_plugins.operators.zip.**UnzipOperator**(*path_to_zip_file=None*, *path_to_zip_folder=None*, *path_to_zip_folder_pattern='*.zip'*, *path_to_unzip_contents=None*, **args*, ***kwargs*)
An operator which takes in a path to a zip file and unzips the contents to a location you define.

Parameters

- **path_to_zip_file** (*string*) – Full path to the zip file you want to Unzip
- **path_to_unzip_contents** – Full path to

where you want to save the contents of the Zip file you're Unzipping :type path_to_unzip_contents: string

class airflow_plugins.operators.zip.**ZipOperator**(*path_to_file_to_zip*, *path_to_save_zip*, **args*, ***kwargs*)
An operator which takes in a path to a file and zips the contents to a location you define.

Parameters

- **path_to_file_to_zip** (*string*) – Full path to the file you want to Zip
- **path_to_save_zip** (*string*) – Full path to where you want to save the Zip file

Git

class airflow_plugins.operators.git.**GitClone**(**args*, ***kwargs*)
Git clone operator.

class airflow_plugins.operators.git.**GitCommit**(**args*, ***kwargs*)
Git commit operator.

class airflow_plugins.operators.git.**GitOperator**(**args*, ***kwargs*)
Base Git operator.

class airflow_plugins.operators.git.**GitPush**(**args*, ***kwargs*)
Git push operator.

Slack

class airflow_plugins.operators.slack.hooks.**SlackHook**(*token=None*, *method='chat.postMessage'*, *api_params=None*, *channel=None*, *username=None*, *text=None*, *attachments=None*, **args*, ***kwargs*)

Slack hook

get_channel_id(*name*)
Returns channel id by name

get_file_content(*url*)
Returns file content

run(***kwargs*)
SlackAPIOperator calls will not fail even if the call is not unsuccessful. It should not prevent a DAG from completing in success.

class airflow_plugins.operators.slack.operators.**Message**(*channel=None*, *username=None*, **args*, ***kwargs*)

Slack message operator

class airflow_plugins.operators.slack.sensors.**SlackMessageSensor**(*channel*, *username=None*, *text_contains=None*, *callback=None*, *params=None*, *headers=None*, *extra_options=None*, **args*, ***kwargs*)

Executes a HTTP get statement and returns False on failure: 404 not found or response_check function returned False

Parameters

- **http_conn_id** (*string*) – The connection to run the sensor against
- **endpoint** (*string*) – The relative part of the full url
- **params** (*a dictionary of string key/value pairs*) – The parameters to be added to the GET url
- **headers** (*a dictionary of string key/value pairs*) – The HTTP headers to be added to the GET request
- **response_check** (*A lambda or defined function.*) – A check against the 'requests' response object. Returns True for 'pass' and False otherwise.
- **extra_options** (*A dictionary of options, where key is string and value depends on the option that's being modified.*) – Extra options for the 'requests' library, see the 'requests' documentation (options to modify timeout, ssl, etc.)

File Sensors

class airflow_plugins.operators.sensors.file_sensor.**FileSensor**(*path*, *modified=None*, *notify_after=28800*, *notify_delta=3600*, *conn_id=None*, **args*, ***kwargs*)

Check file presence on hook

class airflow_plugins.operators.sensors.task_sensor.**TaskRuntimeSensor**(*notify_after*, *notify_delta=3600*, *start_wait=0*, *dag_ids=None*, *task_ids=None*, *operator_ids=None*, *include_subdags=True*, *check_execution_time=True*, **args*, ***kwargs*)

Checks whether particular tasks are still running after a period of time and notify about them if so

Parameters

- **notify_after** (*int (or timedelta)*) – Start sending notifications after given number of seconds (of runtime)
- **notify_delta** (*int (or timedelta)*) – Time interval between successive notifications in seconds, defaults to one hour (60*60 seconds)
- **start_wait** (*int (or timedelta)*) – Wait at start for at least given number of seconds for tasks to be registered (set if this op runs continuously)
- **dag_ids** (*list*) – List of dag_ids determining target task instances, can be set as a mask (e.g. "kiwi_master" for all kiwi master dags)
- **task_ids** (*list*) – List of task_ids determining target task instances
- **operator_ids** (*list*) – List of operators determining target task instances
- **include_subdags** (*bool*) – Whether to include subdags of target dags (dag_ids) (i.e. "kiwi_master" to also match "kiwi_master.storyteller" tasks), default True (always True if dag_ids not set)
- **check_execution_time** (*bool*) – Whether to check task instance execution time, or wall clock time (time elapsed from midnight), default True

Utils

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/storiesbi/airflow-plugins/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

Airflow Plugins could always use more documentation, whether as part of the official Airflow Plugins docs, in doc-strings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/storiesbi/airflow-plugins/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *airflow_plugins* for local development.

1. Fork the *airflow_plugins* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/airflow-plugins.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv airflow-plugins
$ cd airflow-plugins/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 airflow-plugins tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/storiesbi/airflow_plugins/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_airflow_plugins
```

CHAPTER 5

History

0.1.0 (2018-01-18)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

a