

---

# **Airflow Documentation**

***Release 2.0.0.dev0+***

**Apache Airflow**

**Feb 08, 2019**



<b>1</b>	<b>Principles</b>	<b>3</b>
<b>2</b>	<b>Beyond the Horizon</b>	<b>5</b>
<b>3</b>	<b>Content</b>	<b>7</b>
3.1	Project . . . . .	7
3.1.1	History . . . . .	7
3.1.2	Committers . . . . .	7
3.1.3	Resources & links . . . . .	8
3.1.4	Roadmap . . . . .	8
3.2	License . . . . .	8
3.3	Quick Start . . . . .	11
3.3.1	What's Next? . . . . .	12
3.4	Installation . . . . .	12
3.4.1	Getting Airflow . . . . .	12
3.4.2	Extra Packages . . . . .	12
3.4.3	Initiating Airflow Database . . . . .	13
3.5	Tutorial . . . . .	13
3.5.1	Example Pipeline definition . . . . .	14
3.5.2	It's a DAG definition file . . . . .	15
3.5.3	Importing Modules . . . . .	15
3.5.4	Default Arguments . . . . .	15
3.5.5	Instantiate a DAG . . . . .	16
3.5.6	Tasks . . . . .	16
3.5.7	Templating with Jinja . . . . .	16
3.5.8	Setting up Dependencies . . . . .	17
3.5.9	Recap . . . . .	18
3.5.10	Testing . . . . .	19
3.5.10.1	Running the Script . . . . .	19
3.5.10.2	Command Line Metadata Validation . . . . .	19
3.5.10.3	Testing . . . . .	19
3.5.10.4	Backfill . . . . .	20
3.5.11	What's Next? . . . . .	20
3.6	How-to Guides . . . . .	20
3.6.1	Add a new role in RBAC UI . . . . .	21
3.6.2	Setting Configuration Options . . . . .	21
3.6.3	Initializing a Database Backend . . . . .	22

3.6.4	Using Operators . . . . .	23
3.6.4.1	BashOperator . . . . .	27
3.6.4.2	PythonOperator . . . . .	28
3.6.4.3	Google Cloud Storage Operators . . . . .	29
3.6.4.4	Google Compute Engine Operators . . . . .	29
3.6.4.5	Google Cloud Bigtable Operators . . . . .	34
3.6.4.6	Google Cloud Functions Operators . . . . .	38
3.6.4.7	Google Cloud Spanner Operators . . . . .	41
3.6.4.8	Google Cloud Sql Operators . . . . .	46
3.6.4.9	Google Cloud Storage Operators . . . . .	61
3.6.5	Managing Connections . . . . .	63
3.6.5.1	Creating a Connection with the UI . . . . .	63
3.6.5.2	Editing a Connection with the UI . . . . .	64
3.6.5.3	Creating a Connection with Environment Variables . . . . .	65
3.6.5.4	Connection Types . . . . .	65
3.6.6	Securing Connections . . . . .	71
3.6.7	Rotating encryption keys . . . . .	71
3.6.8	Writing Logs . . . . .	72
3.6.8.1	Writing Logs Locally . . . . .	72
3.6.8.2	Writing Logs to Amazon S3 . . . . .	72
3.6.8.3	Writing Logs to Azure Blob Storage . . . . .	72
3.6.8.4	Writing Logs to Google Cloud Storage . . . . .	73
3.6.9	Scaling Out with Celery . . . . .	74
3.6.10	Scaling Out with Dask . . . . .	75
3.6.11	Scaling Out with Mesos (community contributed) . . . . .	75
3.6.11.1	Tasks executed directly on mesos slaves . . . . .	75
3.6.11.2	Tasks executed in containers on mesos slaves . . . . .	76
3.6.12	Running Airflow behind a reverse proxy . . . . .	76
3.6.13	Running Airflow with systemd . . . . .	77
3.6.14	Running Airflow with upstart . . . . .	78
3.6.15	Using the Test Mode Configuration . . . . .	78
3.6.16	Checking Airflow Health Status . . . . .	78
3.7	UI / Screenshots . . . . .	79
3.7.1	DAGs View . . . . .	79
3.7.2	Tree View . . . . .	79
3.7.3	Graph View . . . . .	80
3.7.4	Variable View . . . . .	80
3.7.5	Gantt Chart . . . . .	81
3.7.6	Task Duration . . . . .	82
3.7.7	Code View . . . . .	82
3.7.8	Task Instance Context Menu . . . . .	83
3.8	Concepts . . . . .	83
3.8.1	Core Ideas . . . . .	84
3.8.1.1	DAGs . . . . .	84
3.8.1.2	Operators . . . . .	85
3.8.1.3	Tasks . . . . .	87
3.8.1.4	Task Instances . . . . .	87
3.8.1.5	Workflows . . . . .	87
3.8.2	Additional Functionality . . . . .	88
3.8.2.1	Hooks . . . . .	88
3.8.2.2	Pools . . . . .	88
3.8.2.3	Connections . . . . .	88
3.8.2.4	Queues . . . . .	89
3.8.2.5	XComs . . . . .	89

3.8.2.6	Variables	90
3.8.2.7	Branching	90
3.8.2.8	SubDAGs	91
3.8.2.9	SLAs	94
3.8.2.10	Trigger Rules	94
3.8.2.11	Latest Run Only	95
3.8.2.12	Zombies & Undeads	96
3.8.2.13	Cluster Policy	96
3.8.2.14	Documentation & Notes	96
3.8.2.15	Jinja Templating	97
3.8.3	Packaged dags	97
3.8.4	.airflowignore	98
3.9	Command Line Interface	99
3.9.1	Positional Arguments	99
3.9.2	Sub-commands:	99
3.9.2.1	resetdb	99
3.9.2.2	render	99
3.9.2.3	variables	100
3.9.2.4	connections	100
3.9.2.5	rotate_fernet_key	101
3.9.2.6	users	101
3.9.2.7	pause	102
3.9.2.8	sync_perm	103
3.9.2.9	task_failed_deps	103
3.9.2.10	version	103
3.9.2.11	trigger_dag	103
3.9.2.12	initdb	104
3.9.2.13	test	104
3.9.2.14	unpause	104
3.9.2.15	list_dag_runs	105
3.9.2.16	dag_state	105
3.9.2.17	run	106
3.9.2.18	list_tasks	107
3.9.2.19	backfill	107
3.9.2.20	list_dags	109
3.9.2.21	kerberos	109
3.9.2.22	worker	109
3.9.2.23	webserver	110
3.9.2.24	flower	111
3.9.2.25	scheduler	112
3.9.2.26	task_state	112
3.9.2.27	pool	113
3.9.2.28	roles	113
3.9.2.29	list_jobs	113
3.9.2.30	serve_logs	114
3.9.2.31	clear	114
3.9.2.32	next_execution	115
3.9.2.33	upgradedb	115
3.9.2.34	delete_dag	115
3.10	Scheduling & Triggers	116
3.10.1	DAG Runs	116
3.10.2	Backfill and Catchup	117
3.10.3	External Triggers	118
3.10.4	To Keep in Mind	118

3.11	Plugins	118
3.11.1	What for?	118
3.11.2	Why build on top of Airflow?	119
3.11.3	Interface	119
3.11.4	Example	120
3.11.5	Note on role based views	122
3.11.6	Plugins as Python packages	122
3.12	Security	123
3.12.1	Reporting Vulnerabilities	123
3.12.2	Web Authentication	123
3.12.2.1	Password	123
3.12.2.2	LDAP	123
3.12.2.3	Roll your own	124
3.12.3	Multi-tenancy	124
3.12.4	Kerberos	124
3.12.4.1	Limitations	125
3.12.4.2	Enabling kerberos	125
3.12.4.3	Using kerberos authentication	126
3.12.5	OAuth Authentication	126
3.12.5.1	GitHub Enterprise (GHE) Authentication	126
3.12.5.2	Google Authentication	127
3.12.6	SSL	128
3.12.7	Impersonation	128
3.12.7.1	Default Impersonation	129
3.12.8	Flower Authentication	129
3.12.9	RBAC UI Security	129
3.12.9.1	Default Roles	129
3.12.9.2	Custom Roles	131
3.13	Time zones	132
3.13.1	Concepts	132
3.13.1.1	Naïve and aware datetime objects	132
3.13.1.2	Interpretation of naive datetime objects	132
3.13.1.3	Default time zone	133
3.13.2	Time zone aware DAGs	133
3.13.2.1	Templates	133
3.13.2.2	Cron schedules	134
3.13.2.3	Time deltas	134
3.14	Experimental Rest API	134
3.14.1	Endpoints	134
3.14.2	CLI	135
3.14.3	Authentication	135
3.15	Integration	136
3.15.1	Azure: Microsoft Azure	136
3.15.1.1	Azure Blob Storage	136
3.15.1.2	Azure File Share	136
3.15.1.3	Logging	136
3.15.1.4	Azure CosmosDB	137
3.15.1.5	Azure Data Lake	137
3.15.1.6	Azure Container Instances	137
3.15.2	AWS: Amazon Web Services	137
3.15.2.1	AWS EMR	138
3.15.2.2	AWS S3	138
3.15.2.3	AWS Batch Service	138
3.15.2.4	AWS RedShift	138

3.15.2.5	AWS DynamoDB	138
3.15.2.6	AWS Lambda	139
3.15.2.7	AWS Kinesis	139
3.15.2.8	Amazon SageMaker	139
3.15.3	Databricks	139
3.15.4	GCP: Google Cloud Platform	139
3.15.4.1	Logging	139
3.15.4.2	GoogleCloudBaseHook	140
3.15.4.3	BigQuery	140
3.15.4.4	Cloud Spanner	140
3.15.4.5	Cloud SQL	141
3.15.4.6	Cloud Bigtable	141
3.15.4.7	Compute Engine	142
3.15.4.8	Cloud Functions	142
3.15.4.9	Cloud DataFlow	142
3.15.4.10	Cloud DataProc	142
3.15.4.11	Cloud Datastore	143
3.15.4.12	Cloud ML Engine	143
3.15.4.13	Cloud Storage	144
3.15.4.14	Transfer Service	144
3.15.4.15	Google Kubernetes Engine	144
3.15.5	Qubole	144
3.16	Metrics	145
3.16.1	Configuration	145
3.16.2	Counters	145
3.16.3	Gauges	145
3.16.4	Timers	146
3.17	Kubernetes	146
3.17.1	Kubernetes Executor	146
3.17.2	Kubernetes Operator	146
3.18	Lineage	149
3.18.1	Apache Atlas	150
3.19	Changelog	150
3.19.1	Airflow 1.10.2, 2019-01-19	150
3.19.1.1	New features	150
3.19.1.2	Improvements	151
3.19.1.3	Bug fixes	154
3.19.1.4	Doc-only changes	156
3.19.2	Airflow 1.10.1, 2018-11-13	158
3.19.2.1	New features	158
3.19.2.2	Improvements	158
3.19.2.3	Doc-only changes	159
3.19.2.4	Bug fixes	160
3.19.3	Airflow 1.10.0, 2018-08-03	161
3.19.4	Airflow 1.9.0, 2018-01-02	183
3.19.5	Airflow 1.8.2, 2017-09-04	196
3.19.6	Airflow 1.8.1, 2017-05-09	197
3.19.7	Airflow 1.8.0, 2017-03-12	198
3.19.8	Airflow 1.7.2	204
3.19.9	Airflow 1.7.1, 2016-05-19	208
3.20	FAQ	214
3.20.1	Why isn't my task getting scheduled?	214
3.20.2	How do I trigger tasks based on another task's failure?	215

3.20.3	Why are connection passwords still not encrypted in the metadata db after I installed airflow[crypto]? . . . . .	215
3.20.4	What's the deal with <code>start_date</code> ? . . . . .	215
3.20.5	How can I create DAGs dynamically? . . . . .	216
3.20.6	What are all the <code>airflow run</code> commands in my process list? . . . . .	216
3.20.7	How can my airflow dag run faster? . . . . .	216
3.20.8	How can we reduce the airflow UI page load time? . . . . .	217
3.20.9	How to fix Exception: Global variable <code>explicit_defaults_for_timestamp</code> needs to be on (1)? .	217
3.20.10	How to reduce airflow dag scheduling latency in production? . . . . .	217
3.21	API Reference . . . . .	217
3.21.1	Operators . . . . .	217
3.21.1.1	BaseOperator . . . . .	217
3.21.1.2	BaseSensorOperator . . . . .	221
3.21.1.3	Core Operators . . . . .	222
3.21.1.4	Community-contributed Operators . . . . .	242
3.21.2	Macros . . . . .	332
3.21.2.1	Default Variables . . . . .	332
3.21.2.2	Macros . . . . .	333
3.21.3	Models . . . . .	334
3.21.4	Hooks . . . . .	349
3.21.4.1	Community contributed hooks . . . . .	365
3.21.5	Executors . . . . .	428
3.21.5.1	Community-contributed executors . . . . .	430
	<b>HTTP Routing Table</b>	<b>431</b>
	<b>Python Module Index</b>	<b>433</b>





Airflow is a platform to programmatically author, schedule and monitor workflows.

Use airflow to author workflows as directed acyclic graphs (DAGs) of tasks. The airflow scheduler executes your tasks on an array of workers while following the specified dependencies. Rich command line utilities make performing complex surgeries on DAGs a snap. The rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed.

When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative.

---



# CHAPTER 1

---

## Principles

---

- **Dynamic:** Airflow pipelines are configuration as code (Python), allowing for dynamic pipeline generation. This allows for writing code that instantiates pipelines dynamically.
- **Extensible:** Easily define your own operators, executors and extend the library so that it fits the level of abstraction that suits your environment.
- **Elegant:** Airflow pipelines are lean and explicit. Parameterizing your scripts is built into the core of Airflow using the powerful **Jinja** templating engine.
- **Scalable:** Airflow has a modular architecture and uses a message queue to orchestrate an arbitrary number of workers. Airflow is ready to scale to infinity.



## CHAPTER 2

---

### Beyond the Horizon

---

Airflow **is not** a data streaming solution. Tasks do not move data from one to the other (though tasks can exchange metadata!). Airflow is not in the [Spark Streaming](#) or [Storm](#) space, it is more comparable to [Oozie](#) or [Azkaban](#).

Workflows are expected to be mostly static or slowly changing. You can think of the structure of the tasks in your workflow as slightly more dynamic than a database structure would be. Airflow workflows are expected to look similar from a run to the next, this allows for clarity around unit of work and continuity.



## 3.1 Project

### 3.1.1 History

Airflow was started in October 2014 by Maxime Beauchemin at Airbnb. It was open source from the very first commit and officially brought under the Airbnb Github and announced in June 2015.

The project joined the Apache Software Foundation's incubation program in March 2016.

### 3.1.2 Committers

- @mistercrunch (Maxime “Max” Beauchemin)
- @r39132 (Siddharth “Sid” Anand)
- @criccomini (Chris Riccomini)
- @bolkedebuin (Bolke de Bruin)
- @artwr (Arthur Wiedmer)
- @jlowin (Jeremiah Lowin)
- @aoen (Dan Davydov)
- @msumit (Sumit Maheshwari)
- @alexvanboxel (Alex Van Boxel)
- @saguziel (Alex Guziel)
- @joygao (Joy Gao)
- @fokko (Fokko Driesprong)
- @ash (Ash Berlin-Taylor)

- @kaxilnaik (Kaxil Naik)
- @feng-tao (Tao Feng)
- @hiteshs (Hitesh Shah)
- @jghoman (Jakob Homan)

For the full list of contributors, take a look at [Airflow's Github Contributor page](#):

### 3.1.3 Resources & links

- [Airflow's official documentation](#)
- Mailing list (send emails to [dev-subscribe@airflow.apache.org](mailto:dev-subscribe@airflow.apache.org) and/or [commits-subscribe@airflow.apache.org](mailto:commits-subscribe@airflow.apache.org) to subscribe to each)
- Issues on Apache's Jira
- Slack (chat) Channel
- [More resources and links to Airflow related content on the Wiki](#)

### 3.1.4 Roadmap

Please refer to the Roadmap on [the wiki](#)

## 3.2 License



Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

#### 1. Definitions.

"License" shall mean the terms **and** conditions **for** use, reproduction, **and** distribution **as** defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner **or** entity authorized by the copyright owner that **is** granting the License.

"Legal Entity" shall mean the union of the acting entity **and** all other entities that control, are controlled by, **or** are under common control **with** that entity. For the purposes of this definition, "control" means (i) the power, direct **or** indirect, to cause the direction **or** management of such entity, whether by contract **or** otherwise, **or** (ii) ownership of fifty percent (50%) **or** more of the

(continues on next page)



(continued from previous page)

outstanding shares, **or** (iii) beneficial ownership of such entity.

"You" (**or** "Your") shall mean an individual **or** Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form **for** making modifications, including but **not** limited to software source code, documentation source, **and** configuration files.

"Object" form shall mean **any** form resulting **from mechanical** transformation **or** translation of a Source form, including but **not** limited to compiled **object** code, generated documentation, **and** conversions to other media types.

"Work" shall mean the work of authorship, whether **in** Source **or** Object form, made available under the License, **as** indicated by a copyright notice that **is** included **in or** attached to the work (an example **is** provided **in** the Appendix below).

"Derivative Works" shall mean **any** work, whether **in** Source **or** Object form, that **is** based on (**or** derived from) the Work **and for** which the editorial revisions, annotations, elaborations, **or** other modifications represent, **as** a whole, an original work of authorship. For the purposes of this License, Derivative Works shall **not** include works that remain separable from, **or** merely link (**or** bind by name) to the interfaces of, the Work **and** Derivative Works thereof.

"Contribution" shall mean **any** work of authorship, including the original version of the Work **and any** modifications **or** additions to that Work **or** Derivative Works thereof, that **is** intentionally submitted to Licensor **for** inclusion **in** the Work by the copyright owner **or** by an individual **or** Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means **any** form of electronic, verbal, **or** written communication sent to the Licensor **or** its representatives, including but **not** limited to communication on electronic mailing lists, source code control systems, **and** issue tracking systems that are managed by, **or** on behalf of, the Licensor **for** the purpose of discussing **and** improving the Work, but excluding communication that **is** conspicuously marked **or** otherwise designated **in** writing by the copyright owner **as** "Not a Contribution."

"Contributor" shall mean Licensor **and any** individual **or** Legal Entity on behalf of whom a Contribution has been received by Licensor **and** subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms **and** conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, **and** distribute the Work **and** such Derivative Works **in** Source **or** Object form.
3. Grant of Patent License. Subject to the terms **and** conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (**except as** stated **in** this section) patent license to make, have made, use, offer to sell, sell, import, **and** otherwise transfer the Work,

(continues on next page)

(continued from previous page)

where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone **or** by combination of their Contribution(s) **with** the Work to which such Contribution(s) was submitted. If You institute patent litigation against **any** entity (including a cross-claim **or** counterclaim **in** a lawsuit) alleging that the Work **or** a Contribution incorporated within the Work constitutes direct **or** contributory patent infringement, then **any** patent licenses granted to You under this License **for** that Work shall terminate **as** of the date such litigation **is** filed.

4. Redistribution. You may reproduce **and** distribute copies of the Work **or** Derivative Works thereof **in any** medium, **with or** without modifications, **and in** Source **or** Object form, provided that You meet the following conditions:
  - (a) You must give **any** other recipients of the Work **or** Derivative Works a copy of this License; **and**
  - (b) You must cause **any** modified files to carry prominent notices stating that You changed the files; **and**
  - (c) You must retain, **in** the Source form of **any** Derivative Works that You distribute, **all** copyright, patent, trademark, **and** attribution notices **from the** Source form of the Work, excluding those notices that do **not** pertain to **any** part of the Derivative Works; **and**
  - (d) If the Work includes a "NOTICE" text file **as** part of its distribution, then **any** Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do **not** pertain to **any** part of the Derivative Works, **in** at least one of the following places: within a NOTICE text file distributed **as** part of the Derivative Works; within the Source form **or** documentation, **if** provided along **with** the Derivative Works; **or**, within a display generated by the Derivative Works, **if and** wherever such third-party notices normally appear. The contents of the NOTICE file are **for** informational purposes only **and** do **not** modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside **or as** an addendum to the NOTICE text **from the** Work, provided that such additional attribution notices cannot be construed **as** modifying the License.

You may add Your own copyright statement to Your modifications **and** may provide additional **or** different license terms **and** conditions **for** use, reproduction, **or** distribution of Your modifications, **or for any** such Derivative Works **as** a whole, provided Your use, reproduction, **and** distribution of the Work otherwise complies **with** the conditions stated **in** this License.

5. Submission of Contributions. Unless You explicitly state otherwise, **any** Contribution intentionally submitted **for** inclusion **in** the Work by You to the Licensor shall be under the terms **and** conditions of this License, without **any** additional terms **or** conditions. Notwithstanding the above, nothing herein shall supersede **or** modify

(continues on next page)

(continued from previous page)

- the terms of **any** separate license agreement you may have executed **with** Licensor regarding such Contributions.
6. Trademarks. This License does **not** grant permission to use the trade names, trademarks, service marks, **or** product names of the Licensor, **except as** required **for** reasonable **and** customary use **in** describing the origin of the Work **and** reproducing the content of the NOTICE file.
  7. Disclaimer of Warranty. Unless required by applicable law **or** agreed to **in** writing, Licensor provides the Work (**and** each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express **or** implied, including, without limitation, **any** warranties **or** conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, **or** FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible **for** determining the appropriateness of using **or** redistributing the Work **and** assume **any** risks associated **with** Your exercise of permissions under this License.
  8. Limitation of Liability. In no event **and** under no legal theory, whether **in** tort (including negligence), contract, **or** otherwise, unless required by applicable law (such **as** deliberate **and** grossly negligent acts) **or** agreed to **in** writing, shall **any** Contributor be liable to You **for** damages, including **any** direct, indirect, special, incidental, **or** consequential damages of **any** character arising **as** a result of this License **or** out of the use **or** inability to use the Work (including but **not** limited to damages **for** loss of goodwill, work stoppage, computer failure **or** malfunction, **or** **any and all** other commercial damages **or** losses), even **if** such Contributor has been advised of the possibility of such damages.
  9. Accepting Warranty **or** Additional Liability. While redistributing the Work **or** Derivative Works thereof, You may choose to offer, **and** charge a fee **for**, acceptance of support, warranty, indemnity, **or** other liability obligations **and/or** rights consistent **with** this License. However, **in** accepting such obligations, You may act only on Your own behalf **and** on Your sole responsibility, **not** on behalf of **any** other Contributor, **and** only **if** You agree to indemnify, defend, **and** hold each Contributor harmless **for** **any** liability incurred by, **or** claims asserted against, such Contributor by reason of your accepting **any** such warranty **or** additional liability.

### 3.3 Quick Start

The installation is quick and straightforward.

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

# install from pypi using pip
pip install apache-airflow

# initialize the database
```

(continues on next page)

(continued from previous page)

```
airflow initdb

# start the web server, default port is 8080
airflow webserver -p 8080

# start the scheduler
airflow scheduler

# visit localhost:8080 in the browser and enable the example dag in the home page
```

Upon running these commands, Airflow will create the `$AIRFLOW_HOME` folder and lay an “`airflow.cfg`” file with defaults that get you going fast. You can inspect the file either in `$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 `airflow.executors.sequential_executor.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.

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 run example_bash_operator runme_0 2015-01-01
# run a backfill over 2 days
airflow backfill example_bash_operator -s 2015-01-01 -e 2015-01-02
```

### 3.3.1 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.

## 3.4 Installation

### 3.4.1 Getting Airflow

The easiest way to install the latest stable version of Airflow is with `pip`:

```
pip install apache-airflow
```

You can also install Airflow with support for extra features like `gcp_api` or `postgres`:

```
pip install apache-airflow[postgres,gcp_api]
```

### 3.4.2 Extra Packages

The `apache-airflow` PyPI basic package only installs what’s needed to get started. Subpackages 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.

Behind the scenes, Airflow does conditional imports of operators that require these extra dependencies.

Here's the list of the subpackages and what they enable:

subpackage	install command	enables
all	<code>pip install apache-airflow[all]</code>	All Airflow features known to man
all_dbs	<code>pip install apache-airflow[all_dbs]</code>	All databases integrations
async	<code>pip install apache-airflow[async]</code>	Async worker classes for Gunicorn
azure	<code>pip install apache-airflow[azure]</code>	Microsoft Azure
aws	<code>pip install apache-airflow[aws]</code>	Amazon Web Services
celery	<code>pip install apache-airflow[celery]</code>	CeleryExecutor
cloudant	<code>pip install apache-airflow[cloudant]</code>	Cloudant hook
crypto	<code>pip install apache-airflow[crypto]</code>	Encrypt connection passwords in metadata
devel	<code>pip install apache-airflow[devel]</code>	Minimum dev tools requirements
devel_hadoop	<code>pip install apache-airflow[devel_hadoop]</code>	Airflow + dependencies on the Hadoop stack
druid	<code>pip install apache-airflow[druid]</code>	Druid related operators & hooks
gcp_api	<code>pip install apache-airflow[gcp_api]</code>	Google Cloud Platform hooks and operators
github_enterprise	<code>pip install apache-airflow[github_enterprise]</code>	Github Enterprise auth backend
google_auth	<code>pip install apache-airflow[google_auth]</code>	Google auth backend
hdfs	<code>pip install apache-airflow[hdfs]</code>	HDFS hooks and operators
hive	<code>pip install apache-airflow[hive]</code>	All Hive related operators
jdbc	<code>pip install apache-airflow[jdbc]</code>	JDBC hooks and operators
kerberos	<code>pip install apache-airflow[kerberos]</code>	Kerberos integration for Kerberized Hadoop
kubernetes	<code>pip install apache-airflow[kubernetes]</code>	Kubernetes Executor and operator
ldap	<code>pip install apache-airflow[ldap]</code>	LDAP authentication for users
mssql	<code>pip install apache-airflow[mssql]</code>	Microsoft SQL Server operators and hook
mysql	<code>pip install apache-airflow[mysql]</code>	MySQL operators and hook, support as an executor
password	<code>pip install apache-airflow[password]</code>	Password authentication for users
postgres	<code>pip install apache-airflow[postgres]</code>	PostgreSQL operators and hook, support as an executor
qds	<code>pip install apache-airflow[qds]</code>	Enable QDS (Qubole Data Service) support
rabbitmq	<code>pip install apache-airflow[rabbitmq]</code>	RabbitMQ support as a Celery backend
redis	<code>pip install apache-airflow[redis]</code>	Redis hooks and sensors
samba	<code>pip install apache-airflow[samba]</code>	<code>airflow.operators.hive_to_samba</code>
slack	<code>pip install apache-airflow[slack]</code>	<code>airflow.operators.slack_operator</code>
ssh	<code>pip install apache-airflow[ssh]</code>	SSH hooks and Operator
vertica	<code>pip install apache-airflow[vertica]</code>	Vertica hook support as an Airflow backend

### 3.4.3 Initiating Airflow Database

Airflow requires a database to be initiated before you can run tasks. 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 [Initializing a Database Backend](#) to setup a different database.

After configuration, you'll need to initialize the database before you can run tasks:

```
airflow initdb
```

## 3.5 Tutorial

This tutorial walks you through some of the fundamental Airflow concepts, objects, and their usage while writing your first pipeline.

### 3.5.1 Example Pipeline definition

Here is an example of a basic pipeline definition. Do not worry if this looks complicated, a line by line explanation follows below.

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/airflow/blob/master/airflow/example_dags/tutorial.py
"""
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
}

dag = DAG('tutorial', default_args=default_args, schedule_interval=timedelta(days=1))

# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7) }}"
    echo "{{ params.my_param }}"
{% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
```

(continues on next page)

(continued from previous page)

```
t3.set_upstream(t1)
```

### 3.5.2 It's a DAG definition file

One thing to wrap your head around (it may not be very intuitive for everyone at first) is that this Airflow Python script is really just a configuration file specifying the DAG's structure as code. The actual tasks defined here will run in a different context from the context of this script. Different tasks run on different workers at different points in time, which means that this script cannot be used to cross communicate between tasks. Note that for this purpose we have a more advanced feature called XCom.

People sometimes think of the DAG definition file as a place where they can do some actual data processing - that is not the case at all! The script's purpose is to define a DAG object. It needs to evaluate quickly (seconds, not minutes) since the scheduler will execute it periodically to reflect the changes if any.

### 3.5.3 Importing Modules

An Airflow pipeline is just a Python script that happens to define an Airflow DAG object. Let's start by importing the libraries we will need.

```
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG

# Operators; we need this to operate!
from airflow.operators.bash_operator import BashOperator
```

### 3.5.4 Default Arguments

We're about to create a DAG and some tasks, and we have the choice to explicitly pass a set of arguments to each task's constructor (which would become redundant), or (better!) we can define a dictionary of default parameters that we can use when creating tasks.

```
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
}
```

For more information about the BaseOperator's parameters and what they do, refer to the [airflow.models.BaseOperator](#) documentation.

Also, note that you could easily define different sets of arguments that would serve different purposes. An example of that would be to have different settings between a production and development environment.

### 3.5.5 Instantiate a DAG

We'll need a DAG object to nest our tasks into. Here we pass a string that defines the `dag_id`, which serves as a unique identifier for your DAG. We also pass the default argument dictionary that we just defined and define a `schedule_interval` of 1 day for the DAG.

```
dag = DAG(
    'tutorial', default_args=default_args, schedule_interval=timedelta(days=1))
```

### 3.5.6 Tasks

Tasks are generated when instantiating operator objects. An object instantiated from an operator is called a constructor. The first argument `task_id` acts as a unique identifier for the task.

```
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)
```

Notice how we pass a mix of operator specific arguments (`bash_command`) and an argument common to all operators (`retries`) inherited from `BaseOperator` to the operator's constructor. This is simpler than passing every argument for every constructor call. Also, notice that in the second task we override the `retries` parameter with 3.

The precedence rules for a task are as follows:

1. Explicitly passed arguments
2. Values that exist in the `default_args` dictionary
3. The operator's default value, if one exists

A task must include or inherit the arguments `task_id` and `owner`, otherwise Airflow will raise an exception.

### 3.5.7 Templating with Jinja

Airflow leverages the power of [Jinja Templating](#) and provides the pipeline author with a set of built-in parameters and macros. Airflow also provides hooks for the pipeline author to define their own parameters, macros and templates.

This tutorial barely scratches the surface of what you can do with templating in Airflow, but the goal of this section is to let you know this feature exists, get you familiar with double curly brackets, and point to the most common template variable: `{{ ds }}` (today's "date stamp").

```
templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7) }}"
        echo "{{ params.my_param }}"
    {% endfor %}
    """
```

(continues on next page)



(continued from previous page)

```
t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)
```

Notice that the `templated_command` contains code logic in `{% %}` blocks, references parameters like `{{ ds }}`, calls a function as in `{{ macros.ds_add(ds, 7) }}`, and references a user-defined parameter in `{{ params.my_param }}`.

The `params` hook in `BaseOperator` allows you to pass a dictionary of parameters and/or objects to your templates. Please take the time to understand how the parameter `my_param` makes it through to the template.

Files can also be passed to the `bash_command` argument, like `bash_command='templated_command.sh'`, where the file location is relative to the directory containing the pipeline file (`tutorial.py` in this case). This may be desirable for many reasons, like separating your script's logic and pipeline code, allowing for proper code highlighting in files composed in different languages, and general flexibility in structuring pipelines. It is also possible to define your `template_searchpath` as pointing to any folder locations in the DAG constructor call.

Using that same DAG constructor call, it is possible to define `user_defined_macros` which allow you to specify your own variables. For example, passing `dict(foo='bar')` to this argument allows you to use `{{ foo }}` in your templates. Moreover, specifying `user_defined_filters` allow you to register your own filters. For example, passing `dict(hello=lambda name: 'Hello %s' % name)` to this argument allows you to use `{{ 'world' | hello }}` in your templates. For more information regarding custom filters have a look at the [Jinja Documentation](#)

For more information on the variables and macros that can be referenced in templates, make sure to read through the [Macros](#) section

### 3.5.8 Setting up Dependencies

We have tasks `t1`, `t2` and `t3` that do not depend on each other. Here's a few ways you can define dependencies between them:

```
t1.set_downstream(t2)

# This means that t2 will depend on t1
# running successfully to run.
# It is equivalent to:
t2.set_upstream(t1)

# The bit shift operator can also be
# used to chain operations:
t1 >> t2

# And the upstream dependency with the
# bit shift operator:
t2 << t1

# Chaining multiple dependencies becomes
# concise with the bit shift operator:
t1 >> t2 >> t3

# A list of tasks can also be set as
# dependencies. These operations
```

(continues on next page)

(continued from previous page)

```
# all have the same effect:
t1.set_downstream([t2, t3])
t1 >> [t2, t3]
[t2, t3] << t1
```

Note that when executing your script, Airflow will raise exceptions when it finds cycles in your DAG or when a dependency is referenced more than once.

### 3.5.9 Recap

Alright, so we have a pretty basic DAG. At this point your code should look something like this:

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/airflow/blob/master/airflow/example_dags/tutorial.py
"""
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
}

dag = DAG(
    'tutorial', default_args=default_args, schedule_interval=timedelta(days=1))

# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7) }}"
```

(continues on next page)

(continued from previous page)

```

        echo "{{ params.my_param }}"
    {% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)

```

## 3.5.10 Testing

### 3.5.10.1 Running the Script

Time to run some tests. First let's make sure that the pipeline parses. Let's assume we're saving the code from the previous step in `tutorial.py` in the DAGs folder referenced in your `airflow.cfg`. The default location for your DAGs is `~/airflow/dags`.

```
python ~/airflow/dags/tutorial.py
```

If the script does not raise an exception it means that you haven't done anything horribly wrong, and that your Airflow environment is somewhat sound.

### 3.5.10.2 Command Line Metadata Validation

Let's run a few commands to validate this script further.

```

# print the list of active DAGs
airflow list_dags

# prints the list of tasks in the "tutorial" DAG
airflow list_tasks tutorial

# prints the hierarchy of tasks in the "tutorial" DAG
airflow list_tasks tutorial --tree

```

### 3.5.10.3 Testing

Let's test by running the actual task instances on a specific date. The date specified in this context is an `execution_date`, which simulates the scheduler running your task or dag at a specific date + time:

```

# command layout: command subcommand dag_id task_id date

# testing print_date
airflow test tutorial print_date 2015-06-01

# testing sleep
airflow test tutorial sleep 2015-06-01

```

Now remember what we did with templating earlier? See how this template gets rendered and executed by running this command:

```
# testing templated
airflow test tutorial templated 2015-06-01
```

This should result in displaying a verbose log of events and ultimately running your bash command and printing the result.

Note that the `airflow test` command runs task instances locally, outputs their log to stdout (on screen), doesn't bother with dependencies, and doesn't communicate state (running, success, failed, ...) to the database. It simply allows testing a single task instance.

### 3.5.10.4 Backfill

Everything looks like it's running fine so let's run a backfill. `backfill` will respect your dependencies, emit logs into files and talk to the database to record status. If you do have a webserver up, you'll be able to track the progress. `airflow webserver` will start a web server if you are interested in tracking the progress visually as your backfill progresses.

Note that if you use `depends_on_past=True`, individual task instances will depend on the success of the preceding task instance, except for the `start_date` specified itself, for which this dependency is disregarded.

The date range in this context is a `start_date` and optionally an `end_date`, which are used to populate the run schedule with task instances from this dag.

```
# optional, start a web server in debug mode in the background
# airflow webserver --debug &

# start your backfill on a date range
airflow backfill tutorial -s 2015-06-01 -e 2015-06-07
```

### 3.5.11 What's Next?

That's it, you've written, tested and backfilled your very first Airflow pipeline. Merging your code into a code repository that has a master scheduler running against it should get it to get triggered and run every day.

Here's a few things you might want to do next:

- Take an in-depth tour of the UI - click all the things!
- Keep reading the docs! Especially the sections on:
  - Command line interface
  - Operators
  - Macros
- Write your first pipeline!

## 3.6 How-to Guides

Setting up the sandbox in the [Quick Start](#) section was easy; building a production-grade environment requires a bit more work!

These how-to guides will step you through common tasks in using and configuring an Airflow environment.

### 3.6.1 Add a new role in RBAC UI

There are five roles created for Airflow by default: Admin, User, Op, Viewer, and Public. The master branch adds beta support for DAG level access for RBAC UI. Each DAG comes with two permissions: read and write.

The Admin could create a specific role which is only allowed to read / write certain DAGs. To configure a new role, go to **Security** tab and click **List Roles** in the new UI.

The screenshot shows the Airflow RBAC UI. At the top is a navigation bar with links for DAGs, Security, Browse, Admin, Docs, and About. Below this is the 'Add Role' form. The form has three main sections: 'Permissions', 'Name', and 'User'. The 'Permissions' section has a dropdown menu open, showing a list of permissions: 'python', 'can dag edit on example\_python\_operator', 'can dag read on example\_python\_operator', and 'Select Value'. Below the form is a 'Save' button and a back arrow. At the bottom of the page, there is a table listing existing roles. The table has columns for a checkbox, a search icon, a role name, and a list of permissions. The first row shows a role named 'python\_tester' with the permission '[can dag edit on example\_python\_operator]'.

The image shows the creation of a role which can only write to `example_python_operator`. You can also create roles via the CLI using the `airflow roles` command, e.g.:

```
airflow roles --create Role1 Role2
```

And we could assign the given role to a new user using the `airflow users --add-role` CLI command. Default roles (Admin, User, Viewer, Op) shipped with RBAC could view the details for every dag.

### 3.6.2 Setting Configuration Options

The first time you run Airflow, it will create a file called `airflow.cfg` in your `$AIRFLOW_HOME` directory (`~/airflow` by default). This file contains Airflow's configuration and you can edit it to change any of the settings. You can also set options with environment variables by using this format: `$AIRFLOW__{SECTION}__{KEY}` (note the double underscores).

For example, the metadata database connection string can either be set in `airflow.cfg` like this:

```
[core]
sql_alchemy_conn = my_conn_string
```

or by creating a corresponding environment variable:

```
AIRFLOW__CORE__SQL_ALCHEMY_CONN=my_conn_string
```

You can also derive the connection string at run time by appending `_cmd` to the key like this:

```
[core]
sql_alchemy_conn_cmd = bash_command_to_run
```

The following config options support this `_cmd` version:

- `sql_alchemy_conn` in `[core]` section
- `fernet_key` in `[core]` section
- `broker_url` in `[celery]` section
- `result_backend` in `[celery]` section
- `password` in `[atlas]` section
- `smtp_password` in `[smtp]` section
- `bind_password` in `[ldap]` section
- `git_password` in `[kubernetes]` section

The idea behind this is to not store passwords on boxes in plain text files.

The order of precedence for all config options is as follows -

1. environment variable
2. configuration in `airflow.cfg`
3. command in `airflow.cfg`
4. Airflow's built in defaults

### 3.6.3 Initializing a Database Backend

If you want to take a real test drive of Airflow, you should consider setting up a real database backend and switching to the LocalExecutor.

As Airflow was built to interact with its metadata using the great SQLAlchemy library, you should be able to use any database backend supported as a SQLAlchemy backend. We recommend using **MySQL** or **Postgres**.

---

**Note:** We rely on more strict ANSI SQL settings for MySQL in order to have sane defaults. Make sure to have specified `explicit_defaults_for_timestamp=1` in your `my.cnf` under `[mysqld]`

---

---

**Note:** If you decide to use **Postgres**, we recommend using the `psycopg2` driver and specifying it in your SQLAlchemy connection string. Also note that since SQLAlchemy does not expose a way to target a specific schema in the Postgres connection URI, you may want to set a default schema for your role with a command similar to `ALTER ROLE username SET search_path = airflow, foobar;`

---

Once you’ve setup your database to host Airflow, you’ll need to alter the SQLAlchemy connection string located in your configuration file `$AIRFLOW_HOME/airflow.cfg`. You should then also change the “executor” setting to use “LocalExecutor”, an executor that can parallelize task instances locally.

```
# initialize the database
airflow initdb
```

### 3.6.4 Using Operators

An operator represents a single, ideally idempotent, task. Operators determine what actually executes when your DAG runs.

See the [Operators Concepts](#) documentation and the [Operators API Reference](#) for more information.

- *BashOperator*
  - *Templating*
  - *Troubleshooting*
    - \* *Jinja template not found*
- *PythonOperator*
  - *Passing in arguments*
  - *Templating*
- *Google Cloud Storage Operators*
  - *GoogleCloudStorageToBigQueryOperator*
- *Google Compute Engine Operators*
  - *GceInstanceStartOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *GceInstanceStopOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *GceSetMachineTypeOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*

- *GceInstanceTemplateCopyOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *GceInstanceGroupManagerUpdateTemplateOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *Troubleshooting*
  - \* *More information*
- *Google Cloud Bigtable Operators*
  - *Arguments*
  - *BigtableInstanceCreateOperator*
    - \* *Using the operator*
  - *BigtableInstanceDeleteOperator*
    - \* *Using the operator*
  - *BigtableClusterUpdateOperator*
    - \* *Using the operator*
  - *BigtableTableCreateOperator*
    - \* *Using the operator*
    - \* *Advanced*
  - *BigtableTableDeleteOperator*
    - \* *Using the operator*
  - *BigtableTableWaitForReplicationSensor*
    - \* *Using the operator*
- *Google Cloud Functions Operators*
  - *GcfFunctionDeleteOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *GcfFunctionDeployOperator*
    - \* *Arguments*
    - \* *Using the operator*



- \* *Templating*
- \* *Troubleshooting*
- \* *More information*
- *Google Cloud Spanner Operators*
  - *CloudSpannerInstanceDatabaseDeleteOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *CloudSpannerInstanceDatabaseDeployOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *CloudSpannerInstanceDatabaseUpdateOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *CloudSpannerInstanceDatabaseQueryOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *CloudSpannerInstanceDeleteOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
- *Google Cloud Sql Operators*
  - *CloudSqlInstanceDatabaseCreateOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*

- *CloudSqlInstanceDatabaseDeleteOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *CloudSqlInstanceDatabasePatchOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *CloudSqlInstanceDeleteOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *CloudSqlInstanceExportOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
  - \* *Troubleshooting*
- *CloudSqlInstanceImportOperator*
  - \* *CSV import:*
  - \* *SQL import:*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
  - \* *Troubleshooting*
- *CloudSqlInstanceCreateOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *CloudSqlInstancePatchOperator*

- \* *Arguments*
- \* *Using the operator*
- \* *Templating*
- \* *More information*
- *CloudSqlQueryOperator*
  - \* *Arguments*
  - \* *Using the operator*
  - \* *Templating*
  - \* *More information*
- *Google Cloud Storage Operators*
  - *GoogleCloudStorageBucketCreateAclEntryOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*
  - *GoogleCloudStorageObjectCreateAclEntryOperator*
    - \* *Arguments*
    - \* *Using the operator*
    - \* *Templating*
    - \* *More information*

### 3.6.4.1 BashOperator

Use the *BashOperator* to execute commands in a *Bash* shell.

```
run_this = BashOperator(
    task_id='run_after_loop',
    bash_command='echo 1',
    dag=dag,
)
```

### Templating

You can use *Jinja templates* to parameterize the `bash_command` argument.

```
also_run_this = BashOperator(
    task_id='also_run_this',
    bash_command='echo "run_id={{ run_id }} | dag_run={{ dag_run }}"',
    dag=dag,
)
```

## Troubleshooting

### Jinja template not found

Add a space after the script name when directly calling a Bash script with the `bash_command` argument. This is because Airflow tries to apply a Jinja template to it, which will fail.

```
t2 = BashOperator(
    task_id='bash_example',

    # This fails with `Jinja template not found` error
    # bash_command="/home/batcher/test.sh",

    # This works (has a space after)
    bash_command="/home/batcher/test.sh ",
    dag=dag)
```

### 3.6.4.2 PythonOperator

Use the `PythonOperator` to execute Python callables.

```
def print_context(ds, **kwargs):
    pprint(kwargs)
    print(ds)
    return 'Whatever you return gets printed in the logs'

run_this = PythonOperator(
    task_id='print_the_context',
    provide_context=True,
    python_callable=print_context,
    dag=dag,
)
```

## Passing in arguments

Use the `op_args` and `op_kwargs` arguments to pass additional arguments to the Python callable.

```
def my_sleeping_function(random_base):
    """This is a function that will run within the DAG execution"""
    time.sleep(random_base)

# Generate 5 sleeping tasks, sleeping from 0.0 to 0.4 seconds respectively
for i in range(5):
    task = PythonOperator(
        task_id='sleep_for_' + str(i),
        python_callable=my_sleeping_function,
        op_kwargs={'random_base': float(i) / 10},
        dag=dag,
    )

    run_this >> task
```

## Templating

When you set the `provide_context` argument to `True`, Airflow passes in an additional set of keyword arguments: one for each of the *Jinja template variables* and a `templates_dict` argument.

The `templates_dict` argument is templated, so each value in the dictionary is evaluated as a *Jinja template*.

### 3.6.4.3 Google Cloud Storage Operators

#### GoogleCloudStorageToBigQueryOperator

Use the *GoogleCloudStorageToBigQueryOperator* to execute a BigQuery load job.

```
load_csv = gcs_to_bq.GoogleCloudStorageToBigQueryOperator(
    task_id='gcs_to_bq_example',
    bucket='cloud-samples-data',
    source_objects=['bigquery/us-states/us-states.csv'],
    destination_project_dataset_table='airflow_test.gcs_to_bq_table',
    schema_fields=[
        {'name': 'name', 'type': 'STRING', 'mode': 'NULLABLE'},
        {'name': 'post_abbr', 'type': 'STRING', 'mode': 'NULLABLE'},
    ],
    write_disposition='WRITE_TRUNCATE',
    dag=dag)
```

### 3.6.4.4 Google Compute Engine Operators

#### GceInstanceStartOperator

Use the *GceInstanceStartOperator* to start an existing Google Compute Engine instance.

## Arguments

The following examples of OS environment variables used to pass arguments to the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCE_ZONE = os.environ.get('GCE_ZONE', 'europe-west1-b')
GCE_INSTANCE = os.environ.get('GCE_INSTANCE', 'testinstance')
```

## Using the operator

The code to create the operator:

```
gce_instance_start = GceInstanceStartOperator(
    project_id=GCP_PROJECT_ID,
    zone=GCE_ZONE,
    resource_id=GCE_INSTANCE,
    task_id='gcp_compute_start_task'
)
```

You can also create the operator without project id - project id will be retrieved from the GCP connection id used:

```
gce_instance_start2 = GceInstanceStartOperator(  
    zone=GCE_ZONE,  
    resource_id=GCE_INSTANCE,  
    task_id='gcp_compute_start_task2'  
)
```

## Templating

```
template_fields = ('project_id', 'zone', 'resource_id', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Compute Engine API documentation](#) for start.

## GceInstanceStopOperator

Use the operator to stop Google Compute Engine instance.

For parameter definition, take a look at *GceInstanceStopOperator*

## Arguments

The following examples of OS environment variables used to pass arguments to the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')  
GCE_ZONE = os.environ.get('GCE_ZONE', 'europe-west1-b')  
GCE_INSTANCE = os.environ.get('GCE_INSTANCE', 'testinstance')
```

## Using the operator

The code to create the operator:

```
gce_instance_stop = GceInstanceStopOperator(  
    project_id=GCP_PROJECT_ID,  
    zone=GCE_ZONE,  
    resource_id=GCE_INSTANCE,  
    task_id='gcp_compute_stop_task'  
)
```

You can also create the operator without project id - project id will be retrieved from the GCP connection used:

```
gce_instance_stop2 = GceInstanceStopOperator(  
    zone=GCE_ZONE,  
    resource_id=GCE_INSTANCE,  
    task_id='gcp_compute_stop_task2'  
)
```

## Templating

```
template_fields = ('project_id', 'zone', 'resource_id', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Compute Engine API documentation](#) for stop.

## GceSetMachineTypeOperator

Use the operator to change machine type of a Google Compute Engine instance.

For parameter definition, take a look at [GceSetMachineTypeOperator](#).

## Arguments

The following examples of OS environment variables used to pass arguments to the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCE_ZONE = os.environ.get('GCE_ZONE', 'europe-west1-b')
GCE_INSTANCE = os.environ.get('GCE_INSTANCE', 'testinstance')
```

```
GCE_SHORT_MACHINE_TYPE_NAME = os.environ.get('GCE_SHORT_MACHINE_TYPE_NAME', 'n1-
↳standard-1')
SET_MACHINE_TYPE_BODY = {
    'machineType': 'zones/{}/machineTypes/{}'.format(GCE_ZONE, GCE_SHORT_MACHINE_TYPE_
↳NAME)
}
```

## Using the operator

The code to create the operator:

```
gce_set_machine_type = GceSetMachineTypeOperator(
    project_id=GCP_PROJECT_ID,
    zone=GCE_ZONE,
    resource_id=GCE_INSTANCE,
    body=SET_MACHINE_TYPE_BODY,
    task_id='gcp_compute_set_machine_type'
)
```

You can also create the operator without project id - project id will be retrieved from the GCP connection used:

```
gce_set_machine_type2 = GceSetMachineTypeOperator(
    zone=GCE_ZONE,
    resource_id=GCE_INSTANCE,
    body=SET_MACHINE_TYPE_BODY,
    task_id='gcp_compute_set_machine_type2'
)
```

## Templating

```
template_fields = ('project_id', 'zone', 'resource_id', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Compute Engine API documentation](#) for `setMachineType`.

## GceInstanceTemplateCopyOperator

Use the operator to copy an existing Google Compute Engine instance template applying a patch to it.

For parameter definition, take a look at [GceInstanceTemplateCopyOperator](#).

## Arguments

The following examples of OS environment variables used to pass arguments to the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCE_ZONE = os.environ.get('GCE_ZONE', 'europe-west1-b')
```

```
GCE_TEMPLATE_NAME = os.environ.get('GCE_TEMPLATE_NAME', 'instance-template-test')
GCE_NEW_TEMPLATE_NAME = os.environ.get('GCE_NEW_TEMPLATE_NAME',
                                       'instance-template-test-new')
GCE_NEW_DESCRIPTION = os.environ.get('GCE_NEW_DESCRIPTION', 'Test new description')
GCE_INSTANCE_TEMPLATE_BODY_UPDATE = {
    "name": GCE_NEW_TEMPLATE_NAME,
    "description": GCE_NEW_DESCRIPTION,
    "properties": {
        "machineType": "n1-standard-2"
    }
}
```

## Using the operator

The code to create the operator:

```
gce_instance_template_copy = GceInstanceTemplateCopyOperator(
    project_id=GCP_PROJECT_ID,
    resource_id=GCE_TEMPLATE_NAME,
    body_patch=GCE_INSTANCE_TEMPLATE_BODY_UPDATE,
    task_id='gcp_compute_igm_copy_template_task'
)
```

You can also create the operator without project id - project id will be retrieved from the GCP connection used:

```
gce_instance_template_copy2 = GceInstanceTemplateCopyOperator(
    resource_id=GCE_TEMPLATE_NAME,
    body_patch=GCE_INSTANCE_TEMPLATE_BODY_UPDATE,
    task_id='gcp_compute_igm_copy_template_task_2'
)
```



## Templating

```
template_fields = ('project_id', 'resource_id', 'request_id',
                  'gcp_conn_id', 'api_version')
```

## More information

See [Google Compute Engine API documentation](#) for `instanceTemplates`.

## GceInstanceGroupManagerUpdateTemplateOperator

Use the operator to update template in Google Compute Engine Instance Group Manager.

For parameter definition, take a look at *[GceInstanceGroupManagerUpdateTemplateOperator](#)*.

## Arguments

The following examples of OS environment variables used to pass arguments to the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCE_ZONE = os.environ.get('GCE_ZONE', 'europe-west1-b')
```

```
GCE_INSTANCE_GROUP_MANAGER_NAME = os.environ.get('GCE_INSTANCE_GROUP_MANAGER_NAME',
                                                  'instance-group-test')

SOURCE_TEMPLATE_URL = os.environ.get(
    'SOURCE_TEMPLATE_URL',
    "https://www.googleapis.com/compute/beta/projects/" + GCP_PROJECT_ID +
    "/global/instanceTemplates/instance-template-test")

DESTINATION_TEMPLATE_URL = os.environ.get(
    'DESTINATION_TEMPLATE_URL',
    "https://www.googleapis.com/compute/beta/projects/" + GCP_PROJECT_ID +
    "/global/instanceTemplates/" + GCE_NEW_TEMPLATE_NAME)

UPDATE_POLICY = {
    "type": "OPPORTUNISTIC",
    "minimalAction": "RESTART",
    "maxSurge": {
        "fixed": 1
    },
    "minReadySec": 1800
}
```

## Using the operator

The code to create the operator:

```
gce_instance_group_manager_update_template = \
    GceInstanceGroupManagerUpdateTemplateOperator(
        project_id=GCP_PROJECT_ID,
        resource_id=GCE_INSTANCE_GROUP_MANAGER_NAME,
        zone=GCE_ZONE,
        source_template=SOURCE_TEMPLATE_URL,
        destination_template=DESTINATION_TEMPLATE_URL,
        update_policy=UPDATE_POLICY,
        task_id='gcp_compute_igm_group_manager_update_template'
    )
```

You can also create the operator without project id - project id will be retrieved from the GCP connection used:

```
gce_instance_group_manager_update_template2 = \
    GceInstanceGroupManagerUpdateTemplateOperator(
        resource_id=GCE_INSTANCE_GROUP_MANAGER_NAME,
        zone=GCE_ZONE,
        source_template=SOURCE_TEMPLATE_URL,
        destination_template=DESTINATION_TEMPLATE_URL,
        task_id='gcp_compute_igm_group_manager_update_template_2'
    )
```

## Templating

```
template_fields = ('project_id', 'resource_id', 'zone', 'request_id',
                   'source_template', 'destination_template',
                   'gcp_conn_id', 'api_version')
```

## Troubleshooting

You might find that your `GceInstanceGroupManagerUpdateTemplateOperator` fails with missing permissions. To execute the operation, the service account requires the permissions that the `Service Account User` role provides (assigned via Google Cloud IAM).

## More information

See [Google Compute Engine API documentation](#) for `instanceGroupManagers`.

### 3.6.4.5 Google Cloud Bigtable Operators

#### Arguments

All examples below rely on the following variables, which can be passed via environment variables.

```
GCP_PROJECT_ID = getenv('GCP_PROJECT_ID', 'example-project')
CBT_INSTANCE_ID = getenv('CBT_INSTANCE_ID', 'some-instance-id')
CBT_INSTANCE_DISPLAY_NAME = getenv('CBT_INSTANCE_DISPLAY_NAME', 'Human-readable name')
CBT_INSTANCE_TYPE = getenv('CBT_INSTANCE_TYPE', '2')
CBT_INSTANCE_LABELS = getenv('CBT_INSTANCE_LABELS', '{}')
CBT_CLUSTER_ID = getenv('CBT_CLUSTER_ID', 'some-cluster-id')
```

(continues on next page)

(continued from previous page)

```

CBT_CLUSTER_ZONE = getenv('CBT_CLUSTER_ZONE', 'europe-west1-b')
CBT_CLUSTER_NODES = getenv('CBT_CLUSTER_NODES', '3')
CBT_CLUSTER_NODES_UPDATED = getenv('CBT_CLUSTER_NODES_UPDATED', '5')
CBT_CLUSTER_STORAGE_TYPE = getenv('CBT_CLUSTER_STORAGE_TYPE', '2')
CBT_TABLE_ID = getenv('CBT_TABLE_ID', 'some-table-id')
CBT_POKE_INTERVAL = getenv('CBT_POKE_INTERVAL', '60')

```

## BigtableInstanceCreateOperator

Use the *BigtableInstanceCreateOperator* to create a Google Cloud Bigtable instance.

If the Cloud Bigtable instance with the given ID exists, the operator does not compare its configuration and immediately succeeds. No changes are made to the existing instance.

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

create_instance_task = BigtableInstanceCreateOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=CBT_INSTANCE_ID,
    main_cluster_id=CBT_CLUSTER_ID,
    main_cluster_zone=CBT_CLUSTER_ZONE,
    instance_display_name=CBT_INSTANCE_DISPLAY_NAME,
    instance_type=int(CBT_INSTANCE_TYPE),
    instance_labels=json.loads(CBT_INSTANCE_LABELS),
    cluster_nodes=int(CBT_CLUSTER_NODES),
    cluster_storage_type=int(CBT_CLUSTER_STORAGE_TYPE),
    task_id='create_instance_task',
)
create_instance_task2 = BigtableInstanceCreateOperator(
    instance_id=CBT_INSTANCE_ID,
    main_cluster_id=CBT_CLUSTER_ID,
    main_cluster_zone=CBT_CLUSTER_ZONE,
    instance_display_name=CBT_INSTANCE_DISPLAY_NAME,
    instance_type=int(CBT_INSTANCE_TYPE),
    instance_labels=json.loads(CBT_INSTANCE_LABELS),
    cluster_nodes=int(CBT_CLUSTER_NODES),
    cluster_storage_type=int(CBT_CLUSTER_STORAGE_TYPE),
    task_id='create_instance_task2',
)
create_instance_task >> create_instance_task2

```

## BigtableInstanceDeleteOperator

Use the *BigtableInstanceDeleteOperator* to delete a Google Cloud Bigtable instance.

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
delete_instance_task = BigtableInstanceDeleteOperator(  
    project_id=GCP_PROJECT_ID,  
    instance_id=CBT_INSTANCE_ID,  
    task_id='delete_instance_task',  
)  
delete_instance_task2 = BigtableInstanceDeleteOperator(  
    instance_id=CBT_INSTANCE_ID,  
    task_id='delete_instance_task2',  
)
```

## BigtableClusterUpdateOperator

Use the *BigtableClusterUpdateOperator* to modify number of nodes in a Cloud Bigtable cluster.

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
cluster_update_task = BigtableClusterUpdateOperator(  
    project_id=GCP_PROJECT_ID,  
    instance_id=CBT_INSTANCE_ID,  
    cluster_id=CBT_CLUSTER_ID,  
    nodes=int(CBT_CLUSTER_NODES_UPDATED),  
    task_id='update_cluster_task',  
)  
cluster_update_task2 = BigtableClusterUpdateOperator(  
    instance_id=CBT_INSTANCE_ID,  
    cluster_id=CBT_CLUSTER_ID,  
    nodes=int(CBT_CLUSTER_NODES_UPDATED),  
    task_id='update_cluster_task2',  
)  
cluster_update_task >> cluster_update_task2
```

## BigtableTableCreateOperator

Creates a table in a Cloud Bigtable instance.

If the table with given ID exists in the Cloud Bigtable instance, the operator compares the Column Families. If the Column Families are identical operator succeeds. Otherwise, the operator fails with the appropriate error message.

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

create_table_task = BigtableTableCreateOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    task_id='create_table',
)
create_table_task2 = BigtableTableCreateOperator(
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    task_id='create_table_task2',
)
create_table_task >> create_table_task2

```

## Advanced

When creating a table, you can specify the optional `initial_split_keys` and `column_families`. Please refer to the Python Client for Google Cloud Bigtable documentation [for Table](#) and [for Column Families](#).

## BigtableTableDeleteOperator

Use the *BigtableTableDeleteOperator* to delete a table in Google Cloud Bigtable.

### Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

delete_table_task = BigtableTableDeleteOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    task_id='delete_table_task',
)
delete_table_task2 = BigtableTableDeleteOperator(
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    task_id='delete_table_task2',
)

```

## BigtableTableWaitForReplicationSensor

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

Use the *BigtableTableWaitForReplicationSensor* to wait for the table to replicate fully.

The same arguments apply to this sensor as the *BigtableTableCreateOperator*.

**Note:** If the table or the Cloud Bigtable instance does not exist, this sensor waits for the table until timeout hits and does not raise any exception.

## Using the operator

```
wait_for_table_replication_task = BigtableTableWaitForReplicationSensor(
    project_id=GCP_PROJECT_ID,
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    poke_interval=int(CBT_POKE_INTERVAL),
    timeout=180,
    task_id='wait_for_table_replication_task',
)
wait_for_table_replication_task2 = BigtableTableWaitForReplicationSensor(
    instance_id=CBT_INSTANCE_ID,
    table_id=CBT_TABLE_ID,
    poke_interval=int(CBT_POKE_INTERVAL),
    timeout=180,
    task_id='wait_for_table_replication_task2',
)
```

### 3.6.4.6 Google Cloud Functions Operators

#### GcfFunctionDeleteOperator

Use the operator to delete a function from Google Cloud Functions.

For parameter definition, take a look at *GcfFunctionDeleteOperator*.

#### Arguments

The following examples of OS environment variables show how you can build function name to use in the operator:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_LOCATION = os.environ.get('GCP_LOCATION', 'europe-west1')
GCF_SHORT_FUNCTION_NAME = os.environ.get('GCF_SHORT_FUNCTION_NAME', 'hello').\
    replace("-", "_") # make sure there are no dashes in function name (!)
FUNCTION_NAME = 'projects/{}/locations/{}/functions/{}'.format(GCP_PROJECT_ID,
                                                                GCP_LOCATION,
                                                                GCF_SHORT_FUNCTION_
↪NAME)
```

## Using the operator

```
delete_task = GcfFunctionDeleteOperator(
    task_id="gcf_delete_task",
    name=FUNCTION_NAME
)
```

## Templating

```
template_fields = ('name', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Cloud Functions API documentation](#) for delete.

## GcfFunctionDeployOperator

Use the operator to deploy a function to Google Cloud Functions. If a function with this name already exists, it will be updated.

For parameter definition, take a look at *GcfFunctionDeployOperator*.

## Arguments

In the example DAG the following environment variables are used to parameterize the operator's definition:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_LOCATION = os.environ.get('GCP_LOCATION', 'europe-west1')
GCF_SHORT_FUNCTION_NAME = os.environ.get('GCF_SHORT_FUNCTION_NAME', 'hello').\
    replace("-", "_") # make sure there are no dashes in function name (!)
FUNCTION_NAME = 'projects/{}/locations/{}/functions/{}'.format(GCP_PROJECT_ID,
                                                                GCP_LOCATION,
                                                                GCF_SHORT_FUNCTION_
↪NAME)
```

```
GCF_SOURCE_ARCHIVE_URL = os.environ.get('GCF_SOURCE_ARCHIVE_URL', '')
GCF_SOURCE_UPLOAD_URL = os.environ.get('GCF_SOURCE_UPLOAD_URL', '')
GCF_SOURCE_REPOSITORY = os.environ.get(
    'GCF_SOURCE_REPOSITORY',
    'https://source.developers.google.com/'
    'projects/{}/repos/hello-world/moveable-aliases/master'.format(GCP_PROJECT_ID))
GCF_ZIP_PATH = os.environ.get('GCF_ZIP_PATH', '')
GCF_ENTRYPOINT = os.environ.get('GCF_ENTRYPOINT', 'helloWorld')
GCF_RUNTIME = 'nodejs6'
GCF_VALIDATE_BODY = os.environ.get('GCF_VALIDATE_BODY', True)
```

Some of those variables are used to create the request's body:

```
body = {
    "name": FUNCTION_NAME,
    "entryPoint": GCF_ENTRYPOINT,
    "runtime": GCF_RUNTIME,
    "httpsTrigger": {}
}
```

When a DAG is created, the `default_args` dictionary can be used to pass arguments common with other tasks:

```
default_args = {
    'start_date': dates.days_ago(1)
}
```

Note that neither the body nor the default args are complete in the above examples. Depending on the variables set, there might be different variants on how to pass source code related fields. Currently, you can pass either `sourceArchiveUrl`, `sourceRepository` or `sourceUploadUrl` as described in the [Cloud Functions API specification](#).

Additionally, `default_args` or direct operator args might contain `zip_path` parameter to run the extra step of uploading the source code before deploying it. In this case, you also need to provide an empty `sourceUploadUrl` parameter in the body.

## Using the operator

Depending on the combination of parameters, the Function's source code can be obtained from different sources:

```
if GCF_SOURCE_ARCHIVE_URL:
    body['sourceArchiveUrl'] = GCF_SOURCE_ARCHIVE_URL
elif GCF_SOURCE_REPOSITORY:
    body['sourceRepository'] = {
        'url': GCF_SOURCE_REPOSITORY
    }
elif GCF_ZIP_PATH:
    body['sourceUploadUrl'] = ''
    default_args['zip_path'] = GCF_ZIP_PATH
elif GCF_SOURCE_UPLOAD_URL:
    body['sourceUploadUrl'] = GCF_SOURCE_UPLOAD_URL
else:
    raise Exception("Please provide one of the source_code parameters")
```

The code to create the operator:

```
deploy_task = GcfFunctionDeployOperator(
    task_id="gcf_deploy_task",
    project_id=GCP_PROJECT_ID,
    location=GCP_LOCATION,
    body=body,
    validate_body=GCP_VALIDATE_BODY
)
```

You can also create the operator without project id - project id will be retrieved from the GCP connection used:

```
deploy2_task = GcfFunctionDeployOperator(
    task_id="gcf_deploy2_task",
    location=GCP_LOCATION,
    body=body,
    validate_body=GCP_VALIDATE_BODY
)
```

## Templating

```
template_fields = ('project_id', 'location', 'gcp_conn_id', 'api_version')
```

## Troubleshooting

If during the deploy you see an error similar to:

*“HttpError 403: Missing necessary permission iam.serviceAccounts.actAs for on resource project-name@appspot.gserviceaccount.com. Please grant the roles/iam.serviceAccountUser role.”*

it means that your service account does not have the correct Cloud IAM permissions.



1. Assign your Service Account the Cloud Functions Developer role.
2. Grant the user the Cloud IAM Service Account User role on the Cloud Functions runtime service account.

The typical way of assigning Cloud IAM permissions with *gcloud* is shown below. Just replace `PROJECT_ID` with ID of your Google Cloud Platform project and `SERVICE_ACCOUNT_EMAIL` with the email ID of your service account.

```
gcloud iam service-accounts add-iam-policy-binding \
  PROJECT_ID@appspot.gserviceaccount.com \
  --member="serviceAccount:[SERVICE_ACCOUNT_EMAIL]" \
  --role="roles/iam.serviceAccountUser"
```

You can also do that via the GCP Web console.

See [Adding the IAM service agent user role to the runtime service](#) for details.

If the source code for your function is in Google Source Repository, make sure that your service account has the Source Repository Viewer role so that the source code can be downloaded if necessary.

## More information

See [Google Cloud Functions API documentation](#) for create.

### 3.6.4.7 Google Cloud Spanner Operators

#### CloudSpannerInstanceDatabaseDeleteOperator

Deletes a database from the specified Cloud Spanner instance. If the database does not exist, no action is taken, and the operator succeeds.

For parameter definition, take a look at [CloudSpannerInstanceDatabaseDeleteOperator](#).

## Arguments

Some arguments in the example DAG are taken from environment variables.

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_SPANNER_INSTANCE_ID = os.environ.get('GCP_SPANNER_INSTANCE_ID', 'testinstance')
GCP_SPANNER_DATABASE_ID = os.environ.get('GCP_SPANNER_DATABASE_ID', 'testdatabase')
GCP_SPANNER_CONFIG_NAME = os.environ.get('GCP_SPANNER_CONFIG_NAME',
                                         'projects/example-project/instanceConfigs/
↳eur3')
GCP_SPANNER_NODE_COUNT = os.environ.get('GCP_SPANNER_NODE_COUNT', '1')
GCP_SPANNER_DISPLAY_NAME = os.environ.get('GCP_SPANNER_DISPLAY_NAME', 'Test Instance')
# OPERATION_ID should be unique per operation
OPERATION_ID = 'unique_operation_id'
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

spanner_database_delete_task = CloudSpannerInstanceDatabaseDeleteOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    task_id='spanner_database_delete_task'
)
spanner_database_delete_task2 = CloudSpannerInstanceDatabaseDeleteOperator(
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    task_id='spanner_database_delete_task2'
)

```

## Templating

```
template_fields = ('project_id', 'instance_id', 'gcp_conn_id')
```

## More information

See [Google Cloud Spanner API documentation](#) for database drop call.

## CloudSpannerInstanceDatabaseDeployOperator

Creates a new Cloud Spanner database in the specified instance, or if the desired database exists, assumes success with no changes applied to database configuration. No structure of the database is verified - it's enough if the database exists with the same name.

For parameter definition, take a look at [CloudSpannerInstanceDatabaseDeployOperator](#).

## Arguments

Some arguments in the example DAG are taken from environment variables.

```

GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_SPANNER_INSTANCE_ID = os.environ.get('GCP_SPANNER_INSTANCE_ID', 'testinstance')
GCP_SPANNER_DATABASE_ID = os.environ.get('GCP_SPANNER_DATABASE_ID', 'testdatabase')
GCP_SPANNER_CONFIG_NAME = os.environ.get('GCP_SPANNER_CONFIG_NAME',
                                         'projects/example-project/instanceConfigs/
↳eur3')
GCP_SPANNER_NODE_COUNT = os.environ.get('GCP_SPANNER_NODE_COUNT', '1')
GCP_SPANNER_DISPLAY_NAME = os.environ.get('GCP_SPANNER_DISPLAY_NAME', 'Test Instance')
# OPERATION_ID should be unique per operation
OPERATION_ID = 'unique_operation_id'

```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

spanner_database_deploy_task = CloudSpannerInstanceDatabaseDeployOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    ddl_statements=[
        "CREATE TABLE my_table1 (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
        "CREATE TABLE my_table2 (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
    ],
    task_id='spanner_database_deploy_task'
)
spanner_database_deploy_task2 = CloudSpannerInstanceDatabaseDeployOperator(
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    ddl_statements=[
        "CREATE TABLE my_table1 (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
        "CREATE TABLE my_table2 (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
    ],
    task_id='spanner_database_deploy_task2'
)

```

## Templating

```

template_fields = ('project_id', 'instance_id', 'database_id', 'ddl_statements',
                  'gcp_conn_id')
template_ext = ('.sql', )

```

## More information

See Google Cloud Spanner API documentation for [database create](#)

## CloudSpannerInstanceDatabaseUpdateOperator

Runs a DDL query in a Cloud Spanner database and allows you to modify the structure of an existing database.

You can optionally specify an `operation_id` parameter which simplifies determining whether the statements were executed in case the `update_database` call is replayed (idempotency check). The `operation_id` should be unique within the database, and must be a valid identifier: `[a-z][a-z0-9_]*`. More information can be found in the [documentation of updateDdl API](#)

For parameter definition take a look at *CloudSpannerInstanceDatabaseUpdateOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables.

```

GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_SPANNER_INSTANCE_ID = os.environ.get('GCP_SPANNER_INSTANCE_ID', 'testinstance')
GCP_SPANNER_DATABASE_ID = os.environ.get('GCP_SPANNER_DATABASE_ID', 'testdatabase')
GCP_SPANNER_CONFIG_NAME = os.environ.get('GCP_SPANNER_CONFIG_NAME',
                                         'projects/example-project/instanceConfigs/
→eur3')

```

(continues on next page)

(continued from previous page)

```
GCP_SPANNER_NODE_COUNT = os.environ.get('GCP_SPANNER_NODE_COUNT', '1')
GCP_SPANNER_DISPLAY_NAME = os.environ.get('GCP_SPANNER_DISPLAY_NAME', 'Test Instance')
# OPERATION_ID should be unique per operation
OPERATION_ID = 'unique_operation_id'
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
spanner_database_update_task = CloudSpannerInstanceDatabaseUpdateOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    ddl_statements=[
        "CREATE TABLE my_table3 (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
    ],
    task_id='spanner_database_update_task'
)
```

```
spanner_database_update_idempotent1_task = CloudSpannerInstanceDatabaseUpdateOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    operation_id=OPERATION_ID,
    ddl_statements=[
        "CREATE TABLE my_table_unique (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
    ],
    task_id='spanner_database_update_idempotent1_task'
)
spanner_database_update_idempotent2_task = CloudSpannerInstanceDatabaseUpdateOperator(
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    operation_id=OPERATION_ID,
    ddl_statements=[
        "CREATE TABLE my_table_unique (id INT64, name STRING(MAX)) PRIMARY KEY (id)",
    ],
    task_id='spanner_database_update_idempotent2_task'
)
```

## Templating

```
template_fields = ('project_id', 'instance_id', 'database_id', 'ddl_statements',
                  'gcp_conn_id')
template_ext = ('.sql', )
```

## More information

See Google Cloud Spanner API documentation for [database update\\_ddl](#).

## CloudSpannerInstanceDatabaseQueryOperator

Executes an arbitrary DML query (INSERT, UPDATE, DELETE).

For parameter definition take a look at *CloudSpannerInstanceDatabaseQueryOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables.

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_SPANNER_INSTANCE_ID = os.environ.get('GCP_SPANNER_INSTANCE_ID', 'testinstance')
GCP_SPANNER_DATABASE_ID = os.environ.get('GCP_SPANNER_DATABASE_ID', 'testdatabase')
GCP_SPANNER_CONFIG_NAME = os.environ.get('GCP_SPANNER_CONFIG_NAME',
                                         'projects/example-project/instanceConfigs/
→eur3')
GCP_SPANNER_NODE_COUNT = os.environ.get('GCP_SPANNER_NODE_COUNT', '1')
GCP_SPANNER_DISPLAY_NAME = os.environ.get('GCP_SPANNER_DISPLAY_NAME', 'Test Instance')
# OPERATION_ID should be unique per operation
OPERATION_ID = 'unique_operation_id'
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
spanner_instance_query_task = CloudSpannerInstanceDatabaseQueryOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    query=["DELETE FROM my_table2 WHERE true"],
    task_id='spanner_instance_query_task'
)
spanner_instance_query_task2 = CloudSpannerInstanceDatabaseQueryOperator(
    instance_id=GCP_SPANNER_INSTANCE_ID,
    database_id=GCP_SPANNER_DATABASE_ID,
    query=["DELETE FROM my_table2 WHERE true"],
    task_id='spanner_instance_query_task2'
)
```

## Templating

```
template_fields = ('project_id', 'instance_id', 'database_id', 'query', 'gcp_conn_id')
template_ext = ('.sql',)
```

## More information

See Google Cloud Spanner API documentation for [the DML syntax](#).

## CloudSpannerInstanceDeleteOperator

Deletes a Cloud Spanner instance. If an instance does not exist, no action is taken, and the operator succeeds.

For parameter definition take a look at *CloudSpannerInstanceDeleteOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_SPANNER_INSTANCE_ID = os.environ.get('GCP_SPANNER_INSTANCE_ID', 'testinstance')
GCP_SPANNER_DATABASE_ID = os.environ.get('GCP_SPANNER_DATABASE_ID', 'testdatabase')
GCP_SPANNER_CONFIG_NAME = os.environ.get('GCP_SPANNER_CONFIG_NAME',
                                         'projects/example-project/instanceConfigs/
→eur3')
GCP_SPANNER_NODE_COUNT = os.environ.get('GCP_SPANNER_NODE_COUNT', '1')
GCP_SPANNER_DISPLAY_NAME = os.environ.get('GCP_SPANNER_DISPLAY_NAME', 'Test Instance')
# OPERATION_ID should be unique per operation
OPERATION_ID = 'unique_operation_id'
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
spanner_instance_delete_task = CloudSpannerInstanceDeleteOperator(
    project_id=GCP_PROJECT_ID,
    instance_id=GCP_SPANNER_INSTANCE_ID,
    task_id='spanner_instance_delete_task'
)
spanner_instance_delete_task2 = CloudSpannerInstanceDeleteOperator(
    instance_id=GCP_SPANNER_INSTANCE_ID,
    task_id='spanner_instance_delete_task2'
)
```

## Templating

```
template_fields = ('project_id', 'instance_id', 'gcp_conn_id')
```

## More information

See [Google Cloud Spanner API documentation](#) for instance delete.

### 3.6.4.8 Google Cloud Sql Operators

#### CloudSqlInstanceDatabaseCreateOperator

Creates a new database inside a Cloud SQL instance.

For parameter definition, take a look at *CloudSqlInstanceDatabaseCreateOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_db_create_task = CloudSqlInstanceDatabaseCreateOperator(
    project_id=GCP_PROJECT_ID,
    body=db_create_body,
    instance=INSTANCE_NAME,
    task_id='sql_db_create_task'
)
sql_db_create_task2 = CloudSqlInstanceDatabaseCreateOperator(
    body=db_create_body,
    instance=INSTANCE_NAME,
    task_id='sql_db_create_task2'
)
```

Example request body:

```
db_create_body = {
    "instance": INSTANCE_NAME,
    "name": DB_NAME,
    "project": GCP_PROJECT_ID
}
```

## Templating

```
template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Cloud SQL API documentation](#) for database insert.

## CloudSqlInstanceDatabaseDeleteOperator

Deletes a database from a Cloud SQL instance.

For parameter definition, take a look at *CloudSqlInstanceDatabaseDeleteOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_db_delete_task = CloudSqlInstanceDatabaseDeleteOperator(
    project_id=GCP_PROJECT_ID,
    instance=INSTANCE_NAME,
    database=DB_NAME,
    task_id='sql_db_delete_task'
)
sql_db_delete_task2 = CloudSqlInstanceDatabaseDeleteOperator(
    instance=INSTANCE_NAME,
    database=DB_NAME,
    task_id='sql_db_delete_task2'
)
```

## Templating

```
template_fields = ('project_id', 'instance', 'database', 'gcp_conn_id',
                  'api_version')
```

## More information

See [Google Cloud SQL API documentation](https://cloud.google.com/sql/docs/mysql/admin-api/how-tos/performance#patch) for database delete.

## CloudSqlInstanceDatabasePatchOperator

Updates a resource containing information about a database inside a Cloud SQL instance using patch semantics. See: <https://cloud.google.com/sql/docs/mysql/admin-api/how-tos/performance#patch>

For parameter definition, take a look at *CloudSqlInstanceDatabasePatchOperator*.

## Arguments

Some arguments in the example DAG are taken from environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```



## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_db_patch_task = CloudSqlInstanceDatabasePatchOperator(
    project_id=GCP_PROJECT_ID,
    body=db_patch_body,
    instance=INSTANCE_NAME,
    database=DB_NAME,
    task_id='sql_db_patch_task'
)
sql_db_patch_task2 = CloudSqlInstanceDatabasePatchOperator(
    body=db_patch_body,
    instance=INSTANCE_NAME,
    database=DB_NAME,
    task_id='sql_db_patch_task2'
)
```

Example request body:

```
db_patch_body = {
    "charset": "utf16",
    "collation": "utf16_general_ci"
}
```

## Templating

```
template_fields = ('project_id', 'instance', 'database', 'gcp_conn_id',
                  'api_version')
```

## More information

See [Google Cloud SQL API documentation](#) for database patch.

## CloudSqlInstanceDeleteOperator

Deletes a Cloud SQL instance in Google Cloud Platform.

For parameter definition, take a look at [CloudSqlInstanceDeleteOperator](#).

## Arguments

Some arguments in the example DAG are taken from OS environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_instance_delete_task = CloudSqlInstanceDeleteOperator(
    project_id=GCP_PROJECT_ID,
    instance=INSTANCE_NAME,
    task_id='sql_instance_delete_task'
)
sql_instance_delete_task2 = CloudSqlInstanceDeleteOperator(
    instance=INSTANCE_NAME2,
    task_id='sql_instance_delete_task2'
)
```

## Templating

```
template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Cloud SQL API documentation](#) for delete.

## CloudSqlInstanceExportOperator

Exports data from a Cloud SQL instance to a Cloud Storage bucket as a SQL dump or CSV file.

---

**Note:** This operator is idempotent. If executed multiple times with the same export file URI, the export file in GCS will simply be overridden.

---

For parameter definition take a look at *CloudSqlInstanceExportOperator*.

## Arguments

Some arguments in the example DAG are taken from Airflow variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

```
EXPORT_URI = os.environ.get('GCSQL_MYSQL_EXPORT_URI', 'gs://bucketName/fileName')
IMPORT_URI = os.environ.get('GCSQL_MYSQL_IMPORT_URI', 'gs://bucketName/fileName')
```

Example body defining the export operation:

```
export_body = {
    "exportContext": {
        "fileType": "sql",
        "uri": EXPORT_URI,
        "sqlExportOptions": {
            "schemaOnly": False
        }
    }
}
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_export_task = CloudSqlInstanceExportOperator(
    project_id=GCP_PROJECT_ID,
    body=export_body,
    instance=INSTANCE_NAME,
    task_id='sql_export_task'
)
sql_export_task2 = CloudSqlInstanceExportOperator(
    body=export_body,
    instance=INSTANCE_NAME,
    task_id='sql_export_task2'
)
```

## Templating

```
template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Cloud SQL API documentation](#) for export.

## Troubleshooting

If you receive an “Unauthorized” error in GCP, make sure that the service account of the Cloud SQL instance is authorized to write to the selected GCS bucket.

It is not the service account configured in Airflow that communicates with GCS, but rather the service account of the particular Cloud SQL instance.

To grant the service account with the appropriate WRITE permissions for the GCS bucket you can use the *GoogleCloudStorageBucketCreateAclEntryOperator*, as shown in the example:

```
sql_gcp_add_bucket_permission_task = GoogleCloudStorageBucketCreateAclEntryOperator(
    entity="user-{{ task_instance.xcom_pull(
        \"sql_instance_create_task\", key='service_account_email') \"
    }}",
```

(continues on next page)

(continued from previous page)

```

    role="WRITER",
    bucket=export_url_split[1], # netloc (bucket)
    task_id='sql_gcp_add_bucket_permission_task'
)

```

## CloudSqlInstanceImportOperator

Imports data into a Cloud SQL instance from a SQL dump or CSV file in Cloud Storage.

### CSV import:

This operator is NOT idempotent for a CSV import. If the same file is imported multiple times, the imported data will be duplicated in the database. Moreover, if there are any unique constraints the duplicate import may result in an error.

### SQL import:

This operator is idempotent for a SQL import if it was also exported by Cloud SQL. The exported SQL contains 'DROP TABLE IF EXISTS' statements for all tables to be imported.

If the import file was generated in a different way, idempotence is not guaranteed. It has to be ensured on the SQL file level.

For parameter definition take a look at *CloudSqlInstanceImportOperator*.

## Arguments

Some arguments in the example DAG are taken from Airflow variables:

```

GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')

```

```

EXPORT_URI = os.environ.get('GCSQL_MYSQL_EXPORT_URI', 'gs://bucketName/fileName')
IMPORT_URI = os.environ.get('GCSQL_MYSQL_IMPORT_URI', 'gs://bucketName/fileName')

```

Example body defining the import operation:

```

import_body = {
    "importContext": {
        "fileType": "sql",
        "uri": IMPORT_URI
    }
}

```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

sql_import_task = CloudSqlInstanceImportOperator(
    project_id=GCP_PROJECT_ID,
    body=import_body,
    instance=INSTANCE_NAME2,
    task_id='sql_import_task'
)
sql_import_task2 = CloudSqlInstanceImportOperator(
    body=import_body,
    instance=INSTANCE_NAME2,
    task_id='sql_import_task2'
)

```

## Templating

```

template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')

```

## More information

See [Google Cloud SQL API documentation](#) for import.

## Troubleshooting

If you receive an “Unauthorized” error in GCP, make sure that the service account of the Cloud SQL instance is authorized to read from the selected GCS object.

It is not the service account configured in Airflow that communicates with GCS, but rather the service account of the particular Cloud SQL instance.

To grant the service account with the appropriate READ permissions for the GCS object you can use the *GoogleCloudStorageObjectCreateAclEntryOperator*, as shown in the example:

```

sql_gcp_add_object_permission_task = GoogleCloudStorageObjectCreateAclEntryOperator(
    entity="user-{{ task_instance.xcom_pull("
        "sql_instance_create_task2", key='service_account_email') "
        " }}",
    role="READER",
    bucket=import_url_split[1], # netloc (bucket)
    object_name=import_url_split[2][1:], # path (strip first '/')
    task_id='sql_gcp_add_object_permission_task',
)
prev_task = next_dep(sql_gcp_add_object_permission_task, prev_task)

# For import to work we also need to add the Cloud SQL instance's Service Account
# write access to the whole bucket!.
sql_gcp_add_bucket_permission_2_task = GoogleCloudStorageBucketCreateAclEntryOperator(
    entity="user-{{ task_instance.xcom_pull("
        "sql_instance_create_task2", key='service_account_email') "
        " }}",
    role="WRITER",
    bucket=import_url_split[1], # netloc
    task_id='sql_gcp_add_bucket_permission_2_task',
)

```

## CloudSqlInstanceCreateOperator

Creates a new Cloud SQL instance in Google Cloud Platform.

For parameter definition, take a look at *CloudSqlInstanceCreateOperator*.

If an instance with the same name exists, no action will be taken and the operator will succeed.

## Arguments

Some arguments in the example DAG are taken from OS environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

Example body defining the instance:

```
body = {
    "name": INSTANCE_NAME,
    "settings": {
        "tier": "db-n1-standard-1",
        "backupConfiguration": {
            "binaryLogEnabled": True,
            "enabled": True,
            "startTime": "05:00"
        },
        "activationPolicy": "ALWAYS",
        "dataDiskSizeGb": 30,
        "dataDiskType": "PD_SSD",
        "databaseFlags": [],
        "ipConfiguration": {
            "ipv4Enabled": True,
            "requireSsl": True,
        },
        "locationPreference": {
            "zone": "europe-west4-a"
        },
        "maintenanceWindow": {
            "hour": 5,
            "day": 7,
            "updateTrack": "canary"
        },
        "pricingPlan": "PER_USE",
        "replicationType": "ASYNCHRONOUS",
        "storageAutoResize": False,
        "storageAutoResizeLimit": 0,
        "userLabels": {
            "my-key": "my-value"
        }
    },
    "databaseVersion": "MYSQL_5_7",
    "region": "europe-west4",
}
```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```
sql_instance_create_task = CloudSqlInstanceCreateOperator(
    project_id=GCP_PROJECT_ID,
    body=body,
    instance=INSTANCE_NAME,
    task_id='sql_instance_create_task'
)
```

## Templating

```
template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')
```

## More information

See [Google Cloud SQL API documentation](#) for insert.

## CloudSqlInstancePatchOperator

Updates settings of a Cloud SQL instance in Google Cloud Platform (partial update).

For parameter definition, take a look at [CloudSqlInstancePatchOperator](#).

This is a partial update, so only values for the settings specified in the body will be set / updated. The rest of the existing instance's configuration will remain unchanged.

## Arguments

Some arguments in the example DAG are taken from OS environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
INSTANCE_NAME = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME', 'test-mysql')
INSTANCE_NAME2 = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME2', 'test-mysql2')
DB_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'testdb')
```

Example body defining the instance:

```
patch_body = {
    "name": INSTANCE_NAME,
    "settings": {
        "dataDiskSizeGb": 35,
        "maintenanceWindow": {
            "hour": 3,
            "day": 6,
            "updateTrack": "canary"
        },
        "userLabels": {
            "my-key-patch": "my-value-patch"
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

```

## Using the operator

You can create the operator with or without project id. If project id is missing it will be retrieved from the GCP connection used. Both variants are shown:

```

sql_instance_patch_task = CloudSqlInstancePatchOperator(
    project_id=GCP_PROJECT_ID,
    body=patch_body,
    instance=INSTANCE_NAME,
    task_id='sql_instance_patch_task'
)

sql_instance_patch_task2 = CloudSqlInstancePatchOperator(
    body=patch_body,
    instance=INSTANCE_NAME,
    task_id='sql_instance_patch_task2'
)

```

## Templating

```

template_fields = ('project_id', 'instance', 'gcp_conn_id', 'api_version')

```

## More information

See [Google Cloud SQL API documentation](#) for patch.

## CloudSqlQueryOperator

Performs DDL or DML SQL queries in Google Cloud SQL instance. The DQL (retrieving data from Google Cloud SQL) is not supported. You might run the SELECT queries, but the results of those queries are discarded.

You can specify various connectivity methods to connect to running instance, starting from public IP plain connection through public IP with SSL or both TCP and socket connection via Cloud SQL Proxy. The proxy is downloaded and started/stopped dynamically as needed by the operator.

There is a *gcpcloudsql://* connection type that you should use to define what kind of connectivity you want the operator to use. The connection is a “meta” type of connection. It is not used to make an actual connectivity on its own, but it determines whether Cloud SQL Proxy should be started by *CloudSqlDatabaseHook* and what kind of database connection (Postgres or MySQL) should be created dynamically to connect to Cloud SQL via public IP address or via the proxy. The ‘CloudSqlDatabaseHook’ uses *CloudSqlProxyRunner* to manage Cloud SQL Proxy lifecycle (each task has its own Cloud SQL Proxy)

When you build connection, you should use connection parameters as described in [CloudSqlDatabaseHook](#). You can see examples of connections below for all the possible types of connectivity. Such connection can be reused between different tasks (instances of *CloudSqlQueryOperator*). Each task will get their own proxy started if needed with their own TCP or UNIX socket.



For parameter definition, take a look at [CloudSqlQueryOperator](#).

Since query operator can run arbitrary query, it cannot be guaranteed to be idempotent. SQL query designer should design the queries to be idempotent. For example, both Postgres and MySQL support CREATE TABLE IF NOT EXISTS statements that can be used to create tables in an idempotent way.

## Arguments

If you define connection via `AIRFLOW_CONN_*` URL defined in an environment variable, make sure the URL components in the URL are URL-encoded. See examples below for details.

Note that in case of SSL connections you need to have a mechanism to make the certificate/key files available in predefined locations for all the workers on which the operator can run. This can be provided for example by mounting NFS-like volumes in the same path for all the workers.

Some arguments in the example DAG are taken from the OS environment variables:

```
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', 'example-project')
GCP_REGION = os.environ.get('GCP_REGION', 'europe-west-1b')

GCSQL_POSTGRES_INSTANCE_NAME_QUERY = os.environ.get(
    'GCSQL_POSTGRES_INSTANCE_NAME_QUERY',
    'testpostgres')
GCSQL_POSTGRES_DATABASE_NAME = os.environ.get('GCSQL_POSTGRES_DATABASE_NAME',
    'postgresdb')
GCSQL_POSTGRES_USER = os.environ.get('GCSQL_POSTGRES_USER', 'postgres_user')
GCSQL_POSTGRES_PASSWORD = os.environ.get('GCSQL_POSTGRES_PASSWORD', 'password')
GCSQL_POSTGRES_PUBLIC_IP = os.environ.get('GCSQL_POSTGRES_PUBLIC_IP', '0.0.0.0')
GCSQL_POSTGRES_PUBLIC_PORT = os.environ.get('GCSQL_POSTGRES_PUBLIC_PORT', 5432)
GCSQL_POSTGRES_CLIENT_CERT_FILE = os.environ.get('GCSQL_POSTGRES_CLIENT_CERT_FILE',
    ".key/postgres-client-cert.pem")
GCSQL_POSTGRES_CLIENT_KEY_FILE = os.environ.get('GCSQL_POSTGRES_CLIENT_KEY_FILE',
    ".key/postgres-client-key.pem")
GCSQL_POSTGRES_SERVER_CA_FILE = os.environ.get('GCSQL_POSTGRES_SERVER_CA_FILE',
    ".key/postgres-server-ca.pem")

GCSQL_MYSQL_INSTANCE_NAME_QUERY = os.environ.get('GCSQL_MYSQL_INSTANCE_NAME_QUERY',
    'testmysql')
GCSQL_MYSQL_DATABASE_NAME = os.environ.get('GCSQL_MYSQL_DATABASE_NAME', 'mysqldb')
GCSQL_MYSQL_USER = os.environ.get('GCSQL_MYSQL_USER', 'mysql_user')
GCSQL_MYSQL_PASSWORD = os.environ.get('GCSQL_MYSQL_PASSWORD', 'password')
GCSQL_MYSQL_PUBLIC_IP = os.environ.get('GCSQL_MYSQL_PUBLIC_IP', '0.0.0.0')
GCSQL_MYSQL_PUBLIC_PORT = os.environ.get('GCSQL_MYSQL_PUBLIC_PORT', 3306)
GCSQL_MYSQL_CLIENT_CERT_FILE = os.environ.get('GCSQL_MYSQL_CLIENT_CERT_FILE',
    ".key/mysql-client-cert.pem")
GCSQL_MYSQL_CLIENT_KEY_FILE = os.environ.get('GCSQL_MYSQL_CLIENT_KEY_FILE',
    ".key/mysql-client-key.pem")
GCSQL_MYSQL_SERVER_CA_FILE = os.environ.get('GCSQL_MYSQL_SERVER_CA_FILE',
    ".key/mysql-server-ca.pem")

SQL = [
    'CREATE TABLE IF NOT EXISTS TABLE_TEST (I INTEGER)',
    'CREATE TABLE IF NOT EXISTS TABLE_TEST (I INTEGER)', # shows warnings logged
    'INSERT INTO TABLE_TEST VALUES (0)',
    'CREATE TABLE IF NOT EXISTS TABLE_TEST2 (I INTEGER)',
    'DROP TABLE TABLE_TEST',
```

(continues on next page)

(continued from previous page)

```
'DROP TABLE TABLE_TEST2',
]
```

Example connection definitions for all connectivity cases. Note that all the components of the connection URI should be URL-encoded:

```
HOME_DIR = expanduser("~")

def get_absolute_path(path):
    if path.startswith("/"):
        return path
    else:
        return os.path.join(HOME_DIR, path)

postgres_kwargs = dict(
    user=quote_plus(GCSQL_POSTGRES_USER),
    password=quote_plus(GCSQL_POSTGRES_PASSWORD),
    public_port=GCSQL_POSTGRES_PUBLIC_PORT,
    public_ip=quote_plus(GCSQL_POSTGRES_PUBLIC_IP),
    project_id=quote_plus(GCP_PROJECT_ID),
    location=quote_plus(GCP_REGION),
    instance=quote_plus(GCSQL_POSTGRES_INSTANCE_NAME_QUERY),
    database=quote_plus(GCSQL_POSTGRES_DATABASE_NAME),
    client_cert_file=quote_plus(get_absolute_path(GCSQL_POSTGRES_CLIENT_CERT_FILE)),
    client_key_file=quote_plus(get_absolute_path(GCSQL_POSTGRES_CLIENT_KEY_FILE)),
    server_ca_file=quote_plus(get_absolute_path(GCSQL_POSTGRES_SERVER_CA_FILE))
)

# The connections below are created using one of the standard approaches - via
# environment
# variables named AIRFLOW_CONN_*. The connections can also be created in the database
# of AIRFLOW (using command line or UI).

# Postgres: connect via proxy over TCP
os.environ['AIRFLOW_CONN_PROXY_POSTGRES_TCP'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=postgres&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=True&" \
    "sql_proxy_use_tcp=True".format(**postgres_kwargs)

# Postgres: connect via proxy over UNIX socket (specific proxy version)
os.environ['AIRFLOW_CONN_PROXY_POSTGRES_SOCKET'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=postgres&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=True&" \
    "sql_proxy_version=v1.13&" \
```

(continues on next page)

(continued from previous page)

```

    "sql_proxy_use_tcp=False".format(**postgres_kwargs)

# Postgres: connect directly via TCP (non-SSL)
os.environ['AIRFLOW_CONN_PUBLIC_POSTGRES_TCP'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=postgres&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=False&" \
    "use_ssl=False".format(**postgres_kwargs)

# Postgres: connect directly via TCP (SSL)
os.environ['AIRFLOW_CONN_PUBLIC_POSTGRES_TCP_SSL'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=postgres&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=False&" \
    "use_ssl=True&" \
    "sslcert={client_cert_file}&" \
    "sslkey={client_key_file}&" \
    "sslrootcert={server_ca_file}" \
    .format(**postgres_kwargs)

mysql_kwargs = dict(
    user=quote_plus(GCSQL_MYSQL_USER),
    password=quote_plus(GCSQL_MYSQL_PASSWORD),
    public_port=GCSQL_MYSQL_PUBLIC_PORT,
    public_ip=quote_plus(GCSQL_MYSQL_PUBLIC_IP),
    project_id=quote_plus(GCP_PROJECT_ID),
    location=quote_plus(GCP_REGION),
    instance=quote_plus(GCSQL_MYSQL_INSTANCE_NAME_QUERY),
    database=quote_plus(GCSQL_MYSQL_DATABASE_NAME),
    client_cert_file=quote_plus(get_absolute_path(GCSQL_MYSQL_CLIENT_CERT_FILE)),
    client_key_file=quote_plus(get_absolute_path(GCSQL_MYSQL_CLIENT_KEY_FILE)),
    server_ca_file=quote_plus(get_absolute_path(GCSQL_MYSQL_SERVER_CA_FILE))
)

# MySQL: connect via proxy over TCP (specific proxy version)
os.environ['AIRFLOW_CONN_PROXY_MYSQL_TCP'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=mysql&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=True&" \
    "sql_proxy_version=v1.13&" \
    "sql_proxy_use_tcp=True".format(**mysql_kwargs)

# MySQL: connect via proxy over UNIX socket using pre-downloaded Cloud Sql Proxy_
↪binary
try:
    sql_proxy_binary_path = subprocess.check_output(
        ['which', 'cloud_sql_proxy']).decode('utf-8').rstrip()
except subprocess.CalledProcessError:

```

(continues on next page)

(continued from previous page)

```

sql_proxy_binary_path = "/tmp/anyhow_download_cloud_sql_proxy"

os.environ['AIRFLOW_CONN_PROXY_MYSQL_SOCKET'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=mysql&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=True&" \
    "sql_proxy_binary_path={sql_proxy_binary_path}&" \
    "sql_proxy_use_tcp=False".format(
        sql_proxy_binary_path=quote_plus(sql_proxy_binary_path), **mysql_kwargs)

# MySQL: connect directly via TCP (non-SSL)
os.environ['AIRFLOW_CONN_PUBLIC_MYSQL_TCP'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=mysql&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=False&" \
    "use_ssl=False".format(**mysql_kwargs)

# MySQL: connect directly via TCP (SSL) and with fixed Cloud Sql Proxy binary path
os.environ['AIRFLOW_CONN_PUBLIC_MYSQL_TCP_SSL'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=mysql&" \
    "project_id={project_id}&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=False&" \
    "use_ssl=True&" \
    "sslcert={client_cert_file}&" \
    "sslkey={client_key_file}&" \
    "sslrootcert={server_ca_file}".format(**mysql_kwargs)

# Special case: MySQL: connect directly via TCP (SSL) and with fixed Cloud Sql
# Proxy binary path AND with missing project_id
os.environ['AIRFLOW_CONN_PUBLIC_MYSQL_TCP_SSL_NO_PROJECT_ID'] = \
    "gcpcloudsql://{user}:{password}@{public_ip}:{public_port}/{database}?" \
    "database_type=mysql&" \
    "location={location}&" \
    "instance={instance}&" \
    "use_proxy=False&" \
    "use_ssl=True&" \
    "sslcert={client_cert_file}&" \
    "sslkey={client_key_file}&" \
    "sslrootcert={server_ca_file}".format(**mysql_kwargs)

```

## Using the operator

Example operators below are using all connectivity options. Note connection id from the operator matches the *AIRFLOW\_CONN\_\** postfix uppercase. This is standard AIRFLOW notation for defining connection via environment variables):

```
connection_names = [
    "proxy_postgres_tcp",
    "proxy_postgres_socket",
    "public_postgres_tcp",
    "public_postgres_tcp_ssl",
    "proxy_mysql_tcp",
    "proxy_mysql_socket",
    "public_mysql_tcp",
    "public_mysql_tcp_ssl",
    "public_mysql_tcp_ssl_no_project_id"
]

tasks = []

with models.DAG(
    dag_id='example_gcp_sql_query',
    default_args=default_args,
    schedule_interval=None
) as dag:
    prev_task = None

    for connection_name in connection_names:
        task = CloudSqlQueryOperator(
            gcp_cloudsql_conn_id=connection_name,
            task_id="example_gcp_sql_task_" + connection_name,
            sql=SQL
        )
        tasks.append(task)
        if prev_task:
            prev_task >> task
        prev_task = task
```

## Templating

```
template_fields = ('sql', 'gcp_cloudsql_conn_id', 'gcp_conn_id')
template_ext = ('.sql',)
```

## More information

See [Google Cloud SQL Proxy documentation](#).

### 3.6.4.9 Google Cloud Storage Operators

## GoogleCloudStorageBucketCreateAclEntryOperator

Creates a new ACL entry on the specified bucket.

For parameter definition, take a look at *GoogleCloudStorageBucketCreateAclEntryOperator*

### Arguments

Some arguments in the example DAG are taken from the OS environment variables:

```
GCS_ACL_BUCKET = os.environ.get('GCS_ACL_BUCKET', 'example-bucket')
GCS_ACL_OBJECT = os.environ.get('GCS_ACL_OBJECT', 'example-object')
GCS_ACL_ENTITY = os.environ.get('GCS_ACL_ENTITY', 'example-entity')
GCS_ACL_BUCKET_ROLE = os.environ.get('GCS_ACL_BUCKET_ROLE', 'example-bucket-role')
GCS_ACL_OBJECT_ROLE = os.environ.get('GCS_ACL_OBJECT_ROLE', 'example-object-role')
```

### Using the operator

```
gcs_bucket_create_acl_entry_task = GoogleCloudStorageBucketCreateAclEntryOperator(
    bucket=GCS_ACL_BUCKET,
    entity=GCS_ACL_ENTITY,
    role=GCS_ACL_BUCKET_ROLE,
    task_id="gcs_bucket_create_acl_entry_task"
)
```

### Templating

```
template_fields = ('bucket', 'entity', 'role', 'user_project')
```

### More information

See [Google Cloud Storage BucketAccessControls](#) insert documentation.

## GoogleCloudStorageObjectCreateAclEntryOperator

Creates a new ACL entry on the specified object.

For parameter definition, take a look at *GoogleCloudStorageObjectCreateAclEntryOperator*

### Arguments

Some arguments in the example DAG are taken from the OS environment variables:

```
GCS_ACL_BUCKET = os.environ.get('GCS_ACL_BUCKET', 'example-bucket')
GCS_ACL_OBJECT = os.environ.get('GCS_ACL_OBJECT', 'example-object')
GCS_ACL_ENTITY = os.environ.get('GCS_ACL_ENTITY', 'example-entity')
GCS_ACL_BUCKET_ROLE = os.environ.get('GCS_ACL_BUCKET_ROLE', 'example-bucket-role')
GCS_ACL_OBJECT_ROLE = os.environ.get('GCS_ACL_OBJECT_ROLE', 'example-object-role')
```

## Using the operator

```
gcs_object_create_acl_entry_task = GoogleCloudStorageObjectCreateAclEntryOperator(
    bucket=GCS_ACL_BUCKET,
    object_name=GCS_ACL_OBJECT,
    entity=GCS_ACL_ENTITY,
    role=GCS_ACL_OBJECT_ROLE,
    task_id="gcs_object_create_acl_entry_task"
)
```

## Templating

```
template_fields = ('bucket', 'object_name', 'entity', 'role', 'generation',
                  'user_project')
```

## More information

See [Google Cloud Storage ObjectAccessControls insert documentation](#).

## 3.6.5 Managing Connections

Airflow needs to know how to connect to your environment. Information such as hostname, port, login and passwords to other systems and services is handled in the Admin->Connections section of the UI. The pipeline code you will author will reference the 'conn\_id' of the Connection objects.

The screenshot shows the Airflow web interface. The top navigation bar includes links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The 'Admin' dropdown menu is open, showing options like Pools, Configuration, Users, Connections (which is highlighted), Variables, and XComs. Below the navigation, the 'Connections' section is displayed. It features a 'List (30)' button, a 'Create' link, and a 'With selected' dropdown. A table lists existing connections:

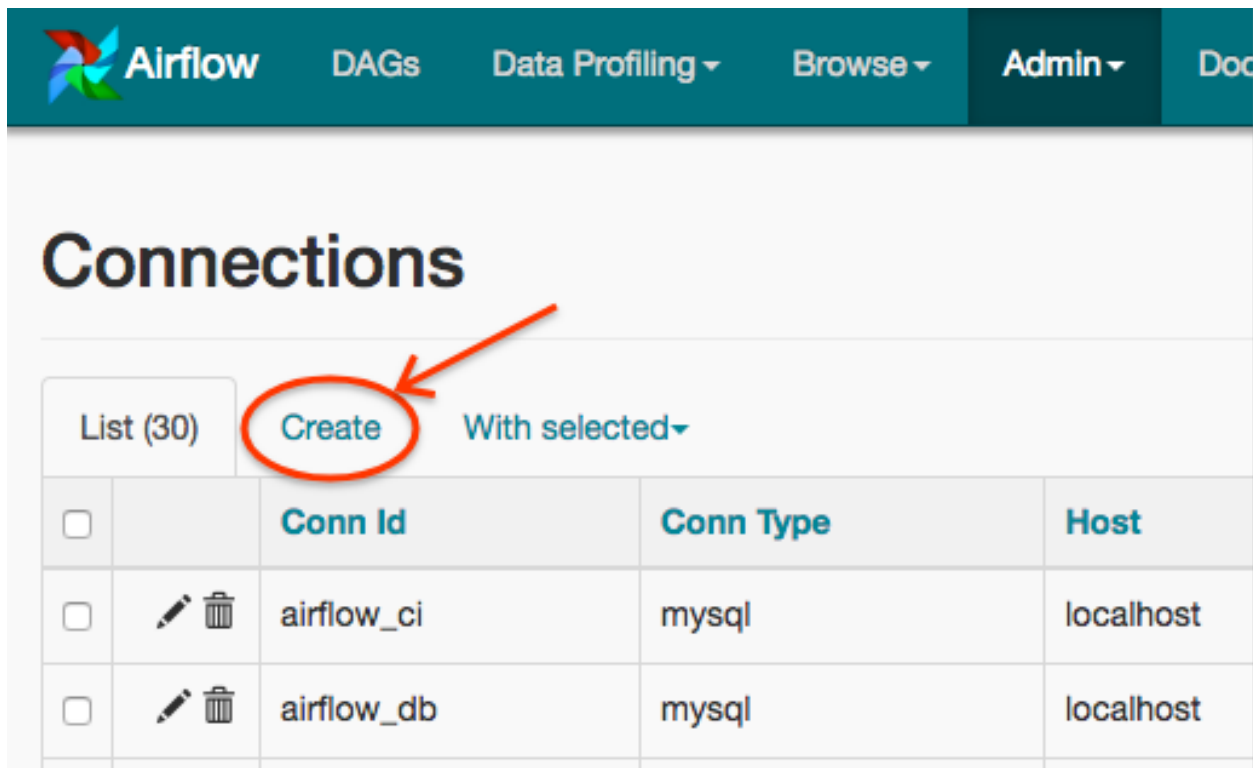
	Conn Id	Conn Type	Host
<input type="checkbox"/>	airflow_ci	mysql	localhost
<input type="checkbox"/>	airflow_db	mysql	localhost

Connections can be created and managed using either the UI or environment variables.

See the [Connections Concepts](#) documentation for more information.

### 3.6.5.1 Creating a Connection with the UI

Open the Admin->Connections section of the UI. Click the `Create` link to create a new connection.

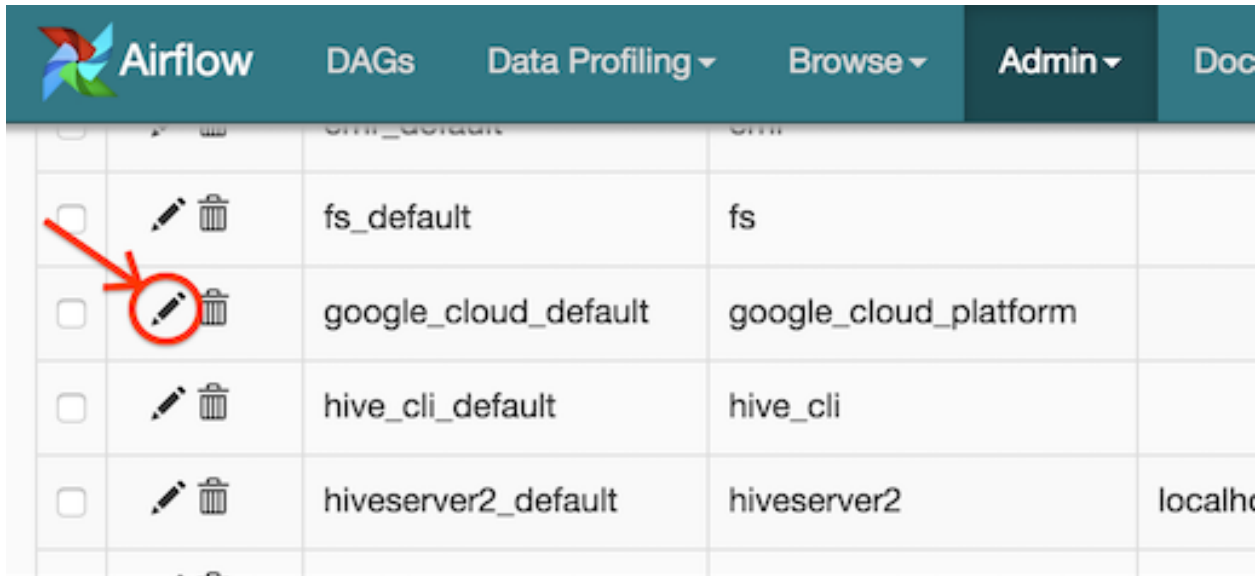










1. Fill in the `Conn Id` field with the desired connection ID. It is recommended that you use lower-case characters and separate words with underscores.
2. Choose the connection type with the `Conn Type` field.
3. Fill in the remaining fields. See [Connection Types](#) for a description of the fields belonging to the different connection types.
4. Click the `Save` button to create the connection.

### 3.6.5.2 Editing a Connection with the UI

Open the `Admin->Connections` section of the UI. Click the pencil icon next to the connection you wish to edit in the connection list.





<input type="checkbox"/>	 	fs_default	fs	
<input type="checkbox"/>	 	google_cloud_default	google_cloud_platform	
<input type="checkbox"/>	 	hive_cli_default	hive_cli	
<input type="checkbox"/>	 	hiveserver2_default	hiveserver2	localho

Modify the connection properties and click the `Save` button to save your changes.

### 3.6.5.3 Creating a Connection with Environment Variables

Connections in Airflow pipelines can be created using environment variables. The environment variable needs to have a prefix of `AIRFLOW_CONN_` for Airflow with the value in a URI format to use the connection properly.

When referencing the connection in the Airflow pipeline, the `conn_id` should be the name of the variable without the prefix. For example, if the `conn_id` is named `postgres_master` the environment variable should be named `AIRFLOW_CONN_POSTGRES_MASTER` (note that the environment variable must be all uppercase). Airflow assumes the value returned from the environment variable to be in a URI format (e.g. `postgres://user:password@localhost:5432/master` or `s3://accesskey:secretkey@S3`).

### 3.6.5.4 Connection Types

#### Google Cloud Platform

The Google Cloud Platform connection type enables the *GCP Integrations*.

#### Authenticating to GCP

There are two ways to connect to GCP using Airflow.

1. Use [Application Default Credentials](#), such as via the metadata server when running on Google Compute Engine.
2. Use a [service account](#) key file (JSON format) on disk.

#### Default Connection IDs

The following connection IDs are used by default.

**`bigquery_default`** Used by the *BigQueryHook* hook.

**`google_cloud_datastore_default`** Used by the *DatastoreHook* hook.

**google\_cloud\_default** Used by those hooks:

- *GoogleCloudBaseHook*
- *DataFlowHook*
- *DataProcHook*
- *MLEngineHook*
- *GoogleCloudStorageHook*
- *BigtableHook*
- *GceHook*
- *GcfHook*
- *CloudSpannerHook*
- *CloudSqlHook*

## Configuring the Connection

**Project Id (optional)** The Google Cloud project ID to connect to. It is used as default project id by operators using it and can usually be overridden at the operator level.

**Keyfile Path** Path to a [service account](#) key file (JSON format) on disk.

Not required if using application default credentials.

**Keyfile JSON** Contents of a [service account](#) key file (JSON format) on disk. It is recommended to [Secure your connections](#) if using this method to authenticate.

Not required if using application default credentials.

**Scopes (comma separated)** A list of comma-separated [Google Cloud scopes](#) to authenticate with.

---

**Note:** Scopes are ignored when using application default credentials. See issue [AIRFLOW-2522](#).

---

## Amazon Web Services

The Amazon Web Services connection type enables the [AWS Integrations](#).

## Authenticating to AWS

Authentication may be performed using any of the [boto3 options](#). Alternatively, one can pass credentials in as a Connection initialisation parameter.

To use IAM instance profile, create an “empty” connection (i.e. one with no Login or Password specified).

## Default Connection IDs

The default connection ID is `aws_default`.

## Configuring the Connection

**Login (optional)** Specify the AWS access key ID.

**Password (optional)** Specify the AWS secret access key.

**Extra (optional)** Specify the extra parameters (as json dictionary) that can be used in AWS connection. The following parameters are supported:

- **aws\_account\_id**: AWS account ID for the connection
- **aws\_iam\_role**: AWS IAM role for the connection
- **external\_id**: AWS external ID for the connection
- **host**: Endpoint URL for the connection
- **region\_name**: AWS region for the connection
- **role\_arn**: AWS role ARN for the connection

Example “extras” field:

```
{
  "aws_iam_role": "aws_iam_role_name",
  "region_name": "ap-southeast-2"
}
```

## MySQL

The MySQL connection type provides connection to a MySQL database.

## Configuring the Connection

**Host (required)** The host to connect to.

**Schema (optional)** Specify the schema name to be used in the database.

**Login (required)** Specify the user name to connect.

**Password (required)** Specify the password to connect.

**Extra (optional)** Specify the extra parameters (as json dictionary) that can be used in MySQL connection. The following parameters are supported:

- **charset**: specify charset of the connection
- **cursor**: one of “sscursor”, “dictcursor”, “ssdictcursor” . Specifies cursor class to be used
- **local\_infile**: controls MySQL’s LOCAL capability (permitting local data loading by clients). See [MySQLdb docs](#) for details.
- **unix\_socket**: UNIX socket used instead of the default socket.
- **ssl**: Dictionary of SSL parameters that control connecting using SSL. Those parameters are server specific and should contain “ca”, “cert”, “key”, “capath”, “cipher” parameters. See [MySQLdb docs](#) for details. Note that to be useful in URL notation, this parameter might also be a string where the SSL dictionary is a string-encoded JSON dictionary.

Example “extras” field:

```
{
  "charset": "utf8",
  "cursorclass": "sscursor",
  "local_infile": true,
  "unix_socket": "/var/socket",
  "ssl": {
    "cert": "/tmp/client-cert.pem",
    "ca": "/tmp/server-ca.pem",
    "key": "/tmp/client-key.pem"
  }
}
```

or

```
{
  "charset": "utf8",
  "cursorclass": "sscursor",
  "local_infile": true,
  "unix_socket": "/var/socket",
  "ssl": "{\"cert\": \"/tmp/client-cert.pem\", \"ca\": \"/tmp/server-ca.pem\", \"key\": \"/tmp/client-key.pem\"}"
}
```

When specifying the connection as URI (in `AIRFLOW_CONN_*` variable) you should specify it following the standard syntax of DB connections - where extras are passed as parameters of the URI. Note that all components of the URI should be URL-encoded.

For example:

```
mysql://mysql_user:XXXXXXXXXX@1.1.1.1:3306/mysql?ssl=%7B%22cert%22%3A+%22%2Ftmp%2Fclient-cert.pem%22%2C+%22ca%22%3A+%22%2Ftmp%2Fserver-ca.pem%22%2C+%22key%22%3A+%22%2Ftmp%2Fclient-key.pem%22%7D
```

---

**Note:** If encounter `UnicodeDecodeError` while working with MySQL connection, check the charset defined is matched to the database charset.

---

## Postgres

The Postgres connection type provides connection to a Postgres database.

### Configuring the Connection

**Host (required)** The host to connect to.

**Schema (optional)** Specify the schema name to be used in the database.

**Login (required)** Specify the user name to connect.

**Password (required)** Specify the password to connect.

**Extra (optional)** Specify the extra parameters (as json dictionary) that can be used in postgres connection. The following parameters out of the standard python parameters are supported:

- **sslmode** - This option determines whether or with what priority a secure SSL TCP/IP connection will be negotiated with the server. There are six modes: 'disable', 'allow', 'prefer', 'require', 'verify-ca', 'verify-full'.
- **sslcert** - This parameter specifies the file name of the client SSL certificate, replacing the default.
- **sslkey** - This parameter specifies the file name of the client SSL key, replacing the default.
- **sslrootcert** - This parameter specifies the name of a file containing SSL certificate authority (CA) certificate(s).
- **sslcrl** - This parameter specifies the file name of the SSL certificate revocation list (CRL).
- **application\_name** - Specifies a value for the application\_name configuration parameter.
- **keepalives\_idle** - Controls the number of seconds of inactivity after which TCP should send a keepalive message to the server.

More details on all Postgres parameters supported can be found in [Postgres documentation](#).

Example “extras” field:

```
{
  "sslmode": "verify-ca",
  "sslcert": "/tmp/client-cert.pem",
  "sslca": "/tmp/server-ca.pem",
  "sslkey": "/tmp/client-key.pem"
}
```

When specifying the connection as URI (in AIRFLOW\_CONN\_\* variable) you should specify it following the standard syntax of DB connections, where extras are passed as parameters of the URI (note that all components of the URI should be URL-encoded).

For example:

```
postgres://postgres_user:XXXXXXXXXX@1.1.1.1:5432/postgresdb?sslmode=verify-ca&
↪sslcert=%2Ftmp%2Fclient-cert.pem&sslkey=%2Ftmp%2Fclient-key.pem&sslrootcert=
↪%2Ftmp%2Fserver-ca.pem
```

## Cloudsql

The `gcpcloudsql://` connection is used by `airflow.contrib.operators.gcp_sql_operator.CloudSqlQueryOperator` to perform query on a Google Cloud SQL database. Google Cloud SQL database can be either Postgres or MySQL, so this is a “meta” connection type. It introduces common schema for both MySQL and Postgres, including what kind of connectivity should be used. Google Cloud SQL supports connecting via public IP or via Cloud SQL Proxy. In the latter case the `CloudSqlDatabaseHook` uses `CloudSqlProxyRunner` to automatically prepare and use temporary Postgres or MySQL connection that will use the proxy to connect (either via TCP or UNIX socket).

## Configuring the Connection

**Host (required)** The host to connect to.

**Schema (optional)** Specify the schema name to be used in the database.

**Login (required)** Specify the user name to connect.

**Password (required)** Specify the password to connect.

**Extra (optional)** Specify the extra parameters (as JSON dictionary) that can be used in Google Cloud SQL connection.

Details of all the parameters supported in extra field can be found in [CloudSqlDatabaseHook](#)

Example “extras” field:

```
{
  "database_type": "mysql",
  "project_id": "example-project",
  "location": "europe-west1",
  "instance": "testinstance",
  "use_proxy": true,
  "sql_proxy_use_tcp": false
}
```

When specifying the connection as URI (in AIRFLOW\_CONN\_\* variable), you should specify it following the standard syntax of DB connection, where extras are passed as parameters of the URI. Note that all components of the URI should be URL-encoded.

For example:

```
gcpcloudsql://user:XXXXXXXX@1.1.1.1:3306/mydb?database_type=mysql&project_
↪id=example-project&location=europe-west1&instance=testinstance&use_proxy=True&
↪sql_proxy_use_tcp=False
```

## SSH

The SSH connection type provides connection to use [SSHHook](#) to run commands on a remote server using [SSHOperator](#) or transfer file from/to the remote server using [SFTPOperator](#).

### Configuring the Connection

**Host (required)** The Remote host to connect.

**Username (optional)** The Username to connect to the remote\_host.

**Password (optional)** Specify the password of the username to connect to the remote\_host.

**Port (optional)** Port of remote host to connect. Default is 22.

**Extra (optional)** Specify the extra parameters (as json dictionary) that can be used in ssh connection. The following parameters out of the standard python parameters are supported:

- **timeout** - An optional timeout (in seconds) for the TCP connect. Default is 10.
- **compress** - `true` to ask the remote client/server to compress traffic; `false` to refuse compression. Default is `true`.
- **no\_host\_key\_check** - Set to `false` to restrict connecting to hosts with no entries in `~/.ssh/known_hosts` (Hosts file). This provides maximum protection against trojan horse attacks, but can be troublesome when the `/etc/ssh/ssh_known_hosts` file is poorly maintained or connections to new hosts are frequently made. This option forces the user to manually add all new hosts. Default is `true`, ssh will automatically add new host keys to the user known hosts files.
- **allow\_host\_key\_change** - Set to `true` if you want to allow connecting to hosts that has host key changed or when you get ‘REMOTE HOST IDENTIFICATION HAS CHANGED’ error. This wont protect

against Man-In-The-Middle attacks. Other possible solution is to remove the host entry from `~/.ssh/known_hosts` file. Default is `false`.

Example “extras” field:

```
{
  "timeout": "10",
  "compress": "false",
  "no_host_key_check": "false",
  "allow_host_key_change": "false"
}
```

When specifying the connection as URI (in `AIRFLOW_CONN_*` variable) you should specify it following the standard syntax of connections, where extras are passed as parameters of the URI (note that all components of the URI should be URL-encoded).

For example:

```
ssh://user:pass@localhost:22?timeout=10&compress=false&no_host_key_check=false&
↪allow_host_key_change=true
```

### 3.6.6 Securing Connections

By default, Airflow will save the passwords for the connection in plain text within the metadata database. The `crypto` package is highly recommended during installation. The `crypto` package does require that your operating system has `libffi-dev` installed.

If `crypto` package was not installed initially, it means that your Fernet key in `airflow.cfg` is empty.

You can still enable encryption for passwords within connections by following below steps:

1. Install `crypto` package `pip install apache-airflow[crypto]`
2. Generate `fernet_key`, using this code snippet below. `fernet_key` must be a base64-encoded 32-byte key.

```
from cryptography.fernet import Fernet
fernet_key= Fernet.generate_key()
print(fernet_key.decode()) # your fernet_key, keep it in secured place!
```

3. Replace `airflow.cfg` `fernet_key` value with the one from step 2. Alternatively, you can store your `fernet_key` in OS environment variable. You do not need to change `airflow.cfg` in this case as Airflow will use environment variable over the value in `airflow.cfg`:

```
# Note the double underscores
export AIRFLOW__CORE__FERNET_KEY=your_fernet_key
```

4. Restart Airflow webserver.
5. For existing connections (the ones that you had defined before installing `airflow[crypto]` and creating a Fernet key), you need to open each connection in the connection admin UI, re-type the password, and save it.

### 3.6.7 Rotating encryption keys

Once connection credentials and variables have been encrypted using a fernet key, changing the key will cause decryption of existing credentials to fail. To rotate the fernet key without invalidating existing encrypted values, prepend the new key to the `fernet_key` setting, run `airflow rotate_fernet_key`, and then drop the original key from `fernet_keys`:

1. Set `fernet_key` to `new_fernet_key`, `old_fernet_key`.
2. Run `airflow rotate_fernet_key` to reencrypt existing credentials with the new fernet key.
3. Set `fernet_key` to `new_fernet_key`.

## 3.6.8 Writing Logs

### 3.6.8.1 Writing Logs Locally

Users can specify a logs folder in `airflow.cfg` using the `base_log_folder` setting. By default, it is in the `AIRFLOW_HOME` directory.

In addition, users can supply a remote location for storing logs and log backups in cloud storage.

In the Airflow Web UI, local logs take precedence over remote logs. If local logs can not be found or accessed, the remote logs will be displayed. Note that logs are only sent to remote storage once a task completes (including failure). In other words, remote logs for running tasks are unavailable. Logs are stored in the log folder as `{dag_id}/{task_id}/{execution_date}/{try_number}.log`.

### 3.6.8.2 Writing Logs to Amazon S3

#### Before you begin

Remote logging uses an existing Airflow connection to read/write logs. If you don't have a connection properly setup, this will fail.

#### Enabling remote logging

To enable this feature, `airflow.cfg` must be configured as in this example:

```
[core]
# Airflow can store logs remotely in AWS S3. Users must supply a remote
# location URL (starting with either 's3://...') and an Airflow connection
# id that provides access to the storage location.
remote_logging = True
remote_base_log_folder = s3://my-bucket/path/to/logs
remote_log_conn_id = MyS3Conn
# Use server-side encryption for logs stored in S3
encrypt_s3_logs = False
```

In the above example, Airflow will try to use `S3Hook('MyS3Conn')`.

### 3.6.8.3 Writing Logs to Azure Blob Storage

Airflow can be configured to read and write task logs in Azure Blob Storage. Follow the steps below to enable Azure Blob Storage logging.

1. Airflow's logging system requires a custom `.py` file to be located in the `PYTHONPATH`, so that it's importable from Airflow. Start by creating a directory to store the config file. `$AIRFLOW_HOME/config` is recommended.
2. Create empty files called `$AIRFLOW_HOME/config/log_config.py` and `$AIRFLOW_HOME/config/__init__.py`.



- Copy the contents of `airflow/config_templates/airflow_local_settings.py` into the `log_config.py` file that was just created in the step above.
- Customize the following portions of the template:

```
# wasb buckets should start with "wasb" just to help Airflow select_
↪correct handler
REMOTE_BASE_LOG_FOLDER = 'wasb-<whatever you want here>'

# Rename DEFAULT_LOGGING_CONFIG to LOGGING_CONFIG
LOGGING_CONFIG = ...
```

- Make sure a Azure Blob Storage (Wasb) connection hook has been defined in Airflow. The hook should have read and write access to the Azure Blob Storage bucket defined above in `REMOTE_BASE_LOG_FOLDER`.
- Update `$AIRFLOW_HOME/airflow.cfg` to contain:

```
remote_logging = True
logging_config_class = log_config.LOGGING_CONFIG
remote_log_conn_id = <name of the Azure Blob Storage connection>
```

- Restart the Airflow webserver and scheduler, and trigger (or wait for) a new task execution.
- Verify that logs are showing up for newly executed tasks in the bucket you've defined.

### 3.6.8.4 Writing Logs to Google Cloud Storage

Follow the steps below to enable Google Cloud Storage logging.

To enable this feature, `airflow.cfg` must be configured as in this example:

```
[core]
# Airflow can store logs remotely in AWS S3, Google Cloud Storage or Elastic Search.
# Users must supply an Airflow connection id that provides access to the storage
# location. If remote_logging is set to true, see UPDATING.md for additional
# configuration requirements.
remote_logging = True
remote_base_log_folder = gs://my-bucket/path/to/logs
remote_log_conn_id = MyGCSConn
```

- Install the `gcp_api` package first, like so: `pip install apache-airflow[gcp_api]`.
- Make sure a Google Cloud Platform connection hook has been defined in Airflow. The hook should have read and write access to the Google Cloud Storage bucket defined above in `remote_base_log_folder`.
- Restart the Airflow webserver and scheduler, and trigger (or wait for) a new task execution.
- Verify that logs are showing up for newly executed tasks in the bucket you've defined.
- Verify that the Google Cloud Storage viewer is working in the UI. Pull up a newly executed task, and verify that you see something like:

```
*** Reading remote log from gs://<bucket where logs should be persisted>/
↪example_bash_operator/run_this_last/2017-10-03T00:00:00/16.log.
[2017-10-03 21:57:50,056] {cli.py:377} INFO - Running on host chrisr-00532
[2017-10-03 21:57:50,093] {base_task_runner.py:115} INFO - Running: ['bash
↪', '-c', u'airflow run example_bash_operator run_this_last 2017-10-
↪03T00:00:00 --job_id 47 --raw -sd DAGS_FOLDER/example_dags/example_bash_
↪operator.py']
```

(continues on next page)

(continued from previous page)

```
[2017-10-03 21:57:51,264] {base_task_runner.py:98} INFO - Subtask: [2017-
↪10-03 21:57:51,263] {__init__.py:45} INFO - Using executor_
↪SequentialExecutor
[2017-10-03 21:57:51,306] {base_task_runner.py:98} INFO - Subtask: [2017-
↪10-03 21:57:51,306] {models.py:186} INFO - Filling up the DagBag from /
↪airflow/dags/example_dags/example_bash_operator.py
```

Note the top line that says it's reading from the remote log file.

### 3.6.9 Scaling Out with Celery

`CeleryExecutor` is one of the ways you can scale out the number of workers. For this to work, you need to setup a Celery backend (**RabbitMQ**, **Redis**, ...) and change your `airflow.cfg` to point the executor parameter to `CeleryExecutor` and provide the related Celery settings.

For more information about setting up a Celery broker, refer to the exhaustive [Celery documentation on the topic](#).

Here are a few imperative requirements for your workers:

- `airflow` needs to be installed, and the CLI needs to be in the path
- Airflow configuration settings should be homogeneous across the cluster
- Operators that are executed on the worker need to have their dependencies met in that context. For example, if you use the `HiveOperator`, the hive CLI needs to be installed on that box, or if you use the `MySQLOperator`, the required Python library needs to be available in the `PYTHONPATH` somehow
- The worker needs to have access to its `DAGS_FOLDER`, and you need to synchronize the filesystems by your own means. A common setup would be to store your `DAGS_FOLDER` in a Git repository and sync it across machines using Chef, Puppet, Ansible, or whatever you use to configure machines in your environment. If all your boxes have a common mount point, having your pipelines files shared there should work as well

To kick off a worker, you need to setup Airflow and kick off the worker subcommand

```
airflow worker
```

Your worker should start picking up tasks as soon as they get fired in its direction.

Note that you can also run “Celery Flower”, a web UI built on top of Celery, to monitor your workers. You can use the shortcut command `airflow flower` to start a Flower web server.

Please note that you must have the `flower` python library already installed on your system. The recommend way is to install the `airflow celery` bundle.

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

Some caveats:

- Make sure to use a database backed result backend
- Make sure to set a visibility timeout in `[celery_broker_transport_options]` that exceeds the ETA of your longest running task
- Tasks can consume resources. Make sure your worker has enough resources to run `worker_concurrency` tasks

### 3.6.10 Scaling Out with Dask

`airflow.executors.dask_executor.DaskExecutor` allows you to run Airflow tasks in a Dask Distributed cluster.

Dask clusters can be run on a single machine or on remote networks. For complete details, consult the [Distributed documentation](#).

To create a cluster, first start a Scheduler:

```
# default settings for a local cluster
DASK_HOST=127.0.0.1
DASK_PORT=8786

dask-scheduler --host $DASK_HOST --port $DASK_PORT
```

Next start at least one Worker on any machine that can connect to the host:

```
dask-worker $DASK_HOST:$DASK_PORT
```

Edit your `airflow.cfg` to set your executor to `airflow.executors.dask_executor.DaskExecutor` and provide the Dask Scheduler address in the `[dask]` section.

Please note:

- Each Dask worker must be able to import Airflow and any dependencies you require.
- Dask does not support queues. If an Airflow task was created with a queue, a warning will be raised but the task will be submitted to the cluster.

### 3.6.11 Scaling Out with Mesos (community contributed)

There are two ways you can run airflow as a mesos framework:

1. Running airflow tasks directly on mesos slaves, requiring each mesos slave to have airflow installed and configured.
2. Running airflow tasks inside a docker container that has airflow installed, which is run on a mesos slave.

#### 3.6.11.1 Tasks executed directly on mesos slaves

`airflow.contrib.executors.mesos_executor.MesosExecutor` allows you to schedule airflow tasks on a Mesos cluster. For this to work, you need a running mesos cluster and you must perform the following steps -

1. Install airflow on a mesos slave where web server and scheduler will run, let's refer to this as the "Airflow server".
2. On the Airflow server, install mesos python eggs from [mesos downloads](#).
3. On the Airflow server, use a database (such as mysql) which can be accessed from all mesos slaves and add configuration in `airflow.cfg`.
4. Change your `airflow.cfg` to point executor parameter to `MesosExecutor` and provide related Mesos settings.
5. On all mesos slaves, install airflow. Copy the `airflow.cfg` from Airflow server (so that it uses same sql alchemy connection).
6. On all mesos slaves, run the following for serving logs:

```
airflow serve_logs
```

7. On Airflow server, to start processing/scheduling DAGs on mesos, run:

```
airflow scheduler -p
```

Note: We need `-p` parameter to pickle the DAGs.

You can now see the airflow framework and corresponding tasks in mesos UI. The logs for airflow tasks can be seen in airflow UI as usual.

For more information about mesos, refer to [mesos documentation](#). For any queries/bugs on `airflow.contrib.executors.mesos_executor.MesosExecutor`, please contact [@kapil-malik](#).

### 3.6.11.2 Tasks executed in containers on mesos slaves

This [gist](#) contains all files and configuration changes necessary to achieve the following:

1. Create a dockerized version of airflow with mesos python eggs installed.

We recommend taking advantage of docker's multi stage builds in order to achieve this. We have one Dockerfile that defines building a specific version of mesos from source (Dockerfile-mesos), in order to create the python eggs. In the airflow Dockerfile (Dockerfile-airflow) we copy the python eggs from the mesos image.

2. Create a mesos configuration block within the `airflow.cfg`.

The configuration block remains the same as the default airflow configuration (default\_airflow.cfg), but has the addition of an option `docker_image_slave`. This should be set to the name of the image you would like mesos to use when running airflow tasks. Make sure you have the proper configuration of the DNS record for your mesos master and any sort of authorization if any exists.

3. Change your `airflow.cfg` to point the executor parameter to *MesosExecutor* (`executor = SequentialExecutor`).
4. Make sure your mesos slave has access to the docker repository you are using for your `docker_image_slave`.

Instructions are available in the [mesos docs](#).

The rest is up to you and how you want to work with a dockerized airflow configuration.

### 3.6.12 Running Airflow behind a reverse proxy

Airflow can be set up behind a reverse proxy, with the ability to set its endpoint with great flexibility.

For example, you can configure your reverse proxy to get:

```
https://lab.mycompany.com/myorg/airflow/
```

To do so, you need to set the following setting in your `airflow.cfg`:

```
base_url = http://my_host/myorg/airflow
```

Additionally if you use Celery Executor, you can get Flower in `/myorg/flower` with:

```
flower_url_prefix = /myorg/flower
```

Your reverse proxy (ex: nginx) should be configured as follow:

- pass the url and http header as it for the Airflow webserver, without any rewrite, for example:

```
server {
    listen 80;
    server_name lab.mycompany.com;

    location /myorg/airflow/ {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

- rewrite the url for the flower endpoint:

```
server {
    listen 80;
    server_name lab.mycompany.com;

    location /myorg/flower/ {
        rewrite ^/myorg/flower/(.*)$ /$1 break; # remove prefix from http header
        proxy_pass http://localhost:5555;
        proxy_set_header Host $host;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

To ensure that Airflow generates URLs with the correct scheme when running behind a TLS-terminating proxy, you should configure the proxy to set the *X-Forwarded-Proto* header, and enable the *ProxyFix* middleware in your *airflow.cfg*:

```
enable_proxy_fix = True
```

**Note:** You should only enable the *ProxyFix* middleware when running Airflow behind a trusted proxy (AWS ELB, nginx, etc.).

### 3.6.13 Running Airflow with systemd

Airflow can integrate with systemd based systems. This makes watching your daemons easy as systemd can take care of restarting a daemon on failure. In the `scripts/systemd` directory you can find unit files that have been tested on Redhat based systems. You can copy those to `/usr/lib/systemd/system`. It is assumed that Airflow will run under `airflow:airflow`. If not (or if you are running on a non Redhat based system) you probably need to adjust the unit files.

Environment configuration is picked up from `/etc/sysconfig/airflow`. An example file is supplied. You can also define here, for example, `AIRFLOW_HOME` or `AIRFLOW_CONFIG`.

### 3.6.14 Running Airflow with upstart

Airflow can integrate with upstart based systems. Upstart automatically starts all airflow services for which you have a corresponding `*.conf` file in `/etc/init` upon system boot. On failure, upstart automatically restarts the process (until it reaches re-spawn limit set in a `*.conf` file).

You can find sample upstart job files in the `scripts/upstart` directory. These files have been tested on Ubuntu 14.04 LTS. You may have to adjust `start` on and `stop` on stanzas to make it work on other upstart systems. Some of the possible options are listed in `scripts/upstart/README`.

Modify `*.conf` files as needed and copy to `/etc/init` directory. It is assumed that airflow will run under `airflow:airflow`. Change `setuid` and `setgid` in `*.conf` files if you use other user/group

You can use `initctl` to manually start, stop, view status of the airflow process that has been integrated with upstart

```
initctl airflow-webserver status
```

### 3.6.15 Using the Test Mode Configuration

Airflow has a fixed set of “test mode” configuration options. You can load these at any time by calling `airflow.configuration.load_test_config()` (note this operation is not reversible!). However, some options (like the `DAG_FOLDER`) are loaded before you have a chance to call `load_test_config()`. In order to eagerly load the test configuration, set `test_mode` in `airflow.cfg`:

```
[tests]
unit_test_mode = True
```

Due to Airflow’s automatic environment variable expansion (see [Setting Configuration Options](#)), you can also set the env var `AIRFLOW__CORE__UNIT_TEST_MODE` to temporarily overwrite `airflow.cfg`.

### 3.6.16 Checking Airflow Health Status

To check the health status of your Airflow instance, you can simply access the endpoint `/health`. It will return a JSON object in which a high-level glance is provided.

```
{
  "metadatabase": {
    "status": "healthy"
  },
  "scheduler": {
    "status": "healthy",
    "latest_scheduler_heartbeat": "2018-12-26 17:15:11+00:00"
  }
}
```

- The status of each component can be either “healthy” or “unhealthy”.
  - The status of `metadatabase` is depending on whether a valid connection can be initiated with the database backend of Airflow.
  - The status of `scheduler` is depending on when the latest scheduler heartbeat happened. If the latest scheduler heartbeat happened 30 seconds (default value) earlier than the current time, scheduler component is considered unhealthy. You can also specify this threshold value by changing `scheduler_health_check_threshold` in scheduler section of the `airflow.cfg` file.

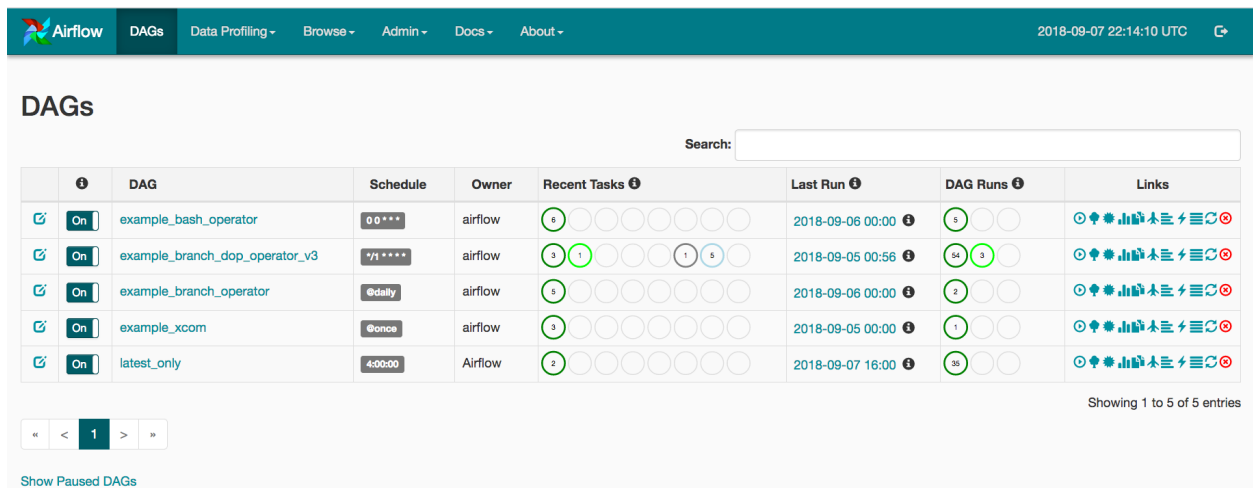
- The response code of `/health` endpoint is not used to label the health status of the application (it would always be 200). Hence please be reminded not to use the response code here for health-check purpose.

## 3.7 UI / Screenshots

The Airflow UI makes it easy to monitor and troubleshoot your data pipelines. Here's a quick overview of some of the features and visualizations you can find in the Airflow UI.

### 3.7.1 DAGs View

List of the DAGs in your environment, and a set of shortcuts to useful pages. You can see exactly how many tasks succeeded, failed, or are currently running at a glance.



	ⓘ	DAG	Schedule	Owner	Recent Tasks ⓘ	Last Run ⓘ	DAG Runs ⓘ	Links
	On	example_bash_operator	0 0 * * *	airflow	5 (3 succeeded, 1 failed, 1 running)	2018-09-06 00:00 ⓘ	5 (3 succeeded, 1 failed, 1 running)	
	On	example_branch_dop_operator_v3	* / 1 * * * *	airflow	3 (1 succeeded, 1 failed, 1 running)	2018-09-05 00:56 ⓘ	94 (3 succeeded, 1 failed, 1 running)	
	On	example_branch_operator	@daily	airflow	5 (3 succeeded, 1 failed, 1 running)	2018-09-06 00:00 ⓘ	2 (3 succeeded, 1 failed, 1 running)	
	On	example_xcom	@once	airflow	5 (3 succeeded, 1 failed, 1 running)	2018-09-05 00:00 ⓘ	1 (3 succeeded, 1 failed, 1 running)	
	On	latest_only	4:00:00	Airflow	2 (3 succeeded, 1 failed, 1 running)	2018-09-07 16:00 ⓘ	35 (3 succeeded, 1 failed, 1 running)	

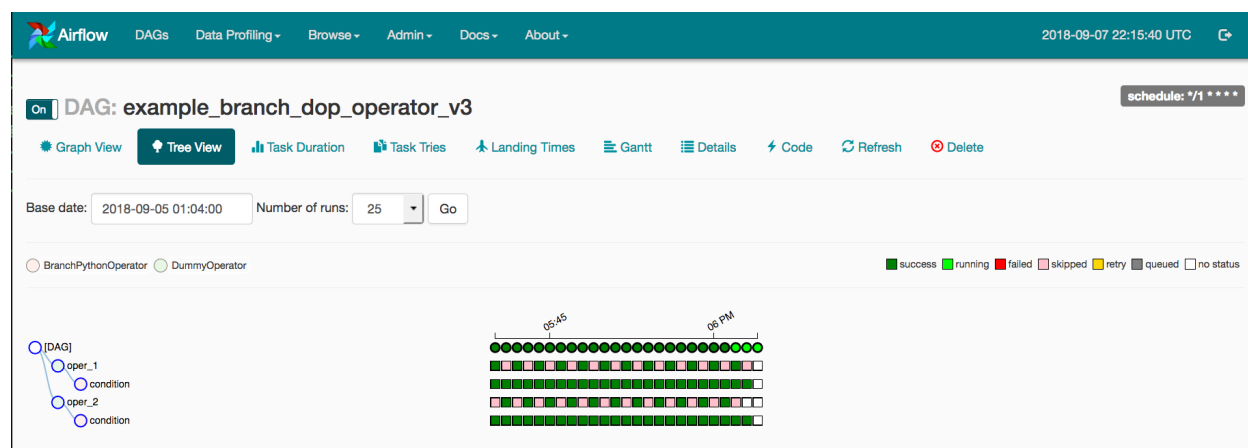
Showing 1 to 5 of 5 entries

« < 1 > »

Show Paused DAGs

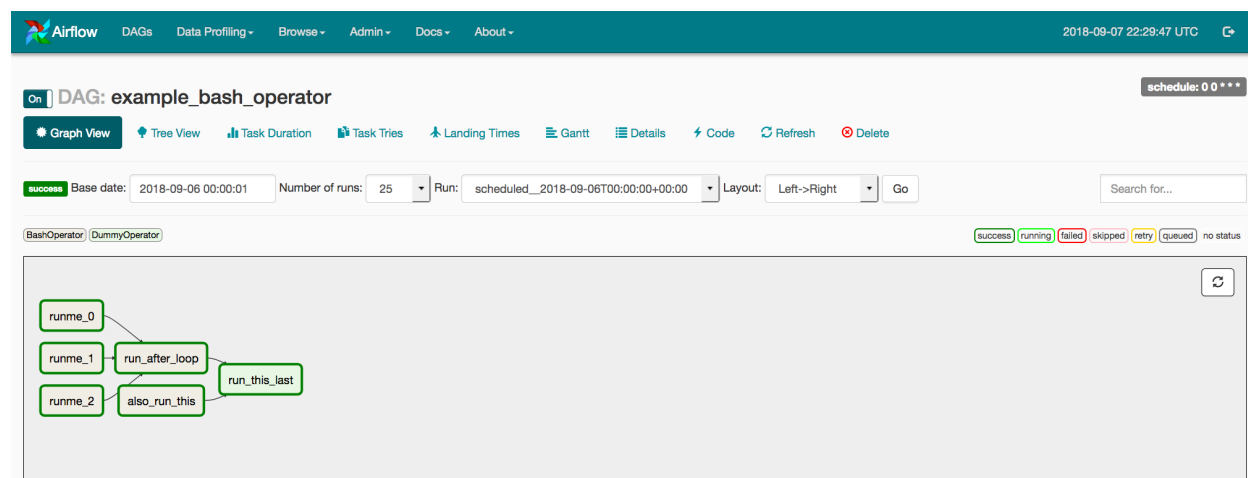
### 3.7.2 Tree View

A tree representation of the DAG that spans across time. If a pipeline is late, you can quickly see where the different steps are and identify the blocking ones.



### 3.7.3 Graph View

The graph view is perhaps the most comprehensive. Visualize your DAG's dependencies and their current status for a specific run.





















### 3.7.4 Variable View

The variable view allows you to list, create, edit or delete the key-value pair of a variable used during jobs. Value of a variable will be hidden if the key contains any words in ('password', 'secret', 'passwd', 'authorization', 'api\_key', 'apikey', 'access\_token') by default, but can be configured to show in clear-text.



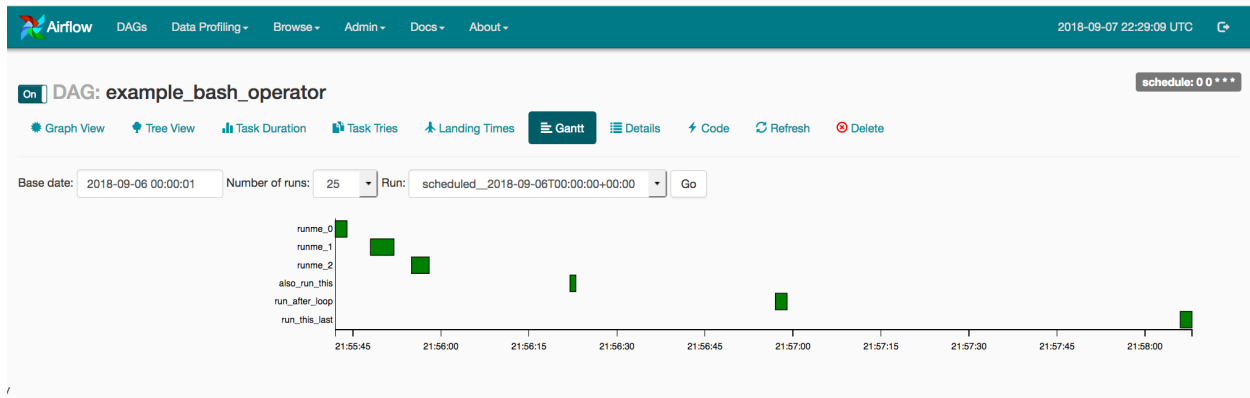


## Variables

List (9)		Create	Add Filter ▾	With selected ▾	Search	
<input type="checkbox"/>		Key			Val	
<input type="checkbox"/>	 	secret_password			*****	
<input type="checkbox"/>	 	not_so_hidden			test value	
<input type="checkbox"/>	 	secret			*****	
<input type="checkbox"/>	 	password			*****	
<input type="checkbox"/>	 	passwd			*****	
<input type="checkbox"/>	 	api_key			*****	
<input type="checkbox"/>	 	apikey			*****	
<input type="checkbox"/>	 	authorization			*****	
<input type="checkbox"/>	 	access_token			*****	

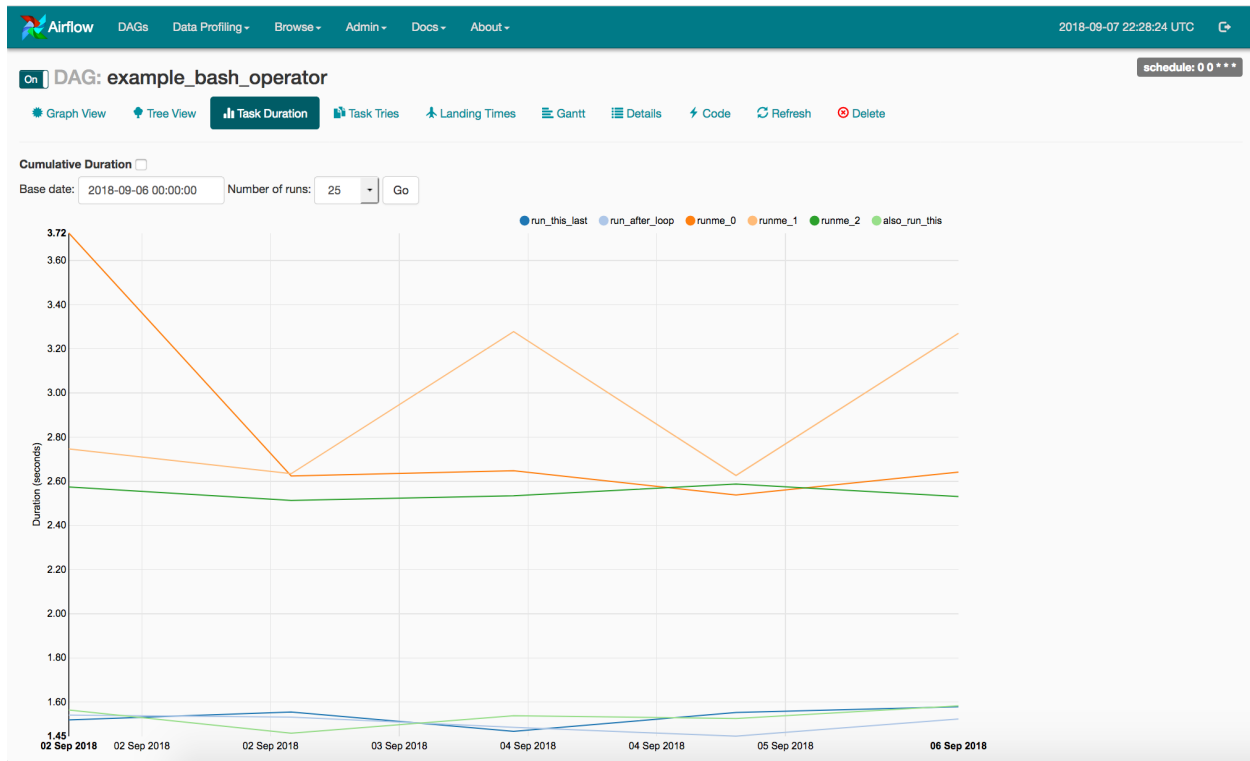
### 3.7.5 Gantt Chart

The Gantt chart lets you analyse task duration and overlap. You can quickly identify bottlenecks and where the bulk of the time is spent for specific DAG runs.



### 3.7.6 Task Duration

The duration of your different tasks over the past N runs. This view lets you find outliers and quickly understand where the time is spent in your DAG over many runs.



### 3.7.7 Code View

Transparency is everything. While the code for your pipeline is in source control, this is a quick way to get to the code that generates the DAG and provide yet more context.

```

1  # -*- coding: utf-8 -*-
2  #
3  # Licensed to the Apache Software Foundation (ASF) under one
4  # or more contributor license agreements. See the NOTICE file
5  # distributed with this work for additional information
6  # regarding copyright ownership. The ASF licenses this file
7  # to you under the Apache License, Version 2.0 (the
8  # "License"); you may not use this file except in compliance
9  # with the License. You may obtain a copy of the License at
10 #
11 # http://www.apache.org/licenses/LICENSE-2.0
12 #
13 # Unless required by applicable law or agreed to in writing,
14 # software distributed under the License is distributed on an
15 # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
16 # KIND, either express or implied. See the License for the
17 # specific language governing permissions and limitations
18 # under the License.
19
20 import airflow
21 from builtins import range
22 from airflow.operators.bash_operator import BashOperator
23 from airflow.operators.dummy_operator import DummyOperator
24 from airflow.models import DAG
25 from datetime import timedelta
26
27
28 args = {
29     'owner': 'airflow',
30     'start_date': airflow.utils.dates.days_ago(2)
31 }
32
33 dag = DAG(
34     dag_id='example_bash_operator', default_args=args,
35     schedule_interval='0 * * *',
36     dagrun_timeout=timedelta(minutes=60))
37
38 cmd = 'ls -l'
39 run_this_last = DummyOperator(task_id='run_this_last', dag=dag)
40
41 # [START howto_operator_bash]
42 run_after_loop = BashOperator(
43     task_id='run_after_loop', bash_command='echo 1', dag=dag)
44 # [END howto_operator_bash]
45 run_this_last.set_downstream(run_after_loop)
46

```

### 3.7.8 Task Instance Context Menu

From the pages seen above (tree view, graph view, gantt, . . .), it is always possible to click on a task instance, and get to this rich context menu that can take you to more detailed metadata, and perform some actions.

## 3.8 Concepts

The Airflow Platform is a tool for describing, executing, and monitoring workflows.

## 3.8.1 Core Ideas

### 3.8.1.1 DAGs

In Airflow, a DAG – or a Directed Acyclic Graph – is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.

For example, a simple DAG could consist of three tasks: A, B, and C. It could say that A has to run successfully before B can run, but C can run anytime. It could say that task A times out after 5 minutes, and B can be restarted up to 5 times in case it fails. It might also say that the workflow will run every night at 10pm, but shouldn't start until a certain date.

In this way, a DAG describes *how* you want to carry out your workflow; but notice that we haven't said anything about *what* we actually want to do! A, B, and C could be anything. Maybe A prepares data for B to analyze while C sends an email. Or perhaps A monitors your location so B can open your garage door while C turns on your house lights. The important thing is that the DAG isn't concerned with what its constituent tasks do; its job is to make sure that whatever they do happens at the right time, or in the right order, or with the right handling of any unexpected issues.

DAGs are defined in standard Python files that are placed in Airflow's `DAG_FOLDER`. Airflow will execute the code in each file to dynamically build the DAG objects. You can have as many DAGs as you want, each describing an arbitrary number of tasks. In general, each one should correspond to a single logical workflow.

---

**Note:** When searching for DAGs, Airflow will only consider files where the string “airflow” and “DAG” both appear in the contents of the `.py` file.

---

## Scope

Airflow will load any DAG object it can import from a DAGfile. Critically, that means the DAG must appear in `globals()`. Consider the following two DAGs. Only `dag_1` will be loaded; the other one only appears in a local scope.

```
dag_1 = DAG('this_dag_will_be_discovered')

def my_function():
    dag_2 = DAG('but_this_dag_will_not')

my_function()
```

Sometimes this can be put to good use. For example, a common pattern with `SubDagOperator` is to define the subdag inside a function so that Airflow doesn't try to load it as a standalone DAG.

## Default Arguments

If a dictionary of `default_args` is passed to a DAG, it will apply them to any of its operators. This makes it easy to apply a common parameter to many operators without having to type it many times.

```
default_args = {
    'start_date': datetime(2016, 1, 1),
    'owner': 'Airflow'
}

dag = DAG('my_dag', default_args=default_args)
```

(continues on next page)

(continued from previous page)

```
op = DummyOperator(task_id='dummy', dag=dag)
print(op.owner) # Airflow
```

## Context Manager

*Added in Airflow 1.8*

DAGs can be used as context managers to automatically assign new operators to that DAG.

```
with DAG('my_dag', start_date=datetime(2016, 1, 1)) as dag:
    op = DummyOperator('op')

op.dag is dag # True
```

### 3.8.1.2 Operators

While DAGs describe *how* to run a workflow, Operators determine what actually gets done.

An operator describes a single task in a workflow. Operators are usually (but not always) atomic, meaning they can stand on their own and don't need to share resources with any other operators. The DAG will make sure that operators run in the correct certain order; other than those dependencies, operators generally run independently. In fact, they may run on two completely different machines.

This is a subtle but very important point: in general, if two operators need to share information, like a filename or small amount of data, you should consider combining them into a single operator. If it absolutely can't be avoided, Airflow does have a feature for operator cross-communication called XCom that is described elsewhere in this document.

Airflow provides operators for many common tasks, including:

- `airflow.operators.bash_operator.BashOperator` - executes a bash command
- `airflow.operators.python_operator.PythonOperator` - calls an arbitrary Python function
- `airflow.operators.email_operator.EmailOperator` - sends an email
- `airflow.operators.http_operator.SimpleHttpOperator` - sends an HTTP request
- `airflow.operators.mysql_operator.MySqlOperator`, `airflow.operators.sqlite_operator.SqliteOperator`, `airflow.operators.postgres_operator.PostgresOperator`, `airflow.operators.mssql_operator.MsSqlOperator`, `airflow.operators.oracle_operator.OracleOperator`, `airflow.operators.jdbc_operator.JdbcOperator`, etc. - executes a SQL command
- Sensor - waits for a certain time, file, database row, S3 key, etc. ...

In addition to these basic building blocks, there are many more specific operators: `airflow.operators.docker_operator.DockerOperator`, `airflow.operators.hive_operator.HiveOperator`, `airflow.operators.s3_file_transform_operator.S3FileTransformOperator` (`airflow.operators.presto_to_mysql.PrestoToMySqlTransfer`, `airflow.operators.slack_operator.SlackAPIOperator`... you get the idea!

The `airflow/contrib/` directory contains yet more operators built by the community. These operators aren't always as complete or well-tested as those in the main distribution, but allow users to more easily add new functionality to the platform.

Operators are only loaded by Airflow if they are assigned to a DAG.

See *Using Operators* for how to use Airflow operators.

## DAG Assignment

*Added in Airflow 1.8*

Operators do not have to be assigned to DAGs immediately (previously `dag` was a required argument). However, once an operator is assigned to a DAG, it can not be transferred or unassigned. DAG assignment can be done explicitly when the operator is created, through deferred assignment, or even inferred from other operators.

```
dag = DAG('my_dag', start_date=datetime(2016, 1, 1))

# sets the DAG explicitly
explicit_op = DummyOperator(task_id='op1', dag=dag)

# deferred DAG assignment
deferred_op = DummyOperator(task_id='op2')
deferred_op.dag = dag

# inferred DAG assignment (linked operators must be in the same DAG)
inferred_op = DummyOperator(task_id='op3')
inferred_op.set_upstream(deferred_op)
```

## Bitshift Composition

*Added in Airflow 1.8*

Traditionally, operator relationships are set with the `set_upstream()` and `set_downstream()` methods. In Airflow 1.8, this can be done with the Python bitshift operators `>>` and `<<`. The following four statements are all functionally equivalent:

```
op1 >> op2
op1.set_downstream(op2)

op2 << op1
op2.set_upstream(op1)
```

When using the bitshift to compose operators, the relationship is set in the direction that the bitshift operator points. For example, `op1 >> op2` means that `op1` runs first and `op2` runs second. Multiple operators can be composed – keep in mind the chain is executed left-to-right and the rightmost object is always returned. For example:

```
op1 >> op2 >> op3 << op4
```

is equivalent to:

```
op1.set_downstream(op2)
op2.set_downstream(op3)
op3.set_upstream(op4)
```

For convenience, the bitshift operators can also be used with DAGs. For example:

```
dag >> op1 >> op2
```

is equivalent to:

```
op1.dag = dag
op1.set_downstream(op2)
```

We can put this all together to build a simple pipeline:

```
with DAG('my_dag', start_date=datetime(2016, 1, 1)) as dag:
    (
        DummyOperator(task_id='dummy_1')
        >> BashOperator(
            task_id='bash_1',
            bash_command='echo "HELLO!"')
        >> PythonOperator(
            task_id='python_1',
            python_callable=lambda: print("GOODBYE!"))
    )
```

Bitshift can also be used with lists. For example:

```
op1 >> [op2, op3]
```

is equivalent to:

```
op1 >> op2
op1 >> op3
```

and equivalent to:

```
op1.set_downstream([op2, op3])
```

### 3.8.1.3 Tasks

Once an operator is instantiated, it is referred to as a “task”. The instantiation defines specific values when calling the abstract operator, and the parameterized task becomes a node in a DAG.

### 3.8.1.4 Task Instances

A task instance represents a specific run of a task and is characterized as the combination of a dag, a task, and a point in time. Task instances also have an indicative state, which could be “running”, “success”, “failed”, “skipped”, “up for retry”, etc.

### 3.8.1.5 Workflows

You’re now familiar with the core building blocks of Airflow. Some of the concepts may sound very similar, but the vocabulary can be conceptualized like this:

- DAG: a description of the order in which work should take place
- Operator: a class that acts as a template for carrying out some work
- Task: a parameterized instance of an operator
- Task Instance: a task that 1) has been assigned to a DAG and 2) has a state associated with a specific run of the DAG

By combining DAGs and Operators to create TaskInstances, you can build complex workflows.

### 3.8.2 Additional Functionality

In addition to the core Airflow objects, there are a number of more complex features that enable behaviors like limiting simultaneous access to resources, cross-communication, conditional execution, and more.

#### 3.8.2.1 Hooks

Hooks are interfaces to external platforms and databases like Hive, S3, MySQL, Postgres, HDFS, and Pig. Hooks implement a common interface when possible, and act as a building block for operators. They also use the `airflow.models.connection.Connection` model to retrieve hostnames and authentication information. Hooks keep authentication code and information out of pipelines, centralized in the metadata database.

Hooks are also very useful on their own to use in Python scripts, Airflow `airflow.operators.PythonOperator`, and in interactive environments like iPython or Jupyter Notebook.

#### 3.8.2.2 Pools

Some systems can get overwhelmed when too many processes hit them at the same time. Airflow pools can be used to **limit the execution parallelism** on arbitrary sets of tasks. The list of pools is managed in the UI (Menu -> Admin -> Pools) by giving the pools a name and assigning it a number of worker slots. Tasks can then be associated with one of the existing pools by using the `pool` parameter when creating tasks (i.e., instantiating operators).

```
aggregate_db_message_job = BashOperator(
    task_id='aggregate_db_message_job',
    execution_timeout=timedelta(hours=3),
    pool='ep_data_pipeline_db_msg_agg',
    bash_command=aggregate_db_message_job_cmd,
    dag=dag)
aggregate_db_message_job.set_upstream(wait_for_empty_queue)
```

The `pool` parameter can be used in conjunction with `priority_weight` to define priorities in the queue, and which tasks get executed first as slots open up in the pool. The default `priority_weight` is 1, and can be bumped to any number. When sorting the queue to evaluate which task should be executed next, we use the `priority_weight`, summed up with all of the `priority_weight` values from tasks downstream from this task. You can use this to bump a specific important task and the whole path to that task gets prioritized accordingly.

Tasks will be scheduled as usual while the slots fill up. Once capacity is reached, runnable tasks get queued and their state will show as such in the UI. As slots free up, queued tasks start running based on the `priority_weight` (of the task and its descendants).

Note that by default tasks aren't assigned to any pool and their execution parallelism is only limited to the executor's setting.

#### 3.8.2.3 Connections

The connection information to external systems is stored in the Airflow metadata database and managed in the UI (Menu -> Admin -> Connections). A `conn_id` is defined there and hostname / login / password / schema information attached to it. Airflow pipelines can simply refer to the centrally managed `conn_id` without having to hard code any of this information anywhere.

Many connections with the same `conn_id` can be defined and when that is the case, and when the **hooks** uses the `get_connection` method from `BaseHook`, Airflow will choose one connection randomly, allowing for some basic load balancing and fault tolerance when used in conjunction with retries.



Airflow also has the ability to reference connections via environment variables from the operating system. But it only supports URI format. If you need to specify `extra` for your connection, please use web UI.

If connections with the same `conn_id` are defined in both Airflow metadata database and environment variables, only the one in environment variables will be referenced by Airflow (for example, given `conn_id postgres_master`, Airflow will search for `AIRFLOW_CONN_POSTGRES_MASTER` in environment variables first and directly reference it if found, before it starts to search in metadata database).

Many hooks have a default `conn_id`, where operators using that hook do not need to supply an explicit connection ID. For example, the default `conn_id` for the `PostgresHook` is `postgres_default`.

See [Managing Connections](#) for how to create and manage connections.

### 3.8.2.4 Queues

When using the CeleryExecutor, the Celery queues that tasks are sent to can be specified. `queue` is an attribute of `BaseOperator`, so any task can be assigned to any queue. The default queue for the environment is defined in the `airflow.cfg`'s `celery -> default_queue`. This defines the queue that tasks get assigned to when not specified, as well as which queue Airflow workers listen to when started.

Workers can listen to one or multiple queues of tasks. When a worker is started (using the command `airflow worker`), a set of comma-delimited queue names can be specified (e.g. `airflow worker -q spark`). This worker will then only pick up tasks wired to the specified queue(s).

This can be useful if you need specialized workers, either from a resource perspective (for say very lightweight tasks where one worker could take thousands of tasks without a problem), or from an environment perspective (you want a worker running from within the Spark cluster itself because it needs a very specific environment and security rights).

### 3.8.2.5 XComs

XComs let tasks exchange messages, allowing more nuanced forms of control and shared state. The name is an abbreviation of “cross-communication”. XComs are principally defined by a key, value, and timestamp, but also track attributes like the task/DAG that created the XCom and when it should become visible. Any object that can be pickled can be used as an XCom value, so users should make sure to use objects of appropriate size.

XComs can be “pushed” (sent) or “pulled” (received). When a task pushes an XCom, it makes it generally available to other tasks. Tasks can push XComs at any time by calling the `xcom_push()` method. In addition, if a task returns a value (either from its `Operator`'s `execute()` method, or from a `PythonOperator`'s `python_callable` function), then an XCom containing that value is automatically pushed.

Tasks call `xcom_pull()` to retrieve XComs, optionally applying filters based on criteria like `key`, `source task_ids`, and `source dag_id`. By default, `xcom_pull()` filters for the keys that are automatically given to XComs when they are pushed by being returned from execute functions (as opposed to XComs that are pushed manually).

If `xcom_pull` is passed a single string for `task_ids`, then the most recent XCom value from that task is returned; if a list of `task_ids` is passed, then a corresponding list of XCom values is returned.

```
# inside a PythonOperator called 'pushing_task'
def push_function():
    return value

# inside another PythonOperator where provide_context=True
def pull_function(**context):
    value = context['task_instance'].xcom_pull(task_ids='pushing_task')
```

It is also possible to pull XCom directly in a template, here's an example of what this may look like:

```
SELECT * FROM {{ task_instance.xcom_pull(task_ids='foo', key='table_name') }}
```

Note that XComs are similar to *Variables*, but are specifically designed for inter-task communication rather than global settings.

### 3.8.2.6 Variables

Variables are a generic way to store and retrieve arbitrary content or settings as a simple key value store within Airflow. Variables can be listed, created, updated and deleted from the UI (Admin -> Variables), code or CLI. In addition, json settings files can be bulk uploaded through the UI. While your pipeline code definition and most of your constants and variables should be defined in code and stored in source control, it can be useful to have some variables or configuration items accessible and modifiable through the UI.

```
from airflow.models import Variable
foo = Variable.get("foo")
bar = Variable.get("bar", deserialize_json=True)
```

The second call assumes json content and will be deserialized into bar. Note that Variable is a sqlalchemy model and can be used as such.

You can use a variable from a jinja template with the syntax :

```
echo {{ var.value.<variable_name> }}
```

or if you need to deserialize a json object from the variable :

```
echo {{ var.json.<variable_name> }}
```

### 3.8.2.7 Branching

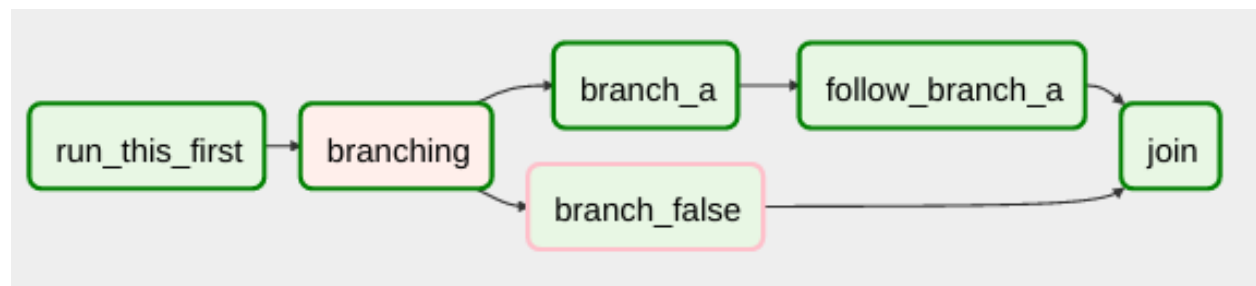
Sometimes you need a workflow to branch, or only go down a certain path based on an arbitrary condition which is typically related to something that happened in an upstream task. One way to do this is by using the BranchPythonOperator.

The BranchPythonOperator is much like the PythonOperator except that it expects a python\_callable that returns a task\_id (or list of task\_ids). The task\_id returned is followed, and all of the other paths are skipped. The task\_id returned by the Python function has to be referencing a task directly downstream from the BranchPythonOperator task.

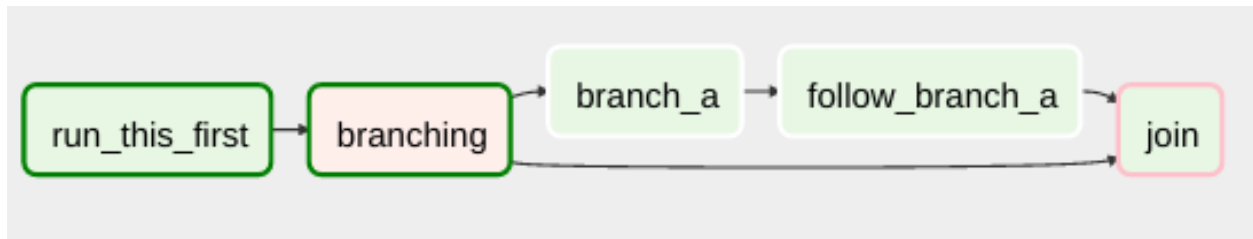
Note that using tasks with depends\_on\_past=True downstream from BranchPythonOperator is logically unsound as skipped status will invariably lead to block tasks that depend on their past successes. skipped states propagates where all directly upstream tasks are skipped.

If you want to skip some tasks, keep in mind that you can't have an empty path, if so make a dummy task.

like this, the dummy task "branch\_false" is skipped



Not like this, where the join task is skipped



The `BranchPythonOperator` can also be used with XComs allowing branching context to dynamically decide what branch to follow based on previous tasks. For example:

```
def branch_func(**kwargs):
    ti = kwargs['ti']
    xcom_value = int(ti.xcom_pull(task_ids='start_task'))
    if xcom_value >= 5:
        return 'continue_task'
    else:
        return 'stop_task'

start_op = BashOperator(
    task_id='start_task',
    bash_command="echo 5",
    xcom_push=True,
    dag=dag)

branch_op = BranchPythonOperator(
    task_id='branch_task',
    provide_context=True,
    python_callable=branch_func,
    dag=dag)

continue_op = DummyOperator(task_id='continue_task', dag=dag)
stop_op = DummyOperator(task_id='stop_task', dag=dag)

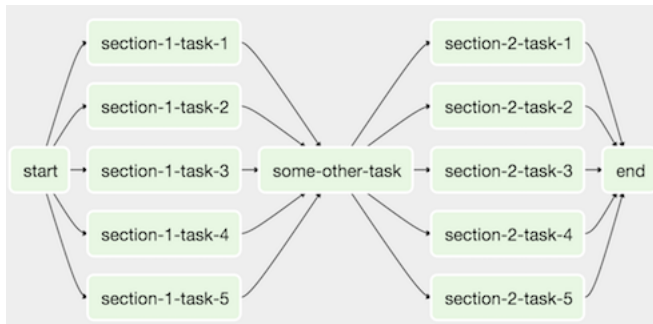
start_op >> branch_op >> [continue_op, stop_op]
```

### 3.8.2.8 SubDAGs

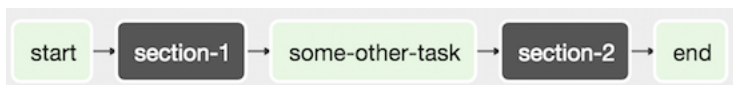
SubDAGs are perfect for repeating patterns. Defining a function that returns a DAG object is a nice design pattern when using Airflow.

Airbnb uses the *stage-check-exchange* pattern when loading data. Data is staged in a temporary table, after which data quality checks are performed against that table. Once the checks all pass the partition is moved into the production table.

As another example, consider the following DAG:



We can combine all of the parallel `task-*` operators into a single SubDAG, so that the resulting DAG resembles the following:



Note that SubDAG operators should contain a factory method that returns a DAG object. This will prevent the SubDAG from being treated like a separate DAG in the main UI. For example:

```
# dags/subdag.py
from airflow.models import DAG
from airflow.operators.dummy_operator import DummyOperator

# Dag is returned by a factory method
def sub_dag(parent_dag_name, child_dag_name, start_date, schedule_interval):
    dag = DAG(
        '%s.%s' % (parent_dag_name, child_dag_name),
        schedule_interval=schedule_interval,
        start_date=start_date,
    )

    dummy_operator = DummyOperator(
        task_id='dummy_task',
        dag=dag,
    )

    return dag
```

This SubDAG can then be referenced in your main DAG file:

```
# main_dag.py
from datetime import datetime, timedelta
from airflow.models import DAG
from airflow.operators.subdag_operator import SubDagOperator
from dags.subdag import sub_dag

PARENT_DAG_NAME = 'parent_dag'
CHILD_DAG_NAME = 'child_dag'

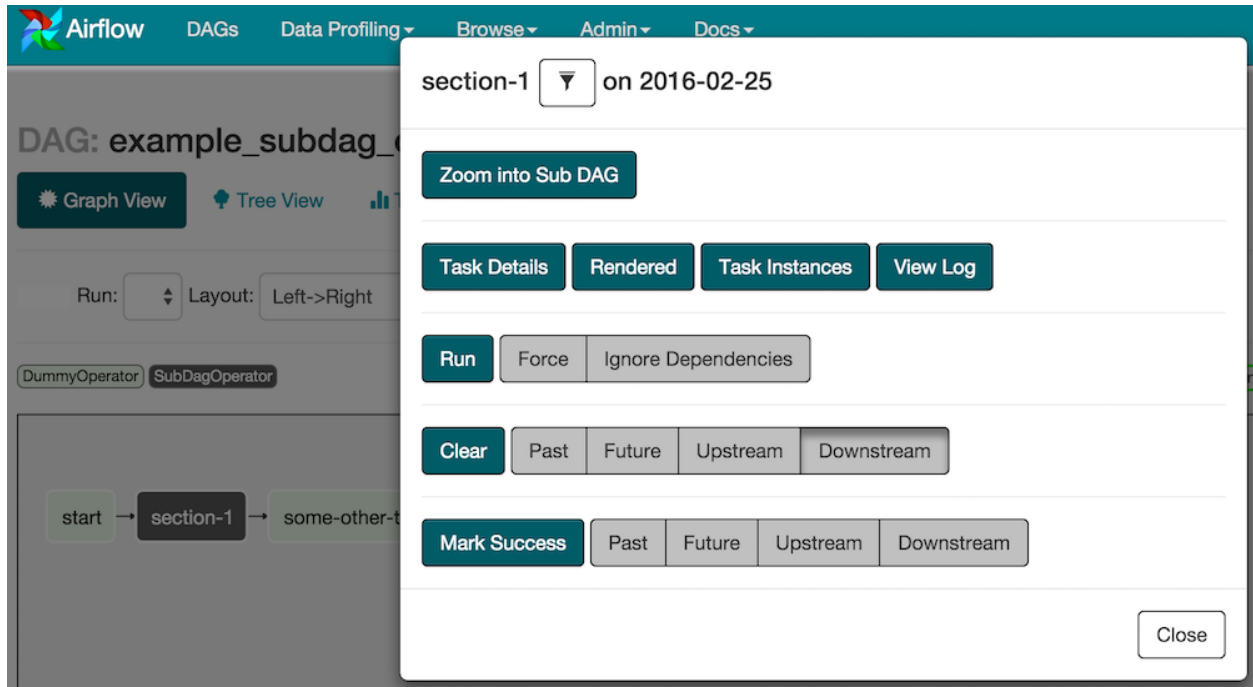
main_dag = DAG(
    dag_id=PARENT_DAG_NAME,
    schedule_interval=timedelta(hours=1),
    start_date=datetime(2016, 1, 1)
)
```

(continues on next page)

(continued from previous page)

```
sub_dag = SubDagOperator(
    subdag=sub_dag(PARENT_DAG_NAME, CHILD_DAG_NAME, main_dag.start_date,
                    main_dag.schedule_interval),
    task_id=CHILD_DAG_NAME,
    dag=main_dag,
)
```

You can zoom into a SubDagOperator from the graph view of the main DAG to show the tasks contained within the SubDAG:



Some other tips when using SubDAGs:

- by convention, a SubDAG's `dag_id` should be prefixed by its parent and a dot. As in `parent.child`
- share arguments between the main DAG and the SubDAG by passing arguments to the SubDAG operator (as demonstrated above)
- SubDAGs must have a schedule and be enabled. If the SubDAG's schedule is set to `None` or `@once`, the SubDAG will succeed without having done anything
- clearing a SubDagOperator also clears the state of the tasks within
- marking success on a SubDagOperator does not affect the state of the tasks within
- refrain from using `depends_on_past=True` in tasks within the SubDAG as this can be confusing
- it is possible to specify an executor for the SubDAG. It is common to use the `SequentialExecutor` if you want to run the SubDAG in-process and effectively limit its parallelism to one. Using `LocalExecutor` can be problematic as it may over-subscribe your worker, running multiple tasks in a single slot

See `airflow/example_dags` for a demonstration.

### 3.8.2.9 SLAs

Service Level Agreements, or time by which a task or DAG should have succeeded, can be set at a task level as a `timedelta`. If one or many instances have not succeeded by that time, an alert email is sent detailing the list of tasks that missed their SLA. The event is also recorded in the database and made available in the web UI under Browse->SLA Misses where events can be analyzed and documented.

### Email Configuration

You can configure the email that is being sent in your `airflow.cfg` by setting a `subject_template` and/or a `html_content_template` in the email section.

```
[email]

email_backend = airflow.utils.email.send_email_smtp

subject_template = /path/to/my_subject_template_file
html_content_template = /path/to/my_html_content_template_file
```

To access the task's information you use [Jinja Templating](#) in your template files.

For example a `html_content_template` file could look like this:

```
Try {{try_number}} out of {{max_tries + 1}}<br>
Exception:<br>{{exception_html}}<br>
Log: <a href="{{ti.log_url}}">Link</a><br>
Host: {{ti.hostname}}<br>
Log file: {{ti.log_filepath}}<br>
Mark success: <a href="{{ti.mark_success_url}}">Link</a><br>
```

### 3.8.2.10 Trigger Rules

Though the normal workflow behavior is to trigger tasks when all their directly upstream tasks have succeeded, Airflow allows for more complex dependency settings.

All operators have a `trigger_rule` argument which defines the rule by which the generated task get triggered. The default value for `trigger_rule` is `all_success` and can be defined as “trigger this task when all directly upstream tasks have succeeded”. All other rules described here are based on direct parent tasks and are values that can be passed to any operator while creating tasks:

- `all_success`: (default) all parents have succeeded
- `all_failed`: all parents are in a failed or `upstream_failed` state
- `all_done`: all parents are done with their execution
- `one_failed`: fires as soon as at least one parent has failed, it does not wait for all parents to be done
- `one_success`: fires as soon as at least one parent succeeds, it does not wait for all parents to be done
- `none_failed`: all parents have not failed (failed or `upstream_failed`) i.e. all parents have succeeded or been skipped
- `dummy`: dependencies are just for show, trigger at will

Note that these can be used in conjunction with `depends_on_past` (boolean) that, when set to `True`, keeps a task from getting triggered if the previous schedule for the task hasn't succeeded.

### 3.8.2.11 Latest Run Only

Standard workflow behavior involves running a series of tasks for a particular date/time range. Some workflows, however, perform tasks that are independent of run time but need to be run on a schedule, much like a standard cron job. In these cases, backfills or running jobs missed during a pause just wastes CPU cycles.

For situations like this, you can use the `LatestOnlyOperator` to skip tasks that are not being run during the most recent scheduled run for a DAG. The `LatestOnlyOperator` skips all immediate downstream tasks, and itself, if the time right now is not between its `execution_time` and the next scheduled `execution_time`.

One must be aware of the interaction between skipped tasks and trigger rules. Skipped tasks will cascade through trigger rules `all_success` and `all_failed` but not `all_done`, `one_failed`, `one_success`, and `dummy`. If you would like to use the `LatestOnlyOperator` with trigger rules that do not cascade skips, you will need to ensure that the `LatestOnlyOperator` is **directly** upstream of the task you would like to skip.

It is possible, through use of trigger rules to mix tasks that should run in the typical date/time dependent mode and those using the `LatestOnlyOperator`.

For example, consider the following dag:

```
#dags/latest_only_with_trigger.py
import datetime as dt

from airflow.models import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.latest_only_operator import LatestOnlyOperator
from airflow.utils.trigger_rule import TriggerRule

dag = DAG(
    dag_id='latest_only_with_trigger',
    schedule_interval=dt.timedelta(hours=4),
    start_date=dt.datetime(2016, 9, 20),
)

latest_only = LatestOnlyOperator(task_id='latest_only', dag=dag)

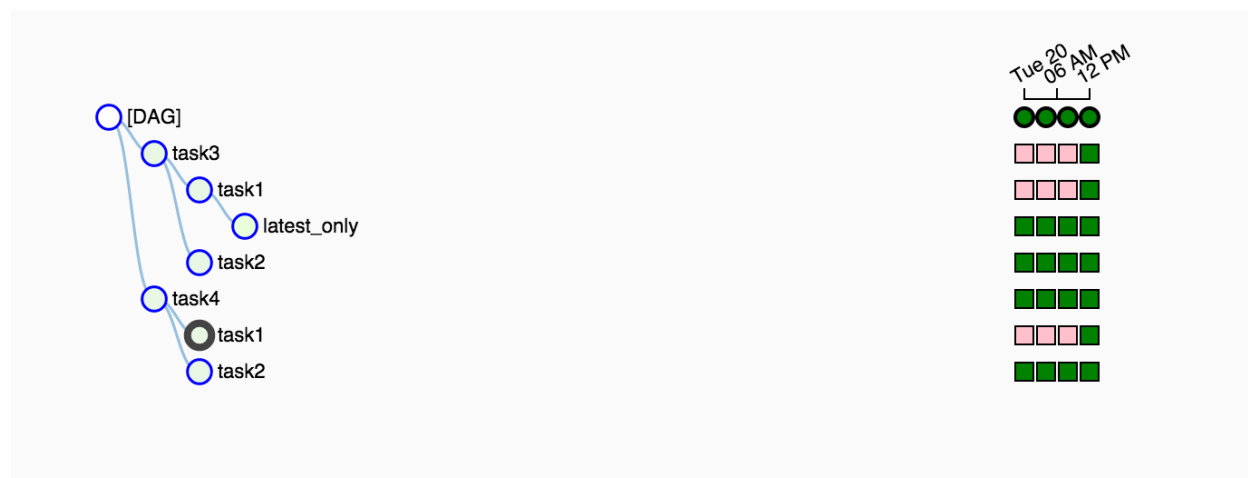
task1 = DummyOperator(task_id='task1', dag=dag)
task1.set_upstream(latest_only)

task2 = DummyOperator(task_id='task2', dag=dag)

task3 = DummyOperator(task_id='task3', dag=dag)
task3.set_upstream([task1, task2])

task4 = DummyOperator(task_id='task4', dag=dag,
                      trigger_rule=TriggerRule.ALL_DONE)
task4.set_upstream([task1, task2])
```

In the case of this dag, the `latest_only` task will show up as skipped for all runs except the latest run. `task1` is directly downstream of `latest_only` and will also skip for all runs except the latest. `task2` is entirely independent of `latest_only` and will run in all scheduled periods. `task3` is downstream of `task1` and `task2` and because of the default `trigger_rule` being `all_success` will receive a cascaded skip from `task1`. `task4` is downstream of `task1` and `task2` but since its `trigger_rule` is set to `all_done` it will trigger as soon as `task1` has been skipped (a valid completion state) and `task2` has succeeded.



### 3.8.2.12 Zombies & Undeads

Task instances die all the time, usually as part of their normal life cycle, but sometimes unexpectedly.

Zombie tasks are characterized by the absence of an heartbeat (emitted by the job periodically) and a `running` status in the database. They can occur when a worker node can't reach the database, when Airflow processes are killed externally, or when a node gets rebooted for instance. Zombie killing is performed periodically by the scheduler's process.

Undead processes are characterized by the existence of a process and a matching heartbeat, but Airflow isn't aware of this task as `running` in the database. This mismatch typically occurs as the state of the database is altered, most likely by deleting rows in the "Task Instances" view in the UI. Tasks are instructed to verify their state as part of the heartbeat routine, and terminate themselves upon figuring out that they are in this "undead" state.

### 3.8.2.13 Cluster Policy

Your local airflow settings file can define a `policy` function that has the ability to mutate task attributes based on other task or DAG attributes. It receives a single argument as a reference to task objects, and is expected to alter its attributes.

For example, this function could apply a specific queue property when using a specific operator, or enforce a task timeout policy, making sure that no tasks run for more than 48 hours. Here's an example of what this may look like inside your `airflow_settings.py`:

```
def policy(task):
    if task.__class__.__name__ == 'HivePartitionSensor':
        task.queue = "sensor_queue"
    if task.timeout > timedelta(hours=48):
        task.timeout = timedelta(hours=48)
```

### 3.8.2.14 Documentation & Notes

It's possible to add documentation or notes to your dags & task objects that become visible in the web interface ("Graph View" for dags, "Task Details" for tasks). There are a set of special task attributes that get rendered as rich content if defined:



attribute	rendered to
doc	monospace
doc_json	json
doc_yaml	yaml
doc_md	markdown
doc_rst	reStructuredText

Please note that for dags, `doc_md` is the only attribute interpreted.

This is especially useful if your tasks are built dynamically from configuration files, it allows you to expose the configuration that led to the related tasks in Airflow.

```
"""
### My great DAG
"""

dag = DAG('my_dag', default_args=default_args)
dag.doc_md = __doc__

t = BashOperator("foo", dag=dag)
t.doc_md = """\
#Title"
Here's a [url](www.airbnb.com)
"""
```

This content will get rendered as markdown respectively in the “Graph View” and “Task Details” pages.

### 3.8.2.15 Jinja Templating

Airflow leverages the power of [Jinja Templating](#) and this can be a powerful tool to use in combination with macros (see the [Macros](#) section).

For example, say you want to pass the execution date as an environment variable to a Bash script using the `BashOperator`.

```
# The execution date as YYYY-MM-DD
date = "{{ ds }}"
t = BashOperator(
    task_id='test_env',
    bash_command='/tmp/test.sh ',
    dag=dag,
    env={'EXECUTION_DATE': date})
```

Here, `{{ ds }}` is a macro, and because the `env` parameter of the `BashOperator` is templated with Jinja, the execution date will be available as an environment variable named `EXECUTION_DATE` in your Bash script.

You can use Jinja templating with every parameter that is marked as “templated” in the documentation. Template substitution occurs just before the `pre_execute` function of your operator is called.

## 3.8.3 Packaged dags

While often you will specify dags in a single `.py` file it might sometimes be required to combine dag and its dependencies. For example, you might want to combine several dags together to version them together or you might want to manage them together or you might need an extra module that is not available by default on the system you are

running airflow on. To allow this you can create a zip file that contains the dag(s) in the root of the zip file and have the extra modules unpacked in directories.

For instance you can create a zip file that looks like this:

```
my_dag1.py
my_dag2.py
package1/__init__.py
package1/functions.py
```

Airflow will scan the zip file and try to load `my_dag1.py` and `my_dag2.py`. It will not go into subdirectories as these are considered to be potential packages.

In case you would like to add module dependencies to your DAG you basically would do the same, but then it is more to use a virtualenv and pip.

```
virtualenv zip_dag
source zip_dag/bin/activate

mkdir zip_dag_contents
cd zip_dag_contents

pip install --install-option="--install-lib=$PWD" my_useful_package
cp ~/my_dag.py .

zip -r zip_dag.zip *
```

---

**Note:** the zip file will be inserted at the beginning of module search list (`sys.path`) and as such it will be available to any other code that resides within the same interpreter.

---

---

**Note:** packaged dags cannot be used with pickling turned on.

---

---

**Note:** packaged dags cannot contain dynamic libraries (eg. `libz.so`) these need to be available on the system if a module needs those. In other words only pure python modules can be packaged.

---

### 3.8.4 .airflowignore

A `.airflowignore` file specifies the directories or files in `DAG_FOLDER` that Airflow should intentionally ignore. Each line in `.airflowignore` specifies a regular expression pattern, and directories or files whose names (not DAG id) match any of the patterns would be ignored (under the hood, `re.findall()` is used to match the pattern). Overall it works like a `.gitignore` file.

`.airflowignore` file should be put in your `DAG_FOLDER`. For example, you can prepare a `.airflowignore` file with contents

```
project_a
tenant_[\d]
```

Then files like “`project_a_dag_1.py`”, “`TESTING_project_a.py`”, “`tenant_1.py`”, “`project_a/dag_1.py`”, and “`tenant_1/dag_1.py`” in your `DAG_FOLDER` would be ignored (If a directory’s name matches any of the patterns, this directory and all its subfolders would not be scanned by Airflow at all. This improves efficiency of DAG finding).

The scope of a `.airflowignore` file is the directory it is in plus all its subfolders. You can also prepare `.airflowignore` file for a subfolder in `DAG_FOLDER` and it would only be applicable for that subfolder.

## 3.9 Command Line Interface

Airflow has a very rich command line interface that allows for many types of operation on a DAG, starting services, and supporting development and testing.

```
usage: airflow [-h]
              {resetdb,render,variables,connections,rotate_fernet_key,users,pause,
↪sync_perm,task_failed_deps,version,trigger_dag,initdb,test,unpause,list_dag_runs,
↪dag_state,run,list_tasks,backfill,list_dags,kerberos,worker,webserver,flower,
↪scheduler,task_state,pool,roles,list_jobs,serve_logs,clear,next_execution,upgradedb,
↪delete_dag}
              ...
```

### 3.9.1 Positional Arguments

<b>subcommand</b>	Possible choices: <code>resetdb</code> , <code>render</code> , <code>variables</code> , <code>connections</code> , <code>rotate_fernet_key</code> , <code>users</code> , <code>pause</code> , <code>sync_perm</code> , <code>task_failed_deps</code> , <code>version</code> , <code>trigger_dag</code> , <code>initdb</code> , <code>test</code> , <code>unpause</code> , <code>list_dag_runs</code> , <code>dag_state</code> , <code>run</code> , <code>list_tasks</code> , <code>backfill</code> , <code>list_dags</code> , <code>kerberos</code> , <code>worker</code> , <code>webserver</code> , <code>flower</code> , <code>scheduler</code> , <code>task_state</code> , <code>pool</code> , <code>roles</code> , <code>list_jobs</code> , <code>serve_logs</code> , <code>clear</code> , <code>next_execution</code> , <code>upgradedb</code> , <code>delete_dag</code>
	<code>sub-command help</code>

### 3.9.2 Sub-commands:

#### 3.9.2.1 resetdb

Burn down and rebuild the metadata database

```
airflow resetdb [-h] [-y]
```

#### Named Arguments

<b>-y, --yes</b>	Do not prompt to confirm reset. Use with care!
	Default: False

#### 3.9.2.2 render

Render a task instance's template(s)

```
airflow render [-h] [-sd SUBDIR] dag_id task_id execution_date
```

## Positional Arguments

<b>dag_id</b>	The id of the dag
<b>task_id</b>	The id of the task
<b>execution_date</b>	The execution date of the DAG

## Named Arguments

<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
----------------------	---

### 3.9.2.3 variables

CRUD operations on variables

```
airflow variables [-h] [-s KEY VAL] [-g KEY] [-j] [-d VAL] [-i FILEPATH]
                  [-e FILEPATH] [-x KEY]
```

## Named Arguments

<b>-s, --set</b>	Set a variable
<b>-g, --get</b>	Get value of a variable
<b>-j, --json</b>	Deserialize JSON variable Default: False
<b>-d, --default</b>	Default value returned if variable does not exist
<b>-i, --import</b>	Import variables from JSON file
<b>-e, --export</b>	Export variables to JSON file
<b>-x, --delete</b>	Delete a variable

### 3.9.2.4 connections

List/Add/Delete connections

```
airflow connections [-h] [-l] [-a] [-d] [--conn_id CONN_ID]
                    [--conn_uri CONN_URI] [--conn_extra CONN_EXTRA]
                    [--conn_type CONN_TYPE] [--conn_host CONN_HOST]
                    [--conn_login CONN_LOGIN] [--conn_password CONN_PASSWORD]
                    [--conn_schema CONN_SCHEMA] [--conn_port CONN_PORT]
```

## Named Arguments

<b>-l, --list</b>	List all connections Default: False
<b>-a, --add</b>	Add a connection Default: False
<b>-d, --delete</b>	Delete a connection Default: False
<b>--conn_id</b>	Connection id, required to add/delete a connection
<b>--conn_uri</b>	Connection URI, required to add a connection without conn_type
<b>--conn_extra</b>	Connection <i>Extra</i> field, optional when adding a connection
<b>--conn_type</b>	Connection type, required to add a connection without conn_uri
<b>--conn_host</b>	Connection host, optional when adding a connection
<b>--conn_login</b>	Connection login, optional when adding a connection
<b>--conn_password</b>	Connection password, optional when adding a connection
<b>--conn_schema</b>	Connection schema, optional when adding a connection
<b>--conn_port</b>	Connection port, optional when adding a connection

### 3.9.2.5 rotate\_fernet\_key

Rotate all encrypted connection credentials and variables; see <https://airflow.readthedocs.io/en/stable/howto/secure-connections.html#rotating-encryption-keys>.

```
airflow rotate_fernet_key [-h]
```

### 3.9.2.6 users

List/Create/Delete/Update users

```
airflow users [-h] [-l] [-c] [-d] [--add-role] [--remove-role] [-i FILEPATH]
               [-e FILEPATH] [--username USERNAME] [--email EMAIL]
               [--firstname FIRSTNAME] [--lastname LASTNAME] [--role ROLE]
               [--password PASSWORD] [--use_random_password]
```

## Named Arguments

<b>-l, --list</b>	List all users Default: False
<b>-c, --create</b>	Create a user Default: False
<b>-d, --delete</b>	Delete a user Default: False

<b>--add-role</b>	Add user to a role Default: False
<b>--remove-role</b>	Remove user from a role Default: False
<b>-i, --import</b>	Import users from JSON file. Example format: [ <pre>{ "email": "foo@bar.org", "firstname": "Jon", "lastname": "Doe",   "roles": ["Public"], "username": "jondoe" }</pre> ]
<b>-e, --export</b>	Export users to JSON file
<b>--username</b>	Username of the user, required to create/delete a user
<b>--email</b>	Email of the user, required to create a user
<b>--firstname</b>	First name of the user, required to create a user
<b>--lastname</b>	Last name of the user, required to create a user
<b>--role</b>	Role of the user. Existing roles include Admin, User, Op, Viewer, and Public. Required to create a user
<b>--password</b>	Password of the user, required to create a user without <code>--use_random_password</code>
<b>--use_random_password</b>	Do not prompt for password. Use random string instead. Required to create a user without <code>--password</code> Default: False

### 3.9.2.7 pause

Pause a DAG

```
airflow pause [-h] [-sd SUBDIR] dag_id
```

### Positional Arguments

**dag\_id** The id of the dag

### Named Arguments

**-sd, --subdir** File location or directory from which to look for the dag. Defaults to `'[AIRFLOW_HOME]/dags'` where `[AIRFLOW_HOME]` is the value you set for `'AIRFLOW_HOME'` config you set in `'airflow.cfg'`  
Default: `"[AIRFLOW_HOME]/dags"`

### 3.9.2.8 sync\_perm

Update permissions for existing roles and DAGs.

```
airflow sync_perm [-h]
```

### 3.9.2.9 task\_failed\_deps

Returns the unmet dependencies for a task instance from the perspective of the scheduler. In other words, why a task instance doesn't get scheduled and then queued by the scheduler, and then run by an executor).

```
airflow task_failed_deps [-h] [-sd SUBDIR] dag_id task_id execution_date
```

#### Positional Arguments

<b>dag_id</b>	The id of the dag
<b>task_id</b>	The id of the task
<b>execution_date</b>	The execution date of the DAG

#### Named Arguments

<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
----------------------	---

### 3.9.2.10 version

Show the version

```
airflow version [-h]
```

### 3.9.2.11 trigger\_dag

Trigger a DAG run

```
airflow trigger_dag [-h] [-sd SUBDIR] [-r RUN_ID] [-c CONF] [-e EXEC_DATE]
                    dag_id
```

#### Positional Arguments

<b>dag_id</b>	The id of the dag
---------------	-------------------

## Named Arguments

<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>-r, --run_id</b>	Helps to identify this run
<b>-c, --conf</b>	JSON string that gets pickled into the DagRun's conf attribute
<b>-e, --exec_date</b>	The execution date of the DAG

### 3.9.2.12 initdb

Initialize the metadata database

```
airflow initdb [-h]
```

### 3.9.2.13 test

Test a task instance. This will run a task without checking for dependencies or recording its state in the database.

```
airflow test [-h] [-sd SUBDIR] [-dr] [-tp TASK_PARAMS]
             dag_id task_id execution_date
```

## Positional Arguments

<b>dag_id</b>	The id of the dag
<b>task_id</b>	The id of the task
<b>execution_date</b>	The execution date of the DAG

## Named Arguments

<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>-dr, --dry_run</b>	Perform a dry run Default: False
<b>-tp, --task_params</b>	Sends a JSON params dict to the task

### 3.9.2.14 unpause

Resume a paused DAG



```
airflow unpause [-h] [-sd SUBDIR] dag_id
```

### Positional Arguments

**dag\_id**                      The id of the dag

### Named Arguments

**-sd, --subdir**              File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

#### 3.9.2.15 list\_dag\_runs

List dag runs given a DAG id. If state option is given, it will only search for all the dagruns with the given state. If no\_backfill option is given, it will filter out all backfill dagruns for given dag id.

```
airflow list_dag_runs [-h] [--no_backfill] [--state STATE] dag_id
```

### Positional Arguments

**dag\_id**                      The id of the dag

### Named Arguments

**--no\_backfill**              filter all the backfill dagruns given the dag id  
Default: False  
**--state**                      Only list the dag runs corresponding to the state

#### 3.9.2.16 dag\_state

Get the status of a dag run

```
airflow dag_state [-h] [-sd SUBDIR] dag_id execution_date
```

### Positional Arguments

**dag\_id**                      The id of the dag  
**execution\_date**              The execution date of the DAG

## Named Arguments

**-sd, --subdir** File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

### 3.9.2.17 run

Run a single task instance

```
airflow run [-h] [-sd SUBDIR] [-m] [-f] [--pool POOL] [--cfg_path CFG_PATH]
            [-l] [-A] [-i] [-I] [--ship_dag] [-p PICKLE] [-int]
            dag_id task_id execution_date
```

## Positional Arguments

**dag\_id** The id of the dag  
**task\_id** The id of the task  
**execution\_date** The execution date of the DAG

## Named Arguments

**-sd, --subdir** File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

**-m, --mark\_success** Mark jobs as succeeded without running them  
Default: False

**-f, --force** Ignore previous task instance state, rerun regardless if task already succeeded/failed  
Default: False

**--pool** Resource pool to use

**--cfg\_path** Path to config file to use instead of airflow.cfg

**-l, --local** Run the task using the LocalExecutor  
Default: False

**-A, --ignore\_all\_dependencies** Ignores all non-critical dependencies, including ignore\_ti\_state and ignore\_task\_deps  
Default: False

**-i, --ignore\_dependencies** Ignore task-specific dependencies, e.g. upstream, depends\_on\_past, and retry delay dependencies  
Default: False

- I, --ignore\_depends\_on\_past** Ignore depends\_on\_past dependencies (but respect upstream dependencies)  
Default: False
- ship\_dag** Pickles (serializes) the DAG and ships it to the worker  
Default: False
- p, --pickle** Serialized pickle object of the entire dag (used internally)
- int, --interactive** Do not capture standard output and error streams (useful for interactive debugging)  
Default: False

### 3.9.2.18 list\_tasks

List the tasks within a DAG

```
airflow list_tasks [-h] [-t] [-sd SUBDIR] dag_id
```

#### Positional Arguments

**dag\_id** The id of the dag

#### Named Arguments

- t, --tree** Tree view  
Default: False
- sd, --subdir** File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

### 3.9.2.19 backfill

Run subsections of a DAG for a specified date range. If reset\_dag\_run option is used, backfill will first prompt users whether airflow should clear all the previous dag\_run and task\_instances within the backfill date range. If rerun\_failed\_tasks is used, backfill will auto re-run the previous failed task instances within the backfill date range.

```
airflow backfill [-h] [-t TASK_REGEX] [-s START_DATE] [-e END_DATE] [-m] [-l]
                  [-x] [-i] [-I] [-sd SUBDIR] [--pool POOL]
                  [--delay_on_limit DELAY_ON_LIMIT] [-dr] [-v] [-c CONF]
                  [--reset_dagruns] [--rerun_failed_tasks]
                  dag_id
```

#### Positional Arguments

**dag\_id** The id of the dag

## Named Arguments

<b>-t, --task_regex</b>	The regex to filter specific task_ids to backfill (optional)
<b>-s, --start_date</b>	Override start_date YYYY-MM-DD
<b>-e, --end_date</b>	Override end_date YYYY-MM-DD
<b>-m, --mark_success</b>	Mark jobs as succeeded without running them Default: False
<b>-l, --local</b>	Run the task using the LocalExecutor Default: False
<b>-x, --donot_pickle</b>	Do not attempt to pickle the DAG object to send over to the workers, just tell the workers to run their version of the code. Default: False
<b>-i, --ignore_dependencies</b>	Skip upstream tasks, run only the tasks matching the regexp. Only works in conjunction with task_regex Default: False
<b>-I, --ignore_first_depends_on_past</b>	Ignores depends_on_past dependencies for the first set of tasks only (subsequent executions in the backfill DO respect depends_on_past). Default: False
<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>--pool</b>	Resource pool to use
<b>--delay_on_limit</b>	Amount of time in seconds to wait when the limit on maximum active dag runs (max_active_runs) has been reached before trying to execute a dag run again. Default: 1.0
<b>-dr, --dry_run</b>	Perform a dry run Default: False
<b>-v, --verbose</b>	Make logging output more verbose Default: False
<b>-c, --conf</b>	JSON string that gets pickled into the DagRun's conf attribute
<b>--reset_dagruns</b>	if set, the backfill will delete existing backfill-related DAG runs and start anew with fresh, running DAG runs Default: False
<b>--rerun_failed_tasks</b>	if set, the backfill will auto-rerun all the failed tasks for the backfill date range instead of throwing exceptions Default: False

### 3.9.2.20 list\_dags

List all the DAGs

```
airflow list_dags [-h] [--sd SUBDIR] [-r]
```

#### Named Arguments

<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>-r, --report</b>	Show DagBag loading report Default: False

### 3.9.2.21 kerberos

Start a kerberos ticket renewer

```
airflow kerberos [-h] [-kt [KEYTAB]] [--pid [PID]] [-D] [--stdout STDOUT]
                  [--stderr STDERR] [-l LOG_FILE]
                  [principal]
```

#### Positional Arguments

<b>principal</b>	kerberos principal
------------------	--------------------

#### Named Arguments

<b>-kt, --keytab</b>	keytab Default: airflow.keytab
<b>--pid</b>	PID file location
<b>-D, --daemon</b>	Daemonize instead of running in the foreground Default: False
<b>--stdout</b>	Redirect stdout to this file
<b>--stderr</b>	Redirect stderr to this file
<b>-l, --log-file</b>	Location of the log file

### 3.9.2.22 worker

Start a Celery worker node

```
airflow worker [-h] [-p] [-q QUEUES] [-c CONCURRENCY] [-cn CELERY_HOSTNAME]
               [--pid [PID]] [-D] [--stdout STDOUT] [--stderr STDERR]
               [-l LOG_FILE] [-a AUTOSCALE]
```

## Named Arguments

<b>-p, --do_pickle</b>	Attempt to pickle the DAG object to send over to the workers, instead of letting workers run their version of the code. Default: False
<b>-q, --queues</b>	Comma delimited list of queues to serve Default: default
<b>-c, --concurrency</b>	The number of worker processes Default: 16
<b>-cn, --celery_hostname</b>	Set the hostname of celery worker if you have multiple workers on a single machine.
<b>--pid</b>	PID file location
<b>-D, --daemon</b>	Daemonize instead of running in the foreground Default: False
<b>--stdout</b>	Redirect stdout to this file
<b>--stderr</b>	Redirect stderr to this file
<b>-l, --log-file</b>	Location of the log file
<b>-a, --autoscale</b>	Minimum and Maximum number of worker to autoscale

### 3.9.2.23 webserver

Start a Airflow webserver instance

```
airflow webserver [-h] [-p PORT] [-w WORKERS]
                  [-k {sync,eventlet,gevent,tornado}] [-t WORKER_TIMEOUT]
                  [-hn HOSTNAME] [--pid [PID]] [-D] [--stdout STDOUT]
                  [--stderr STDERR] [-A ACCESS_LOGFILE] [-E ERROR_LOGFILE]
                  [-l LOG_FILE] [--ssl_cert SSL_CERT] [--ssl_key SSL_KEY] [-d]
```

## Named Arguments

<b>-p, --port</b>	The port on which to run the server Default: 8080
<b>-w, --workers</b>	Number of workers to run the webserver on Default: 4
<b>-k, --workerclass</b>	Possible choices: sync, eventlet, gevent, tornado The worker class to use for Gunicorn Default: sync
<b>-t, --worker_timeout</b>	The timeout for waiting on webserver workers Default: 120

<b>-hn, --hostname</b>	Set the hostname on which to run the web server Default: 0.0.0.0
<b>--pid</b>	PID file location
<b>-D, --daemon</b>	Daemonize instead of running in the foreground Default: False
<b>--stdout</b>	Redirect stdout to this file
<b>--stderr</b>	Redirect stderr to this file
<b>-A, --access_logfile</b>	The logfile to store the webserver access log. Use '-' to print to stderr. Default: -
<b>-E, --error_logfile</b>	The logfile to store the webserver error log. Use '-' to print to stderr. Default: -
<b>-l, --log-file</b>	Location of the log file
<b>--ssl_cert</b>	Path to the SSL certificate for the webserver
<b>--ssl_key</b>	Path to the key to use with the SSL certificate
<b>-d, --debug</b>	Use the server that ships with Flask in debug mode Default: False

### 3.9.2.24 flower

Start a Celery Flower

```
airflow flower [-h] [-hn HOSTNAME] [-p PORT] [-fc FLOWER_CONF] [-u URL_PREFIX]
               [-ba BASIC_AUTH] [-a BROKER_API] [--pid [PID]] [-D]
               [--stdout STDOUT] [--stderr STDERR] [-l LOG_FILE]
```

### Named Arguments

<b>-hn, --hostname</b>	Set the hostname on which to run the server Default: 0.0.0.0
<b>-p, --port</b>	The port on which to run the server Default: 5555
<b>-fc, --flower_conf</b>	Configuration file for flower
<b>-u, --url_prefix</b>	URL prefix for Flower
<b>-ba, --basic_auth</b>	Securing Flower with Basic Authentication. Accepts user:password pairs separated by a comma. Example: flower_basic_auth = user1:password1,user2:password2
<b>-a, --broker_api</b>	Broker api
<b>--pid</b>	PID file location
<b>-D, --daemon</b>	Daemonize instead of running in the foreground Default: False

<b>--stdout</b>	Redirect stdout to this file
<b>--stderr</b>	Redirect stderr to this file
<b>-l, --log-file</b>	Location of the log file

### 3.9.2.25 scheduler

Start a scheduler instance

```
airflow scheduler [-h] [-d DAG_ID] [-sd SUBDIR] [-n NUM_RUNS] [-p]
                  [--pid [PID]] [-D] [--stdout STDOUT] [--stderr STDERR]
                  [-l LOG_FILE]
```

#### Named Arguments

<b>-d, --dag_id</b>	The id of the dag to run
<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>-n, --num_runs</b>	Set the number of runs to execute before exiting Default: -1
<b>-p, --do_pickle</b>	Attempt to pickle the DAG object to send over to the workers, instead of letting workers run their version of the code. Default: False
<b>--pid</b>	PID file location
<b>-D, --daemon</b>	Daemonize instead of running in the foreground Default: False
<b>--stdout</b>	Redirect stdout to this file
<b>--stderr</b>	Redirect stderr to this file
<b>-l, --log-file</b>	Location of the log file

### 3.9.2.26 task\_state

Get the status of a task instance

```
airflow task_state [-h] [-sd SUBDIR] dag_id task_id execution_date
```

#### Positional Arguments

<b>dag_id</b>	The id of the dag
<b>task_id</b>	The id of the task
<b>execution_date</b>	The execution date of the DAG



## Named Arguments

**-sd, --subdir** File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

### 3.9.2.27 pool

CRUD operations on pools

```
airflow pool [-h] [-s NAME SLOT_COUNT POOL_DESCRIPTION] [-g NAME] [-x NAME]
             [-i FILEPATH] [-e FILEPATH]
```

## Named Arguments

**-s, --set** Set pool slot count and description, respectively  
**-g, --get** Get pool info  
**-x, --delete** Delete a pool  
**-i, --import** Import pool from JSON file  
**-e, --export** Export pool to JSON file

### 3.9.2.28 roles

Create/List roles

```
airflow roles [-h] [-c] [-l] [role [role ...]]
```

## Positional Arguments

**role** The name of a role

## Named Arguments

**-c, --create** Create a new role  
Default: False  
**-l, --list** List roles  
Default: False

### 3.9.2.29 list\_jobs

List the jobs

```
airflow list_jobs [-h] [-d DAG_ID] [--state STATE] [--limit LIMIT]
```

### Named Arguments

<b>-d, --dag_id</b>	The id of the dag to run
<b>--state</b>	Only list the dag runs corresponding to the state
<b>--limit</b>	Return a limited number of records

### 3.9.2.30 serve\_logs

Serve logs generate by worker

```
airflow serve_logs [-h]
```

### 3.9.2.31 clear

Clear a set of task instance, as if they never ran

```
airflow clear [-h] [-t TASK_REGEX] [-s START_DATE] [-e END_DATE] [-sd SUBDIR]
              [-u] [-d] [-c] [-f] [-r] [-x] [-xp] [-dx]
              dag_id
```

### Positional Arguments

<b>dag_id</b>	The id of the dag
---------------	-------------------

### Named Arguments

<b>-t, --task_regex</b>	The regex to filter specific task_ids to backfill (optional)
<b>-s, --start_date</b>	Override start_date YYYY-MM-DD
<b>-e, --end_date</b>	Override end_date YYYY-MM-DD
<b>-sd, --subdir</b>	File location or directory from which to look for the dag. Defaults to '[AIRFLOW_HOME]/dags' where [AIRFLOW_HOME] is the value you set for 'AIRFLOW_HOME' config you set in 'airflow.cfg' Default: "[AIRFLOW_HOME]/dags"
<b>-u, --upstream</b>	Include upstream tasks Default: False
<b>-d, --downstream</b>	Include downstream tasks Default: False
<b>-c, --no_confirm</b>	Do not request confirmation Default: False

- f, --only\_failed** Only failed jobs  
Default: False
- r, --only\_running** Only running jobs  
Default: False
- x, --exclude\_subdags** Exclude subdags  
Default: False
- xp, --exclude\_parentdag** Exclude ParentDAGS if the task cleared is a part of a SubDAG  
Default: False
- dx, --dag\_regex** Search dag\_id as regex instead of exact string  
Default: False

### 3.9.2.32 next\_execution

Get the next execution datetime of a DAG.

```
airflow next_execution [-h] [-sd SUBDIR] dag_id
```

#### Positional Arguments

**dag\_id** The id of the dag

#### Named Arguments

**-sd, --subdir** File location or directory from which to look for the dag. Defaults to '[AIRFLOW\_HOME]/dags' where [AIRFLOW\_HOME] is the value you set for 'AIRFLOW\_HOME' config you set in 'airflow.cfg'  
Default: "[AIRFLOW\_HOME]/dags"

### 3.9.2.33 upgradedb

Upgrade the metadata database to latest version

```
airflow upgradedb [-h]
```

### 3.9.2.34 delete\_dag

Delete all DB records related to the specified DAG

```
airflow delete_dag [-h] [-y] dag_id
```

#### Positional Arguments

**dag\_id** The id of the dag

## Named Arguments

**-y, --yes** Do not prompt to confirm reset. Use with care!  
Default: False

## 3.10 Scheduling & Triggers

The Airflow scheduler monitors all tasks and all DAGs, and triggers the task instances whose dependencies have been met. Behind the scenes, it spins up a subprocess, which monitors and stays in sync with a folder for all DAG objects it may contain, and periodically (every minute or so) collects DAG parsing results and inspects active tasks to see whether they can be triggered.

The Airflow scheduler is designed to run as a persistent service in an Airflow production environment. To kick it off, all you need to do is execute `airflow scheduler`. It will use the configuration specified in `airflow.cfg`.

Note that if you run a DAG on a `schedule_interval` of one day, the run stamped 2016-01-01 will be triggered soon after 2016-01-01T23:59. In other words, the job instance is started once the period it covers has ended.

**Let’s Repeat That** The scheduler runs your job one `schedule_interval` AFTER the start date, at the END of the period.

The scheduler starts an instance of the executor specified in the your `airflow.cfg`. If it happens to be the `airflow.contrib.executors.local_executor.LocalExecutor`, tasks will be executed as subprocesses; in the case of `airflow.executors.celery_executor.CeleryExecutor`, `airflow.executors.dask_executor.DaskExecutor`, and `airflow.contrib.executors.mesos_executor.MesosExecutor`, tasks are executed remotely.

To start a scheduler, simply run the command:

```
airflow scheduler
```

### 3.10.1 DAG Runs

A DAG Run is an object representing an instantiation of the DAG in time.

Each DAG may or may not have a schedule, which informs how DAG Runs are created. `schedule_interval` is defined as a DAG arguments, and receives preferably a `cron expression` as a `str`, or a `datetime.timedelta` object. Alternatively, you can also use one of these cron “preset”:

preset	meaning	cron
None	Don’t schedule, use for exclusively “externally triggered” DAGs	
@once	Schedule once and only once	
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@daily	Run once a day at midnight	0 0 * * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@yearly	Run once a year at midnight of January 1	0 0 1 1 *

**Note:** Use `schedule_interval=None` and not `schedule_interval='None'` when you don’t want to schedule your DAG.

Your DAG will be instantiated for each schedule, while creating a DAG Run entry for each schedule.

DAG runs have a state associated to them (running, failed, success) and informs the scheduler on which set of schedules should be evaluated for task submissions. Without the metadata at the DAG run level, the Airflow scheduler would have much more work to do in order to figure out what tasks should be triggered and come to a crawl. It might also create undesired processing when changing the shape of your DAG, by say adding in new tasks.

### 3.10.2 Backfill and Catchup

An Airflow DAG with a `start_date`, possibly an `end_date`, and a `schedule_interval` defines a series of intervals which the scheduler turn into individual Dag Runs and execute. A key capability of Airflow is that these DAG Runs are atomic, idempotent items, and the scheduler, by default, will examine the lifetime of the DAG (from start to end/now, one interval at a time) and kick off a DAG Run for any interval that has not been run (or has been cleared). This concept is called Catchup.

If your DAG is written to handle its own catchup (IE not limited to the interval, but instead to “Now” for instance.), then you will want to turn catchup off (Either on the DAG itself with `dag.catchup = False`) or by default at the configuration file level with `catchup_by_default = False`. What this will do, is to instruct the scheduler to only create a DAG Run for the most current instance of the DAG interval series.

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/airflow/blob/master/airflow/example_dags/tutorial.py
"""
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 12, 1),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5)
}

dag = DAG(
    'tutorial',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval='@daily',
    catchup=False)
```

In the example above, if the DAG is picked up by the scheduler daemon on 2016-01-02 at 6 AM, (or from the command line), a single DAG Run will be created, with an `execution_date` of 2016-01-01, and the next one will be created just after midnight on the morning of 2016-01-03 with an execution date of 2016-01-02.

If the `dag.catchup` value had been `True` instead, the scheduler would have created a DAG Run for each completed interval between 2015-12-01 and 2016-01-02 (but not yet one for 2016-01-02, as that interval hasn’t completed) and the scheduler will execute them sequentially. This behavior is great for atomic datasets that can easily be split into periods. Turning catchup off is great if your DAG Runs perform backfill internally.

### 3.10.3 External Triggers

Note that DAG Runs can also be created manually through the CLI while running an `airflow trigger_dag` command, where you can define a specific `run_id`. The DAG Runs created externally to the scheduler get associated to the trigger's timestamp, and will be displayed in the UI alongside scheduled DAG runs.

In addition, you can also manually trigger a DAG Run using the web UI (tab "DAGs" -> column "Links" -> button "Trigger Dag").

### 3.10.4 To Keep in Mind

- The first DAG Run is created based on the minimum `start_date` for the tasks in your DAG.
- Subsequent DAG Runs are created by the scheduler process, based on your DAG's `schedule_interval`, sequentially.
- When clearing a set of tasks' state in hope of getting them to re-run, it is important to keep in mind the DAG Run's state too as it defines whether the scheduler should look into triggering tasks for that run.

Here are some of the ways you can **unblock tasks**:

- From the UI, you can **clear** (as in delete the status of) individual task instances from the task instances dialog, while defining whether you want to include the past/future and the upstream/downstream dependencies. Note that a confirmation window comes next and allows you to see the set you are about to clear. You can also clear all task instances associated with the dag.
- The CLI command `airflow clear -h` has lots of options when it comes to clearing task instance states, including specifying date ranges, targeting `task_ids` by specifying a regular expression, flags for including upstream and downstream relatives, and targeting task instances in specific states (`failed`, or `success`)
- Clearing a task instance will no longer delete the task instance record. Instead it updates `max_tries` and set the current task instance state to be `None`.
- Marking task instances as failed can be done through the UI. This can be used to stop running task instances.
- Marking task instances as successful can be done through the UI. This is mostly to fix false negatives, or for instance when the fix has been applied outside of Airflow.
- The `airflow backfill` CLI subcommand has a flag to `--mark_success` and allows selecting subsections of the DAG as well as specifying date ranges.

## 3.11 Plugins

Airflow has a simple plugin manager built-in that can integrate external features to its core by simply dropping files in your `$AIRFLOW_HOME/plugins` folder.

The python modules in the `plugins` folder get imported, and **hooks**, **operators**, **sensors**, **macros**, **executors** and web **views** get integrated to Airflow's main collections and become available for use.

### 3.11.1 What for?

Airflow offers a generic toolbox for working with data. Different organizations have different stacks and different needs. Using Airflow plugins can be a way for companies to customize their Airflow installation to reflect their ecosystem.

Plugins can be used as an easy way to write, share and activate new sets of features.

There's also a need for a set of more complex applications to interact with different flavors of data and metadata.

Examples:

- A set of tools to parse Hive logs and expose Hive metadata (CPU /IO / phases/ skew /...)
- An anomaly detection framework, allowing people to collect metrics, set thresholds and alerts
- An auditing tool, helping understand who accesses what
- A config-driven SLA monitoring tool, allowing you to set monitored tables and at what time they should land, alert people, and expose visualizations of outages
- ...

### 3.11.2 Why build on top of Airflow?

Airflow has many components that can be reused when building an application:

- A web server you can use to render your views
- A metadata database to store your models
- Access to your databases, and knowledge of how to connect to them
- An array of workers that your application can push workload to
- Airflow is deployed, you can just piggy back on its deployment logistics
- Basic charting capabilities, underlying libraries and abstractions

### 3.11.3 Interface

To create a plugin you will need to derive the `airflow.plugins_manager.AirflowPlugin` class and reference the objects you want to plug into Airflow. Here's what the class you need to derive looks like:

```
class AirflowPlugin(object):
    # The name of your plugin (str)
    name = None
    # A list of class(es) derived from BaseOperator
    operators = []
    # A list of class(es) derived from BaseSensorOperator
    sensors = []
    # A list of class(es) derived from BaseHook
    hooks = []
    # A list of class(es) derived from BaseExecutor
    executors = []
    # A list of references to inject into the macros namespace
    macros = []
    # A list of objects created from a class derived
    # from flask_admin.BaseView
    admin_views = []
    # A list of Blueprint object created from flask.Blueprint. For use with the flask_
    ↪admin based GUI
    flask_blueprints = []
    # A list of menu links (flask_admin.base.MenuLink). For use with the flask_admin_
    ↪based GUI
    menu_links = []
    # A list of dictionaries containing FlaskAppBuilder BaseView object and some_
    ↪metadata. See example below
```

(continues on next page)

(continued from previous page)

```

appbuilder_views = []
# A list of dictionaries containing FlaskAppBuilder BaseView object and some_
↪metadata. See example below
appbuilder_menu_items = []

```

You can derive it by inheritance (please refer to the example below). Please note name inside this class must be specified.

After the plugin is imported into Airflow, you can invoke it using statement like

```

from airflow.{type, like "operators", "sensors"}.{name specified inside the plugin_
↪class} import *

```

When you write your own plugins, make sure you understand them well. There are some essential properties for each type of plugin. For example,

- For Operator plugin, an execute method is compulsory.
- For Sensor plugin, a poke method returning a Boolean value is compulsory.

Make sure you restart the webserver and scheduler after making changes to plugins so that they take effect.

### 3.11.4 Example

The code below defines a plugin that injects a set of dummy object definitions in Airflow.

```

# This is the class you derive to create a plugin
from airflow.plugins_manager import AirflowPlugin

from flask import Blueprint
from flask_admin import BaseView, expose
from flask_admin.base import MenuLink
from flask_appbuilder import BaseView as AppBuilderBaseView

# Importing base classes that we need to derive
from airflow.hooks.base_hook import BaseHook
from airflow.models import BaseOperator
from airflow.sensors.base_sensor_operator import BaseSensorOperator
from airflow.executors.base_executor import BaseExecutor

# Will show up under airflow.hooks.test_plugin.PluginHook
class PluginHook(BaseHook):
    pass

# Will show up under airflow.operators.test_plugin.PluginOperator
class PluginOperator(BaseOperator):
    pass

# Will show up under airflow.sensors.test_plugin.PluginSensorOperator
class PluginSensorOperator(BaseSensorOperator):
    pass

# Will show up under airflow.executors.test_plugin.PluginExecutor
class PluginExecutor(BaseExecutor):
    pass

```

(continues on next page)



(continued from previous page)

```

# Will show up under airflow.macros.test_plugin.plugin_macro
def plugin_macro():
    pass

# Creating a flask admin BaseView
class TestView(BaseView):
    @expose('/')
    def test(self):
        # in this example, put your test_plugin/test.html template at airflow/plugins/
        ↪ templates/test_plugin/test.html
        return self.render("test_plugin/test.html", content="Hello galaxy!")
v = TestView(category="Test Plugin", name="Test View")

# Creating a flask blueprint to integrate the templates and static folder
bp = Blueprint(
    "test_plugin", __name__,
    template_folder='templates', # registers airflow/plugins/templates as a Jinja_
    ↪ template folder
    static_folder='static',
    static_url_path='/static/test_plugin')

ml = MenuLink(
    category='Test Plugin',
    name='Test Menu Link',
    url='https://airflow.apache.org/')

# Creating a flask appbuilder BaseView
class TestAppBuilderBaseView(AppBuilderBaseView):
    default_view = "test"

    @expose("/")
    def test(self):
        return self.render("test_plugin/test.html", content="Hello galaxy!")
v_appbuilder_view = TestAppBuilderBaseView()
v_appbuilder_package = {"name": "Test View",
                        "category": "Test Plugin",
                        "view": v_appbuilder_view}

# Creating a flask appbuilder Menu Item
appbuilder_mitem = {"name": "Google",
                    "category": "Search",
                    "category_icon": "fa-th",
                    "href": "https://www.google.com"}

# Defining the plugin class
class AirflowTestPlugin(AirflowPlugin):
    name = "test_plugin"
    operators = [PluginOperator]
    sensors = [PluginSensorOperator]
    hooks = [PluginHook]
    executors = [PluginExecutor]
    macros = [plugin_macro]
    admin_views = [v]
    flask_blueprints = [bp]
    menu_links = [ml]
    appbuilder_views = [v_appbuilder_package]
    appbuilder_menu_items = [appbuilder_mitem]

```

### 3.11.5 Note on role based views

Airflow 1.10 introduced role based views using FlaskAppBuilder. You can configure which UI is used by setting `rbac = True`. To support plugin views and links for both versions of the UI and maintain backwards compatibility, the fields `appbuilder_views` and `appbuilder_menu_items` were added to the `AirflowTestPlugin` class.

### 3.11.6 Plugins as Python packages

It is possible to load plugins via `setuptools` `entrypoint` mechanism. To do this link your plugin using an entrypoint in your package. If the package is installed, airflow will automatically load the registered plugins from the entrypoint list.

**Note:** Neither the entrypoint name (eg, `my_plugin`) nor the name of the plugin class will contribute towards the module and class name of the plugin itself. The structure is determined by `airflow.plugins_manager.AirflowPlugin.name` and the class name of the plugin component with the pattern `airflow.{component}.{name}.{component_class_name}`.

```
# my_package/my_plugin.py
from airflow.plugins_manager import AirflowPlugin
from airflow.models import BaseOperator
from airflow.hooks.base_hook import BaseHook

class MyOperator(BaseOperator):
    pass

class MyHook(BaseHook):
    pass

class MyAirflowPlugin(AirflowPlugin):
    name = 'my_namespace'
    operators = [MyOperator]
    hooks = [MyHook]
```

```
from setuptools import setup

setup(
    name="my-package",
    ...
    entry_points = {
        'airflow.plugins': [
            'my_plugin = my_package.my_plugin:MyAirflowPlugin'
        ]
    }
)
```

This will create a hook, and an operator accessible at:

- `airflow.hooks.my_namespace.MyHook`
- `airflow.operators.my_namespace.MyOperator`

## 3.12 Security

By default, all gates are opened. An easy way to restrict access to the web application is to do it at the network level, or by using SSH tunnels.

It is however possible to switch on authentication by either using one of the supplied backends or creating your own.

Be sure to checkout *Experimental Rest API* for securing the API.

---

**Note:** Airflow uses the config parser of Python. This config parser interpolates ‘%’-signs. Make sure escape any % signs in your config file (but not environment variables) as %, otherwise Airflow might leak these passwords on a config parser exception to a log.

---

### 3.12.1 Reporting Vulnerabilities

The Apache Software Foundation takes security issues very seriously. Apache Airflow specifically offers security features and is responsive to issues around its features. If you have any concern around Airflow Security or believe you have uncovered a vulnerability, we suggest that you get in touch via the e-mail address [security@apache.org](mailto:security@apache.org). In the message, try to provide a description of the issue and ideally a way of reproducing it. The security team will get back to you after assessing the description.

Note that this security address should be used only for undisclosed vulnerabilities. Dealing with fixed issues or general questions on how to use the security features should be handled regularly via the user and the dev lists. Please report any security problems to the project security address before disclosing it publicly.

The [ASF Security team’s page](#) describes how vulnerability reports are handled, and includes PGP keys if you wish to use that.

### 3.12.2 Web Authentication

#### 3.12.2.1 Password

One of the simplest mechanisms for authentication is requiring users to specify a password before logging in.

Please use command line interface `airflow users --create` to create accounts, or do that in the UI.

#### 3.12.2.2 LDAP

To turn on LDAP authentication configure your `airflow.cfg` as follows. Please note that the example uses an encrypted connection to the ldap server as we do not want passwords be readable on the network level.

Additionally, if you are using Active Directory, and are not explicitly specifying an OU that your users are in, you will need to change `search_scope` to “SUBTREE”.

Valid `search_scope` options can be found in the [ldap3 Documentation](#)

```
[webserver]
authenticate = True
auth_backend = airflow.contrib.auth.backends.ldap_auth

[ldap]
# set a connection without encryption: uri = ldap://<your.ldap.server>:<port>
uri = ldaps://<your.ldap.server>:<port>
```

(continues on next page)

(continued from previous page)

```

user_filter = objectClass=*
# in case of Active Directory you would use: user_name_attr = sAMAccountName
user_name_attr = uid
# group_member_attr should be set accordingly with *_filter
# eg :
#     group_member_attr = groupMembership
#     superuser_filter = groupMembership=CN=airflow-super-users...
group_member_attr = memberOf
superuser_filter = memberOf=CN=airflow-super-users,OU=Groups,OU=RWC,OU=US,OU=NORAM,
↳DC=example,DC=com
data_profiler_filter = memberOf=CN=airflow-data-profilers,OU=Groups,OU=RWC,OU=US,
↳OU=NORAM,DC=example,DC=com
bind_user = cn=Manager,dc=example,dc=com
bind_password = insecure
basedn = dc=example,dc=com
cacert = /etc/ca/ldap_ca.crt
# Set search_scope to one of them:  BASE, LEVEL , SUBTREE
# Set search_scope to SUBTREE if using Active Directory, and not specifying an_
↳Organizational Unit
search_scope = LEVEL

```

The `superuser_filter` and `data_profiler_filter` are optional. If defined, these configurations allow you to specify LDAP groups that users must belong to in order to have superuser (admin) and data-profiler permissions. If undefined, all users will be superusers and data profilers.

### 3.12.2.3 Roll your own

Airflow uses `flask_login` and exposes a set of hooks in the `airflow.default_login` module. You can alter the content and make it part of the `PYTHONPATH` and configure it as a backend in `airflow.cfg`.

```

[webserver]
authenticate = True
auth_backend = mypackage.auth

```

### 3.12.3 Multi-tenancy

You can filter the list of dags in webserver by owner name when authentication is turned on by setting `webserver:filter_by_owner` in your config. With this, a user will see only the dags which it is owner of, unless it is a superuser.

```

[webserver]
filter_by_owner = True

```

### 3.12.4 Kerberos

Airflow has initial support for Kerberos. This means that airflow can renew kerberos tickets for itself and store it in the ticket cache. The hooks and dags can make use of ticket to authenticate against kerberized services.

### 3.12.4.1 Limitations

Please note that at this time, not all hooks have been adjusted to make use of this functionality. Also it does not integrate kerberos into the web interface and you will have to rely on network level security for now to make sure your service remains secure.

Celery integration has not been tried and tested yet. However, if you generate a key tab for every host and launch a ticket renewer next to every worker it will most likely work.

### 3.12.4.2 Enabling kerberos

#### Airflow

To enable kerberos you will need to generate a (service) key tab.

```
# in the kadmin.local or kadmin shell, create the airflow principal
kadmin: addprinc -randkey airflow/fully.qualified.domain.name@YOUR-REALM.COM

# Create the airflow keytab file that will contain the airflow principal
kadmin: xst -norandkey -k airflow.keytab airflow/fully.qualified.domain.name
```

Now store this file in a location where the airflow user can read it (chmod 600). And then add the following to your `airflow.cfg`

```
[core]
security = kerberos

[kerberos]
keytab = /etc/airflow/airflow.keytab
reinit_frequency = 3600
principal = airflow
```

Launch the ticket renewer by

```
# run ticket renewer
airflow kerberos
```

#### Hadoop

If want to use impersonation this needs to be enabled in `core-site.xml` of your hadoop config.

```
<property>
  <name>hadoop.proxyuser.airflow.groups</name>
  <value>*</value>
</property>

<property>
  <name>hadoop.proxyuser.airflow.users</name>
  <value>*</value>
</property>

<property>
  <name>hadoop.proxyuser.airflow.hosts</name>
  <value>*</value>
</property>
```

Of course if you need to tighten your security replace the asterisk with something more appropriate.

### 3.12.4.3 Using kerberos authentication

The hive hook has been updated to take advantage of kerberos authentication. To allow your DAGs to use it, simply update the connection details with, for example:

```
{ "use_beeline": true, "principal": "hive/_HOST@EXAMPLE.COM" }
```

Adjust the principal to your settings. The `_HOST` part will be replaced by the fully qualified domain name of the server.

You can specify if you would like to use the dag owner as the user for the connection or the user specified in the login section of the connection. For the login user, specify the following as extra:

```
{ "use_beeline": true, "principal": "hive/_HOST@EXAMPLE.COM", "proxy_user": "login" }
```

For the DAG owner use:

```
{ "use_beeline": true, "principal": "hive/_HOST@EXAMPLE.COM", "proxy_user": "owner" }
```

and in your DAG, when initializing the `HiveOperator`, specify:

```
run_as_owner=True
```

To use kerberos authentication, you must install Airflow with the *kerberos* extras group:

```
pip install apache-airflow[kerberos]
```

## 3.12.5 OAuth Authentication

### 3.12.5.1 GitHub Enterprise (GHE) Authentication

The GitHub Enterprise authentication backend can be used to authenticate users against an installation of GitHub Enterprise using OAuth2. You can optionally specify a team whitelist (composed of slug cased team names) to restrict login to only members of those teams.

```
[webserver]
authenticate = True
auth_backend = airflow.contrib.auth.backends.github_enterprise_auth

[github_enterprise]
host = github.example.com
client_id = oauth_key_from_github_enterprise
client_secret = oauth_secret_from_github_enterprise
oauth_callback_route = /example/ghe_oauth/callback
allowed_teams = 1, 345, 23
```

---

**Note:** If you do not specify a team whitelist, anyone with a valid account on your GHE installation will be able to login to Airflow.

---

To use GHE authentication, you must install Airflow with the *github\_enterprise* extras group:

```
pip install apache-airflow[github_enterprise]
```

## Setting up GHE Authentication

An application must be setup in GHE before you can use the GHE authentication backend. In order to setup an application:

1. Navigate to your GHE profile
2. Select 'Applications' from the left hand nav
3. Select the 'Developer Applications' tab
4. Click 'Register new application'
5. Fill in the required information (the 'Authorization callback URL' must be fully qualified e.g. [http://airflow.example.com/example/ghe\\_oauth/callback](http://airflow.example.com/example/ghe_oauth/callback))
6. Click 'Register application'
7. Copy 'Client ID', 'Client Secret', and your callback route to your airflow.cfg according to the above example

## Using GHE Authentication with github.com

It is possible to use GHE authentication with github.com:

1. [Create an OAuth App](#)
2. Copy 'Client ID', 'Client Secret' to your airflow.cfg according to the above example
3. Set `host = github.com` and `oauth_callback_route = /oauth/callback` in airflow.cfg

### 3.12.5.2 Google Authentication

The Google authentication backend can be used to authenticate users against Google using OAuth2. You must specify the email domains to restrict login, separated with a comma, to only members of those domains.

```
[webserver]
authenticate = True
auth_backend = airflow.contrib.auth.backends.google_auth

[google]
client_id = google_client_id
client_secret = google_client_secret
oauth_callback_route = /oauth2callback
domain = example1.com,example2.com
```

To use Google authentication, you must install Airflow with the `google_auth` extras group:

```
pip install apache-airflow[google_auth]
```

## Setting up Google Authentication

An application must be setup in the Google API Console before you can use the Google authentication backend. In order to setup an application:

1. Navigate to <https://console.developers.google.com/apis/>
2. Select ‘Credentials’ from the left hand nav
3. Click ‘Create credentials’ and choose ‘OAuth client ID’
4. Choose ‘Web application’
5. Fill in the required information (the ‘Authorized redirect URIs’ must be fully qualified e.g. <http://airflow.example.com/oauth2callback>)
6. Click ‘Create’
7. Copy ‘Client ID’, ‘Client Secret’, and your redirect URI to your `airflow.cfg` according to the above example

### 3.12.6 SSL

SSL can be enabled by providing a certificate and key. Once enabled, be sure to use “<https://>” in your browser.

```
[webserver]
web_server_ssl_cert = <path to cert>
web_server_ssl_key = <path to key>
```

Enabling SSL will not automatically change the web server port. If you want to use the standard port 443, you’ll need to configure that too. Be aware that super user privileges (or `cap_net_bind_service` on Linux) are required to listen on port 443.

```
# Optionally, set the server to listen on the standard SSL port.
web_server_port = 443
base_url = http://<hostname or IP>:443
```

Enable CeleryExecutor with SSL. Ensure you properly generate client and server certs and keys.

```
[celery]
ssl_active = True
ssl_key = <path to key>
ssl_cert = <path to cert>
ssl_cacert = <path to cacert>
```

### 3.12.7 Impersonation

Airflow has the ability to impersonate a unix user while running task instances based on the task’s `run_as_user` parameter, which takes a user’s name.

**NOTE:** For impersonations to work, Airflow must be run with `sudo` as subtasks are run with `sudo -u` and permissions of files are changed. Furthermore, the unix user needs to exist on the worker. Here is what a simple sudoers file entry could look like to achieve this, assuming as airflow is running as the `airflow` user. Note that this means that the airflow user must be trusted and treated the same way as the root user.

```
airflow ALL=(ALL) NOPASSWD: ALL
```

Subtasks with impersonation will still log to the same folder, except that the files they log to will have permissions changed such that only the unix user can write to it.



### 3.12.7.1 Default Impersonation

To prevent tasks that don't use impersonation to be run with *sudo* privileges, you can set the `core:default_impersonation` config which sets a default user impersonate if *run\_as\_user* is not set.

```
[core]
default_impersonation = airflow
```

### 3.12.8 Flower Authentication

Basic authentication for Celery Flower is supported.

You can specify the details either as an optional argument in the Flower process launching command, or as a configuration item in your `airflow.cfg`. For both cases, please provide *user:password* pairs separated by a comma.

```
airflow flower --basic_auth=user1:password1,user2:password2
```

```
[celery]
flower_basic_auth = user1:password1,user2:password2
```

### 3.12.9 RBAC UI Security

Security of Airflow Webserver UI is handled by Flask AppBuilder (FAB). Please read its related [security document](#) regarding its security model.

#### 3.12.9.1 Default Roles

Airflow ships with a set of roles by default: Admin, User, Op, Viewer, and Public. Only Admin users could configure/alter the permissions for other roles. But it is not recommended that Admin users alter these default roles in any way by removing or adding permissions to these roles.

#### Admin

Admin users have all possible permissions, including granting or revoking permissions from other users.

#### Public

Public users (anonymous) don't have any permissions.

#### Viewer

Viewer users have limited viewer permissions

```
VIEWER_PERMS = {
    'menu_access',
    'can_index',
    'can_list',
    'can_show',
```

(continues on next page)

(continued from previous page)

```

'can_chart',
'can_dag_stats',
'can_dag_details',
'can_task_stats',
'can_code',
'can_log',
'can_get_logs_with_metadata',
'can_tries',
'can_graph',
'can_tree',
'can_task',
'can_task_instances',
'can_xcom',
'can_gantt',
'can_landing_times',
'can_duration',
'can_blocked',
'can_rendered',
'can_pickle_info',
'can_version',
}

```

on limited web views

```

VIEWER_VMS = {
    'Airflow',
    'DagModelView',
    'Browse',
    'DAG Runs',
    'DagRunModelView',
    'Task Instances',
    'TaskInstanceModelView',
    'SLA Misses',
    'SlaMissModelView',
    'Jobs',
    'JobModelView',
    'Logs',
    'LogModelView',
    'Docs',
    'Documentation',
    'Github',
    'About',
    'Version',
    'VersionView',
}

```

## User

User users have Viewer permissions plus additional user permissions

```

USER_PERMS = {
    'can_dagrun_clear',
    'can_run',
    'can_trigger',
    'can_add',
}

```

(continues on next page)

(continued from previous page)

```

    'can_edit',
    'can_delete',
    'can_paused',
    'can_refresh',
    'can_success',
    'muldelete',
    'set_failed',
    'set_running',
    'set_success',
    'clear',
    'can_clear',
}

```

on User web views which is the same as Viewer web views.

## Op

Op users have User permissions plus additional op permissions

```

OP_PERMS = {
    'can_conf',
    'can_varimport',
}

```

on User web views plus these additional op web views

```

OP_VMS = {
    'Admin',
    'Configurations',
    'ConfigurationView',
    'Connections',
    'ConnectionModelView',
    'Pools',
    'PoolModelView',
    'Variables',
    'VariableModelView',
    'XComs',
    'XComModelView',
}

```

### 3.12.9.2 Custom Roles

#### DAG Level Role

Admin can create a set of roles which are only allowed to view a certain set of dags. This is called DAG level access. Each dag defined in the dag model table is treated as a View which has two permissions associated with it (can\_dag\_read and can\_dag\_edit). There is a special view called all\_dags which allows the role to access all the dags. The default Admin, Viewer, User, Op roles can all access all\_dags view.

## 3.13 Time zones

Support for time zones is enabled by default. Airflow stores datetime information in UTC internally and in the database. It allows you to run your DAGs with time zone dependent schedules. At the moment Airflow does not convert them to the end user's time zone in the user interface. There it will always be displayed in UTC. Also templates used in Operators are not converted. Time zone information is exposed and it is up to the writer of DAG what do with it.

This is handy if your users live in more than one time zone and you want to display datetime information according to each user's wall clock.

Even if you are running Airflow in only one time zone it is still good practice to store data in UTC in your database (also before Airflow became time zone aware this was also to recommended or even required setup). The main reason is Daylight Saving Time (DST). Many countries have a system of DST, where clocks are moved forward in spring and backward in autumn. If you're working in local time, you're likely to encounter errors twice a year, when the transitions happen. (The *pendulum* and *pytz* documentation discusses these issues in greater detail.) This probably doesn't matter for a simple DAG, but it's a problem if you are in, for example, financial services where you have end of day deadlines to meet.

The time zone is set in *airflow.cfg*. By default it is set to *utc*, but you change it to use the system's settings or an arbitrary IANA time zone, e.g. *Europe/Amsterdam*. It is dependent on *pendulum*, which is more accurate than *pytz*. *Pendulum* is installed when you install Airflow.

Please note that the Web UI currently only runs in UTC.

### 3.13.1 Concepts

#### 3.13.1.1 Naïve and aware datetime objects

Python's `datetime.datetime` objects have a `tzinfo` attribute that can be used to store time zone information, represented as an instance of a subclass of `datetime.tzinfo`. When this attribute is set and describes an offset, a datetime object is aware. Otherwise, it's naive.

You can use `timezone.is_localized()` and `timezone.is_naive()` to determine whether datetimes are aware or naive.

Because Airflow uses time-zone-aware datetime objects. If your code creates datetime objects they need to be aware too.

```
from airflow.utils import timezone

now = timezone.utcnow()
a_date = timezone.datetime(2017,1,1)
```

#### 3.13.1.2 Interpretation of naive datetime objects

Although Airflow operates fully time zone aware, it still accepts naive date time objects for *start\_dates* and *end\_dates* in your DAG definitions. This is mostly in order to preserve backwards compatibility. In case a naive *start\_date* or *end\_date* is encountered the default time zone is applied. It is applied in such a way that it is assumed that the naive date time is already in the default time zone. In other words if you have a default time zone setting of *Europe/Amsterdam* and create a naive datetime *start\_date* of `datetime(2017,1,1)` it is assumed to be a *start\_date* of Jan 1, 2017 Amsterdam time.

```
default_args=dict(
    start_date=datetime(2016, 1, 1),
    owner='Airflow'
)

dag = DAG('my_dag', default_args=default_args)
op = DummyOperator(task_id='dummy', dag=dag)
print(op.owner) # Airflow
```

Unfortunately, during DST transitions, some datetimes don't exist or are ambiguous. In such situations, *pendulum* raises an exception. That's why you should always create aware datetime objects when time zone support is enabled.

In practice, this is rarely an issue. Airflow gives you aware datetime objects in the models and DAGs, and most often, new datetime objects are created from existing ones through *timedelta* arithmetic. The only datetime that's often created in application code is the current time, and *timezone.utcnow()* automatically does the right thing.

### 3.13.1.3 Default time zone

The default time zone is the time zone defined by the *default\_timezone* setting under *[core]*. If you just installed Airflow it will be set to *utc*, which is recommended. You can also set it to *system* or an IANA time zone (e.g. 'Europe/Amsterdam'). DAGs are also evaluated on Airflow workers, it is therefore important to make sure this setting is equal on all Airflow nodes.

```
[core]
default_timezone = utc
```

## 3.13.2 Time zone aware DAGs

Creating a time zone aware DAG is quite simple. Just make sure to supply a time zone aware *start\_date*. It is recommended to use *pendulum* for this, but *pytz* (to be installed manually) can also be used for this.

```
import pendulum

local_tz = pendulum.timezone("Europe/Amsterdam")

default_args=dict(
    start_date=datetime(2016, 1, 1, tzinfo=local_tz),
    owner='Airflow'
)

dag = DAG('my_tz_dag', default_args=default_args)
op = DummyOperator(task_id='dummy', dag=dag)
print(dag.timezone) # <Timezone [Europe/Amsterdam]>
```

Please note that while it is possible to set a *start\_date* and *end\_date* for Tasks always the DAG timezone or global timezone (in that order) will be used to calculate the next execution date. Upon first encounter the start date or end date will be converted to UTC using the timezone associated with *start\_date* or *end\_date*, then for calculations this timezone information will be disregarded.

### 3.13.2.1 Templates

Airflow returns time zone aware datetimes in templates, but does not convert them to local time so they remain in UTC. It is left up to the DAG to handle this.

```
import pendulum

local_tz = pendulum.timezone("Europe/Amsterdam")
local_tz.convert(execution_date)
```

### 3.13.2.2 Cron schedules

In case you set a cron schedule, Airflow assumes you will always want to run at the exact same time. It will then ignore day light savings time. Thus, if you have a schedule that says run at the end of interval every day at 08:00 GMT+1 it will always run at the end of interval 08:00 GMT+1, regardless if day light savings time is in place.

### 3.13.2.3 Time deltas

For schedules with time deltas Airflow assumes you always will want to run with the specified interval. So if you specify a `timedelta(hours=2)` you will always want to run two hours later. In this case day light savings time will be taken into account.

## 3.14 Experimental Rest API

Airflow exposes an experimental Rest API. It is available through the webserver. Endpoints are available at `/api/experimental/`. Please note that we expect the endpoint definitions to change.

### 3.14.1 Endpoints

**POST** `/api/experimental/dags/<DAG_ID>/dag_runs`

Creates a `dag_run` for a given dag id.

Trigger DAG with config, example:

```
curl -X POST \
  http://localhost:8080/api/experimental/dags/<DAG_ID>/dag_runs \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{"conf":{"key":"value"}}'
```

**GET** `/api/experimental/dags/<DAG_ID>/dag_runs`

Returns a list of Dag Runs for a specific DAG ID.

**GET** `/api/experimental/dags/<string:dag_id>/dag_runs/<string:execution_date>`

Returns a JSON with a `dag_run`'s public instance variables. The format for the `<string:execution_date>` is expected to be "YYYY-mm-DDTHH:MM:SS", for example: "2016-11-16T11:34:15".

**GET** `/api/experimental/test`

To check REST API server correct work. Return status 'OK'.

**GET** `/api/experimental/dags/<DAG_ID>/tasks/<TASK_ID>`

Returns info for a task.

**GET** `/api/experimental/dags/<DAG_ID>/dag_runs/<string:execution_date>/tasks/<TASK_ID>`

Returns a JSON with a task instance's public instance variables. The format for the `<string:execution_date>` is expected to be "YYYY-mm-DDTHH:MM:SS", for example: "2016-11-16T11:34:15".

**GET /api/experimental/dags/<DAG\_ID>/paused/<string:paused>**  
 ‘<string:paused>’ must be a ‘true’ to pause a DAG and ‘false’ to unpause.

**GET /api/experimental/latest\_runs**  
 Returns the latest DagRun for each DAG formatted for the UI.

**GET /api/experimental/pools**  
 Get all pools.

**GET /api/experimental/pools/<string:name>**  
 Get pool by a given name.

**POST /api/experimental/pools**  
 Create a pool.

**DELETE /api/experimental/pools/<string:name>**  
 Delete pool.

### 3.14.2 CLI

For some functions the cli can use the API. To configure the CLI to use the API when available configure as follows:

```
[cli]
api_client = airflow.api.client.json_client
endpoint_url = http://<WEBSERVER>:<PORT>
```

### 3.14.3 Authentication

Authentication for the API is handled separately to the Web Authentication. The default is to not require any authentication on the API – i.e. wide open by default. This is not recommended if your Airflow webserver is publicly accessible, and you should probably use the deny all backend:

```
[api]
auth_backend = airflow.api.auth.backend.deny_all
```

Two “real” methods for authentication are currently supported for the API.

To enable Password authentication, set the following in the configuration:

```
[api]
auth_backend = airflow.contrib.auth.backends.password_auth
```

Its usage is similar to the Password Authentication used for the Web interface.

To enable Kerberos authentication, set the following in the configuration:

```
[api]
auth_backend = airflow.api.auth.backend.kerberos_auth

[kerberos]
keytab = <KEYTAB>
```

The Kerberos service is configured as `airflow/fully.qualified.domainname@REALM`. Make sure this principal exists in the keytab file.

## 3.15 Integration

- *Azure: Microsoft Azure*
- *AWS: Amazon Web Services*
- *Databricks*
- *GCP: Google Cloud Platform*
- *Qubole*

### 3.15.1 Azure: Microsoft Azure

Airflow has limited support for Microsoft Azure: interfaces exist only for Azure Blob Storage and Azure Data Lake. Hook, Sensor and Operator for Blob Storage and Azure Data Lake Hook are in contrib section.

#### 3.15.1.1 Azure Blob Storage

All classes communicate via the Windows Azure Storage Blob protocol. Make sure that a Airflow connection of type *wasb* exists. Authorization can be done by supplying a login (=Storage account name) and password (=KEY), or login and SAS token in the extra field (see connection *wasb\_default* for an example).

**`airflow.contrib.hooks.wasb_hook.WasbHook`** Interface with Azure Blob Storage.

**`airflow.contrib.sensors.wasb_sensor.WasbBlobSensor`** Checks if a blob is present on Azure Blob storage.

**`airflow.contrib.operators.wasb_delete_blob_operator.WasbDeleteBlobOperator`** Deletes blob(s) on Azure Blob Storage.

**`airflow.contrib.sensors.wasb_sensor.WasbPrefixSensor`** Checks if blobs matching a prefix are present on Azure Blob storage.

**`airflow.contrib.operators.file_to_wasb.FileToWasbOperator`** Uploads a local file to a container as a blob.

#### 3.15.1.2 Azure File Share

Cloud variant of a SMB file share. Make sure that a Airflow connection of type *wasb* exists. Authorization can be done by supplying a login (=Storage account name) and password (=Storage account key), or login and SAS token in the extra field (see connection *wasb\_default* for an example).

**`airflow.contrib.hooks.azure_fileshare_hook.AzureFileShareHook`**: Interface with Azure File Share.

#### 3.15.1.3 Logging

Airflow can be configured to read and write task logs in Azure Blob Storage. See *Writing Logs to Azure Blob Storage*.



### 3.15.1.4 Azure CosmosDB

AzureCosmosDBHook communicates via the Azure Cosmos library. Make sure that a Airflow connection of type `azure_cosmos` exists. Authorization can be done by supplying a login (=Endpoint uri), password (=secret key) and extra fields `database_name` and `collection_name` to specify the default database and collection to use (see connection `azure_cosmos_default` for an example).

**`airflow.contrib.hooks.azure_cosmos_hook.AzureCosmosDBHook`** Interface with Azure CosmosDB.

**`airflow.contrib.operators.azure_cosmos_operator.AzureCosmosInsertDocumentOperator`** Simple operator to insert document into CosmosDB.

**`airflow.contrib.sensors.azure_cosmos_sensor.AzureCosmosDocumentSensor`** Simple sensor to detect document existence in CosmosDB.

### 3.15.1.5 Azure Data Lake

AzureDataLakeHook communicates via a REST API compatible with WebHDFS. Make sure that a Airflow connection of type `azure_data_lake` exists. Authorization can be done by supplying a login (=Client ID), password (=Client Secret) and extra fields `tenant` (Tenant) and `account_name` (Account Name) (see connection `azure_data_lake_default` for an example).

**`airflow.contrib.hooks.azure_data_lake_hook.AzureDataLakeHook`** Interface with Azure Data Lake.

**`airflow.contrib.operators.adls_list_operator.AzureDataLakeStorageListOperator`** Lists the files located in a specified Azure Data Lake path.

**`airflow.contrib.operators.adls_to_gcs.AdlsToGoogleCloudStorageOperator`** Copies files from an Azure Data Lake path to a Google Cloud Storage bucket.

### 3.15.1.6 Azure Container Instances

Azure Container Instances provides a method to run a docker container without having to worry about managing infrastructure. The AzureContainerInstanceHook requires a service principal. The credentials for this principal can either be defined in the extra field `key_path`, as an environment variable named `AZURE_AUTH_LOCATION`, or by providing a login/password and `tenantId` in extras.

The AzureContainerRegistryHook requires a host/login/password to be defined in the connection.

**`airflow.contrib.hooks.azure_container_volume_hook.AzureContainerVolumeHook`** Interface with Azure Container Volumes

**`airflow.contrib.operators.azure_container_instances_operator.AzureContainerInstancesOperator`** Start/Monitor a new ACI.

**`airflow.contrib.hooks.azure_container_instance_hook.AzureContainerInstanceHook`** Wrapper around a single ACI.

**`airflow.contrib.hooks.azure_container_registry_hook.AzureContainerRegistryHook`** Interface with ACR

## 3.15.2 AWS: Amazon Web Services

Airflow has extensive support for Amazon Web Services. But note that the Hooks, Sensors and Operators are in the contrib section.

### 3.15.2.1 AWS EMR

*airflow.contrib.hooks.emr\_hook.EmrHook* Interface with AWS EMR.

*airflow.contrib.operators.emr\_add\_steps\_operator.EmrAddStepsOperator* Adds steps to an existing EMR JobFlow.

*airflow.contrib.operators.emr\_create\_job\_flow\_operator.EmrCreateJobFlowOperator* Creates an EMR JobFlow, reading the config from the EMR connection.

*airflow.contrib.operators.emr\_terminate\_job\_flow\_operator.EmrTerminateJobFlowOperator* Terminates an EMR JobFlow.

### 3.15.2.2 AWS S3

*airflow.hooks.S3\_hook.S3Hook* Interface with AWS S3.

*airflow.operators.s3\_file\_transform\_operator.S3FileTransformOperator* Copies data from a source S3 location to a temporary location on the local filesystem.

*airflow.contrib.operators.s3\_list\_operator.S3ListOperator* Lists the files matching a key prefix from a S3 location.

*airflow.contrib.operators.s3\_to\_gcs\_operator.S3ToGoogleCloudStorageOperator* Syncs an S3 location with a Google Cloud Storage bucket.

*airflow.contrib.operators.s3\_to\_gcs\_transfer\_operator.S3ToGoogleCloudStorageTransferOperator* Syncs an S3 bucket with a Google Cloud Storage bucket using the GCP Storage Transfer Service.

*airflow.operators.s3\_to\_hive\_operator.S3ToHiveTransfer* Moves data from S3 to Hive. The operator downloads a file from S3, stores the file locally before loading it into a Hive table.

### 3.15.2.3 AWS Batch Service

*airflow.contrib.operators.awsbatch\_operator.AWSBatchOperator* Execute a task on AWS Batch Service.

### 3.15.2.4 AWS RedShift

*airflow.contrib.sensors.aws\_redshift\_cluster\_sensor.AwsRedshiftClusterSensor* Waits for a Redshift cluster to reach a specific status.

*airflow.contrib.hooks.redshift\_hook.RedshiftHook* Interact with AWS Redshift, using the boto3 library.

*airflow.operators.redshift\_to\_s3\_operator.RedshiftToS3Transfer* Executes an unload command to S3 as CSV with or without headers.

*airflow.operators.s3\_to\_redshift\_operator.S3ToRedshiftTransfer* Executes an copy command from S3 as CSV with or without headers.

### 3.15.2.5 AWS DynamoDB

*airflow.contrib.operators.hive\_to\_dynamodb.HiveToDynamoDBTransferOperator* Moves data from Hive to DynamoDB.

*airflow.contrib.hooks.aws\_dynamodb\_hook.AwsDynamoDBHook* Interface with AWS DynamoDB.

### 3.15.2.6 AWS Lambda

`airflow.contrib.hooks.aws_lambda_hook.AwsLambdaHook` Interface with AWS Lambda.

### 3.15.2.7 AWS Kinesis

`airflow.contrib.hooks.aws_firehose_hook.AwsFirehoseHook` Interface with AWS Kinesis Firehose.

### 3.15.2.8 Amazon SageMaker

For more instructions on using Amazon SageMaker in Airflow, please see [the SageMaker Python SDK README](#).

`airflow.contrib.hooks.sagemaker_hook.SageMakerHook` Interface with Amazon SageMaker.

`airflow.contrib.operators.sagemaker_training_operator.SageMakerTrainingOperator`  
Create a SageMaker training job.

`airflow.contrib.operators.sagemaker_tuning_operator.SageMakerTuningOperator`  
Create a SageMaker tuning job.

`airflow.contrib.operators.sagemaker_model_operator.SageMakerModelOperator` Create a SageMaker model.

`airflow.contrib.operators.sagemaker_transform_operator.SageMakerTransformOperator`  
Create a SageMaker transform job.

`airflow.contrib.operators.sagemaker_endpoint_config_operator.SageMakerEndpointConfigOperator`  
Create a SageMaker endpoint config.

`airflow.contrib.operators.sagemaker_endpoint_operator.SageMakerEndpointOperator`  
Create a SageMaker endpoint.

## 3.15.3 Databricks

[Databricks](#) has contributed an Airflow operator which enables submitting runs to the Databricks platform. Internally the operator talks to the `api/2.0/jobs/runs/submit` endpoint.

`airflow.contrib.operators.databricks_operator.DatabricksSubmitRunOperator`  
Submits a Spark job run to Databricks using the `api/2.0/jobs/runs/submit` API endpoint.

## 3.15.4 GCP: Google Cloud Platform

Airflow has extensive support for the Google Cloud Platform. But note that most Hooks and Operators are in the contrib section. Meaning that they have a *beta* status, meaning that they can have breaking changes between minor releases.

See the [GCP connection type](#) documentation to configure connections to GCP.

### 3.15.4.1 Logging

Airflow can be configured to read and write task logs in Google Cloud Storage. See [Writing Logs to Google Cloud Storage](#).

### 3.15.4.2 GoogleCloudBaseHook

All hooks is based on `airflow.contrib.hooks.gcp_api_base_hook.GoogleCloudBaseHook`.

### 3.15.4.3 BigQuery

`airflow.contrib.operators.bigquery_check_operator.BigQueryCheckOperator` Performs checks against a SQL query that will return a single row with different values.

`airflow.contrib.operators.bigquery_check_operator.BigQueryIntervalCheckOperator` Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from `days_back` before.

`airflow.contrib.operators.bigquery_check_operator.BigQueryValueCheckOperator` Performs a simple value check using SQL code.

`airflow.contrib.operators.bigquery_get_data.BigQueryGetDataOperator` Fetches the data from a BigQuery table and returns data in a python list

`airflow.contrib.operators.bigquery_operator.BigQueryCreateEmptyDatasetOperator` Creates an empty BigQuery dataset.

`airflow.contrib.operators.bigquery_operator.BigQueryCreateEmptyTableOperator` Creates a new, empty table in the specified BigQuery dataset optionally with schema.

`airflow.contrib.operators.bigquery_operator.BigQueryCreateExternalTableOperator` Creates a new, external table in the dataset with the data in Google Cloud Storage.

`airflow.contrib.operators.bigquery_operator.BigQueryDeleteDatasetOperator` Deletes an existing BigQuery dataset.

`airflow.contrib.operators.bigquery_operator.BigQueryOperator` Executes BigQuery SQL queries in a specific BigQuery database.

`airflow.contrib.operators.bigquery_table_delete_operator.BigQueryTableDeleteOperator` Deletes an existing BigQuery table.

`airflow.contrib.operators.bigquery_to_bigquery.BigQueryToBigQueryOperator` Copy a BigQuery table to another BigQuery table.

`airflow.contrib.operators.bigquery_to_gcs.BigQueryToCloudStorageOperator` Transfers a BigQuery table to a Google Cloud Storage bucket

They also use `airflow.contrib.hooks.bigquery_hook.BigQueryHook` to communicate with Google Cloud Platform.

### 3.15.4.4 Cloud Spanner

`airflow.contrib.operators.gcp_spanner_operator.CloudSpannerInstanceDatabaseDeleteOperator` deletes an existing database from a Google Cloud Spanner instance or returns success if the database is missing.

`airflow.contrib.operators.gcp_spanner_operator.CloudSpannerInstanceDatabaseDeployOperator` creates a new database in a Google Cloud instance or returns success if the database already exists.

`airflow.contrib.operators.gcp_spanner_operator.CloudSpannerInstanceDatabaseQueryOperator` executes an arbitrary DML query (INSERT, UPDATE, DELETE).

`airflow.contrib.operators.gcp_spanner_operator.CloudSpannerInstanceDatabaseUpdateOperator` updates the structure of a Google Cloud Spanner database.

*airflow.contrib.operators.gcp\_spanner\_operator.CloudSpannerInstanceDeleteOperator* deletes a Google Cloud Spanner instance.

*airflow.contrib.operators.gcp\_spanner\_operator.CloudSpannerInstanceDeployOperator* creates a new Google Cloud Spanner instance, or if an instance with the same name exists, updates the instance.

They also use *airflow.contrib.hooks.gcp\_spanner\_hook.CloudSpannerHook* to communicate with Google Cloud Platform.

### 3.15.4.5 Cloud SQL

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceCreateOperator* create a new Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceDatabaseCreateOperator* creates a new database inside a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceDatabaseDeleteOperator* deletes a database from a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceDatabasePatchOperator* updates a database inside a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceDeleteOperator* delete a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceExportOperator* exports data from a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstanceImportOperator* imports data into a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlInstancePatchOperator* patch a Cloud SQL instance.

*airflow.contrib.operators.gcp\_sql\_operator.CloudSqlQueryOperator* run query in a Cloud SQL instance.

They also use *airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* and *airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlHook* to communicate with Google Cloud Platform.

### 3.15.4.6 Cloud Bigtable

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableClusterUpdateOperator* updates the number of nodes in a Google Cloud Bigtable cluster.

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableInstanceCreateOperator* creates a Cloud Bigtable instance.

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableInstanceDeleteOperator* deletes a Google Cloud Bigtable instance.

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableTableCreateOperator* creates a table in a Google Cloud Bigtable instance.

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableTableDeleteOperator* deletes a table in a Google Cloud Bigtable instance.

*airflow.contrib.operators.gcp\_bigtable\_operator.BigtableTableWaitForReplicationSensor* (sensor) waits for a table to be fully replicated.

They also use `airflow.contrib.hooks.gcp_bigtable_hook.BigtableHook` to communicate with Google Cloud Platform.

#### 3.15.4.7 Compute Engine

`airflow.contrib.operators.gcp_compute_operator.GceInstanceStartOperator` start an existing Google Compute Engine instance.

`airflow.contrib.operators.gcp_compute_operator.GceInstanceStopOperator` stop an existing Google Compute Engine instance.

`airflow.contrib.operators.gcp_compute_operator.GceSetMachineTypeOperator` change the machine type for a stopped instance.

`airflow.contrib.operators.gcp_compute_operator.GceInstanceTemplateCopyOperator` copy the Instance Template, applying specified changes.

`airflow.contrib.operators.gcp_compute_operator.GceInstanceGroupManagerUpdateTemplateOperator` patch the Instance Group Manager, replacing source Instance Template URL with the destination one.

The operators have the common base operator `airflow.contrib.operators.gcp_compute_operator.GceBaseOperator`

They also use `airflow.contrib.hooks.gcp_compute_hook.GceHook` to communicate with Google Cloud Platform.

#### 3.15.4.8 Cloud Functions

`airflow.contrib.operators.gcp_function_operator.GcfFunctionDeployOperator` deploy Google Cloud Function to Google Cloud Platform

`airflow.contrib.operators.gcp_function_operator.GcfFunctionDeleteOperator` delete Google Cloud Function in Google Cloud Platform

They also use `airflow.contrib.hooks.gcp_function_hook.GcfHook` to communicate with Google Cloud Platform.

#### 3.15.4.9 Cloud DataFlow

`airflow.contrib.operators.dataflow_operator.DataFlowJavaOperator` launching Cloud Dataflow jobs written in Java.

`airflow.contrib.operators.dataflow_operator.DataflowTemplateOperator` launching a templated Cloud DataFlow batch job.

`airflow.contrib.operators.dataflow_operator.DataFlowPythonOperator` launching Cloud Dataflow jobs written in python.

They also use `airflow.contrib.hooks.gcp_dataflow_hook.DataFlowHook` to communicate with Google Cloud Platform.

#### 3.15.4.10 Cloud DataProc

`airflow.contrib.operators.dataproc_operator.DataprocClusterCreateOperator` Create a new cluster on Google Cloud Dataproc.

`airflow.contrib.operators.dataproc_operator.DataprocClusterDeleteOperator` Delete a cluster on Google Cloud Dataproc.

`airflow.contrib.operators.dataproc_operator.DataprocClusterScaleOperator` Scale up or down a cluster on Google Cloud Dataproc.

`airflow.contrib.operators.dataproc_operator.DataProcHadoopOperator` Start a Hadoop Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataProcHiveOperator` Start a Hive query Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataProcPigOperator` Start a Pig query Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataProcPySparkOperator` Start a PySpark Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataProcSparkOperator` Start a Spark Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataProcSparkSqlOperator` Start a Spark SQL query Job on a Cloud DataProc cluster.

`airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateInstantiateInlineOperator` Instantiate a WorkflowTemplate Inline on Google Cloud Dataproc.

`airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateInstantiateOperator` Instantiate a WorkflowTemplate on Google Cloud Dataproc.

#### 3.15.4.11 Cloud Datastore

`airflow.contrib.operators.datastore_export_operator.DatastoreExportOperator` Export entities from Google Cloud Datastore to Cloud Storage.

`airflow.contrib.operators.datastore_import_operator.DatastoreImportOperator` Import entities from Cloud Storage to Google Cloud Datastore.

They also use `airflow.contrib.hooks.datastore_hook.DatastoreHook` to communicate with Google Cloud Platform.

#### 3.15.4.12 Cloud ML Engine

`airflow.contrib.operators.mlengine_operator.MLEngineBatchPredictionOperator` Start a Cloud ML Engine batch prediction job.

`airflow.contrib.operators.mlengine_operator.MLEngineModelOperator` Manages a Cloud ML Engine model.

`airflow.contrib.operators.mlengine_operator.MLEngineTrainingOperator` Start a Cloud ML Engine training job.

`airflow.contrib.operators.mlengine_operator.MLEngineVersionOperator` Manages a Cloud ML Engine model version.

They also use `airflow.contrib.hooks.gcp_mlengine_hook.MLEngineHook` to communicate with Google Cloud Platform.



### 3.15.4.13 Cloud Storage

*airflow.contrib.operators.file\_to\_gcs.FileToGoogleCloudStorageOperator* Uploads a file to Google Cloud Storage.

*airflow.contrib.operators.gcs\_acl\_operator.GoogleCloudStorageBucketCreateAclEntryOperator* Creates a new ACL entry on the specified bucket.

*airflow.contrib.operators.gcs\_acl\_operator.GoogleCloudStorageObjectCreateAclEntryOperator* Creates a new ACL entry on the specified object.

*airflow.contrib.operators.gcs\_download\_operator.GoogleCloudStorageDownloadOperator* Downloads a file from Google Cloud Storage.

*airflow.contrib.operators.gcs\_list\_operator.GoogleCloudStorageListOperator* List all objects from the bucket with the give string prefix and delimiter in name.

*airflow.contrib.operators.gcs\_operator.GoogleCloudStorageCreateBucketOperator* Creates a new cloud storage bucket.

*airflow.contrib.operators.gcs\_to\_bq.GoogleCloudStorageToBigQueryOperator* Loads files from Google cloud storage into BigQuery.

*airflow.contrib.operators.gcs\_to\_gcs.GoogleCloudStorageToGoogleCloudStorageOperator* Copies objects from a bucket to another, with renaming if requested.

*airflow.contrib.operators.mysql\_to\_gcs.MySqlToGoogleCloudStorageOperator* Copy data from any MySQL Database to Google cloud storage in JSON format.

They also use *airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* to communicate with Google Cloud Platform.

### 3.15.4.14 Transfer Service

*airflow.contrib.operators.gcs\_to\_gcs\_transfer\_operator.GoogleCloudStorageToGoogleCloudStorageTransferOperator* Copies objects from a bucket to another using Google Transfer service.

They also use *airflow.contrib.hooks.gcp\_transfer\_hook.GCPTransferServiceHook* to communicate with Google Cloud Platform.

### 3.15.4.15 Google Kubernetes Engine

*airflow.contrib.operators.gcp\_container\_operator.GKEClusterCreateOperator* Creates a Kubernetes Cluster in Google Cloud Platform

*airflow.contrib.operators.gcp\_container\_operator.GKEClusterDeleteOperator* Deletes a Kubernetes Cluster in Google Cloud Platform

*airflow.contrib.operators.gcp\_container\_operator.GKEPodOperator* Executes a task in a Kubernetes pod in the specified Google Kubernetes Engine cluster

They also use *airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook* to communicate with Google Cloud Platform.

## 3.15.5 Qubole

Apache Airflow has a native operator and hooks to talk to [Qubole](#), which lets you submit your big data jobs directly to Qubole from Apache Airflow.



*`airflow.contrib.operators.qubole_operator.QuboleOperator`* Execute tasks (commands) on QDS (<https://qubole.com>).

*`airflow.contrib.sensors.qubole_sensor.QubolePartitionSensor`* Wait for a Hive partition to show up in QHS (Qubole Hive Service) and check for its presence via QDS APIs

*`airflow.contrib.sensors.qubole_sensor.QuboleFileSensor`* Wait for a file or folder to be present in cloud storage and check for its presence via QDS APIs

*`airflow.contrib.operators.qubole_check_operator.QuboleCheckOperator`* Performs checks against Qubole Commands. `QuboleCheckOperator` expects a command that will be executed on QDS.

*`airflow.contrib.operators.qubole_check_operator.QuboleValueCheckOperator`* Performs a simple value check using Qubole command. By default, each value on the first row of this Qubole command is compared with a pre-defined value

## 3.16 Metrics

### 3.16.1 Configuration

Airflow can be set up to send metrics to `StatsD`:

```
[scheduler]
statsd_on = True
statsd_host = localhost
statsd_port = 8125
statsd_prefix = airflow
```

### 3.16.2 Counters

Name	Description
<job_name>_start	Number of started <job_name> job, ex. SchedulerJob, LocalTaskJob
<job_name>_end	Number of ended <job_name> job, ex. SchedulerJob, LocalTaskJob
operator_failures_<operator_name>	Operator <operator_name> failures
operator_successes_<operator_name>	Operator <operator_name> successes
ti_failures	Overall task instances failures
ti_successes	Overall task instances successes
zombies_killed	Zombie tasks killed
scheduler_heartbeat	Scheduler heartbeats

### 3.16.3 Gauges

Name	Description
collect_dags	Seconds taken to scan and import DAGs
dagbag_import_errors	DAG import errors
dagbag_size	DAG bag size

### 3.16.4 Timers

Name	Description
dagrun.dependency-check.<dag_id>	Seconds taken to check DAG dependencies

## 3.17 Kubernetes

### 3.17.1 Kubernetes Executor

The kubernetes executor is introduced in Apache Airflow 1.10.0. The Kubernetes executor will create a new pod for every task instance.

Example helm charts are available at *scripts/ci/kubernetes/kube/{airflow,volumes,postgres}.yaml* in the source distribution. The volumes are optional and depend on your configuration. There are two volumes available:

- Dags: by storing all the dags onto the persistent disks, all the workers can read the dags from there. Another option is using git-sync, before starting the container, a git pull of the dags repository will be performed and used throughout the lifecycle of the pod.
- Logs: by storing the logs onto a persistent disk, all the logs will be available for all the workers and the webserver itself. If you don't configure this, the logs will be lost after the worker pods shuts down. Another option is to use S3/GCS/etc to store the logs.

### 3.17.2 Kubernetes Operator

```
from airflow.contrib.operators import KubernetesOperator
from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator
from airflow.contrib.kubernetes.secret import Secret
from airflow.contrib.kubernetes.volume import Volume
from airflow.contrib.kubernetes.volume_mount import VolumeMount

secret_file = Secret('volume', '/etc/sql_conn', 'airflow-secrets', 'sql_alchemy_conn')
secret_env = Secret('env', 'SQL_CONN', 'airflow-secrets', 'sql_alchemy_conn')
volume_mount = VolumeMount('test-volume',
                            mount_path='/root/mount_file',
                            sub_path=None,
                            read_only=True)

volume_config = {
    'persistentVolumeClaim':
        {
            'claimName': 'test-volume'
        }
}
volume = Volume(name='test-volume', configs=volume_config)

affinity = {
    'nodeAffinity': {
        'preferredDuringSchedulingIgnoredDuringExecution': [
            {
                "weight": 1,
                "preference": {
```

(continues on next page)

(continued from previous page)

```

        "matchExpressions": {
            "key": "disktype",
            "operator": "In",
            "values": ["ssd"]
        }
    }
}
],
},
"podAffinity": {
    "requiredDuringSchedulingIgnoredDuringExecution": [
        {
            "labelSelector": {
                "matchExpressions": [
                    {
                        "key": "security",
                        "operator": "In",
                        "values": ["S1"]
                    }
                ]
            },
            "topologyKey": "failure-domain.beta.kubernetes.io/zone"
        }
    ]
},
"podAntiAffinity": {
    "requiredDuringSchedulingIgnoredDuringExecution": [
        {
            "labelSelector": {
                "matchExpressions": [
                    {
                        "key": "security",
                        "operator": "In",
                        "values": ["S2"]
                    }
                ]
            },
            "topologyKey": "kubernetes.io/hostname"
        }
    ]
}
}

tolerations = [
    {
        'key': "key",
        'operator': 'Equal',
        'value': 'value'
    }
]

k = KubernetesPodOperator(namespace='default',
                           image="ubuntu:16.04",
                           cmds=["bash", "-cx"],
                           arguments=["echo", "10"],
                           labels={"foo": "bar"},
                           secrets=[secret_file, secret_env])

```

(continues on next page)

(continued from previous page)

```

        volumes=[volume],
        volume_mounts=[volume_mount]
        name="test",
        task_id="task",
        affinity=affinity,
        is_delete_operator_pod=True,
        hostnetwork=False,
        tolerations=tolerations
    )

```

**class** `airflow.contrib.operators.kubernetes_pod_operator.KubernetesPodOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Execute a task in a Kubernetes Pod

### Parameters

- **image** (*str*) – Docker image you wish to launch. Defaults to dockerhub.io, but fully qualified URLs will point to custom repositories
- **namespace** (*str*) – the namespace to run within kubernetes
- **cmds** (*list[str]*) – entrypoint of the container. (templated) The docker images's entrypoint is used if this is not provide.
- **arguments** (*list[str]*) – arguments of the entrypoint. (templated) The docker image's CMD is used if this is not provided.
- **image\_pull\_policy** (*str*) – Specify a policy to cache or always pull an image
- **image\_pull\_secrets** (*str*) – Any image pull secrets to be given to the pod. If more than one secret is required, provide a comma separated list: secret\_a,secret\_b
- **volume\_mounts** (*list[airflow.contrib.kubernetes.volume\_mount.VolumeMount]*) – volumeMounts for launched pod
- **volumes** (*list[airflow.contrib.kubernetes.volume.Volume]*) – volumes for launched pod. Includes ConfigMaps and PersistentVolumes
- **labels** (*dict*) – labels to apply to the Pod
- **startup\_timeout\_seconds** (*int*) – timeout in seconds to startup the pod
- **name** (*str*) – name of the task you want to run, will be used to generate a pod id
- **env\_vars** (*dict*) – Environment variables initialized in the container. (templated)
- **secrets** (*list[airflow.contrib.kubernetes.secret.Secret]*) – Kubernetes secrets to inject in the container, They can be exposed as environment vars or files in a volume.
- **in\_cluster** (*bool*) – run kubernetes client with in\_cluster configuration
- **cluster\_context** (*str*) – context that points to kubernetes cluster. Ignored when in\_cluster is True. If None, current-context is used.
- **get\_logs** (*bool*) – get the stdout of the container as logs of the tasks
- **affinity** (*dict*) – A dict containing a group of affinity scheduling rules
- **node\_selectors** (*dict*) – A dict containing a group of scheduling rules
- **config\_file** (*str*) – The path to the Kubernetes config file

- **xcom\_push** (*bool*) – If xcom\_push is True, the content of the file /airflow/xcom/return.json in the container will also be pushed to an XCom when the container completes.
- **hostnetwork** (*bool*) – If True enable host networking on the pod
- **tolerations** (*list tolerations*) – A list of kubernetes tolerations

**class** airflow.contrib.kubernetes.secret.**Secret** (*deploy\_type, deploy\_target, secret, key*)  
 Defines Kubernetes Secret Volume

## 3.18 Lineage

**Note:** Lineage support is very experimental and subject to change.

Airflow can help track origins of data, what happens to it and where it moves over time. This can aid having audit trails and data governance, but also debugging of data flows.

Airflow tracks data by means of inlets and outlets of the tasks. Let's work from an example and see how it works.

```
from airflow.operators.bash_operator import BashOperator
from airflow.operators.dummy_operator import DummyOperator
from airflow.lineage.datasets import File
from airflow.models import DAG
from datetime import timedelta

FILE_CATEGORIES = ["CAT1", "CAT2", "CAT3"]

args = {
    'owner': 'airflow',
    'start_date': airflow.utils.dates.days_ago(2)
}

dag = DAG(
    dag_id='example_lineage', default_args=args,
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60))

f_final = File("/tmp/final")
run_this_last = DummyOperator(task_id='run_this_last', dag=dag,
    inlets={"auto": True},
    outlets={"datasets": [f_final,]})

f_in = File("/tmp/whole_directory/")
outlets = []
for file in FILE_CATEGORIES:
    f_out = File("/tmp/{}/{{{ execution_date }}}".format(file))
    outlets.append(f_out)
run_this = BashOperator(
    task_id='run_me_first', bash_command='echo 1', dag=dag,
    inlets={"datasets": [f_in,]},
    outlets={"datasets": outlets}
)
run_this.set_downstream(run_this_last)
```

Tasks take the parameters *inlets* and *outlets*. Inlets can be manually defined by a list of dataset `{"datasets": [dataset1, dataset2]}` or can be configured to look for outlets from upstream tasks `{"task_ids": ["task_id1", "task_id2"]}` or can be configured to pick up outlets from direct upstream tasks `{"auto": True}` or a combination of them. Outlets are defined as list of dataset `{"datasets": [dataset1, dataset2]}`. Any fields for the dataset are templated with the context when the task is being executed.

---

**Note:** Operators can add inlets and outlets automatically if the operator supports it.

---

In the example DAG task `run_me_first` is a `BashOperator` that takes 3 inlets: `CAT1`, `CAT2`, `CAT3`, that are generated from a list. Note that `execution_date` is a templated field and will be rendered when the task is running.

---

**Note:** Behind the scenes Airflow prepares the lineage metadata as part of the `pre_execute` method of a task. When the task has finished execution `post_execute` is called and lineage metadata is pushed into XCOM. Thus if you are creating your own operators that override this method make sure to decorate your method with `prepare_lineage` and `apply_lineage` respectively.

---

### 3.18.1 Apache Atlas

Airflow can send its lineage metadata to Apache Atlas. You need to enable the *atlas* backend and configure it properly, e.g. in your *airflow.cfg*:

```
[lineage]
backend = airflow.lineage.backend.atlas

[atlas]
username = my_username
password = my_password
host = host
port = 21000
```

Please make sure to have the *atlasclient* package installed.

## 3.19 Changelog

### 3.19.1 Airflow 1.10.2, 2019-01-19

#### 3.19.1.1 New features

- [AIRFLOW-2658] Add GCP specific k8s pod operator (#3532)
- [AIRFLOW-2440] Google Cloud SQL import/export operator (#4251)
- [AIRFLOW-3212] Add AwsGlueCatalogPartitionSensor (#4112)
- [AIRFLOW-2750] Add subcommands to delete and list users
- [AIRFLOW-3480] Add GCP Spanner Database Operators (#4353)
- [AIRFLOW-3560] Add DayOfWeek Sensor (#4363)
- [AIRFLOW-3371] BigQueryHook's Ability to Create View (#4213)
- [AIRFLOW-3332] Add method to allow inserting rows into BQ table (#4179)

- [AIRFLOW-3055] add get\_dataset and get\_datasets\_list to bigquery\_hook (#3894)
- [AIRFLOW-2887] Added BigQueryCreateEmptyDatasetOperator and create\_emty\_dataset to bigquery\_hook (#3876)
- [AIRFLOW-2758] Add a sensor for MongoDB
- [AIRFLOW-2640] Add Cassandra table sensor
- [AIRFLOW-3398] Google Cloud Spanner instance database query operator (#4314)
- [AIRFLOW-3310] Google Cloud Spanner deploy / delete operators (#4286)
- [AIRFLOW-3406] Implement an Azure CosmosDB operator (#4265)
- [AIRFLOW-3434] Allows creating intermediate dirs in SFTPOperator (#4270)
- [AIRFLOW-3345] Add Google Cloud Storage (GCS) operators for ACL (#4192)
- [AIRFLOW-3266] Add AWS Athena Hook and Operator (#4111)
- [AIRFLOW-3346] Add hook and operator for GCP transfer service (#4189)
- [AIRFLOW-2983] Add prev\_ds\_nodash and next\_ds\_nodash macro (#3821)
- [AIRFLOW-3403] Add AWS Athena Sensor (#4244)
- [AIRFLOW-3323] Support HTTP basic authentication for Airflow Flower (#4166)
- [AIRFLOW-3410] Add feature to allow Host Key Change for SSH Op (#4249)
- [AIRFLOW-3275] Add Google Cloud SQL Query operator (#4170)
- [AIRFLOW-2691] Manage JS dependencies via npm
- [AIRFLOW-2795] Oracle to Oracle Transfer Operator (#3639)
- [AIRFLOW-2596] Add Oracle to Azure Datalake Transfer Operator
- [AIRFLOW-3220] Add Instance Group Manager Operators for GCE (#4167)
- [AIRFLOW-2882] Add import and export for pool cli using JSON
- [AIRFLOW-2965] CLI tool to show the next execution datetime (#3834)
- [AIRFLOW-2874] Enables FAB's theme support (#3719)
- [AIRFLOW-3336] Add new TriggerRule for 0 upstream failures (#4182)

### 3.19.1.2 Improvements

- [AIRFLOW-3680] Consistency update in tests for All GCP-related operators (#4493)
- [AIRFLOW-3675] Use googlapiclient for google apis (#4484)
- [AIRFLOW-3205] Support multipart uploads to GCS (#4084)
- [AIRFLOW-2826] Add GoogleCloudKMSHook (#3677)
- [AIRFLOW-3676] Add required permission to CloudSQL export/import example (#4489)
- [AIRFLOW-3679] Added Google Cloud Base Hook to documentation (#4487)
- [AIRFLOW-3594] Unify different License Header
- [AIRFLOW-3197] Remove invalid parameter KeepJobFlowAliveWhenNoSteps in example DAG (#4404)
- [AIRFLOW-3504] Refine the functionality of "/health" endpoint (#4309)

- [AIRFLOW-3103][AIRFLOW-3147] Update flask-appbuilder (#3937)
- [AIRFLOW-3168] More resilient database use in CI (#4014)
- [AIRFLOW-3076] Remove preloading of MySQL testdata (#3911)
- [AIRFLOW-3035] Allow custom 'job\_error\_states' in dataproc ops (#3884)
- [AIRFLOW-3246] Make hmsclient optional in airflow.hooks.hive\_hooks (#4080)
- [AIRFLOW-3059] Log how many rows are read from Postgres (#3905)
- [AIRFLOW-2463] Make task instance context available for hive queries
- [AIRFLOW-3190] Make flake8 compliant (#4035)
- [AIRFLOW-1998] Implemented DatabricksRunNowOperator for jobs/run-now ... (#3813)
- [AIRFLOW-2267] Airflow DAG level access (#3197)
- [AIRFLOW-2359] Add set failed for DagRun and task in tree view (#3255)
- [AIRFLOW-3008] Move Kubernetes example DAGs to contrib
- [AIRFLOW-3402] Support global k8s affinity and toleration configs (#4247)
- [AIRFLOW-3610] Add region param for EMR jobflow creation (#4418)
- [AIRFLOW-3531] Fix test for GCS to GCS Transfer Hook (#4452)
- [AIRFLOW-3531] Add gcs to gcs transfer operator. (#4331)
- [AIRFLOW-3034]: Readme updates : Add Slack & Twitter, remove Gitter
- [AIRFLOW-3028] Update Text & Images in Readme.md
- [AIRFLOW-208] Add badge to show supported Python versions (#3839)
- [AIRFLOW-2238] Update PR tool to push directly to Github
- [AIRFLOW-2238] Flake8 fixes on dev/airflow-pr
- [AIRFLOW-2238] Update PR tool to remove outdated info (#3978)
- [AIRFLOW-3005] Replace 'Airbnb Airflow' with 'Apache Airflow' (#3845)
- [AIRFLOW-3150] Make execution\_date templated in TriggerDagRunOperator (#4359)
- [AIRFLOW-1196][AIRFLOW-2399] Add templated field in TriggerDagRunOperator (#4228)
- [AIRFLOW-3340] Placeholder support in connections form (#4185)
- [AIRFLOW-3446] Add Google Cloud BigTable operators (#4354)
- [AIRFLOW-1921] Add support for https and user auth (#2879)
- [AIRFLOW-2770] Read *dags\_in\_image* config value as a boolean (#4319)
- [AIRFLOW-3022] Add volume mount to KubernetesExecutorConfig (#3855)
- [AIRFLOW-2917] Set AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN only when needed (#3766)
- [AIRFLOW-2712] Pass annotations to KubernetesExecutorConfig
- [AIRFLOW-461] Support autodetected schemas in BigQuery run\_load (#3880)
- [AIRFLOW-2997] Support cluster fields in bigquery (#3838)
- [AIRFLOW-2916] Arg *verify* for AwsHook() & S3 sensors/operators (#3764)
- [AIRFLOW-491] Add feature to pass extra api configs to BQ Hook (#3733)



- [AIRFLOW-2889] Fix typos detected by [github.com/client9/misspell](https://github.com/client9/misspell) (#3732)
- [AIRFLOW-850] Add a PythonSensor (#4349)
- [AIRFLOW-2747] Explicit re-schedule of sensors (#3596)
- [AIRFLOW-3392] Add index on dag\_id in sla\_miss table (#4235)
- [AIRFLOW-3001] Add index 'ti\_dag\_date' to taskinstance (#3885)
- [AIRFLOW-2861] Add index on log table (#3709)
- [AIRFLOW-3518] Performance fixes for topological\_sort of Tasks (#4322)
- [AIRFLOW-3521] Fetch more than 50 items in *airflow-jira compare* script (#4300)
- [AIRFLOW-1919] Add option to query for DAG runs given a DAG ID
- [AIRFLOW-3444] Explicitly set transfer operator description. (#4279)
- [AIRFLOW-3411] Add OpenFaaS hook (#4267)
- [AIRFLOW-2785] Add context manager entry points to mongoHook
- [AIRFLOW-2524] Add SageMaker doc to AWS integration section (#4278)
- [AIRFLOW-3479] Keeps records in Log Table when DAG is deleted (#4287)
- [AIRFLOW-2948] Arg check & better doc - SSHOperator & SFTPOperator (#3793)
- [AIRFLOW-2245] Add remote\_host of SSH/SFTP operator as templated field (#3765)
- [AIRFLOW-2670] Update SSH Operator's Hook to respect timeout (#3666)
- [AIRFLOW-3380] Add metrics documentation (#4219)
- [AIRFLOW-3361] Log the task\_id in the PendingDeprecationWarning from BaseOperator (#4030)
- [AIRFLOW-3213] Create ADLS to GCS operator (#4134)
- [AIRFLOW-3395] added the REST API endpoints to the doc (#4236)
- [AIRFLOW-3294] Update connections form and integration docs (#4129)
- [AIRFLOW-3236] Create AzureDataLakeStorageListOperator (#4094)
- [AIRFLOW-3062] Add Qubole in integration docs (#3946)
- [AIRFLOW-3306] Disable flask-sqlalchemy modification tracking. (#4146)
- [AIRFLOW-2867] Refactor Code to conform standards (#3714)
- [AIRFLOW-2753] Add dataproc\_job\_id instance var holding actual DP jobId
- [AIRFLOW-3132] Enable specifying auto\_remove option for DockerOperator (#3977)
- [AIRFLOW-2731] Raise psutil restriction to <6.0.0
- [AIRFLOW-3384] Allow higher versions of SQLAlchemy and Jinja2 (#4227)
- [Airflow-2760] Decouple DAG parsing loop from scheduler loop (#3873)
- [AIRFLOW-3004] Add config disabling scheduler cron (#3899)
- [AIRFLOW-3175] Fix docstring format in *airflow/jobs.py* (#4025)
- [AIRFLOW-3589] Visualize reschedule state in all views (#4408)
- [AIRFLOW-2698] Simplify Kerberos code (#3563)
- [AIRFLOW-2499] Dockerise CI pipeline (#3393)

- [AIRFLOW-3432] Add test for feature “Delete DAG in UI” (#4266)
- [AIRFLOW-3301] Update DockerOperator CI test for PR #3977 (#4138)
- [AIRFLOW-3478] Make sure that the session is closed
- [AIRFLOW-3687] Add missing @apply\_defaults decorators (#4498)
- [AIRFLOW-3691] Update notice to 2019 (#4503)
- [AIRFLOW-3689] Update pop-up message when deleting DAG in RBAC UI (#4505)
- [AIRFLOW-2801] Skip test\_mark\_success\_no\_kill in PostgreSQL on CI (#3642)
- [AIRFLOW-3693] Replace psycpg2-binary by psycpg2 (#4508)
- [AIRFLOW-3700] Change the lowest allowed version of “requests” (#4517)
- [AIRFLOW-3704] Support SSL Protection When Redis is Used as Broker for CeleryExecutor (#4521)
- [AIRFLOW-3681] All GCP operators have now optional GCP Project ID (#4500)
- [Airflow 2782] Upgrades Dagre D3 version to latest possible
- [Airflow 2783] Implement eslint for JS code check (#3641)
- [AIRFLOW-2805] Display multiple timezones on UI (#3687)
- [AIRFLOW-3302] Small CSS fixes (#4140)
- [Airflow-2766] Respect shared datetime across tabs
- [AIRFLOW-2776] Compress tree view JSON
- [AIRFLOW-2407] Use feature detection for reload() (#3298)
- [AIRFLOW-3452] Removed an unused/dangerous display-none (#4295)
- [AIRFLOW-3348] Update run statistics on dag refresh (#4197)
- [AIRFLOW-3125] Monitor Task Instances creation rates (#3966)

### 3.19.1.3 Bug fixes

- [AIRFLOW-3191] Fix not being able to specify execution\_date when creating dagrun (#4037)
- [AIRFLOW-3657] Fix zendesk integration (#4466)
- [AIRFLOW-3605] Load plugins from entry\_points (#4412)
- [AIRFLOW-3646] Rename plugins\_manager.py to test\_xx to trigger tests (#4464)
- [AIRFLOW-3655] Escape links generated in model views (#4463)
- [AIRFLOW-3662] Add dependency for Enum (#4468)
- [AIRFLOW-3630] Cleanup of GCP Cloud SQL Connection (#4451)
- [AIRFLOW-1837] Respect task start\_date when different from dag’s (#4010)
- [AIRFLOW-2829] Brush up the CI script for minikube
- [AIRFLOW-3519] Fix example http operator (#4455)
- [AIRFLOW-2811] Fix scheduler\_ops\_metrics.py to work (#3653)
- [AIRFLOW-2751] add job properties update in hive to druid operator.
- [AIRFLOW-2918] Remove unused imports

- [AIRFLOW-2918] Fix Flake8 violations (#3931)
- [AIRFLOW-2771] Add except type to broad S3Hook try catch clauses
- [AIRFLOW-2918] Fix Flake8 violations (#3772)
- [AIRFLOW-2099] Handle getsource() calls gracefully
- [AIRFLOW-3397] Fix integrity error in rbac AirflowSecurityManager (#4305)
- [AIRFLOW-3281] Fix Kubernetes operator with git-sync (#3770)
- [AIRFLOW-2615] Limit DAGs parsing to once only
- [AIRFLOW-2952] Fix Kubernetes CI (#3922)
- [AIRFLOW-2933] Enable Codecov on Docker-CI Build (#3780)
- [AIRFLOW-2082] Resolve a bug in adding password\_auth to api as auth method (#4343)
- [AIRFLOW-3612] Remove incubation/incubator mention (#4419)
- [AIRFLOW-3581] Fix next\_ds/prev\_ds semantics for manual runs (#4385)
- [AIRFLOW-3527] Update Cloud SQL Proxy to have shorter path for UNIX socket (#4350)
- [AIRFLOW-3316] For gcs\_to\_bq: add missing init of schema\_fields var (#4430)
- [AIRFLOW-3583] Fix AirflowException import (#4389)
- [AIRFLOW-3578] Fix Type Error for BigQueryOperator (#4384)
- [AIRFLOW-2755] Added *kubernetes.worker\_days\_folder* configuration (#3612)
- [AIRFLOW-2655] Fix inconsistency of default config of kubernetes worker
- [AIRFLOW-2645][AIRFLOW-2617] Add worker\_container\_image\_pull\_policy
- [AIRFLOW-2661] fix config dags\_volume\_subpath and logs\_volume\_subpath
- [AIRFLOW-3550] Standardize GKE hook (#4364)
- [AIRFLOW-2863] Fix GKEClusterHook catching wrong exception (#3711)
- [AIRFLOW-2939][AIRFLOW-3568] Fix TypeError in GCSToS3Op & S3ToGCSTOp (#4371)
- [AIRFLOW-3327] Add support for location in BigQueryHook (#4324)
- [AIRFLOW-3438] Fix default values in BigQuery Hook & BigQueryOperator (...)
- [AIRFLOW-3355] Fix BigQueryCursor.execute to work with Python3 (#4198)
- [AIRFLOW-3447] Add 2 options for ts\_nodash Macro (#4323)
- [AIRFLOW-1552] Airflow Filter\_by\_owner not working with password\_auth (#4276)
- [AIRFLOW-3484] Fix Over-logging in the k8s executor (#4296)
- [AIRFLOW-3309] Add MongoDB connection (#4154)
- [AIRFLOW-3414] Fix reload\_module in DagFileProcessorAgent (#4253)
- [AIRFLOW-1252] API accept JSON when invoking a trigger dag (#2334)
- [AIRFLOW-3425] Fix setting default scope in hook (#4261)
- [AIRFLOW-3416] Fixes Python 3 compatibility with CloudSqlQueryOperator (#4254)
- [AIRFLOW-3263] Ignore exception when 'run' kills already killed job (#4108)
- [AIRFLOW-3264] URL decoding when parsing URI for connection (#4109)

- [AIRFLOW-3365][AIRFLOW-3366] Allow celery\_broker\_transport\_options to be set with environment variables (#4211)
- [AIRFLOW-2642] fix wrong value git-sync initcontainer env GIT\_SYNC\_ROOT (#3519)
- [AIRFLOW-3353] Pin redis version (#4195)
- [AIRFLOW-3251] KubernetesPodOperator now uses 'image\_pull\_secrets' argument when creating Pods (#4188)
- [AIRFLOW-2705] Move class-level moto decorator to method-level
- [AIRFLOW-3233] Fix deletion of DAGs in the UI (#4069)
- [AIRFLOW-2908] Allow retries with KubernetesExecutor. (#3758)
- [AIRFLOW-1561] Fix scheduler to pick up example DAGs without other DAGs (#2635)
- [AIRFLOW-3352] Fix expose\_config not honoured on RBAC UI (#4194)
- [AIRFLOW-3592] Fix logs when task is in rescheduled state (#4492)
- [AIRFLOW-3634] Fix GCP Spanner Test (#4440)
- [AIRFLOW-XXX] Fix PythonVirtualenvOperator tests (#3968)
- [AIRFLOW-3239] Fix/refine tests for api/common/experimental/ (#4255)
- [AIRFLOW-2951] Update dag\_run table end\_date when state change (#3798)
- [AIRFLOW-2756] Fix bug in set DAG run state workflow (#3606)
- [AIRFLOW-3690] Fix bug to set state of a task for manually-triggered DAGs (#4504)
- [AIRFLOW-3319] KubernetesExecutor: Need in try\_number in labels if getting them later (#4163)
- [AIRFLOW-3724] Fix the broken refresh button on Graph View in RBAC UI
- [AIRFLOW-3732] Fix issue when trying to edit connection in RBAC UI
- [AIRFLOW-2866] Fix missing CSRF token head when using RBAC UI (#3804)
- [AIRFLOW-3259] Fix internal server error when displaying charts (#4114)
- [AIRFLOW-3271] Fix issue with persistence of RBAC Permissions modified via UI (#4118)
- [AIRFLOW-3141] Handle duration View for missing dag (#3984)
- [AIRFLOW-2766] Respect shared datetime across tabs
- [AIRFLOW-1413] Fix FTPSensor failing on error message with unexpected (#2450)
- [AIRFLOW-3378] KubernetesPodOperator does not delete on timeout failure (#4218)
- [AIRFLOW-3245] Fix list processing in resolve\_template\_files (#4086)
- [AIRFLOW-2703] Catch transient DB exceptions from scheduler's heartbeat it does not crash (#3650)
- [AIRFLOW-1298] Clear UPSTREAM\_FAILED using the clean cli (#3886)

#### **3.19.1.4 Doc-only changes**

- [AIRFLOW-XXX] GCP operators documentation clarifications (#4273)
- [AIRFLOW-XXX] Docs: Fix paths to GCS transfer operator (#4479)
- [AIRFLOW-XXX] Add missing GCP operators to Docs (#4260)
- [AIRFLOW-XXX] Fix Docstrings for Operators (#3820)

- [AIRFLOW-XXX] Fix inconsistent comment in `example_python_operator.py` (#4337)
- [AIRFLOW-XXX] Fix incorrect parameter in SFTPOperator example (#4344)
- [AIRFLOW-XXX] Add missing remote logging field (#4333)
- [AIRFLOW-XXX] Revise template variables documentation (#4172)
- [AIRFLOW-XXX] Fix typo in docstring of `gcs_to_bq` (#3833)
- [AIRFLOW-XXX] Fix display of SageMaker operators/hook docs (#4263)
- [AIRFLOW-XXX] Better instructions for airflow flower (#4214)
- [AIRFLOW-XXX] Make pip install commands consistent (#3752)
- [AIRFLOW-XXX] Add *BigQueryGetDataOperator* to Integration Docs (#4063)
- [AIRFLOW-XXX] Don't spam test logs with "bad cron expression" messages (#3973)
- [AIRFLOW-XXX] Update committer list based on latest TLP discussion (#4427)
- [AIRFLOW-XXX] Fix incorrect statement in contributing guide (#4104)
- [AIRFLOW-XXX] Fix Broken Link in CONTRIBUTING.md
- [AIRFLOW-XXX] Update Contributing Guide - Git Hooks (#4120)
- [AIRFLOW-3426] Correct Python Version Documentation Reference (#4259)
- [AIRFLOW-2663] Add instructions to install SSH dependencies
- [AIRFLOW-XXX] Clean up installation extra packages table (#3750)
- [AIRFLOW-XXX] Remove redundant space in Kerberos (#3866)
- [AIRFLOW-3086] Add extras group for google auth to `setup.py` (#3917)
- [AIRFLOW-XXX] Add Kubernetes Dependency in Extra Packages Doc (#4281)
- [AIRFLOW-3696] Add Version info to Airflow Documentation (#4512)
- [AIRFLOW-XXX] Correct Typo in sensor's exception (#4545)
- [AIRFLOW-XXX] Fix a typo of config (#4544)
- [AIRFLOW-XXX] Fix BashOperator Docstring (#4052)
- [AIRFLOW-3018] Fix Minor issues in Documentation
- [AIRFLOW-XXX] Fix Minor issues with Azure Cosmos Operator (#4289)
- [AIRFLOW-3382] Fix incorrect docstring in DatastoreHook (#4222)
- [AIRFLOW-XXX] Fix copy&paste mistake (#4212)
- [AIRFLOW-3260] Correct misleading BigQuery error (#4098)
- [AIRFLOW-XXX] Fix Typo in SFTPOperator docstring (#4016)
- [AIRFLOW-XXX] Fixing the issue in Documentation (#3998)
- [AIRFLOW-XXX] Fix undocumented params in `S3_hook`
- [AIRFLOW-XXX] Fix SlackWebhookOperator execute method comment (#3963)
- [AIRFLOW-3070] Refine web UI authentication-related docs (#3863)

## 3.19.2 Airflow 1.10.1, 2018-11-13

### 3.19.2.1 New features

- [AIRFLOW-2524] Airflow integration with AWS Sagemaker
- [AIRFLOW-2657] Add ability to delete DAG from web ui
- [AIRFLOW-2780] Adds IMAP Hook to interact with a mail server
- [AIRFLOW-2794] Add delete support for Azure blob
- [AIRFLOW-2912] Add operators for Google Cloud Functions
- [AIRFLOW-2974] Add Start/Restart/Terminate methods Databricks Hook
- [AIRFLOW-2989] No Parameter to change bootDiskType for DataprocClusterCreateOperator
- [AIRFLOW-3078] Basic operators for Google Compute Engine
- [AIRFLOW-3147] Update Flask-AppBuilder version
- [AIRFLOW-3231] Basic operators for Google Cloud SQL (deploy / patch / delete)
- [AIRFLOW-3276] Google Cloud SQL database create / patch / delete operators

### 3.19.2.2 Improvements

- [AIRFLOW-393] Add progress callbacks for FTP downloads
- [AIRFLOW-520] Show Airflow version on web page
- [AIRFLOW-843] Exceptions now available in context during on\_failure\_callback
- [AIRFLOW-2476] Update tabulate dependency to v0.8.2
- [AIRFLOW-2592] Bump Bleach dependency
- [AIRFLOW-2622] Add “confirm=False” option to SFTPOperator
- [AIRFLOW-2662] support affinity & nodeSelector policies for kubernetes executor/operator
- [AIRFLOW-2709] Improve error handling in Databricks hook
- [AIRFLOW-2723] Update lxml dependency to >= 4.0.
- [AIRFLOW-2763] No precheck mechanism in place during worker initialisation for the connection to metadata database
- [AIRFLOW-2789] Add ability to create single node cluster to DataprocClusterCreateOperator
- [AIRFLOW-2797] Add ability to create Google Dataproc cluster with custom image
- [AIRFLOW-2854] kubernetes\_pod\_operator add more configuration items
- [AIRFLOW-2855] Need to Check Validity of Cron Expression When Process DAG File/Zip File
- [AIRFLOW-2904] Clean an unnecessary line in airflow/executors/celery\_executor.py
- [AIRFLOW-2921] A trivial incorrectness in CeleryExecutor()
- [AIRFLOW-2922] Potential deal-lock bug in CeleryExecutor()
- [AIRFLOW-2932] GoogleCloudStorageHook - allow compression of file
- [AIRFLOW-2949] Syntax Highlight for Single Quote

- [AIRFLOW-2951] dag\_run end\_date Null after a dag is finished
- [AIRFLOW-2956] Kubernetes tolerations for pod operator
- [AIRFLOW-2997] Support for clustered tables in Bigquery hooks/operators
- [AIRFLOW-3006] Fix error when schedule\_interval="None"
- [AIRFLOW-3008] Move Kubernetes related example DAGs to contrib/example\_dags
- [AIRFLOW-3025] Allow to specify dns and dns-search parameters for DockerOperator
- [AIRFLOW-3067] (www\_rbac) Flask flash messages are not displayed properly (no background color)
- [AIRFLOW-3069] Decode output of S3 file transform operator
- [AIRFLOW-3072] Assign permission get\_logs\_with\_metadata to viewer role
- [AIRFLOW-3090] INFO logs are too verbose
- [AIRFLOW-3103] Update Flask-Login
- [AIRFLOW-3112] Align SFTP hook with SSH hook
- [AIRFLOW-3119] Enable loglevel on celery worker and inherit from airflow.cfg
- [AIRFLOW-3137] Make ProxyFix middleware optional
- [AIRFLOW-3173] Add \_cmd options for more password config options
- [AIRFLOW-3177] Change scheduler\_heartbeat metric from gauge to counter
- [AIRFLOW-3193] Pin docker requirement version to v3
- [AIRFLOW-3195] Druid Hook: Log ingestion spec and task id
- [AIRFLOW-3197] EMR Hook is missing some parameters to valid on the AWS API
- [AIRFLOW-3232] Make documentation for GCF Functions operator more readable
- [AIRFLOW-3262] Can't get log containing Response when using SimpleHttpOperator
- [AIRFLOW-3265] Add support for "unix\_socket" in connection extra for Mysql Hook

### 3.19.2.3 Doc-only changes

- [AIRFLOW-1441] Tutorial Inconsistencies Between Example Pipeline Definition and Recap
- [AIRFLOW-2682] Add how-to guide(s) for how to use basic operators like BashOperator and PythonOperator
- [AIRFLOW-3104] .airflowignore feature is not mentioned at all in documentation
- [AIRFLOW-3237] Refactor example DAGs
- [AIRFLOW-3187] Update airflow.gif file with a slower version
- [AIRFLOW-3159] Update Airflow documentation on GCP Logging
- [AIRFLOW-3030] Command Line docs incorrect subdir
- [AIRFLOW-2990] Docstrings for Hooks/Operators are in incorrect format
- [AIRFLOW-3127] Celery SSL Documentation is out-dated
- [AIRFLOW-2779] Add license headers to doc files
- [AIRFLOW-2779] Add project version to license

### 3.19.2.4 Bug fixes

- [AIRFLOW-839] `docker_operator.py` attempts to log status key without first checking existence
- [AIRFLOW-1104] Concurrency check in scheduler should count queued tasks as well as running
- [AIRFLOW-1163] Add support for `x-forwarded-*` headers to support access behind AWS ELB
- [AIRFLOW-1195] Cleared tasks in `SubDagOperator` do not trigger Parent `dag_runs`
- [AIRFLOW-1508] Skipped state not part of `State.task_states`
- [AIRFLOW-1762] Use `key_file` in `SSHHook.create_tunnel()`
- [AIRFLOW-1837] Differing `start_dates` on tasks not respected by scheduler.
- [AIRFLOW-1874] Support standard SQL in `Check`, `ValueCheck` and `IntervalCheck` `BigQuery` operators
- [AIRFLOW-1917] `print()` from python operators end up with extra new line
- [AIRFLOW-1970] Database cannot be initialized if an invalid fernet key is provided
- [AIRFLOW-2145] Deadlock after clearing a running task
- [AIRFLOW-2216] Cannot specify a profile for AWS Hook to load with s3 config file
- [AIRFLOW-2574] `initdb` fails when `mysql` password contains percent sign
- [AIRFLOW-2707] Error accessing log files from web UI
- [AIRFLOW-2716] Replace new Python 3.7 keywords
- [AIRFLOW-2744] RBAC app doesn't integrate plugins (blueprints etc)
- [AIRFLOW-2772] `BigQuery` hook does not allow specifying both the partition field name and table name at the same time
- [AIRFLOW-2778] Bad Import in `collect_dag` in `DagBag`
- [AIRFLOW-2786] Variables view fails to render if a variable has an empty key
- [AIRFLOW-2799] Filtering UI objects by datetime is broken
- [AIRFLOW-2800] Remove airflow/ low-hanging linting errors
- [AIRFLOW-2825] `S3ToHiveTransfer` operator may not may able to handle GZIP file with uppercase ext in S3
- [AIRFLOW-2848] `dag_id` is missing in metadata table "job" for `LocalTaskJob`
- [AIRFLOW-2860] `DruidHook`: time variable is not updated correctly when checking for timeout
- [AIRFLOW-2865] Race condition between `on_success_callback` and `LocalTaskJob`'s cleanup
- [AIRFLOW-2893] Stuck dataflow job due to `jobName` mismatch.
- [AIRFLOW-2895] Prevent scheduler from spamming heartbeats/logs
- [AIRFLOW-2900] Code not visible for Packaged DAGs
- [AIRFLOW-2905] Switch to regional dataflow job service.
- [AIRFLOW-2907] Sendgrid - Attachments - ERROR - Object of type 'bytes' is not JSON serializable
- [AIRFLOW-2938] Invalid 'extra' field in connection can raise an `AttributeError` when attempting to edit
- [AIRFLOW-2979] Deprecated Celery Option not in Options list
- [AIRFLOW-2981] `TypeError` in dataflow operators when using GCS jar or `py_file`
- [AIRFLOW-2984] Cannot convert `naive_datetime` when task has a naive `start_date/end_date`



- [AIRFLOW-2994] flatten\_results in BigQueryOperator/BigQueryHook should default to None
- [AIRFLOW-3002] ValueError in dataflow operators when using GCS jar or py\_file
- [AIRFLOW-3012] Email on sla miss is send only to first address on the list
- [AIRFLOW-3046] ECS Operator mistakenly reports success when task is killed due to EC2 host termination
- [AIRFLOW-3064] No output from *airflow test* due to default logging config
- [AIRFLOW-3072] Only admin can view logs in RBAC UI
- [AIRFLOW-3079] Improve initdb to support MSSQL Server
- [AIRFLOW-3089] Google auth doesn't work under http
- [AIRFLOW-3099] Errors raised when some blocs are missing in airflow.cfg
- [AIRFLOW-3109] Default user permission should contain 'can\_clear'
- [AIRFLOW-3111] Confusing comments and instructions for log templates in UPDATING.md and default\_airflow.cfg
- [AIRFLOW-3124] Broken webserver debug mode (RBAC)
- [AIRFLOW-3136] Scheduler Failing the Task retries run while processing Executor Events
- [AIRFLOW-3138] Migration cc1e65623dc7 creates issues with postgres
- [AIRFLOW-3161] Log Url link does not link to task instance logs in RBAC UI
- [AIRFLOW-3162] HttpHook fails to parse URL when port is specified
- [AIRFLOW-3183] Potential Bug in utils/dag\_processing/DagFileProcessorManager.max\_runs\_reached()
- [AIRFLOW-3203] Bugs in DockerOperator & Some operator test scripts were named incorrectly
- [AIRFLOW-3238] Dags, removed from the filesystem, are not deactivated on initdb
- [AIRFLOW-3268] Cannot pass SSL dictionary to mysql connection via URL
- [AIRFLOW-3277] Invalid timezone transition handling for cron schedules
- [AIRFLOW-3295] Require encryption in DaskExecutor when certificates are configured.
- [AIRFLOW-3297] EmrStepSensor marks cancelled step as successful

### 3.19.3 Airflow 1.10.0, 2018-08-03

- [AIRFLOW-2870] Use abstract TaskInstance for migration
- [AIRFLOW-2859] Implement own UtcDateTime (#3708)
- [AIRFLOW-2140] Don't require kubernetes for the SparkSubmit hook
- [AIRFLOW-2869] Remove smart quote from default config
- [AIRFLOW-2857] Fix Read the Docs env
- [AIRFLOW-2817] Force explicit choice on GPL dependency
- [AIRFLOW-2716] Replace async and await py3.7 keywords
- [AIRFLOW-2810] Fix typo in Xcom model timestamp
- [AIRFLOW-2710] Clarify fernet key value in documentation
- [AIRFLOW-2606] Fix DB schema and SQLAlchemy model

- [AIRFLOW-2646] Fix setup.py not to install snakebite on Python3
- [AIRFLOW-2604] Add index to task\_fail
- [AIRFLOW-2650] Mark SchedulerJob as succeed when hitting Ctrl-c
- [AIRFLOW-2678] Fix db schema unit test to remove checking fab models
- [AIRFLOW-2624] Fix webserver login as anonymous
- [AIRFLOW-2654] Fix incorret URL on refresh in Graph View of FAB UI
- [AIRFLOW-2668] Handle missing optional cryptography dependency
- [AIRFLOW-2681] Include last dag run of externally triggered DAGs in UI.
- [AIRFLOW-1840] Support back-compat on old celery config
- [AIRFLOW-2612][AIRFLOW-2534] Clean up Hive-related tests
- [AIRFLOW-2608] Implements/Standardize custom exceptions for experimental APIs
- [AIRFLOW-2607] Fix failing TestLocalClient
- [AIRFLOW-2638] dbapi\_hook: support REPLACE INTO
- [AIRFLOW-2542][AIRFLOW-1790] Rename AWS Batch Operator queue to job\_queue
- [AIRFLOW-2567] Extract result from the kubernetes pod as Xcom
- [AIRFLOW-XXX] Adding REA Group to readme
- [AIRFLOW-2601] Allow user to specify k8s config
- [AIRFLOW-2559] Azure Fileshare hook
- [AIRFLOW-1786] Enforce correct behavior for soft-fail sensors
- [AIRFLOW-2355] Airflow trigger tag parameters in subdag
- [AIRFLOW-2613] Fix Airflow searching .zip bug
- [AIRFLOW-2627] Add a sensor for Cassandra
- [AIRFLOW-2634][AIRFLOW-2534] Remove dependency for impyla
- [AIRFLOW-2611] Fix wrong dag volume mount path for kubernetes executor
- [AIRFLOW-2562] Add Google Kubernetes Engine Operators
- [AIRFLOW-2630] Fix classname in test\_sql\_sensor.py
- [AIRFLOW-2534] Fix bug in HiveServer2Hook
- [AIRFLOW-2586] Stop getting AIRFLOW\_HOME value from config file in bash operator
- [AIRFLOW-2605] Fix autocommit for MySqlHook
- [AIRFLOW-2539][AIRFLOW-2359] Move remaing log config to configuration file
- [AIRFLOW-1656] Tree view dags query changed
- [AIRFLOW-2617] add imagePullPolicy config for kubernetes executor
- [AIRFLOW-2429] Fix security/task/sensors/ti\_deps folders flake8 error
- [AIRFLOW-2550] Implements API endpoint to list DAG runs
- [AIRFLOW-2512][AIRFLOW-2522] Use google-auth instead of oauth2client
- [AIRFLOW-2429] Fix operators folder flake8 error

- [AIRFLOW-2585] Fix several bugs in CassandraHook and CassandraToGCSOperator
- [AIRFLOW-2597] Restore original dbapi.run() behavior
- [AIRFLOW-2590] Fix commit in DbApiHook.run() for no-autocommit DB
- [AIRFLOW-1115] fix github oauth api URL
- [AIRFLOW-2587] Add TIMESTAMP type mapping to MySqlToHiveTransfer
- [AIRFLOW-2591][AIRFLOW-2581] Set default value of autocommit to False in DbApiHook.run()
- [AIRFLOW-59] Implement bulk\_dump and bulk\_load for the Postgres hook
- [AIRFLOW-2533] Fix path to DAG's on kubernetes executor workers
- [AIRFLOW-2581] RFLOW-2581] Fix DbApiHook autocommit
- [AIRFLOW-2578] Add option to use proxies in JiraHook
- [AIRFLOW-2575] Make gcs to gcs operator work with large files
- [AIRFLOW-437] Send TI context in kill zombies
- [AIRFLOW-2566] Change backfill to rerun failed tasks
- [AIRFLOW-1021] Fix double login for new users with LDAP
- [AIRFLOW-XXX] Typo fix
- [AIRFLOW-2561] Fix typo in EmailOperator
- [AIRFLOW-2573] Cast BigQuery TIMESTAMP field to float
- [AIRFLOW-2560] Adding support for internalIpOnly to DataprocClusterCreateOperator
- [AIRFLOW-2565] templatize cluster\_label
- [AIRFLOW-83] add mongo hook and operator
- [AIRFLOW-2558] Clear task/dag is clearing all executions
- [AIRFLOW-XXX] Fix doc typos
- [AIRFLOW-2513] Change *bql* to *sql* for BigQuery Hooks & Ops
- [AIRFLOW-2557] Fix pagination for s3
- [AIRFLOW-2545] Eliminate DeprecationWarning
- [AIRFLOW-2500] Fix MySqlToHiveTransfer to transfer unsigned type properly
- [AIRFLOW-2462] Change PasswordUser setter to correct syntax
- [AIRFLOW-2525] Fix a bug introduced by commit dabf1b9
- [AIRFLOW-2553] Add webserver.pid to .gitignore
- [AIRFLOW-1863][AIRFLOW-2529] Add dag run selection widgets to gantt view
- [AIRFLOW-2504] Log username correctly and add extra to search columns
- [AIRFLOW-2551] Encode binary data with base64 standard rather than base64 url
- [AIRFLOW-2537] Add reset-dagrun option to backfill command
- [AIRFLOW-2526] dag\_run.conf can override params
- [AIRFLOW-2544][AIRFLOW-1967] Guard against next major release of Celery, Flower
- [AIRFLOW-XXX] Add Yieldr to who is using airflow

- [AIRFLOW-2547] Describe how to run tests using Docker
- [AIRFLOW-2538] Update faq doc on how to reduce airflow scheduler latency
- [AIRFLOW-2529] Improve graph view performance and usability
- [AIRFLOW-2517] backfill support passing key values through CLI
- [AIRFLOW-2532] Support logs\_volume\_subpath for KubernetesExecutor
- [AIRFLOW-2466] consider task\_id in \_change\_state\_for\_tis\_without\_dagrun
- [AIRFLOW-2519] Fix CeleryExecutor with SQLAlchemy
- [AIRFLOW-2402] Fix RBAC task log
- [AIRFLOW-XXX] Add M4U to user list
- [AIRFLOW-2536] docs about how to deal with airflow initdb failure
- [AIRFLOW-2530] KubernetesOperator supports multiple clusters
- [AIRFLOW-1499] Eliminate duplicate and unneeded code
- [AIRFLOW-2521] backfill - make variable name and logging messages more accurate
- [AIRFLOW-2429] Fix hook, macros folder flake8 error
- [Airflow-XXX] add Prime to company list
- [AIRFLOW-2525] Fix PostgresHook.copy\_expert to work with "COPY FROM"
- [AIRFLOW-2515] Add dependency on thrift\_sasl to hive extra
- [AIRFLOW-2523] Add how-to for managing GCP connections
- [AIRFLOW-2510] Introduce new macros: prev\_ds and next\_ds
- [AIRFLOW-1730] Unpickle value of XCom queried from DB
- [AIRFLOW-2518] Fix broken ToC links in integration.rst
- [AIRFLOW-1472] Fix SLA misses triggering on skipped tasks.
- [AIRFLOW-2520] CLI - make backfill less verbose
- [AIRFLOW-2107] add time\_partitioning to run\_query on BigQueryBaseCursor
- [AIRFLOW-1057][AIRFLOW-1380][AIRFLOW-2362][2362] AIRFLOW Update DockerOperator to new API
- [AIRFLOW-2415] Make airflow DAG templating render numbers
- [AIRFLOW-2473] Fix wrong skip condition for TransferTests
- [AIRFLOW-2472] Implement MySqlHook.bulk\_dump
- [AIRFLOW-2419] Use default view for subdag operator
- [AIRFLOW-2498] Fix Unexpected argument in SFTP Sensor
- [AIRFLOW-2509] Separate config docs into how-to guides
- [AIRFLOW-2429] Add BaseExecutor back
- [AIRFLOW-2429] Fix dag, example\_dags, executors flake8 error
- [AIRFLOW-2502] Change Single triple quotes to double for docstrings
- [AIRFLOW-2503] Fix broken links in CONTRIBUTING.md
- [AIRFLOW-2501] Refer to devel instructions in docs contrib guide

- [AIRFLOW-2429] Fix contrib folder's flake8 errors
- [AIRFLOW-2471] Fix HiveCliHook.load\_df to use unused parameters
- [AIRFLOW-2495] Update celery to 4.1.1
- [AIRFLOW-2429] Fix api, bin, config\_templates folders flake8 error
- [AIRFLOW-2493] Mark template\_fields of all Operators in the API document as "templated"
- [AIRFLOW-2489] Update FlaskAppBuilder to 1.11.1
- [AIRFLOW-2448] Enhance HiveCliHook.load\_df to work with datetime
- [AIRFLOW-2487] Enhance druid ingestion hook
- [AIRFLOW-2397] Support affinity policies for Kubernetes executor/operator
- [AIRFLOW-2482] Add test for rewrite method in GCS Hook
- [AIRFLOW-2481] Fix flaky Kubernetes test
- [AIRFLOW-2479] Improve doc FAQ section
- [AIRFLOW-2485] Fix Incorrect logging for Qubole Sensor
- [AIRFLOW-2486] Remove unnecessary slash after port
- [AIRFLOW-2429] Make Airflow flake8 compliant
- [AIRFLOW-2491] Resolve flask version conflict
- [AIRFLOW-2484] Remove duplicate key in MySQL to GCS Op
- [AIRFLOW-2458] Add cassandra-to-gcs operator
- [AIRFLOW-2477] Improve time units for task duration and landing times charts for RBAC UI
- [AIRFLOW-2474] Only import snakebite if using py2
- [AIRFLOW-48] Parse connection uri querystring
- [AIRFLOW-2467][AIRFLOW-2] Update import direct warn message to use the module name
- [AIRFLOW-XXX] Fix order of companies
- [AIRFLOW-2452] Document field\_dict must be OrderedDict
- [AIRFLOW-2420] Azure Data Lake Hook
- [AIRFLOW-2213] Add Quoble check operator
- [AIRFLOW-2465] Fix wrong module names in the doc
- [AIRFLOW-1929] Modifying TriggerDagRunOperator to accept execution\_date
- [AIRFLOW-2460] Users can now use volume mounts and volumes
- [AIRFLOW-2110][AIRFLOW-2122] Enhance Http Hook
- [AIRFLOW-XXX] Updated contributors list
- [AIRFLOW-2435] Add launch\_type to ECSOperator to allow FARGATE
- [AIRFLOW-2451] Remove extra slash ('/') char when using wildcard in gcs\_to\_gcs operator
- [AIRFLOW-2461] Add support for cluster scaling on dataproc operator
- [AIRFLOW-2376] Fix no hive section error
- [AIRFLOW-2425] Add lineage support

- [AIRFLOW-2430] Extend query batching to additional slow queries
- [AIRFLOW-2453] Add default nil value for kubernetes/git\_subpath
- [AIRFLOW-2396] Add support for resources in kubernetes operator
- [AIRFLOW-2169] Encode binary data with base64 before importing to BigQuery
- [AIRFLOW-XXX] Add spotahome in user list
- [AIRFLOW-2457] Update FAB version requirement
- [AIRFLOW-2454][Airflow 2454] Support imagePullPolicy for k8s
- [AIRFLOW-2450] update supported k8s versions to 1.9 and 1.10
- [AIRFLOW-2333] Add Segment Hook and TrackEventOperator
- [AIRFLOW-2442][AIRFLOW-2] Airflow run command leaves database connections open
- [AIRFLOW-2016] assign template\_fields for Dataproc Workflow Template sub-classes, not base class
- [AIRFLOW-2446] Add S3ToRedshiftTransfer into the “Integration” doc
- [AIRFLOW-2449] Fix operators.py to run all test cases
- [AIRFLOW-2424] Add dagrun status endpoint and increased k8s test coverage
- [AIRFLOW-2441] Fix bugs in HiveCliHook.load\_df
- [AIRFLOW-2358][AIRFLOW-201804] Make the Kubernetes example optional
- [AIRFLOW-2436] Remove cli\_logger in initdb
- [AIRFLOW-2444] Remove unused option(include\_adhoc) in cli backfill command
- [AIRFLOW-2447] Fix TestHiveMetastoreHook to run all cases
- [AIRFLOW-2445] Allow templating in kubernetes operator
- [AIRFLOW-2086][AIRFLOW-2393] Customize default dagrun number in tree view
- [AIRFLOW-2437] Add PubNub to list of current airflow users
- [AIRFLOW-XXX] Add Quantopian to list of Airflow users
- [AIRFLOW-1978] Add WinRM windows operator and hook
- [AIRFLOW-2427] Add tests to named hive sensor
- [AIRFLOW-2412] Fix HiveCliHook.load\_file to address HIVE-10541
- [AIRFLOW-2431] Add the navigation bar color parameter for RBAC UI
- [AIRFLOW-2407] Resolve Python undefined names
- [AIRFLOW-1952] Add the navigation bar color parameter
- [AIRFLOW-2222] Implement GoogleCloudStorageHook.rewrite
- [AIRFLOW-2426] Add Google Cloud Storage Hook tests
- [AIRFLOW-2418] Bump Flask-WTF
- [AIRFLOW-2417] Wait for pod is not running to end task
- [AIRFLOW-1914] Add other charset support to email utils
- [AIRFLOW-XXX] Update README.md with [Craig@Work](#)
- [AIRFLOW-1899] Fix Kubernetes tests

- [AIRFLOW-1812] Update logging example
- [AIRFLOW-2313] Add TTL parameters for Dataproc
- [AIRFLOW-2411] add dataproc\_jars to templated\_fields
- [AIRFLOW-XXX] Add Reddit to Airflow users
- [AIRFLOW-XXX] Fix wrong table header in scheduler.rst
- [AIRFLOW-2409] Supply password as a parameter
- [AIRFLOW-2410][AIRFLOW-75] Set the timezone in the RBAC Web UI
- [AIRFLOW-2394] default cmds and arguments in kubernetes operator
- [AIRFLOW-2406] Add Apache2 License Shield to Readme
- [AIRFLOW-2404] Add additional documentation for unqueued task
- [AIRFLOW-2400] Add Ability to set Environment Variables for K8s
- [AIRFLOW-XXX] Add Twine Labs as an Airflow user
- [AIRFLOW-1853] Show only the desired number of runs in tree view
- [AIRFLOW-2401] Document the use of variables in Jinja template
- [AIRFLOW-2403] Fix License Headers
- [AIRFLOW-1313] Fix license header
- [AIRFLOW-2398] Add BounceX to list of current airflow users
- [AIRFLOW-2363] Fix return type bug in TaskHandler
- [AIRFLOW-2389] Create a pinot db api hook
- [AIRFLOW-2390] Resolve FlaskWTFDeprecationWarning
- [AIRFLOW-1933] Fix some typos
- [AIRFLOW-1960] Add support for secrets in kubernetes operator
- [AIRFLOW-1313] Add vertica\_to\_mysql operator
- [AIRFLOW-1575] Add AWS Kinesis Firehose Hook for inserting batch records
- [AIRFLOW-2266][AIRFLOW-2343] Remove google-cloud-dataflow dependency
- [AIRFLOW-2370] Implement `--use_random_password` in `create_user`
- [AIRFLOW-2348] Strip path prefix from the `destination_object` when `source_object` contains a wildcard[]
- [AIRFLOW-2391] Fix to Flask 0.12.2
- [AIRFLOW-2381] Fix the flaky `ApiPasswordTests` test
- [AIRFLOW-2378] Add Groupon to list of current users
- [AIRFLOW-2382] Fix wrong description for delimiter
- [AIRFLOW-2380] Add support for environment variables in Spark submit operator.
- [AIRFLOW-2377] Improve Sendgrid sender support
- [AIRFLOW-2331] Support init action timeout on dataproc cluster create
- [AIRFLOW-1835] Update docs: Variable file is json
- [AIRFLOW-1781] Make search case-insensitive in LDAP group

- [AIRFLOW-2042] Fix browser menu appearing over the autocomplete menu
- [AIRFLOW-XXX] Remove wheelhouse files from travis not owned by travis
- [AIRFLOW-2336] Use hmsclient in hive\_hook
- [AIRFLOW-2041] Correct Syntax in python examples
- [AIRFLOW-74] SubdagOperators can consume all celeryd worker processes
- [AIRFLOW-2369] Fix gcs tests
- [AIRFLOW-2365] Fix autocommit attribute check
- [AIRFLOW-2068] MesosExecutor allows optional Docker image
- [AIRFLOW-1652] Push DatabricksRunSubmitOperator metadata into XCOM
- [AIRFLOW-2234] Enable insert\_rows for PrestoHook
- [AIRFLOW-2208][Airflow-22208] Link to same DagRun graph from TaskInstance view
- [AIRFLOW-1153] Allow HiveOperators to take hiveconfs
- [AIRFLOW-775] Fix autocommit settings with Jdbc hook
- [AIRFLOW-2364] Warn when setting autocommit on a connection which does not support it
- [AIRFLOW-2357] Add persistent volume for the logs
- [AIRFLOW-766] Skip conn.commit() when in Auto-commit
- [AIRFLOW-2351] Check for valid default\_args start\_date
- [AIRFLOW-1433] Set default rbac to initdb
- [AIRFLOW-2270] Handle removed tasks in backfill
- [AIRFLOW-2344] Fix *connections -l* to work with pipe/redirect
- [AIRFLOW-2300] Add S3 Select functionality to S3ToHiveTransfer
- [AIRFLOW-1314] Cleanup the config
- [AIRFLOW-1314] Polish some of the Kubernetes docs/config
- [AIRFLOW-1314] Improve error handling
- [AIRFLOW-1999] Add per-task GCP service account support
- [AIRFLOW-1314] Rebasing against master
- [AIRFLOW-1314] Small cleanup to address PR comments (#24)
- [AIRFLOW-1314] Add executor\_config and tests
- [AIRFLOW-1314] Improve k8s support
- [AIRFLOW-1314] Use VolumeClaim for transporting DAGs
- [AIRFLOW-1314] Create integration testing environment
- [AIRFLOW-1314] Git Mode to pull in DAGs for Kubernetes Executor
- [AIRFLOW-1314] Add support for volume mounts & Secrets in Kubernetes Executor
- [AIRFLOW=1314] Basic Kubernetes Mode
- [AIRFLOW-2326][AIRFLOW-2222] remove contrib.gcs\_copy\_operator
- [AIRFLOW-2328] Fix empty GCS blob in S3ToGoogleCloudStorageOperator



- [AIRFLOW-2350] Fix grammar in UPDATING.md
- [AIRFLOW-2302] Fix documentation
- [AIRFLOW-2345] pip is not used in this setup.py
- [AIRFLOW-2347] Add Banco de Formaturas to Readme
- [AIRFLOW-2346] Add Investorise as official user of Airflow
- [AIRFLOW-2330] Do not append destination prefix if not given
- [AIRFLOW-2240][DASK] Added TLS/SSL support for the dask-distributed scheduler.
- [AIRFLOW-2309] Fix duration calculation on TaskFail
- [AIRFLOW-2335] fix issue with jdk8 download for ci
- [AIRFLOW-2184] Add druid\_checker\_operator
- [AIRFLOW-2299] Add S3 Select functionality to S3FileTransformOperator
- [AIRFLOW-2254] Put header as first row in unload
- [AIRFLOW-610] Respect \_cmd option in config before defaults
- [AIRFLOW-2287] Fix incorrect ASF headers
- [AIRFLOW-XXX] Add Zego as an Apache Airflow user
- [AIRFLOW-952] fix save empty extra field in UI
- [AIRFLOW-1325] Add ElasticSearch log handler and reader
- [AIRFLOW-2301] Sync files of an S3 key with a GCS path
- [AIRFLOW-2293] Fix S3FileTransformOperator to work with boto3
- [AIRFLOW-3212][AIRFLOW-2314] Remove only leading slash in GCS path
- [AIRFLOW-1509][AIRFLOW-442] SFTP Sensor
- [AIRFLOW-2291] Add optional params to ML Engine
- [AIRFLOW-1774] Allow consistent templating of arguments in MLEngineBatchPredictionOperator
- [AIRFLOW-2302] Add missing operators and hooks
- [AIRFLOW-2312] Docs Typo Correction: Corresponding
- [AIRFLOW-1623] Trigger on\_kill method in operators
- [AIRFLOW-2162] When impersonating another user, pass env variables to sudo
- [AIRFLOW-2304] Update quickstart doc to mention scheduler part
- [AIRFLOW-1633] docker\_operator needs a way to set shm\_size
- [AIRFLOW-1340] Add S3 to Redshift transfer operator
- [AIRFLOW-2303] Lists the keys inside an S3 bucket
- [AIRFLOW-2209] restore flask\_login imports
- [AIRFLOW-2306] Add Bonnier Broadcasting to list of current users
- [AIRFLOW-2305][AIRFLOW-2027] Fix CI failure caused by []
- [AIRFLOW-2281] Add support for Sendgrid categories
- [AIRFLOW-2027] Only trigger sleep in scheduler after all files have parsed

- [AIRFLOW-2256] SparkOperator: Add Client Standalone mode and retry mechanism
- [AIRFLOW-2284] GCS to S3 operator
- [AIRFLOW-2287] Update license notices
- [AIRFLOW-2296] Add Cinimex DataLab to Readme
- [AIRFLOW-2298] Add Kalibrr to who uses airflow
- [AIRFLOW-2292] Fix docstring for S3Hook.get\_wildcard\_key
- [AIRFLOW-XXX] Update PR template
- [AIRFLOW-XXX] Remove outdated migrations.sql
- [AIRFLOW-2287] Add license header to docs/Makefile
- [AIRFLOW-2286] Add tokopedia to the readme
- [AIRFLOW-2273] Add Discord webhook operator/hook
- [AIRFLOW-2282] Fix grammar in UPDATING.md
- [AIRFLOW-2200] Add snowflake operator with tests
- [AIRFLOW-2178] Add handling on SLA miss errors
- [AIRFLOW-2169] Fix type 'bytes' is not JSON serializable in python3
- [AIRFLOW-2215] Pass environment to subprocess.Popen in base\_task\_runner
- [AIRFLOW-2253] Add Airflow CLI instrumentation
- [AIRFLOW-2274] Fix Dataflow tests
- [AIRFLOW-2269] Add Custom Ink as an Airflow user
- [AIRFLOW-2259] Dataflow Hook Index out of range
- [AIRFLOW-2233] Update updating.md to include the info of hdfs\_sensors renaming
- [AIRFLOW-2217] Add Slack webhook operator
- [AIRFLOW-1729] improve dagBag time
- [AIRFLOW-2264] Improve create\_user cli help message
- [AIRFLOW-2260] [AIRFLOW-2260] SSHOperator add command template .sh files
- [AIRFLOW-2261] Check config/env for remote base log folder
- [AIRFLOW-2258] Allow import of Parquet-format files into BigQuery
- [AIRFLOW-1430] Include INSTALL instructions to avoid GPL
- [AIRFLOW-1430] Solve GPL dependency
- [AIRFLOW-2251] Add Thinknear as an Airflow user
- [AIRFLOW-2244] bugfix: remove legacy LongText code from models.py
- [AIRFLOW-2247] Fix RedshiftToS3Transfer not to fail with ValueError
- [AIRFLOW-2249] Add side-loading support for Zendesk Hook
- [AIRFLOW-XXX] Add Qplum to Airflow users
- [AIRFLOW-2228] Enhancements in ValueCheckOperator
- [AIRFLOW-1206] Typos

- [AIRFLOW-2060] Update pendulum version to 1.4.4
- [AIRFLOW-2248] Fix wrong param name in RedshiftToS3Transfer doc
- [AIRFLOW-1433][AIRFLOW-85] New Airflow Webserver UI with RBAC support
- [AIRFLOW-1235] Fix webserver's odd behaviour
- [AIRFLOW-1460] Allow restoration of REMOVED TI's
- [airflow-2235] Fix wrong docstrings in two operators
- [AIRFLOW-XXX] Fix chronological order for companies using Airflow
- [AIRFLOW-2124] Upload Python file to a bucket for Dataproc
- [AIRFLOW-2212] Fix ungenerated sensor API reference
- [AIRFLOW-2226] Rename google\_cloud\_storage\_default to google\_cloud\_default
- [AIRFLOW-2211] Rename hdfs\_sensors.py to hdfs\_sensor.py for consistency
- [AIRFLOW-2225] Update document to include DruidDbApiHook
- [Airflow-2202] Add filter support in HiveMetastoreHook().max\_partition()
- [AIRFLOW-2220] Remove duplicate numeric list entry in security.rst
- [AIRFLOW-XXX] Update tutorial documentation
- [AIRFLOW-2215] Update celery task to preserve environment variables and improve logging on exception
- [AIRFLOW-2185] Use state instead of query param
- [AIRFLOW-2183] Refactor DruidHook to enable sql
- [AIRFLOW-2203] Defer cycle detection
- [AIRFLOW-2203] Remove Useless Commands.
- [AIRFLOW-2203] Cache signature in apply\_defaults
- [AIRFLOW-2203] Speed up Operator Resources
- [AIRFLOW-2203] Cache static rules (trigger/weight)
- [AIRFLOW-2203] Store task ids as sets not lists
- [AIRFLOW-2205] Remove unsupported args from JdbcHook doc
- [AIRFLOW-2207] Fix flaky test that uses app.cached\_app()
- [AIRFLOW-2206] Remove unsupported args from JdbcOperator doc
- [AIRFLOW-2140] Add Kubernetes scheduler to SparkSubmitOperator
- [AIRFLOW-XXX] Add Xero to list of users
- [AIRFLOW-2204] Fix webserver debug mode
- [AIRFLOW-102] Fix test\_complex\_template always succeeds
- [AIRFLOW-442] Add SFTPHook
- [AIRFLOW-2169] Add schema to MySqlToGoogleCloudStorageOperator
- [AIRFLOW-2184][AIRFLOW-2138] Google Cloud Storage allow wildcards
- [AIRFLOW-1588] Cast Variable value to string
- [AIRFLOW-2199] Fix invalid reference to logger

- [AIRFLOW-2191] Change scheduler heartbeat logs from info to debug
- [AIRFLOW-2106] Salesforce hook sandbox option
- [AIRFLOW-2197] Silence hostname\_callable config error message
- [AIRFLOW-2150] Use lighter call in HiveMetastoreHook().max\_partition()
- [AIRFLOW-2186] Change the way logging is carried out in few ops
- [AIRFLOW-2181] Convert password\_auth and test\_password\_endpoints from DOS to UNIX
- [AIRFLOW-2187] Fix Broken Travis CI due to AIRFLOW-2123
- [AIRFLOW-2175] Check that filepath is not None
- [AIRFLOW-2173] Don't check task IDs for concurrency reached check
- [AIRFLOW-2168] Remote logging for Azure Blob Storage
- [AIRFLOW-XXX] Add DocuTAP to list of users
- [AIRFLOW-2176] Change the way logging is carried out in BQ Get Data Operator
- [AIRFLOW-2177] Add mock test for GCS Download op
- [AIRFLOW-2123] Install CI dependencies from setup.py
- [AIRFLOW-2129] Presto hook calls \_parse\_exception\_message but defines \_get\_pretty\_exception\_message
- [AIRFLOW-2174] Fix typos and wrongly rendered documents
- [AIRFLOW-2171] Store delegated credentials
- [AIRFLOW-2166] Restore BQ run\_query dialect param
- [AIRFLOW-2163] Add HBC Digital to users of airflow
- [AIRFLOW-2065] Fix race-conditions when creating loggers
- [AIRFLOW-2147] Plugin manager: added 'sensors' attribute
- [AIRFLOW-2059] taskinstance query is awful, un-indexed, and does not scale
- [AIRFLOW-2159] Fix a few typos in salesforce\_hook
- [AIRFLOW-2132] Add step to initialize database
- [AIRFLOW-2160] Fix bad rowid deserialization
- [AIRFLOW-2161] Add Vevo to list of companies using Airflow
- [AIRFLOW-2149] Add link to apache Beam documentation to create self executing Jar
- [AIRFLOW-2151] Allow getting the session from AwsHook
- [AIRFLOW-2097] tz referenced before assignment
- [AIRFLOW-2152] Add Multiply to list of companies using Airflow
- [AIRFLOW-1551] Add operator to trigger Jenkins job
- [AIRFLOW-2034] Fix mixup between %s and { } when using str.format Convention is to use .format for string formatting outside logging, else use lazy format See comment in related issue <https://github.com/apache/airflow/pull/2823/files> Identified problematic case using following command line `.git/COMMIT_EDITMSG:grep -r '%s'./* | grep 'format('`
- [AIRFLOW-2102] Add custom\_args to Sendgrid personalizations
- [AIRFLOW-1035][AIRFLOW-1053] import unicode\_literals to parse Unicode in HQL

- [AIRFLOW-2127] Keep loggers during DB migrations
- [AIRFLOW-2146] Resolve issues with BQ using DbApiHook methods
- [AIRFLOW-2087] Scheduler Report shows incorrect Total task number
- [AIRFLOW-2139] Remove unnecessary boilerplate to get DataFrame using pandas\_gbq
- [AIRFLOW-2125] Using binary package psycpg2-binary
- [AIRFLOW-2142] Include message on mkdir failure
- [AIRFLOW-1615] SSHHook: use port specified by Connection
- [AIRFLOW-2122] Handle boolean values in sshHook
- [AIRFLOW-XXX] Add Tile to the list of users
- [AIRFLOW-2130] Add missing Operators to API Reference docs
- [AIRFLOW-XXX] Add timeout units (seconds)
- [AIRFLOW-2134] Add Alan to the list of companies that use Airflow
- [AIRFLOW-2133] Remove references to GitHub issues in CONTRIBUTING
- [AIRFLOW-2131] Remove confusing AirflowImport docs
- [AIRFLOW-1852] Allow hostname to be overridable.
- [AIRFLOW-2126] Add Bluecore to active users
- [AIRFLOW-1618] Add feature to create GCS bucket
- [AIRFLOW-2108] Fix log indentation in BashOperator
- [AIRFLOW-2115] Fix doc links to PythonHosted
- [AIRFLOW-XXX] Add contributor from Easy company
- [AIRFLOW-1882] Add ignoreUnknownValues option to gcs\_to\_bq operator
- [AIRFLOW-2089] Add on kill for SparkSubmit in Standalone Cluster
- [AIRFLOW-2113] Address missing DagRun callbacks Given that the handle\_callback method belongs to the DAG object, we are able to get the list of task directly with get\_task and reduce the communication with the database, making airflow more lightweight.
- [AIRFLOW-2112] Fix svg width for Recent Tasks on UI.
- [AIRFLOW-2116] Set CI Cloudant version to <2.0
- [AIRFLOW-XXX] Add PMC to list of companies using Airflow
- [AIRFLOW-2100] Fix Broken Documentation Links
- [AIRFLOW-1404] Add 'flatten\_results' & 'maximum\_bytes\_billed' to BQ Operator
- [AIRFLOW-800] Initialize valid Google BigQuery Connection
- [AIRFLOW-1319] Fix misleading SparkSubmitOperator and SparkSubmitHook docstring
- [AIRFLOW-1983] Parse environment parameter as template
- [AIRFLOW-2095] Add operator to create External BigQuery Table
- [AIRFLOW-2085] Add SparkJdbc operator
- [AIRFLOW-1002] Add ability to clean all dependencies of removed DAG
- [AIRFLOW-2094] Jinjafied project\_id, region & zone in DataProc{\*} Operators

- [AIRFLOW-2092] Fixed incorrect parameter in docstring for FTPHook
- [AIRFLOW-XXX] Add SocialCops to Airflow users
- [AIRFLOW-2088] Fix duplicate keys in MySQL to GCS Helper function
- [AIRFLOW-2091] Fix incorrect docstring parameter in BigQuery Hook
- [AIRFLOW-2090] Fix typo in DataStore Hook
- [AIRFLOW-1157] Fix missing pools crashing the scheduler
- [AIRFLOW-713] Jinjafy {EmrCreateJobFlow,EmrAddSteps}Operator attributes
- [AIRFLOW-2083] Docs: Use “its” instead of “it’s” where appropriate
- [AIRFLOW-2066] Add operator to create empty BQ table
- [AIRFLOW-XXX] add Karmic to list of companies
- [AIRFLOW-2073] Make FileSensor fail when the file doesn’t exist
- [AIRFLOW-2078] Improve task\_stats and dag\_stats performance
- [AIRFLOW-2080] Use a log-out icon instead of a power button
- [AIRFLOW-2077] Fetch all pages of list\_objects\_v2 response
- [AIRFLOW-XXX] Add TM to list of companies
- [AIRFLOW-1985] Impersonation fixes for using *run\_as\_user*
- [AIRFLOW-2018][AIRFLOW-2] Make Sensors backward compatible
- [AIRFLOW-XXX] Fix typo in concepts doc (dag\_md)
- [AIRFLOW-2069] Allow Bytes to be uploaded to S3
- [AIRFLOW-2074] Fix log var name in GHE auth
- [AIRFLOW-1927] Convert naive datetimes for TaskInstances
- [AIRFLOW-1760] Password auth for experimental API
- [AIRFLOW-2038] Add missing kubernetes dependency for dev
- [AIRFLOW-2040] Escape special chars in task instance logs URL
- [AIRFLOW-1968][AIRFLOW-1520] Add role\_arn and aws\_account\_id/aws\_iam\_role support back to aws hook
- [AIRFLOW-2048] Fix task instance failure string formatting
- [AIRFLOW-2046] Fix kerberos error to work with python 3.x
- [AIRFLOW-2063] Add missing docs for GCP
- [AIRFLOW-XXX] Fix typo in docs
- [AIRFLOW-1793] Use docker\_url instead of invalid base\_url
- [AIRFLOW-2055] Elaborate on slightly ambiguous documentation
- [AIRFLOW-2039] BigQueryOperator supports priority property
- [AIRFLOW-2053] Fix quote character bug in BQ hook
- [AIRFLOW-2057] Add Overstock to list of companies
- [AIRFLOW-XXX] Add Plaid to Airflow users

- [AIRFLOW-2044] Add SparkSubmitOperator to documentation
- [AIRFLOW-2037] Add methods to get Hash values of a GCS object
- [AIRFLOW-2050] Fix Travis permission problem
- [AIRFLOW-2043] Add Intercom to list of companies
- [AIRFLOW-2023] Add debug logging around number of queued files
- [AIRFLOW-XXX] Add Pernod-ricard as a airflow user
- [AIRFLOW-1453] Add 'steps' into template\_fields in EmrAddSteps
- [AIRFLOW-2015] Add flag for interactive runs
- [AIRFLOW-1895] Fix primary key integrity for mysql
- [AIRFLOW-2030] Fix KeyError:i in DbApiHook for insert
- [AIRFLOW-1943] Add External BigQuery Table feature
- [AIRFLOW-2033] Add Google Cloud Storage List Operator
- [AIRFLOW-2006] Add local log catching to kubernetes operator
- [AIRFLOW-2031] Add missing gcp\_conn\_id in the example in DataFlow docstrings
- [AIRFLOW-2029] Fix AttributeError in BigQueryPandasConnector
- [AIRFLOW-2028] Add JobTeaser to official users list
- [AIRFLOW-2016] Add support for Dataproc Workflow Templates
- [AIRFLOW-2025] Reduced Logging verbosity
- [AIRFLOW-1267][AIRFLOW-1874] Add dialect parameter to BigQueryHook
- [AIRFLOW-XXX] Fixed a typo
- [AIRFLOW-XXX] Typo node to nodes
- [AIRFLOW-2019] Update DataflowHook for updating Streaming type job
- [AIRFLOW-2017][Airflow 2017] adding query output to PostgresOperator
- [AIRFLOW-1889] Split sensors into separate files
- [AIRFLOW-1950] Optionally pass xcom\_pull task\_ids
- [AIRFLOW-1755] Allow mount below root
- [AIRFLOW-511][Airflow 511] add success/failure callbacks on dag level
- [AIRFLOW-192] Add weight\_rule param to BaseOperator
- [AIRFLOW-2008] Use callable for python column defaults
- [AIRFLOW-1984] Fix to AWS Batch operator
- [AIRFLOW-2000] Support non-main dataflow job class
- [AIRFLOW-2003] Use flask-caching instead of flask-cache
- [AIRFLOW-2002] Do not swallow exception on logging import
- [AIRFLOW-2004] Import flash from flask not flask.login
- [AIRFLOW-1997] Fix GCP operator doc strings
- [AIRFLOW-1996] Update DataflowHook waitfordone for Streaming type job[]

- [AIRFLOW-1995][Airflow 1995] add on\_kill method to SqoopOperator
- [AIRFLOW-1770] Allow HiveOperator to take in a file
- [AIRFLOW-1994] Change background color of Scheduled state Task Instances
- [AIRFLOW-1436][AIRFLOW-1475] EmrJobFlowSensor considers Cancelled step as Successful
- [AIRFLOW-1517] Kubernetes operator PR fixes
- [AIRFLOW-1517] addressed PR comments
- [AIRFLOW-1517] started documentation of k8s operator
- [AIRFLOW-1517] Restore authorship of resources
- [AIRFLOW-1517] Remove authorship of resources
- [AIRFLOW-1517] Add minikube for kubernetes integration tests
- [AIRFLOW-1517] Restore authorship of resources
- [AIRFLOW-1517] fixed license issues
- [AIRFLOW-1517] Created more accurate failures for kube cluster issues
- [AIRFLOW-1517] Remove authorship of resources
- [AIRFLOW-1517] Add minikube for kubernetes integration tests
- [AIRFLOW-1988] Change BG color of None state TIs
- [AIRFLOW-790] Clean up TaskInstances without DagRuns
- [AIRFLOW-1949] Fix var upload, str() produces “b’...’” which is not json
- [AIRFLOW-1930] Convert func.now() to timezone.utcnow()
- [AIRFLOW-1688] Support load.time\_partitioning in bigquery\_hook
- [AIRFLOW-1975] Make TriggerDagRunOperator callback optional
- [AIRFLOW-1480] Render template attributes for ExternalTaskSensor fields
- [AIRFLOW-1958] Add **\*\***kwargs to send\_email
- [AIRFLOW-1976] Fix for missing log/logger attribute FileProcessHandler
- [AIRFLOW-1982] Fix Executor event log formatting
- [AIRFLOW-1971] Propagate hive config on impersonation
- [AIRFLOW-1969] Always use HTTPS URIs for Google OAuth2
- [AIRFLOW-1954] Add DataFlowTemplateOperator
- [AIRFLOW-1963] Add config for HiveOperator mapred\_queue
- [AIRFLOW-1946][AIRFLOW-1855] Create a BigQuery Get Data Operator
- [AIRFLOW-1953] Add labels to dataflow operators
- [AIRFLOW-1967] Update Celery to 4.0.2
- [AIRFLOW-1964] Add Upsight to list of Airflow users
- [AIRFLOW-XXX] Changelog for 1.9.0
- [AIRFLOW-1470] Implement BashSensor operator
- [AIRFLOW-XXX] Pin sqlalchemy dependency



- [AIRFLOW-1955] Do not reference unassigned variable
- [AIRFLOW-1957] Add contributor to BalanceHero in Readme
- [AIRFLOW-1517] Restore authorship of secrets and init container
- [AIRFLOW-1517] Remove authorship of secrets and init container
- [AIRFLOW-1935] Add BalanceHero to readme
- [AIRFLOW-1939] add astronomer contributors
- [AIRFLOW-1517] Kubernetes Operator
- [AIRFLOW-1928] Fix @once with catchup=False
- [AIRFLOW-1937] Speed up scheduling by committing in batch
- [AIRFLOW-1821] Enhance default logging config by removing extra loggers
- [AIRFLOW-1904] Correct DAG fileloc to the right filepath
- [AIRFLOW-1909] Update docs with supported versions of MySQL server
- [AIRFLOW-1915] Relax flask-wtf dependency specification
- [AIRFLOW-1920] Update CONTRIBUTING.md to reflect enforced linting rules
- [AIRFLOW-1942] Update Sphinx docs to remove deprecated import structure
- [AIRFLOW-1846][AIRFLOW-1697] Hide Ad Hoc Query behind secure\_mode config
- [AIRFLOW-1948] Include details for on\_kill failure
- [AIRFLOW-1938] Clean up unused exception
- [AIRFLOW-1932] Add GCP Pub/Sub Pull and Ack
- [AIRFLOW-XXX] Purge coveralls
- [AIRFLOW-XXX] Remove unused coveralls token
- [AIRFLOW-1938] Remove tag version check in setup.py
- [AIRFLOW-1916] Don't upload logs to remote from *run -raw*
- [AIRFLOW-XXX] Fix failing PubSub tests on Python3
- [AIRFLOW-XXX] Upgrade to python 3.5 and disable dask tests
- [AIRFLOW-1913] Add new GCP PubSub operators
- [AIRFLOW-1525] Fix minor LICENSE and NOTICE issues
- [AIRFLOW-1687] fix fernet error without encryption
- [AIRFLOW-1912] airflow.processor should not propagate logging
- [AIRFLOW-1911] Rename celeryd\_concurrency
- [AIRFLOW-1885] Fix IndexError in ready\_prefix\_on\_cmdline
- [AIRFLOW-1854] Improve Spark Submit operator for standalone cluster mode
- [AIRFLOW-1908] Fix celery broker options config load
- [AIRFLOW-1907] Pass max\_ingestion\_time to Druid hook
- [AIRFLOW-1909] Add away to list of users
- [AIRFLOW-1893][AIRFLOW-1901] Propagate PYTHONPATH when using impersonation

- [AIRFLOW-1892] Modify BQ hook to extract data filtered by column
- [AIRFLOW-1829] Support for schema updates in query jobs
- [AIRFLOW-1840] Make celery configuration congruent with Celery 4
- [AIRFLOW-1878] Fix stderr/stdout redirection for tasks
- [AIRFLOW-1897][AIRFLOW-1873] Task Logs for running instance not visible in WebUI
- [AIRFLOW-1896] FIX bleach <> html5lib incompatibility
- [AIRFLOW-1884][AIRFLOW-1059] Reset orphaned task state for external dagruns
- [AIRFLOW-XXX] Fix typo in comment
- [AIRFLOW-1869] Do not emit spurious warning on missing logs
- [AIRFLOW-1888] Add AWS Redshift Cluster Sensor
- [AIRFLOW-1887] Renamed endpoint url variable
- [AIRFLOW-1873] Set TI.try\_number to right value depending TI state
- [AIRFLOW-1891] Fix non-ascii typo in default configuration template
- [AIRFLOW-1879] Handle ti log entirely within ti
- [AIRFLOW-1869] Write more error messages into gcs and file logs
- [AIRFLOW-1876] Write subtask id to task log header
- [AIRFLOW-1554] Fix wrong DagFileProcessor termination method call
- [AIRFLOW-342] Do not use amqp, rpc as result backend
- [AIRFLOW-966] Make celery broker\_transport\_options configurable
- [AIRFLOW-1881] Make operator log in task log
- [AIRFLOW-XXX] Added DataReply to the list of Airflow Users
- [AIRFLOW-1883] Get File Size for objects in Google Cloud Storage
- [AIRFLOW-1872] Set context for all handlers including parents
- [AIRFLOW-1855][AIRFLOW-1866] Add GCS Copy Operator to copy multiple files
- [AIRFLOW-1870] Enable flake8 tests
- [AIRFLOW-1785] Enable Python 3 tests
- [AIRFLOW-1850] Copy cmd before masking
- [AIRFLOW-1665] Reconnect on database errors
- [AIRFLOW-1559] Dispose SQLAlchemy engines on exit
- [AIRFLOW-1559] Close file handles in subprocesses
- [AIRFLOW-1559] Make database pooling optional
- [AIRFLOW-1848][Airflow-1848] Fix DataFlowPythonOperator py\_file extension doc comment
- [AIRFLOW-1843] Add Google Cloud Storage Sensor with prefix
- [AIRFLOW-1803] Time zone documentation
- [AIRFLOW-1826] Update views to use timezone aware objects
- [AIRFLOW-1827] Fix api endpoint date parsing

- [AIRFLOW-1806] Use naive datetime when using cron
- [AIRFLOW-1809] Update tests to use timezone aware objects
- [AIRFLOW-1806] Use naive datetime for cron scheduling
- [AIRFLOW-1807] Force use of time zone aware db fields
- [AIRFLOW-1808] Convert all utcnow() to time zone aware
- [AIRFLOW-1804] Add time zone configuration options
- [AIRFLOW-1802] Convert database fields to timezone aware
- [AIRFLOW-XXX] Add dask lock files to excludes
- [AIRFLOW-1790] Add support for AWS Batch operator
- [AIRFLOW-XXX] Update README.md
- [AIRFLOW-1820] Remove timestamp from metric name
- [AIRFLOW-1810] Remove unused mysql import in migrations.
- [AIRFLOW-1838] Properly log collect\_dags exception
- [AIRFLOW-1842] Fixed Super class name for the gcs to gcs copy operator
- [AIRFLOW-1845] Modal background now covers long or tall pages
- [AIRFLOW-1229] Add link to Run Id, incl execution\_date
- [AIRFLOW-1842] Add gcs to gcs copy operator with renaming if required
- [AIRFLOW-1841] change False to None in operator and hook
- [AIRFLOW-1839] Fix more bugs in S3Hook boto -> boto3 migration
- [AIRFLOW-1830] Support multiple domains in Google authentication backend
- [AIRFLOW-1831] Add driver-classpath spark submit
- [AIRFLOW-1795] Correctly call S3Hook after migration to boto3
- [AIRFLOW-1811] Fix render Druid operator
- [AIRFLOW-1819] Fix slack operator unittest bug
- [AIRFLOW-1805] Allow Slack token to be passed through connection
- [AIRFLOW-1816] Add region param to Dataproc operators
- [AIRFLOW-868] Add postgres\_to\_gcs operator and unittests
- [AIRFLOW-1613] make mysql\_to\_gcs\_operator py3 compatible
- [AIRFLOW-1817] use boto3 for s3 dependency
- [AIRFLOW-1813] Bug SSH Operator empty buffer
- [AIRFLOW-1801][AIRFLOW-288] Url encode execution dates
- [AIRFLOW-1563] Catch OSError while symlinking the latest log directory
- [AIRFLOW-1794] Remove uses of Exception.message for Python 3
- [AIRFLOW-1799] Fix logging line which raises errors
- [AIRFLOW-1102] Upgrade Gunicorn >=19.4.0
- [AIRFLOW-1756] Fix S3TaskHandler to work with Boto3-based S3Hook

- [AIRFLOW-1797] S3Hook.load\_string didn't work on Python3
- [AIRFLOW-646] Add docutils to setup\_requires
- [AIRFLOW-1792] Missing intervals DruidOperator
- [AIRFLOW-1789][AIRFLOW-1712] Log SSHOperator stderr to log.warning
- [AIRFLOW-1787] Fix task instance batch clear and set state bugs
- [AIRFLOW-1780] Fix long output lines with unicode from hanging parent
- [AIRFLOW-387] Close SQLAlchemy sessions properly
- [AIRFLOW-1779] Add keepalive packets to ssh hook
- [AIRFLOW-1669] Fix Docker and pin Moto to 1.1.19
- [AIRFLOW-71] Add support for private Docker images
- [AIRFLOW-XXX] Give a clue what the 'ds' variable is
- [AIRFLOW-XXX] Correct typos in the faq docs page
- [AIRFLOW-1571] Add AWS Lambda Hook
- [AIRFLOW-1675] Fix docstrings for API docs
- [AIRFLOW-1712][AIRFLOW-756][AIRFLOW-751] Log SSHOperator output
- [AIRFLOW-1776] Capture stdout and stderr for logging
- [AIRFLOW-1765] Make experimental API securable without needing Kerberos.
- [AIRFLOW-1764] The web interface should not use the experimental API
- [AIRFLOW-1771] Rename heartbeat to avoid confusion
- [AIRFLOW-1769] Add support for templates in VirtualenvOperator
- [AIRFLOW-1763] Fix S3TaskHandler unit tests
- [AIRFLOW-1315] Add Qubole File & Partition Sensors
- [AIRFLOW-1018] Make processor use logging framework
- [AIRFLOW-1695] Add RedshiftHook using boto3
- [AIRFLOW-1706] Fix query error for MSSQL backend
- [AIRFLOW-1711] Use ldap3 dict for group membership
- [AIRFLOW-1723] Make sendgrid a plugin
- [AIRFLOW-1757] Add missing options to SparkSubmitOperator
- [AIRFLOW-1734][Airflow 1734] Sqoop hook/operator enhancements
- [AIRFLOW-1761] Fix type in scheduler.rst
- [AIRFLOW-1731] Set pythonpath for logging
- [AIRFLOW-1641] Handle executor events in the scheduler
- [AIRFLOW-1744] Make sure max\_tries can be set
- [AIRFLOW-1732] Improve dataflow hook logging
- [AIRFLOW-1736] Add HotelQuickly to Who Uses Airflow
- [AIRFLOW-1657] Handle failing qubole operator

- [AIRFLOW-1677] Fix typo in example\_qubole\_operator
- [AIRFLOW-926] Fix JDBC Hook
- [AIRFLOW-1520] Boto3 S3Hook, S3Log
- [AIRFLOW-1716] Fix multiple `__init__` def in SimpleDag
- [AIRFLOW-XXX] Fix DateTime in Tree View
- [AIRFLOW-1719] Fix small typo
- [AIRFLOW-1432] Charts label for Y axis not visible
- [AIRFLOW-1743] Verify ldap filters correctly
- [AIRFLOW-1745] Restore default signal disposition
- [AIRFLOW-1741] Correctly hide second chart on task duration page
- [AIRFLOW-1728] Add networkUri, subnet, tags to Dataproc operator
- [AIRFLOW-1726] Add copy\_expert psycopg2 method to PostgresHook
- [AIRFLOW-1330] Add conn\_type argument to CLI when adding connection
- [AIRFLOW-1698] Remove SCHEDULER\_RUNS env var in systemd
- [AIRFLOW-1694] Stop using itertools.izip
- [AIRFLOW-1692] Change test\_views filename to support Windows
- [AIRFLOW-1722] Fix typo in scheduler autorestart output filename
- [AIRFLOW-1723] Support sendgrid in email backend
- [AIRFLOW-1718] Set num\_retries on Dataproc job request execution
- [AIRFLOW-1727] Add unit tests for DataProcHook
- [AIRFLOW-1631] Fix timing issue in unit test
- [AIRFLOW-1631] Fix local executor unbound parallelism
- [AIRFLOW-1724] Add Fundera to Who uses Airflow?
- [AIRFLOW-1683] Cancel BigQuery job on timeout.
- [AIRFLOW-1714] Fix misspelling: s/seperate/separate/
- [AIRFLOW-1681] Add batch clear in task instance view
- [AIRFLOW-1696] Fix dataproc version label error
- [AIRFLOW-1613] Handle binary field in MySqlToGoogleCloudStorageOperator
- [AIRFLOW-1697] Mode to disable charts endpoint
- [AIRFLOW-1691] Add better Google cloud logging documentation
- [AIRFLOW-1690] Add detail to gcs error messages
- [AIRFLOW-1682] Make S3TaskHandler write to S3 on close
- [AIRFLOW-1634] Adds task\_concurrency feature
- [AIRFLOW-1676] Make GCSTaskHandler write to GCS on close
- [AIRFLOW-1678] Fix erroneously repeated word in function docstrings
- [AIRFLOW-1323] Made Dataproc operator parameter names consistent

- [AIRFLOW-1590] fix unused module and variable
- [AIRFLOW-1671] Add @apply\_defaults back to gcs download operator
- [AIRFLOW-988] Fix repeating SLA miss callbacks
- [AIRFLOW-1611] Customize logging
- [AIRFLOW-1668] Expose keepalives\_idle for Postgres connections
- [AIRFLOW-1658] Kill Druid task on timeout
- [AIRFLOW-1669][AIRFLOW-1368] Fix Docker import
- [AIRFLOW-891] Make webserver clock include date
- [AIRFLOW-1560] Add AWS DynamoDB hook and operator for inserting batch items
- [AIRFLOW-1654] Show tooltips for link icons in DAGs view
- [AIRFLOW-1660] Change webpage width to full-width
- [AIRFLOW-1664] write file as binary instead of str
- [AIRFLOW-1659] Fix invalid obj attribute bug in file\_task\_handler.py
- [AIRFLOW-1635] Allow creating GCP connection without requiring a JSON file
- [AIRFLOW-1650] Fix custom celery config loading
- [AIRFLOW-1647] Fix Spark-sql hook
- [AIRFLOW-1587] Fix CeleryExecutor import error
- [Airflow-1640][AIRFLOW-1640] Add qubole default connection
- [AIRFLOW-1576] Added region param to Dataproc{\*}Operators
- [AIRFLOW-1643] Add healthjump to officially using list
- [AIRFLOW-1626] Add Azri Solutions to Airflow users
- [AIRFLOW-1636] Add AWS and EMR connection type
- [AIRFLOW-1527] Refactor celery config
- [AIRFLOW-1639] Fix Fernet error handling
- [AIRFLOW-1637] Fix Travis CI build status link
- [AIRFLOW-1628] Fix docstring of sqlsensor
- [AIRFLOW-1331] add SparkSubmitOperator option
- [AIRFLOW-1627] Only query pool in SubDAG init when necessary
- [AIRFLOW-1629] Make extra a textarea in edit connections form
- [AIRFLOW-1368] Automatically remove Docker container on exit
- [AIRFLOW-289] Make airflow timezone independent
- [AIRFLOW-1356] Add *-celery\_hostname* to *airflow worker*
- [AIRFLOW-1247] Fix ignore\_all\_dependencies argument ignored
- [AIRFLOW-1621] Add tests for server side paging
- [AIRFLOW-1591] Avoid attribute error when rendering logging filename
- [AIRFLOW-1031] Replace hard-code to DagRun.ID\_PREFIX

- [AIRFLOW-1604] Rename logger to log
- [AIRFLOW-1512] Add PythonVirtualenvOperator
- [AIRFLOW-1617] Fix XSS vulnerability in Variable endpoint
- [AIRFLOW-1497] Reset hidden fields when changing connection type
- [AIRFLOW-1619] Add poll\_sleep parameter to GCP dataflow operator
- [AIRFLOW-XXX] Remove landscape.io config
- [AIRFLOW-XXX] Remove non working service badges
- [AIRFLOW-1177] Fix Variable.setdefault w/existing JSON
- [AIRFLOW-1600] Fix exception handling in get\_fernet
- [AIRFLOW-1614] Replace inspect.stack() with sys.\_getframe()
- [AIRFLOW-1519] Add server side paging in DAGs list
- [AIRFLOW-1309] Allow hive\_to\_druid to take tblproperties
- [AIRFLOW-1613] Make MySqlToGoogleCloudStorageOperator compatible with python3
- [AIRFLOW-1603] add PAYMILL to companies list
- [AIRFLOW-1609] Fix gitignore to ignore all venvs
- [AIRFLOW-1601] Add configurable task cleanup time

### 3.19.4 Airflow 1.9.0, 2018-01-02

- [AIRFLOW-1525] Fix minor LICENSE and NOTICE issues
- [AIRFLOW-XXX] Bump version to 1.9.0
- [AIRFLOW-1897][AIRFLOW-1873] Task Logs for running instance not visible in WebUI
- [AIRFLOW-XXX] Make sure session is committed
- [AIRFLOW-1896] FIX bleach <> html5lib incompatibility
- [AIRFLOW-XXX] Fix log handler test
- [AIRFLOW-1873] Set TI.try\_number to right value depending TI state
- [AIRFLOW-1554] Fix wrong DagFileProcessor termination method call
- [AIRFLOW-1872] Set context for all handlers including parents
- [AIRFLOW-XXX] Add dask lock files to excludes
- [AIRFLOW-1839] Fix more bugs in S3Hook boto -> boto3 migration
- [AIRFLOW-1795] Correctly call S3Hook after migration to boto3
- [AIRFLOW-1813] Bug SSH Operator empty buffer
- [AIRFLOW-1794] Remove uses of Exception.message for Python 3
- [AIRFLOW-1799] Fix logging line which raises errors
- [AIRFLOW-1102] Upgrade Gunicorn >=19.4.0
- [AIRFLOW-1756] Fix S3TaskHandler to work with Boto3-based S3Hook
- [AIRFLOW-1797] S3Hook.load\_string didn't work on Python3

- [AIRFLOW-1792] Missing intervals DruidOperator
- [AIRFLOW-1789][AIRFLOW-1712] Log SSHOperator stderr to log.warning
- [AIRFLOW-1669] Fix Docker and pin Moto to 1.1.19
- [AIRFLOW-71] Add support for private Docker images
- [AIRFLOW-1779] Add keepalive packets to ssh hook
- [AIRFLOW-XXX] Give a clue what the 'ds' variable is
- [AIRFLOW-XXX] Correct typos in the faq docs page
- [AIRFLOW-1571] Add AWS Lambda Hook
- [AIRFLOW-1675] Fix docstrings for API docs
- [AIRFLOW-1712][AIRFLOW-756][AIRFLOW-751] Log SSHOperator output
- [AIRFLOW-1776] Capture stdout and stderr for logging
- [AIRFLOW-1765] Make experimental API securable without needing Kerberos.
- [AIRFLOW-1764] The web interface should not use the experimental API
- [AIRFLOW-1634] Adds task\_concurrency feature
- [AIRFLOW-1018] Make processor use logging framework
- [AIRFLOW-1695] Add RedshiftHook using boto3
- [AIRFLOW-1706] Fix query error for MSSQL backend
- [AIRFLOW-1711] Use ldap3 dict for group membership
- [AIRFLOW-1757] Add missing options to SparkSubmitOperator
- [AIRFLOW-1734][Airflow 1734] Sqoop hook/operator enhancements
- [AIRFLOW-1731] Set pythonpath for logging
- [AIRFLOW-1641] Handle executor events in the scheduler
- [AIRFLOW-1744] Make sure max\_tries can be set
- [AIRFLOW-1330] Add conn\_type argument to CLI when adding connection
- [AIRFLOW-926] Fix JDBC Hook
- [AIRFLOW-1520] Boto3 S3Hook, S3Log
- [AIRFLOW-XXX] Fix DateTime in Tree View
- [AIRFLOW-1432] Charts label for Y axis not visible
- [AIRFLOW-1743] Verify ldap filters correctly
- [AIRFLOW-1745] Restore default signal disposition
- [AIRFLOW-1741] Correctly hide second chart on task duration page
- [AIRFLOW-1726] Add copy\_expert psycopg2 method to PostgresHook
- [AIRFLOW-1698] Remove SCHEDULER\_RUNS env var in systemd
- [AIRFLOW-1694] Stop using itertools.izip
- [AIRFLOW-1692] Change test\_views filename to support Windows
- [AIRFLOW-1722] Fix typo in scheduler autorestart output filename



- [AIRFLOW-1691] Add better Google cloud logging documentation
- [AIRFLOW-1690] Add detail to gcs error messages
- [AIRFLOW-1682] Make S3TaskHandler write to S3 on close
- [AIRFLOW-1676] Make GCSTaskHandler write to GCS on close
- [AIRFLOW-1635] Allow creating GCP connection without requiring a JSON file
- [AIRFLOW-1323] Made Dataproc operator parameter names consistent
- [AIRFLOW-1590] fix unused module and variable
- [AIRFLOW-988] Fix repeating SLA miss callbacks
- [AIRFLOW-1611] Customize logging
- [AIRFLOW-1668] Expose keepalives\_idle for Postgres connections
- [AIRFLOW-1658] Kill Druid task on timeout
- [AIRFLOW-1669][AIRFLOW-1368] Fix Docker import
- [AIRFLOW-1560] Add AWS DynamoDB hook and operator for inserting batch items
- [AIRFLOW-1654] Show tooltips for link icons in DAGs view
- [AIRFLOW-1660] Change webpage width to full-width
- [AIRFLOW-1664] write file as binary instead of str
- [AIRFLOW-1659] Fix invalid obj attribute bug in file\_task\_handler.py
- [AIRFLOW-1650] Fix custom celery config loading
- [AIRFLOW-1647] Fix Spark-sql hook
- [AIRFLOW-1587] Fix CeleryExecutor import error
- [AIRFLOW-1636] Add AWS and EMR connection type
- [AIRFLOW-1527] Refactor celery config
- [AIRFLOW-1639] Fix Fernet error handling
- [AIRFLOW-1628] Fix docstring of sqlsensor
- [AIRFLOW-1331] add SparkSubmitOperator option
- [AIRFLOW-1627] Only query pool in SubDAG init when necessary
- [AIRFLOW-1629] Make extra a textarea in edit connections form
- [AIRFLOW-1621] Add tests for server side paging
- [AIRFLOW-1519] Add server side paging in DAGs list
- [AIRFLOW-289] Make airflow timezone independent
- [AIRFLOW-1356] Add `-celery_hostname` to *airflow worker*
- [AIRFLOW-1591] Avoid attribute error when rendering logging filename
- [AIRFLOW-1031] Replace hard-code to DagRun.ID\_PREFIX
- [AIRFLOW-1604] Rename logger to log
- [AIRFLOW-1512] Add PythonVirtualenvOperator
- [AIRFLOW-1617] Fix XSS vulnerability in Variable endpoint

- [AIRFLOW-1497] Reset hidden fields when changing connection type
- [AIRFLOW-1177] Fix Variable.setdefault w/existing JSON
- [AIRFLOW-1600] Fix exception handling in get\_fernet
- [AIRFLOW-1614] Replace inspect.stack() with sys.\_getframe()
- [AIRFLOW-1613] Make MySqlToGoogleCloudStorageOperator compaitible with python3
- [AIRFLOW-1609] Fix gitignore to ignore all venvs
- [AIRFLOW-1601] Add configurable task cleanup time
- [AIRFLOW-XXX] Bumping Airflow 1.9.0alpha0 version
- [AIRFLOW-1608] Handle pending job state in GCP Dataflow hook
- [AIRFLOW-1606] Use non static DAG.sync\_to\_db
- [AIRFLOW-1606][Airflow-1606][AIRFLOW-1605][AIRFLOW-160] DAG.sync\_to\_db is now a normal method
- [AIRFLOW-1602] LoggingMixin in DAG class
- [AIRFLOW-1593] expose load\_string in WasbHook
- [AIRFLOW-1597] Add GameWisp as Airflow user
- [AIRFLOW-1594] Don't install test packages into python root.[]
- [AIRFLOW-1582] Improve logging within Airflow
- [AIRFLOW-1476] add INSTALL instruction for source releases
- [AIRFLOW-XXX] Save username and password in airflow-pr
- [AIRFLOW-1522] Increase text size for var field in variables for MySQL
- [AIRFLOW-950] Missing AWS integrations on documentation::integrations
- [AIRFLOW-XXX] 1.8.2 release notes
- [AIRFLOW-1573] Remove *thrift* < 0.10.0 requirement
- [AIRFLOW-1584] Remove insecure /headers endpoint
- [AIRFLOW-1586] Add mapping for date type to mysql\_to\_gcs operator
- [AIRFLOW-1579] Adds support for jagged rows in Bigquery hook for BQ load jobs
- [AIRFLOW-1577] Add token support to DatabricksHook
- [AIRFLOW-1580] Error in string formating
- [AIRFLOW-1567] Updated docs for Google ML Engine operators/hooks
- [AIRFLOW-1574] add 'to' attribute to templated vars of email operator
- [AIRFLOW-1572] add carbonite to company list
- [AIRFLOW-1568] Fix typo in BigQueryHook
- [AIRFLOW-1493][AIRFLOW-XXXX][WIP] fixed dumb thing
- [AIRFLOW-1567][Airflow-1567] Renamed cloudml hook and operator to mlengine
- [AIRFLOW-1568] Add datastore export/import operators
- [AIRFLOW-1564] Use Jinja2 to render logging filename

- [AIRFLOW-1562] Spark-sql logging contains deadlock
- [AIRFLOW-1556][Airflow 1556] Add support for SQL parameters in BigQueryBaseCursor
- [AIRFLOW-108] Add CreditCards.com to companies list
- [AIRFLOW-1541] Add channel to template fields of slack\_operator
- [AIRFLOW-1535] Add service account/scopes in dataproc
- [AIRFLOW-1384] Add to README.md CaDC/ARGO
- [AIRFLOW-1546] add Zymergen 80to org list in README
- [AIRFLOW-1545] Add Nextdoor to companies list
- [AIRFLOW-1544] Add DataFox to companies list
- [AIRFLOW-1529] Add logic supporting quoted newlines in Google BigQuery load jobs
- [AIRFLOW-1521] Fix emplate rendering for BigqueryTableDeleteOperator
- [AIRFLOW-1324] Generalize Druid operator and hook
- [AIRFLOW-1516] Fix error handling getting fernet
- [AIRFLOW-1420][AIRFLOW-1473] Fix deadlock check
- [AIRFLOW-1495] Fix migration on index on job\_id
- [AIRFLOW-1483] Making page size consistent in list
- [AIRFLOW-1495] Add TaskInstance index on job\_id
- [AIRFLOW-855] Replace PickleType with LargeBinary in XCom
- [AIRFLOW-1505] Document when Jinja substitution occurs
- [AIRFLOW-1504] Log dataproc cluster name
- [AIRFLOW-1239] Fix unicode error for logs in base\_task\_runner
- [AIRFLOW-1280] Fix Gantt chart height
- [AIRFLOW-1507] Template parameters in file\_to\_gcs operator
- [AIRFLOW-1452] workaround lock on method
- [AIRFLOW-1385] Make Airflow task logging configurable
- [AIRFLOW-940] Handle error on variable decrypt
- [AIRFLOW-1492] Add gauge for task successes/failures
- [AIRFLOW-1443] Update Airflow configuration documentation
- [AIRFLOW-1486] Unexpected S3 writing log error
- [AIRFLOW-1487] Added links to all companies officially using Airflow
- [AIRFLOW-1489] Fix typo in BigQueryCheckOperator
- [AIRFLOW-1349] Fix backfill to respect limits
- [AIRFLOW-1478] Chart owner column should be sortable
- [AIRFLOW-1397][AIRFLOW-1] No Last Run column data dispalyed in Airflow UI 1.8.1
- [AIRFLOW-1474] Add dag\_id regex feature for *airflow clear* command
- [AIRFLOW-1445] Changing HivePartitionSensor UI color to lighter shade

- [AIRFLOW-1359] Use default\_args in Cloud ML eval
- [AIRFLOW-1389] Support createDisposition in BigQueryOperator
- [AIRFLOW-1349] Refactor BackfillJob \_execute
- [AIRFLOW-1459] Fixed broken integration .rst formatting
- [AIRFLOW-1448] Revert “Fix cli reading logfile in memory”
- [AIRFLOW-1398] Allow ExternalTaskSensor to wait on multiple runs of a task
- [AIRFLOW-1399] Fix cli reading logfile in memory
- [AIRFLOW-1442] Remove extra space from ignore\_all\_deps generated command
- [AIRFLOW-1438] Change batch size per query in scheduler
- [AIRFLOW-1439] Add max billing tier for the BQ Hook and Operator
- [AIRFLOW-1437] Modify BigQueryTableDeleteOperator
- [Airflow 1332] Split logs based on try number
- [AIRFLOW-1385] Create abstraction for Airflow task logging
- [AIRFLOW-756][AIRFLOW-751] Replace ssh hook, operator & sftp operator with paramiko based
- [AIRFLOW-1393][AIRFLOW-1393] Enable Py3 tests in contrib/spark\_submit\_hook[
- [AIRFLOW-1345] Dont expire TIs on each scheduler loop
- [AIRFLOW-1059] Reset orphaned tasks in batch for scheduler
- [AIRFLOW-1255] Fix SparkSubmitHook output deadlock
- [AIRFLOW-1359] Add Google CloudML utils for model evaluation
- [AIRFLOW-1247] Fix ignore all dependencies argument ignored
- [AIRFLOW-1401] Standardize cloud ml operator arguments
- [AIRFLOW-1394] Add quote\_character param to GCS hook and operator
- [AIRFLOW-1402] Cleanup SafeConfigParser DeprecationWarning
- [AIRFLOW-1326][AIRFLOW-1326][AIRFLOW-1184] Don’t split argument array – it’s already an array.[
- [AIRFLOW-1384] Add ARGO/CaDC as a Airflow user
- [AIRFLOW-1357] Fix scheduler zip file support
- [AIRFLOW-1382] Add working dir option to DockerOperator
- [AIRFLOW-1388] Add Cloud ML Engine operators to integration doc
- [AIRFLOW-1387] Add unicode string prefix
- [AIRFLOW-1366] Add max\_tries to task instance
- [AIRFLOW-1300] Enable table creation with TBLPROPERTIES
- [AIRFLOW-1271] Add Google CloudML Training Operator
- [AIRFLOW-300] Add Google Pubsub hook and operator
- [AIRFLOW-1343] Fix dataproc label format
- [AIRFLOW-1367] Pass Content-ID To reference inline images in an email, we need to be able to add <img src=”cid: { }”/> to the HTML. However currently the Content-ID (cid) is not passed, so we need to add it

- [AIRFLOW-1265] Fix celery executor parsing CELERY\_SSL\_ACTIVE
- [AIRFLOW-1272] Google Cloud ML Batch Prediction Operator
- [AIRFLOW-1352][AIRFLOW-1335] Revert MemoryHandler change ()[]
- [AIRFLOW-1350] Add query\_uri param to Hive/SparkSQL DataProc operator
- [AIRFLOW-1334] Check if tasks are backfill on scheduler in a join
- [AIRFLOW-1343] Add Airflow default label to the dataproc operator
- [AIRFLOW-1273] Add Google Cloud ML version and model operators
- [AIRFLOW-1273]AIRFLOW-1273] Add Google Cloud ML version and model operators
- [AIRFLOW-1321] Fix hidden field key to ignore case
- [AIRFLOW-1337] Make log\_format key names lowercase
- [AIRFLOW-1338][AIRFLOW-782] Add GCP dataflow hook runner change to UPDATING.md
- [AIRFLOW-801] Remove outdated docstring on BaseOperator
- [AIRFLOW-1344] Fix text encoding bug when reading logs for Python 3.5
- [AIRFLOW-1338] Fix incompatible GCP dataflow hook
- [AIRFLOW-1333] Enable copy function for Google Cloud Storage Hook
- [AIRFLOW-1337] Allow log format customization via airflow.cfg
- [AIRFLOW-1320] Update LetsBonus users in README
- [AIRFLOW-1335] Use MemoryHandler for buffered logging
- [AIRFLOW-1339] Add Drivy to the list of users
- [AIRFLOW-1275] Put 'airflow pool' into API
- [AIRFLOW-1296] Propagate SKIPPED to all downstream tasks
- [AIRFLOW-1317] Fix minor issues in API reference
- [AIRFLOW-1308] Disable nanny usage for Dask
- [AIRFLOW-1172] Support nth weekday of the month cron expression
- [AIRFLOW-936] Add clear/mark success for DAG in the UI
- [AIRFLOW-1294] Backfills can loose tasks to execute
- [AIRFLOW-1299] Support imageVersion in Google Dataproc cluster
- [AIRFLOW-1291] Update NOTICE and LICENSE files to match ASF requirements
- [AIRFLOW-1301] Add New Relic to list of companies
- [AIRFLOW-1289] Removes restriction on number of scheduler threads
- [AIRFLOW-1024] Ignore celery executor errors (#49)
- [AIRFLOW-1265] Fix exception while loading celery configurations
- [AIRFLOW-1290] set docs author to 'Apache Airflow'
- [AIRFLOW-1242] Allowing project\_id to have a colon in it.
- [AIRFLOW-1282] Fix known event column sorting
- [AIRFLOW-1166] Speed up \_change\_state\_for\_tis\_without\_dagrun

- [AIRFLOW-1208] Speed-up cli tests
- [AIRFLOW-1192] Some enhancements to qubole\_operator
- [AIRFLOW-1281] Sort variables by key field by default
- [AIRFLOW-1277] Forbid KE creation with empty fields
- [AIRFLOW-1276] Forbid event creation with end\_data earlier than start\_date
- [AIRFLOW-1263] Dynamic height for charts
- [AIRFLOW-1266] Increase width of gantt y axis
- [AIRFLOW-1244] Forbid creation of a pool with empty name
- [AIRFLOW-1274][HTTPSSENSOR] Rename parameter params to data
- [AIRFLOW-654] Add SSL Config Option for CeleryExecutor w/ RabbitMQ - Add BROKER\_USE\_SSL config to give option to send AMQP messages over SSL - Can be set using usual airflow options (e.g. airflow.cfg, env vars, etc.)
- [AIRFLOW-1256] Add United Airlines to readme
- [AIRFLOW-1251] Add eRevalue to Airflow users
- [AIRFLOW-908] Print hostname at the start of cli run
- [AIRFLOW-1237] Fix IN-predicate sqlalchemy warning
- [AIRFLOW-1243] DAGs table has no default entries to show
- [AIRFLOW-1245] Fix random failure in test\_trigger\_dag\_for\_date
- [AIRFLOW-1248] Fix wrong conf name for worker timeout
- [AIRFLOW-1197] : SparkSubmitHook on\_kill error
- [AIRFLOW-1191] : SparkSubmitHook custom cmd
- [AIRFLOW-1234] Cover utils.operator\_helpers with UTs
- [AIRFLOW-1217] Enable Sqoop logging
- [AIRFLOW-645] Support HTTPS connections in HttpHook
- [AIRFLOW-1231] Use flask\_wtf.CSRFProtect
- [AIRFLOW-1232] Remove deprecated readfp warning
- [AIRFLOW-1233] Cover utils.json with unit tests
- [AIRFLOW-1227] Remove empty column on the Logs view
- [AIRFLOW-1226] Remove empty column on the Jobs view
- [AIRFLOW-1221] Fix templating bug with DatabricksSubmitRunOperator
- [AIRFLOW-1210] Enable DbApiHook unit tests
- [AIRFLOW-1199] Fix create modal
- [AIRFLOW-1200] Forbid creation of a variable with an empty key
- [AIRFLOW-1207] Enable utils.helpers unit tests
- [AIRFLOW-1213] Add hcatalog parameters to sqoop
- [AIRFLOW-1201] Update deprecated 'nose-parameterized'
- [AIRFLOW-1186] Sort dag.get\_task\_instances by execution\_date

- [AIRFLOW-1203] Pin Google API client version to fix OAuth issue
- [AIRFLOW-1145] Fix `closest_date_partition` function with `before` set to `True` If we're looking for the closest date before, we should take the latest date in the list of date before.
- [AIRFLOW-1180] Fix flask-wtf version for `test_csrf_rejection`
- [AIRFLOW-993] Update date inference logic
- [AIRFLOW-1170] `DbApiHook` `insert_rows` inserts parameters separately
- [AIRFLOW-1041] Do not shadow `xcom_push` method[]
- [AIRFLOW-860][AIRFLOW-935] Fix plugin executor import cycle and executor selection
- [AIRFLOW-1189] Fix get a `DataFrame` using `BigQueryHook` failing
- [AIRFLOW-1184] `SparkSubmitHook` does not split args
- [AIRFLOW-1182] `SparkSubmitOperator` template field
- [AIRFLOW-823] Allow specifying execution date in `task_info` API
- [AIRFLOW-1175] Add Pronto Tools to Airflow user list
- [AIRFLOW-1150] Fix scripts execution in `sparksql` hook[]
- [AIRFLOW-1141] remove `crawl_for_tasks`
- [AIRFLOW-1193] Add Checkr to company using Airflow
- [AIRFLOW-1168] Add `closing()` to all connections and cursors
- [AIRFLOW-1188] Add `max_bad_records` param to `GoogleCloudStorageToBigQueryOperator`
- [AIRFLOW-1187][AIRFLOW-1185] Fix PyPi package names in documents
- [AIRFLOW-1185] Fix PyPi URL in templates
- [AIRFLOW-XXX] Updating CHANGELOG, README, and UPDATING after 1.8.1 release
- [AIRFLOW-1181] Add delete and list functionality to `gcs_hook`
- [AIRFLOW-1179] Fix Pandas 0.2x breaking Google BigQuery change
- [AIRFLOW-1167] Support microseconds in `FTPHook` modification time
- [AIRFLOW-1173] Add Robinhood to who uses Airflow
- [AIRFLOW-945][AIRFLOW-941] Remove `psycpg2` connection workaround
- [AIRFLOW-1140] `DatabricksSubmitRunOperator` should template the "json" field.
- [AIRFLOW-1160] Update Spark parameters for Mesos
- [AIRFLOW 1149][AIRFLOW-1149] Allow for custom filters in Jinja2 templates
- [AIRFLOW-1036] Randomize exponential backoff
- [AIRFLOW-1155] Add Tails.com to community
- [AIRFLOW-1142] Do not reset orphaned state for backfills
- [AIRFLOW-492] Make sure stat updates cannot fail a task
- [AIRFLOW-1119] Fix unload query so headers are on first row[]
- [AIRFLOW-1089] Add Spark application arguments
- [AIRFLOW-1125] Document encrypted connections

- [AIRFLOW-1122] Increase stroke width in UI
- [AIRFLOW-1138] Add missing licenses to files in scripts directory
- (AIRFLOW-11-38) [AIRFLOW-1136] Capture invalid arguments for Sqoop
- [AIRFLOW-1127] Move license notices to LICENSE
- [AIRFLOW-1118] Add evo.company to Airflow users
- [AIRFLOW-1121][AIRFLOW-1004] Fix *airflow webserver -pid* to write out pid file
- [AIRFLOW-1124] Do not set all tasks to scheduled in backfill
- [AIRFLOW-1120] Update version view to include Apache prefix
- [AIRFLOW-1091] Add script that can compare jira target against merges
- [AIRFLOW-1107] Add support for ftps non-default port
- [AIRFLOW-1000] Rebrand distribution to Apache Airflow
- [AIRFLOW-1094] Run unit tests under contrib in Travis
- [AIRFLOW-1112] Log which pool when pool is full in scheduler
- [AIRFLOW-1106] Add Groupalia/Letsbonus to the ReadMe
- [AIRFLOW-1109] Use kill signal to kill processes and log results
- [AIRFLOW-1074] Don't count queued tasks for concurrency limits
- [AIRFLOW-1095] Make ldap\_auth memberOf come from configuration
- [AIRFLOW-1090] Add HBO
- [AIRFLOW-1035] Use binary exponential backoff
- [AIRFLOW-1081] Improve performance of duration chart
- [AIRFLOW-1078] Fix latest\_runs endpoint for old flask versions
- [AIRFLOW-1085] Enhance the SparkSubmitOperator
- [AIRFLOW-1050] Do not count up\_for\_retry as not ready
- [AIRFLOW-1028] Databricks Operator for Airflow
- [AIRFLOW-1075] Security docs cleanup
- [AIRFLOW-1033][AIFRLOW-1033] Fix ti\_deps for no schedule dags
- [AIRFLOW-1016] Allow HTTP HEAD request method on HTTPSensor
- [AIRFLOW-970] Load latest\_runs on homepage async
- [AIRFLOW-111] Include queued tasks in scheduler concurrency check
- [AIRFLOW-1001] Fix landing times if there is no following schedule
- [AIRFLOW-1065] Add functionality for Azure Blob Storage over wasb://
- [AIRFLOW-947] Improve exceptions for unavailable Presto cluster
- [AIRFLOW-1067] use example.com in examples
- [AIRFLOW-1064] Change default sort to job\_id for TaskInstanceModelView
- [AIRFLOW-1030][AIRFLOW-1] Fix hook import for HttpSensor
- [AIRFLOW-1051] Add a test for resetdb to CliTests



- [AIRFLOW-1004][AIRFLOW-276] Fix *airflow webserver -D* to run in background
- [AIRFLOW-1062] Fix DagRun#find to return correct result
- [AIRFLOW-1011] Fix bug in BackfillJob.\_execute() for SubDAGs
- [AIRFLOW-1038] Specify celery serialization options explicitly
- [AIRFLOW-1054] Fix broken import in test\_dag
- [AIRFLOW-1007] Use Jinja sandbox for chart\_data endpoint
- [AIRFLOW-719] Fix race condition in ShortCircuit, Branch and LatestOnly
- [AIRFLOW-1043] Fix doc strings of operators
- [AIRFLOW-840] Make ticket renewer python3 compatible
- [AIRFLOW-985] Extend the sqoop operator and hook
- [AIRFLOW-1034] Make it possible to connect to S3 in sigv4 regions
- [AIRFLOW-1045] Make log level configurable via airflow.cfg
- [AIRFLOW-1047] Sanitize strings passed to Markup
- [AIRFLOW-1040] Fix some small typos in comments and docstrings
- [AIRFLOW-1017] get\_task\_instance shouldn't throw exception when no TI
- [AIRFLOW-1006] Add config\_templates to MANIFEST
- [AIRFLOW-999] Add support for Redis database
- [AIRFLOW-1009] Remove SQLOperator from Concepts page
- [AIRFLOW-1006] Move config templates to separate files
- [AIRFLOW-1005] Improve Airflow startup time
- [AIRFLOW-1010] Add convenience script for signing releases
- [AIRFLOW-995] Remove reference to actual Airflow issue
- [AIRFLOW-681] homepage doc link should pointing to apache repo not airbnb repo
- [AIRFLOW-705][AIRFLOW-706] Fix run\_command bugs
- [AIRFLOW-990] Fix Py27 unicode logging in DockerOperator
- [AIRFLOW-963] Fix non-rendered code examples
- [AIRFLOW-969] Catch bad python\_callable argument
- [AIRFLOW-984] Enable subclassing of SubDagOperator
- [AIRFLOW-997] Update setup.cfg to point to Apache
- [AIRFLOW-994] Add MiNODES to the official airflow user list
- [AIRFLOW-995][AIRFLOW-1] Update GitHub PR Template
- [AIRFLOW-989] Do not mark dag run successful if unfinished tasks
- [AIRFLOW-903] New configuration setting for the default dag view
- [AIRFLOW-979] Add GovTech GDS
- [AIRFLOW-933] Replace eval with literal\_eval to prevent RCE
- [AIRFLOW-974] Fix makedirs race condition

- [AIRFLOW-917] Fix formatting of error message
- [AIRFLOW-770] Refactor BaseHook so env vars are always read
- [AIRFLOW-900] Double trigger should not kill original task instance
- [AIRFLOW-900] Fixes bugs in LocalTaskJob for double run protection
- [AIRFLOW-932][AIRFLOW-932][AIRFLOW-921][AIRFLOW-910] Do not mark tasks removed when back-filling[
- [AIRFLOW-961] run onkill when SIGTERM
- [AIRFLOW-910] Use parallel task execution for backfills
- [AIRFLOW-967] Wrap strings in native for py2 ldap compatibility
- [AIRFLOW-958] Improve tooltip readability
- AIRFLOW-959 Cleanup and reorganize .gitignore
- AIRFLOW-960 Add .editorconfig file
- [AIRFLOW-931] Do not set QUEUED in TaskInstances
- [AIRFLOW-956] Get docs working on readthedocs.org
- [AIRFLOW-954] Fix configparser ImportError
- [AIRFLOW-941] Use defined parameters for pycpg2
- [AIRFLOW-943] Update Digital First Media in users list
- [AIRFLOW-942] Add mytaxi to Airflow users
- [AIRFLOW-939] add .swp to gitignore
- [AIRFLOW-719] Prevent DAGs from ending prematurely
- [AIRFLOW-938] Use test for True in task\_stats queries
- [AIRFLOW-937] Improve performance of task\_stats
- [AIRFLOW-933] use ast.literal\_eval rather eval because ast.literal\_eval does not execute input.
- [AIRFLOW-925] Revert airflow.hooks change that cherry-pick picked
- [AIRFLOW-919] Running tasks with no start date shouldn't break a DAGs UI
- [AIRFLOW-802][AIRFLOW-1] Add spark-submit operator/hook
- [AIRFLOW-725] Use keyring to store credentials for JIRA
- [AIRFLOW-916] Remove deprecated readfp function
- [AIRFLOW-911] Add coloring and timing to tests
- [AIRFLOW-906] Update Code icon from lightning bolt to file
- [AIRFLOW-897] Prevent dagruns from failing with unfinished tasks
- [AIRFLOW-896] Remove unicode to 8-bit conversion in BigQueryOperator
- [AIRFLOW-899] Tasks in SCHEDULED state should be white in the UI instead of black
- [AIRFLOW-895] Address Apache release incomplicancies
- [AIRFLOW-893][AIRFLOW-510] Fix crashing webserver when a dagrun has no start date
- [AIRFLOW-880] Make webserver serve logs in a sane way for remote logs

- [AIRFLOW-889] Fix minor error in the docstrings for BaseOperator
- [AIRFLOW-809][AIRFLOW-1] Use `__eq__` ColumnOperator When Testing Booleans
- [AIRFLOW-875] Add template to HttpSensor params
- [AIRFLOW-866] Add FTPSensor
- [AIRFLOW-881] Check if SubDagOperator is in DAG context manager
- [AIRFLOW-885] Add change.org to the users list
- [AIRFLOW-836] Use POST and CSRF for state changing endpoints
- [AIRFLOW-862] Fix Unit Tests for DaskExecutor
- [AIRFLOW-887] Support future v0.16
- [AIRFLOW-886] Pass result to `post_execute()` hook
- [AIRFLOW-871] change `logging.warn()` into `warning()`
- [AIRFLOW-882] Remove unnecessary `dag>>op` assignment in docs
- [AIRFLOW-861] make `pickle_info` endpoint be `login_required`
- [AIRFLOW-869] Refactor mark success functionality
- [AIRFLOW-877] Remove `.sql` template extension from GCS download operator
- [AIRFLOW-826] Add Zendesk hook
- [AIRFLOW-842] do not query the DB with an empty IN clause
- [AIRFLOW-834] change `raise StopIteration` into `return`
- [AIRFLOW-832] Let debug server run without SSL
- [AIRFLOW-862] Add DaskExecutor
- [AIRFLOW-858] Configurable database name for DB operators
- [AIRFLOW-863] Example DAGs should have recent start dates
- [AIRFLOW-853] use utf8 encoding for stdout line decode
- [AIRFLOW-857] Use library assert statements instead of conditionals
- [AIRFLOW-856] Make sure execution date is set for local client
- [AIRFLOW-854] Add OKI as Airflow user
- [AIRFLOW-830][AIRFLOW-829][AIRFLOW-88] Reduce Travis log verbosity
- [AIRFLOW-814] Fix `Presto*CheckOperator.__init__`
- [AIRFLOW-793] Enable compressed loading in `S3ToHiveTransfer`
- [AIRFLOW-844] Fix `cgroups` directory creation
- [AIRFLOW-831] Restore import to fix broken tests
- [AIRFLOW-794] Access `DAGS_FOLDER` and `SQL_ALCHEMY_CONN` exclusively from settings
- [AIRFLOW-694] Fix config behaviour for empty envvar
- [AIRFLOW-365] Set `dag.fileloc` explicitly and use for Code view
- [AIRFLOW-781] Allow DataFlowOperators to accept jobs stored in GCS

### 3.19.5 Airflow 1.8.2, 2017-09-04

- [AIRFLOW-809][AIRFLOW-1] Use `__eq__` ColumnOperator When Testing Booleans
- [AIRFLOW-1296] Propagate SKIPPED to all downstream tasks
- Re-enable caching for hadoop components
- Pin Hive and Hadoop to a specific version and create writable warehouse dir
- [AIRFLOW-1308] Disable nanny usage for Dask
- Updating CHANGELOG for 1.8.2rc1
- [AIRFLOW-1294] Backfills can loose tasks to execute
- [AIRFLOW-1291] Update NOTICE and LICENSE files to match ASF requirements
- [AIRFLOW-XXX] Set version to 1.8.2rc1
- [AIRFLOW-1160] Update Spark parameters for Mesos
- [AIRFLOW 1149][AIRFLOW-1149] Allow for custom filters in Jinja2 templates
- [AIRFLOW-1119] Fix unload query so headers are on first row[]
- [AIRFLOW-1089] Add Spark application arguments
- [AIRFLOW-1078] Fix latest\_runs endpoint for old flask versions
- [AIRFLOW-1074] Don't count queued tasks for concurrency limits
- [AIRFLOW-1064] Change default sort to job\_id for TaskInstanceModelView
- [AIRFLOW-1038] Specify celery serialization options explicitly
- [AIRFLOW-1036] Randomize exponential backoff
- [AIRFLOW-993] Update date inference logic
- [AIRFLOW-1167] Support microseconds in FTPHook modification time
- [AIRFLOW-1179] Fix Pandas 0.2x breaking Google BigQuery change
- [AIRFLOW-1263] Dynamic height for charts
- [AIRFLOW-1266] Increase width of gantt y axis
- [AIRFLOW-1290] set docs author to 'Apache Airflow'
- [AIRFLOW-1282] Fix known event column sorting
- [AIRFLOW-1166] Speed up `_change_state_for_tis_without_dagrun`
- [AIRFLOW-1192] Some enhancements to `qubole_operator`
- [AIRFLOW-1281] Sort variables by key field by default
- [AIRFLOW-1244] Forbid creation of a pool with empty name
- [AIRFLOW-1243] DAGs table has no default entries to show
- [AIRFLOW-1227] Remove empty column on the Logs view
- [AIRFLOW-1226] Remove empty column on the Jobs view
- [AIRFLOW-1199] Fix create modal
- [AIRFLOW-1200] Forbid creation of a variable with an empty key
- [AIRFLOW-1186] Sort `dag.get_task_instances` by `execution_date`

- [AIRFLOW-1145] Fix `closest_date_partition` function with `before` set to `True` If we're looking for the closest date before, we should take the latest date in the list of date before.
- [AIRFLOW-1180] Fix flask-wtf version for `test_csrf_rejection`
- [AIRFLOW-1170] `DbApiHook.insert_rows` inserts parameters separately
- [AIRFLOW-1150] Fix scripts execution in `sparksql` hook[]
- [AIRFLOW-1168] Add `closing()` to all connections and cursors
- [AIRFLOW-XXX] Updating CHANGELOG, README, and UPDATING after 1.8.1 release

### 3.19.6 Airflow 1.8.1, 2017-05-09

- [AIRFLOW-1142] SubDAG Tasks Not Executed Even Though All Dependencies Met
- [AIRFLOW-1138] Add licenses to files in scripts directory
- [AIRFLOW-1127] Move license notices to `LICENSE` instead of `NOTICE`
- [AIRFLOW-1124] Do not set all task instances to scheduled on backfill
- [AIRFLOW-1120] Update version view to include Apache prefix
- [AIRFLOW-1062] `DagRun#find` returns wrong result if `external_trigger=False` is specified
- [AIRFLOW-1054] Fix broken import on `test_dag`
- [AIRFLOW-1050] Retries ignored - regression
- [AIRFLOW-1033] `TypeError: can't compare datetime.datetime to None`
- [AIRFLOW-1017] `get_task_instance` should return `None` instead of throw an exception for non-existent TIs
- [AIRFLOW-1011] Fix bug in `BackfillJob._execute()` for SubDAGs
- [AIRFLOW-1004] `airflow webserver -D` runs in foreground
- [AIRFLOW-1001] Landing Time shows “unsupported operand type(s) for -: ‘datetime.datetime’ and ‘None-Type’” on `example_subdag_operator`
- [AIRFLOW-1000] Rebrand to Apache Airflow instead of Airflow
- [AIRFLOW-989] Clear Task Regression
- [AIRFLOW-974] `airflow.util.file.mkdir` has a race condition
- [AIRFLOW-906] Update Code icon from lightning bolt to file
- [AIRFLOW-858] Configurable database name for DB operators
- [AIRFLOW-853] `ssh_execute_operator.py` stdout decode default to ASCII
- [AIRFLOW-832] Fix debug server
- [AIRFLOW-817] Trigger dag fails when using CLI + API
- [AIRFLOW-816] Make sure to pull `nvd3` from local resources
- [AIRFLOW-815] Add previous/next execution dates to available default variables.
- [AIRFLOW-813] Fix unterminated unit tests in `tests.job` (`tests/job.py`)
- [AIRFLOW-812] Scheduler job terminates when there is no dag file
- [AIRFLOW-806] UI should properly ignore DAG doc when it is `None`
- [AIRFLOW-794] Consistent access to `DAGS_FOLDER` and `SQL_ALCHEMY_CONN`

- [AIRFLOW-785] ImportError if cgroupspy is not installed
- [AIRFLOW-784] Cannot install with funcsigns > 1.0.0
- [AIRFLOW-780] The UI no longer shows broken DAGs
- [AIRFLOW-777] dag\_is\_running is initialized to True instead of False
- [AIRFLOW-719] Skipped operations make DAG finish prematurely
- [AIRFLOW-694] Empty env vars do not overwrite non-empty config values
- [AIRFLOW-492] Insert into dag\_stats table results into failed task while task itself succeeded
- [AIRFLOW-139] Executing VACUUM with PostgresOperator
- [AIRFLOW-111] DAG concurrency is not honored
- [AIRFLOW-88] Improve clarity Travis CI reports

### **3.19.7 Airflow 1.8.0, 2017-03-12**

- [AIRFLOW-900] Double trigger should not kill original task instance
- [AIRFLOW-900] Fixes bugs in LocalTaskJob for double run protection
- [AIRFLOW-932] Do not mark tasks removed when backfilling
- [AIRFLOW-961] run onkill when SIGTERMd
- [AIRFLOW-910] Use parallel task execution for backfills
- [AIRFLOW-967] Wrap strings in native for py2 ldap compatibility
- [AIRFLOW-941] Use defined parameters for pycopg2
- [AIRFLOW-719] Prevent DAGs from ending prematurely
- [AIRFLOW-938] Use test for True in task\_stats queries
- [AIRFLOW-937] Improve performance of task\_stats
- [AIRFLOW-933] use ast.literal\_eval rather eval because ast.literal\_eval does not execute input.
- [AIRFLOW-925] Revert airflow.hooks change that cherry-pick picked
- [AIRFLOW-919] Running tasks with no start date shouldn't break a DAGs UI
- [AIRFLOW-802] Add spark-submit operator/hook
- [AIRFLOW-897] Prevent dagruns from failing with unfinished tasks
- [AIRFLOW-861] make pickle\_info endpoint be login\_required
- [AIRFLOW-853] use utf8 encoding for stdout line decode
- [AIRFLOW-856] Make sure execution date is set for local client
- [AIRFLOW-830][AIRFLOW-829][AIRFLOW-88] Reduce Travis log verbosity
- [AIRFLOW-831] Restore import to fix broken tests
- [AIRFLOW-794] Access DAGS\_FOLDER and SQL\_ALCHEMY\_CONN exclusively from settings
- [AIRFLOW-694] Fix config behaviour for empty envvar
- [AIRFLOW-365] Set dag.fileloc explicitly and use for Code view
- [AIRFLOW-931] Do not set QUEUED in TaskInstances

- [AIRFLOW-899] Tasks in SCHEDULED state should be white in the UI instead of black
- [AIRFLOW-895] Address Apache release incompliancies
- [AIRFLOW-893][AIRFLOW-510] Fix crashing webservers when a dagrun has no start date
- [AIRFLOW-793] Enable compressed loading in S3ToHiveTransfer
- [AIRFLOW-863] Example DAGs should have recent start dates
- [AIRFLOW-869] Refactor mark success functionality
- [AIRFLOW-856] Make sure execution date is set for local client
- [AIRFLOW-814] Fix Presto\*CheckOperator.\_\_init\_\_
- [AIRFLOW-844] Fix cgroups directory creation
- [AIRFLOW-816] Use static nvd3 and d3
- [AIRFLOW-821] Fix py3 compatibility
- [AIRFLOW-817] Check for None value of execution\_date in endpoint
- [AIRFLOW-822] Close db before exception
- [AIRFLOW-815] Add prev/next execution dates to template variables
- [AIRFLOW-813] Fix unterminated unit tests in SchedulerJobTest
- [AIRFLOW-813] Fix unterminated scheduler unit tests
- [AIRFLOW-806] UI should properly ignore DAG doc when it is None
- [AIRFLOW-812] Fix the scheduler termination bug.
- [AIRFLOW-780] Fix dag import errors no longer working
- [AIRFLOW-783] Fix py3 incompatibility in BaseTaskRunner
- [AIRFLOW-810] Correct down\_revision dag\_id/state index creation
- [AIRFLOW-807] Improve scheduler performance for large DAGs
- [AIRFLOW-798] Check return\_code before forcing termination
- [AIRFLOW-139] Let psycopg2 handle autocommit for PostgresHook
- [AIRFLOW-776] Add missing cgroups devel dependency
- [AIRFLOW-777] Fix expression to check if a DagRun is in running state
- [AIRFLOW-785] Don't import CgroupTaskRunner at global scope
- [AIRFLOW-784] Pin funcsigns to 1.0.0
- [AIRFLOW-624] Fix setup.py to not import airflow.version as version
- [AIRFLOW-779] Task should fail with specific message when deleted
- [AIRFLOW-778] Fix completey broken MetastorePartitionSensor
- [AIRFLOW-739] Set pickle\_info log to debug
- [AIRFLOW-771] Make S3 logs append instead of clobber
- [AIRFLOW-773] Fix flaky datetime addition in api test
- [AIRFLOW-219][AIRFLOW-398] Cgroups + impersonation
- [AIRFLOW-683] Add jira hook, operator and sensor

- [AIRFLOW-762] Add Google DataProc delete operator
- [AIRFLOW-760] Update systemd config
- [AIRFLOW-759] Use previous dag\_run to verify depend\_on\_past
- [AIRFLOW-757] Set child\_process\_log\_directory default more sensible
- [AIRFLOW-692] Open XCom page to super-admins only
- [AIRFLOW-737] Fix HDFS Sensor directory.
- [AIRFLOW-747] Fix retry\_delay not honoured
- [AIRFLOW-558] Add Support for dag.catchup=(True|False) Option
- [AIRFLOW-489] Allow specifying execution date in trigger\_dag API
- [AIRFLOW-738] Commit deleted xcom items before insert
- [AIRFLOW-729] Add Google Cloud Dataproc cluster creation operator
- [AIRFLOW-728] Add Google BigQuery table sensor
- [AIRFLOW-741] Log to debug instead of info for app.py
- [AIRFLOW-731] Fix period bug for NamedHivePartitionSensor
- [AIRFLOW-740] Pin jinja2 to < 2.9.0
- [AIRFLOW-663] Improve time units for task performance charts
- [AIRFLOW-665] Fix email attachments
- [AIRFLOW-734] Fix SMTP auth regression when not using user/pass
- [AIRFLOW-702] Fix LDAP Regex Bug
- [AIRFLOW-717] Add Cloud Storage updated sensor
- [AIRFLOW-695] Retries do not execute because dagrun is in FAILED state
- [AIRFLOW-673] Add operational metrics test for SchedulerJob
- [AIRFLOW-727] try\_number is not increased
- [AIRFLOW-715] A more efficient HDFS Sensor:
- [AIRFLOW-716] Allow AVRO BigQuery load-job without schema
- [AIRFLOW-718] Allow the query URI for DataProc Pig
- Log needs to be part of try/catch block
- [AIRFLOW-721] Descendant process can disappear before termination
- [AIRFLOW-403] Bash operator's kill method leaves underlying processes running
- [AIRFLOW-657] Add AutoCommit Parameter for MSSQL
- [AIRFLOW-641] Improve pull request instructions
- [AIRFLOW-685] Add test for MySqlHook.bulk\_load()
- [AIRFLOW-686] Match auth backend config section
- [AIRFLOW-691] Add SSH KeepAlive option to SSH\_hook
- [AIRFLOW-709] Use same engine for migrations and reflection
- [AIRFLOW-700] Update to reference to web authentication documentation



- [AIRFLOW-649] Support non-sched DAGs in LatestOnlyOp
- [AIRFLOW-712] Fix AIRFLOW-667 to use proper HTTP error properties
- [AIRFLOW-710] Add OneFineStay as official user
- [AIRFLOW-703][AIRFLOW-1] Stop Xcom being cleared too early
- [AIRFLOW-679] Stop concurrent task instances from running
- [AIRFLOW-704][AIRFLOW-1] Fix invalid syntax in BQ hook
- [AIRFLOW-667] Handle BigQuery 503 error
- [AIRFLOW-680] Disable connection pool for commands
- [AIRFLOW-678] Prevent scheduler from double triggering TIs
- [AIRFLOW-677] Kill task if it fails to heartbeat
- [AIRFLOW-674] Ability to add descriptions for DAGs
- [AIRFLOW-682] Bump MAX\_PERIODS to make mark\_success work for large DAGs
- Use jdk selector to set required jdk
- [AIRFLOW-647] Restore dag.get\_active\_runs
- [AIRFLOW-662] Change seasons to months in project description
- [AIRFLOW-656] Add dag/task/date index to xcom table
- [AIRFLOW-658] Improve schema\_update\_options in GCP
- [AIRFLOW-41] Fix pool oversubscription
- [AIRFLOW-489] Add API Framework
- [AIRFLOW-653] Add some missing endpoint tests
- [AIRFLOW-652] Remove obsolete endpoint
- [AIRFLOW-345] Add contrib ECSOperator
- [AIRFLOW-650] Adding Celect to user list
- [AIRFLOW-510] Filter Paused Dags, show Last Run & Trigger Dag
- [AIRFLOW-643] Improve date handling for sf\_hook
- [AIRFLOW-638] Add schema\_update\_options to GCP ops
- [AIRFLOW-640] Install and enable nose-ignore-docstring
- [AIRFLOW-639]AIRFLOW-639] Alphasort package names
- [AIRFLOW-375] Fix pylint errors
- [AIRFLOW-347] Show empty DAG runs in tree view
- [AIRFLOW-628] Adding SalesforceHook to contrib/hooks
- [AIRFLOW-514] hive hook loads data from pandas DataFrame into hive and infers types
- [AIRFLOW-565] Fixes DockerOperator on Python3.x
- [AIRFLOW-635] Encryption option for S3 hook
- [AIRFLOW-137] Fix max\_active\_runs on clearing tasks
- [AIRFLOW-343] Fix schema plumbing in HiveServer2Hook

- [AIRFLOW-130] Fix ssh operator macosx
- [AIRFLOW-633] Show TI attributes in TI view
- [AIRFLOW-626][AIRFLOW-1] HTML Content does not show up when sending email with attachment
- [AIRFLOW-533] Set autocommit via set\_autocommit
- [AIRFLOW-629] stop pinning lxml
- [AIRFLOW-464] Add setdefault method to Variable
- [AIRFLOW-626][AIRFLOW-1] HTML Content does not show up when sending email with attachment
- [AIRFLOW-591] Add datadog hook & sensor
- [AIRFLOW-561] Add RedshiftToS3Transfer operator
- [AIRFLOW-570] Pass root to date form on gantt
- [AIRFLOW-504] Store fractional seconds in MySQL tables
- [AIRFLOW-623] LDAP attributes not always a list
- [AIRFLOW-611] source\_format in BigQueryBaseCursor
- [AIRFLOW-619] Fix exception in Gannt chart
- [AIRFLOW-618] Cast DateTimes to avoid sqllite errors
- [AIRFLOW-422] Add JSON endpoint for task info
- [AIRFLOW-616][AIRFLOW-617] Minor fixes to PR tool UX
- [AIRFLOW-179] Fix DbApiHook with non-ASCII chars
- [AIRFLOW-566] Add timeout while fetching logs
- [AIRFLOW-615] Set graph glyphicon first
- [AIRFLOW-609] Add application\_name to PostgresHook
- [AIRFLOW-604] Revert .first() to .one()
- [AIRFLOW-370] Create AirflowConfigException in exceptions.py
- [AIRFLOW-582] Fixes TI.get\_dagrun filter (removes start\_date)
- [AIRFLOW-568] Fix double task\_stats count if a DagRun is active
- [AIRFLOW-585] Fix race condition in backfill execution loop
- [AIRFLOW-580] Prevent landscape warning on .format
- [AIRFLOW-597] Check if content is None, not false-equivalent
- [AIRFLOW-586] test\_dag\_v1 fails from 0 to 3 a.m.
- [AIRFLOW-453] Add XCom Admin Page
- [AIRFLOW-588] Add Google Cloud Storage Object sensor[]
- [AIRFLOW-592] example\_xcom import Error
- [AIRFLOW-587] Fix incorrect scope for Google Auth[]
- [AIRFLOW-589] Add templatable job\_name[]
- [AIRFLOW-227] Show running config in config view
- [AIRFLOW-319]AIRFLOW-319] xcom push response in HTTP Operator

- [AIRFLOW-385] Add symlink to latest scheduler log directory
- [AIRFLOW-583] Fix decode error in gcs\_to\_bq
- [AIRFLOW-96] s3\_conn\_id using environment variable
- [AIRFLOW-575] Clarify tutorial and FAQ about *schedule\_interval* always inheriting from DAG object
- [AIRFLOW-577] Output BigQuery job for improved debugging
- [AIRFLOW-560] Get URI & SQLA engine from Connection
- [AIRFLOW-518] Require DataProfilingMixin for Variables CRUD
- [AIRFLOW-553] Fix load path for filters.js
- [AIRFLOW-554] Add Jinja support to Spark-sql
- [AIRFLOW-550] Make ssl config check empty string safe
- [AIRFLOW-500] Use id for github allowed teams
- [AIRFLOW-556] Add UI PR guidelines
- [AIRFLOW-358][AIRFLOW-430] Add *connections* cli
- [AIRFLOW-548] Load DAGs immediately & continually
- [AIRFLOW-539] Updated BQ hook and BQ operator to support Standard SQL.
- [AIRFLOW-378] Add string casting to params of spark-sql operator
- [AIRFLOW-544] Add Pause/Resume toggle button
- [AIRFLOW-333][AIRFLOW-258] Fix non-module plugin components
- [AIRFLOW-542] Add tooltip to DAGs links icons
- [AIRFLOW-530] Update docs to reflect connection environment var has to be in uppercase
- [AIRFLOW-525] Update template\_fields in Qubole Op
- [AIRFLOW-480] Support binary file download from GCS
- [AIRFLOW-198] Implement latest\_only\_operator
- [AIRFLOW-91] Add SSL config option for the webserver
- [AIRFLOW-191] Fix connection leak with PostgreSQL backend
- [AIRFLOW-512] Fix 'bellow' typo in docs & comments
- [AIRFLOW-509][AIRFLOW-1] Create operator to delete tables in BigQuery
- [AIRFLOW-498] Remove hard-coded gcp project id
- [AIRFLOW-505] Support unicode characters in authors' names
- [AIRFLOW-494] Add per-operator success/failure metrics
- [AIRFLOW-488] Fix test\_simple fail
- [AIRFLOW-468] Update Panda requirement to 0.17.1
- [AIRFLOW-159] Add cloud integration section + GCP documentation
- [AIRFLOW-477][AIRFLOW-478] Restructure security section for clarity
- [AIRFLOW-467] Allow defining of project\_id in BigQueryHook
- [AIRFLOW-483] Change print to logging statement

- [AIRFLOW-475] make the segment granularity in Druid hook configurable

### **3.19.8 Airflow 1.7.2**

- [AIRFLOW-463] Link Airflow icon to landing page
- [AIRFLOW-149] Task Dependency Engine + Why Isn't My Task Running View
- [AIRFLOW-361] Add default failure handler for the Qubole Operator
- [AIRFLOW-353] Fix dag run status update failure
- [AIRFLOW-447] Store source URIs in Python 3 compatible list
- [AIRFLOW-443] Make module names unique when importing
- [AIRFLOW-444] Add Google authentication backend
- [AIRFLOW-446][AIRFLOW-445] Adds missing dataproc submit options
- [AIRFLOW-431] Add CLI for CRUD operations on pools
- [AIRFLOW-329] Update Dag Overview Page with Better Status Columns
- [AIRFLOW-360] Fix style warnings in models.py
- [AIRFLOW-425] Add white fill for null state tasks in tree view.
- [AIRFLOW-69] Use dag runs in backfill jobs
- [AIRFLOW-415] Make dag\_id not found error clearer
- [AIRFLOW-416] Use ordinals in README's company list
- [AIRFLOW-369] Allow setting default DAG orientation
- [AIRFLOW-410] Add 2 Q/A to the FAQ in the docs
- [AIRFLOW-407] Add different colors for some sensors
- [AIRFLOW-414] Improve error message for missing FERNET\_KEY
- [AIRFLOW-406] Sphinx/rst fixes
- [AIRFLOW-412] Fix lxml dependency
- [AIRFLOW-413] Fix unset path bug when backfilling via pickle
- [AIRFLOW-78] Airflow clear leaves dag\_runs
- [AIRFLOW-402] Remove NamedHivePartitionSensor static check, add docs
- [AIRFLOW-394] Add an option to the Task Duration graph to show cumulative times
- [AIRFLOW-404] Retry download if unpacking fails for hive
- [AIRFLOW-276] Unicorn rolling restart
- [AIRFLOW-399] Remove dags/testdruid.py
- [AIRFLOW-400] models.py/DAG.set\_dag\_runs\_state() does not correctly set state
- [AIRFLOW-395] Fix colon/equal signs typo for resources in default config
- [AIRFLOW-397] Documentation: Fix typo "instatiating" to "instantiating"
- [AIRFLOW-395] Remove trailing commas from resources in config
- [AIRFLOW-388] Add a new chart for Task\_Tries for each DAG

- [AIRFLOW-322] Fix typo in FAQ section
- [AIRFLOW-375] Pylint fixes
- limit scope to user email only AIRFLOW-386
- [AIRFLOW-383] Cleanup example qubole operator dag
- [AIRFLOW-160] Parse DAG files through child processes
- [AIRFLOW-381] Manual UI Dag Run creation: require dag\_id field
- [AIRFLOW-373] Enhance CLI variables functionality
- [AIRFLOW-379] Enhance Variables page functionality: import/export variables
- [AIRFLOW-331] modify the LDAP authentication config lines in 'Security' sample codes
- [AIRFLOW-356][AIRFLOW-355][AIRFLOW-354] Replace nobr, enable DAG only exists locally message, change edit DAG icon
- [AIRFLOW-362] Import `__future__` division
- [AIRFLOW-359] Pin flask-login to 0.2.11
- [AIRFLOW-261] Add bcc and cc fields to EmailOperator
- [AIRFLOW-348] Fix code style warnings
- [AIRFLOW-349] Add metric for number of zombies killed
- [AIRFLOW-340] Remove unused dependency on Babel
- [AIRFLOW-339]: Ability to pass a flower conf file
- [AIRFLOW-341][operators] Add resource requirement attributes to operators
- [AIRFLOW-335] Fix simple style errors/warnings
- [AIRFLOW-337] Add `__repr__` to VariableAccessor and VariableJsonAccessor
- [AIRFLOW-334] Fix using undefined variable
- [AIRFLOW-315] Fix blank lines code style warnings
- [AIRFLOW-306] Add Spark-sql Hook and Operator
- [AIRFLOW-327] Add rename method to the FTPHook
- [AIRFLOW-321] Fix a wrong code example about tests/dags
- [AIRFLOW-316] Always check DB state for Backfill Job execution
- [AIRFLOW-264] Adding workload management for Hive
- [AIRFLOW-297] support exponential backoff option for retry delay
- [AIRFLOW-31][AIRFLOW-200] Add note to updating.md
- [AIRFLOW-307] There is no `__neq__` python magic method.
- [AIRFLOW-309] Add requirements of develop dependencies to docs
- [AIRFLOW-307] Rename `__neq__` to `__ne__` python magic method.
- [AIRFLOW-313] Fix code style for sqoop\_hook.py
- [AIRFLOW-311] Fix wrong path in CONTRIBUTING.md
- [AIRFLOW-24] DataFlow Java Operator

- [AIRFLOW-308] Add link to refresh DAG within DAG view header
- [AIRFLOW-314] Fix BigQuery cursor run\_table\_upsert method
- [AIRFLOW-298] fix incubator disclaimer in docs
- [AIRFLOW-284] HiveServer2Hook fix for cursor scope for get\_results
- [AIRFLOW-260] More graceful exit when issues can't be closed
- [AIRFLOW-260] Handle case when no version is found
- [AIRFLOW-228] Handle empty version list in PR tool
- [AIRFLOW-302] Improve default squash commit message
- [AIRFLOW-187] Improve prompt styling
- [AIRFLOW-187] Fix typo in argument name
- [AIRFLOW-187] Move "Close XXX" message to end of squash commit
- [AIRFLOW-247] Add EMR hook, operators and sensors. Add AWS base hook
- [AIRFLOW-301] Fix broken unit test
- [AIRFLOW-100] Add execution\_date\_fn to ExternalTaskSensor
- [AIRFLOW-282] Remove PR Tool logic that depends on version formatting
- [AIRFLOW-291] Add index for state in TI table
- [AIRFLOW-269] Add some unit tests for PostgreSQL
- [AIRFLOW-296] template\_ext is being treated as a string rather than a tuple in qubole operator
- [AIRFLOW-286] Improve FTPHook to implement context manager interface
- [AIRFLOW-243] Create NamedHivePartitionSensor
- [AIRFLOW-246] Improve dag\_stats endpoint query
- [AIRFLOW-189] Highlighting of Parent/Child nodes in Graphs
- [AIRFLOW-255] Check dagrun timeout when comparing active runs
- [AIRFLOW-281] Add port to mssql\_hook
- [AIRFLOW-285] Use Airflow 2.0 style imports for all remaining hooks/operators
- [AIRFLOW-40] Add LDAP group filtering feature.
- [AIRFLOW-277] Multiple deletions does not work in Task Instances view if using SQLite backend
- [AIRFLOW-200] Make hook/operator imports lazy, and print proper exceptions
- [AIRFLOW-283] Make store\_to\_xcom\_key a templated field in GoogleCloudStorageDownloadOperator
- [AIRFLOW-278] Support utf-8 encoding for SQL
- [AIRFLOW-280] clean up tmp druid table no matter if an ingestion job succeeds or not
- [AIRFLOW-274] Add XCom functionality to GoogleCloudStorageDownloadOperator
- [AIRFLOW-273] Create an svg version of the airflow logo.
- [AIRFLOW-275] Update contributing guidelines
- [AIRFLOW-244] Modify hive operator to inject analysis data
- [AIRFLOW-162] Allow variable to be accessible into templates

- [AIRFLOW-248] Add Apache license header to all files
- [AIRFLOW-263] Remove temp backtick file
- [AIRFLOW-252] Raise Sqlite exceptions when deleting tasks instance in WebUI
- [AIRFLOW-180] Fix timeout behavior for sensors
- [AIRFLOW-262] Simplify commands in MANIFEST.in
- [AIRFLOW-31] Add zope dependency
- [AIRFLOW-6] Remove dependency on Highcharts
- [AIRFLOW-234] make task that aren't *running* self-terminate
- [AIRFLOW-256] Fix test\_scheduler\_reschedule heartbeat
- Add Python 3 compatibility fix
- [AIRFLOW-31] Use standard imports for hooks/operators
- [AIRFLOW-173] Initial implementation of FileSensor
- [AIRFLOW-224] Collect orphaned tasks and reschedule them
- [AIRFLOW-239] Fix tests indentation
- [AIRFLOW-225] Better units for task duration graph
- [AIRFLOW-241] Add testing done section to PR template
- [AIRFLOW-222] Show duration of task instances in ui
- [AIRFLOW-231] Do not eval user input in PrestoHook
- [AIRFLOW-216] Add Sqoop Hook and Operator
- [AIRFLOW-171] Add upgrade notes on email and S3 to 1.7.1.2
- [AIRFLOW-238] Make compatible with flask-admin 1.4.1
- [AIRFLOW-230] [HiveServer2Hook] adding multi statements support
- [AIRFLOW-142] setup\_env.sh doesn't download hive tarball if hdp is specified as distro
- [AIRFLOW-223] Make parametrable the IP on which Flower binds to
- [AIRFLOW-218] Added option to enable webserver gunicorn access/err logs
- [AIRFLOW-213] Add "Closes #X" phrase to commit messages
- [AIRFLOW-68] Align start\_date with the schedule\_interval
- [AIRFLOW-9] Improving docs to meet Apache's standards
- [AIRFLOW-131] Make XCom.clear more selective
- [AIRFLOW-214] Fix occasion of detached taskinstance
- [AIRFLOW-206] Add commit to close PR
- [AIRFLOW-206] Always load local log files if they exist
- [AIRFLOW-211] Fix JIRA "resolve" vs "close" behavior
- [AIRFLOW-64] Add note about relative DAGS\_FOLDER
- [AIRFLOW-114] Sort plugins dropdown
- [AIRFLOW-209] Add scheduler tests and improve lineage handling

- [AIRFLOW-207] Improve JIRA auth workflow
- [AIRFLOW-187] Improve PR tool UX
- [AIRFLOW-155] Documentation of Qubole Operator
- Optimize and refactor process\_dag
- [AIRFLOW-185] Handle empty versions list
- [AIRFLOW-201] Fix for HiveMetastoreHook + kerberos
- [AIRFLOW-202]: Fixes stray print line
- [AIRFLOW-196] Fix bug that exception is not handled in HttpSensor
- [AIRFLOW-195] : Add toggle support to subdag clearing in the CLI
- [AIRFLOW-23] Support for Google Cloud DataProc
- [AIRFLOW-25] Configuration for Celery always required
- [AIRFLOW-190] Add codecov and remove download count
- [AIRFLOW-168] Correct evaluation of @once schedule
- [AIRFLOW-183] Fetch log from remote when worker returns 4xx/5xx response
- [AIRFLOW-181] Fix failing unpacking of hadoop by redownloading
- [AIRFLOW-176] remove unused formatting key
- [AIRFLOW-167]: Add dag\_state option in cli
- [AIRFLOW-178] Fix bug so that zip file is detected in DAG folder
- [AIRFLOW-176] Improve PR Tool JIRA workflow
- AIRFLOW-45: Support Hidden Airflow Variables
- [AIRFLOW-175] Run git-reset before checkout in PR tool
- [AIRFLOW-157] Make PR tool Py3-compatible; add JIRA command
- [AIRFLOW-170] Add missing @apply\_defaults

### **3.19.9 Airflow 1.7.1, 2016-05-19**

- Fix : Don't treat premature tasks as could\_not\_run tasks
- AIRFLOW-92 Avoid unneeded upstream\_failed session closes apache/airflow#1485
- Add logic to lock DB and avoid race condition
- Handle queued tasks from multiple jobs/executors
- AIRFLOW-52 Warn about overwriting tasks in a DAG
- Fix corner case with joining processes/queues (#1473)
- [AIRFLOW-52] Fix bottlenecks when working with many tasks
- Add columns to toggle extra detail in the connection list view.
- Log the number of errors when importing DAGs
- Log dagbag metrics duplicate messages in queue into Statsd (#1406)
- Clean up issue template (#1419)



- correct missed `arg.foreground` to `arg.daemon` in cli
- Reinstate imports for github enterprise auth
- Use `os.execvp` instead of `subprocess.Popen` for the webserver
- Revert from using “`--foreground`” to “`--daemon`”
- Implement a Cloudant hook
- Add missing args to *airflow clear*
- Fixed a bug in the scheduler: `num_runs` used where `runs` intended
- Add multiprocessing support to the scheduler
- Partial fix to make sure `next_run_date` cannot be `None`
- Support list/get/set variables in the CLI
- Properly handle BigQuery booleans in BigQuery hook.
- Added the ability to view XCom variables in webserver
- Change DAG.tasks from a list to a dict
- Add support for zipped dags
- Stop creating hook on instantiating of S3 operator
- User subquery in views to find running DAGs
- Prevent DAGs from being reloaded on every scheduler iteration
- Add a missing word to docs
- Document the parameters of *DbApiHook*
- added oracle operator with existing oracle hook
- Add PyOpenSSL to Google cloud `gcp_api`.
- Remove executor error unit test
- Add DAG inference, deferral, and context manager
- Don't return error when writing files to Google cloud storage.
- Fix GCS logging for `gcp_api`.
- Ensure `attr` is in scope for error message
- Fixing misnamed `PULL_REQUEST_TEMPLATE`
- Extract `non_pooled_task_slot_count` into a configuration param
- Update `plugins.rst` for clarity on the example (#1309)
- Fix s3 logging issue
- Add twitter feed example dag
- Github `ISSUE_TEMPLATE` & `PR_TEMPLATE` cleanup
- Reduce logger verbosity
- Adding a PR Template
- Add Lucid to list of users
- Fix usage of `asciiart`

- Use session instead of outdated main\_session for are\_dependencies\_met
- Fix celery flower port allocation
- Fix for missing edit actions due to flask-admin upgrade
- Fix typo in comment in prioritize\_queued method
- Add HipchatOperator
- Include all example dags in backfill unit test
- Make sure skipped jobs are actually skipped
- Fixing a broken example dag, example\_skip\_dag.py
- Add consistent and thorough signal handling and logging
- Allow Operators to specify SKIPPED status internally
- Update docstring for executor trap unit test
- Doc: explain the usage of Jinja templating for templated params
- Don't schedule runs before the DAG's start\_date
- Fix infinite retries with pools, with test
- Fix handling of deadlocked jobs
- Show only Airflow's deprecation warnings
- Set DAG\_FOLDER for unit tests
- Missing comma in setup.py
- Deprecate \*args and \*\*kwargs in BaseOperator
- Raise deep scheduler exceptions to force a process restart.
- Change inconsistent example DAG owners
- Fix module path of send\_email\_smtp in configuration
- added Gentner Lab to list of users
- Increase timeout time for unit test
- Fix reading strings from conf
- CHORE - Remove Trailing Spaces
- Fix SSHExecuteOperator crash when using a custom ssh port
- Add note about airflow components to template
- Rewrite BackfillJob logic for clarity
- Add unit tests
- Fix miscellaneous bugs and clean up code
- Fix logic for determining DagRun states
- Make SchedulerJob not run EVERY queued task
- Improve BackfillJob handling of queued/deadlocked tasks
- Introduce ignore\_depends\_on\_past parameters
- Use Popen with CeleryExecutor

- Rename user table to users to avoid conflict with postgres
- Beware of negative pool slots.
- Add support for calling `_format` from boto to S3\_Hook
- Add pypi meta data and sync version number
- Set `dags_are_paused_at_creation`'s default value to True
- Resurface S3Log class eaten by rebase/push -f
- Add missing `session.commit()` at end of `initdb`
- Validate that subdag tasks have pool slots available, and test
- Use `urlparse` for remote GCS logs, and add unit tests
- Make webserver worker timeout configurable
- Fixed scheduling for `@once` interval
- Use `psycpg2`'s API for serializing postgres cell values
- Make the `provide_session` decorator more robust
- update link to Lyft's website
- use `num_shards` instead of `partitions` to be consistent with batch ingestion
- Add documentation links to README
- Update docs with separate configuration section
- Fix `airflow.utils` deprecation warning code being Python 3 incompatible
- Extract `dbapi` cell serialization into its own method
- Set Postgres `autocommit` as supported only if server version is `< 7.4`
- Use refactored `utils` module in unit test imports
- Add changelog for 1.7.0
- Use `LocalExecutor` on Travis if possible
- remove unused `logging.errno`, `MiniHiveCluster` imports
- remove extra import of `logging` lib
- Fix required `gcloud` version
- Refactoring `utils` into smaller submodules
- Properly measure number of task retry attempts
- Add function to get configuration as dict, plus unit tests
- Merge branch 'master' into `hivemeta_sasl`
- Add wiki link to README.md
- [hotfix] make `email.Utils > email.utils` for py3
- Add the missing "Date" header to the warning e-mails
- Add the missing "Date" header to the warning e-mails
- Check name of `SubDag` class instead of class itself
- [hotfix] removing `repo_token` from `.coveralls.yml`

- Set the service\_name in coverals.yml
- Fixes #1223
- Update Airflow docs for remote logging
- Add unit tests for trapping Executor errors
- Make sure Executors properly trap errors
- Fix HttpOpSensorTest to use fake resquest session
- Linting
- Add an example on pool usage in the documentation
- Add two methods to bigquery hook's base cursor: run\_table\_upsert, which adds a table or updates an existing table; and run\_grant\_dataset\_view\_access, which grants view access to a given dataset for a given table.
- Tasks references upstream and downstream tasks using strings instead of references
- Fix typos in models.py
- Fix broken links in documentation
- [hotfix] fixing the Scheduler CLI to make dag\_id optional
- Update link to Common Pitfalls wiki page in README
- Allow disabling periodic committing when inserting rows with DbApiHook
- added Glassdoor to "who uses airflow"
- Fix typo preventing from launching webserver
- Documentation badge
- Fixing ISSUE\_TEMPLATE name to include .md suffix
- Adding an ISSUE\_TEMPLATE to ensure that issues are adequately defined
- Linting & debugging
- Refactoring the CLI to be data-driven
- Updating the Bug Reporting protocol in the Contributing.md file
- Fixing the docs
- clean up references to old session
- remove session reference
- resolve conflict
- clear xcom data when task instance starts
- replace main\_session with @provide\_session
- Add extras to installation.rst
- Changes to Contributing to reflect more closely the current state of development.
- Modifying README to link to the wiki committer list
- docs: fixes a spelling mistake in default config
- Set killMode to 'control-group' for webservice.service
- Set KillMode to 'control-group' for worker.service

- Linting
- Fix WebHdfsSensor
- Adding more licenses to pass checks
- fixing landscape's config
- [hotfix] typo that made it in master
- [hotfix] fixing landscape requirement detection
- Make testing on hive conditional
- Merge remote-tracking branch 'upstream/master' into minicluster
- Update README.md
- Throwing in a few license to pass the build
- Adding a reqs.txt for landscape.io
- Pointing to a reqs file
- Some linting
- Adding a .landscape.yml file
- badge for pypi version
- Add license and ignore for sql and csv
- Use correct connection id
- Use correct table name
- Provide data for ci tests
- new badge for showing staleness of reqs
- removing requirements.txt as it is uni-dimensional
- Make it work on py3
- Remove decode for logging
- Also keep py2 compatible
- More py3 fixes
- Convert to bytes for py3 compat
- Make sure to be py3 compatible
- Use unicodcsv to make it py3 compatible
- Replace tab with spaces Remove unused import
- Merge remote-tracking branch 'upstream/master'
- Support decimal types in MySQL to GCS
- Make sure to write binary as string can be unicode
- Ignore metastore
- More impyla fixes
- Test HivemetaStore if python 2
- Allow users to set hdfs\_namenode\_principal in HDFSHook config

- Add tests for Hiveserver2 and fix some issues from impyla
- Merge branch 'impyla' into minicluster
- This patch allows for testing of hive operators and hooks. Sasl is used (NoSasl in connection string is not possible). Tests have been adjusted.
- Treat SKIPPED and SUCCESS the same way when evaluating depends\_on\_past=True
- fix bigquery hook
- version cap for gcp\_api
- Fix typo when returning VerticaHook
- Adding fernet key to use it as part of stdout commands
- Adding support for ssl parameters. (picking up from jthomas123)
- more detail in error message.
- make sure paths don't conflict bc of trailing /
- change gcs\_hook to self.hook
- refactor remote log read/write and add GCS support
- Only use multipart upload in S3Hook if file is large enough
- Merge branch 'airbnb/master'
- Add GSSAPI SASL to HiveMetaStoreHook.
- Add warning for deprecated setting
- Use kerberos\_service\_name = 'hive' as standard instead of 'impala'.
- Use GSSAPI instead of KERBEROS and provide backwards compatibility
- ISSUE-1123 Use impyla instead of pyhs2
- set celery\_executor to use queue name as exchange

## 3.20 FAQ

### 3.20.1 Why isn't my task getting scheduled?

There are very many reasons why your task might not be getting scheduled. Here are some of the common causes:

- Does your script “compile”, can the Airflow engine parse it and find your DAG object. To test this, you can run `airflow list_dags` and confirm that your DAG shows up in the list. You can also run `airflow list_tasks foo_dag_id --tree` and confirm that your task shows up in the list as expected. If you use the CeleryExecutor, you may want to confirm that this works both where the scheduler runs as well as where the worker runs.
- Does the file containing your DAG contain the string “airflow” and “DAG” somewhere in the contents? When searching the DAG directory, Airflow ignores files not containing “airflow” and “DAG” in order to prevent the DagBag parsing from importing all python files collocated with user's DAGs.
- Is your `start_date` set properly? The Airflow scheduler triggers the task soon after the `start_date` + `scheduler_interval` is passed.

- Is your `schedule_interval` set properly? The default `schedule_interval` is one day (`datetime.timedelta(1)`). You must specify a different `schedule_interval` directly to the DAG object you instantiate, not as a `default_param`, as task instances do not override their parent DAG's `schedule_interval`.
- Is your `start_date` beyond where you can see it in the UI? If you set your `start_date` to some time say 3 months ago, you won't be able to see it in the main view in the UI, but you should be able to see it in the Menu -> Browse -> Task Instances.
- Are the dependencies for the task met. The task instances directly upstream from the task need to be in a success state. Also, if you have set `depends_on_past=True`, the previous task instance needs to have succeeded (except if it is the first run for that task). Also, if `wait_for_downstream=True`, make sure you understand what it means. You can view how these properties are set from the Task Instance Details page for your task.
- Are the DagRuns you need created and active? A DagRun represents a specific execution of an entire DAG and has a state (running, success, failed, ...). The scheduler creates new DagRun as it moves forward, but never goes back in time to create new ones. The scheduler only evaluates running DagRuns to see what task instances it can trigger. Note that clearing tasks instances (from the UI or CLI) does set the state of a DagRun back to running. You can bulk view the list of DagRuns and alter states by clicking on the schedule tag for a DAG.
- Is the `concurrency` parameter of your DAG reached? `concurrency` defines how many running task instances a DAG is allowed to have, beyond which point things get queued.
- Is the `max_active_runs` parameter of your DAG reached? `max_active_runs` defines how many running concurrent instances of a DAG there are allowed to be.

You may also want to read the Scheduler section of the docs and make sure you fully understand how it proceeds.

### 3.20.2 How do I trigger tasks based on another task's failure?

Check out the `Trigger Rule` section in the Concepts section of the documentation.

### 3.20.3 Why are connection passwords still not encrypted in the metadata db after I installed airflow[crypto]?

Check out the `Securing Connections` section in the How-to Guides section of the documentation.

### 3.20.4 What's the deal with `start_date`?

`start_date` is partly legacy from the pre-DagRun era, but it is still relevant in many ways. When creating a new DAG, you probably want to set a global `start_date` for your tasks using `default_args`. The first DagRun to be created will be based on the `min(start_date)` for all your task. From that point on, the scheduler creates new DagRuns based on your `schedule_interval` and the corresponding task instances run as your dependencies are met. When introducing new tasks to your DAG, you need to pay special attention to `start_date`, and may want to reactivate inactive DagRuns to get the new task onboarded properly.

We recommend against using dynamic values as `start_date`, especially `datetime.now()` as it can be quite confusing. The task is triggered once the period closes, and in theory an `@hourly` DAG would never get to an hour after now as `now()` moves along.

Previously we also recommended using rounded `start_date` in relation to your `schedule_interval`. This meant an `@hourly` would be at 00:00 minutes:seconds, a `@daily` job at midnight, a `@monthly` job on the first of the month. This is no longer required. Airflow will now auto align the `start_date` and the `schedule_interval`, by using the `start_date` as the moment to start looking.

You can use any sensor or a `TimeDeltaSensor` to delay the execution of tasks within the schedule interval. While `schedule_interval` does allow specifying a `datetime.timedelta` object, we recommend using the macros or cron expressions instead, as it enforces this idea of rounded schedules.

When using `depends_on_past=True` it's important to pay special attention to `start_date` as the past dependency is not enforced only on the specific schedule of the `start_date` specified for the task. It's also important to watch DagRun activity status in time when introducing new `depends_on_past=True`, unless you are planning on running a backfill for the new task(s).

Also important to note is that the tasks `start_date`, in the context of a backfill CLI command, get overridden by the backfill's command `start_date`. This allows for a backfill on tasks that have `depends_on_past=True` to actually start, if that wasn't the case, the backfill just wouldn't start.

### 3.20.5 How can I create DAGs dynamically?

Airflow looks in your `DAGS_FOLDER` for modules that contain DAG objects in their global namespace, and adds the objects it finds in the `DagBag`. Knowing this all we need is a way to dynamically assign variable in the global namespace, which is easily done in python using the `globals()` function for the standard library which behaves like a simple dictionary.

```
for i in range(10):
    dag_id = 'foo_{}'.format(i)
    globals()[dag_id] = DAG(dag_id)
    # or better, call a function that returns a DAG object!
```

### 3.20.6 What are all the airflow run commands in my process list?

There are many layers of `airflow run` commands, meaning it can call itself.

- Basic `airflow run`: fires up an executor, and tell it to run an `airflow run --local` command. If using Celery, this means it puts a command in the queue for it to run remotely on the worker. If using LocalExecutor, that translates into running it in a subprocess pool.
- Local `airflow run --local`: starts an `airflow run --raw` command (described below) as a subprocess and is in charge of emitting heartbeats, listening for external kill signals and ensures some cleanup takes place if the subprocess fails.
- Raw `airflow run --raw` runs the actual operator's `execute` method and performs the actual work.

### 3.20.7 How can my airflow dag run faster?

There are three variables we could control to improve airflow dag performance:

- `parallelism`: This variable controls the number of task instances that the airflow worker can run simultaneously. User could increase the `parallelism` variable in the `airflow.cfg`.
- `concurrency`: The Airflow scheduler will run no more than `$concurrency` task instances for your DAG at any given time. Concurrency is defined in your Airflow DAG. If you do not set the concurrency on your DAG, the scheduler will use the default value from the `dag_concurrency` entry in your `airflow.cfg`.
- `max_active_runs`: the Airflow scheduler will run no more than `max_active_runs` DagRuns of your DAG at a given time. If you do not set the `max_active_runs` in your DAG, the scheduler will use the default value from the `max_active_runs_per_dag` entry in your `airflow.cfg`.



### 3.20.8 How can we reduce the airflow UI page load time?

If your dag takes long time to load, you could reduce the value of `default_dag_run_display_number` configuration in `airflow.cfg` to a smaller value. This configurable controls the number of dag run to show in UI with default value 25.

### 3.20.9 How to fix Exception: Global variable `explicit_defaults_for_timestamp` needs to be on (1)?

This means `explicit_defaults_for_timestamp` is disabled in your mysql server and you need to enable it by:

1. Set `explicit_defaults_for_timestamp = 1` under the `mysqld` section in your `my.cnf` file.
2. Restart the Mysql server.

### 3.20.10 How to reduce airflow dag scheduling latency in production?

- `max_threads`: Scheduler will spawn multiple threads in parallel to schedule dags. This is controlled by `max_threads` with default value of 2. User should increase this value to a larger value(e.g numbers of cpus where scheduler runs - 1) in production.
- `scheduler_heartbeat_sec`: User should consider to increase `scheduler_heartbeat_sec` config to a higher value(e.g 60 secs) which controls how frequent the airflow scheduler gets the heartbeat and updates the job's entry in database.

## 3.21 API Reference

### 3.21.1 Operators

Operators allow for generation of certain types of tasks that become nodes in the DAG when instantiated. All operators derive from `BaseOperator` and inherit many attributes and methods that way. Refer to the [BaseOperator](#) documentation for more details.

There are 3 main types of operators:

- Operators that performs an **action**, or tell another system to perform an action
- **Transfer** operators move data from one system to another
- **Sensors** are a certain type of operator that will keep running until a certain criterion is met. Examples include a specific file landing in HDFS or S3, a partition appearing in Hive, or a specific time of the day. Sensors are derived from `BaseSensorOperator` and run a `poke` method at a specified `poke_interval` until it returns `True`.

#### 3.21.1.1 BaseOperator

All operators are derived from `BaseOperator` and acquire much functionality through inheritance. Since this is the core of the engine, it's worth taking the time to understand the parameters of `BaseOperator` to understand the primitive features that can be leveraged in your DAGs.

```
class airflow.models.BaseOperator (**kwargs)
    Bases: airflow.utils.log.logging_mixin.LoggingMixin
```

Abstract base class for all operators. Since operators create objects that become nodes in the dag, BaseOperator contains many recursive methods for dag crawling behavior. To derive this class, you are expected to override the constructor as well as the ‘execute’ method.

Operators derived from this class should perform or trigger certain tasks synchronously (wait for completion). Example of operators could be an operator that runs a Pig job (PigOperator), a sensor operator that waits for a partition to land in Hive (HiveSensorOperator), or one that moves data from Hive to MySQL (Hive2MySQLOperator). Instances of these operators (tasks) target specific operations, running specific scripts, functions or data transfers.

This class is abstract and shouldn’t be instantiated. Instantiating a class derived from this one results in the creation of a task object, which ultimately becomes a node in DAG objects. Task dependencies should be set by using the set\_upstream and/or set\_downstream methods.

#### Parameters

- **task\_id** (*str*) – a unique, meaningful id for the task
- **owner** (*str*) – the owner of the task, using the unix username is recommended
- **retries** (*int*) – the number of retries that should be performed before failing the task
- **retry\_delay** (*datetime.timedelta*) – delay between retries
- **retry\_exponential\_backoff** (*bool*) – allow progressive longer waits between retries by using exponential backoff algorithm on retry delay (delay will be converted into seconds)
- **max\_retry\_delay** (*datetime.timedelta*) – maximum delay interval between retries
- **start\_date** (*datetime.datetime*) – The start\_date for the task, determines the execution\_date for the first task instance. The best practice is to have the start\_date rounded to your DAG’s schedule\_interval. Daily jobs have their start\_date some day at 00:00:00, hourly jobs have their start\_date at 00:00 of a specific hour. Note that Airflow simply looks at the latest execution\_date and adds the schedule\_interval to determine the next execution\_date. It is also very important to note that different tasks’ dependencies need to line up in time. If task A depends on task B and their start\_date are offset in a way that their execution\_date don’t line up, A’s dependencies will never be met. If you are looking to delay a task, for example running a daily task at 2AM, look into the TimeSensor and TimeDeltaSensor. We advise against using dynamic start\_date and recommend using fixed ones. Read the FAQ entry about start\_date for more information.
- **end\_date** (*datetime.datetime*) – if specified, the scheduler won’t go beyond this date
- **depends\_on\_past** (*bool*) – when set to true, task instances will run sequentially while relying on the previous task’s schedule to succeed. The task instance for the start\_date is allowed to run.
- **wait\_for\_downstream** (*bool*) – when set to true, an instance of task X will wait for tasks immediately downstream of the previous instance of task X to finish successfully before it runs. This is useful if the different instances of a task X alter the same asset, and this asset is used by tasks downstream of task X. Note that depends\_on\_past is forced to True wherever wait\_for\_downstream is used.

- **queue** (*str*) – which queue to target when running this job. Not all executors implement queue management, the CeleryExecutor does support targeting specific queues.
- **dag** (`airflow.models.DAG`) – a reference to the dag the task is attached to (if any)
- **priority\_weight** (*int*) – priority weight of this task against other task. This allows the executor to trigger higher priority tasks before others when things get backed up. Set `priority_weight` as a higher number for more important tasks.
- **weight\_rule** (*str*) – weighting method used for the effective total priority weight of the task. Options are: { `downstream` | `upstream` | `absolute` } default is `downstream` When set to `downstream` the effective weight of the task is the aggregate sum of all downstream descendants. As a result, upstream tasks will have higher weight and will be scheduled more aggressively when using positive weight values. This is useful when you have multiple dag run instances and desire to have all upstream tasks to complete for all runs before each dag can continue processing downstream tasks. When set to `upstream` the effective weight is the aggregate sum of all upstream ancestors. This is the opposite where downstream tasks have higher weight and will be scheduled more aggressively when using positive weight values. This is useful when you have multiple dag run instances and prefer to have each dag complete before starting upstream tasks of other dags. When set to `absolute`, the effective weight is the exact `priority_weight` specified without additional weighting. You may want to do this when you know exactly what priority weight each task should have. Additionally, when set to `absolute`, there is bonus effect of significantly speeding up the task creation process as for very large DAGS. Options can be set as string or using the constants defined in the static class `airflow.utils.WeightRule`
- **pool** (*str*) – the slot pool this task should run in, slot pools are a way to limit concurrency for certain tasks
- **sla** (`datetime.timedelta`) – time by which the job is expected to succeed. Note that this represents the `timedelta` after the period is closed. For example if you set an SLA of 1 hour, the scheduler would send an email soon after 1:00AM on the 2016-01-02 if the 2016-01-01 instance has not succeeded yet. The scheduler pays special attention for jobs with an SLA and sends alert emails for sla misses. SLA misses are also recorded in the database for future reference. All tasks that share the same SLA time get bundled in a single email, sent soon after that time. SLA notification are sent once and only once for each task instance.
- **execution\_timeout** (`datetime.timedelta`) – max time allowed for the execution of this task instance, if it goes beyond it will raise and fail.
- **on\_failure\_callback** (*callable*) – a function to be called when a task instance of this task fails. a context dictionary is passed as a single parameter to this function. Context contains references to related objects to the task instance and is documented under the macros section of the API.
- **on\_retry\_callback** (*callable*) – much like the `on_failure_callback` except that it is executed when retries occur.
- **on\_success\_callback** (*callable*) – much like the `on_failure_callback` except that it is executed when the task succeeds.
- **trigger\_rule** (*str*) – defines the rule by which dependencies are applied for the task to get triggered. Options are: { `all_success` | `all_failed` | `all_done` | `one_success` | `one_failed` | `none_failed` | `dummy` } default is `all_success`. Options can be set as string or using the constants defined in the static class `airflow.utils.TriggerRule`

- **resources** (*dict*) – A map of resource parameter names (the argument names of the Resources constructor) to their values.
- **run\_as\_user** (*str*) – unix username to impersonate while running the task
- **task\_concurrency** (*int*) – When set, a task will be able to limit the concurrent runs across execution\_dates
- **executor\_config** (*dict*) – Additional task-level configuration parameters that are interpreted by a specific executor. Parameters are namespaced by the name of executor.

**Example:** to run this task in a specific docker container through the KubernetesExecutor

```
MyOperator(...,
    executor_config={
        "KubernetesExecutor":
            {"image": "myCustomDockerImage"}
    }
)
```

- **do\_xcom\_push** (*bool*) – if True, an XCom is pushed containing the Operator's result

**clear** (*\*\*kwargs*)

Clears the state of task instances associated with the task, following the parameters specified.

**dag**

Returns the Operator's DAG if set, otherwise raises an error

**deps**

Returns the list of dependencies for the operator. These differ from execution context dependencies in that they are specific to tasks and can be extended/overridden by subclasses.

**downstream\_list**

@property: list of tasks directly downstream

**execute** (*context*)

This is the main method to derive when creating an operator. Context is the same dictionary used as when rendering jinja templates.

Refer to `get_template_context` for more context.

**get\_direct\_relative\_ids** (*upstream=False*)

Get the direct relative ids to the current task, upstream or downstream.

**get\_direct\_relatives** (*upstream=False*)

Get the direct relatives to the current task, upstream or downstream.

**get\_flat\_relative\_ids** (*upstream=False, found\_descendants=None*)

Get a flat list of relatives' ids, either upstream or downstream.

**get\_flat\_relatives** (*upstream=False*)

Get a flat list of relatives, either upstream or downstream.

**get\_task\_instances** (*session, start\_date=None, end\_date=None*)

Get a set of task instance related to this task for a specific date range.

**has\_dag** ()

Returns True if the Operator has been assigned to a DAG.

**on\_kill** ()

Override this method to cleanup subprocesses when a task instance gets killed. Any use of the threading, subprocess or multiprocessing module within an operator needs to be cleaned up or it will leave ghost processes behind.

**post\_execute** (*context*, \**args*, \*\**kwargs*)

This hook is triggered right after `self.execute()` is called. It is passed the execution context and any results returned by the operator.

**pre\_execute** (*context*, \**args*, \*\**kwargs*)

This hook is triggered right before `self.execute()` is called.

**prepare\_template** ()

Hook that is triggered after the templated fields get replaced by their content. If you need your operator to alter the content of the file before the template is rendered, it should override this method to do so.

**render\_template** (*attr*, *content*, *context*)

Renders a template either from a file or directly in a field, and returns the rendered result.

**render\_template\_from\_field** (*attr*, *content*, *context*, *jinja\_env*)

Renders a template from a field. If the field is a string, it will simply render the string and return the result. If it is a collection or nested set of collections, it will traverse the structure and render all elements in it. If the field has another type, it will return it as it is.

**run** (*start\_date=None*, *end\_date=None*, *ignore\_first\_depends\_on\_past=False*, *ignore\_ti\_state=False*, *mark\_success=False*)

Run a set of task instances for a date range.

**schedule\_interval**

The schedule interval of the DAG always wins over individual tasks so that tasks within a DAG always line up. The task still needs a `schedule_interval` as it may not be attached to a DAG.

**set\_downstream** (*task\_or\_task\_list*)

Set a task or a task list to be directly downstream from the current task.

**set\_upstream** (*task\_or\_task\_list*)

Set a task or a task list to be directly upstream from the current task.

**upstream\_list**

@property: list of tasks directly upstream

**xcom\_pull** (*context*, *task\_ids=None*, *dag\_id=None*, *key=u'return\_value'*, *include\_prior\_dates=None*)

See `TaskInstance.xcom_pull()`

**xcom\_push** (*context*, *key*, *value*, *execution\_date=None*)

See `TaskInstance.xcom_push()`

### 3.21.1.2 BaseSensorOperator

All sensors are derived from `BaseSensorOperator`. All sensors inherit the `timeout` and `poke_interval` on top of the `BaseOperator` attributes.

**class** `airflow.sensors.base_sensor_operator.BaseSensorOperator` (\*\**kwargs*)

Bases: `airflow.models.BaseOperator`, `airflow.models.SkipMixin`

Sensor operators are derived from this class and inherit these attributes.

Sensor operators keep executing at a time interval and succeed when a criteria is met and fail if and when they time out.

#### Parameters

- **soft\_fail** (*bool*) – Set to true to mark the task as SKIPPED on failure
- **poke\_interval** (*int*) – Time in seconds that the job should wait in between each tries
- **timeout** (*int*) – Time, in seconds before the task times out and fails.

- **mode** (*str*) – How the sensor operates. Options are: { `poke` | `reschedule` }, default is `poke`. When set to `poke` the sensor is taking up a worker slot for its whole execution time and sleeps between pokes. Use this mode if the expected runtime of the sensor is short or if a short poke interval is required. When set to `reschedule` the sensor task frees the worker slot when the criteria is not yet met and it's rescheduled at a later time. Use this mode if the expected time until the criteria is met is. The poke interval should be more than one minute to prevent too much load on the scheduler.

**deps**

Adds one additional dependency for all sensor operators that checks if a sensor task instance can be rescheduled.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

### 3.21.1.3 Core Operators

#### Operators

**class** `airflow.operators.bash_operator.BashOperator` (*\*\*kwargs*)

Bases: `airflow.models.BaseOperator`

Execute a Bash script, command or set of commands.

**See also:**

For more information on how to use this operator, take a look at the guide: [BashOperator](#)

**Parameters**

- **bash\_command** (*str*) – The command, set of commands or reference to a bash script (must be `.sh`) to be executed. (templated)
- **xcom\_push** (*bool*) – If `xcom_push` is `True`, the last line written to stdout will also be pushed to an XCom when the bash command completes.
- **env** (*dict*) – If `env` is not `None`, it must be a mapping that defines the environment variables for the new process; these are used instead of inheriting the current process environment, which is the default behavior. (templated)
- **output\_encoding** (*str*) – Output encoding of bash command

On execution of this operator the task will be up for retry when exception is raised. However, if a sub-command exits with non-zero value Airflow will not recognize it as failure unless the whole shell exits with a failure. The easiest way of achieving this is to prefix the command with `set -e`; Example:

```
bash_command = "set -e; python3 script.py '{{ next_execution_date }}'"
```

**execute** (*context*)

Execute the bash command in a temporary directory which will be cleaned afterwards

**class** `airflow.operators.python_operator.BranchPythonOperator` (*\*\*kwargs*)

Bases: `airflow.operators.python_operator.PythonOperator`, `airflow.models.SkipMixin`

Allows a workflow to “branch” or follow a path following the execution of this task.

It derives the `PythonOperator` and expects a Python function that returns a single `task_id` or list of `task_ids` to follow. The `task_id(s)` returned should point to a task directly downstream from `{self}`. All other “branches”

or directly downstream tasks are marked with a state of `skipped` so that these paths can't move forward. The `skipped` states are propagated downstream to allow for the DAG state to fill up and the DAG run's state to be inferred.

Note that using tasks with `depends_on_past=True` downstream from `BranchPythonOperator` is logically unsound as `skipped` status will invariably lead to block tasks that depend on their past successes. `skipped` states propagates where all directly upstream tasks are `skipped`.

**class** `airflow.operators.check_operator.CheckOperator` (*\*\*kwargs*)

Bases: `airflow.models.BaseOperator`

Performs checks against a db. The `CheckOperator` expects a sql query that will return a single row. Each value on that first row is evaluated using python bool casting. If any of the values return `False` the check is failed and errors out.

Note that Python bool casting evals the following as `False`:

- `False`
- `0`
- Empty string (`""`)
- Empty list (`[]`)
- Empty dictionary or set (`{}`)

Given a query like `SELECT COUNT(*) FROM foo`, it will fail only if the count `== 0`. You can craft much more complex query that could, for instance, check that the table has the same number of rows as the source table upstream, or that the count of today's partition is greater than yesterday's partition, or that a set of metrics are less than 3 standard deviation for the 7 day average.

This operator can be used as a data quality check in your pipeline, and depending on where you put it in your DAG, you have the choice to stop the critical path, preventing from publishing dubious data, or on the side and receive email alerts without stopping the progress of the DAG.

Note that this is an abstract class and `get_db_hook` needs to be defined. Whereas a `get_db_hook` is hook that gets a single record from an external source.

**Parameters** `sql` (*str*) – the sql to be executed. (templated)

**class** `airflow.operators.docker_operator.DockerOperator` (*\*\*kwargs*)

Bases: `airflow.models.BaseOperator`

Execute a command inside a docker container.

A temporary directory is created on the host and mounted into a container to allow storing files that together exceed the default disk size of 10GB in a container. The path to the mounted directory can be accessed via the environment variable `AIRFLOW_TMP_DIR`.

If a login to a private registry is required prior to pulling the image, a Docker connection needs to be configured in Airflow and the connection ID be provided with the parameter `docker_conn_id`.

**Parameters**

- **image** (*str*) – Docker image from which to create the container. If image tag is omitted, "latest" will be used.
- **api\_version** (*str*) – Remote API version. Set to `auto` to automatically detect the server's version.
- **auto\_remove** (*bool*) – Auto-removal of the container on daemon side when the container's process exits. The default is `False`.
- **command** (*str or list*) – Command to be run in the container. (templated)



- **cpus** (*float*) – Number of CPUs to assign to the container. This value gets multiplied with 1024. See <https://docs.docker.com/engine/reference/run/#cpu-share-constraint>
- **dns** (*list[str]*) – Docker custom DNS servers
- **dns\_search** (*list[str]*) – Docker custom DNS search domain
- **docker\_url** (*str*) – URL of the host running the docker daemon. Default is `unix://var/run/docker.sock`
- **environment** (*dict*) – Environment variables to set in the container. (templated)
- **force\_pull** (*bool*) – Pull the docker image on every run. Default is False.
- **mem\_limit** (*float or str*) – Maximum amount of memory the container can use. Either a float value, which represents the limit in bytes, or a string like `128m` or `1g`.
- **network\_mode** (*str*) – Network mode for the container.
- **tls\_ca\_cert** (*str*) – Path to a PEM-encoded certificate authority to secure the docker connection.
- **tls\_client\_cert** (*str*) – Path to the PEM-encoded certificate used to authenticate docker client.
- **tls\_client\_key** (*str*) – Path to the PEM-encoded key used to authenticate docker client.
- **tls\_hostname** (*str or bool*) – Hostname to match against the docker server certificate or False to disable the check.
- **tls\_ssl\_version** (*str*) – Version of SSL to use when communicating with docker daemon.
- **tmp\_dir** (*str*) – Mount point inside the container to a temporary directory created on the host by the operator. The path is also made available via the environment variable `AIRFLOW_TMP_DIR` inside the container.
- **user** (*int or str*) – Default user inside the docker container.
- **volumes** – List of volumes to mount into the container, e.g. `['/host/path:/container/path', '/host/path2:/container/path2:ro']`.
- **working\_dir** (*str*) – Working directory to set on the container (equivalent to the `-w` switch the docker client)
- **xcom\_push** (*bool*) – Does the stdout will be pushed to the next step using XCom. The default is False.
- **xcom\_all** (*bool*) – Push all the stdout or just the last line. The default is False (last line).
- **docker\_conn\_id** (*str*) – ID of the Airflow connection to use
- **shm\_size** (*int*) – Size of `/dev/shm` in bytes. The size must be greater than 0. If omitted uses system default.

**class** `airflow.operators.druid_check_operator.DruidCheckOperator` (*\*\*kwargs*)

Bases: `airflow.operators.check_operator.CheckOperator`

Performs checks against Druid. The `DruidCheckOperator` expects a sql query that will return a single row. Each value on that first row is evaluated using python `bool` casting. If any of the values return `False` the check is failed and errors out.

Note that Python `bool` casting evals the following as `False`:

- `False`



- 0
- Empty string ("")
- Empty list ([])
- Empty dictionary or set ({})

Given a query like `SELECT COUNT(*) FROM foo`, it will fail only if the count == 0. You can craft much more complex query that could, for instance, check that the table has the same number of rows as the source table upstream, or that the count of today's partition is greater than yesterday's partition, or that a set of metrics are less than 3 standard deviation for the 7 day average. This operator can be used as a data quality check in your pipeline, and depending on where you put it in your DAG, you have the choice to stop the critical path, preventing from publishing dubious data, or on the side and receive email alerts without stopping the progress of the DAG.

#### Parameters

- **sql** (*str*) – the sql to be executed
- **druid\_broker\_conn\_id** (*str*) – reference to the druid broker

**get\_db\_hook** ()

Return the druid db api hook.

**get\_first** (*sql*)

Executes the druid sql to druid broker and returns the first resulting row.

**Parameters** **sql** (*str*) – the sql statement to be executed (*str*)

**class** airflow.operators.dummy\_operator.**DummyOperator** (\*\**kwargs*)

Bases: *airflow.models.BaseOperator*

Operator that does literally nothing. It can be used to group tasks in a DAG.

**class** airflow.operators.email\_operator.**EmailOperator** (\*\**kwargs*)

Bases: *airflow.models.BaseOperator*

Sends an email.

#### Parameters

- **to** (*list or string (comma or semicolon delimited)*) – list of emails to send the email to. (templated)
- **subject** (*str*) – subject line for the email. (templated)
- **html\_content** (*str*) – content of the email, html markup is allowed. (templated)
- **files** (*list*) – file names to attach in email
- **cc** (*list or string (comma or semicolon delimited)*) – list of recipients to be added in CC field
- **bcc** (*list or string (comma or semicolon delimited)*) – list of recipients to be added in BCC field
- **mime\_subtype** (*str*) – MIME sub content type
- **mime\_charset** (*str*) – character set parameter added to the Content-Type header.

**class** airflow.operators.generic\_transfer.**GenericTransfer** (\*\**kwargs*)

Bases: *airflow.models.BaseOperator*

Moves data from a connection to another, assuming that they both provide the required methods in their respective hooks. The source hook needs to expose a *get\_records* method, and the destination a *insert\_rows* method.

This is meant to be used on small-ish datasets that fit in memory.

#### Parameters

- **sql** (*str*) – SQL query to execute against the source database. (templated)
- **destination\_table** (*str*) – target table. (templated)
- **source\_conn\_id** (*str*) – source connection
- **destination\_conn\_id** (*str*) – source connection
- **preoperator** (*str* or *list[str]*) – sql statement or list of statements to be executed prior to loading the data. (templated)

**class** airflow.operators.hive\_to\_samba\_operator.**Hive2SambaOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Executes hql code in a specific Hive database and loads the results of the query as a csv to a Samba location.

#### Parameters

- **hql** (*str*) – the hql to be exported. (templated)
- **destination\_filepath** (*str*) – the file path to where the file will be pushed onto samba
- **samba\_conn\_id** (*str*) – reference to the samba destination
- **hiveserver2\_conn\_id** (*str*) – reference to the hiveserver2 service

**class** airflow.operators.hive\_operator.**HiveOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Executes hql code or hive script in a specific Hive database.

#### Parameters

- **hql** (*str*) – the hql to be executed. Note that you may also use a relative path from the dag file of a (template) hive script. (templated)
- **hive\_cli\_conn\_id** (*str*) – reference to the Hive database. (templated)
- **hiveconfs** (*dict*) – if defined, these key value pairs will be passed to hive as `-hiveconf "key"="value"`
- **hiveconf\_jinja\_translate** (*bool*) – when True, hiveconf-type templating `${var}` gets translated into jinja-type templating `{{ var }}` and `${hiveconf:var}` gets translated into jinja-type templating `{{ var }}`. Note that you may want to use this along with the `DAG(user_defined_macros=myargs)` parameter. View the DAG object documentation for more details.
- **script\_begin\_tag** (*str*) – If defined, the operator will get rid of the part of the script before the first occurrence of *script\_begin\_tag*
- **mapred\_queue** (*str*) – queue used by the Hadoop CapacityScheduler. (templated)
- **mapred\_queue\_priority** (*str*) – priority within CapacityScheduler queue. Possible settings include: VERY\_HIGH, HIGH, NORMAL, LOW, VERY\_LOW
- **mapred\_job\_name** (*str*) – This name will appear in the jobtracker. This can make monitoring easier.

**class** airflow.operators.hive\_stats\_operator.**HiveStatsCollectionOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Gathers partition statistics using a dynamically generated Presto query, inserts the stats into a MySQL table with this format. Stats overwrite themselves if you rerun the same date/partition.

```
CREATE TABLE hive_stats (
  ds VARCHAR(16),
  table_name VARCHAR(500),
  metric VARCHAR(200),
  value BIGINT
);
```

### Parameters

- **table** (*str*) – the source table, in the format `database.table_name`. (templated)
- **partition** (*dict of {col:value}*) – the source partition. (templated)
- **extra\_exprs** (*dict*) – dict of expression to run against the table where keys are metric names and values are Presto compatible expressions
- **col\_blacklist** (*list*) – list of columns to blacklist, consider blacklisting blobs, large json columns, ...
- **assignment\_func** (*function*) – a function that receives a column name and a type, and returns a dict of metric names and an Presto expressions. If None is returned, the global defaults are applied. If an empty dictionary is returned, no stats are computed for that column.

**class** `airflow.operators.hive_to_druid.HiveToDruidTransfer` (\*\*kwargs)

Bases: `airflow.models.BaseOperator`

Moves data from Hive to Druid, [del]note that for now the data is loaded into memory before being pushed to Druid, so this operator should be used for smallish amount of data.[/del]

### Parameters

- **sql** (*str*) – SQL query to execute against the Druid database. (templated)
- **druid\_datasource** (*str*) – the datasource you want to ingest into in druid
- **ts\_dim** (*str*) – the timestamp dimension
- **metric\_spec** (*list*) – the metrics you want to define for your data
- **hive\_cli\_conn\_id** (*str*) – the hive connection id
- **druid\_ingest\_conn\_id** (*str*) – the druid ingest connection id
- **metastore\_conn\_id** (*str*) – the metastore connection id
- **hadoop\_dependency\_coordinates** (*list[str]*) – list of coordinates to squeeze into the ingest json
- **intervals** (*list*) – list of time intervals that defines segments, this is passed as is to the json object. (templated)
- **hive\_tblproperties** (*dict*) – additional properties for tblproperties in hive for the staging table
- **job\_properties** (*dict*) – additional properties for job

**construct\_ingest\_query** (*static\_path, columns*)

Builds an ingest query for an HDFS TSV load.

### Parameters

- **static\_path** (*str*) – The path on hdfs where the data is
- **columns** (*list*) – List of all the columns that are available

**class** airflow.operators.hive\_to\_mysql.**HiveToMySQLTransfer** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Moves data from Hive to MySQL, note that for now the data is loaded into memory before being pushed to MySQL, so this operator should be used for smallish amount of data.

#### Parameters

- **sql** (*str*) – SQL query to execute against Hive server. (templated)
- **mysql\_table** (*str*) – target MySQL table, use dot notation to target a specific database. (templated)
- **mysql\_conn\_id** (*str*) – source mysql connection
- **hiveserver2\_conn\_id** (*str*) – destination hive connection
- **mysql\_preoperator** (*str*) – sql statement to run against mysql prior to import, typically use to truncate or delete in place of the data coming in, allowing the task to be idempotent (running the task twice won't double load data). (templated)
- **mysql\_postoperator** (*str*) – sql statement to run against mysql after the import, typically used to move data from staging to production and issue cleanup commands. (templated)
- **bulk\_load** (*bool*) – flag to use bulk\_load option. This loads mysql directly from a tab-delimited text file using the LOAD DATA LOCAL INFILE command. This option requires an extra connection parameter for the destination MySQL connection: {'local\_infile': true}.

**class** airflow.operators.check\_operator.**IntervalCheckOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from days\_back before.

Note that this is an abstract class and get\_db\_hook needs to be defined. Whereas a get\_db\_hook is hook that gets a single record from an external source.

#### Parameters

- **table** (*str*) – the table name
- **days\_back** (*int*) – number of days between ds and the ds we want to check against. Defaults to 7 days
- **metrics\_threshold** (*dict*) – a dictionary of ratios indexed by metrics

**class** airflow.operators.jdbc\_operator.**JdbcOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Executes sql code in a database using jdbc driver.

Requires jaydebeapi.

#### Parameters

- **sql** (Can receive a *str* representing a sql statement, a list of *str* (sql statements), or reference to a template file. Template reference are recognized by *str* ending in '.sql')
- **jdbc\_conn\_id** (*str*) – reference to a predefined database

- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)
- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.

**class** airflow.operators.latest\_only\_operator.**LatestOnlyOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*, *airflow.models.SkipMixin*

Allows a workflow to skip tasks that are not running during the most recent schedule interval.

If the task is run outside of the latest schedule interval, all directly downstream tasks will be skipped.

**class** airflow.operators.mssql\_operator.**MssqlOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific Microsoft SQL database

#### Parameters

- **sql** (*str or string pointing to a template file with .sql extension. (templated)*) – the sql code to be executed
- **mssql\_conn\_id** (*str*) – reference to a specific mssql database
- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.
- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)
- **database** (*str*) – name of database which overwrite defined one in connection

**class** airflow.operators.mssql\_to\_hive.**MssqlToHiveTransfer** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Moves data from Microsoft SQL Server to Hive. The operator runs your query against Microsoft SQL Server, stores the file locally before loading it into a Hive table. If the `create` or `recreate` arguments are set to True, a `CREATE TABLE` and `DROP TABLE` statements are generated. Hive data types are inferred from the cursor's metadata. Note that the table generated in Hive uses `STORED AS textfile` which isn't the most efficient serialization format. If a large amount of data is loaded and/or if the table gets queried considerably, you may want to use this operator only to stage the data into a temporary table before loading it into its final destination using a `HiveOperator`.

#### Parameters

- **sql** (*str*) – SQL query to execute against the Microsoft SQL Server database. (templated)
- **hive\_table** (*str*) – target Hive table, use dot notation to target a specific database. (templated)
- **create** (*bool*) – whether to create the table if it doesn't exist
- **recreate** (*bool*) – whether to drop and recreate the table at every execution
- **partition** (*dict*) – target partition as a dict of partition columns and values. (templated)
- **delimiter** (*str*) – field delimiter in the file
- **mssql\_conn\_id** (*str*) – source Microsoft SQL Server connection
- **hive\_conn\_id** (*str*) – destination hive connection
- **tblproperties** (*dict*) – TBLPROPERTIES of the hive table being created

```
class airflow.operators.mysql_operator.MySqlOperator (**kwargs)
```

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific MySQL database

#### Parameters

- **sql** (*str* or *list[str]*) – the sql code to be executed. Can receive a str representing a sql statement, a list of str (sql statements), or reference to a template file. Template reference are recognized by str ending in '.sql' (templated)
- **mysql\_conn\_id** (*str*) – reference to a specific mysql database
- **parameters** (*mapping* or *iterable*) – (optional) the parameters to render the SQL query with.
- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)
- **database** (*str*) – name of database which overwrite defined one in connection

```
class airflow.operators.mysql_to_hive.MySqlToHiveTransfer (**kwargs)
```

Bases: *airflow.models.BaseOperator*

Moves data from MySql to Hive. The operator runs your query against MySQL, stores the file locally before loading it into a Hive table. If the `create` or `recreate` arguments are set to `True`, a `CREATE TABLE` and `DROP TABLE` statements are generated. Hive data types are inferred from the cursor's metadata. Note that the table generated in Hive uses `STORED AS textfile` which isn't the most efficient serialization format. If a large amount of data is loaded and/or if the table gets queried considerably, you may want to use this operator only to stage the data into a temporary table before loading it into its final destination using a `HiveOperator`.

#### Parameters

- **sql** (*str*) – SQL query to execute against the MySQL database. (templated)
- **hive\_table** (*str*) – target Hive table, use dot notation to target a specific database. (templated)
- **create** (*bool*) – whether to create the table if it doesn't exist
- **recreate** (*bool*) – whether to drop and recreate the table at every execution
- **partition** (*dict*) – target partition as a dict of partition columns and values. (templated)
- **delimiter** (*str*) – field delimiter in the file
- **mysql\_conn\_id** (*str*) – source mysql connection
- **hive\_conn\_id** (*str*) – destination hive connection
- **tblproperties** (*dict*) – TBLPROPERTIES of the hive table being created

```
class airflow.operators.oracle_operator.OracleOperator (**kwargs)
```

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific Oracle database

#### Parameters

- **sql** (*str* or *list[str]*) – the sql code to be executed. Can receive a str representing a sql statement, a list of str (sql statements), or reference to a template file. Template reference are recognized by str ending in '.sql' (templated)
- **oracle\_conn\_id** (*str*) – reference to a specific Oracle database

- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.
- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)

**class** airflow.operators.pig\_operator.**PigOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Executes pig script.

#### Parameters

- **pig** (*str*) – the pig latin script to be executed. (templated)
- **pig\_cli\_conn\_id** (*str*) – reference to the Hive database
- **pigparams\_jinja\_translate** (*bool*) – when True, pig params-type templating `${var}` gets translated into jinja-type templating `{{ var }}`. Note that you may want to use this along with the `DAG(user_defined_macros=myargs)` parameter. View the DAG object documentation for more details.

**class** airflow.operators.postgres\_operator.**PostgresOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific Postgres database

#### Parameters

- **sql** (Can receive a *str* representing a sql statement, a list of *str* (sql statements), or reference to a template file. Template reference are recognized by *str* ending in `'.sql'`) – the sql code to be executed. (templated)
- **postgres\_conn\_id** (*str*) – reference to a specific postgres database
- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)
- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.
- **database** (*str*) – name of database which overwrite defined one in connection

**class** airflow.operators.presto\_check\_operator.**PrestoCheckOperator** (\*\*kwargs)

Bases: *airflow.operators.check\_operator.CheckOperator*

Performs checks against Presto. The `PrestoCheckOperator` expects a sql query that will return a single row. Each value on that first row is evaluated using python `bool` casting. If any of the values return `False` the check is failed and errors out.

Note that Python `bool` casting evals the following as `False`:

- `False`
- `0`
- Empty string (`""`)
- Empty list (`[]`)
- Empty dictionary or set (`{}`)

Given a query like `SELECT COUNT(*) FROM foo`, it will fail only if the count `== 0`. You can craft much more complex query that could, for instance, check that the table has the same number of rows as the source

table upstream, or that the count of today's partition is greater than yesterday's partition, or that a set of metrics are less than 3 standard deviation for the 7 day average.

This operator can be used as a data quality check in your pipeline, and depending on where you put it in your DAG, you have the choice to stop the critical path, preventing from publishing dubious data, or on the side and receive email alerts without stopping the progress of the DAG.

#### Parameters

- **sql** (*str*) – the sql to be executed
- **presto\_conn\_id** (*str*) – reference to the Presto database

**class** airflow.operators.presto\_check\_operator.**PrestoIntervalCheckOperator** (\*\*kwargs)  
Bases: *airflow.operators.check\_operator.IntervalCheckOperator*

Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from days\_back before.

#### Parameters

- **table** (*str*) – the table name
- **days\_back** (*int*) – number of days between ds and the ds we want to check against. Defaults to 7 days
- **metrics\_threshold** (*dict*) – a dictionary of ratios indexed by metrics
- **presto\_conn\_id** (*str*) – reference to the Presto database

**class** airflow.operators.presto\_to\_mysql.**PrestoToMySQLTransfer** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Moves data from Presto to MySQL, note that for now the data is loaded into memory before being pushed to MySQL, so this operator should be used for smallish amount of data.

#### Parameters

- **sql** (*str*) – SQL query to execute against Presto. (templated)
- **mysql\_table** (*str*) – target MySQL table, use dot notation to target a specific database. (templated)
- **mysql\_conn\_id** (*str*) – source mysql connection
- **presto\_conn\_id** (*str*) – source presto connection
- **mysql\_preoperator** (*str*) – sql statement to run against mysql prior to import, typically use to truncate or delete in place of the data coming in, allowing the task to be idempotent (running the task twice won't double load data). (templated)

**class** airflow.operators.presto\_check\_operator.**PrestoValueCheckOperator** (\*\*kwargs)  
Bases: *airflow.operators.check\_operator.ValueCheckOperator*

Performs a simple value check using sql code.

#### Parameters

- **sql** (*str*) – the sql to be executed
- **presto\_conn\_id** (*str*) – reference to the Presto database

**class** airflow.operators.python\_operator.**PythonOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Executes a Python callable



**See also:**

For more information on how to use this operator, take a look at the guide: [PythonOperator](#)

**Parameters**

- **python\_callable** (*python callable*) – A reference to an object that is callable
- **op\_kwargs** (*dict*) – a dictionary of keyword arguments that will get unpacked in your function
- **op\_args** (*list*) – a list of positional arguments that will get unpacked when calling your callable
- **provide\_context** (*bool*) – if set to true, Airflow will pass a set of keyword arguments that can be used in your function. This set of kwargs correspond exactly to what you can use in your jinja templates. For this to work, you need to define `**kwargs` in your function header.
- **templates\_dict** (*dict[str]*) – a dictionary where the values are templates that will get templated by the Airflow engine sometime between `__init__` and `execute` takes place and are made available in your callable's context after the template has been applied. (templated)
- **templates\_exts** (*list[str]*) – a list of file extensions to resolve while processing templated fields, for examples `['.sql', '.hql']`

```
class airflow.operators.python_operator.PythonVirtualenvOperator (**kwargs)
```

Bases: [airflow.operators.python\\_operator.PythonOperator](#)

Allows one to run a function in a virtualenv that is created and destroyed automatically (with certain caveats).

The function must be defined using `def`, and not be part of a class. All imports must happen inside the function and no variables outside of the scope may be referenced. A global scope variable named `virtualenv_string_args` will be available (populated by `string_args`). In addition, one can pass stuff through `op_args` and `op_kwargs`, and one can use a return value. Note that if your virtualenv runs in a different Python major version than Airflow, you cannot use return values, `op_args`, or `op_kwargs`. You can use `string_args` though.

**Parameters**

- **python\_callable** (*function*) – A python function with no references to outside variables, defined with `def`, which will be run in a virtualenv
- **requirements** (*list[str]*) – A list of requirements as specified in a pip install command
- **python\_version** (*str*) – The Python version to run the virtualenv with. Note that both 2 and 2.7 are acceptable forms.
- **use\_dill** (*bool*) – Whether to use dill to serialize the args and result (pickle is default). This allow more complex types but requires you to include dill in your requirements.
- **system\_site\_packages** (*bool*) – Whether to include system\_site\_packages in your virtualenv. See virtualenv documentation for more information.
- **op\_args** – A list of positional arguments to pass to `python_callable`.
- **op\_kwargs** (*dict*) – A dict of keyword arguments to pass to `python_callable`.
- **string\_args** (*list[str]*) – Strings that are present in the global var `virtualenv_string_args`, available to `python_callable` at runtime as a `list[str]`. Note that args are split by newline.

- **templates\_dict** (*dict of str*) – a dictionary where the values are templates that will get templated by the Airflow engine sometime between `__init__` and `execute` takes place and are made available in your callable's context after the template has been applied
- **templates\_exts** (*list[str]*) – a list of file extensions to resolve while processing templated fields, for examples `['.sql', '.hql']`

**class** `airflow.operators.redshift_to_s3_operator.RedshiftToS3Transfer` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Executes an UNLOAD command to s3 as a CSV with headers

#### Parameters

- **schema** (*str*) – reference to a specific schema in redshift database
- **table** (*str*) – reference to a specific table in redshift database
- **s3\_bucket** (*str*) – reference to a specific S3 bucket
- **s3\_key** (*str*) – reference to a specific S3 key
- **redshift\_conn\_id** (*str*) – reference to a specific redshift database
- **aws\_conn\_id** (*str*) – reference to a specific S3 connection
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is `False`), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.
- **unload\_options** (*list*) – reference to a list of UNLOAD options

**class** `airflow.operators.s3_file_transform_operator.S3FileTransformOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Copies data from a source S3 location to a temporary location on the local filesystem. Runs a transformation on this file as specified by the transformation script and uploads the output to a destination S3 location.

The locations of the source and the destination files in the local filesystem is provided as an first and second arguments to the transformation script. The transformation script is expected to read the data from source, transform it and write the output to the local destination file. The operator then takes over control and uploads the local destination file to S3.

S3 Select is also available to filter the source contents. Users can omit the transformation script if S3 Select expression is specified.

#### Parameters

- **source\_s3\_key** (*str*) – The key to be retrieved from S3. (templated)
- **source\_aws\_conn\_id** (*str*) – source s3 connection
- **source\_verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is `False`), but SSL certificates will not be verified.

- **path/to/cert/bundle.pem**: A filename of the CA cert bundle to uses. You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

This is also applicable to `dest_verify`.

- **dest\_s3\_key** (*str*) – The key to be written from S3. (templated)
- **dest\_aws\_conn\_id** (*str*) – destination s3 connection
- **replace** (*bool*) – Replace dest S3 key if it already exists
- **transform\_script** (*str*) – location of the executable transformation script
- **select\_expression** (*str*) – S3 Select expression

```
class airflow.operators.s3_to_hive_operator.S3ToHiveTransfer (**kwargs)
```

Bases: `airflow.models.BaseOperator`

Moves data from S3 to Hive. The operator downloads a file from S3, stores the file locally before loading it into a Hive table. If the `create` or `recreate` arguments are set to `True`, a `CREATE TABLE` and `DROP TABLE` statements are generated. Hive data types are inferred from the cursor's metadata from.

Note that the table generated in Hive uses `STORED AS textfile` which isn't the most efficient serialization format. If a large amount of data is loaded and/or if the tables gets queried considerably, you may want to use this operator only to stage the data into a temporary table before loading it into its final destination using a `HiveOperator`.

#### Parameters

- **s3\_key** (*str*) – The key to be retrieved from S3. (templated)
- **field\_dict** (*dict*) – A dictionary of the fields name in the file as keys and their Hive types as values
- **hive\_table** (*str*) – target Hive table, use dot notation to target a specific database. (templated)
- **create** (*bool*) – whether to create the table if it doesn't exist
- **recreate** (*bool*) – whether to drop and recreate the table at every execution
- **partition** (*dict*) – target partition as a dict of partition columns and values. (templated)
- **headers** (*bool*) – whether the file contains column names on the first line
- **check\_headers** (*bool*) – whether the column names on the first line should be checked against the keys of `field_dict`
- **wildcard\_match** (*bool*) – whether the `s3_key` should be interpreted as a Unix wildcard pattern
- **delimiter** (*str*) – field delimiter in the file
- **aws\_conn\_id** (*str*) – source s3 connection
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False**: do not validate SSL certificates. SSL will still be used (unless `use_ssl` is `False`), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem**: A filename of the CA cert bundle to uses. You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

- **hive\_cli\_conn\_id** (*str*) – destination hive connection
- **input\_compressed** (*bool*) – Boolean to determine if file decompression is required to process headers
- **tblproperties** (*dict*) – TBLPROPERTIES of the hive table being created
- **select\_expression** (*str*) – S3 Select expression

**class** airflow.operators.s3\_to\_redshift\_operator.**S3ToRedshiftTransfer** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Executes an COPY command to load files from s3 to Redshift

#### Parameters

- **schema** (*str*) – reference to a specific schema in redshift database
- **table** (*str*) – reference to a specific table in redshift database
- **s3\_bucket** (*str*) – reference to a specific S3 bucket
- **s3\_key** (*str*) – reference to a specific S3 key
- **redshift\_conn\_id** (*str*) – reference to a specific redshift database
- **aws\_conn\_id** (*str*) – reference to a specific S3 connection
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is False), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.
- **copy\_options** (*list*) – reference to a list of COPY options

**class** airflow.operators.python\_operator.**ShortCircuitOperator** (\*\*kwargs)  
 Bases: *airflow.operators.python\_operator.PythonOperator*, *airflow.models.SkipMixin*

Allows a workflow to continue only if a condition is met. Otherwise, the workflow “short-circuits” and downstream tasks are skipped.

The ShortCircuitOperator is derived from the PythonOperator. It evaluates a condition and short-circuits the workflow if the condition is False. Any downstream tasks are marked with a state of “skipped”. If the condition is True, downstream tasks proceed as normal.

The condition is determined by the result of *python\_callable*.

**class** airflow.operators.http\_operator.**SimpleHttpOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Calls an endpoint on an HTTP system to execute an action

#### Parameters

- **http\_conn\_id** (*str*) – The connection to run the operator against
- **endpoint** (*str*) – The relative part of the full url. (templated)
- **method** (*str*) – The HTTP method to use, default = “POST”

- **data** (For POST/PUT, depends on the content-type parameter, for GET a dictionary of key/value string pairs) – The data to pass. POST-data in POST/PUT and params in the URL for a GET request. (templated)
- **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.)
- **xcom\_push** (bool) – Push the response to Xcom (default: False). If xcom\_push is True, response of an HTTP request will also be pushed to an XCom.
- **log\_response** (bool) – Log the response (default: False)

**class** airflow.operators.slack\_operator.**SlackAPIOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Base Slack Operator The SlackAPIPostOperator is derived from this operator. In the future additional Slack API Operators will be derived from this class as well

#### Parameters

- **slack\_conn\_id** (str) – Slack connection ID which its password is Slack API token
- **token** (str) – Slack API token (<https://api.slack.com/web>)
- **method** (str) – The Slack API Method to Call (<https://api.slack.com/methods>)
- **api\_params** (dict) – API Method call parameters (<https://api.slack.com/methods>)

**construct\_api\_call\_params** ()

Used by the execute function. Allows templating on the source fields of the api\_call\_params dict before construction

Override in child classes. Each SlackAPIOperator child class is responsible for having a construct\_api\_call\_params function which sets self.api\_call\_params with a dict of API call parameters (<https://api.slack.com/methods>)

**execute** (\*\*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.operators.slack\_operator.**SlackAPIPostOperator** (\*\*kwargs)

Bases: *airflow.operators.slack\_operator.SlackAPIOperator*

Posts messages to a slack channel

#### Parameters

- **channel** (str) – channel in which to post message on slack name (#general) or ID (C12318391). (templated)
- **username** (str) – Username that airflow will be posting to Slack as. (templated)
- **text** (str) – message to send to slack. (templated)
- **icon\_url** (str) – url to icon used for this message

- **attachments** (*array of hashes*) – extra formatting details. (templated) - see <https://api.slack.com/docs/attachments>.

**construct\_api\_call\_params** ()

Used by the execute function. Allows templating on the source fields of the api\_call\_params dict before construction

Override in child classes. Each SlackAPIOperator child class is responsible for having a construct\_api\_call\_params function which sets self.api\_call\_params with a dict of API call parameters (<https://api.slack.com/methods>)

**class** airflow.operators.sqlite\_operator.**SqliteOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific Sqlite database

#### Parameters

- **sql** (*str or string pointing to a template file. File must have a '.sql' extensions.*) – the sql code to be executed. (templated)
- **sqlite\_conn\_id** (*str*) – reference to a specific sqlite database
- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.

**class** airflow.operators.subdag\_operator.**SubDagOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

This runs a sub dag. By convention, a sub dag's dag\_id should be prefixed by its parent and a dot. As in *parent.child*.

#### Parameters

- **subdag** (*airflow.models.DAG*) – the DAG object to run as a subdag of the current DAG.
- **dag** (*airflow.models.DAG*) – the parent DAG for the subdag.
- **executor** (*airflow.executors.base\_executor.BaseExecutor*) – the executor for this subdag. Default to use SequentialExecutor. Please find AIRFLOW-74 for more details.

**class** airflow.operators.dagrun\_operator.**TriggerDagRunOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Triggers a DAG run for a specified dag\_id

#### Parameters

- **trigger\_dag\_id** (*str*) – the dag\_id to trigger (templated)
- **python\_callable** (*python callable*) – a reference to a python function that will be called while passing it the context object and a placeholder object obj for your callable to fill and return if you want a DagRun created. This obj object contains a run\_id and payload attribute that you can modify in your function. The run\_id should be a unique identifier for that DAG run, and the payload has to be a pickleable object that will be made available to your tasks while executing that DAG run. Your function header should look like `def foo(context, dag_run_obj):`
- **execution\_date** (*str or datetime.datetime*) – Execution date for the dag (templated)

```
class airflow.operators.check_operator.ValueCheckOperator (**kwargs)
```

Bases: `airflow.models.BaseOperator`

Performs a simple value check using sql code.

Note that this is an abstract class and `get_db_hook` needs to be defined. Whereas a `get_db_hook` is hook that gets a single record from an external source.

**Parameters** `sql` (*str*) – the sql to be executed. (templated)

## Sensors

```
class airflow.sensors.external_task_sensor.ExternalTaskSensor (**kwargs)
```

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a different DAG or a task in a different DAG to complete for a specific `execution_date`

### Parameters

- **external\_dag\_id** (*str*) – The `dag_id` that contains the task you want to wait for
- **external\_task\_id** (*str*) – The `task_id` that contains the task you want to wait for. If None the sensor waits for the DAG
- **allowed\_states** (*list*) – list of allowed states, default is ['success']
- **execution\_delta** (*datetime.timedelta*) – time difference with the previous execution to look at, the default is the same `execution_date` as the current task or DAG. For yesterday, use [positive!] `datetime.timedelta(days=1)`. Either `execution_delta` or `execution_date_fn` can be passed to `ExternalTaskSensor`, but not both.
- **execution\_date\_fn** (*callable*) – function that receives the current execution date and returns the desired execution dates to query. Either `execution_delta` or `execution_date_fn` can be passed to `ExternalTaskSensor`, but not both.
- **check\_existence** (*bool*) – Set to `True` to check if the external task exists (when `external_task_id` is not None) or check if the DAG to wait for exists (when `external_task_id` is None), and immediately cease waiting if the external task or DAG does not exist (default value: `False`).

```
poke (**kwargs)
```

Function that the sensors defined while deriving this class should override.

```
class airflow.sensors.hive_partition_sensor.HivePartitionSensor (**kwargs)
```

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a partition to show up in Hive.

Note: Because `partition` supports general logical operators, it can be inefficient. Consider using `NamedHivePartitionSensor` instead if you don't need the full flexibility of `HivePartitionSensor`.

### Parameters

- **table** (*str*) – The name of the table to wait for, supports the dot notation (`my_database.my_table`)
- **partition** (*str*) – The partition clause to wait for. This is passed as is to the metastore Thrift client `get_partitions_by_filter` method, and apparently supports SQL like notation as in `ds='2015-01-01' AND type='value'` and comparison operators as in `"ds>=2015-01-01"`
- **metastore\_conn\_id** (*str*) – reference to the metastore thrift service connection id



**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.http_sensor.HttpSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Executes a HTTP GET statement and returns False on failure caused by 404 Not Found or *response\_check* returning False.

HTTP Error codes other than 404 (like 403) or Connection Refused Error would fail the sensor itself directly (no more poking).

#### Parameters

- **http\_conn\_id** (*str*) – The connection to run the sensor against
- **method** (*str*) – The HTTP request method to use
- **endpoint** (*str*) – The relative part of the full url
- **request\_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.)

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.metastore_partition_sensor.MetastorePartitionSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.sql_sensor.SqlSensor`

An alternative to the HivePartitionSensor that talk directly to the MySQL db. This was created as a result of observing sub optimal queries generated by the Metastore thrift service when hitting subpartitioned tables. The Thrift service’s queries were written in a way that wouldn’t leverage the indexes.

#### Parameters

- **schema** (*str*) – the schema
- **table** (*str*) – the table
- **partition\_name** (*str*) – the partition name, as defined in the PARTITIONS table of the Metastore. Order of the fields does matter. Examples: `ds=2016-01-01` or `ds=2016-01-01/sub=foo` for a sub partitioned table
- **mysql\_conn\_id** (*str*) – a reference to the MySQL conn\_id for the metastore

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.named_hive_partition_sensor.NamedHivePartitionSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a set of partitions to show up in Hive.

#### Parameters



- **partition\_names** (*list[str]*) – List of fully qualified names of the partitions to wait for. A fully qualified name is of the form `schema.table/pk1=pv1/pk2=pv2`, for example, `default.users/ds=2016-01-01`. This is passed as is to the metastore Thrift client `get_partitions_by_name` method. Note that you cannot use logical or comparison operators as in `HivePartitionSensor`.
- **metastore\_conn\_id** (*str*) – reference to the metastore thrift service connection id

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.s3_key_sensor.S3KeySensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a key (a file-like instance on S3) to be present in a S3 bucket. S3 being a key/value it does not support folders. The path is just a key a resource.

#### Parameters

- **bucket\_key** (*str*) – The key being waited on. Supports full `s3://` style url or relative path from root level. When it's specified as a full `s3://` url, please leave `bucket_name` as *None*.
- **bucket\_name** (*str*) – Name of the S3 bucket. Only needed when `bucket_key` is not provided as a full `s3://` url.
- **wildcard\_match** (*bool*) – whether the `bucket_key` should be interpreted as a Unix wildcard pattern
- **aws\_conn\_id** (*str*) – a reference to the s3 connection
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is *False*), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.s3_prefix_sensor.S3PrefixSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a prefix to exist. A prefix is the first part of a key, thus enabling checking of constructs similar to `glob airfl*` or SQL LIKE `'airfl%'`. There is the possibility to precise a delimiter to indicate the hierarchy or keys, meaning that the match will stop at that delimiter. Current code accepts sane delimiters, i.e. characters that are NOT special characters in the Python regex engine.

#### Parameters

- **bucket\_name** (*str*) – Name of the S3 bucket
- **prefix** (*str*) – The prefix being waited on. Relative path from bucket root level.
- **delimiter** (*str*) – The delimiter intended to show hierarchy. Defaults to `'/'`.
- **aws\_conn\_id** (*str*) – a reference to the s3 connection
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:

- **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is False), but SSL certificates will not be verified.
- **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.sql_sensor.SqlSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Runs a sql statement until a criteria is met. It will keep trying while sql returns no row, or if the first cell in (0, '0', '').

#### Parameters

- **conn\_id** (*str*) – The connection to run the sensor against
- **sql** (*str*) – The sql to run. To pass, it needs to return at least one cell that contains a non-zero / empty string value.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.time_delta_sensor.TimeDeltaSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a timedelta after the task's execution\_date + schedule\_interval. In Airflow, the daily task stamped with execution\_date 2016-01-01 can only start running on 2016-01-02. The timedelta here represents the time after the execution period has closed.

**Parameters** **delta** (*datetime.timedelta*) – time length to wait after execution\_date before succeeding

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.time_sensor.TimeSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits until the specified time of the day.

**Parameters** **target\_time** (*datetime.time*) – time after which the job succeeds

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.sensors.web_hdfs_sensor.WebHdfsSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a file or folder to land in HDFS

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

### 3.21.1.4 Community-contributed Operators

## Operators

**class** `airflow.contrib.operators.aws_athena_operator.AWSAthenaOperator` (\*\*kwargs)  
 Bases: `airflow.models.BaseOperator`

An operator that submit presto query to athena.

### Parameters

- **query** (*str*) – Presto to be run on athena. (templated)
- **database** (*str*) – Database to select. (templated)
- **output\_location** (*str*) – s3 path to write the query results into. (templated)
- **aws\_conn\_id** (*str*) – aws connection to use
- **sleep\_time** (*int*) – Time to wait between two consecutive call to check query status on athena

**execute** (*context*)

Run Presto Query on Athena

**on\_kill** ()

Cancel the submitted athena query

**class** `airflow.contrib.operators.awsbatch_operator.AWSBatchOperator` (\*\*kwargs)  
 Bases: `airflow.models.BaseOperator`

Execute a job on AWS Batch Service

### Parameters

- **job\_name** (*str*) – the name for the job that will run on AWS Batch (templated)
- **job\_definition** (*str*) – the job definition name on AWS Batch
- **job\_queue** (*str*) – the queue name on AWS Batch
- **overrides** (*dict*) – the same parameter that boto3 will receive on containerOverrides (templated): [http://boto3.readthedocs.io/en/latest/reference/services/batch.html#submit\\_job](http://boto3.readthedocs.io/en/latest/reference/services/batch.html#submit_job)
- **max\_retries** (*int*) – exponential backoff retries while waiter is not merged, 4200 = 48 hours
- **aws\_conn\_id** (*str*) – connection id of AWS credentials / region name. If None, credential boto3 strategy will be used (<http://boto3.readthedocs.io/en/latest/guide/configuration.html>).
- **region\_name** (*str*) – region name to use in AWS Hook. Override the region\_name in connection (if provided)

**class** `airflow.contrib.operators.adls_to_gcs.AdlsToGoogleCloudStorageOperator` (\*\*kwargs)  
 Bases: `airflow.contrib.operators.adls_list_operator.AzureDataLakeStorageListOperator`

Synchronizes an Azure Data Lake Storage path with a GCS bucket

### Parameters

- **src\_adls** (*str*) – The Azure Data Lake path to find the objects (templated)
- **dest\_gcs** (*str*) – The Google Cloud Storage bucket and prefix to store the objects. (templated)
- **replace** (*bool*) – If true, replaces same-named files in GCS

- **azure\_data\_lake\_conn\_id** (*str*) – The connection ID to use when connecting to Azure Data Lake Storage.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google Cloud Storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**Examples:** The following Operator would copy a single file named `hello/world.avro` from ADLS to the GCS bucket `mybucket`. Its full resulting gcs path will be `gs://mybucket/hello/world.avro`

```
copy_single_file = AdlsToGoogleCloudStorageOperator(
    task_id='copy_single_file',
    src_adls='hello/world.avro',
    dest_gcs='gs://mybucket',
    replace=False,
    azure_data_lake_conn_id='azure_data_lake_default',
    google_cloud_storage_conn_id='google_cloud_default'
)
```

The following Operator would copy all parquet files from ADLS to the GCS bucket `mybucket`.

```
copy_all_files = AdlsToGoogleCloudStorageOperator(
    task_id='copy_all_files',
    src_adls='*.parquet',
    dest_gcs='gs://mybucket',
    replace=False,
    azure_data_lake_conn_id='azure_data_lake_default',
    google_cloud_storage_conn_id='google_cloud_default'
)
```

The following Operator would copy all parquet files from ADLS path ```/hello/world``` to the GCS bucket ```mybucket```. ::

```
copy_world_files = AdlsToGoogleCloudStorageOperator(
    task_id='copy_world_files',
    src_adls='hello/world/*.parquet',
    dest_gcs='gs://mybucket',
    replace=False,
    azure_data_lake_conn_id='azure_data_lake_default',
    google_cloud_storage_conn_id='google_cloud_default'
)
```

**class** `airflow.contrib.operators.azure_container_instances_operator.AzureContainerInstancesOperator`

Bases: `airflow.models.BaseOperator`

Start a container on Azure Container Instances

#### Parameters

- **ci\_conn\_id** (*str*) – connection id of a service principal which will be used to start the container instance
- **registry\_conn\_id** (*str*) – connection id of a user which can login to a private docker registry. If None, we assume a public registry
- **resource\_group** (*str*) – name of the resource group wherein this container instance should be started

- **name** (*str*) – name of this container instance. Please note this name has to be unique in order to run containers in parallel.
- **image** (*str*) – the docker image to be used
- **region** (*str*) – the region wherein this container instance should be started
- **environment\_variables** (*dict*) – key,value pairs containing environment variables which will be passed to the running container
- **volumes** (*list*[<conn\_id, account\_name, share\_name, mount\_path, read\_only>]) – list of volumes to be mounted to the container. Currently only Azure Fileshares are supported.
- **memory\_in\_gb** (*double*) – the amount of memory to allocate to this container
- **cpu** (*double*) – the number of cpus to allocate to this container
- **command** (*str*) – the command to run inside the container

### Example

```
>>> a = AzureContainerInstancesOperator(
    'azure_service_principal',
    'azure_registry_user',
    'my-resource-group',
    'my-container-name-{{ ds }}',
    'myprivateregistry.azurecr.io/my_container:latest',
    'westeurope',
    {'EXECUTION_DATE': '{{ ds }}'},
    [('azure_wasb_conn_id',
     'my_storage_container',
     'my_fileshare',
     '/input-data',
     True), ],
    memory_in_gb=14.0,
    cpu=4.0,
    command='python /app/myfile.py',
    task_id='start_container'
)
```

**class** airflow.contrib.operators.azure\_cosmos\_operator.**AzureCosmosInsertDocumentOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Inserts a new document into the specified Cosmos database and collection It will create both the database and collection if they do not already exist

### Parameters

- **database\_name** (*str*) – The name of the database. (templated)
- **collection\_name** (*str*) – The name of the collection. (templated)
- **document** (*dict*) – The document to insert
- **azure\_cosmos\_conn\_id** (*str*) – reference to a CosmosDB connection.

**class** airflow.contrib.operators.adls\_list\_operator.**AzureDataLakeStorageListOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

List all files from the specified path

This operator returns a python list with the names of files which can be used by *xcom* in the downstream tasks.

**Parameters**

- **path** (*str*) – The Azure Data Lake path to find the objects. Supports glob strings (templated)
- **azure\_data\_lake\_conn\_id** (*str*) – The connection ID to use when connecting to Azure Data Lake Storage.

**Example:** The following Operator would list all the Parquet files from `folder/output/` folder in the specified ADLS account

```
adls_files = AzureDataLakeStorageListOperator(
    task_id='adls_files',
    path='folder/output/*.parquet',
    azure_data_lake_conn_id='azure_data_lake_default'
)
```

**class** airflow.contrib.operators.bigquery\_check\_operator.**BigQueryCheckOperator** (\*\*kwargs)  
Bases: *airflow.operators.check\_operator.CheckOperator*

Performs checks against BigQuery. The `BigQueryCheckOperator` expects a sql query that will return a single row. Each value on that first row is evaluated using python bool casting. If any of the values return False the check is failed and errors out.

Note that Python bool casting evals the following as False:

- False
- 0
- Empty string ("")
- Empty list ([])
- Empty dictionary or set ({})

Given a query like `SELECT COUNT(*) FROM foo`, it will fail only if the count == 0. You can craft much more complex query that could, for instance, check that the table has the same number of rows as the source table upstream, or that the count of today's partition is greater than yesterday's partition, or that a set of metrics are less than 3 standard deviation for the 7 day average.

This operator can be used as a data quality check in your pipeline, and depending on where you put it in your DAG, you have the choice to stop the critical path, preventing from publishing dubious data, or on the side and receive email alerts without stopping the progress of the DAG.

**Parameters**

- **sql** (*str*) – the sql to be executed
- **bigquery\_conn\_id** (*str*) – reference to the BigQuery database
- **use\_legacy\_sql** (*bool*) – Whether to use legacy SQL (true) or standard SQL (false).

**class** airflow.contrib.operators.bigquery\_operator.**BigQueryCreateEmptyDatasetOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

This operator is used to create new dataset for your Project in Big query. <https://cloud.google.com/bigquery/docs/reference/rest/v2/datasets#resource>

**Parameters**

- **project\_id** (*str*) – The name of the project where we want to create the dataset. Don't need to provide, if projectId in dataset\_reference.

- **dataset\_id** (*str*) – The id of dataset. Don't need to provide, if datasetId in dataset\_reference.
- **dataset\_reference** – Dataset reference that could be provided with request body. More info: <https://cloud.google.com/bigquery/docs/reference/rest/v2/datasets#resource>

**class** airflow.contrib.operators.bigquery\_operator.**BigQueryCreateEmptyTableOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Creates a new, empty table in the specified BigQuery dataset, optionally with schema.

The schema to be used for the BigQuery table may be specified in one of two ways. You may either directly pass the schema fields in, or you may point the operator to a Google cloud storage object name. The object in Google cloud storage must be a JSON file with the schema fields in it. You can also create a table without schema.

#### Parameters

- **project\_id** (*str*) – The project to create the table into. (templated)
- **dataset\_id** (*str*) – The dataset to create the table into. (templated)
- **table\_id** (*str*) – The Name of the table to be created. (templated)
- **schema\_fields** (*list*) – If set, the schema field list as defined here: <https://cloud.google.com/bigquery/docs/reference/rest/v2/jobs#configuration.load.schema>

#### Example:

```
schema_fields=[{"name": "emp_name", "type": "STRING", "mode":
↪ "REQUIRED"},
                {"name": "salary", "type": "INTEGER", "mode":
↪ "NULLABLE"}]
```

- **gcs\_schema\_object** (*str*) – Full path to the JSON file containing schema (templated). For example: gs://test-bucket/dir1/dir2/employee\_schema.json
- **time\_partitioning** (*dict*) – configure optional time partitioning fields i.e. partition by field, type and expiration as per API specifications.

#### See also:

<https://cloud.google.com/bigquery/docs/reference/rest/v2/tables#timePartitioning>

- **bigquery\_conn\_id** (*str*) – Reference to a specific BigQuery hook.
- **google\_cloud\_storage\_conn\_id** (*str*) – Reference to a specific Google cloud storage hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **labels** (*dict*) – a dictionary containing labels for the table, passed to BigQuery

#### Example (with schema JSON in GCS):

```
CreateTable = BigQueryCreateEmptyTableOperator(
    task_id='BigQueryCreateEmptyTableOperator_task',
    dataset_id='ODS',
    table_id='Employees',
    project_id='internal-gcp-project',
    gcs_schema_object='gs://schema-bucket/employee_schema.json',
    bigquery_conn_id='airflow-service-account',
    google_cloud_storage_conn_id='airflow-service-account'
)
```

Corresponding Schema file (`employee_schema.json`):

```
[
  {
    "mode": "NULLABLE",
    "name": "emp_name",
    "type": "STRING"
  },
  {
    "mode": "REQUIRED",
    "name": "salary",
    "type": "INTEGER"
  }
]
```

Example (with schema in the DAG):

```
CreateTable = BigQueryCreateEmptyTableOperator(
    task_id='BigQueryCreateEmptyTableOperator_task',
    dataset_id='ODS',
    table_id='Employees',
    project_id='internal-gcp-project',
    schema_fields=[{"name": "emp_name", "type": "STRING", "mode":
↪ "REQUIRED"},
                    {"name": "salary", "type": "INTEGER", "mode":
↪ "NULLABLE"}],
    bigquery_conn_id='airflow-service-account',
    google_cloud_storage_conn_id='airflow-service-account'
)
```

**class** `airflow.contrib.operators.bigquery_operator.BigQueryCreateExternalTableOperator` (\*\*kwargs)  
 Bases: `airflow.models.BaseOperator`

Creates a new external table in the dataset with the data in Google Cloud Storage.

The schema to be used for the BigQuery table may be specified in one of two ways. You may either directly pass the schema fields in, or you may point the operator to a Google cloud storage object name. The object in Google cloud storage must be a JSON file with the schema fields in it.

#### Parameters

- **bucket** (*str*) – The bucket to point the external table to. (templated)
- **source\_objects** (*list*) – List of Google cloud storage URIs to point table to. (templated) If `source_format` is `'DATASTORE_BACKUP'`, the list must only contain a single URI.
- **destination\_project\_dataset\_table** (*str*) – The dotted (<project>.<dataset>.<table>) BigQuery table to load data into (templated). If <project> is not included, project will be the project defined in the connection json.
- **schema\_fields** (*list*) – If set, the schema field list as defined here: <https://cloud.google.com/bigquery/docs/reference/rest/v2/jobs#configuration.load.schema>

Example:

```
schema_fields=[{"name": "emp_name", "type": "STRING", "mode":
↪ "REQUIRED"},
                {"name": "salary", "type": "INTEGER", "mode":
↪ "NULLABLE"}]
```



Should not be set when `source_format` is 'DATASTORE\_BACKUP'.

- **schema\_object** (*str*) – If set, a GCS object path pointing to a .json file that contains the schema for the table. (templated)
- **source\_format** (*str*) – File format of the data.
- **compression** (*str*) – [Optional] The compression type of the data source. Possible values include GZIP and NONE. The default value is NONE. This setting is ignored for Google Cloud Bigtable, Google Cloud Datastore backups and Avro formats.
- **skip\_leading\_rows** (*int*) – Number of rows to skip when loading from a CSV.
- **field\_delimiter** (*str*) – The delimiter to use for the CSV.
- **max\_bad\_records** (*int*) – The maximum number of bad records that BigQuery can ignore when running the job.
- **quote\_character** (*str*) – The value that is used to quote data sections in a CSV file.
- **allow\_quoted\_newlines** (*bool*) – Whether to allow quoted newlines (true) or not (false).
- **allow\_jagged\_rows** (*bool*) – Accept rows that are missing trailing optional columns. The missing values are treated as nulls. If false, records with missing trailing columns are treated as bad records, and if there are too many bad records, an invalid error is returned in the job result. Only applicable to CSV, ignored for other formats.
- **bigquery\_conn\_id** (*str*) – Reference to a specific BigQuery hook.
- **google\_cloud\_storage\_conn\_id** (*str*) – Reference to a specific Google cloud storage hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **src\_fmt\_configs** (*dict*) – configure optional fields specific to the source format
- **labels** (*dict*) – a dictionary containing labels for the table, passed to BigQuery

```
class airflow.contrib.operators.bigquery_operator.BigQueryDeleteDatasetOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

This operator deletes an existing dataset from your Project in Big query. <https://cloud.google.com/bigquery/docs/reference/rest/v2/datasets/delete>

#### Parameters

- **project\_id** (*str*) – The project id of the dataset.
- **dataset\_id** (*str*) – The dataset to be deleted.

#### Example:

```
delete_temp_data = BigQueryDeleteDatasetOperator(dataset_id = 'temp-dataset',
                                                  project_id = 'temp-project',
                                                  bigquery_conn_id='_my_gcp_conn_',
                                                  task_id='Deletetemp',
                                                  dag=dag)
```

```
class airflow.contrib.operators.bigquery_get_data.BigQueryGetDataOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Fetches the data from a BigQuery table (alternatively fetch data for selected columns) and returns data in a python list. The number of elements in the returned list will be equal to the number of rows fetched. Each element in the list will again be a list where element would represent the columns values for that row.

**Example Result:** `[['Tony', '10'], ['Mike', '20'], ['Steve', '15']]`

**Note:** If you pass fields to `selected_fields` which are in different order than the order of columns already in BQ table, the data will still be in the order of BQ table. For example if the BQ table has 3 columns as `[A, B, C]` and you pass `'B,A'` in the `selected_fields` the data would still be of the form `'A, B'`.

**Example:**

```
get_data = BigQueryGetDataOperator(
    task_id='get_data_from_bq',
    dataset_id='test_dataset',
    table_id='Transaction_partitions',
    max_results='100',
    selected_fields='DATE',
    bigquery_conn_id='airflow-service-account'
)
```

#### Parameters

- **dataset\_id** (*str*) – The dataset ID of the requested table. (templated)
- **table\_id** (*str*) – The table ID of the requested table. (templated)
- **max\_results** (*str*) – The maximum number of records (rows) to be fetched from the table. (templated)
- **selected\_fields** (*str*) – List of fields to return (comma-separated). If unspecified, all fields are returned.
- **bigquery\_conn\_id** (*str*) – reference to a specific BigQuery hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**class** `airflow.contrib.operators.bigquery_check_operator.BigQueryIntervalCheckOperator` (\*\*kwargs)  
 Bases: `airflow.operators.check_operator.IntervalCheckOperator`

Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from `days_back` before.

This method constructs a query like so

```
SELECT {metrics_threshold_dict_key} FROM {table}
WHERE {date_filter_column}=<date>
```

#### Parameters

- **table** (*str*) – the table name
- **days\_back** (*int*) – number of days between ds and the ds we want to check against. Defaults to 7 days
- **metrics\_threshold** (*dict*) – a dictionary of ratios indexed by metrics, for example `'COUNT(*)': 1.5` would require a 50 percent or less difference between the current day, and the prior `days_back`.

- **use\_legacy\_sql** (*bool*) – Whether to use legacy SQL (true) or standard SQL (false).

**class** airflow.contrib.operators.bigquery\_operator.**BigQueryOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Executes BigQuery SQL queries in a specific BigQuery database

#### Parameters

- **sql** (Can receive a *str* representing a sql statement, a list of *str* (sql statements), or reference to a template file. Template reference are recognized by *str* ending in *'.sql'*.) – the sql code to be executed (templated)
- **destination\_dataset\_table** (*str*) – A dotted (<project>.<project>.<dataset>.<table> that, if set, will store the results of the query. (templated)
- **write\_disposition** (*str*) – Specifies the action that occurs if the destination table already exists. (default: 'WRITE\_EMPTY')
- **create\_disposition** (*str*) – Specifies whether the job is allowed to create new tables. (default: 'CREATE\_IF\_NEEDED')
- **allow\_large\_results** (*bool*) – Whether to allow large results.
- **flatten\_results** (*bool*) – If true and query uses legacy SQL dialect, flattens all nested and repeated fields in the query results. *allow\_large\_results* must be true if this is set to false. For standard SQL queries, this flag is ignored and results are never flattened.
- **bigquery\_conn\_id** (*str*) – reference to a specific BigQuery hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **udf\_config** (*list*) – The User Defined Function configuration for the query. See <https://cloud.google.com/bigquery/user-defined-functions> for details.
- **use\_legacy\_sql** (*bool*) – Whether to use legacy SQL (true) or standard SQL (false).
- **maximum\_billing\_tier** (*int*) – Positive integer that serves as a multiplier of the basic price. Defaults to None, in which case it uses the value set in the project.
- **maximum\_bytes\_billed** (*float*) – Limits the bytes billed for this job. Queries that will have bytes billed beyond this limit will fail (without incurring a charge). If unspecified, this will be set to your project default.
- **api\_resource\_configs** (*dict*) – a dictionary that contain params 'configuration' applied for Google BigQuery Jobs API: <https://cloud.google.com/bigquery/docs/reference/rest/v2/jobs> for example, {'query': {'useQueryCache': False}}. You could use it if you need to provide some params that are not supported by BigQueryOperator like args.
- **schema\_update\_options** (*tuple*) – Allows the schema of the destination table to be updated as a side effect of the load job.
- **query\_params** (*dict*) – a dictionary containing query parameter types and values, passed to BigQuery.
- **labels** (*dict*) – a dictionary containing labels for the job/query, passed to BigQuery
- **priority** (*str*) – Specifies a priority for the query. Possible values include INTERACTIVE and BATCH. The default value is INTERACTIVE.

- **time\_partitioning** (*dict*) – configure optional time partitioning fields i.e. partition by field, type and expiration as per API specifications.
- **cluster\_fields** (*list[str]*) – Request that the result of this query be stored sorted by one or more columns. This is only available in conjunction with time\_partitioning. The order of columns given determines the sort order.
- **location** (*str*) – The geographic location of the job. Required except for US and EU. See details at [https://cloud.google.com/bigquery/docs/locations#specifying\\_your\\_location](https://cloud.google.com/bigquery/docs/locations#specifying_your_location)

**class** airflow.contrib.operators.bigquery\_table\_delete\_operator.**BigQueryTableDeleteOperator**  
 Bases: *airflow.models.BaseOperator*

Deletes BigQuery tables

#### Parameters

- **deletion\_dataset\_table** (*str*) – A dotted (<project>.<project>.<dataset>.<table>) that indicates which table will be deleted. (templated)
- **bigquery\_conn\_id** (*str*) – reference to a specific BigQuery hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **ignore\_if\_missing** (*bool*) – if True, then return success even if the requested table does not exist.

**class** airflow.contrib.operators.bigquery\_to\_bigquery.**BigQueryToBigQueryOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Copies data from one BigQuery table to another.

See also:

For more details about these parameters: <https://cloud.google.com/bigquery/docs/reference/v2/jobs#configuration.copy>

#### Parameters

- **source\_project\_dataset\_tables** (*list/string*) – One or more dotted (project:|project.)<dataset>.<table> BigQuery tables to use as the source data. If <project> is not included, project will be the project defined in the connection json. Use a list if there are multiple source tables. (templated)
- **destination\_project\_dataset\_table** (*str*) – The destination BigQuery table. Format is: (project:|project.)<dataset>.<table> (templated)
- **write\_disposition** (*str*) – The write disposition if the table already exists.
- **create\_disposition** (*str*) – The create disposition if the table doesn't exist.
- **bigquery\_conn\_id** (*str*) – reference to a specific BigQuery hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **labels** (*dict*) – a dictionary containing labels for the job/query, passed to BigQuery

**class** airflow.contrib.operators.bigquery\_to\_gcs.**BigQueryToCloudStorageOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Transfers a BigQuery table to a Google Cloud Storage bucket.

See also:

For more details about these parameters: <https://cloud.google.com/bigquery/docs/reference/v2/jobs>

### Parameters

- **source\_project\_dataset\_table** (*str*) – The dotted (<project>.|<project>:.)<dataset>.<table> BigQuery table to use as the source data. If <project> is not included, project will be the project defined in the connection json. (templated)
- **destination\_cloud\_storage\_uris** (*list*) – The destination Google Cloud Storage URI (e.g. gs://some-bucket/some-file.txt). (templated) Follows convention defined here: <https://cloud.google.com/bigquery/exporting-data-from-bigquery#exportingmultiple>
- **compression** (*str*) – Type of compression to use.
- **export\_format** (*str*) – File format to export.
- **field\_delimiter** (*str*) – The delimiter to use when extracting to a CSV.
- **print\_header** (*bool*) – Whether to print a header for a CSV file extract.
- **bigquery\_conn\_id** (*str*) – reference to a specific BigQuery hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **labels** (*dict*) – a dictionary containing labels for the job/query, passed to BigQuery

**class** airflow.contrib.operators.bigquery\_check\_operator.**BigQueryValueCheckOperator** (\*\*kwargs)  
 Bases: *airflow.operators.check\_operator.ValueCheckOperator*

Performs a simple value check using sql code.

### Parameters

- **sql** (*str*) – the sql to be executed
- **use\_legacy\_sql** (*bool*) – Whether to use legacy SQL (true) or standard SQL (false).

**class** airflow.contrib.operators.gcp\_bigtable\_operator.**BigtableClusterUpdateOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*, *airflow.contrib.operators.gcp\_bigtable\_operator.BigtableValidationMixin*

Updates a Cloud Bigtable cluster.

For more details about updating a Cloud Bigtable cluster, have a look at the reference: <https://googleapis.github.io/google-cloud-python/latest/bigtable/cluster.html#google.cloud.bigtable.cluster.Cluster.update>

See also:

For more information on how to use this operator, take a look at the guide: *BigtableClusterUpdateOperator*

### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.
- **cluster\_id** (*str*) – The ID of the Cloud Bigtable cluster to update.
- **nodes** (*int*) – The desired number of nodes for the Cloud Bigtable cluster.
- **project\_id** (*str*) – Optional, the ID of the GCP project.

```
class airflow.contrib.operators.gcp_bigtable_operator.BigtableInstanceCreateOperator (**kwargs)
    Bases: airflow.models.BaseOperator, airflow.contrib.operators.gcp\_bigtable\_operator.BigtableValidationMixin
```

Creates a new Cloud Bigtable instance. If the Cloud Bigtable instance with the given ID exists, the operator does not compare its configuration and immediately succeeds. No changes are made to the existing instance.

For more details about instance creation have a look at the reference: <https://googleapis.github.io/google-cloud-python/latest/bigtable/instance.html#google.cloud.bigtable.instance.Instance.create>

**See also:**

For more information on how to use this operator, take a look at the guide: [BigtableInstanceCreateOperator](#)

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance to create.
- **main\_cluster\_id** (*str*) – The ID for main cluster for the new instance.
- **main\_cluster\_zone** (*str*) – The zone for main cluster See <https://cloud.google.com/bigtable/docs/locations> for more details.
- **project\_id** (*str*) – Optional, the ID of the GCP project. If set to None or missing, the default project\_id from the GCP connection is used.
- **replica\_cluster\_id** (*str*) – (optional) The ID for replica cluster for the new instance.
- **replica\_cluster\_zone** (*str*) – (optional) The zone for replica cluster.
- **instance\_type** (*enums.IntEnum*) – (optional) The type of the instance.
- **instance\_display\_name** (*str*) – (optional) Human-readable name of the instance. Defaults to instance\_id.
- **instance\_labels** (*dict*) – (optional) Dictionary of labels to associate with the instance.
- **cluster\_nodes** (*int*) – (optional) Number of nodes for cluster.
- **cluster\_storage\_type** (*enums.IntEnum*) – (optional) The type of storage.
- **timeout** (*int*) – (optional) timeout (in seconds) for instance creation. If None is not specified, Operator will wait indefinitely.

```
class airflow.contrib.operators.gcp_bigtable_operator.BigtableInstanceDeleteOperator (**kwargs)
    Bases: airflow.models.BaseOperator, airflow.contrib.operators.gcp\_bigtable\_operator.BigtableValidationMixin
```

Deletes the Cloud Bigtable instance, including its clusters and all related tables.

For more details about deleting instance have a look at the reference: <https://googleapis.github.io/google-cloud-python/latest/bigtable/instance.html#google.cloud.bigtable.instance.Instance.delete>

**See also:**

For more information on how to use this operator, take a look at the guide: [BigtableInstanceDeleteOperator](#)

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance to delete.
- **project\_id** (*str*) – Optional, the ID of the GCP project. If set to None or missing, the default project\_id from the GCP connection is used.

```
class airflow.contrib.operators.gcp_bigtable_operator.BigtableTableCreateOperator (**kwargs)
    Bases: airflow.models.BaseOperator, airflow.contrib.operators.gcp_bigtable_operator.BigtableValidationMixin
```

Creates the table in the Cloud Bigtable instance.

For more details about creating table have a look at the reference: <https://googleapis.github.io/google-cloud-python/latest/bigtable/table.html#google.cloud.bigtable.table.Table.create>

**See also:**

For more information on how to use this operator, take a look at the guide: [BigtableTableCreateOperator](#)

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance that will hold the new table.
- **table\_id** (*str*) – The ID of the table to be created.
- **project\_id** (*str*) – Optional, the ID of the GCP project. If set to None or missing, the default project\_id from the GCP connection is used.
- **initial\_split\_keys** (*list*) – (Optional) list of row keys in bytes that will be used to initially split the table into several tablets.
- **column\_families** (*dict*) – (Optional) A map columns to create. The key is the column\_id str and the value is a `google.cloud.bigtable.column_family.GarbageCollectionRule`

```
class airflow.contrib.operators.gcp_bigtable_operator.BigtableTableDeleteOperator (**kwargs)
    Bases: airflow.models.BaseOperator, airflow.contrib.operators.gcp_bigtable_operator.BigtableValidationMixin
```

Deletes the Cloud Bigtable table.

For more details about deleting table have a look at the reference: <https://googleapis.github.io/google-cloud-python/latest/bigtable/table.html#google.cloud.bigtable.table.Table.delete>

**See also:**

For more information on how to use this operator, take a look at the guide: [BigtableTableDeleteOperator](#)

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.
- **table\_id** (*str*) – The ID of the table to be deleted.
- **project\_id** (*str*) – Optional, the ID of the GCP project. If set to None or missing, the default project\_id from the GCP connection is used.

**Parm app\_profile\_id** Application profile.

```
class airflow.contrib.operators.gcp_bigtable_operator.BigtableTableWaitForReplicationSensor
    Bases: airflow.sensors.base\_sensor\_operator.BaseSensorOperator, airflow.contrib.operators.gcp_bigtable_operator.BigtableValidationMixin
```

Sensor that waits for Cloud Bigtable table to be fully replicated to its clusters. No exception will be raised if the instance or the table does not exist.

For more details about cluster states for a table, have a look at the reference: [https://googleapis.github.io/google-cloud-python/latest/bigtable/table.html#google.cloud.bigtable.table.Table.get\\_cluster\\_states](https://googleapis.github.io/google-cloud-python/latest/bigtable/table.html#google.cloud.bigtable.table.Table.get_cluster_states)

**See also:**



For more information on how to use this operator, take a look at the guide: [BigtableTableWaitForReplication-Sensor](#)

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.
- **table\_id** (*str*) – The ID of the table to check replication status.
- **project\_id** (*str*) – Optional, the ID of the GCP project.

#### **poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.operators.cassandra\_to\_gcs.**CassandraToGoogleCloudStorageOperator** (*\*\*kwargs*)  
Bases: [airflow.models.BaseOperator](#)

Copy data from Cassandra to Google cloud storage in JSON format

Note: Arrays of arrays are not supported.

#### **classmethod convert\_map\_type** (*name, value*)

Converts a map to a repeated RECORD that contains two fields: 'key' and 'value', each will be converted to its corresponding data type in BQ.

#### **classmethod convert\_tuple\_type** (*name, value*)

Converts a tuple to RECORD that contains n fields, each will be converted to its corresponding data type in bq and will be named 'field\_<index>', where index is determined by the order of the tuple elements defined in cassandra.

#### **classmethod convert\_user\_type** (*name, value*)

Converts a user type to RECORD that contains n fields, where n is the number of attributes. Each element in the user type class will be converted to its corresponding data type in BQ.

**class** airflow.contrib.operators.gcp\_spanner\_operator.**CloudSpannerInstanceDatabaseDeleteOperator**  
Bases: [airflow.models.BaseOperator](#)

Deletes a Cloud Spanner database.

#### See also:

For more information on how to use this operator, take a look at the guide: [CloudSpannerInstanceDatabaseDeleteOperator](#)

#### Parameters

- **instance\_id** (*str*) – Cloud Spanner instance ID.
- **database\_id** (*str*) – Cloud Spanner database ID.
- **project\_id** (*str*) – Optional, the ID of the project that owns the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_spanner\_operator.**CloudSpannerInstanceDatabaseDeployOperator**  
Bases: [airflow.models.BaseOperator](#)

Creates a new Cloud Spanner database, or if database exists, the operator does nothing.

#### See also:



For more information on how to use this operator, take a look at the guide: [CloudSpannerInstanceDatabaseDeployOperator](#)

#### Parameters

- **instance\_id** (*str*) – The Cloud Spanner instance ID.
- **database\_id** (*str*) – The Cloud Spanner database ID.
- **ddl\_statements** (*list[str]*) – The string list containing DDL for the new database.
- **project\_id** (*str*) – Optional, the ID of the project that owns the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_spanner\_operator.CloudSpannerInstanceDatabaseQueryOperator

Bases: [airflow.models.BaseOperator](#)

Executes an arbitrary DML query (INSERT, UPDATE, DELETE).

#### See also:

For more information on how to use this operator, take a look at the guide: [CloudSpannerInstanceDatabaseQueryOperator](#)

#### Parameters

- **instance\_id** (*str*) – The Cloud Spanner instance ID.
- **database\_id** (*str*) – The Cloud Spanner database ID.
- **query** (*str or list*) – The query or list of queries to be executed. Can be a path to a SQL file.
- **project\_id** (*str*) – Optional, the ID of the project that owns the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_spanner\_operator.CloudSpannerInstanceDatabaseUpdateOperator

Bases: [airflow.models.BaseOperator](#)

Updates a Cloud Spanner database with the specified DDL statement.

#### See also:

For more information on how to use this operator, take a look at the guide: [CloudSpannerInstanceDatabaseUpdateOperator](#)

#### Parameters

- **instance\_id** (*str*) – The Cloud Spanner instance ID.
- **database\_id** (*str*) – The Cloud Spanner database ID.
- **ddl\_statements** (*list[str]*) – The string list containing DDL to apply to the database.
- **project\_id** (*str*) – Optional, the ID of the project that owns the the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.

- **operation\_id**(*str*) – (Optional) Unique per database operation id that can be specified to implement idempotency check.
- **gcp\_conn\_id**(*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_spanner\_operator.**CloudSpannerInstanceDeleteOperator** (\*\*kwargs)  
 Bases: [airflow.models.BaseOperator](#)

Deletes a Cloud Spanner instance. If an instance does not exist, no action is taken and the operator succeeds.

**See also:**

For more information on how to use this operator, take a look at the guide: [CloudSpannerInstanceDeleteOperator](#)

#### Parameters

- **instance\_id**(*str*) – The Cloud Spanner instance ID.
- **project\_id**(*str*) – Optional, the ID of the project that owns the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id**(*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_spanner\_operator.**CloudSpannerInstanceDeployOperator** (\*\*kwargs)  
 Bases: [airflow.models.BaseOperator](#)

Creates a new Cloud Spanner instance, or if an instance with the same instance\_id exists in the specified project, updates the Cloud Spanner instance.

#### Parameters

- **instance\_id**(*str*) – Cloud Spanner instance ID.
- **configuration\_name**(*str*) – The name of the Cloud Spanner instance configuration defining how the instance will be created. Required for instances that do not yet exist.
- **node\_count**(*int*) – (Optional) The number of nodes allocated to the Cloud Spanner instance.
- **display\_name**(*str*) – (Optional) The display name for the Cloud Spanner instance in the GCP Console. (Must be between 4 and 30 characters.) If this value is not set in the constructor, the name is the same as the instance ID.
- **project\_id**(*str*) – Optional, the ID of the project which owns the Cloud Spanner Database. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id**(*str*) – The connection ID used to connect to Google Cloud Platform.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlBaseOperator** (\*\*kwargs)  
 Bases: [airflow.models.BaseOperator](#)

Abstract base operator for Google Cloud SQL operators to inherit from.

#### Parameters

- **instance**(*str*) – Cloud SQL instance ID. This does not include the project ID.
- **project\_id**(*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id**(*str*) – The connection ID used to connect to Google Cloud Platform.

- **api\_version** (*str*) – API version used (e.g. v1beta4).

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceCreateOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Creates a new Cloud SQL instance. If an instance with the same name exists, no action will be taken and the operator will succeed.

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstanceCreateOperator*

#### Parameters

- **body** (*dict*) – Body required by the Cloud SQL insert API, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/insert#request-body>
- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).
- **validate\_body** (*bool*) – True if body should be validated, False otherwise.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceDatabaseCreateOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Creates a new database inside a Cloud SQL instance.

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstanceDatabaseCreateOperator*

#### Parameters

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/databases/insert#request-body>
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).
- **validate\_body** (*bool*) – Whether the body should be validated. Defaults to True.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceDatabaseDeleteOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Deletes a database from a Cloud SQL instance.

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstanceDatabaseDeleteOperator*

#### Parameters

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **database** (*str*) – Name of the database to be deleted in the instance.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceDatabasePatchOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Updates a resource containing information about a database inside a Cloud SQL instance using patch semantics.  
 See: <https://cloud.google.com/sql/docs/mysql/admin-api/how-tos/performance#patch>

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstanceDatabasePatch-Operator*

#### Parameters

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **database** (*str*) – Name of the database to be updated in the instance.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/databases/patch#request-body>
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).
- **validate\_body** (*bool*) – Whether the body should be validated. Defaults to True.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceDeleteOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Deletes a Cloud SQL instance.

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstanceDeleteOperator*

#### Parameters

- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceExportOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Exports data from a Cloud SQL instance to a Cloud Storage bucket as a SQL dump or CSV file.

Note: This operator is idempotent. If executed multiple times with the same export file URI, the export file in GCS will simply be overridden.

#### See also:

For more information on how to use this operator, take a look at the guide: [CloudSqlInstanceImportOperator](#)

#### Parameters

- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/export#request-body>
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).
- **validate\_body** (*bool*) – Whether the body should be validated. Defaults to True.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstanceImportOperator** (\*\*kwargs)  
 Bases: [airflow.contrib.operators.gcp\\_sql\\_operator.CloudSqlBaseOperator](#)

Imports data into a Cloud SQL instance from a SQL dump or CSV file in Cloud Storage.

#### CSV IMPORT:

This operator is NOT idempotent for a CSV import. If the same file is imported multiple times, the imported data will be duplicated in the database. Moreover, if there are any unique constraints the duplicate import may result in an error.

#### SQL IMPORT:

This operator is idempotent for a SQL import if it was also exported by Cloud SQL. The exported SQL contains 'DROP TABLE IF EXISTS' statements for all tables to be imported.

If the import file was generated in a different way, idempotence is not guaranteed. It has to be ensured on the SQL file level.

#### See also:

For more information on how to use this operator, take a look at the guide: [CloudSqlInstanceImportOperator](#)

#### Parameters

- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/export#request-body>
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).
- **validate\_body** (*bool*) – Whether the body should be validated. Defaults to True.

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlInstancePatchOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_sql\_operator.CloudSqlBaseOperator*

Updates settings of a Cloud SQL instance.

Caution: This is a partial update, so only included values for the settings will be updated.

In the request body, supply the relevant portions of an instance resource, according to the rules of patch semantics. <https://cloud.google.com/sql/docs/mysql/admin-api/how-tos/performance#patch>

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlInstancePatchOperator*

#### Parameters

- **body** (*dict*) – Body required by the Cloud SQL patch API, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/patch#request-body>
- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (e.g. v1beta4).

**class** airflow.contrib.operators.gcp\_sql\_operator.**CloudSqlQueryOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Performs DML or DDL query on an existing Cloud Sql instance. It optionally uses cloud-sql-proxy to establish secure connection with the database.

**See also:**

For more information on how to use this operator, take a look at the guide: *CloudSqlQueryOperator*

#### Parameters

- **sql** (*str or list[str]*) – SQL query or list of queries to run (should be DML or DDL query - this operator does not return any data from the database, so it is useless to pass it DQL queries. Note that it is responsibility of the author of the queries to make sure that the queries are idempotent. For example you can use CREATE TABLE IF NOT EXISTS to create a table.
- **parameters** (*mapping or iterable*) – (optional) the parameters to render the SQL query with.
- **autocommit** (*bool*) – if True, each command is automatically committed. (default value: False)
- **gcp\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud Platform for cloud-sql-proxy authentication.
- **gcp\_cloudsql\_conn\_id** (*str*) – The connection ID used to connect to Google Cloud SQL its schema should be gcpcloudsql://. See *CloudSqlDatabaseHook* for details on how to define gcpcloudsql:// connection.

**class** airflow.contrib.operators.dataflow\_operator.**DataFlowJavaOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Start a Java Cloud DataFlow batch job. The parameters of the operation will be passed to the job.

**Example:**

```

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date':
        (2016, 8, 1),
    'email': ['alex@vanboxel.be'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=30),
    'dataflow_default_options': {
        'project': 'my-gcp-project',
        'zone': 'us-central1-f',
        'stagingLocation': 'gs://bucket/tmp/dataflow/staging/',
    }
}

dag = DAG('test-dag', default_args=default_args)

task = DataFlowJavaOperator(
    gcp_conn_id='gcp_default',
    task_id='normalize-cal',
    jar='{{var.value.gcp_dataflow_base}}pipeline-ingress-cal-normalize-1.0.jar',
    options={
        'autoscalingAlgorithm': 'BASIC',
        'maxNumWorkers': '50',
        'start': '{{ds}}',
        'partitionType': 'DAY'
    },
    dag=dag)

```

**See also:**

For more detail on job submission have a look at the reference: <https://cloud.google.com/dataflow/pipelines/specifying-exec-params>

**Parameters**

- **jar** (*str*) – The reference to a self executing DataFlow jar (templated).
- **job\_name** (*str*) – The ‘jobName’ to use when executing the DataFlow job (templated). This ends up being set in the pipeline options, so any entry with key ‘jobName’ in options will be overwritten.
- **dataflow\_default\_options** (*dict*) – Map of default job options.
- **options** (*dict*) – Map of job specific options.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **poll\_sleep** (*int*) – The time in seconds to sleep between polling Google Cloud Platform for the dataflow job status while the job is in the JOB\_STATE\_RUNNING state.
- **job\_class** (*str*) – The name of the dataflow job class to be executed, it is often not the main class configured in the dataflow jar file.

`jar`, `options`, and `job_name` are templated so you can use variables in them.

Note that both `dataflow_default_options` and `options` will be merged to specify pipeline execution parameter, and `dataflow_default_options` is expected to save high-level options, for instances, project and zone information, which apply to all dataflow operators in the DAG.

It's a good practice to define `dataflow_*` parameters in the `default_args` of the dag like the project, zone and staging location.

```
default_args = {
    'dataflow_default_options': {
        'project': 'my-gcp-project',
        'zone': 'europe-west1-d',
        'stagingLocation': 'gs://my-staging-bucket/staging/'
    }
}
```

You need to pass the path to your dataflow as a file reference with the `jar` parameter, the jar needs to be a self executing jar (see documentation here: <https://beam.apache.org/documentation/runners/dataflow/#self-executing-jar>). Use options to pass on options to your job.

```
t1 = DataFlowJavaOperator(
    task_id='datapflow_example',
    jar='{{var.value.gcp_dataflow_base}}pipeline/build/libs/pipeline-example-1.0.
    ↪jar',
    options={
        'autoscalingAlgorithm': 'BASIC',
        'maxNumWorkers': '50',
        'start': '{{ds}}',
        'partitionType': 'DAY',
        'labels': {'foo' : 'bar'}
    },
    gcp_conn_id='gcp-airflow-service-account',
    dag=my-dag)
```

**class** `airflow.contrib.operators.dataflow_operator.DataFlowPythonOperator` (*\*\*kwargs*)  
Bases: `airflow.models.BaseOperator`

Launching Cloud Dataflow jobs written in python. Note that both `dataflow_default_options` and `options` will be merged to specify pipeline execution parameter, and `dataflow_default_options` is expected to save high-level options, for instances, project and zone information, which apply to all dataflow operators in the DAG.

**See also:**

For more detail on job submission have a look at the reference: <https://cloud.google.com/dataflow/pipelines/specifying-exec-params>

### Parameters

- **py\_file** (*str*) – Reference to the python dataflow pipeline file.py, e.g., `/some/local/file/path/to/your/python/pipeline/file`.
- **job\_name** (*str*) – The ‘job\_name’ to use when executing the DataFlow job (templated). This ends up being set in the pipeline options, so any entry with key ‘jobName’ or ‘job\_name’ in options will be overwritten.
- **py\_options** – Additional python options, e.g., `["-m", "-v"]`.
- **dataflow\_default\_options** (*dict*) – Map of default job options.
- **options** (*dict*) – Map of job specific options.



- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **poll\_sleep** (*int*) – The time in seconds to sleep between polling Google Cloud Platform for the dataflow job status while the job is in the JOB\_STATE\_RUNNING state.

**execute** (*context*)

Execute the python dataflow job.

**class** `airflow.contrib.operators.dataproc_operator.DataProcHadoopOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Start a Hadoop Job on a Cloud DataProc cluster.

#### Parameters

- **main\_jar** (*str*) – URI of the job jar provisioned on Cloud Storage. (use this or the `main_class`, not both together).
- **main\_class** (*str*) – Name of the job class. (use this or the `main_jar`, not both together).
- **arguments** (*list*) – Arguments for the job. (templated)
- **archives** (*list*) – List of archived files that will be unpacked in the work directory. Should be stored in Cloud Storage.
- **files** (*list*) – List of files to be copied to the working directory
- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the `task_id` appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes. (templated)
- **cluster\_name** (*str*) – The name of the DataProc cluster. (templated)
- **dataproc\_hadoop\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_hadoop\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

**class** `airflow.contrib.operators.dataproc_operator.DataProcHiveOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Start a Hive query Job on a Cloud DataProc cluster.

#### Parameters

- **query** (*str*) – The query or reference to the query file (q extension).
- **query\_uri** (*str*) – The uri of a hive script on Cloud Storage.
- **variables** (*dict*) – Map of named parameters for the query.
- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the task\_id appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes.
- **cluster\_name** (*str*) – The name of the DataProc cluster.
- **dataproc\_hive\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_hive\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

**class** airflow.contrib.operators.dataproc\_operator.**DataProcPigOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Start a Pig query Job on a Cloud DataProc cluster. The parameters of the operation will be passed to the cluster.

It's a good practice to define dataproc\_\* parameters in the default\_args of the dag like the cluster name and UDFs.

```
default_args = {
    'cluster_name': 'cluster-1',
    'dataproc_pig_jars': [
        'gs://example/udf/jar/datafu/1.2.0/datafu.jar',
        'gs://example/udf/jar/gpig/1.2/gpig.jar'
    ]
}
```

You can pass a pig script as string or file reference. Use variables to pass on variables for the pig script to be resolved on the cluster or use the parameters to be resolved in the script as template parameters.

**Example:**

```
t1 = DataProcPigOperator(
    task_id='dataproc_pig',
    query='a_pig_script.pig',
    variables={'out': 'gs://example/output/{{ds}}'},
    dag=dag)
```

**See also:**

For more detail on about job submission have a look at the reference: <https://cloud.google.com/dataproc/reference/rest/v1/projects.regions.jobs>

**Parameters**

- **query** (*str*) – The query or reference to the query file (pg or pig extension). (templated)
- **query\_uri** (*str*) – The uri of a pig script on Cloud Storage.
- **variables** (*dict*) – Map of named parameters for the query. (templated)
- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the task\_id appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes. (templated)
- **cluster\_name** (*str*) – The name of the DataProc cluster. (templated)
- **dataproc\_pig\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_pig\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

```
class airflow.contrib.operators.dataproc_operator.DataProcPySparkOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Start a PySpark Job on a Cloud DataProc cluster.

**Parameters**

- **main** (*str*) – [Required] The Hadoop Compatible Filesystem (HCFS) URI of the main Python file to use as the driver. Must be a .py file.
- **arguments** (*list*) – Arguments for the job. (templated)
- **archives** (*list*) – List of archived files that will be unpacked in the work directory. Should be stored in Cloud Storage.
- **files** (*list*) – List of files to be copied to the working directory
- **pyfiles** (*list*) – List of Python files to pass to the PySpark framework. Supported file types: .py, .egg, and .zip

- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the task\_id appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes. (templated)
- **cluster\_name** (*str*) – The name of the DataProc cluster.
- **dataproc\_pyspark\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_pyspark\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

**class** airflow.contrib.operators.dataproc\_operator.**DataProcSparkOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Start a Spark Job on a Cloud DataProc cluster.

#### Parameters

- **main\_jar** (*str*) – URI of the job jar provisioned on Cloud Storage. (use this or the main\_class, not both together).
- **main\_class** (*str*) – Name of the job class. (use this or the main\_jar, not both together).
- **arguments** (*list*) – Arguments for the job. (templated)
- **archives** (*list*) – List of archived files that will be unpacked in the work directory. Should be stored in Cloud Storage.
- **files** (*list*) – List of files to be copied to the working directory
- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the task\_id appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes. (templated)
- **cluster\_name** (*str*) – The name of the DataProc cluster. (templated)
- **dataproc\_spark\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_spark\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

**class** `airflow.contrib.operators.dataproc_operator.DataProcSparkSqlOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Start a Spark SQL query Job on a Cloud DataProc cluster.

#### Parameters

- **query** (*str*) – The query or reference to the query file (q extension). (templated)
- **query\_uri** (*str*) – The uri of a spark sql script on Cloud Storage.
- **variables** (*dict*) – Map of named parameters for the query. (templated)
- **job\_name** (*str*) – The job name used in the DataProc cluster. This name by default is the task\_id appended with the execution data, but can be templated. The name will always be appended with a random number to avoid name clashes. (templated)
- **cluster\_name** (*str*) – The name of the DataProc cluster. (templated)
- **dataproc\_spark\_properties** (*dict*) – Map for the Pig properties. Ideal to put in default arguments
- **dataproc\_spark\_jars** (*list*) – URIs to jars provisioned in Cloud Storage (example: for UDFs and libs) and are ideal to put in default arguments.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **region** (*str*) – The specified region where the dataproc cluster is created.
- **job\_error\_states** (*list*) – Job states that should be considered error states. Any states in this list will result in an error being raised and failure of the task. Eg, if the CANCELLED state should also be considered a task failure, pass in ['ERROR', 'CANCELLED']. Possible values are currently only 'ERROR' and 'CANCELLED', but could change in the future. Defaults to ['ERROR'].

**Variables** **dataproc\_job\_id** (*str*) – The actual “jobId” as submitted to the Dataproc API. This is useful for identifying or linking to the job in the Google Cloud Console Dataproc UI, as the actual “jobId” submitted to the Dataproc API is appended with an 8 character random string.

**class** `airflow.contrib.operators.databricks_operator.DatabricksRunNowOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Runs an existing Spark job run to Databricks using the `api/2.0/jobs/run-now` API endpoint.

There are two ways to instantiate this operator.

In the first way, you can take the JSON payload that you typically use to call the `api/2.0/jobs/run-now` endpoint and pass it directly to our `DatabricksRunNowOperator` through the `json` parameter. For example

```

json = {
    "job_id": 42,
    "notebook_params": {
        "dry-run": "true",
        "oldest-time-to-consider": "1457570074236"
    }
}

notebook_run = DatabricksRunNowOperator(task_id='notebook_run', json=json)

```

Another way to accomplish the same thing is to use the named parameters of the `DatabricksRunNowOperator` directly. Note that there is exactly one named parameter for each top level parameter in the `run-now` endpoint. In this method, your code would look like this:

```

job_id=42

notebook_params = {
    "dry-run": "true",
    "oldest-time-to-consider": "1457570074236"
}

python_params = ["douglas adams", "42"]

spark_submit_params = ["--class", "org.apache.spark.examples.SparkPi"]

notebook_run = DatabricksRunNowOperator(
    job_id=job_id,
    notebook_params=notebook_params,
    python_params=python_params,
    spark_submit_params=spark_submit_params
)

```

In the case where both the `json` parameter **AND** the named parameters are provided, they will be merged together. If there are conflicts during the merge, the named parameters will take precedence and override the top level `json` keys.

**Currently the named parameters that `DatabricksRunNowOperator` supports are**

- `job_id`
- `json`
- `notebook_params`
- `python_params`
- `spark_submit_params`

#### Parameters

- **`job_id` (*str*)** – the `job_id` of the existing Databricks job. This field will be templated.

**See also:**

<https://docs.databricks.com/api/latest/jobs.html#run-now>

- **`json` (*dict*)** – A JSON object containing API parameters which will be passed directly to the `api/2.0/jobs/run-now` endpoint. The other named parameters (i.e. `notebook_params`, `spark_submit_params`..) to this operator will be merged with this `json` dictionary if they are provided. If there are conflicts during the merge, the named parameters will take precedence and override the top level `json` keys. (templated)

**See also:**

For more information about templating see *Jinja Templating*. <https://docs.databricks.com/api/latest/jobs.html#run-now>

- **notebook\_params** (*dict*) – A dict from keys to values for jobs with notebook task, e.g. “notebook\_params”: {“name”: “john doe”, “age”: “35”}. The map is passed to the notebook and will be accessible through the `dbutils.widgets.get` function. See *Widgets* for more information. If not specified upon run-now, the triggered run will use the job’s base parameters. `notebook_params` cannot be specified in conjunction with `jar_params`. The json representation of this field (i.e. {“notebook\_params”: {“name”: “john doe”, “age”: “35”}}) cannot exceed 10,000 bytes. This field will be templated.

**See also:**

<https://docs.databricks.com/user-guide/notebooks/widgets.html>

- **python\_params** (*list[str]*) – A list of parameters for jobs with python tasks, e.g. “python\_params”: [“john doe”, “35”]. The parameters will be passed to python file as command line parameters. If specified upon run-now, it would overwrite the parameters specified in job setting. The json representation of this field (i.e. {“python\_params”: [“john doe”, “35”]}) cannot exceed 10,000 bytes. This field will be templated.

**See also:**

<https://docs.databricks.com/api/latest/jobs.html#run-now>

- **spark\_submit\_params** (*list[str]*) – A list of parameters for jobs with spark submit task, e.g. “spark\_submit\_params”: [“-class”, “org.apache.spark.examples.SparkPi”]. The parameters will be passed to spark-submit script as command line parameters. If specified upon run-now, it would overwrite the parameters specified in job setting. The json representation of this field cannot exceed 10,000 bytes. This field will be templated.

**See also:**

<https://docs.databricks.com/api/latest/jobs.html#run-now>

- **timeout\_seconds** (*int32*) – The timeout for this run. By default a value of 0 is used which means to have no timeout. This field will be templated.
- **databricks\_conn\_id** (*str*) – The name of the Airflow connection to use. By default and in the common case this will be `databricks_default`. To use token based authentication, provide the key `token` in the extra field for the connection.
- **polling\_period\_seconds** (*int*) – Controls the rate which we poll for the result of this run. By default the operator will poll every 30 seconds.
- **databricks\_retry\_limit** (*int*) – Amount of times retry if the Databricks backend is unreachable. Its value must be greater than or equal to 1.
- **do\_xcom\_push** (*bool*) – Whether we should push `run_id` and `run_page_url` to xcom.

```
class airflow.contrib.operators.databricks_operator.DatabricksSubmitRunOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Submits a Spark job run to Databricks using the [api/2.0/jobs/runs/submit](https://docs.databricks.com/api/2.0/jobs/runs/submit) API endpoint.

There are two ways to instantiate this operator.

In the first way, you can take the JSON payload that you typically use to call the `api/2.0/jobs/runs/submit` endpoint and pass it directly to our `DatabricksSubmitRunOperator` through the `json` parameter. For example

```

json = {
    'new_cluster': {
        'spark_version': '2.1.0-db3-scala2.11',
        'num_workers': 2
    },
    'notebook_task': {
        'notebook_path': '/Users/airflow@example.com/PrepareData',
    },
}
notebook_run = DatabricksSubmitRunOperator(task_id='notebook_run', json=json)

```

Another way to accomplish the same thing is to use the named parameters of the `DatabricksSubmitRunOperator` directly. Note that there is exactly one named parameter for each top level parameter in the `runs/submit` endpoint. In this method, your code would look like this:

```

new_cluster = {
    'spark_version': '2.1.0-db3-scala2.11',
    'num_workers': 2
}
notebook_task = {
    'notebook_path': '/Users/airflow@example.com/PrepareData',
}
notebook_run = DatabricksSubmitRunOperator(
    task_id='notebook_run',
    new_cluster=new_cluster,
    notebook_task=notebook_task)

```

In the case where both the `json` parameter **AND** the named parameters are provided, they will be merged together. If there are conflicts during the merge, the named parameters will take precedence and override the top level `json` keys.

**Currently the named parameters that `DatabricksSubmitRunOperator` supports are**

- `spark_jar_task`
- `notebook_task`
- `new_cluster`
- `existing_cluster_id`
- `libraries`
- `run_name`
- `timeout_seconds`

### Parameters

- **`json` (*dict*)** – A JSON object containing API parameters which will be passed directly to the `api/2.0/jobs/runs/submit` endpoint. The other named parameters (i.e. `spark_jar_task`, `notebook_task`..) to this operator will be merged with this json dictionary if they are provided. If there are conflicts during the merge, the named parameters will take precedence and override the top level json keys. (templated)

#### See also:

For more information about templating see *Jinja Templating*. <https://docs.databricks.com/api/latest/jobs.html#runs-submit>



- **spark\_jar\_task** (*dict*) – The main class and parameters for the JAR task. Note that the actual JAR is specified in the `libraries`. *EITHER* `spark_jar_task` *OR* `notebook_task` should be specified. This field will be templated.

See also:

<https://docs.databricks.com/api/latest/jobs.html#jobssparkjartask>

- **notebook\_task** (*dict*) – The notebook path and parameters for the notebook task. *EITHER* `spark_jar_task` *OR* `notebook_task` should be specified. This field will be templated.

See also:

<https://docs.databricks.com/api/latest/jobs.html#jobsnotebooktask>

- **new\_cluster** (*dict*) – Specs for a new cluster on which this task will be run. *EITHER* `new_cluster` *OR* `existing_cluster_id` should be specified. This field will be templated.

See also:

<https://docs.databricks.com/api/latest/jobs.html#jobsclusterspecnewcluster>

- **existing\_cluster\_id** (*str*) – ID for existing cluster on which to run this task. *EITHER* `new_cluster` *OR* `existing_cluster_id` should be specified. This field will be templated.
- **libraries** (*list of dicts*) – Libraries which this run will use. This field will be templated.

See also:

<https://docs.databricks.com/api/latest/libraries.html#managedlibrarieslibrary>

- **run\_name** (*str*) – The run name used for this task. By default this will be set to the `airflow.task_id`. This `task_id` is a required parameter of the superclass `BaseOperator`. This field will be templated.
- **timeout\_seconds** (*int*32) – The timeout for this run. By default a value of 0 is used which means to have no timeout. This field will be templated.
- **databricks\_conn\_id** (*str*) – The name of the Airflow connection to use. By default and in the common case this will be `databricks_default`. To use token based authentication, provide the key `token` in the extra field for the connection.
- **polling\_period\_seconds** (*int*) – Controls the rate which we poll for the result of this run. By default the operator will poll every 30 seconds.
- **databricks\_retry\_limit** (*int*) – Amount of times retry if the Databricks backend is unreachable. Its value must be greater than or equal to 1.
- **databricks\_retry\_delay** (*float*) – Number of seconds to wait between retries (it might be a floating point number).
- **do\_xcom\_push** (*bool*) – Whether we should push `run_id` and `run_page_url` to xcom.

```
class airflow.contrib.operators.dataflow_operator.DataflowTemplateOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Start a Templated Cloud DataFlow batch job. The parameters of the operation will be passed to the job.

#### Parameters

- **template** (*str*) – The reference to the DataFlow template.

- **job\_name** – The ‘jobName’ to use when executing the DataFlow template (templated).
- **dataflow\_default\_options** (*dict*) – Map of default job environment options.
- **parameters** (*dict*) – Map of job specific parameters for the template.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **poll\_sleep** (*int*) – The time in seconds to sleep between polling Google Cloud Platform for the dataflow job status while the job is in the JOB\_STATE\_RUNNING state.

It’s a good practice to define `dataflow_*` parameters in the `default_args` of the dag like the project, zone and staging location.

See also:

<https://cloud.google.com/dataflow/docs/reference/rest/v1b3/LaunchTemplateParameters> <https://cloud.google.com/dataflow/docs/reference/rest/v1b3/RuntimeEnvironment>

```
default_args = {
    'dataflow_default_options': {
        'project': 'my-gcp-project',
        'region': 'europe-west1',
        'zone': 'europe-west1-d',
        'tempLocation': 'gs://my-staging-bucket/staging/',
    }
}
```

You need to pass the path to your dataflow template as a file reference with the `template` parameter. Use `parameters` to pass on parameters to your job. Use `environment` to pass on runtime environment variables to your job.

```
t1 = DataflowTemplateOperator(
    task_id='datapflow_example',
    template='{{var.value.gcp_dataflow_base}}',
    parameters={
        'inputFile': "gs://bucket/input/my_input.txt",
        'outputFile': "gs://bucket/output/my_output.txt"
    },
    gcp_conn_id='gcp-airflow-service-account',
    dag=my_dag)
```

`template`, `dataflow_default_options`, `parameters`, and `job_name` are templated so you can use variables in them.

Note that `dataflow_default_options` is expected to save high-level options for project information, which apply to all dataflow operators in the DAG.

See also:

<https://cloud.google.com/dataflow/docs/reference/rest/v1b3/LaunchTemplateParameters>  
<https://cloud.google.com/dataflow/docs/reference/rest/v1b3/RuntimeEnvironment> For more detail on job template execution have a look at the reference: <https://cloud.google.com/dataflow/docs/templates/executing-templates>

**class** `airflow.contrib.operators.dataproc_operator.DataprocClusterCreateOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Create a new cluster on Google Cloud Dataproc. The operator will wait until the creation is successful or an error occurs in the creation process.

The parameters allow to configure the cluster. Please refer to

<https://cloud.google.com/dataproc/docs/reference/rest/v1/projects.regions.clusters>

for a detailed explanation on the different parameters. Most of the configuration parameters detailed in the link are available as a parameter to this operator.

### Parameters

- **cluster\_name** (*str*) – The name of the DataProc cluster to create. (templated)
- **project\_id** (*str*) – The ID of the google cloud project in which to create the cluster. (templated)
- **num\_workers** (*int*) – The # of workers to spin up. If set to zero will spin up cluster in a single node mode
- **storage\_bucket** (*str*) – The storage bucket to use, setting to None lets dataproc generate a custom one for you
- **init\_actions\_uris** (*list[str]*) – List of GCS uri's containing dataproc initialization scripts
- **init\_action\_timeout** (*str*) – Amount of time executable scripts in init\_actions\_uris has to complete
- **metadata** (*dict*) – dict of key-value google compute engine metadata entries to add to all instances
- **image\_version** (*str*) – the version of software inside the Dataproc cluster
- **custom\_image** (*str*) – custom Dataproc image for more info see <https://cloud.google.com/dataproc/docs/guides/dataproc-images>
- **properties** (*dict*) – dict of properties to set on config files (e.g. spark-defaults.conf), see <https://cloud.google.com/dataproc/docs/reference/rest/v1/projects.regions.clusters#SoftwareConfig>
- **master\_machine\_type** (*str*) – Compute engine machine type to use for the master node
- **master\_disk\_type** (*str*) – Type of the boot disk for the master node (default is pd-standard). Valid values: pd-ssd (Persistent Disk Solid State Drive) or pd-standard (Persistent Disk Hard Disk Drive).
- **master\_disk\_size** (*int*) – Disk size for the master node
- **worker\_machine\_type** (*str*) – Compute engine machine type to use for the worker nodes
- **worker\_disk\_type** (*str*) – Type of the boot disk for the worker node (default is pd-standard). Valid values: pd-ssd (Persistent Disk Solid State Drive) or pd-standard (Persistent Disk Hard Disk Drive).
- **worker\_disk\_size** (*int*) – Disk size for the worker nodes
- **num\_preemptible\_workers** (*int*) – The # of preemptible worker nodes to spin up
- **labels** (*dict*) – dict of labels to add to the cluster
- **zone** (*str*) – The zone where the cluster will be located. (templated)

- **network\_uri** (*str*) – The network uri to be used for machine communication, cannot be specified with subnetwork\_uri
- **subnetwork\_uri** (*str*) – The subnetwork uri to be used for machine communication, cannot be specified with network\_uri
- **internal\_ip\_only** (*bool*) – If true, all instances in the cluster will only have internal IP addresses. This can only be enabled for subnetwork enabled networks
- **tags** (*list[str]*) – The GCE tags to add to all instances
- **region** (*str*) – leave as ‘global’, might become relevant in the future. (templated)
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **service\_account** (*str*) – The service account of the dataproc instances.
- **service\_account\_scopes** (*list[str]*) – The URIs of service account scopes to be included.
- **idle\_delete\_ttl** (*int*) – The longest duration that cluster would keep alive while staying idle. Passing this threshold will cause cluster to be auto-deleted. A duration in seconds.
- **auto\_delete\_time** (*datetime.datetime*) – The time when cluster will be auto-deleted.
- **auto\_delete\_ttl** (*int*) – The life duration of cluster, the cluster will be auto-deleted at the end of this duration. A duration in seconds. (If auto\_delete\_time is set this parameter will be ignored)
- **customer\_managed\_key** (*str*) – The customer-managed key used for disk encryption (projects/[PROJECT\_STORING\_KEYS]/locations/[LOCATION]/keyRings/[KEY\_RING\_NAME]/cryptoKeys/[K

**class** airflow.contrib.operators.dataproc\_operator.DataprocClusterDeleteOperator (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Delete a cluster on Google Cloud Dataproc. The operator will wait until the cluster is destroyed.

#### Parameters

- **cluster\_name** (*str*) – The name of the cluster to delete. (templated)
- **project\_id** (*str*) – The ID of the google cloud project in which the cluster runs. (templated)
- **region** (*str*) – leave as ‘global’, might become relevant in the future. (templated)
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**class** airflow.contrib.operators.dataproc\_operator.DataprocClusterScaleOperator (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Scale, up or down, a cluster on Google Cloud Dataproc. The operator will wait until the cluster is re-scaled.

**Example:**

```
t1 = DataprocClusterScaleOperator(
    task_id='dataproc_scale',
    project_id='my-project',
    cluster_name='cluster-1',
    num_workers=10,
    num_preemptible_workers=10,
    graceful_decommission_timeout='1h',
    dag=dag)
```

**See also:**

For more detail on about scaling clusters have a look at the reference: <https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/scaling-clusters>

**Parameters**

- **cluster\_name** (*str*) – The name of the cluster to scale. (templated)
- **project\_id** (*str*) – The ID of the google cloud project in which the cluster runs. (templated)
- **region** (*str*) – The region for the dataproc cluster. (templated)
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **num\_workers** (*int*) – The new number of workers
- **num\_preemptible\_workers** (*int*) – The new number of preemptible workers
- **graceful\_decommission\_timeout** (*str*) – Timeout for graceful YARN decommissioning. Maximum value is 1d
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

```
class airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateBaseOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

```
class airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateInstantiateInlineOperator (**kwargs)
    Bases: airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateBaseOperator
```

Instantiate a WorkflowTemplate Inline on Google Cloud Dataproc. The operator will wait until the WorkflowTemplate is finished executing.

**See also:**

Please refer to: <https://cloud.google.com/dataproc/docs/reference/rest/v1beta2/projects.regions.workflowTemplates/instantiateInline>

**Parameters**

- **template** (*map*) – The template contents. (templated)
- **project\_id** (*str*) – The ID of the google cloud project in which the template runs
- **region** (*str*) – leave as 'global', might become relevant in the future
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

```
class airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateInstantiateOperator
    Bases: airflow.contrib.operators.dataproc_operator.DataprocWorkflowTemplateBaseOperator
```

Instantiate a WorkflowTemplate on Google Cloud Dataproc. The operator will wait until the WorkflowTemplate is finished executing.

**See also:**

Please refer to: <https://cloud.google.com/dataproc/docs/reference/rest/v1beta2/projects.regions.workflowTemplates/instantiate>

#### Parameters

- **template\_id** (*str*) – The id of the template. (templated)
- **project\_id** (*str*) – The ID of the google cloud project in which the template runs
- **region** (*str*) – leave as ‘global’, might become relevant in the future
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

```
class airflow.contrib.operators.datastore_export_operator.DatastoreExportOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Export entities from Google Cloud Datastore to Cloud Storage

#### Parameters

- **bucket** (*str*) – name of the cloud storage bucket to backup data
- **namespace** (*str*) – optional namespace path in the specified Cloud Storage bucket to backup data. If this namespace does not exist in GCS, it will be created.
- **datastore\_conn\_id** (*str*) – the name of the Datastore connection id to use
- **cloud\_storage\_conn\_id** (*str*) – the name of the cloud storage connection id to force-write backup
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **entity\_filter** (*dict*) – description of what data from the project is included in the export, refer to <https://cloud.google.com/datastore/docs/reference/rest/Shared.Types/EntityFilter>
- **labels** (*dict*) – client-assigned labels for cloud storage
- **polling\_interval\_in\_seconds** (*int*) – number of seconds to wait before polling for execution status again
- **overwrite\_existing** (*bool*) – if the storage bucket + namespace is not empty, it will be emptied prior to exports. This enables overwriting existing backups.
- **xcom\_push** (*bool*) – push operation name to xcom for reference

```
class airflow.contrib.operators.datastore_import_operator.DatastoreImportOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Import entities from Cloud Storage to Google Cloud Datastore

#### Parameters

- **bucket** (*str*) – container in Cloud Storage to store data

- **file** (*str*) – path of the backup metadata file in the specified Cloud Storage bucket. It should have the extension `.overall_export_metadata`
- **namespace** (*str*) – optional namespace of the backup metadata file in the specified Cloud Storage bucket.
- **entity\_filter** (*dict*) – description of what data from the project is included in the export, refer to <https://cloud.google.com/datastore/docs/reference/rest/Shared.Types/EntityFilter>
- **labels** (*dict*) – client-assigned labels for cloud storage
- **datastore\_conn\_id** (*str*) – the name of the connection id to use
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **polling\_interval\_in\_seconds** (*int*) – number of seconds to wait before polling for execution status again
- **xcom\_push** (*bool*) – push operation name to xcom for reference

**class** airflow.contrib.operators.discord\_webhook\_operator.**DiscordWebhookOperator** (\*\*kwargs)  
 Bases: *airflow.operators.http\_operator.SimpleHttpOperator*

This operator allows you to post messages to Discord using incoming webhooks. Takes a Discord connection ID with a default relative webhook endpoint. The default endpoint can be overridden using the `webhook_endpoint` parameter (<https://discordapp.com/developers/docs/resources/webhook>).

Each Discord webhook can be pre-configured to use a specific username and `avatar_url`. You can override these defaults in this operator.

#### Parameters

- **http\_conn\_id** (*str*) – Http connection ID with host as `“https://discord.com/api/”` and default webhook endpoint in the extra field in the form of `{“webhook_endpoint”: “webhooks/{webhook.id}/{webhook.token}”}`
- **webhook\_endpoint** (*str*) – Discord webhook endpoint in the form of `“webhooks/{webhook.id}/{webhook.token}”`
- **message** (*str*) – The message you want to send to your Discord channel (max 2000 characters). (templated)
- **username** (*str*) – Override the default username of the webhook. (templated)
- **avatar\_url** (*str*) – Override the default avatar of the webhook
- **tts** (*bool*) – Is a text-to-speech message
- **proxy** (*str*) – Proxy to use to make the Discord webhook call

**execute** (*context*)

Call the `DiscordWebhookHook` to post message

**class** airflow.contrib.operators.druid\_operator.**DruidOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Allows to submit a task directly to druid

#### Parameters

- **json\_index\_file** (*str*) – The filepath to the druid index specification
- **druid\_ingest\_conn\_id** (*str*) – The connection id of the Druid overlord which accepts index jobs

```
class airflow.contrib.operators.ecs_operator.ECSOperator (**kwargs)
```

Bases: `airflow.models.BaseOperator`

Execute a task on AWS EC2 Container Service

#### Parameters

- **task\_definition** (*str*) – the task definition name on EC2 Container Service
- **cluster** (*str*) – the cluster name on EC2 Container Service
- **overrides** (*dict*) – the same parameter that boto3 will receive (templated): [http://boto3.readthedocs.org/en/latest/reference/services/ecs.html#ECS.Client.run\\_task](http://boto3.readthedocs.org/en/latest/reference/services/ecs.html#ECS.Client.run_task)
- **aws\_conn\_id** (*str*) – connection id of AWS credentials / region name. If None, credential boto3 strategy will be used (<http://boto3.readthedocs.io/en/latest/guide/configuration.html>).
- **region\_name** (*str*) – region name to use in AWS Hook. Override the region\_name in connection (if provided)
- **launch\_type** (*str*) – the launch type on which to run your task ('EC2' or 'FARGATE')
- **group** (*str*) – the name of the task group associated with the task
- **placement\_constraints** (*list*) – an array of placement constraint objects to use for the task
- **platform\_version** (*str*) – the platform version on which your task is running
- **network\_configuration** (*dict*) – the network configuration for the task

```
class airflow.contrib.operators.emr_add_steps_operator.EmrAddStepsOperator (**kwargs)
```

Bases: `airflow.models.BaseOperator`

An operator that adds steps to an existing EMR job\_flow.

#### Parameters

- **job\_flow\_id** (*str*) – id of the JobFlow to add steps to. (templated)
- **aws\_conn\_id** (*str*) – aws connection to uses
- **steps** (*list*) – boto3 style steps to be added to the jobflow. (templated)

```
class airflow.contrib.operators.emr_create_job_flow_operator.EmrCreateJobFlowOperator (**kwargs)
```

Bases: `airflow.models.BaseOperator`

Creates an EMR JobFlow, reading the config from the EMR connection. A dictionary of JobFlow overrides can be passed that override the config from the connection.

#### Parameters

- **aws\_conn\_id** (*str*) – aws connection to uses
- **emr\_conn\_id** (*str*) – emr connection to use
- **job\_flow\_overrides** (*dict*) – boto3 style arguments to override emr\_connection extra. (templated)

```
class airflow.contrib.operators.emr_terminate_job_flow_operator.EmrTerminateJobFlowOperator
```

Bases: `airflow.models.BaseOperator`

Operator to terminate EMR JobFlows.

#### Parameters

- **job\_flow\_id** (*str*) – id of the JobFlow to terminate. (templated)



- **aws\_conn\_id** (*str*) – aws connection to uses

**class** airflow.contrib.operators.file\_to\_gcs.**FileToGoogleCloudStorageOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Uploads a file to Google Cloud Storage. Optionally can compress the file for upload.

#### Parameters

- **src** (*str*) – Path to the local file. (templated)
- **dst** (*str*) – Destination path within the specified bucket. (templated)
- **bucket** (*str*) – The bucket to upload to. (templated)
- **google\_cloud\_storage\_conn\_id** (*str*) – The Airflow connection ID to upload with
- **mime\_type** (*str*) – The mime-type string
- **delegate\_to** (*str*) – The account to impersonate, if any
- **gzip** (*bool*) – Allows for file to be compressed and uploaded as gzip

**execute** (*context*)

Uploads the file to Google cloud storage

**class** airflow.contrib.operators.file\_to\_wasb.**FileToWasbOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Uploads a file to Azure Blob Storage.

#### Parameters

- **file\_path** (*str*) – Path to the file to load. (templated)
- **container\_name** (*str*) – Name of the container. (templated)
- **blob\_name** (*str*) – Name of the blob. (templated)
- **wasb\_conn\_id** (*str*) – Reference to the wasb connection.
- **load\_options** (*dict*) – Optional keyword arguments that *WasbHook.load\_file()* takes.

**execute** (*context*)

Upload a file to Azure Blob Storage.

**class** airflow.contrib.operators.gcp\_container\_operator.**GKEClusterCreateOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Create a Google Kubernetes Engine Cluster of specified dimensions The operator will wait until the cluster is created.

The **minimum** required to define a cluster to create is:

**dict** () ::

```
cluster_def = {'name': 'my-cluster-name', 'initial_node_count': 1}
```

or

```
Cluster proto :: from google.cloud.container_v1.types import Cluster
cluster_def = Cluster(name='my-cluster-name', initial_node_count=1)
```

**Operator Creation:**

```
operator = GKEClusterCreateOperator(
    task_id='cluster_create',
    project_id='my-project',
    location='my-location'
    body=cluster_def)
```

**See also:**

For more detail on about creating clusters have a look at the reference: `google.cloud.container_v1.types.Cluster`

**Parameters**

- **project\_id** (*str*) – The Google Developers Console [project ID or project number]
- **location** (*str*) – The name of the Google Compute Engine zone in which the cluster resides.
- **body** (*dict* or *google.cloud.container\_v1.types.Cluster*) – The Cluster definition to create, can be protobuf or python dict, if dict it must match protobuf message Cluster
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **api\_version** (*str*) – The api version to use

**class** `airflow.contrib.operators.gcp_container_operator.GKEClusterDeleteOperator` (\*\*kwargs)  
Bases: `airflow.models.BaseOperator`

Deletes the cluster, including the Kubernetes endpoint and all worker nodes.

To delete a certain cluster, you must specify the `project_id`, the name of the cluster, the `location` that the cluster is in, and the `task_id`.

**Operator Creation:**

```
operator = GKEClusterDeleteOperator(
    task_id='cluster_delete',
    project_id='my-project',
    location='cluster-location'
    name='cluster-name')
```

**See also:**

For more detail about deleting clusters have a look at the reference: [https://google-cloud-python.readthedocs.io/en/latest/container/gapic/v1/api.html#google.cloud.container\\_v1.ClusterManagerClient.delete\\_cluster](https://google-cloud-python.readthedocs.io/en/latest/container/gapic/v1/api.html#google.cloud.container_v1.ClusterManagerClient.delete_cluster)

**Parameters**

- **project\_id** (*str*) – The Google Developers Console [project ID or project number]
- **name** (*str*) – The name of the resource to delete, in this case cluster name
- **location** (*str*) – The name of the Google Compute Engine zone in which the cluster resides.
- **gcp\_conn\_id** (*str*) – The connection ID to use connecting to Google Cloud Platform.
- **api\_version** (*str*) – The api version to use

**class** airflow.contrib.operators.gcp\_container\_operator.**GKEPodOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.kubernetes\_pod\_operator.KubernetesPodOperator*

Executes a task in a Kubernetes pod in the specified Google Kubernetes Engine cluster

This Operator assumes that the system has gcloud installed and either has working default application credentials or has configured a connection id with a service account.

The **minimum** required to define a cluster to create are the variables `task_id`, `project_id`, `location`, `cluster_name`, `name`, `namespace`, and `image`

#### Operator Creation:

```
operator = GKEPodOperator(task_id='pod_op',
                          project_id='my-project',
                          location='us-central1-a',
                          cluster_name='my-cluster-name',
                          name='task-name',
                          namespace='default',
                          image='perl')
```

#### See also:

For more detail about application authentication have a look at the reference: [https://cloud.google.com/docs/authentication/production#providing\\_credentials\\_to\\_your\\_application](https://cloud.google.com/docs/authentication/production#providing_credentials_to_your_application)

#### Parameters

- **project\_id** (*str*) – The Google Developers Console project id
- **location** (*str*) – The name of the Google Kubernetes Engine zone in which the cluster resides, e.g. 'us-central1-a'
- **cluster\_name** (*str*) – The name of the Google Kubernetes Engine cluster the pod should be spawned in
- **gcp\_conn\_id** (*str*) – The google cloud connection id to use. This allows for users to specify a service account.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceBaseOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Abstract base operator for Google Compute Engine operators to inherit from.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceInstanceGroupManagerUpdateTemplateOperator**  
 Bases: *airflow.contrib.operators.gcp\_compute\_operator.GceBaseOperator*

Patches the Instance Group Manager, replacing source template URL with the destination one. API V1 does not have update/patch operations for Instance Group Manager, so you must use beta or newer API version. Beta is the default.

#### See also:

For more information on how to use this operator, take a look at the guide: [GceInstanceGroupManagerUpdateTemplateOperator](#)

#### Parameters

- **resource\_id** (*str*) – Name of the Instance Group Manager
- **zone** (*str*) – Google Cloud Platform zone where the Instance Group Manager exists.

- **source\_template** (*str*) – URL of the template to replace.
- **destination\_template** (*str*) – URL of the target template.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.
- **request\_id** (*str*) – Optional, unique request\_id that you might add to achieve full idempotence (for example when client call times out repeating the request with the same request id will not create a new instance template again). It should be in UUID format as defined in RFC 4122.
- **gcp\_conn\_id** (*str*) – Optional, The connection ID used to connect to Google Cloud Platform. Defaults to 'google\_cloud\_default'.
- **api\_version** (*str*) – Optional, API version used (for example v1 - or beta). Defaults to v1.
- **validate\_body** (*bool*) – Optional, If set to False, body validation is not performed. Defaults to False.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceInstanceStartOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_compute\_operator.GceBaseOperator*

Starts an instance in Google Compute Engine.

**See also:**

For more information on how to use this operator, take a look at the guide: [GceInstanceStartOperator](#)

#### Parameters

- **zone** (*str*) – Google Cloud Platform zone where the instance exists.
- **resource\_id** (*str*) – Name of the Compute Engine instance resource.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – Optional, The connection ID used to connect to Google Cloud Platform. Defaults to 'google\_cloud\_default'.
- **api\_version** (*str*) – Optional, API version used (for example v1 - or beta). Defaults to v1.
- **validate\_body** – Optional, If set to False, body validation is not performed. Defaults to False.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceInstanceStopOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_compute\_operator.GceBaseOperator*

Stops an instance in Google Compute Engine.

**See also:**

For more information on how to use this operator, take a look at the guide: [GceInstanceStopOperator](#)

#### Parameters

- **zone** (*str*) – Google Cloud Platform zone where the instance exists.
- **resource\_id** (*str*) – Name of the Compute Engine instance resource.

- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – Optional, The connection ID used to connect to Google Cloud Platform. Defaults to 'google\_cloud\_default'.
- **api\_version** (*str*) – Optional, API version used (for example v1 - or beta). Defaults to v1.
- **validate\_body** – Optional, If set to False, body validation is not performed. Defaults to False.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceInstanceTemplateCopyOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_compute\_operator.GceBaseOperator*

Copies the instance template, applying specified changes.

**See also:**

For more information on how to use this operator, take a look at the guide: *[GceInstanceTemplateCopyOperator](#)*

#### Parameters

- **resource\_id** (*str*) – Name of the Instance Template
- **body\_patch** (*dict*) – Patch to the body of instanceTemplates object following rfc7386 PATCH semantics. The body\_patch content follows <https://cloud.google.com/compute/docs/reference/rest/v1/instanceTemplates> Name field is required as we need to rename the template, all the other fields are optional. It is important to follow PATCH semantics - arrays are replaced fully, so if you need to update an array you should provide the whole target array as patch element.
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.
- **request\_id** (*str*) – Optional, unique request\_id that you might add to achieve full idempotence (for example when client call times out repeating the request with the same request id will not create a new instance template again). It should be in UUID format as defined in RFC 4122.
- **gcp\_conn\_id** (*str*) – Optional, The connection ID used to connect to Google Cloud Platform. Defaults to 'google\_cloud\_default'.
- **api\_version** (*str*) – Optional, API version used (for example v1 - or beta). Defaults to v1.
- **validate\_body** (*bool*) – Optional, If set to False, body validation is not performed. Defaults to False.

**class** airflow.contrib.operators.gcp\_compute\_operator.**GceSetMachineTypeOperator** (\*\*kwargs)  
 Bases: *airflow.contrib.operators.gcp\_compute\_operator.GceBaseOperator*

Changes the machine type for a stopped instance to the machine type specified in the request.

**See also:**

For more information on how to use this operator, take a look at the guide: *[GceSetMachineTypeOperator](#)*

#### Parameters

- **zone** (*str*) – Google Cloud Platform zone where the instance exists.
- **resource\_id** (*str*) – Name of the Compute Engine instance resource.
- **body** (*dict*) – Body required by the Compute Engine setMachineType API, as described in <https://cloud.google.com/compute/docs/reference/rest/v1/instances/setMachineType#request-body>
- **project\_id** (*str*) – Optional, Google Cloud Platform Project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.
- **gcp\_conn\_id** (*str*) – Optional, The connection ID used to connect to Google Cloud Platform. Defaults to 'google\_cloud\_default'.
- **api\_version** (*str*) – Optional, API version used (for example v1 - or beta). Defaults to v1.
- **validate\_body** (*bool*) – Optional, If set to False, body validation is not performed. Defaults to False.

**class** airflow.contrib.operators.gcp\_function\_operator.**GcfFunctionDeleteOperator** (\*\*kwargs)  
 Bases: [airflow.models.BaseOperator](#)

Deletes the specified function from Google Cloud Functions.

**See also:**

For more information on how to use this operator, take a look at the guide: [GcfFunctionDeleteOperator](#)

#### Parameters

- **name** (*str*) – A fully-qualified function name, matching the pattern: `^projects/[^]+/locations/[^]+/functions/[^]+$`
- **gcp\_conn\_id** (*str*) – The connection ID to use to connect to Google Cloud Platform.
- **api\_version** (*str*) – API version used (for example v1 or v1beta1).

**class** airflow.contrib.operators.gcp\_function\_operator.**GcfFunctionDeployOperator** (\*\*kwargs)  
 Bases: [airflow.models.BaseOperator](#)

Creates a function in Google Cloud Functions. If a function with this name already exists, it will be updated.

**See also:**

For more information on how to use this operator, take a look at the guide: [GcfFunctionDeployOperator](#)

#### Parameters

- **location** (*str*) – Google Cloud Platform region where the function should be created.
- **body** (*dict* or *google.cloud.functions.v1.CloudFunction*) – Body of the Cloud Functions definition. The body must be a Cloud Functions dictionary as described in: <https://cloud.google.com/functions/docs/reference/rest/v1/projects.locations.functions>. Different API versions require different variants of the Cloud Functions dictionary.
- **project\_id** (*str*) – (Optional) Google Cloud Platform project ID where the function should be created.
- **gcp\_conn\_id** (*str*) – (Optional) The connection ID used to connect to Google Cloud Platform - default 'google\_cloud\_default'.

- **api\_version** (*str*) – (Optional) API version used (for example v1 - default - or v1beta1).
- **zip\_path** (*str*) – Path to zip file containing source code of the function. If the path is set, the sourceUploadUrl should not be specified in the body or it should be empty. Then the zip file will be uploaded using the upload URL generated via generateUploadUrl from the Cloud Functions API.
- **validate\_body** (*bool*) – If set to False, body validation is not performed.

**class** airflow.contrib.operators.gcs\_acl\_operator.GoogleCloudStorageBucketCreateAclEntryOperator  
Bases: *airflow.models.BaseOperator*

Creates a new ACL entry on the specified bucket.

#### See also:

For more information on how to use this operator, take a look at the guide: *GoogleCloudStorageBucketCreateAclEntryOperator*

#### Parameters

- **bucket** (*str*) – Name of a bucket.
- **entity** (*str*) – The entity holding the permission, in one of the following forms: user-userId, user-email, group-groupId, group-email, domain-domain, project-team-projectId, allUsers, allAuthenticatedUsers
- **role** (*str*) – The access permission for the entity. Acceptable values are: “OWNER”, “READER”, “WRITER”.
- **user\_project** (*str*) – (Optional) The project to be billed for this request. Required for Requester Pays buckets.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google Cloud Storage.

**class** airflow.contrib.operators.gcs\_operator.GoogleCloudStorageCreateBucketOperator (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Creates a new bucket. Google Cloud Storage uses a flat namespace, so you can’t create a bucket with a name that is already in use.

#### See also:

For more information, see Bucket Naming Guidelines: <https://cloud.google.com/storage/docs/bucketnaming.html#requirements>

#### Parameters

- **bucket\_name** (*str*) – The name of the bucket. (templated)
- **storage\_class** (*str*) – This defines how objects in the bucket are stored and determines the SLA and the cost of storage (templated). Values include
  - MULTI\_REGIONAL
  - REGIONAL
  - STANDARD
  - NEARLINE
  - COLDLINE.

If this value is not specified when the bucket is created, it will default to STANDARD.

- **location** (*str*) – The location of the bucket. (templated) Object data for objects in the bucket resides in physical storage within this region. Defaults to US.

See also:

<https://developers.google.com/storage/docs/bucket-locations>

- **project\_id** (*str*) – The ID of the GCP Project. (templated)
- **labels** (*dict*) – User-provided labels, in key/value pairs.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

:Example:: The following Operator would create a new bucket `test-bucket` with `MULTI_REGIONAL` storage class in EU region

```
CreateBucket = GoogleCloudStorageCreateBucketOperator(
    task_id='CreateNewBucket',
    bucket_name='test-bucket',
    storage_class='MULTI_REGIONAL',
    location='EU',
    labels={'env': 'dev', 'team': 'airflow'},
    google_cloud_storage_conn_id='airflow-service-account'
)
```

**class** airflow.contrib.operators.gcs\_download\_operator.**GoogleCloudStorageDownloadOperator** (*\**  
Bases: *airflow.models.BaseOperator*

Downloads a file from Google Cloud Storage.

#### Parameters

- **bucket** (*str*) – The Google cloud storage bucket where the object is. (templated)
- **object** (*str*) – The name of the object to download in the Google cloud storage bucket. (templated)
- **filename** (*str*) – The file path on the local file system (where the operator is being executed) that the file should be downloaded to. (templated) If no filename passed, the downloaded data will not be stored on the local file system.
- **store\_to\_xcom\_key** (*str*) – If this param is set, the operator will push the contents of the downloaded file to XCom with the key set in this parameter. If not set, the downloaded data will not be pushed to XCom. (templated)
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**class** airflow.contrib.operators.gcs\_list\_operator.**GoogleCloudStorageListOperator** (*\*\*kwargs*)  
Bases: *airflow.models.BaseOperator*

List all objects from the bucket with the give string prefix and delimiter in name.

**This operator returns a python list with the name of objects which can be used by *xcom* in the downstream task.**



### Parameters

- **bucket** (*str*) – The Google cloud storage bucket to find the objects. (templated)
- **prefix** (*str*) – Prefix string which filters objects whose name begin with this prefix. (templated)
- **delimiter** (*str*) – The delimiter by which you want to filter the objects. (templated)  
For e.g to lists the CSV files from in a directory in GCS you would use `delimiter='.csv'`.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**Example:** The following Operator would list all the Avro files from `sales/sales-2017` folder in `data` bucket.

```
GCS_Files = GoogleCloudStorageListOperator(
    task_id='GCS_Files',
    bucket='data',
    prefix='sales/sales-2017/',
    delimiter='.avro',
    google_cloud_storage_conn_id=google_cloud_conn_id
)
```

**class** `airflow.contrib.operators.gcs_acl_operator.GoogleCloudStorageObjectCreateAclEntryOperator`

Bases: `airflow.models.BaseOperator`

Creates a new ACL entry on the specified object.

**See also:**

For more information on how to use this operator, take a look at the guide: [GoogleCloudStorageObjectCreateAclEntryOperator](#)

### Parameters

- **bucket** (*str*) – Name of a bucket.
- **object\_name** (*str*) – Name of the object. For information about how to URL encode object names to be path safe, see: [https://cloud.google.com/storage/docs/json\\_api/#encoding](https://cloud.google.com/storage/docs/json_api/#encoding)
- **entity** (*str*) – The entity holding the permission, in one of the following forms: `user-userId`, `user-email`, `group-groupId`, `group-email`, `domain-domain`, `project-team-projectId`, `allUsers`, `allAuthenticatedUsers`
- **role** (*str*) – The access permission for the entity. Acceptable values are: “OWNER”, “READER”.
- **generation** (*str*) – (Optional) If present, selects a specific revision of this object (as opposed to the latest version, the default).
- **user\_project** (*str*) – (Optional) The project to be billed for this request. Required for Requester Pays buckets.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google Cloud Storage.

```
class airflow.contrib.operators.gcs_to_bq.GoogleCloudStorageToBigQueryOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Loads files from Google cloud storage into BigQuery.

The schema to be used for the BigQuery table may be specified in one of two ways. You may either directly pass the schema fields in, or you may point the operator to a Google cloud storage object name. The object in Google cloud storage must be a JSON file with the schema fields in it.

**See also:**

For more information on how to use this operator, take a look at the guide: [GoogleCloudStorageToBigQuery-Operator](#)

### Parameters

- **bucket** (*str*) – The bucket to load from. (templated)
- **source\_objects** (*list[str]*) – List of Google cloud storage URIs to load from. (templated) If source\_format is 'DATASTORE\_BACKUP', the list must only contain a single URI.
- **destination\_project\_dataset\_table** (*str*) – The dotted (<project>.<dataset>.<table> BigQuery table to load data into. If <project> is not included, project will be the project defined in the connection json. (templated)
- **schema\_fields** (*list*) – If set, the schema field list as defined here: <https://cloud.google.com/bigquery/docs/reference/v2/jobs#configuration.load> Should not be set when source\_format is 'DATASTORE\_BACKUP'.
- **schema\_object** (*str*) – If set, a GCS object path pointing to a .json file that contains the schema for the table. (templated)
- **source\_format** (*str*) – File format to export.
- **compression** (*str*) – [Optional] The compression type of the data source. Possible values include GZIP and NONE. The default value is NONE. This setting is ignored for Google Cloud Bigtable, Google Cloud Datastore backups and Avro formats.
- **create\_disposition** (*str*) – The create disposition if the table doesn't exist.
- **skip\_leading\_rows** (*int*) – Number of rows to skip when loading from a CSV.
- **write\_disposition** (*str*) – The write disposition if the table already exists.
- **field\_delimiter** (*str*) – The delimiter to use when loading from a CSV.
- **max\_bad\_records** (*int*) – The maximum number of bad records that BigQuery can ignore when running the job.
- **quote\_character** (*str*) – The value that is used to quote data sections in a CSV file.
- **ignore\_unknown\_values** (*bool*) – [Optional] Indicates if BigQuery should allow extra values that are not represented in the table schema. If true, the extra values are ignored. If false, records with extra columns are treated as bad records, and if there are too many bad records, an invalid error is returned in the job result.
- **allow\_quoted\_newlines** (*bool*) – Whether to allow quoted newlines (true) or not (false).
- **allow\_jagged\_rows** (*bool*) – Accept rows that are missing trailing optional columns. The missing values are treated as nulls. If false, records with missing trailing columns are treated as bad records, and if there are too many bad records, an invalid error is returned in the job result. Only applicable to CSV, ignored for other formats.

- **max\_id\_key** (*str*) – If set, the name of a column in the BigQuery table that's to be loaded. This will be used to select the MAX value from BigQuery after the load occurs. The results will be returned by the `execute()` command, which in turn gets stored in XCom for future operators to use. This can be helpful with incremental loads—during future executions, you can pick up from the max ID.
- **bigquery\_conn\_id** (*str*) – Reference to a specific BigQuery hook.
- **google\_cloud\_storage\_conn\_id** (*str*) – Reference to a specific Google cloud storage hook.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **schema\_update\_options** (*list*) – Allows the schema of the destination table to be updated as a side effect of the load job.
- **src\_fmt\_configs** (*dict*) – configure optional fields specific to the source format
- **external\_table** (*bool*) – Flag to specify if the destination table should be a BigQuery external table. Default Value is False.
- **time\_partitioning** (*dict*) – configure optional time partitioning fields i.e. partition by field, type and expiration as per API specifications. Note that 'field' is not available in concurrency with `dataset.table$partition`.
- **cluster\_fields** (*list[str]*) – Request that the result of this load be stored sorted by one or more columns. This is only available in conjunction with `time_partitioning`. The order of columns given determines the sort order. Not applicable for external tables.

**class** `airflow.contrib.operators.gcs_to_gcs.GoogleCloudStorageToGoogleCloudStorageOperator` (  
 Bases: `airflow.models.BaseOperator`

Copies objects from a bucket to another, with renaming if requested.

#### Parameters

- **source\_bucket** (*str*) – The source Google cloud storage bucket where the object is. (templated)
- **source\_object** (*str*) – The source name of the object to copy in the Google cloud storage bucket. (templated) You can use only one wildcard for objects (filenames) within your bucket. The wildcard can appear inside the object name or at the end of the object name. Appending a wildcard to the bucket name is unsupported.
- **destination\_bucket** (*str*) – The destination Google cloud storage bucket where the object should be. (templated)
- **destination\_object** (*str*) – The destination name of the object in the destination Google cloud storage bucket. (templated) If a wildcard is supplied in the `source_object` argument, this is the prefix that will be prepended to the final destination objects' paths. Note that the source path's part before the wildcard will be removed; if it needs to be retained it should be appended to `destination_object`. For example, with prefix `foo/*` and `destination_object` `blah/`, the file `foo/baz` will be copied to `blah/baz`; to retain the prefix write the `destination_object` as e.g. `blah/foo`, in which case the copied file will be named `blah/foo/baz`.
- **move\_object** (*bool*) – When `move_object` is True, the object is moved instead of copied to the new location. This is the equivalent of a `mv` command as opposed to a `cp` command.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.

- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **last\_modified\_time** (*datetime.datetime*) – When specified, if the object(s) were modified after last\_modified\_time, they will be copied/moved. If tzinfo has not been set, UTC will be assumed.

### Example

The following Operator would copy a single file named `sales/sales-2017/january.avro` in the data bucket to the file named `copied_sales/2017/january-backup.avro` in the data\_backup bucket

```
copy_single_file = GoogleCloudStorageToGoogleCloudStorageOperator(
    task_id='copy_single_file',
    source_bucket='data',
    source_object='sales/sales-2017/january.avro',
    destination_bucket='data_backup',
    destination_object='copied_sales/2017/january-backup.avro',
    google_cloud_storage_conn_id=google_cloud_conn_id
)
```

The following Operator would copy all the Avro files from `sales/sales-2017` folder (i.e. with names starting with that prefix) in data bucket to the `copied_sales/2017` folder in the data\_backup bucket.

```
copy_files = GoogleCloudStorageToGoogleCloudStorageOperator(
    task_id='copy_files',
    source_bucket='data',
    source_object='sales/sales-2017/*.avro',
    destination_bucket='data_backup',
    destination_object='copied_sales/2017/',
    google_cloud_storage_conn_id=google_cloud_conn_id
)
```

The following Operator would move all the Avro files from `sales/sales-2017` folder (i.e. with names starting with that prefix) in data bucket to the same folder in the data\_backup bucket, deleting the original files in the process.

```
move_files = GoogleCloudStorageToGoogleCloudStorageOperator(
    task_id='move_files',
    source_bucket='data',
    source_object='sales/sales-2017/*.avro',
    destination_bucket='data_backup',
    move_object=True,
    google_cloud_storage_conn_id=google_cloud_conn_id
)
```

**class** `airflow.contrib.operators.gcs_to_gcs_transfer_operator.GoogleCloudStorageToGoogleCloudStorageTransferOperator`  
 Bases: `airflow.models.BaseOperator`

Copies objects from a bucket to another using the GCP Storage Transfer Service.

### Parameters

- **source\_bucket** (*str*) – The source Google cloud storage bucket where the object is. (templated)
- **destination\_bucket** (*str*) – The destination Google cloud storage bucket where the object should be. (templated)
- **project\_id** (*str*) – The ID of the Google Cloud Platform Console project that owns the job

- **gcp\_conn\_id** (*str*) – Optional connection ID to use when connecting to Google Cloud Storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **description** (*str*) – Optional transfer service job description
- **schedule** (*dict*) – Optional transfer service schedule; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/transferJobs>. If not set, run transfer job once as soon as the operator runs
- **object\_conditions** (*dict*) – Optional transfer service object conditions; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/TransferSpec#ObjectConditions>
- **transfer\_options** (*dict*) – Optional transfer service transfer options; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/TransferSpec#TransferOptions>
- **wait** (*bool*) – Wait for transfer to finish; defaults to *True*

**Example:**

```
gcs_to_gcs_transfer_op = GoogleCloudStorageToGoogleCloudStorageTransferOperator(
    task_id='gcs_to_gcs_transfer_example',
    source_bucket='my-source-bucket',
    destination_bucket='my-destination-bucket',
    project_id='my-gcp-project',
    dag=my_dag)
```

**class** airflow.contrib.operators.gcs\_to\_s3.**GoogleCloudStorageToS3Operator** (*\*\*kwargs*)  
 Bases: *airflow.contrib.operators.gcs\_list\_operator.GoogleCloudStorageListOperator*  
 Synchronizes a Google Cloud Storage bucket with an S3 bucket.

**Parameters**

- **bucket** (*str*) – The Google Cloud Storage bucket to find the objects. (templated)
- **prefix** (*str*) – Prefix string which filters objects whose name begin with this prefix. (templated)
- **delimiter** (*str*) – The delimiter by which you want to filter the objects. (templated)  
 For e.g to lists the CSV files from in a directory in GCS you would use `delimiter='csv'`.
- **google\_cloud\_storage\_conn\_id** (*str*) – The connection ID to use when connecting to Google Cloud Storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **dest\_aws\_conn\_id** (*str*) – The destination S3 connection
- **dest\_s3\_key** (*str*) – The base S3 key to be used to store the files. (templated)
- **dest\_verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is `False`), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

```
class airflow.contrib.operators.hipchat_operator.HipChatAPIOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Base HipChat Operator. All derived HipChat operators reference from HipChat's official REST API documentation at <https://www.hipchat.com/docs/apiv2>. Before using any HipChat API operators you need to get an authentication token at <https://www.hipchat.com/docs/apiv2/auth>. In the future additional HipChat operators will be derived from this class as well.

#### Parameters

- **token** (*str*) – HipChat REST API authentication token
- **base\_url** (*str*) – HipChat REST API base url.

#### prepare\_request ()

Used by the execute function. Set the request method, url, and body of HipChat's REST API call. Override in child class. Each HipChatAPI child operator is responsible for having a prepare\_request method call which sets self.method, self.url, and self.body.

```
class airflow.contrib.operators.hipchat_operator.HipChatAPISendRoomNotificationOperator (**kwargs)
    Bases: airflow.contrib.operators.hipchat_operator.HipChatAPIOperator
```

Send notification to a specific HipChat room. More info: [https://www.hipchat.com/docs/apiv2/method/send\\_room\\_notification](https://www.hipchat.com/docs/apiv2/method/send_room_notification)

#### Parameters

- **room\_id** (*str*) – Room in which to send notification on HipChat. (templated)
- **message** (*str*) – The message body. (templated)
- **frm** (*str*) – Label to be shown in addition to sender's name
- **message\_format** (*str*) – How the notification is rendered: html or text
- **color** (*str*) – Background color of the msg: yellow, green, red, purple, gray, or random
- **attach\_to** (*str*) – The message id to attach this notification to
- **notify** (*bool*) – Whether this message should trigger a user notification
- **card** (*dict*) – HipChat-defined card object

#### prepare\_request ()

Used by the execute function. Set the request method, url, and body of HipChat's REST API call. Override in child class. Each HipChatAPI child operator is responsible for having a prepare\_request method call which sets self.method, self.url, and self.body.

```
class airflow.contrib.operators.hive_to_dynamodb.HiveToDynamoDBTransferOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Moves data from Hive to DynamoDB, note that for now the data is loaded into memory before being pushed to DynamoDB, so this operator should be used for smallish amount of data.

#### Parameters

- **sql** (*str*) – SQL query to execute against the hive database. (templated)
- **table\_name** (*str*) – target DynamoDB table
- **table\_keys** (*list*) – partition key and sort key
- **pre\_process** (*function*) – implement pre-processing of source data
- **pre\_process\_args** (*list*) – list of pre\_process function arguments
- **pre\_process\_kwargs** (*dict*) – dict of pre\_process function arguments

- **region\_name** (*str*) – aws region name (example: us-east-1)
- **schema** (*str*) – hive database schema
- **hiveserver2\_conn\_id** (*str*) – source hive connection
- **aws\_conn\_id** (*str*) – aws connection

**class** airflow.contrib.operators.imap\_attachment\_to\_s3\_operator.**ImapAttachmentToS3Operator** (  
Bases: *airflow.models.BaseOperator*

Transfers a mail attachment from a mail server into s3 bucket.

#### Parameters

- **imap\_attachment\_name** (*str*) – The file name of the mail attachment that you want to transfer.
- **s3\_key** (*str*) – The destination file name in the s3 bucket for the attachment.
- **imap\_mail\_folder** (*str*) – The folder on the mail server to look for the attachment.
- **imap\_check\_regex** (*bool*) – If set checks the *imap\_attachment\_name* for a regular expression.
- **s3\_overwrite** (*bool*) – If set overwrites the s3 key if already exists.
- **imap\_conn\_id** (*str*) – The reference to the connection details of the mail server.
- **s3\_conn\_id** (*str*) – The reference to the s3 connection details.

**execute** (*context*)

This function executes the transfer from the email server (via imap) into s3.

**Parameters** **context** (*dict*) – The context while executing.

**class** airflow.contrib.operators.jenkins\_job\_trigger\_operator.**JenkinsJobTriggerOperator** (*\*\*kwargs*)  
Bases: *airflow.models.BaseOperator*

Trigger a Jenkins Job and monitor it's execution. This operator depend on python-jenkins library, version >= 0.4.15 to communicate with jenkins server. You'll also need to configure a Jenkins connection in the connections screen.

#### Parameters

- **jenkins\_connection\_id** (*str*) – The jenkins connection to use for this job
- **job\_name** (*str*) – The name of the job to trigger
- **parameters** (*str*) – The parameters block to provide to jenkins. (templated)
- **sleep\_time** (*int*) – How long will the operator sleep between each status request for the job (min 1, default 10)
- **max\_try\_before\_job\_appears** (*int*) – The maximum number of requests to make while waiting for the job to appears on jenkins server (default 10)

**build\_job** (*jenkins\_server*)

This function makes an API call to Jenkins to trigger a build for 'job\_name' It returned a dict with 2 keys : body and headers. headers contains also a dict-like object which can be queried to get the location to poll in the queue.

**Parameters** **jenkins\_server** – The jenkins server where the job should be triggered

**Returns** Dict containing the response body (key body) and the headers coming along (headers)



**poll\_job\_in\_queue** (*location*, *jenkins\_server*)

This method poll the jenkins queue until the job is executed. When we trigger a job through an API call, the job is first put in the queue without having a build number assigned. Thus we have to wait the job exit the queue to know its build number. To do so, we have to add /api/json (or /api/xml) to the location returned by the build\_job call and poll this file. When a 'executable' block appears in the json, it means the job execution started and the field 'number' then contains the build number.

#### Parameters

- **location** – Location to poll, returned in the header of the build\_job call
- **jenkins\_server** – The jenkins server to poll

**Returns** The build\_number corresponding to the triggered job

**class** airflow.contrib.operators.jira\_operator.**JiraOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

JiraOperator to interact and perform action on Jira issue tracking system. This operator is designed to use Jira Python SDK: <http://jira.readthedocs.io>

#### Parameters

- **jira\_conn\_id** (*str*) – reference to a pre-defined Jira Connection
- **jira\_method** (*str*) – method name from Jira Python SDK to be called
- **jira\_method\_args** (*dict*) – required method parameters for the jira\_method. (templated)
- **result\_processor** (*function*) – function to further process the response from Jira
- **get\_jira\_resource\_method** (*function*) – function or operator to get jira resource on which the provided jira\_method will be executed

**class** airflow.contrib.operators.kubernetes\_pod\_operator.**KubernetesPodOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Execute a task in a Kubernetes Pod

#### Parameters

- **image** (*str*) – Docker image you wish to launch. Defaults to dockerhub.io, but fully qualified URLs will point to custom repositories
- **namespace** (*str*) – the namespace to run within kubernetes
- **cmds** (*list[str]*) – entrypoint of the container. (templated) The docker images's entrypoint is used if this is not provide.
- **arguments** (*list[str]*) – arguments of the entrypoint. (templated) The docker image's CMD is used if this is not provided.
- **image\_pull\_policy** (*str*) – Specify a policy to cache or always pull an image
- **image\_pull\_secrets** (*str*) – Any image pull secrets to be given to the pod. If more than one secret is required, provide a comma separated list: secret\_a,secret\_b
- **volume\_mounts** (*list[airflow.contrib.kubernetes.volume\_mount.VolumeMount]*) – volumeMounts for launched pod
- **volumes** (*list[airflow.contrib.kubernetes.volume.Volume]*) – volumes for launched pod. Includes ConfigMaps and PersistentVolumes
- **labels** (*dict*) – labels to apply to the Pod



- **startup\_timeout\_seconds** (*int*) – timeout in seconds to startup the pod
- **name** (*str*) – name of the task you want to run, will be used to generate a pod id
- **env\_vars** (*dict*) – Environment variables initialized in the container. (templated)
- **secrets** (*list*[*airflow.contrib.kubernetes.secret.Secret*]) – Kubernetes secrets to inject in the container, They can be exposed as environment vars or files in a volume.
- **in\_cluster** (*bool*) – run kubernetes client with in\_cluster configuration
- **cluster\_context** (*str*) – context that points to kubernetes cluster. Ignored when in\_cluster is True. If None, current-context is used.
- **get\_logs** (*bool*) – get the stdout of the container as logs of the tasks
- **affinity** (*dict*) – A dict containing a group of affinity scheduling rules
- **node\_selectors** (*dict*) – A dict containing a group of scheduling rules
- **config\_file** (*str*) – The path to the Kubernetes config file
- **xcom\_push** (*bool*) – If xcom\_push is True, the content of the file /airflow/xcom/return.json in the container will also be pushed to an XCom when the container completes.
- **hostnetwork** (*bool*) – If True enable host networking on the pod
- **tolerations** (*list tolerations*) – A list of kubernetes tolerations

**class** airflow.contrib.operators.mlengine\_operator.**MLEngineBatchPredictionOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Start a Google Cloud ML Engine prediction job.

NOTE: For model origin, users should consider exactly one from the three options below:

1. Populate `uri` field only, which should be a GCS location that points to a tensorflow savedModel directory.
2. Populate `model_name` field only, which refers to an existing model, and the default version of the model will be used.
3. Populate both `model_name` and `version_name` fields, which refers to a specific version of a specific model.

In options 2 and 3, both model and version name should contain the minimal identifier. For instance, call:

```
MLEngineBatchPredictionOperator(
    ...,
    model_name='my_model',
    version_name='my_version',
    ...)
```

if the desired model version is `projects/my_project/models/my_model/versions/my_version`.

See <https://cloud.google.com/ml-engine/reference/rest/v1/projects.jobs> for further documentation on the parameters.

#### Parameters

- **project\_id** (*str*) – The Google Cloud project name where the prediction job is submitted. (templated)

- **job\_id** (*str*) – A unique id for the prediction job on Google Cloud ML Engine. (templated)
- **data\_format** (*str*) – The format of the input data. It will default to 'DATA\_FORMAT\_UNSPECIFIED' if is not provided or is not one of ["TEXT", "TF\_RECORD", "TF\_RECORD\_GZIP"].
- **input\_paths** (*list[str]*) – A list of GCS paths of input data for batch prediction. Accepting wildcard operator \*, but only at the end. (templated)
- **output\_path** (*str*) – The GCS path where the prediction results are written to. (templated)
- **region** (*str*) – The Google Compute Engine region to run the prediction job in. (templated)
- **model\_name** (*str*) – The Google Cloud ML Engine model to use for prediction. If version\_name is not provided, the default version of this model will be used. Should not be None if version\_name is provided. Should be None if uri is provided. (templated)
- **version\_name** (*str*) – The Google Cloud ML Engine model version to use for prediction. Should be None if uri is provided. (templated)
- **uri** (*str*) – The GCS path of the saved model to use for prediction. Should be None if model\_name is provided. It should be a GCS path pointing to a tensorflow SavedModel. (templated)
- **max\_worker\_count** (*int*) – The maximum number of workers to be used for parallel processing. Defaults to 10 if not specified.
- **runtime\_version** (*str*) – The Google Cloud ML Engine runtime version to use for batch prediction.
- **gcp\_conn\_id** (*str*) – The connection ID used for connection to Google Cloud Platform.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**Raises** ValueError: if a unique model/version origin cannot be determined.

```
class airflow.contrib.operators.mlengine_operator.MLEngineModelOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

Operator for managing a Google Cloud ML Engine model.

#### Parameters

- **project\_id** (*str*) – The Google Cloud project name to which MLEngine model belongs. (templated)
- **model** (*dict*) – A dictionary containing the information about the model. If the *operation* is *create*, then the *model* parameter should contain all the information about this model such as *name*.  
If the *operation* is *get*, the *model* parameter should contain the *name* of the model.
- **operation** (*str*) – The operation to perform. Available operations are:
  - *create*: Creates a new model as provided by the *model* parameter.
  - *get*: Gets a particular model where the name is specified in *model*.
- **gcp\_conn\_id** (*str*) – The connection ID to use when fetching connection info.

- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**class** airflow.contrib.operators.mlengine\_operator.**MLEngineTrainingOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Operator for launching a MLEngine training job.

#### Parameters

- **project\_id** (*str*) – The Google Cloud project name within which MLEngine training job should run (templated).
- **job\_id** (*str*) – A unique templated id for the submitted Google MLEngine training job. (templated)
- **package\_uris** (*str*) – A list of package locations for MLEngine training job, which should include the main training program + any additional dependencies. (templated)
- **training\_python\_module** (*str*) – The Python module name to run within MLEngine training job after installing ‘package\_uris’ packages. (templated)
- **training\_args** (*str*) – A list of templated command line arguments to pass to the MLEngine training program. (templated)
- **region** (*str*) – The Google Compute Engine region to run the MLEngine training job in (templated).
- **scale\_tier** (*str*) – Resource tier for MLEngine training job. (templated)
- **master\_type** (*str*) – Cloud ML Engine machine name. Must be set when scale\_tier is CUSTOM. (templated)
- **runtime\_version** (*str*) – The Google Cloud ML runtime version to use for training. (templated)
- **python\_version** (*str*) – The version of Python used in training. (templated)
- **job\_dir** (*str*) – A Google Cloud Storage path in which to store training outputs and other data needed for training. (templated)
- **gcp\_conn\_id** (*str*) – The connection ID to use when fetching connection info.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **mode** (*str*) – Can be one of ‘DRY\_RUN’/‘CLOUD’. In ‘DRY\_RUN’ mode, no real training job will be launched, but the MLEngine training job request will be printed out. In ‘CLOUD’ mode, a real MLEngine training job creation request will be issued.

**class** airflow.contrib.operators.mlengine\_operator.**MLEngineVersionOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

Operator for managing a Google Cloud ML Engine version.

#### Parameters

- **project\_id** (*str*) – The Google Cloud project name to which MLEngine model belongs.
- **model\_name** (*str*) – The name of the Google Cloud ML Engine model that the version belongs to. (templated)

- **version\_name** (*str*) – A name to use for the version being operated upon. If not None and the *version* argument is None or does not have a value for the *name* key, then this will be populated in the payload for the *name* key. (templated)
- **version** (*dict*) – A dictionary containing the information about the version. If the *operation* is *create*, *version* should contain all the information about this version such as name, and deploymentUrl. If the *operation* is *get* or *delete*, the *version* parameter should contain the *name* of the version. If it is None, the only *operation* possible would be *list*. (templated)
- **operation** (*str*) – The operation to perform. Available operations are:
  - *create*: Creates a new version in the model specified by *model\_name*, in which case the *version* parameter should contain all the information to create that version (e.g. *name*, *deploymentUrl*).
  - *get*: Gets full information of a particular version in the model specified by *model\_name*. The name of the version should be specified in the *version* parameter.
  - *list*: Lists all available versions of the model specified by *model\_name*.
  - *delete*: Deletes the version specified in *version* parameter from the model specified by *model\_name*. The name of the version should be specified in the *version* parameter.
- **gcp\_conn\_id** (*str*) – The connection ID to use when fetching connection info.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**class** airflow.contrib.operators.mongo\_to\_s3.**MongoToS3Operator** (\*\*kwargs)

Bases: [airflow.models.BaseOperator](#)

**Mongo -> S3** A more specific baseOperator meant to move data from mongo via pymongo to s3 via boto

**things to note** .execute() is written to depend on .transform() .transform() is meant to be extended by child classes to perform transformations unique to those operators needs

**execute** (*context*)

Executed by task\_instance at runtime

**static transform** (*docs*)

**Processes pyMongo cursor and returns an iterable with each element being** a JSON serializable dictionary

Base transform() assumes no processing is needed ie. docs is a pyMongo cursor of documents and cursor just needs to be passed through

Override this method for custom transformations

**class** airflow.contrib.operators.mysql\_to\_gcs.**MySQLToGoogleCloudStorageOperator** (\*\*kwargs)

Bases: [airflow.models.BaseOperator](#)

Copy data from MySQL to Google cloud storage in JSON format.

#### Parameters

- **sql** (*str*) – The SQL to execute on the MySQL table.
- **bucket** (*str*) – The bucket to upload to.
- **filename** (*str*) – The filename to use as the object name when uploading to Google cloud storage. A {} should be specified in the filename to allow the operator to inject file numbers in cases where the file is split due to size.

- **schema\_filename** (*str*) – If set, the filename to use as the object name when uploading a .json file containing the BigQuery schema fields for the table that was dumped from MySQL.
- **approx\_max\_file\_size\_bytes** (*long*) – This operator supports the ability to split large table dumps into multiple files (see notes in the filename param docs above). Google cloud storage allows for files to be a maximum of 4GB. This param allows developers to specify the file size of the splits.
- **mysql\_conn\_id** (*str*) – Reference to a specific MySQL hook.
- **google\_cloud\_storage\_conn\_id** (*str*) – Reference to a specific Google cloud storage hook.
- **schema** (*str or list*) – The schema to use, if any. Should be a list of dict or a str. Pass a string if using Jinja template, otherwise, pass a list of dict. Examples could be seen: [https://cloud.google.com/bigquery/docs/schemas#specifying\\_a\\_json\\_schema\\_file](https://cloud.google.com/bigquery/docs/schemas#specifying_a_json_schema_file)
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**classmethod type\_map** (*mysql\_type*)

Helper function that maps from MySQL fields to BigQuery fields. Used when a schema\_filename is set.

**class** airflow.contrib.operators.oracle\_to\_azure\_data\_lake\_transfer.**OracleToAzureDataLakeTransfer**

Bases: *airflow.models.BaseOperator*

Moves data from Oracle to Azure Data Lake. The operator runs the query against Oracle and stores the file locally before loading it into Azure Data Lake.

#### Parameters

- **filename** (*str*) – file name to be used by the csv file.
- **azure\_data\_lake\_conn\_id** (*str*) – destination azure data lake connection.
- **azure\_data\_lake\_path** (*str*) – destination path in azure data lake to put the file.
- **oracle\_conn\_id** (*str*) – source Oracle connection.
- **sql** (*str*) – SQL query to execute against the Oracle database. (templated)
- **sql\_params** (*str*) – Parameters to use in sql query. (templated)
- **delimiter** (*str*) – field delimiter in the file.
- **encoding** (*str*) – encoding type for the file.
- **quotechar** (*str*) – Character to use in quoting.
- **quoting** (*str*) – Quoting strategy. See unicodcsv quoting for more information.

**class** airflow.contrib.operators.oracle\_to\_oracle\_transfer.**OracleToOracleTransfer** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Moves data from Oracle to Oracle.

#### Parameters

- **oracle\_destination\_conn\_id** (*str*) – destination Oracle connection.
- **destination\_table** (*str*) – destination table to insert rows.
- **oracle\_source\_conn\_id** (*str*) – source Oracle connection.
- **source\_sql** (*str*) – SQL query to execute against the source Oracle database. (templated)

- **source\_sql\_params** (*dict*) – Parameters to use in sql query. (templated)
- **rows\_chunk** (*int*) – number of rows per chunk to commit.

**class** airflow.contrib.operators.postgres\_to\_gcs\_operator.**PostgresToGoogleCloudStorageOperator**

Bases: *airflow.models.BaseOperator*

Copy data from Postgres to Google Cloud Storage in JSON format.

**classmethod** **convert\_types** (*value*)

Takes a value from Postgres, and converts it to a value that's safe for JSON/Google Cloud Storage/BigQuery. Dates are converted to UTC seconds. Decimals are converted to floats. Times are converted to seconds.

**classmethod** **type\_map** (*postgres\_type*)

Helper function that maps from Postgres fields to BigQuery fields. Used when a schema\_filename is set.

**class** airflow.contrib.operators.pubsub\_operator.**PubSubPublishOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Publish messages to a PubSub topic.

Each Task publishes all provided messages to the same topic in a single GCP project. If the topic does not exist, this task will fail.

```
from base64 import b64encode as b64e

m1 = {'data': b64e('Hello, World!'),
      'attributes': {'type': 'greeting'}}
m2 = {'data': b64e('Knock, knock')}
m3 = {'attributes': {'foo': ''}}

t1 = PubSubPublishOperator(
    project='my-project', topic='my_topic',
    messages=[m1, m2, m3],
    create_topic=True,
    dag=dag)
```

project, topic, and messages are templated so you can use variables in them.

**class** airflow.contrib.operators.pubsub\_operator.**PubSubSubscriptionCreateOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

Create a PubSub subscription.

By default, the subscription will be created in topic\_project. If subscription\_project is specified and the GCP credentials allow, the Subscription can be created in a different project from its topic.

By default, if the subscription already exists, this operator will not cause the DAG to fail. However, the topic must exist in the project.

```
with DAG('successful DAG') as dag:
    (
        dag
        >> PubSubSubscriptionCreateOperator(
            topic_project='my-project', topic='my-topic',
            subscription='my-subscription')
        >> PubSubSubscriptionCreateOperator(
            topic_project='my-project', topic='my-topic',
            subscription='my-subscription')
    )
```

The operator can be configured to fail if the subscription already exists.

```
with DAG('failing DAG') as dag:
    (
        dag
        >> PubSubSubscriptionCreateOperator(
            topic_project='my-project', topic='my-topic',
            subscription='my-subscription')
        >> PubSubSubscriptionCreateOperator(
            topic_project='my-project', topic='my-topic',
            subscription='my-subscription', fail_if_exists=True)
    )
```

Finally, subscription is not required. If not passed, the operator will generate a universally unique identifier for the subscription's name.

```
with DAG('DAG') as dag:
    (
        dag >> PubSubSubscriptionCreateOperator(
            topic_project='my-project', topic='my-topic')
    )
```

topic\_project, topic, subscription, and subscription are templated so you can use variables in them.

**class** airflow.contrib.operators.pubsub\_operator.PubSubSubscriptionDeleteOperator (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Delete a PubSub subscription.

By default, if the subscription does not exist, this operator will not cause the DAG to fail.

```
with DAG('successful DAG') as dag:
    (
        dag
        >> PubSubSubscriptionDeleteOperator(project='my-project',
                                           subscription='non-existing')
    )
```

The operator can be configured to fail if the subscription already exists.

```
with DAG('failing DAG') as dag:
    (
        dag
        >> PubSubSubscriptionDeleteOperator(
            project='my-project', subscription='non-existing',
            fail_if_not_exists=True)
    )
```

project, and subscription are templated so you can use variables in them.

**class** airflow.contrib.operators.pubsub\_operator.PubSubTopicCreateOperator (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Create a PubSub topic.

By default, if the topic already exists, this operator will not cause the DAG to fail.

```
with DAG('successful DAG') as dag:
    (
```

(continues on next page)

(continued from previous page)

```

dag
>> PubSubTopicCreateOperator(project='my-project',
                             topic='my_new_topic')
>> PubSubTopicCreateOperator(project='my-project',
                             topic='my_new_topic')
)

```

The operator can be configured to fail if the topic already exists.

```

with DAG('failing DAG') as dag:
    (
        dag
        >> PubSubTopicCreateOperator(project='my-project',
                                     topic='my_new_topic')
        >> PubSubTopicCreateOperator(project='my-project',
                                     topic='my_new_topic',
                                     fail_if_exists=True)
    )

```

Both project and topic are templated so you can use variables in them.

**class** `airflow.contrib.operators.pubsub_operator.PubSubTopicDeleteOperator` (\*\*kwargs)  
 Bases: `airflow.models.BaseOperator`

Delete a PubSub topic.

By default, if the topic does not exist, this operator will not cause the DAG to fail.

```

with DAG('successful DAG') as dag:
    (
        dag
        >> PubSubTopicDeleteOperator(project='my-project',
                                     topic='non_existing_topic')
    )

```

The operator can be configured to fail if the topic does not exist.

```

with DAG('failing DAG') as dag:
    (
        dag
        >> PubSubTopicCreateOperator(project='my-project',
                                     topic='non_existing_topic',
                                     fail_if_not_exists=True)
    )

```

Both project and topic are templated so you can use variables in them.

**class** `airflow.contrib.operators.qubole_check_operator.QuboleCheckOperator` (\*\*kwargs)  
 Bases: `airflow.operators.check_operator.CheckOperator`, `airflow.contrib.operators.qubole_operator.QuboleOperator`

Performs checks against Qubole Commands. `QuboleCheckOperator` expects a command that will be executed on QDS. By default, each value on first row of the result of this Qubole Command is evaluated using python `bool` casting. If any of the values return `False`, the check is failed and errors out.

Note that Python `bool` casting evals the following as `False`:

- `False`
- `0`



- Empty string (" ")
- Empty list ([])
- Empty dictionary or set ({} )

Given a query like `SELECT COUNT(*) FROM foo`, it will fail only if the count == 0. You can craft much more complex query that could, for instance, check that the table has the same number of rows as the source table upstream, or that the count of today's partition is greater than yesterday's partition, or that a set of metrics are less than 3 standard deviation for the 7 day average.

This operator can be used as a data quality check in your pipeline, and depending on where you put it in your DAG, you have the choice to stop the critical path, preventing from publishing dubious data, or on the side and receive email alerts without stopping the progress of the DAG.

**Parameters** `qubole_conn_id` (*str*) – Connection id which consists of qds auth\_token

**kwargs:**

Arguments specific to Qubole command can be referred from QuboleOperator docs.

**results\_parser\_callable** This is an optional parameter to extend the flexibility of parsing the results of Qubole command to the users. This is a python callable which can hold the logic to parse list of rows returned by Qubole command. By default, only the values on first row are used for performing checks. This callable should return a list of records on which the checks have to be performed.

---

**Note:** All fields in common with template fields of QuboleOperator and CheckOperator are template-supported.

---

**class** `airflow.contrib.operators.qubole_operator.QuboleOperator` (*\*\*kwargs*)  
 Bases: `airflow.models.BaseOperator`

Execute tasks (commands) on QDS (<https://qubole.com>).

**Parameters** `qubole_conn_id` (*str*) – Connection id which consists of qds auth\_token

**kwargs:**

**command\_type** type of command to be executed, e.g. hivecmd, shellcmd, hadoopcmd

**tags** array of tags to be assigned with the command

**cluster\_label** cluster label on which the command will be executed

**name** name to be given to command

**notify** whether to send email on command completion or not (default is False)

**Arguments specific to command types**

**hivecmd:**

**query** inline query statement

**script\_location** s3 location containing query statement

**sample\_size** size of sample in bytes on which to run query

**macros** macro values which were used in query

**sample\_size** size of sample in bytes on which to run query

**hive-version** Specifies the hive version to be used. eg: 0.13,1.2,etc.

**prestocmd:**

**query** inline query statement

**script\_location** s3 location containing query statement

**macros** macro values which were used in query

**hadoopcmd:**

**sub\_command** must be one these ["jar", "s3distcp", "streaming"] followed by 1 or more args

**shellcmd:**

**script** inline command with args

**script\_location** s3 location containing query statement

**files** list of files in s3 bucket as file1,file2 format. These files will be copied into the working directory where the qubole command is being executed.

**archives** list of archives in s3 bucket as archive1,archive2 format. These will be unarchived into the working directory where the qubole command is being executed

**parameters** any extra args which need to be passed to script (only when script\_location is supplied)

**pigcmd:**

**script** inline query statement (latin\_statements)

**script\_location** s3 location containing pig query

**parameters** any extra args which need to be passed to script (only when script\_location is supplied)

**sparkcmd:**

**program** the complete Spark Program in Scala, SQL, Command, R, or Python

**cmdline** spark-submit command line, all required information must be specify in cmdline itself.

**sql** inline sql query

**script\_location** s3 location containing query statement

**language** language of the program, Scala, SQL, Command, R, or Python

**app\_id** ID of an Spark job server app

**arguments** spark-submit command line arguments

**user\_program\_arguments** arguments that the user program takes in

**macros** macro values which were used in query

**note\_id** Id of the Notebook to run

**dbtapquerycmd:**

**db\_tap\_id** data store ID of the target database, in Qubole.

**query** inline query statement

**macros** macro values which were used in query

**dbexportcmd:**

**mode** Can be 1 for Hive export or 2 for HDFS/S3 export

**schema** Db schema name assumed accordingly by database if not specified

**hive\_table** Name of the hive table

**partition\_spec** partition specification for Hive table.

**dbtap\_id** data store ID of the target database, in Qubole.

**db\_table** name of the db table

**db\_update\_mode** allowinsert or updateonly

**db\_update\_keys** columns used to determine the uniqueness of rows

**export\_dir** HDFS/S3 location from which data will be exported.

**fields\_terminated\_by** hex of the char used as column separator in the dataset

**use\_customer\_cluster** To use cluster to run command

**customer\_cluster\_label** the label of the cluster to run the command on

**additional\_options** Additional Sqoop options which are needed enclose options in double or single quotes e.g. '-map-column-hive id=int,data=string'

**dbimportcmd:**

**mode** 1 (simple), 2 (advance)

**hive\_table** Name of the hive table

**schema** Db schema name assumed accordingly by database if not specified

**hive\_serde** Output format of the Hive Table

**dbtap\_id** data store ID of the target database, in Qubole.

**db\_table** name of the db table

**where\_clause** where clause, if any

**parallelism** number of parallel db connections to use for extracting data

**extract\_query** SQL query to extract data from db. \$CONDITIONS must be part of the where clause.

**boundary\_query** Query to be used get range of row IDs to be extracted

**split\_column** Column used as row ID to split data into ranges (mode 2)

**use\_customer\_cluster** To use cluster to run command

**customer\_cluster\_label** the label of the cluster to run the command on

**additional\_options** Additional Sqoop options which are needed enclose options in double or single quotes

**class** `airflow.contrib.operators.qubole_check_operator.QuboleValueCheckOperator` (\*\*kwargs)

Bases: `airflow.operators.check_operator.ValueCheckOperator`, `airflow.contrib.operators.qubole_operator.QuboleOperator`

Performs a simple value check using Qubole command. By default, each value on the first row of this Qubole command is compared with a pre-defined value. The check fails and errors out if the output of the command is not within the permissible limit of expected value.

**Parameters**

- **qubole\_conn\_id** (*str*) – Connection id which consists of qds auth\_token

- **pass\_value** (*str or int or float*) – Expected value of the query results.
- **tolerance** (*int or float*) – Defines the permissible pass\_value range, for example if tolerance is 2, the Qubole command output can be anything between  $-2 \times \text{pass\_value}$  and  $2 \times \text{pass\_value}$ , without the operator erring out.

kwargs:

Arguments specific to Qubole command can be referred from QuboleOperator docs.

**results\_parser\_callable** This is an optional parameter to extend the flexibility of parsing the results of Qubole command to the users. This is a python callable which can hold the logic to parse list of rows returned by Qubole command. By default, only the values on first row are used for performing checks. This callable should return a list of records on which the checks have to be performed.

---

**Note:** All fields in common with template fields of QuboleOperator and ValueCheckOperator are template-supported.

---

**class** airflow.contrib.operators.s3\_copy\_object\_operator.**S3CopyObjectOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Creates a copy of an object that is already stored in S3.

Note: the S3 connection used here needs to have access to both source and destination bucket/key.

#### Parameters

- **source\_bucket\_key** (*str*) – The key of the source object.  
 It can be either full s3:// style url or relative path from root level.  
 When it's specified as a full s3:// url, please omit source\_bucket\_name.
- **dest\_bucket\_key** (*str*) – The key of the object to copy to.  
 The convention to specify *dest\_bucket\_key* is the same as *source\_bucket\_key*.
- **source\_bucket\_name** (*str*) – Name of the S3 bucket where the source object is in.  
 It should be omitted when *source\_bucket\_key* is provided as a full s3:// url.
- **dest\_bucket\_name** (*str*) – Name of the S3 bucket to where the object is copied.  
 It should be omitted when *dest\_bucket\_key* is provided as a full s3:// url.
- **source\_version\_id** (*str*) – Version ID of the source object (OPTIONAL)
- **aws\_conn\_id** (*str*) – Connection id of the S3 connection to use
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified.  
 You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used, but** SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by boto-core.

**class** airflow.contrib.operators.s3\_delete\_objects\_operator.**S3DeleteObjectsOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

To enable users to delete single object or multiple objects from a bucket using a single HTTP request.

Users may specify up to 1000 keys to delete.

#### Parameters

- **bucket** (*str*) – Name of the bucket in which you are going to delete object(s)
- **keys** (*str or list*) – The key(s) to delete from S3 bucket.  
When `keys` is a string, it's supposed to be the key name of the single object to delete.  
When `keys` is a list, it's supposed to be the list of the keys to delete.  
You may specify up to 1000 keys.
- **aws\_conn\_id** (*str*) – Connection id of the S3 connection to use
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified.  
You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used**, but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

```
class airflow.contrib.operators.s3_list_operator.S3ListOperator (**kwargs)
    Bases: airflow.models.BaseOperator
```

List all objects from the bucket with the given string prefix in name.

This operator returns a python list with the name of objects which can be used by `xcom` in the downstream task.

#### Parameters

- **bucket** (*str*) – The S3 bucket where to find the objects. (templated)
- **prefix** (*str*) – Prefix string to filters the objects whose name begin with such prefix. (templated)
- **delimiter** (*str*) – the delimiter marks key hierarchy. (templated)
- **aws\_conn\_id** (*str*) – The connection ID to use when connecting to S3 storage.
- **verify** (*bool or str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is False), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

**Example:** The following operator would list all the files (excluding subfolders) from the S3 `customers/2018/04/` key in the data bucket.

```
s3_file = S3ListOperator(
    task_id='list_3s_files',
    bucket='data',
    prefix='customers/2018/04/',
```

(continues on next page)

(continued from previous page)

```

    delimiter='/',
    aws_conn_id='aws_customers_conn'
)

```

**class** airflow.contrib.operators.s3\_to\_gcs\_operator.S3ToGoogleCloudStorageOperator (\*\*kwargs)

Bases: *airflow.contrib.operators.s3\_list\_operator.S3ListOperator*

Synchronizes an S3 key, possibly a prefix, with a Google Cloud Storage destination path.

#### Parameters

- **bucket** (*str*) – The S3 bucket where to find the objects. (templated)
- **prefix** (*str*) – Prefix string which filters objects whose name begin with such prefix. (templated)
- **delimiter** (*str*) – the delimiter marks key hierarchy. (templated)
- **aws\_conn\_id** (*str*) – The source S3 connection
- **verify** (*bool* or *str*) – Whether or not to verify SSL certificates for S3 connection. By default SSL certificates are verified. You can provide the following values:
  - **False: do not validate SSL certificates. SSL will still be used** (unless `use_ssl` is False), but SSL certificates will not be verified.
  - **path/to/cert/bundle.pem: A filename of the CA cert bundle to uses.** You can specify this argument if you want to use a different CA cert bundle than the one used by boto3.
- **dest\_gcs\_conn\_id** (*str*) – The destination connection ID to use when connecting to Google Cloud Storage.
- **dest\_gcs** (*str*) – The destination Google Cloud Storage bucket and prefix where you want to store the files. (templated)
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **replace** (*bool*) – Whether you want to replace existing destination files or not.

#### Example:

```

s3_to_gcs_op = S3ToGoogleCloudStorageOperator(
    task_id='s3_to_gcs_example',
    bucket='my-s3-bucket',
    prefix='data/customers-201804',
    dest_gcs_conn_id='google_cloud_default',
    dest_gcs='gs://my.gcs.bucket/some/customers/',
    replace=False,
    dag=my_dag)

```

Note that bucket, prefix, delimiter and dest\_gcs are templated, so you can use variables in them if you wish.

**class** airflow.contrib.operators.s3\_to\_gcs\_transfer\_operator.S3ToGoogleCloudStorageTransferOperator

Bases: *airflow.models.BaseOperator*

Synchronizes an S3 bucket with a Google Cloud Storage bucket using the GCP Storage Transfer Service.

#### Parameters

- **s3\_bucket** (*str*) – The S3 bucket where to find the objects. (templated)
- **gcs\_bucket** (*str*) – The destination Google Cloud Storage bucket where you want to store the files. (templated)
- **project\_id** (*str*) – Optional ID of the Google Cloud Platform Console project that owns the job
- **aws\_conn\_id** (*str*) – The source S3 connection
- **gcp\_conn\_id** (*str*) – The destination connection ID to use when connecting to Google Cloud Storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.
- **description** (*str*) – Optional transfer service job description
- **schedule** (*dict*) – Optional transfer service schedule; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/transferJobs>. If not set, run transfer job once as soon as the operator runs
- **object\_conditions** (*dict*) – Optional transfer service object conditions; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/TransferSpec>
- **transfer\_options** (*dict*) – Optional transfer service transfer options; see <https://cloud.google.com/storage-transfer/docs/reference/rest/v1/TransferSpec>
- **wait** (*bool*) – Wait for transfer to finish

**Example:**

```
s3_to_gcs_transfer_op = S3ToGoogleCloudStorageTransferOperator(
    task_id='s3_to_gcs_transfer_example',
    s3_bucket='my-s3-bucket',
    project_id='my-gcp-project',
    gcs_bucket='my-gcs-bucket',
    dag=my_dag)
```

**class** airflow.contrib.operators.s3\_to\_sftp\_operator.**S3ToSFTPOperator** (*\*\*kwargs*)  
 Bases: *airflow.models.BaseOperator*

This operator enables the transferring of files from S3 to a SFTP server.

**Parameters**

- **sftp\_conn\_id** (*string*) – The sftp connection id. The name or identifier for establishing a connection to the SFTP server.
- **sftp\_path** (*string*) – The sftp remote path. This is the specified file path for uploading file to the SFTP server.
- **s3\_conn\_id** (*string*) – The s3 connection id. The name or identifier for establishing a connection to S3
- **s3\_bucket** (*string*) – The targeted s3 bucket. This is the S3 bucket from where the file is downloaded.
- **s3\_key** (*string*) – The targeted s3 key. This is the specified file path for downloading the file from S3.

**static get\_s3\_key** (*s3\_key*)

This parses the correct format for S3 keys regardless of how the S3 url is passed.

**class** airflow.contrib.operators.sftp\_operator.**SFTPOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

SFTPOperator for transferring files from remote host to local or vice a versa. This operator uses ssh\_hook to open sftp transport channel that serve as basis for file transfer.

#### Parameters

- **ssh\_hook** (*airflow.contrib.hooks.ssh\_hook.SSHHook*) – predefined ssh\_hook to use for remote execution. Either *ssh\_hook* or *ssh\_conn\_id* needs to be provided.
- **ssh\_conn\_id** (*str*) – connection id from airflow Connections. *ssh\_conn\_id* will be ignored if *ssh\_hook* is provided.
- **remote\_host** (*str*) – remote host to connect (templated) Nullable. If provided, it will replace the *remote\_host* which was defined in *ssh\_hook* or predefined in the connection of *ssh\_conn\_id*.
- **local\_filepath** (*str*) – local file path to get or put. (templated)
- **remote\_filepath** (*str*) – remote file path to get or put. (templated)
- **operation** (*str*) – specify operation ‘get’ or ‘put’, defaults to put
- **confirm** (*bool*) – specify if the SFTP operation should be confirmed, defaults to True
- **create\_intermediate\_dirs** (*bool*) – create missing intermediate directories when copying from remote to local and vice-versa. Default is False.

Example: The following task would copy `file.txt` to the remote host at `/tmp/tmp1/tmp2/` while creating `tmp`, ‘`tmp1`’ and `tmp2` if they don’t exist. If the parameter is not passed it would error as the directory does not exist.

```
put_file = SFTPOperator(
    task_id="test_sftp",
    ssh_conn_id="ssh_default",
    local_filepath="/tmp/file.txt",
    remote_filepath="/tmp/tmp1/tmp2/file.txt",
    operation="put",
    create_intermediate_dirs=True,
    dag=dag
)
```

**class** airflow.contrib.operators.sftp\_to\_s3\_operator.**SFTPToS3Operator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

This operator enables the transferring of files from a SFTP server to Amazon S3.

#### Parameters

- **sftp\_conn\_id** (*string*) – The sftp connection id. The name or identifier for establishing a connection to the SFTP server.
- **sftp\_path** (*string*) – The sftp remote path. This is the specified file path for downloading the file from the SFTP server.
- **s3\_conn\_id** (*string*) – The s3 connection id. The name or identifier for establishing a connection to S3
- **s3\_bucket** (*string*) – The targeted s3 bucket. This is the S3 bucket to where the file is uploaded.



- **s3\_key** (*string*) – The targeted s3 key. This is the specified path for uploading the file to S3.

**static get\_s3\_key** (*s3\_key*)

This parses the correct format for S3 keys regardless of how the S3 url is passed.

**class** airflow.contrib.operators.ssh\_operator.**SSHOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

SSHOperator to execute commands on given remote host using the ssh\_hook.

#### Parameters

- **ssh\_hook** (*airflow.contrib.hooks.ssh\_hook.SSHHook*) – predefined ssh\_hook to use for remote execution. Either *ssh\_hook* or *ssh\_conn\_id* needs to be provided.
- **ssh\_conn\_id** (*str*) – connection id from airflow Connections. *ssh\_conn\_id* will be ignored if *ssh\_hook* is provided.
- **remote\_host** (*str*) – remote host to connect (templated) Nullable. If provided, it will replace the *remote\_host* which was defined in *ssh\_hook* or predefined in the connection of *ssh\_conn\_id*.
- **command** (*str*) – command to execute on remote host. (templated)
- **timeout** (*int*) – timeout (in seconds) for executing the command.
- **do\_xcom\_push** (*bool*) – return the stdout which also get set in xcom by airflow platform

**class** airflow.contrib.operators.sagemaker\_base\_operator.**SageMakerBaseOperator** (*\*\*kwargs*)

Bases: *airflow.models.BaseOperator*

This is the base operator for all SageMaker operators.

#### Parameters

- **config** (*dict*) – The configuration necessary to start a training job (templated)
- **aws\_conn\_id** (*str*) – The AWS connection ID to use.

**class** airflow.contrib.operators.sagemaker\_endpoint\_config\_operator.**SageMakerEndpointConfigOperator** (*\*\*kwargs*)

Bases: *airflow.contrib.operators.sagemaker\_base\_operator.SageMakerBaseOperator*

Create a SageMaker endpoint config.

This operator returns The ARN of the endpoint config created in Amazon SageMaker

#### Parameters

- **config** (*dict*) – The configuration necessary to create an endpoint config.  
For details of the configuration parameter see *SageMaker.Client.create\_endpoint\_config()*
- **aws\_conn\_id** (*str*) – The AWS connection ID to use.

**class** airflow.contrib.operators.sagemaker\_endpoint\_operator.**SageMakerEndpointOperator** (*\*\*kwargs*)

Bases: *airflow.contrib.operators.sagemaker\_base\_operator.SageMakerBaseOperator*

Create a SageMaker endpoint.

This operator returns The ARN of the endpoint created in Amazon SageMaker

#### Parameters

- **config** (*dict*) – The configuration necessary to create an endpoint.

If you need to create a SageMaker endpoint based on an existed SageMaker model and an existed SageMaker endpoint config:

```
config = endpoint_configuration;
```

If you need to create all of SageMaker model, SageMaker endpoint-config and SageMaker endpoint:

```
config = {
    'Model': model_configuration,
    'EndpointConfig': endpoint_config_configuration,
    'Endpoint': endpoint_configuration
}
```

For details of the configuration parameter of `model_configuration` see `SageMaker.Client.create_model()`

For details of the configuration parameter of `endpoint_config_configuration` see `SageMaker.Client.create_endpoint_config()`

For details of the configuration parameter of `endpoint_configuration` see `SageMaker.Client.create_endpoint()`

- **aws\_conn\_id** (*str*) – The AWS connection ID to use.
- **wait\_for\_completion** (*bool*) – Whether the operator should wait until the endpoint creation finishes.
- **check\_interval** (*int*) – If wait is set to True, this is the time interval, in seconds, that this operation waits before polling the status of the endpoint creation.
- **max\_ingestion\_time** (*int*) – If wait is set to True, this operation fails if the endpoint creation doesn't finish within `max_ingestion_time` seconds. If you set this parameter to None it never times out.
- **operation** (*str*) – Whether to create an endpoint or update an endpoint. Must be either 'create' or 'update'.

```
class airflow.contrib.operators.sagemaker_model_operator.SageMakerModelOperator (**kwargs)
    Bases: airflow.contrib.operators.sagemaker_base_operator.SageMakerBaseOperator
```

Create a SageMaker model.

This operator returns The ARN of the model created in Amazon SageMaker

#### Parameters

- **config** (*dict*) – The configuration necessary to create a model.

For details of the configuration parameter see `SageMaker.Client.create_model()`

- **aws\_conn\_id** (*str*) – The AWS connection ID to use.

```
class airflow.contrib.operators.sagemaker_training_operator.SageMakerTrainingOperator (**kwargs)
    Bases: airflow.contrib.operators.sagemaker_base_operator.SageMakerBaseOperator
```

Initiate a SageMaker training job.

This operator returns The ARN of the training job created in Amazon SageMaker.

### Parameters

- **config** (*dict*) – The configuration necessary to start a training job (templated).  
For details of the configuration parameter see `SageMaker.Client.create_training_job()`
- **aws\_conn\_id** (*str*) – The AWS connection ID to use.
- **wait\_for\_completion** (*bool*) – If wait is set to True, the time interval, in seconds, that the operation waits to check the status of the training job.
- **print\_log** (*bool*) – if the operator should print the cloudwatch log during training
- **check\_interval** (*int*) – if wait is set to be true, this is the time interval in seconds which the operator will check the status of the training job
- **max\_ingestion\_time** (*int*) – If wait is set to True, the operation fails if the training job doesn't finish within max\_ingestion\_time seconds. If you set this parameter to None, the operation does not timeout.

**class** airflow.contrib.operators.sagemaker\_transform\_operator.**SageMakerTransformOperator** (*\*\*kwargs*)  
 Bases: `airflow.contrib.operators.sagemaker_base_operator.SageMakerBaseOperator`

Initiate a SageMaker transform job.

This operator returns The ARN of the model created in Amazon SageMaker.

### Parameters

- **config** (*dict*) – The configuration necessary to start a transform job (templated).  
If you need to create a SageMaker transform job based on an existed SageMaker model:

```
config = transform_config
```

If you need to create both SageMaker model and SageMaker Transform job:

```
config = {
    'Model': model_config,
    'Transform': transform_config
}
```

For details of the configuration parameter of transform\_config see `SageMaker.Client.create_transform_job()`

For details of the configuration parameter of model\_config, See: `SageMaker.Client.create_model()`

- **aws\_conn\_id** (*string*) – The AWS connection ID to use.
- **wait\_for\_completion** (*bool*) – Set to True to wait until the transform job finishes.
- **check\_interval** (*int*) – If wait is set to True, the time interval, in seconds, that this operation waits to check the status of the transform job.
- **max\_ingestion\_time** (*int*) – If wait is set to True, the operation fails if the transform job doesn't finish within max\_ingestion\_time seconds. If you set this parameter to None, the operation does not timeout.

**class** airflow.contrib.operators.sagemaker\_tuning\_operator.**SageMakerTuningOperator** (*\*\*kwargs*)  
 Bases: `airflow.contrib.operators.sagemaker_base_operator.SageMakerBaseOperator`

Initiate a SageMaker hyperparameter tuning job.

This operator returns The ARN of the tuning job created in Amazon SageMaker.

#### Parameters

- **config** (*dict*) – The configuration necessary to start a tuning job (templated).  
For details of the configuration parameter see `SageMaker.Client.create_hyper_parameter_tuning_job()`
- **aws\_conn\_id** (*str*) – The AWS connection ID to use.
- **wait\_for\_completion** (*bool*) – Set to True to wait until the tuning job finishes.
- **check\_interval** (*int*) – If wait is set to True, the time interval, in seconds, that this operation waits to check the status of the tuning job.
- **max\_ingestion\_time** (*int*) – If wait is set to True, the operation fails if the tuning job doesn't finish within max\_ingestion\_time seconds. If you set this parameter to None, the operation does not timeout.

```
class airflow.contrib.operators.segment_track_event_operator.SegmentTrackEventOperator (**kwargs)
Bases: airflow.models.BaseOperator
```

Send Track Event to Segment for a specified user\_id and event

#### Parameters

- **user\_id** (*str*) – The ID for this user in your database. (templated)
- **event** (*str*) – The name of the event you're tracking. (templated)
- **properties** (*dict*) – A dictionary of properties for the event. (templated)
- **segment\_conn\_id** (*str*) – The connection ID to use when connecting to Segment.
- **segment\_debug\_mode** (*bool*) – Determines whether Segment should run in debug mode. Defaults to False

```
class airflow.contrib.operators.slack_webhook_operator.SlackWebhookOperator (**kwargs)
Bases: airflow.operators.http_operator.SimpleHttpOperator
```

This operator allows you to post messages to Slack using incoming webhooks. Takes both Slack webhook token directly and connection that has Slack webhook token. If both supplied, Slack webhook token will be used.

Each Slack webhook token can be pre-configured to use a specific channel, username and icon. You can override these defaults in this hook.

#### Parameters

- **http\_conn\_id** (*str*) – connection that has Slack webhook token in the extra field
- **webhook\_token** (*str*) – Slack webhook token
- **message** (*str*) – The message you want to send on Slack
- **attachments** (*list*) – The attachments to send on Slack. Should be a list of dictionaries representing Slack attachments.
- **channel** (*str*) – The channel the message should be posted to
- **username** (*str*) – The username to post to slack with
- **icon\_emoji** (*str*) – The emoji to use as icon for the user posting to Slack
- **link\_names** (*bool*) – Whether or not to find and link channel and usernames in your message

- **proxy** (*str*) – Proxy to use to make the Slack webhook call

**execute** (*context*)

Call the SlackWebhookHook to post the provided Slack message

**class** airflow.contrib.operators.snowflake\_operator.**SnowflakeOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Executes sql code in a Snowflake database

#### Parameters

- **snowflake\_conn\_id** (*str*) – reference to specific snowflake connection id
- **sql** (*Can receive a str representing a sql statement, a list of str (sql statements), or reference to a template file. Template reference are recognized by str ending in '.sql'*) – the sql code to be executed. (templated)
- **warehouse** (*str*) – name of warehouse which overwrite defined one in connection
- **database** (*str*) – name of database which overwrite defined one in connection

**class** airflow.contrib.operators.sns\_publish\_operator.**SnsPublishOperator** (\*\*kwargs)  
Bases: *airflow.models.BaseOperator*

Publish a message to Amazon SNS.

#### Parameters

- **aws\_conn\_id** (*str*) – aws connection to use
- **target\_arn** (*str*) – either a TopicArn or an EndpointArn
- **message** (*str*) – the default message you want to send (templated)

**class** airflow.contrib.operators.spark\_jdbc\_operator.**SparkJDBCOperator** (\*\*kwargs)  
Bases: *airflow.contrib.operators.spark\_submit\_operator.SparkSubmitOperator*

This operator extends the SparkSubmitOperator specifically for performing data transfers to/from JDBC-based databases with Apache Spark. As with the SparkSubmitOperator, it assumes that the “spark-submit” binary is available on the PATH.

#### Parameters

- **spark\_app\_name** (*str*) – Name of the job (default airflow-spark-jdbc)
- **spark\_conn\_id** (*str*) – Connection id as configured in Airflow administration
- **spark\_conf** (*dict*) – Any additional Spark configuration properties
- **spark\_py\_files** (*str*) – Additional python files used (.zip, .egg, or .py)
- **spark\_files** (*str*) – Additional files to upload to the container running the job
- **spark\_jars** (*str*) – Additional jars to upload and add to the driver and executor class-path
- **num\_executors** (*int*) – number of executor to run. This should be set so as to manage the number of connections made with the JDBC database
- **executor\_cores** (*int*) – Number of cores per executor
- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G)
- **driver\_memory** (*str*) – Memory allocated to the driver (e.g. 1000M, 2G)
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-submit for debugging

- **keytab** (*str*) – Full path to the file that contains the keytab
- **principal** (*str*) – The name of the kerberos principal used for keytab
- **cmd\_type** (*str*) – Which way the data should flow. 2 possible values: `spark_to_jdbc`: data written by spark from metastore to jdbc `jdbc_to_spark`: data written by spark from jdbc to metastore
- **jdbc\_table** (*str*) – The name of the JDBC table
- **jdbc\_conn\_id** (*str*) – Connection id used for connection to JDBC database
- **jdbc\_driver** (*str*) – Name of the JDBC driver to use for the JDBC connection. This driver (usually a jar) should be passed in the ‘jars’ parameter
- **metastore\_table** (*str*) – The name of the metastore table,
- **jdbc\_truncate** (*bool*) – (`spark_to_jdbc` only) Whether or not Spark should truncate or drop and recreate the JDBC table. This only takes effect if ‘save\_mode’ is set to Overwrite. Also, if the schema is different, Spark cannot truncate, and will drop and recreate
- **save\_mode** (*str*) – The Spark save-mode to use (e.g. overwrite, append, etc.)
- **save\_format** (*str*) – (`jdbc_to_spark`-only) The Spark save-format to use (e.g. parquet)
- **batch\_size** (*int*) – (`spark_to_jdbc` only) The size of the batch to insert per round trip to the JDBC database. Defaults to 1000
- **fetch\_size** (*int*) – (`jdbc_to_spark` only) The size of the batch to fetch per round trip from the JDBC database. Default depends on the JDBC driver
- **num\_partitions** (*int*) – The maximum number of partitions that can be used by Spark simultaneously, both for `spark_to_jdbc` and `jdbc_to_spark` operations. This will also cap the number of JDBC connections that can be opened
- **partition\_column** (*str*) – (`jdbc_to_spark`-only) A numeric column to be used to partition the metastore table by. If specified, you must also specify: `num_partitions`, `lower_bound`, `upper_bound`
- **lower\_bound** (*int*) – (`jdbc_to_spark`-only) Lower bound of the range of the numeric partition column to fetch. If specified, you must also specify: `num_partitions`, `partition_column`, `upper_bound`
- **upper\_bound** (*int*) – (`jdbc_to_spark`-only) Upper bound of the range of the numeric partition column to fetch. If specified, you must also specify: `num_partitions`, `partition_column`, `lower_bound`
- **create\_table\_column\_types** – (`spark_to_jdbc`-only) The database column data types to use instead of the defaults, when creating the table. Data type information should be specified in the same format as CREATE TABLE columns syntax (e.g: “name CHAR(64), comments VARCHAR(1024)”). The specified types should be valid spark sql data types.

**execute** (*context*)

Call the SparkSubmitHook to run the provided spark job

**class** `airflow.contrib.operators.spark_sql_operator.SparkSqlOperator` (\*\*kwargs)  
Bases: `airflow.models.BaseOperator`

Execute Spark SQL query

#### Parameters

- **sql** (*str*) – The SQL query to execute. (templated)
- **conf** (*str* (*format*: *PROP=VALUE*)) – arbitrary Spark configuration property

- **conn\_id** (*str*) – connection\_id string
- **total\_executor\_cores** (*int*) – (Standalone & Mesos only) Total cores for all executors (Default: all the available cores on the worker)
- **executor\_cores** (*int*) – (Standalone & YARN only) Number of cores per executor (Default: 2)
- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G) (Default: 1G)
- **keytab** (*str*) – Full path to the file that contains the keytab
- **master** (*str*) – spark://host:port, mesos://host:port, yarn, or local
- **name** (*str*) – Name of the job
- **num\_executors** (*int*) – Number of executors to launch
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-sql
- **yarn\_queue** (*str*) – The YARN queue to submit to (Default: “default”)

**execute** (*context*)

Call the SparkSqlHook to run the provided sql query

**class** airflow.contrib.operators.spark\_submit\_operator.**SparkSubmitOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

This hook is a wrapper around the spark-submit binary to kick off a spark-submit job. It requires that the “spark-submit” binary is in the PATH or the spark-home is set in the extra on the connection.

#### Parameters

- **application** (*str*) – The application that submitted as a job, either jar or py file. (templated)
- **conf** (*dict*) – Arbitrary Spark configuration properties (templated)
- **conn\_id** (*str*) – The connection id as configured in Airflow administration. When an invalid connection\_id is supplied, it will default to yarn.
- **files** (*str*) – Upload additional files to the executor running the job, separated by a comma. Files will be placed in the working directory of each executor. For example, serialized objects. (templated)
- **py\_files** (*str*) – Additional python files used by the job, can be .zip, .egg or .py. (templated)
- **jars** (*str*) – Submit additional jars to upload and place them in executor classpath. (templated)
- **driver\_classpath** (*str*) – Additional, driver-specific, classpath settings. (templated)
- **java\_class** (*str*) – the main class of the Java application
- **packages** (*str*) – Comma-separated list of maven coordinates of jars to include on the driver and executor classpaths. (templated)
- **exclude\_packages** (*str*) – Comma-separated list of maven coordinates of jars to exclude while resolving the dependencies provided in ‘packages’ (templated)
- **repositories** (*str*) – Comma-separated list of additional remote repositories to search for the maven coordinates given with ‘packages’
- **total\_executor\_cores** (*int*) – (Standalone & Mesos only) Total cores for all executors (Default: all the available cores on the worker)



- **executor\_cores** (*int*) – (Standalone & YARN only) Number of cores per executor (Default: 2)
- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G) (Default: 1G)
- **driver\_memory** (*str*) – Memory allocated to the driver (e.g. 1000M, 2G) (Default: 1G)
- **keytab** (*str*) – Full path to the file that contains the keytab (templated)
- **principal** (*str*) – The name of the kerberos principal used for keytab (templated)
- **name** (*str*) – Name of the job (default airflow-spark). (templated)
- **num\_executors** (*int*) – Number of executors to launch
- **application\_args** (*list*) – Arguments for the application being submitted (templated)
- **env\_vars** (*dict*) – Environment variables for spark-submit. It supports yarn and k8s mode too. (templated)
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-submit process for debugging
- **spark\_binary** (*string*) – The command to use for spark submit. Some distros may use spark2-submit.

**execute** (*context*)

Call the SparkSubmitHook to run the provided spark job

**class** airflow.contrib.operators.sqoop\_operator.**SqoopOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Execute a Sqoop job. Documentation for Apache Sqoop can be found here: <https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html>

**execute** (*context*)

Execute sqoop job

**class** airflow.contrib.operators.vertica\_operator.**VerticaOperator** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Executes sql code in a specific Vertica database

#### Parameters

- **vertica\_conn\_id** (*str*) – reference to a specific Vertica database
- **sql** (Can receive a *str* representing a sql statement, a *list* of *str* (sql statements), or reference to a *template file*. Template reference are recognized by *str* ending in *'.sql'*) – the sql code to be executed. (templated)

**class** airflow.contrib.operators.vertica\_to\_hive.**VerticaToHiveTransfer** (\*\*kwargs)

Bases: *airflow.models.BaseOperator*

Moves data from Vertica to Hive. The operator runs your query against Vertica, stores the file locally before loading it into a Hive table. If the create or recreate arguments are set to True, a CREATE TABLE and DROP TABLE statements are generated. Hive data types are inferred from the cursor's metadata. Note that the table generated in Hive uses STORED AS textfile which isn't the most efficient serialization format. If a large amount of data is loaded and/or if the table gets queried considerably, you may want to use this operator only to stage the data into a temporary table before loading it into its final destination using a HiveOperator.

#### Parameters

- **sql** (*str*) – SQL query to execute against the Vertica database. (templated)



- **hive\_table** (*str*) – target Hive table, use dot notation to target a specific database. (templated)
- **create** (*bool*) – whether to create the table if it doesn't exist
- **recreate** (*bool*) – whether to drop and recreate the table at every execution
- **partition** (*dict*) – target partition as a dict of partition columns and values. (templated)
- **delimiter** (*str*) – field delimiter in the file
- **vertica\_conn\_id** (*str*) – source Vertica connection
- **hive\_conn\_id** (*str*) – destination hive connection

**class** airflow.contrib.operators.vertica\_to\_mysql.**VerticaToMySQLTransfer** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Moves data from Vertica to MySQL.

#### Parameters

- **sql** (*str*) – SQL query to execute against the Vertica database. (templated)
- **vertica\_conn\_id** (*str*) – source Vertica connection
- **mysql\_table** (*str*) – target MySQL table, use dot notation to target a specific database. (templated)
- **mysql\_conn\_id** (*str*) – source mysql connection
- **mysql\_preoperator** (*str*) – sql statement to run against MySQL prior to import, typically use to truncate or delete in place of the data coming in, allowing the task to be idempotent (running the task twice won't double load data). (templated)
- **mysql\_postoperator** (*str*) – sql statement to run against MySQL after the import, typically used to move data from staging to production and issue cleanup commands. (templated)
- **bulk\_load** (*bool*) – flag to use bulk\_load option. This loads MySQL directly from a tab-delimited text file using the LOAD DATA LOCAL INFILE command. This option requires an extra connection parameter for the destination MySQL connection: {'local\_infile': true}.

**class** airflow.contrib.operators.wasb\_delete\_blob\_operator.**WasbDeleteBlobOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

Deletes blob(s) on Azure Blob Storage.

#### Parameters

- **container\_name** (*str*) – Name of the container. (templated)
- **blob\_name** (*str*) – Name of the blob. (templated)
- **wasb\_conn\_id** (*str*) – Reference to the wasb connection.
- **check\_options** – Optional keyword arguments that *WasbHook.check\_for\_blob()* takes.
- **is\_prefix** (*bool*) – If blob\_name is a prefix, delete all files matching prefix.
- **ignore\_if\_missing** (*bool*) – if True, then return success even if the blob does not exist.

**class** airflow.contrib.operators.winrm\_operator.**WinRMOperator** (\*\*kwargs)  
 Bases: *airflow.models.BaseOperator*

WinRMOperator to execute commands on given remote host using the winrm\_hook.

### Parameters

- **winrm\_hook** (`airflow.contrib.hooks.winrm_hook.WinRMHook`) – predefined ssh\_hook to use for remote execution
- **ssh\_conn\_id** (`str`) – connection id from airflow Connections
- **remote\_host** (`str`) – remote host to connect
- **command** (`str`) – command to execute on remote host. (templated)
- **timeout** (`int`) – timeout for executing the command.
- **do\_xcom\_push** (`bool`) – return the stdout which also get set in xcom by airflow platform

### Sensors

**class** `airflow.contrib.sensors.aws_athena_sensor.AthenaSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Asks for the state of the Query until it reaches a failure state or success state. If it fails, failing the task.

### Parameters

- **query\_execution\_id** (`str`) – query\_execution\_id to check the state of
- **max\_retries** (`int`) – Number of times to poll for query state before returning the current state, defaults to None
- **aws\_conn\_id** (`str`) – aws connection to use, defaults to 'aws\_default'
- **sleep\_time** (`int`) – Time to wait between two consecutive call to check query status on athena, defaults to 10

**poke** (`context`)

Function that the sensors defined while deriving this class should override.

**class** `airflow.contrib.sensors.aws_glue_catalog_partition_sensor.AwsGlueCatalogPartitionSensor`

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits for a partition to show up in AWS Glue Catalog.

### Parameters

- **table\_name** (`str`) – The name of the table to wait for, supports the dot notation (`my_database.my_table`)
- **expression** (`str`) – The partition clause to wait for. This is passed as is to the AWS Glue Catalog API's `get_partitions` function, and supports SQL like notation as in `ds='2015-01-01' AND type='value'` and comparison operators as in `"ds>=2015-01-01"`. See <https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-catalog-partitions.html> #aws-glue-api-catalog-partitions-GetPartitions
- **aws\_conn\_id** (`str`) – ID of the Airflow connection where credentials and extra configuration are stored
- **region\_name** (`str`) – Optional aws region name (example: `us-east-1`). Uses region from connection if not specified.
- **database\_name** (`str`) – The name of the catalog database where the partitions reside.
- **poke\_interval** (`int`) – Time in seconds that the job should wait in between each tries

**get\_hook()**  
Gets the AwsGlueCatalogHook

**poke(context)**  
Checks for existence of the partition in the AWS Glue Catalog table

**class** airflow.contrib.sensors.aws\_redshift\_cluster\_sensor.**AwsRedshiftClusterSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Waits for a Redshift cluster to reach a specific status.

#### Parameters

- **cluster\_identifier** (*str*) – The identifier for the cluster being pinged.
- **target\_status** (*str*) – The cluster status desired.

**poke(context)**  
Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.azure\_cosmos\_sensor.**AzureCosmosDocumentSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Checks for the existence of a document which matches the given query in CosmosDB. Example:

```
>>> azure_cosmos_sensor = AzureCosmosDocumentSensor(database_name="somedatabase_
↪name",
...           collection_name="somecollection_name",
...           document_id="unique-doc-id",
...           azure_cosmos_conn_id="azure_cosmos_default",
...           task_id="azure_cosmos_sensor")
```

**poke(context)**  
Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.bash\_sensor.**BashSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Executes a bash command/script and returns True if and only if the return code is 0.

#### Parameters

- **bash\_command** (*str*) – The command, set of commands or reference to a bash script (must be '.sh') to be executed.
- **env** (*dict*) – If env is not None, it must be a mapping that defines the environment variables for the new process; these are used instead of inheriting the current process environment, which is the default behavior. (templated)
- **output\_encoding** (*str*) – output encoding of bash command.

**poke(context)**  
Execute the bash command in a temporary directory which will be cleaned afterwards

**class** airflow.contrib.sensors.bigquery\_sensor.**BigQueryTableSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Checks for the existence of a table in Google Bigquery.

#### Parameters

- **project\_id** (*str*) – The Google cloud project in which to look for the table. The connection supplied to the hook must provide access to the specified project.

- **dataset\_id** (*str*) – The name of the dataset in which to look for the table. storage bucket.
- **table\_id** (*str*) – The name of the table to check the existence of.
- **bigquery\_conn\_id** (*str*) – The connection ID to use when connecting to Google Big-Query.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.cassandra\_record\_sensor.**CassandraRecordSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Checks for the existence of a record in a Cassandra cluster.

For example, if you want to wait for a record that has values ‘v1’ and ‘v2’ for each primary keys ‘p1’ and ‘p2’ to be populated in keyspace ‘k’ and table ‘t’, instantiate it as follows:

```
>>> cassandra_sensor = CassandraRecordSensor(table="k.t",
...                                         keys={"p1": "v1", "p2": "v2"},
...                                         cassandra_conn_id="cassandra_default",
...                                         task_id="cassandra_sensor")
```

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.cassandra\_table\_sensor.**CassandraTableSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Checks for the existence of a table in a Cassandra cluster.

For example, if you want to wait for a table called ‘t’ to be created in a keyspace ‘k’, instantiate it as follows:

```
>>> cassandra_sensor = CassandraTableSensor(table="k.t",
...                                         cassandra_conn_id="cassandra_default",
...                                         task_id="cassandra_sensor")
```

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.celery\_queue\_sensor.**CeleryQueueSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Waits for a Celery queue to be empty. By default, in order to be considered empty, the queue must not have any tasks in the reserved, scheduled or active states.

#### Parameters

- **celery\_queue** (*str*) – The name of the Celery queue to wait for.
- **target\_task\_id** (*str*) – Task id for checking

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.datadog\_sensor.**DatadogSensor** (\*\*kwargs)  
Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

A sensor to listen, with a filter, to datadog event streams and determine if some event was emitted.

Depends on the datadog API, which has to be deployed on the same server where Airflow runs.

### Parameters

- **datadog\_conn\_id** – The connection to datadog, containing metadata for api keys.
- **datadog\_conn\_id** – str

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** `airflow.contrib.sensors.weekday_sensor.DayOfWeekSensor` (*\*\*kwargs*)

Bases: `airflow.sensors.base_sensor_operator.BaseSensorOperator`

Waits until the first specified day of the week. For example, if the execution day of the task is '2018-12-22' (Saturday) and you pass 'FRIDAY', the task will wait until next Friday.

**Example** (with single day):

```
weekend_check = DayOfWeekSensor(
    task_id='weekend_check',
    week_day='Saturday',
    use_task_execution_day=True,
    dag=dag)
```

**Example** (with multiple day using set):

```
weekend_check = DayOfWeekSensor(
    task_id='weekend_check',
    week_day={'Saturday', 'Sunday'},
    use_task_execution_day=True,
    dag=dag)
```

**Example** (with WeekDay enum):

```
# import WeekDay Enum
from airflow.contrib.utils.weekday import WeekDay

weekend_check = DayOfWeekSensor(
    task_id='weekend_check',
    week_day={WeekDay.SATURDAY, WeekDay.SUNDAY},
    use_task_execution_day=True,
    dag=dag)
```

### Parameters

- **week\_day** (*set or str or airflow.contrib.utils.weekday.WeekDay*) – Day of the week to check (full name). Optionally, a set of days can also be provided using a set. Example values:
  - "MONDAY",
  - {"Saturday", "Sunday"}
  - {WeekDay.TUESDAY}
  - {WeekDay.SATURDAY, WeekDay.SUNDAY}
- **use\_task\_execution\_day** (*bool*) – If True, uses task's execution day to compare with week\_day. Execution Date is Useful for backfilling. If False, uses system's day of the week. Useful when you don't want to run anything on weekdays on the system.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.emr\_base\_sensor.**EmrBaseSensor** (\*\*kwargs)

Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Contains general sensor behavior for EMR. Subclasses should implement `get_emr_response()` and `state_from_response()` methods. Subclasses should also implement `NON_TERMINAL_STATES` and `FAILED_STATE` constants.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.emr\_job\_flow\_sensor.**EmrJobFlowSensor** (\*\*kwargs)

Bases: *airflow.contrib.sensors.emr\_base\_sensor.EmrBaseSensor*

Asks for the state of the JobFlow until it reaches a terminal state. If it fails the sensor errors, failing the task.

**Parameters** `job_flow_id` (*str*) – job\_flow\_id to check the state of

**class** airflow.contrib.sensors.emr\_step\_sensor.**EmrStepSensor** (\*\*kwargs)

Bases: *airflow.contrib.sensors.emr\_base\_sensor.EmrBaseSensor*

Asks for the state of the step until it reaches a terminal state. If it fails the sensor errors, failing the task.

**Parameters**

- `job_flow_id` (*str*) – job\_flow\_id which contains the step check the state of
- `step_id` (*str*) – step to check the state of

**class** airflow.contrib.sensors.ftp\_sensor.**FTPSSensor** (\*\*kwargs)

Bases: *airflow.contrib.sensors.ftp\_sensor.FTPSensor*

Waits for a file or directory to be present on FTP over SSL.

**class** airflow.contrib.sensors.ftp\_sensor.**FTPSensor** (\*\*kwargs)

Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Waits for a file or directory to be present on FTP.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**template\_fields** = ('path',)

Errors that are transient in nature, and where action can be retried

**class** airflow.contrib.sensors.file\_sensor.**FileSensor** (\*\*kwargs)

Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Waits for a file or folder to land in a filesystem.

If the path given is a directory then this sensor will only return true if any files exist inside it (either directly, or within a subdirectory)

**Parameters**

- `fs_conn_id` (*str*) – reference to the File (path) connection id
- `filepath` – File or folder name (relative to the base path set within the connection)

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.gcs_sensor.GoogleCloudStorageObjectSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Checks for the existence of a file in Google Cloud Storage.

#### Parameters

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.
- **google\_cloud\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.gcs_sensor.GoogleCloudStorageObjectUpdatedSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Checks if an object is updated in Google Cloud Storage.

#### Parameters

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to download in the Google cloud storage bucket.
- **ts\_func** (*function*) – Callback for defining the update condition. The default callback returns `execution_date + schedule_interval`. The callback takes the context as parameter.
- **google\_cloud\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.gcs_sensor.GoogleCloudStoragePrefixSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Checks for the existence of a files at prefix in Google Cloud Storage bucket.

#### Parameters

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **prefix** (*str*) – The name of the prefix to check in the Google cloud storage bucket.
- **google\_cloud\_conn\_id** (*str*) – The connection ID to use when connecting to Google cloud storage.
- **delegate\_to** (*str*) – The account to impersonate, if any. For this to work, the service account making the request must have domain-wide delegation enabled.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.imap_attachment_sensor.ImapAttachmentSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Waits for a specific attachment on a mail server.

**Parameters**

- **attachment\_name** (*str*) – The name of the attachment that will be checked.
- **check\_regex** (*bool*) – If set to True the attachment's name will be parsed as regular expression. Through this you can get a broader set of attachments that it will look for than just only the equality of the attachment name. The default value is False.
- **mail\_folder** (*str*) – The mail folder in where to search for the attachment. The default value is 'INBOX'.
- **conn\_id** (*str*) – The connection to run the sensor against. The default value is 'imap\_default'.

**poke** (*context*)

Pokes for a mail attachment on the mail server.

**Parameters** **context** (*dict*) – The context that is being provided when poking.

**Returns** True if attachment with the given name is present and False if not.

**Return type** *bool*

**class** airflow.contrib.sensors.jira\_sensor.**JiraSensor** (*\*\*kwargs*)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*  
 Monitors a jira ticket for any change.

**Parameters**

- **jira\_conn\_id** (*str*) – reference to a pre-defined Jira Connection
- **method\_name** (*str*) – method name from jira-python-sdk to be execute
- **method\_params** (*dict*) – parameters for the method method\_name
- **result\_processor** (*function*) – function that return boolean and act as a sensor response

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.jira\_sensor.**JiraTicketSensor** (*\*\*kwargs*)  
 Bases: *airflow.contrib.sensors.jira\_sensor.JiraSensor*  
 Monitors a jira ticket for given change in terms of function.

**Parameters**

- **jira\_conn\_id** (*str*) – reference to a pre-defined Jira Connection
- **ticket\_id** (*str*) – id of the ticket to be monitored
- **field** (*str*) – field of the ticket to be monitored
- **expected\_value** (*str*) – expected value of the field
- **result\_processor** (*function*) – function that return boolean and act as a sensor response

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.mongo\_sensor.**MongoSensor** (*\*\*kwargs*)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*  
 Checks for the existence of a document which matches the given query in MongoDB. Example:



```
>>> mongo_sensor = MongoSensor(collection="coll",
...                             query={"key": "value"},
...                             mongo_conn_id="mongo_default",
...                             task_id="mongo_sensor")
```

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.pubsub_sensor.PubSubPullSensor(**kwargs)
Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Pulls messages from a PubSub subscription and passes them through XCom.

This sensor operator will pull up to `max_messages` messages from the specified PubSub subscription. When the subscription returns messages, the `poke` method's criteria will be fulfilled and the messages will be returned from the operator and passed through XCom for downstream tasks.

If `ack_messages` is set to `True`, messages will be immediately acknowledged before being returned, otherwise, downstream tasks will be responsible for acknowledging them.

`project` and `subscription` are templated so you can use variables in them.

**execute** (*context*)

Overridden to allow messages to be passed

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.python_sensor.PythonSensor(**kwargs)
Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Waits for a Python callable to return `True`.

User could put input argument in `templates_dict` e.g `templates_dict = {'start_ds': 1970}` and access the argument by calling `kwargs['templates_dict']['start_ds']` in the the callable

#### Parameters

- **python\_callable** (*python callable*) – A reference to an object that is callable
- **op\_kwargs** (*dict*) – a dictionary of keyword arguments that will get unpacked in your function
- **op\_args** (*list*) – a list of positional arguments that will get unpacked when calling your callable
- **provide\_context** (*bool*) – if set to `true`, Airflow will pass a set of keyword arguments that can be used in your function. This set of kwargs correspond exactly to what you can use in your jinja templates. For this to work, you need to define `**kwargs` in your function header.
- **templates\_dict** (*dict of str*) – a dictionary where the values are templates that will get templated by the Airflow engine sometime between `__init__` and `execute` takes place and are made available in your callable's context after the template has been applied.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.qubole_sensor.QuboleFileSensor(**kwargs)
Bases: airflow.contrib.sensors.qubole_sensor.QuboleSensor
```

Wait for a file or folder to be present in cloud storage and check for its presence via QDS APIs

**Parameters**

- **qubole\_conn\_id** (*str*) – Connection id which consists of qds auth\_token
- **data** (a *JSON object*) – a JSON object containing payload, whose presence needs to be checked Check this [example](#) for sample payload structure.

---

**Note:** Both data and qubole\_conn\_id fields support templating. You can also use .txt files for template-driven use cases.

---

**class** airflow.contrib.sensors.qubole\_sensor.**QubolePartitionSensor** (\*\*kwargs)  
 Bases: *airflow.contrib.sensors.qubole\_sensor.QuboleSensor*

Wait for a Hive partition to show up in QHS (Qubole Hive Service) and check for its presence via QDS APIs

**Parameters**

- **qubole\_conn\_id** (*str*) – Connection id which consists of qds auth\_token
- **data** (a *JSON object*) – a JSON object containing payload, whose presence needs to be checked. Check this [example](#) for sample payload structure.

---

**Note:** Both data and qubole\_conn\_id fields support templating. You can also use .txt files for template-driven use cases.

---

**class** airflow.contrib.sensors.qubole\_sensor.**QuboleSensor** (\*\*kwargs)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Base class for all Qubole Sensors

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.redis\_key\_sensor.**RedisKeySensor** (\*\*kwargs)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Checks for the existence of a key in a Redis

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.sftp\_sensor.**SFTPSENSOR** (\*\*kwargs)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Waits for a file or directory to be present on SFTP.

**Parameters**

- **path** (*str*) – Remote file or directory path
- **sftp\_conn\_id** (*str*) – The connection to run the sensor against

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

**class** airflow.contrib.sensors.sagemaker\_base\_sensor.**SageMakerBaseSensor** (\*\*kwargs)  
 Bases: *airflow.sensors.base\_sensor\_operator.BaseSensorOperator*

Contains general sensor behavior for SageMaker. Subclasses should implement get\_sagemaker\_response() and state\_from\_response() methods. Subclasses should also implement NON\_TERMINAL\_STATES and FAILED\_STATE methods.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.sagemaker_endpoint_sensor.SageMakerEndpointSensor (**kwargs)
    Bases: airflow.contrib.sensors.sagemaker_base_sensor.SageMakerBaseSensor
```

Asks for the state of the endpoint state until it reaches a terminal state. If it fails the sensor errors, the task fails.

**Parameters** **job\_name** (*str*) – job\_name of the endpoint instance to check the state of

```
class airflow.contrib.sensors.sagemaker_training_sensor.SageMakerTrainingSensor (**kwargs)
    Bases: airflow.contrib.sensors.sagemaker_base_sensor.SageMakerBaseSensor
```

Asks for the state of the training state until it reaches a terminal state. If it fails the sensor errors, failing the task.

**Parameters**

- **job\_name** (*str*) – name of the SageMaker training job to check the state of
- **print\_log** (*bool*) – if the operator should print the cloudwatch log

```
class airflow.contrib.sensors.sagemaker_transform_sensor.SageMakerTransformSensor (**kwargs)
    Bases: airflow.contrib.sensors.sagemaker_base_sensor.SageMakerBaseSensor
```

Asks for the state of the transform state until it reaches a terminal state. The sensor will error if the job errors, throwing a AirflowException containing the failure reason.

**Parameters** **job\_name** (*string*) – job\_name of the transform job instance to check the state of

```
class airflow.contrib.sensors.sagemaker_tuning_sensor.SageMakerTuningSensor (**kwargs)
    Bases: airflow.contrib.sensors.sagemaker_base_sensor.SageMakerBaseSensor
```

Asks for the state of the tuning state until it reaches a terminal state. The sensor will error if the job errors, throwing a AirflowException containing the failure reason.

**Parameters** **job\_name** (*str*) – job\_name of the tuning instance to check the state of

```
class airflow.contrib.sensors.wasb_sensor.WasbBlobSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Waits for a blob to arrive on Azure Blob Storage.

**Parameters**

- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.
- **wasb\_conn\_id** (*str*) – Reference to the wasb connection.
- **check\_options** (*dict*) – Optional keyword arguments that *WasbHook.check\_for\_blob()* takes.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

```
class airflow.contrib.sensors.wasb_sensor.WasbPrefixSensor (**kwargs)
    Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator
```

Waits for blobs matching a prefix to arrive on Azure Blob Storage.

**Parameters**

- **container\_name** (*str*) – Name of the container.
- **prefix** (*str*) – Prefix of the blob.
- **wasb\_conn\_id** (*str*) – Reference to the wasb connection.

- **check\_options** (*dict*) – Optional keyword arguments that *WasbHook.check\_for\_prefix()* takes.

**poke** (*context*)

Function that the sensors defined while deriving this class should override.

## 3.21.2 Macros

Here's a list of variables and macros that can be used in templates

### 3.21.2.1 Default Variables

The Airflow engine passes a few variables by default that are accessible in all templates

Variable	Description
{{ ds }}	the execution date as YYYY-MM-DD
{{ ds_nodash }}	the execution date as YYYYMMDD
{{ prev_ds }}	the previous execution date as YYYY-MM-DD if {{ ds }} is 2018-01-08 and scheduled
{{ prev_ds_nodash }}	the previous execution date as YYYYMMDD if exists, else None
{{ next_ds }}	the next execution date as YYYY-MM-DD if {{ ds }} is 2018-01-01 and scheduled
{{ next_ds_nodash }}	the next execution date as YYYYMMDD if exists, else None
{{ yesterday_ds }}	the day before the execution date as YYYY-MM-DD
{{ yesterday_ds_nodash }}	the day before the execution date as YYYYMMDD
{{ tomorrow_ds }}	the day after the execution date as YYYY-MM-DD
{{ tomorrow_ds_nodash }}	the day after the execution date as YYYYMMDD
{{ ts }}	same as <code>execution_date.isoformat()</code> . Example: 2018-01-01T00:00:00+00:00
{{ ts_nodash }}	same as <code>ts</code> without <code>-</code> , <code>:</code> and <code>TimeZone</code> info. Example: 20180101T000000
{{ ts_nodash_with_tz }}	same as <code>ts</code> without <code>-</code> and <code>:</code> . Example: 20180101T000000+0000
{{ execution_date }}	the <code>execution_date</code> ( <code>pendulum.Pendulum</code> )
{{ prev_execution_date }}	the previous execution date (if available) ( <code>pendulum.Pendulum</code> )
{{ next_execution_date }}	the next execution date ( <code>pendulum.Pendulum</code> )
{{ dag }}	the DAG object
{{ task }}	the Task object
{{ macros }}	a reference to the macros package, described below
{{ task_instance }}	the <code>task_instance</code> object
{{ end_date }}	same as {{ ds }}
{{ latest_date }}	same as {{ ds }}
{{ ti }}	same as {{ task_instance }}
{{ params }}	a reference to the user-defined params dictionary which can be overridden by the dictionary
{{ var.value.my_var }}	global defined variables represented as a dictionary
{{ var.json.my_var.path }}	global defined variables represented as a dictionary with deserialized JSON object, append
{{ task_instance_key_str }}	a unique, human-readable key to the task instance formatted <code>{dag_id}_{task_id}_{</code>
{{ conf }}	the full configuration object located at <code>airflow.configuration.conf</code> which repre
{{ run_id }}	the <code>run_id</code> of the current DAG run
{{ dag_run }}	a reference to the <code>DagRun</code> object
{{ test_mode }}	whether the task instance was called using the CLI's test subcommand

Note that you can access the object's attributes and methods with simple dot notation. Here are some examples of what is possible: `{{ task.owner }}`, `{{ task.task_id }}`, `{{ ti.hostname }}`, ... Refer to the models documentation for more information on the objects' attributes and methods.

The `var` template variable allows you to access variables defined in Airflow's UI. You can access them as either

plain-text or JSON. If you use JSON, you are also able to walk nested structures, such as dictionaries like: `{{ var.json.my_dict_var.key1 }}`

### 3.21.2.2 Macros

Macros are a way to expose objects to your templates and live under the `macros` namespace in your templates.

A few commonly used libraries and methods are made available.

Variable	Description
<code>macros.datetime</code>	The standard lib's <code>datetime.datetime</code>
<code>macros.timedelta</code>	The standard lib's <code>datetime.timedelta</code>
<code>macros.dateutil</code>	A reference to the <code>dateutil</code> package
<code>macros.time</code>	The standard lib's <code>time</code>
<code>macros.uuid</code>	The standard lib's <code>uuid</code>
<code>macros.random</code>	The standard lib's <code>random</code>

Some airflow specific macros are also defined:

`airflow.macros.ds_add(ds, days)`

Add or subtract days from a YYYY-MM-DD

#### Parameters

- **ds** (*str*) – anchor date in YYYY-MM-DD format to add to
- **days** (*int*) – number of days to add to the ds, you can use negative values

```
>>> ds_add('2015-01-01', 5)
'2015-01-06'
>>> ds_add('2015-01-06', -5)
'2015-01-01'
```

`airflow.macros.ds_format(ds, input_format, output_format)`

Takes an input string and outputs another string as specified in the output format

#### Parameters

- **ds** (*str*) – input string which contains a date
- **input\_format** (*str*) – input string format. E.g. `%Y-%m-%d`
- **output\_format** (*str*) – output string format E.g. `%Y-%m-%d`

```
>>> ds_format('2015-01-01', "%Y-%m-%d", "%m-%d-%y")
'01-01-15'
>>> ds_format('1/5/2015', "%m/%d/%Y", "%Y-%m-%d")
'2015-01-05'
```

`airflow.macros.random()` → x in the interval [0, 1).

`airflow.macros.hive.closest_ds_partition(table, ds, before=True, schema='default', metastore_conn_id='metastore_default')`

This function finds the date in a list closest to the target date. An optional parameter can be given to get the closest before or after.

#### Parameters

- **table** (*str*) – A hive table name

- **ds** (*list[datetime.date]*) – A datestamp %Y-%m-%d e.g. yyyy-mm-dd
- **before** (*bool or None*) – closest before (True), after (False) or either side of ds

**Returns** The closest date

**Return type** *str* or *None*

```
>>> tbl = 'airflow.static_babynames_partitioned'
>>> closest_ds_partition(tbl, '2015-01-02')
'2015-01-01'
```

`airflow.macros.hive.max_partition` (*table*, *schema*='default', *field*=None, *filter\_map*=None, *metastore\_conn\_id*='metastore\_default')

Gets the max partition for a table.

#### Parameters

- **schema** (*str*) – The hive schema the table lives in
- **table** (*str*) – The hive table you are interested in, supports the dot notation as in “my\_database.my\_table”, if a dot is found, the schema param is disregarded
- **metastore\_conn\_id** (*str*) – The hive connection you are interested in. If your default is set you don’t need to use this parameter.
- **filter\_map** (*map*) – partition\_key:partition\_value map used for partition filtering, e.g. {'key1': 'value1', 'key2': 'value2'}. Only partitions matching all partition\_key:partition\_value pairs will be considered as candidates of max partition.
- **field** (*str*) – the field to get the max value from. If there’s only one partition field, this will be inferred

```
>>> max_partition('airflow.static_babynames_partitioned')
'2015-01-01'
```

### 3.21.3 Models

Models are built on top of the SQLAlchemy ORM Base class, and instances are persisted in the database.

**class** `airflow.models.BaseOperator` (\*\*kwargs)

Bases: `airflow.utils.log.logging_mixin.LoggingMixin`

Abstract base class for all operators. Since operators create objects that become nodes in the dag, BaseOperator contains many recursive methods for dag crawling behavior. To derive this class, you are expected to override the constructor as well as the ‘execute’ method.

Operators derived from this class should perform or trigger certain tasks synchronously (wait for completion). Example of operators could be an operator that runs a Pig job (PigOperator), a sensor operator that waits for a partition to land in Hive (HiveSensorOperator), or one that moves data from Hive to MySQL (Hive2MySQLOperator). Instances of these operators (tasks) target specific operations, running specific scripts, functions or data transfers.

This class is abstract and shouldn’t be instantiated. Instantiating a class derived from this one results in the creation of a task object, which ultimately becomes a node in DAG objects. Task dependencies should be set by using the `set_upstream` and/or `set_downstream` methods.

#### Parameters

- **task\_id** (*str*) – a unique, meaningful id for the task
- **owner** (*str*) – the owner of the task, using the unix username is recommended

- **retries** (*int*) – the number of retries that should be performed before failing the task
- **retry\_delay** (*datetime.timedelta*) – delay between retries
- **retry\_exponential\_backoff** (*bool*) – allow progressive longer waits between retries by using exponential backoff algorithm on retry delay (delay will be converted into seconds)
- **max\_retry\_delay** (*datetime.timedelta*) – maximum delay interval between retries
- **start\_date** (*datetime.datetime*) – The `start_date` for the task, determines the `execution_date` for the first task instance. The best practice is to have the `start_date` rounded to your DAG's `schedule_interval`. Daily jobs have their `start_date` some day at 00:00:00, hourly jobs have their `start_date` at 00:00 of a specific hour. Note that Airflow simply looks at the latest `execution_date` and adds the `schedule_interval` to determine the next `execution_date`. It is also very important to note that different tasks' dependencies need to line up in time. If task A depends on task B and their `start_date` are offset in a way that their `execution_date` don't line up, A's dependencies will never be met. If you are looking to delay a task, for example running a daily task at 2AM, look into the `TimeSensor` and `TimeDeltaSensor`. We advise against using dynamic `start_date` and recommend using fixed ones. Read the FAQ entry about `start_date` for more information.
- **end\_date** (*datetime.datetime*) – if specified, the scheduler won't go beyond this date
- **depends\_on\_past** (*bool*) – when set to true, task instances will run sequentially while relying on the previous task's schedule to succeed. The task instance for the `start_date` is allowed to run.
- **wait\_for\_downstream** (*bool*) – when set to true, an instance of task X will wait for tasks immediately downstream of the previous instance of task X to finish successfully before it runs. This is useful if the different instances of a task X alter the same asset, and this asset is used by tasks downstream of task X. Note that `depends_on_past` is forced to True wherever `wait_for_downstream` is used.
- **queue** (*str*) – which queue to target when running this job. Not all executors implement queue management, the CeleryExecutor does support targeting specific queues.
- **dag** (*airflow.models.DAG*) – a reference to the dag the task is attached to (if any)
- **priority\_weight** (*int*) – priority weight of this task against other task. This allows the executor to trigger higher priority tasks before others when things get backed up. Set `priority_weight` as a higher number for more important tasks.
- **weight\_rule** (*str*) – weighting method used for the effective total priority weight of the task. Options are: { `downstream` | `upstream` | `absolute` } default is `downstream` When set to `downstream` the effective weight of the task is the aggregate sum of all downstream descendants. As a result, upstream tasks will have higher weight and will be scheduled more aggressively when using positive weight values. This is useful when you have multiple dag run instances and desire to have all upstream tasks to complete for all runs before each dag can continue processing downstream tasks. When set to `upstream` the effective weight is the aggregate sum of all upstream ancestors. This is the opposite where downstream tasks have higher weight and will be scheduled more aggressively when using positive weight values. This is useful when you have multiple dag run instances and prefer to have each dag complete before starting upstream tasks of other dags. When set to `absolute`, the effective weight is the exact `priority_weight` specified without additional weighting. You may want to do this when you know exactly what priority weight



each task should have. Additionally, when set to `absolute`, there is bonus effect of significantly speeding up the task creation process as for very large DAGs. Options can be set as string or using the constants defined in the static class `airflow.utils.WeightRule`

- **pool** (*str*) – the slot pool this task should run in, slot pools are a way to limit concurrency for certain tasks
- **sla** (*datetime.timedelta*) – time by which the job is expected to succeed. Note that this represents the `timedelta` after the period is closed. For example if you set an SLA of 1 hour, the scheduler would send an email soon after 1:00AM on the 2016-01-02 if the 2016-01-01 instance has not succeeded yet. The scheduler pays special attention for jobs with an SLA and sends alert emails for sla misses. SLA misses are also recorded in the database for future reference. All tasks that share the same SLA time get bundled in a single email, sent soon after that time. SLA notification are sent once and only once for each task instance.
- **execution\_timeout** (*datetime.timedelta*) – max time allowed for the execution of this task instance, if it goes beyond it will raise and fail.
- **on\_failure\_callback** (*callable*) – a function to be called when a task instance of this task fails. a context dictionary is passed as a single parameter to this function. Context contains references to related objects to the task instance and is documented under the macros section of the API.
- **on\_retry\_callback** (*callable*) – much like the `on_failure_callback` except that it is executed when retries occur.
- **on\_success\_callback** (*callable*) – much like the `on_failure_callback` except that it is executed when the task succeeds.
- **trigger\_rule** (*str*) – defines the rule by which dependencies are applied for the task to get triggered. Options are: { `all_success` | `all_failed` | `all_done` | `one_success` | `one_failed` | `none_failed` | `dummy` } default is `all_success`. Options can be set as string or using the constants defined in the static class `airflow.utils.TriggerRule`
- **resources** (*dict*) – A map of resource parameter names (the argument names of the Resources constructor) to their values.
- **run\_as\_user** (*str*) – unix username to impersonate while running the task
- **task\_concurrency** (*int*) – When set, a task will be able to limit the concurrent runs across `execution_dates`
- **executor\_config** (*dict*) – Additional task-level configuration parameters that are interpreted by a specific executor. Parameters are namespaced by the name of executor.

**Example:** to run this task in a specific docker container through the `KubernetesExecutor`

```
MyOperator(...,
    executor_config={
        "KubernetesExecutor":
            {"image": "myCustomDockerImage"}
    }
)
```

- **do\_xcom\_push** (*bool*) – if True, an XCom is pushed containing the Operator's result

**clear** (*\*\*kwargs*)

Clears the state of task instances associated with the task, following the parameters specified.



**dag**  
Returns the Operator's DAG if set, otherwise raises an error

**deps**  
Returns the list of dependencies for the operator. These differ from execution context dependencies in that they are specific to tasks and can be extended/overridden by subclasses.

**downstream\_list**  
@property: list of tasks directly downstream

**execute** (*context*)  
This is the main method to derive when creating an operator. Context is the same dictionary used as when rendering jinja templates.  
  
Refer to `get_template_context` for more context.

**get\_direct\_relative\_ids** (*upstream=False*)  
Get the direct relative ids to the current task, upstream or downstream.

**get\_direct\_relatives** (*upstream=False*)  
Get the direct relatives to the current task, upstream or downstream.

**get\_flat\_relative\_ids** (*upstream=False, found\_descendants=None*)  
Get a flat list of relatives' ids, either upstream or downstream.

**get\_flat\_relatives** (*upstream=False*)  
Get a flat list of relatives, either upstream or downstream.

**get\_task\_instances** (*session, start\_date=None, end\_date=None*)  
Get a set of task instance related to this task for a specific date range.

**has\_dag** ()  
Returns True if the Operator has been assigned to a DAG.

**on\_kill** ()  
Override this method to cleanup subprocesses when a task instance gets killed. Any use of the threading, subprocess or multiprocessing module within an operator needs to be cleaned up or it will leave ghost processes behind.

**post\_execute** (*context, \*args, \*\*kwargs*)  
This hook is triggered right after `self.execute()` is called. It is passed the execution context and any results returned by the operator.

**pre\_execute** (*context, \*args, \*\*kwargs*)  
This hook is triggered right before `self.execute()` is called.

**prepare\_template** ()  
Hook that is triggered after the templated fields get replaced by their content. If you need your operator to alter the content of the file before the template is rendered, it should override this method to do so.

**render\_template** (*attr, content, context*)  
Renders a template either from a file or directly in a field, and returns the rendered result.

**render\_template\_from\_field** (*attr, content, context, jinja\_env*)  
Renders a template from a field. If the field is a string, it will simply render the string and return the result. If it is a collection or nested set of collections, it will traverse the structure and render all elements in it. If the field has another type, it will return it as it is.

**run** (*start\_date=None, end\_date=None, ignore\_first\_depends\_on\_past=False, ignore\_ti\_state=False, mark\_success=False*)  
Run a set of task instances for a date range.

**schedule\_interval**

The schedule interval of the DAG always wins over individual tasks so that tasks within a DAG always line up. The task still needs a `schedule_interval` as it may not be attached to a DAG.

**set\_downstream** (*task\_or\_task\_list*)

Set a task or a task list to be directly downstream from the current task.

**set\_upstream** (*task\_or\_task\_list*)

Set a task or a task list to be directly upstream from the current task.

**upstream\_list**

@property: list of tasks directly upstream

**xcom\_pull** (*context, task\_ids=None, dag\_id=None, key=u'return\_value', include\_prior\_dates=None*)

See TaskInstance.xcom\_pull()

**xcom\_push** (*context, key, value, execution\_date=None*)

See TaskInstance.xcom\_push()

```
class airflow.models.DAG(dag_id, description=u'', schedule_interval=datetime.timedelta(1),
                        start_date=None, end_date=None, full_filepath=None,
                        template_searchpath=None, user_defined_macros=None,
                        user_defined_filters=None, default_args=None, concurrency=16,
                        max_active_runs=16, dagrun_timeout=None, sla_miss_callback=None,
                        default_view=None, orientation='LR', catchup=True,
                        on_success_callback=None, on_failure_callback=None,
                        params=None, access_control=None)
```

Bases: airflow.dag.base\_dag.BaseDag, airflow.utils.log.logging\_mixin.LoggingMixin

A dag (directed acyclic graph) is a collection of tasks with directional dependencies. A dag also has a schedule, a start date and an end date (optional). For each schedule, (say daily or hourly), the DAG needs to run each individual tasks as their dependencies are met. Certain tasks have the property of depending on their own past, meaning that they can't run until their previous schedule (and upstream tasks) are completed.

DAGs essentially act as namespaces for tasks. A `task_id` can only be added once to a DAG.

**Parameters**

- **dag\_id** (*str*) – The id of the DAG
- **description** (*str*) – The description for the DAG to e.g. be shown on the webserver
- **schedule\_interval** (*datetime.timedelta or dateutil.relativedelta.relativedelta or str that acts as a cron expression*) – Defines how often that DAG runs, this timedelta object gets added to your latest task instance's `execution_date` to figure out the next schedule
- **start\_date** (*datetime.datetime*) – The timestamp from which the scheduler will attempt to backfill
- **end\_date** (*datetime.datetime*) – A date beyond which your DAG won't run, leave to None for open ended scheduling
- **template\_searchpath** (*str or list[str]*) – This list of folders (non relative) defines where jinja will look for your templates. Order matters. Note that jinja/airflow includes the path of your DAG file by default
- **user\_defined\_macros** (*dict*) – a dictionary of macros that will be exposed in your jinja templates. For example, passing `dict(foo='bar')` to this argument allows you to `{{ foo }}` in all jinja templates related to this DAG. Note that you can pass any type of object here.

- **user\_defined\_filters** (*dict*) – a dictionary of filters that will be exposed in your jinja templates. For example, passing `dict(hello=lambda name: 'Hello %s' % name)` to this argument allows you to `{{ 'world' | hello }}` in all jinja templates related to this DAG.
- **default\_args** (*dict*) – A dictionary of default parameters to be used as constructor keyword parameters when initialising operators. Note that operators have the same hook, and precede those defined here, meaning that if your dict contains `'depends_on_past': True` here and `'depends_on_past': False` in the operator's call `default_args`, the actual value will be `False`.
- **params** (*dict*) – a dictionary of DAG level parameters that are made accessible in templates, namespaced under *params*. These params can be overridden at the task level.
- **concurrency** (*int*) – the number of task instances allowed to run concurrently
- **max\_active\_runs** (*int*) – maximum number of active DAG runs, beyond this number of DAG runs in a running state, the scheduler won't create new active DAG runs
- **dagrun\_timeout** (*datetime.timedelta*) – specify how long a DagRun should be up before timing out / failing, so that new DagRuns can be created
- **sla\_miss\_callback** (*types.FunctionType*) – specify a function to call when reporting SLA timeouts.
- **default\_view** (*str*) – Specify DAG default view (tree, graph, duration, gantt, landing\_times)
- **orientation** (*str*) – Specify DAG orientation in graph view (LR, TB, RL, BT)
- **catchup** (*bool*) – Perform scheduler catchup (or only run latest)? Defaults to True
- **on\_failure\_callback** (*callable*) – A function to be called when a DagRun of this dag fails. A context dictionary is passed as a single parameter to this function.
- **on\_success\_callback** (*callable*) – Much like the `on_failure_callback` except that it is executed when the dag succeeds.
- **access\_control** (*dict*) – Specify optional DAG-level permissions, e.g., `{'role1': {'can_dag_read'}, 'role2': {'can_dag_read', 'can_dag_edit'}}`

**add\_task** (*task*)

Add a task to the DAG

**Parameters** *task* (*task*) – the task you want to add

**add\_tasks** (*tasks*)

Add a list of tasks to the DAG

**Parameters** *tasks* (*list of tasks*) – a list of tasks you want to add

**clear** (*\*\*kwargs*)

Clears a set of task instances associated with the current dag for a specified date range.

**cli** ()

Exposes a CLI specific to this DAG

**concurrency\_reached**

Returns a boolean indicating whether the concurrency limit for this DAG has been reached

**create\_dagrun** (*\*\*kwargs*)

Creates a dag run from this dag including the tasks associated with this dag. Returns the dag run.

**Parameters**

- **run\_id** (*str*) – defines the the run id for this dag run
- **execution\_date** (*datetime.datetime*) – the execution date of this dag run
- **state** (*airflow.utils.state.State*) – the state of the dag run
- **start\_date** (*datetime*) – the date this dag run should be evaluated
- **external\_trigger** (*bool*) – whether this dag run is externally triggered
- **session** (*sqlalchemy.orm.session.Session*) – database session

**static deactivate\_stale\_dags** (\*args, \*\*kwargs)

Deactivate any DAGs that were last touched by the scheduler before the expiration date. These DAGs were likely deleted.

**Parameters** **expiration\_date** (*datetime*) – set inactive DAGs that were touched before this time

**Returns** None

**static deactivate\_unknown\_dags** (\*args, \*\*kwargs)

Given a list of known DAGs, deactivate any other DAGs that are marked as active in the ORM

**Parameters** **active\_dag\_ids** (*list[unicode]*) – list of DAG IDs that are active

**Returns** None

**filepath**

File location of where the dag object is instantiated

**folder**

Folder location of where the dag object is instantiated

**following\_schedule** (*dtm*)

Calculates the following schedule for this dag in UTC.

**Parameters** **dtm** – utc datetime

**Returns** utc datetime

**get\_active\_runs** (\*\*kwargs)

Returns a list of dag run execution dates currently running

**Parameters** **session** –

**Returns** List of execution dates

**get\_dagrun** (\*\*kwargs)

Returns the dag run for a given execution date if it exists, otherwise none.

**Parameters**

- **execution\_date** – The execution date of the DagRun to find.
- **session** –

**Returns** The DagRun if found, otherwise None.

**get\_default\_view** ()

This is only there for backward compatible jinja2 templates

**get\_num\_active\_runs** (\*\*kwargs)

Returns the number of active “running” dag runs

**Parameters**

- **external\_trigger** (*bool*) – True for externally triggered active dag runs
- **session** –

**Returns** number greater than 0 for active dag runs

**static get\_num\_task\_instances** (*\*args, \*\*kwargs*)

Returns the number of task instances in the given DAG.

**Parameters**

- **session** – ORM session
- **dag\_id** (*unicode*) – ID of the DAG to get the task concurrency of
- **task\_ids** (*list[unicode]*) – A list of valid task IDs for the given DAG
- **states** (*list[state]*) – A list of states to filter by if supplied

**Returns** The number of running tasks

**Return type** *int*

**get\_run\_dates** (*start\_date, end\_date=None*)

Returns a list of dates between the interval received as parameter using this dag's schedule interval. Returned dates can be used for execution dates.

**Parameters**

- **start\_date** (*datetime*) – the start date of the interval
- **end\_date** (*datetime*) – the end date of the interval, defaults to `timezone.utcnow()`

**Returns** a list of dates within the interval following the dag's schedule

**Return type** *list*

**get\_template\_env** ()

Returns a jinja2 Environment while taking into account the DAGs `template_searchpath`, `user_defined_macros` and `user_defined_filters`

**handle\_callback** (*\*\*kwargs*)

Triggers the appropriate callback depending on the value of success, namely the `on_failure_callback` or `on_success_callback`. This method gets the context of a single TaskInstance part of this DagRun and passes that to the callable along with a 'reason', primarily to differentiate DagRun failures.

**Parameters**

- **dagrun** – DagRun object
- **success** – Flag to specify if failure or success callback should be called
- **reason** – Completion reason
- **session** – Database session

**is\_fixed\_time\_schedule** ()

Figures out if the DAG schedule has a fixed time (e.g. 3 AM).

**Returns** True if the schedule has a fixed time, False if not.

**is\_paused**

Returns a boolean indicating whether this DAG is paused

**latest\_execution\_date**

Returns the latest date for which at least one dag run exists

**normalize\_schedule** (*dtm*)

Returns dtm + interval unless dtm is first interval then it returns dtm

**previous\_schedule** (*dtm*)

Calculates the previous schedule for this dag in UTC

**Parameters** *dtm* – utc datetime

**Returns** utc datetime

**run** (*start\_date=None, end\_date=None, mark\_success=False, local=False, executor=None, do\_not\_pickle=False, ignore\_task\_deps=False, ignore\_first\_depends\_on\_past=False, pool=None, delay\_on\_limit\_secs=1.0, verbose=False, conf=None, rerun\_failed\_tasks=False*)  
Runs the DAG.

**Parameters**

- **start\_date** (*datetime.datetime*) – the start date of the range to run
- **end\_date** (*datetime.datetime*) – the end date of the range to run
- **mark\_success** (*bool*) – True to mark jobs as succeeded without running them
- **local** (*bool*) – True to run the tasks using the LocalExecutor
- **executor** (*BaseExecutor*) – The executor instance to run the tasks
- **do\_not\_pickle** (*bool*) – True to avoid pickling DAG object and send to workers
- **ignore\_task\_deps** (*bool*) – True to skip upstream tasks
- **ignore\_first\_depends\_on\_past** (*bool*) – True to ignore depends\_on\_past dependencies for the first set of tasks only
- **pool** (*str*) – Resource pool to use
- **delay\_on\_limit\_secs** (*float*) – Time in seconds to wait before next attempt to run dag run when max\_active\_runs limit has been reached
- **verbose** (*bool*) – Make logging output more verbose
- **conf** (*dict*) – user defined dictionary passed from CLI

**set\_dependency** (*upstream\_task\_id, downstream\_task\_id*)

Simple utility method to set dependency between two tasks that already have been added to the DAG using `add_task()`

**sub\_dag** (*task\_regex, include\_downstream=False, include\_upstream=True*)

Returns a subset of the current dag as a deep copy of the current dag based on a regex that should match one or many tasks, and includes upstream and downstream neighbours based on the flag passed.

**subdags**

Returns a list of the subdag objects associated to this DAG

**sync\_to\_db** (*\*\*kwargs*)

Save attributes about this DAG to the DB. Note that this method can be called for both DAGs and SubDAGs. A SubDag is actually a SubDagOperator.

**Parameters**

- **dag** (*airflow.models.DAG*) – the DAG object to save to the DB
- **sync\_time** (*datetime*) – The time that the DAG should be marked as sync'ed

**Returns** None

**test\_cycle()**

Check to see if there are any cycles in the DAG. Returns False if no cycle found, otherwise raises exception.

**topological\_sort()**

Sorts tasks in topographical order, such that a task comes after any of its upstream dependencies.

Heavily inspired by: <http://blog.jupo.org/2012/04/06/topological-sorting-acyclic-directed-graphs/>

**Returns** list of tasks in topological order

**tree\_view()**

Shows an ascii tree representation of the DAG

**class** airflow.models.DagBag(*dag\_folder=None, executor=None, include\_examples=True*)

Bases: airflow.dag.base\_dag.BaseDagBag, airflow.utils.log.logging\_mixin.LoggingMixin

A dagbag is a collection of dags, parsed out of a folder tree and has high level configuration settings, like what database to use as a backend and what executor to use to fire off tasks. This makes it easier to run distinct environments for say production and development, tests, or for different teams or security profiles. What would have been system level settings are now dagbag level so that one system can run multiple, independent settings sets.

#### Parameters

- **dag\_folder** (*unicode*) – the folder to scan to find DAGs
- **executor** – the executor to use when executing task instances in this DagBag
- **include\_examples** (*bool*) – whether to include the examples that ship with airflow or not
- **has\_logged** – an instance boolean that gets flipped from False to True after a file has been skipped. This is to prevent overloading the user with logging messages about skipped files. Therefore only once per DagBag is a file logged being skipped.

**bag\_dag** (*dag, parent\_dag, root\_dag*)

Adds the DAG into the bag, recurses into sub dags. Throws AirflowDagCycleException if a cycle is detected in this dag or its subdags

**collect\_dags** (*dag\_folder=None, only\_if\_updated=True, include\_examples=True*)

Given a file path or a folder, this method looks for python modules, imports them and adds them to the dagbag collection.

Note that if a `.airflowignore` file is found while processing the directory, it will behave much like a `.gitignore`, ignoring files that match any of the regex patterns specified in the file.

**Note:** The patterns in `.airflowignore` are treated as un-anchored regexes, not shell-like glob patterns.

**dagbag\_report()**

Prints a report around DagBag loading stats

**get\_dag** (*dag\_id*)

Gets the DAG out of the dictionary, and refreshes it if expired

**kill\_zombies** (*\*\*kwargs*)

Fail given zombie tasks, which are tasks that haven't had a heartbeat for too long, in the current DagBag.

#### Parameters

- **zombies** (*airflow.utils.dag\_processing.SimpleTaskInstance*) – zombie task instances to kill.
- **session** (*sqlalchemy.orm.session.Session*) – DB session.

**process\_file** (*filepath*, *only\_if\_updated=True*, *safe\_mode=True*)

Given a path to a python module or zip file, this method imports the module and look for dag objects within it.

**size**()

**Returns** the amount of dags contained in this dagbag

**class** airflow.models.DagModel (\*\*kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

**create\_dagrun** (\*\*kwargs)

Creates a dag run from this dag including the tasks associated with this dag. Returns the dag run.

#### Parameters

- **run\_id** (*str*) – defines the the run id for this dag run
- **execution\_date** (*datetime.datetime*) – the execution date of this dag run
- **state** (*airflow.utils.state.State*) – the state of the dag run
- **start\_date** (*datetime.datetime*) – the date this dag run should be evaluated
- **external\_trigger** (*bool*) – whether this dag run is externally triggered
- **session** (*sqlalchemy.orm.session.Session*) – database session

**class** airflow.models.DagRun (\*\*kwargs)

Bases: sqlalchemy.ext.declarative.api.Base, airflow.utils.log.logging\_mixin.LoggingMixin

DagRun describes an instance of a Dag. It can be created by the scheduler (for regular runs) or by an external trigger

**static find** (\*args, \*\*kwargs)

Returns a set of dag runs for the given search criteria.

#### Parameters

- **dag\_id** (*int*, *list*) – the dag\_id to find dag runs for
- **run\_id** (*str*) – defines the the run id for this dag run
- **execution\_date** (*datetime.datetime*) – the execution date
- **state** (*airflow.utils.state.State*) – the state of the dag run
- **external\_trigger** (*bool*) – whether this dag run is externally triggered
- **no\_backfills** (*bool*) – return no backfills (True), return all (False). Defaults to False
- **session** (*sqlalchemy.orm.session.Session*) – database session

**get\_dag**()

Returns the Dag associated with this DagRun.

**Returns** DAG

**classmethod get\_latest\_runs** (\*\*kwargs)

Returns the latest DagRun for each DAG.

**get\_previous\_dagrun** (\*\*kwargs)

The previous DagRun, if there is one

**get\_previous\_scheduled\_dagrun** (\*\*kwargs)

The previous, SCHEDULED DagRun, if there is one



**static** `get_run(session, dag_id, execution_date)`

**Parameters**

- **dag\_id** (*unicode*) – DAG ID
- **execution\_date** (*datetime*) – execution date

**Returns** DagRun corresponding to the given dag\_id and execution date if one exists. None otherwise.

**Return type** *DagRun*

**get\_task\_instance** (*\*\*kwargs*)

Returns the task instance specified by task\_id for this dag run

**Parameters** **task\_id** – the task id

**get\_task\_instances** (*\*\*kwargs*)

Returns the task instances for this dag run

**refresh\_from\_db** (*\*\*kwargs*)

Reloads the current dagrun from the database :param session: database session

**update\_state** (*\*\*kwargs*)

Determines the overall state of the DagRun based on the state of its TaskInstances.

**Returns** State

**verify\_integrity** (*\*\*kwargs*)

Verifies the DagRun by checking for removed tasks or tasks that are not in the database yet. It will set state to removed or add the task if required.

**exception** `airflow.models.InvalidFernetToken`

Bases: `exceptions.Exception`

**class** `airflow.models.NullFernet`

Bases: `future.types.newobject.newobject`

A “Null” encryptor class that doesn’t encrypt or decrypt but that presents a similar interface to Fernet.

The purpose of this is to make the rest of the code not have to know the difference, and to only display the message once, not 20 times when *airflow initdb* is ran.

**class** `airflow.models.Pool` (*\*\*kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

**open\_slots** (*\*\*kwargs*)

Returns the number of slots open at the moment

**queued\_slots** (*\*\*kwargs*)

Returns the number of slots used at the moment

**used\_slots** (*\*\*kwargs*)

Returns the number of slots used at the moment

**class** `airflow.models.TaskInstance` (*task, execution\_date, state=None*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `airflow.utils.log.logging_mixin.LoggingMixin`

Task instances store the state of a task instance. This table is the authority and single source of truth around what tasks have run and the state they are in.

The SQLAlchemy model doesn’t have a SQLAlchemy foreign key to the task or dag model deliberately to have more control over transactions.

Database transactions on this table should insure double triggers and any confusion around what task instances are or aren't ready to run even while multiple schedulers may be firing task instances.

**are\_dependencies\_met** (*\*\*kwargs*)

Returns whether or not all the conditions are met for this task instance to be run given the context for the dependencies (e.g. a task instance being force run from the UI will ignore some dependencies).

#### Parameters

- **dep\_context** (*DepContext*) – The execution context that determines the dependencies that should be evaluated.
- **session** (*sqlalchemy.orm.session.Session*) – database session
- **verbose** (*bool*) – whether log details on failed dependencies on info or debug log level

**are\_dependents\_done** (*\*\*kwargs*)

Checks whether the dependents of this task instance have all succeeded. This is meant to be used by `wait_for_downstream`.

This is useful when you do not want to start processing the next schedule of a task until the dependents are done. For instance, if the task DROPS and recreates a table.

**clear\_xcom\_data** (*\*\*kwargs*)

Clears all XCom data from the database for the task instance

**command** (*mark\_success=False, ignore\_all\_deps=False, ignore\_depends\_on\_past=False, ignore\_task\_deps=False, ignore\_ti\_state=False, local=False, pickle\_id=None, raw=False, job\_id=None, pool=None, cfg\_path=None*)

Returns a command that can be executed anywhere where airflow is installed. This command is part of the message sent to executors by the orchestrator.

**command\_as\_list** (*mark\_success=False, ignore\_all\_deps=False, ignore\_task\_deps=False, ignore\_depends\_on\_past=False, ignore\_ti\_state=False, local=False, pickle\_id=None, raw=False, job\_id=None, pool=None, cfg\_path=None*)

Returns a command that can be executed anywhere where airflow is installed. This command is part of the message sent to executors by the orchestrator.

**current\_state** (*\*\*kwargs*)

Get the very latest state from the database, if a session is passed, we use and looking up the state becomes part of the session, otherwise a new session is used.

**error** (*\*\*kwargs*)

Forces the task instance's state to FAILED in the database.

**static generate\_command** (*dag\_id, task\_id, execution\_date, mark\_success=False, ignore\_all\_deps=False, ignore\_depends\_on\_past=False, ignore\_task\_deps=False, ignore\_ti\_state=False, local=False, pickle\_id=None, file\_path=None, raw=False, job\_id=None, pool=None, cfg\_path=None*)

Generates the shell command required to execute this task instance.

#### Parameters

- **dag\_id** (*unicode*) – DAG ID
- **task\_id** (*unicode*) – Task ID
- **execution\_date** (*datetime*) – Execution date for the task
- **mark\_success** (*bool*) – Whether to mark the task as successful
- **ignore\_all\_deps** (*bool*) – Ignore all ignorable dependencies. Overrides the other `ignore_*` parameters.

- **ignore\_depends\_on\_past** (*bool*) – Ignore depends\_on\_past parameter of DAGs (e.g. for Backfills)
- **ignore\_task\_deps** (*bool*) – Ignore task-specific dependencies such as depends\_on\_past and trigger rule
- **ignore\_ti\_state** (*bool*) – Ignore the task instance’s previous failure/success
- **local** (*bool*) – Whether to run the task locally
- **pickle\_id** (*unicode*) – If the DAG was serialized to the DB, the ID associated with the pickled DAG
- **file\_path** – path to the file containing the DAG definition
- **raw** – raw mode (needs more details)
- **job\_id** – job ID (needs more details)
- **pool** (*unicode*) – the Airflow pool that the task should run in
- **cfg\_path** (*basestring*) – the Path to the configuration file

**Returns** shell command that can be used to run the task instance

**get\_dagrun** (*\*\*kwargs*)

Returns the DagRun for this TaskInstance

**Parameters** **session** –

**Returns** DagRun

**init\_on\_load** ()

Initialize the attributes that aren’t stored in the DB.

**init\_run\_context** (*raw=False*)

Sets the log context.

**is\_eligible\_to\_retry** ()

Is task instance is eligible for retry

**is\_premature**

Returns whether a task is in UP\_FOR\_RETRY state and its retry interval has elapsed.

**key**

Returns a tuple that identifies the task instance uniquely

**next\_retry\_datetime** ()

Get datetime of the next retry if the task instance fails. For exponential backoff, retry\_delay is used as base and will be converted to seconds.

**pool\_full** (*\*\*kwargs*)

Returns a boolean as to whether the slot pool has room for this task to run

**previous\_ti**

The task instance for the task that ran before this task instance

**ready\_for\_retry** ()

Checks on whether the task instance is in the right state and timeframe to be retried.

**refresh\_from\_db** (*\*\*kwargs*)

Refreshes the task instance from the database based on the primary key

**Parameters** **lock\_for\_update** – if True, indicates that the database should lock the TaskInstance (issuing a FOR UPDATE clause) until the session is committed.

**try\_number**

Return the try number that this task number will be when it is actually run.

If the TI is currently running, this will match the column in the database, in all other cases this will be incremented

**xcom\_pull** (*task\_ids=None, dag\_id=None, key=u'return\_value', include\_prior\_dates=False*)

Pull XComs that optionally meet certain criteria.

The default value for *key* limits the search to XComs that were returned by other tasks (as opposed to those that were pushed manually). To remove this filter, pass *key=None* (or any desired value).

If a single *task\_id* string is provided, the result is the value of the most recent matching XCom from that *task\_id*. If multiple *task\_ids* are provided, a tuple of matching values is returned. *None* is returned whenever no matches are found.

**Parameters**

- **key** (*str*) – A key for the XCom. If provided, only XComs with matching keys will be returned. The default key is 'return\_value', also available as a constant `XCOM_RETURN_KEY`. This key is automatically given to XComs returned by tasks (as opposed to being pushed manually). To remove the filter, pass *key=None*.
- **task\_ids** (*str or iterable of strings (representing task\_ids)*) – Only XComs from tasks with matching ids will be pulled. Can pass *None* to remove the filter.
- **dag\_id** (*str*) – If provided, only pulls XComs from this DAG. If *None* (default), the DAG of the calling task is used.
- **include\_prior\_dates** (*bool*) – If *False*, only XComs from the current execution\_date are returned. If *True*, XComs from previous dates are returned as well.

**xcom\_push** (*key, value, execution\_date=None*)

Make an XCom available for tasks to pull.

**Parameters**

- **key** (*str*) – A key for the XCom
- **value** (*any pickleable object*) – A value for the XCom. The value is pickled and stored in the database.
- **execution\_date** (*datetime*) – if provided, the XCom will not be visible until this date. This can be used, for example, to send a message to a task on a future date without it being immediately visible.

**class** airflow.models.Variable (\*\*kwargs)

Bases: sqlalchemy.ext.declarative.api.Base, airflow.utils.log.logging\_mixin.LoggingMixin

**classmethod** setdefault (*key, default, deserialize\_json=False*)

Like a Python builtin dict object, setdefault returns the current value for a key, and if it isn't there, stores the default value and returns it.

**Parameters**

- **key** (*str*) – Dict key for this Variable
- **default** (*Mixed*) – Default value to set and return if the variable isn't already in the DB
- **deserialize\_json** – Store this as a JSON encoded value in the DB and un-encode it when retrieving a value

**Returns** Mixed

`airflow.models.clear_task_instances(tis, session, activate_dag_runs=True, dag=None)`

Clears a set of task instances, but makes sure the running ones get killed.

**Parameters**

- **tis** – a list of task instances
- **session** – current session
- **activate\_dag\_runs** – flag to check for active dag run
- **dag** – DAG object

`airflow.models.get_fernet()`

Deferred load of Fernet key.

This function could fail either because Cryptography is not installed or because the Fernet key is invalid.

**Returns** Fernet object

**Raises** `airflow.exceptions.AirflowException` if there's a problem trying to load Fernet

`airflow.models.get_last_dagrun(dag_id, session, include_externally_triggered=False)`

Returns the last dag run for a dag, None if there was none. Last dag run can be any type of run eg. scheduled or backfilled. Overridden DagRuns are ignored.

### 3.21.4 Hooks

Hooks are interfaces to external platforms and databases, implementing a common interface when possible and acting as building blocks for operators.

**class** `airflow.hooks.base_hook.BaseHook(source)`

Bases: `airflow.utils.log.logging_mixin.LoggingMixin`

Abstract base class for hooks, hooks are meant as an interface to interact with external systems. `MySQLHook`, `HiveHook`, `PigHook` return object that can handle the connection and interaction to specific instances of these systems, and expose consistent methods to interact with them.

**class** `airflow.hooks.dbapi_hook.DbApiHook(*args, **kwargs)`

Bases: `airflow.hooks.base_hook.BaseHook`

Abstract base class for sql hooks.

**bulk\_dump** (*table*, *tmp\_file*)

Dumps a database table into a tab-delimited file

**Parameters**

- **table** (*str*) – The name of the source table
- **tmp\_file** (*str*) – The path of the target file

**bulk\_load** (*table*, *tmp\_file*)

Loads a tab-delimited file into a database table

**Parameters**

- **table** (*str*) – The name of the target table
- **tmp\_file** (*str*) – The path of the file to load into the table

**get\_autocommit** (*conn*)

Get autocommit setting for the provided connection. Return True if conn.autocommit is set to True. Return False if conn.autocommit is not set or set to False or conn does not support autocommit.

**Parameters** *conn* (*connection object*.) – Connection to get autocommit setting from.

**Returns** connection autocommit setting.

**Return type** `bool`

**get\_conn** ()

Returns a connection object

**get\_cursor** ()

Returns a cursor

**get\_first** (*sql*, *parameters=None*)

Executes the sql and returns the first resulting row.

**Parameters**

- **sql** (*str* or *list*) – the sql statement to be executed (str) or a list of sql statements to execute
- **parameters** (*mapping* or *iterable*) – The parameters to render the SQL query with.

**get\_pandas\_df** (*sql*, *parameters=None*)

Executes the sql and returns a pandas dataframe

**Parameters**

- **sql** (*str* or *list*) – the sql statement to be executed (str) or a list of sql statements to execute
- **parameters** (*mapping* or *iterable*) – The parameters to render the SQL query with.

**get\_records** (*sql*, *parameters=None*)

Executes the sql and returns a set of records.

**Parameters**

- **sql** (*str* or *list*) – the sql statement to be executed (str) or a list of sql statements to execute
- **parameters** (*mapping* or *iterable*) – The parameters to render the SQL query with.

**insert\_rows** (*table*, *rows*, *target\_fields=None*, *commit\_every=1000*, *replace=False*)

A generic way to insert a set of tuples into a table, a new transaction is created every commit\_every rows

**Parameters**

- **table** (*str*) – Name of the target table
- **rows** (*iterable of tuples*) – The rows to insert into the table
- **target\_fields** (*iterable of strings*) – The names of the columns to fill in the table
- **commit\_every** (*int*) – The maximum number of rows to insert in one transaction. Set to 0 to insert all rows in one transaction.
- **replace** (*bool*) – Whether to replace instead of insert



- **commit\_every** (*int*) – The maximum number of rows to insert in one transaction. Set to 0 to insert all rows in one transaction.
- **replace** (*bool*) – Whether to replace instead of insert

**set\_autocommit** (*conn, autocommit*)  
Sets the autocommit flag on the connection

**class** airflow.hooks.druid\_hook.**DruidHook** (*druid\_ingest\_conn\_id='druid\_ingest\_default',  
timeout=1, max\_ingestion\_time=None*)  
Bases: *airflow.hooks.base\_hook.BaseHook*

Connection to Druid overlord for ingestion

#### Parameters

- **druid\_ingest\_conn\_id** (*str*) – The connection id to the Druid overlord machine which accepts index jobs
- **timeout** (*int*) – The interval between polling the Druid job for the status of the ingestion job. Must be greater than or equal to 1
- **max\_ingestion\_time** (*int*) – The maximum ingestion time before assuming the job failed

**class** airflow.hooks.hive\_hooks.**HiveCliHook** (*hive\_cli\_conn\_id=u'hive\_cli\_default',  
run\_as=None, mapred\_queue=None,  
mapred\_queue\_priority=None,  
mapred\_job\_name=None*)  
Bases: *airflow.hooks.base\_hook.BaseHook*

Simple wrapper around the hive CLI.

It also supports the beeline a lighter CLI that runs JDBC and is replacing the heavier traditional CLI. To enable beeline, set the use\_beeline param in the extra field of your connection as in { "use\_beeline": true }

Note that you can also set default hive CLI parameters using the hive\_cli\_params to be used in your connection as in {"hive\_cli\_params": "-hiveconf mapred.job.tracker=some.jobtracker:444"} Parameters passed here can be overridden by run\_cli's hive\_conf param

The extra connection parameter auth gets passed as in the jdbc connection string as is.

#### Parameters

- **mapred\_queue** (*str*) – queue used by the Hadoop Scheduler (Capacity or Fair)
- **mapred\_queue\_priority** (*str*) – priority within the job queue. Possible settings include: VERY\_HIGH, HIGH, NORMAL, LOW, VERY\_LOW
- **mapred\_job\_name** (*str*) – This name will appear in the jobtracker. This can make monitoring easier.

**load\_df** (*df, table, field\_dict=None, delimiter=u', ', encoding=u'utf8', pandas\_kwargs=None, \*\*kwargs*)

Loads a pandas DataFrame into hive.

Hive data types will be inferred if not passed but column names will not be sanitized.

#### Parameters

- **df** (*pandas.DataFrame*) – DataFrame to load into a Hive table
- **table** (*str*) – target Hive table, use dot notation to target a specific database



- **field\_dict** (*collections.OrderedDict*) – mapping from column name to hive data type. Note that it must be *OrderedDict* so as to keep columns' order.
- **delimiter** (*str*) – field delimiter in the file
- **encoding** (*str*) – str encoding to use when writing *DataFrame* to file
- **pandas\_kwargs** (*dict*) – passed to *DataFrame.to\_csv*
- **kwargs** – passed to *self.load\_file*

**load\_file** (*filepath, table, delimiter=u', ', field\_dict=None, create=True, overwrite=True, partition=None, recreate=False, tblproperties=None*)  
Loads a local file into Hive

Note that the table generated in Hive uses `STORED AS textfile` which isn't the most efficient serialization format. If a large amount of data is loaded and/or if the tables gets queried considerably, you may want to use this operator only to stage the data into a temporary table before loading it into its final destination using a *HiveOperator*.

#### Parameters

- **filepath** (*str*) – local filepath of the file to load
- **table** (*str*) – target Hive table, use dot notation to target a specific database
- **delimiter** (*str*) – field delimiter in the file
- **field\_dict** (*collections.OrderedDict*) – A dictionary of the fields name in the file as keys and their Hive types as values. Note that it must be *OrderedDict* so as to keep columns' order.
- **create** (*bool*) – whether to create the table if it doesn't exist
- **overwrite** (*bool*) – whether to overwrite the data in table or partition
- **partition** (*dict*) – target partition as a dict of partition columns and values
- **recreate** (*bool*) – whether to drop and recreate the table at every execution
- **tblproperties** (*dict*) – TBLPROPERTIES of the hive table being created

**run\_cli** (*hql, schema=None, verbose=True, hive\_conf=None*)

Run an hql statement using the hive cli. If *hive\_conf* is specified it should be a dict and the entries will be set as key/value pairs in *HiveConf*

**Parameters** *hive\_conf* (*dict*) – if specified these key value pairs will be passed to hive as `-hiveconf "key"="value"`. Note that they will be passed after the *hive\_cli\_params* and thus will override whatever values are specified in the database.

```
>>> hh = HiveCliHook()
>>> result = hh.run_cli("USE airflow;")
>>> ("OK" in result)
True
```

**test\_hql** (*hql*)

Test an hql statement using the hive cli and EXPLAIN

**class** *airflow.hooks.hive\_hooks.HiveMetastoreHook* (*metastore\_conn\_id=u'metastore\_default'*)  
Bases: *airflow.hooks.base\_hook.BaseHook*

Wrapper to interact with the Hive Metastore

**check\_for\_named\_partition** (*schema, table, partition\_name*)

Checks whether a partition with a given name exists

**Parameters**

- **schema** (*str*) – Name of hive schema (database) @table belongs to
- **table** – Name of hive table @partition belongs to

**Partition** Name of the partitions to check for (eg  $a=b/c=d$ )

**Return type** `bool`

```
>>> hh = HiveMetastoreHook()
>>> t = 'static_babynames_partitioned'
>>> hh.check_for_named_partition('airflow', t, "ds=2015-01-01")
True
>>> hh.check_for_named_partition('airflow', t, "ds=xxx")
False
```

**check\_for\_partition** (*schema, table, partition*)

Checks whether a partition exists

**Parameters**

- **schema** (*str*) – Name of hive schema (database) @table belongs to
- **table** – Name of hive table @partition belongs to

**Partition** Expression that matches the partitions to check for (eg  $a = 'b' \text{ AND } c = 'd'$ )

**Return type** `bool`

```
>>> hh = HiveMetastoreHook()
>>> t = 'static_babynames_partitioned'
>>> hh.check_for_partition('airflow', t, "ds='2015-01-01'")
True
```

**get\_databases** (*pattern=u'\*'*)

Get a metastore table object

**get\_metastore\_client** ()

Returns a Hive thrift client.

**get\_partitions** (*schema, table\_name, filter=None*)

Returns a list of all partitions in a table. Works only for tables with less than 32767 (java short max val). For subpartitioned table, the number might easily exceed this.

```
>>> hh = HiveMetastoreHook()
>>> t = 'static_babynames_partitioned'
>>> parts = hh.get_partitions(schema='airflow', table_name=t)
>>> len(parts)
1
>>> parts
[{'ds': '2015-01-01'}]
```

**get\_table** (*table\_name, db=u'default'*)

Get a metastore table object

```
>>> hh = HiveMetastoreHook()
>>> t = hh.get_table(db='airflow', table_name='static_babynames')
>>> t.tableName
'static_babynames'
>>> [col.name for col in t.sd.cols]
['state', 'year', 'name', 'gender', 'num']
```

**get\_tables** (*db*, *pattern=u'\*'*)

Get a metastore table object

**max\_partition** (*schema*, *table\_name*, *field=None*, *filter\_map=None*)

Returns the maximum value for all partitions with given field in a table. If only one partition key exist in the table, the key will be used as field. filter\_map should be a partition\_key:partition\_value map and will be used to filter out partitions.

#### Parameters

- **schema** (*str*) – schema name.
- **table\_name** (*str*) – table name.
- **field** (*str*) – partition key to get max partition from.
- **filter\_map** (*map*) – partition\_key:partition\_value map used for partition filtering.

```
>>> hh = HiveMetastoreHook()
>>> filter_map = {'ds': '2015-01-01', 'ds': '2014-01-01'}
>>> t = 'static_babynames_partitioned'
>>> hh.max_partition(schema='airflow', ... table_name=t, field='ds',
↪ filter_map=filter_map)
'2015-01-01'
```

**table\_exists** (*table\_name*, *db=u'default'*)

Check if table exists

```
>>> hh = HiveMetastoreHook()
>>> hh.table_exists(db='airflow', table_name='static_babynames')
True
>>> hh.table_exists(db='airflow', table_name='does_not_exist')
False
```

**class** airflow.hooks.hive\_hooks.**HiveServer2Hook** (*hiveserver2\_conn\_id=u'hiveserver2\_default'*)

Bases: *airflow.hooks.base\_hook.BaseHook*

Wrapper around the pyhive library

Note that the default authMechanism is PLAIN, to override it you can specify it in the extra of your connection in the UI as in

**get\_conn** (*schema=None*)

Returns a Hive connection object.

**get\_pandas\_df** (*hql*, *schema=u'default'*)

Get a pandas dataframe from a Hive query

#### Parameters

- **hql** (*str or list*) – hql to be executed.
- **schema** (*str*) – target schema, default to 'default'.

**Returns** result of hql execution

**Return type** DataFrame

```
>>> hh = HiveServer2Hook()
>>> sql = "SELECT * FROM airflow.static_babynames LIMIT 100"
>>> df = hh.get_pandas_df(sql)
>>> len(df.index)
100
```

**Returns** `pandas.DataFrame`

**get\_records** (*hql*, *schema*=*u'default'*, *hive\_conf*=*None*)

Get a set of records from a Hive query.

#### Parameters

- **hql** (*str* or *list*) – hql to be executed.
- **schema** (*str*) – target schema, default to 'default'.
- **hive\_conf** (*dict*) – hive\_conf to execute alone with the hql.

**Returns** result of hive execution

**Return type** `list`

```
>>> hh = HiveServer2Hook()
>>> sql = "SELECT * FROM airflow.static_babynames LIMIT 100"
>>> len(hh.get_records(sql))
100
```

**get\_results** (*hql*, *schema*=*u'default'*, *fetch\_size*=*None*, *hive\_conf*=*None*)

Get results of the provided hql in target schema.

#### Parameters

- **hql** (*str* or *list*) – hql to be executed.
- **schema** (*str*) – target schema, default to 'default'.
- **fetch\_size** (*int*) – max size of result to fetch.
- **hive\_conf** (*dict*) – hive\_conf to execute alone with the hql.

**Returns** results of hql execution, dict with data (list of results) and header

**Return type** `dict`

**to\_csv** (*hql*, *csv\_filepath*, *schema*=*u'default'*, *delimiter*=*u','*, *lineterminator*=*u'\n'*, *output\_header*=*True*, *fetch\_size*=*1000*, *hive\_conf*=*None*)

Execute hql in target schema and write results to a csv file.

#### Parameters

- **hql** (*str* or *list*) – hql to be executed.
- **csv\_filepath** (*str*) – filepath of csv to write results into.
- **schema** (*str*) – target schema, default to 'default'.
- **delimiter** (*str*) – delimiter of the csv file, default to ','.
- **lineterminator** (*str*) – lineterminator of the csv file.
- **output\_header** (*bool*) – header of the csv file, default to True.
- **fetch\_size** (*int*) – number of result rows to write into the csv file, default to 1000.
- **hive\_conf** (*dict*) – hive\_conf to execute alone with the hql.

**class** `airflow.hooks.http_hook.HttpHook` (*method*=*'POST'*, *http\_conn\_id*=*'http\_default'*)

Bases: `airflow.hooks.base_hook.BaseHook`

Interact with HTTP servers.

#### Parameters

- **http\_conn\_id** (*str*) – connection that has the base API url i.e <https://www.google.com/> and optional authentication credentials. Default headers can also be specified in the Extra field in json format.
- **method** (*str*) – the API method to be called

**check\_response** (*response*)

Checks the status code and raise an AirflowException exception on non 2XX or 3XX status codes

**Parameters** **response** (*requests.response*) – A requests response object

**get\_conn** (*headers=None*)

Returns http session for use with requests

**Parameters** **headers** (*dict*) – additional headers to be passed through as a dictionary

**run** (*endpoint, data=None, headers=None, extra\_options=None*)

Performs the request

**Parameters**

- **endpoint** (*str*) – the endpoint to be called i.e. resource/v1/query?
- **data** (*dict*) – payload to be uploaded or request parameters
- **headers** (*dict*) – additional headers to be passed through as a dictionary
- **extra\_options** (*dict*) – additional options to be used when executing the request i.e. {'check\_response': False} to avoid checking raising exceptions on non 2XX or 3XX status codes

**run\_and\_check** (*session, prepped\_request, extra\_options*)

Grabs extra options like timeout and actually runs the request, checking for the result

**Parameters**

- **session** (*requests.Session*) – the session to be used to execute the request
- **prepped\_request** (*session.prepare\_request*) – the prepared request generated in run()
- **extra\_options** (*dict*) – additional options to be used when executing the request i.e. {'check\_response': False} to avoid checking raising exceptions on non 2XX or 3XX status codes

**run\_with\_advanced\_retry** (*\_retry\_args, \*args, \*\*kwargs*)

Runs Hook.run() with a Tenacity decorator attached to it. This is useful for connectors which might be disturbed by intermittent issues and should not instantly fail.

**Parameters** **\_retry\_args** (*dict*) – Arguments which define the retry behaviour. See Tenacity documentation at <https://github.com/jd/tenacity>

:Example:

```
hook = HttpHook(http_conn_id='my_conn', method='GET')
retry_args = dict(
    wait=tenacity.wait_exponential(),
    stop=tenacity.stop_after_attempt(10),
    retry=requests.exceptions.ConnectionError
)
hook.run_with_advanced_retry(
    endpoint='v1/test',
    _retry_args=retry_args
)
```

```
class airflow.hooks.jdbc_hook.JdbcHook(*args, **kwargs)
```

Bases: [airflow.hooks.dbapi\\_hook.DbApiHook](#)

General hook for jdbc db access.

JDBC URL, username and password will be taken from the predefined connection. Note that the whole JDBC URL must be specified in the “host” field in the DB. Raises an airflow error if the given connection id doesn’t exist.

```
get_conn()
```

Returns a connection object

```
set_autocommit(conn, autocommit)
```

Enable or disable autocommit for the given connection.

**Parameters** **conn** – The connection

**Returns**

```
class airflow.hooks.mssql_hook.MsSqlHook(*args, **kwargs)
```

Bases: [airflow.hooks.dbapi\\_hook.DbApiHook](#)

Interact with Microsoft SQL Server.

```
get_autocommit(conn)
```

Get autocommit setting for the provided connection. Return True if conn.autocommit is set to True. Return False if conn.autocommit is not set or set to False or conn does not support autocommit.

**Parameters** **conn** (*connection object*.) – Connection to get autocommit setting from.

**Returns** connection autocommit setting.

**Return type** `bool`

```
get_conn()
```

Returns a mssql connection object

```
set_autocommit(conn, autocommit)
```

Sets the autocommit flag on the connection

```
class airflow.hooks.mysql_hook.MySqlHook(*args, **kwargs)
```

Bases: [airflow.hooks.dbapi\\_hook.DbApiHook](#)

Interact with MySQL.

You can specify charset in the extra field of your connection as {"charset": "utf8"}. Also you can choose cursor as {"cursor": "SSCursor"}. Refer to the MySQLdb.cursors for more details.

```
bulk_dump(table, tmp_file)
```

Dumps a database table into a tab-delimited file

```
bulk_load(table, tmp_file)
```

Loads a tab-delimited file into a database table

```
get_autocommit(conn)
```

MySql connection gets autocommit in a different way.

**Parameters** **conn** (*connection object*.) – connection to get autocommit setting from.

**Returns** connection autocommit setting

**Return type** `bool`

```
get_conn()
```

Returns a mysql connection object

**set\_autocommit** (*conn, autocommit*)

MySQL connection sets autocommit in a different way.

**class** `airflow.hooks.oracle_hook.OracleHook` (*\*args, \*\*kwargs*)

Bases: `airflow.hooks.dbapi_hook.DbApiHook`

Interact with Oracle SQL.

**bulk\_insert\_rows** (*table, rows, target\_fields=None, commit\_every=5000*)

A performant bulk insert for cx\_Oracle that uses prepared statements via `executemany()`. For best performance, pass in *rows* as an iterator.

**get\_conn** ()

Returns a oracle connection object Optional parameters for using a custom DSN connection (instead of using a server alias from tnsnames.ora) The dsn (data source name) is the TNS entry (from the Oracle names server or tnsnames.ora file) or is a string like the one returned from `makedsn()`.

#### Parameters

- **dsn** – the host address for the Oracle server
- **service\_name** – the db\_unique\_name of the database that you are connecting to (CONNECT\_DATA part of TNS)

You can set these parameters in the extra fields of your connection as in { "dsn": "some.host.address" , "service\_name": "some.service.name" }

**insert\_rows** (*table, rows, target\_fields=None, commit\_every=1000*)

A generic way to insert a set of tuples into a table, the whole set of inserts is treated as one transaction Changes from standard DbApiHook implementation:

- Oracle SQL queries in cx\_Oracle can not be terminated with a semicolon (;)
- Replace NaN values with NULL using `numpy.nan_to_num` (not using `is_nan()` because of input types error for strings)
- Coerce datetime cells to Oracle DATETIME format during insert

**class** `airflow.hooks.pig_hook.PigCliHook` (*pig\_cli\_conn\_id='pig\_cli\_default'*)

Bases: `airflow.hooks.base_hook.BaseHook`

Simple wrapper around the pig CLI.

Note that you can also set default pig CLI properties using the `pig_properties` to be used in your connection as in { "pig\_properties": "-Dpig.tmpfilecompression=true" }

**run\_cli** (*pig, verbose=True*)

Run an pig script using the pig cli

```
>>> ph = PigCliHook()
>>> result = ph.run_cli("ls /;")
>>> ("hdfs://" in result)
True
```

**class** `airflow.hooks.postgres_hook.PostgresHook` (*\*args, \*\*kwargs*)

Bases: `airflow.hooks.dbapi_hook.DbApiHook`

Interact with Postgres. You can specify ssl parameters in the extra field of your connection as { "sslmode": "require", "sslcert": "/path/to/cert.pem", etc}.

Note: For Redshift, use `keepalives_idle` in the extra connection parameters and set it to less than 300 seconds.

**bulk\_dump** (*table, tmp\_file*)

Dumps a database table into a tab-delimited file

**bulk\_load** (*table, tmp\_file*)

Loads a tab-delimited file into a database table

**copy\_expert** (*sql, filename, open=<built-in function open>*)

Executes SQL using psycopg2 copy\_expert method. Necessary to execute COPY command without access to a superuser.

Note: if this method is called with a “COPY FROM” statement and the specified input file does not exist, it creates an empty file and no data is loaded, but the operation succeeds. So if users want to be aware when the input file does not exist, they have to check its existence by themselves.

**get\_conn** ()

Returns a connection object

**class** airflow.hooks.presto\_hook.**PrestoHook** (\*args, \*\*kwargs)

Bases: *airflow.hooks.dbapi\_hook.DbApiHook*

Interact with Presto through PyHive!

```
>>> ph = PrestoHook()
>>> sql = "SELECT count(1) AS num FROM airflow.static_babynames"
>>> ph.get_records(sql)
[[340698]]
```

**get\_conn** ()

Returns a connection object

**get\_first** (*hql, parameters=None*)

Returns only the first row, regardless of how many rows the query returns.

**get\_pandas\_df** (*hql, parameters=None*)

Get a pandas dataframe from a sql query.

**get\_records** (*hql, parameters=None*)

Get a set of records from Presto

**insert\_rows** (*table, rows, target\_fields=None*)

A generic way to insert a set of tuples into a table.

#### Parameters

- **table** (*str*) – Name of the target table
- **rows** (*iterable of tuples*) – The rows to insert into the table
- **target\_fields** (*iterable of strings*) – The names of the columns to fill in the table

**run** (*hql, parameters=None*)

Execute the statement against Presto. Can be used to create views.

**class** airflow.hooks.S3\_hook.**S3Hook** (*aws\_conn\_id='aws\_default', verify=None*)

Bases: *airflow.contrib.hooks.aws\_hook.AwsHook*

Interact with AWS S3, using the boto3 library.

**check\_for\_bucket** (*bucket\_name*)

Check if bucket\_name exists.

**Parameters** **bucket\_name** (*str*) – the name of the bucket

**check\_for\_key** (*key, bucket\_name=None*)

Checks if a key exists in a bucket



**Parameters**

- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which the file is stored

**check\_for\_prefix** (*bucket\_name, prefix, delimiter*)

Checks that a prefix exists in a bucket

**Parameters**

- **bucket\_name** (*str*) – the name of the bucket
- **prefix** (*str*) – a key prefix
- **delimiter** (*str*) – the delimiter marks key hierarchy.

**check\_for\_wildcard\_key** (*wildcard\_key, bucket\_name=None, delimiter=""*)

Checks that a key matching a wildcard expression exists in a bucket

**Parameters**

- **wildcard\_key** (*str*) – the path to the key
- **bucket\_name** (*str*) – the name of the bucket
- **delimiter** (*str*) – the delimiter marks key hierarchy

**copy\_object** (*source\_bucket\_key, dest\_bucket\_key, source\_bucket\_name=None, dest\_bucket\_name=None, source\_version\_id=None*)

Creates a copy of an object that is already stored in S3.

Note: the S3 connection used here needs to have access to both source and destination bucket/key.

**Parameters**

- **source\_bucket\_key** (*str*) – The key of the source object.  
It can be either full s3:// style url or relative path from root level.  
When it's specified as a full s3:// url, please omit source\_bucket\_name.
- **dest\_bucket\_key** (*str*) – The key of the object to copy to.  
The convention to specify *dest\_bucket\_key* is the same as *source\_bucket\_key*.
- **source\_bucket\_name** (*str*) – Name of the S3 bucket where the source object is in.  
It should be omitted when *source\_bucket\_key* is provided as a full s3:// url.
- **dest\_bucket\_name** (*str*) – Name of the S3 bucket to where the object is copied.  
It should be omitted when *dest\_bucket\_key* is provided as a full s3:// url.
- **source\_version\_id** (*str*) – Version ID of the source object (OPTIONAL)

**create\_bucket** (*bucket\_name, region\_name=None*)

Creates an Amazon S3 bucket.

**Parameters**

- **bucket\_name** (*str*) – The name of the bucket
- **region\_name** (*str*) – The name of the aws region in which to create the bucket.

**delete\_objects** (*bucket, keys*)

**Parameters**

- **bucket** (*str*) – Name of the bucket in which you are going to delete object(s)

- **keys** (*str* or *list*) – The key(s) to delete from S3 bucket.

When *keys* is a string, it's supposed to be the key name of the single object to delete.

When *keys* is a list, it's supposed to be the list of the keys to delete.

**get\_bucket** (*bucket\_name*)

Returns a boto3.S3.Bucket object

**Parameters** **bucket\_name** (*str*) – the name of the bucket

**get\_key** (*key*, *bucket\_name=None*)

Returns a boto3.s3.Object

**Parameters**

- **key** (*str*) – the path to the key
- **bucket\_name** (*str*) – the name of the bucket

**get\_wildcard\_key** (*wildcard\_key*, *bucket\_name=None*, *delimiter=""*)

Returns a boto3.s3.Object object matching the wildcard expression

**Parameters**

- **wildcard\_key** (*str*) – the path to the key
- **bucket\_name** (*str*) – the name of the bucket
- **delimiter** (*str*) – the delimiter marks key hierarchy

**list\_keys** (*bucket\_name*, *prefix=""*, *delimiter=""*, *page\_size=None*, *max\_items=None*)

Lists keys in a bucket under prefix and not containing delimiter

**Parameters**

- **bucket\_name** (*str*) – the name of the bucket
- **prefix** (*str*) – a key prefix
- **delimiter** (*str*) – the delimiter marks key hierarchy.
- **page\_size** (*int*) – pagination size
- **max\_items** (*int*) – maximum items to return

**list\_prefixes** (*bucket\_name*, *prefix=""*, *delimiter=""*, *page\_size=None*, *max\_items=None*)

Lists prefixes in a bucket under prefix

**Parameters**

- **bucket\_name** (*str*) – the name of the bucket
- **prefix** (*str*) – a key prefix
- **delimiter** (*str*) – the delimiter marks key hierarchy.
- **page\_size** (*int*) – pagination size
- **max\_items** (*int*) – maximum items to return

**load\_bytes** (*bytes\_data*, *key*, *bucket\_name=None*, *replace=False*, *encrypt=False*)

Loads bytes to S3

This is provided as a convenience to drop a string in S3. It uses the boto infrastructure to ship a file to s3.

**Parameters**

- **bytes\_data** (*bytes*) – bytes to set as content for the key.

- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which to store the file
- **replace** (*bool*) – A flag to decide whether or not to overwrite the key if it already exists
- **encrypt** (*bool*) – If True, the file will be encrypted on the server-side by S3 and will be stored in an encrypted form while at rest in S3.

**load\_file** (*filename, key, bucket\_name=None, replace=False, encrypt=False*)

Loads a local file to S3

#### Parameters

- **filename** (*str*) – name of the file to load.
- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which to store the file
- **replace** (*bool*) – A flag to decide whether or not to overwrite the key if it already exists. If replace is False and the key exists, an error will be raised.
- **encrypt** (*bool*) – If True, the file will be encrypted on the server-side by S3 and will be stored in an encrypted form while at rest in S3.

**load\_file\_obj** (*file\_obj, key, bucket\_name=None, replace=False, encrypt=False*)

Loads a file object to S3

#### Parameters

- **file\_obj** (*file-like object*) – The file-like object to set as the content for the S3 key.
- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which to store the file
- **replace** (*bool*) – A flag that indicates whether to overwrite the key if it already exists.
- **encrypt** (*bool*) – If True, S3 encrypts the file on the server, and the file is stored in encrypted form at rest in S3.

**load\_string** (*string\_data, key, bucket\_name=None, replace=False, encrypt=False, encoding='utf-8'*)

Loads a string to S3

This is provided as a convenience to drop a string in S3. It uses the boto infrastructure to ship a file to s3.

#### Parameters

- **string\_data** (*str*) – str to set as content for the key.
- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which to store the file
- **replace** (*bool*) – A flag to decide whether or not to overwrite the key if it already exists
- **encrypt** (*bool*) – If True, the file will be encrypted on the server-side by S3 and will be stored in an encrypted form while at rest in S3.

**read\_key** (*key, bucket\_name=None*)

Reads a key from S3

#### Parameters

- **key** (*str*) – S3 key that will point to the file

- **bucket\_name** (*str*) – Name of the bucket in which the file is stored

**select\_key** (*key*, *bucket\_name=None*, *expression='SELECT \* FROM S3Object'*, *expression\_type='SQL'*, *input\_serialization=None*, *output\_serialization=None*)

Reads a key with S3 Select.

#### Parameters

- **key** (*str*) – S3 key that will point to the file
- **bucket\_name** (*str*) – Name of the bucket in which the file is stored
- **expression** (*str*) – S3 Select expression
- **expression\_type** (*str*) – S3 Select expression type
- **input\_serialization** (*dict*) – S3 Select input data serialization format
- **output\_serialization** (*dict*) – S3 Select output data serialization format

**Returns** retrieved subset of original data by S3 Select

**Return type** *str*

**See also:**

For more details about S3 Select parameters: [http://boto3.readthedocs.io/en/latest/reference/services/s3.html#S3.Client.select\\_object\\_content](http://boto3.readthedocs.io/en/latest/reference/services/s3.html#S3.Client.select_object_content)

**class** airflow.hooks.samba\_hook.**SambaHook** (*samba\_conn\_id*)

Bases: *airflow.hooks.base\_hook.BaseHook*

Allows for interaction with an samba server.

**class** airflow.hooks.slack\_hook.**SlackHook** (*token=None*, *slack\_conn\_id=None*)

Bases: *airflow.hooks.base\_hook.BaseHook*

Interact with Slack, using slackclient library.

**class** airflow.hooks.sqlite\_hook.**SqliteHook** (*\*args*, *\*\*kwargs*)

Bases: *airflow.hooks.dbapi\_hook.DbApiHook*

Interact with SQLite.

**get\_conn** ()

Returns a sqlite connection object

**class** airflow.hooks.webhdfs\_hook.**WebHDFSHook** (*webhdfs\_conn\_id='webhdfs\_default'*, *proxy\_user=None*)

Bases: *airflow.hooks.base\_hook.BaseHook*

Interact with HDFS. This class is a wrapper around the hdfscli library.

**check\_for\_path** (*hdfs\_path*)

Check for the existence of a path in HDFS by querying FileStatus.

**get\_conn** ()

Returns a hdfscli InsecureClient object.

**load\_file** (*source*, *destination*, *overwrite=True*, *parallelism=1*, *\*\*kwargs*)

Uploads a file to HDFS

#### Parameters

- **source** (*str*) – Local path to file or folder. If a folder, all the files inside of it will be uploaded (note that this implies that folders empty of files will not be created remotely).

- **destination** (*str*) – PTarget HDFS path. If it already exists and is a directory, files will be uploaded inside.
- **overwrite** (*bool*) – Overwrite any existing file or directory.
- **parallelism** (*int*) – Number of threads to use for parallelization. A value of 0 (or negative) uses as many threads as there are files.
- **\*\*kwargs** – Keyword arguments forwarded to `upload()`.

**class** `airflow.hooks.zendesk_hook.ZendeskHook` (*zendesk\_conn\_id*)  
 Bases: `airflow.hooks.base_hook.BaseHook`

A hook to talk to Zendesk

**call** (*path*, *query=None*, *get\_all\_pages=True*, *side\_loading=False*)  
 Call Zendesk API and return results

#### Parameters

- **path** – The Zendesk API to call
- **query** – Query parameters
- **get\_all\_pages** – Accumulate results over all pages before returning. Due to strict rate limiting, this can often timeout. Waits for recommended period between tries after a timeout.
- **side\_loading** – Retrieve related records as part of a single request. In order to enable side-loading, add an 'include' query parameter containing a comma-separated list of resources to load. For more information on side-loading see [https://developer.zendesk.com/rest\\_api/docs/core/side\\_loading](https://developer.zendesk.com/rest_api/docs/core/side_loading)

### 3.21.4.1 Community contributed hooks

**class** `airflow.contrib.hooks.aws_athena_hook.AWSAthenaHook` (*aws\_conn\_id='aws\_default'*,  
*sleep\_time=30*, *\*args*,  
*\*\*kwargs*)

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with AWS Athena to run, poll queries and return query results

#### Parameters

- **aws\_conn\_id** (*str*) – aws connection to use.
- **sleep\_time** (*int*) – Time to wait between two consecutive call to check query status on athena

**check\_query\_status** (*query\_execution\_id*)  
 Fetch the status of submitted athena query. Returns None or one of valid query states.

**Parameters** *query\_execution\_id* (*str*) – Id of submitted athena query

**Returns** *str*

**get\_conn** ()  
 check if aws conn exists already or create one and return it

**Returns** boto3 session

**get\_query\_results** (*query\_execution\_id*)  
 Fetch submitted athena query results. returns none if query is in intermediate state or failed/cancelled state  
 else dict of query output

**Parameters** `query_execution_id` (*str*) – Id of submitted athena query

**Returns** dict

**poll\_query\_status** (*query\_execution\_id*, *max\_tries=None*)

Poll the status of submitted athena query until query state reaches final state. Returns one of the final states

**Parameters**

- **query\_execution\_id** (*str*) – Id of submitted athena query
- **max\_tries** (*int*) – Number of times to poll for query state before function exits

**Returns** str

**run\_query** (*query*, *query\_context*, *result\_configuration*, *client\_request\_token=None*)

Run Presto query on athena with provided config and return submitted query\_execution\_id

**Parameters**

- **query** (*str*) – Presto query to run
- **query\_context** (*dict*) – Context in which query need to be run
- **result\_configuration** (*dict*) – Dict with path to store results in and config related to encryption
- **client\_request\_token** (*str*) – Unique token created by user to avoid multiple executions of same query

**Returns** str

**stop\_query** (*query\_execution\_id*)

Cancel the submitted athena query

**Parameters** `query_execution_id` (*str*) – Id of submitted athena query

**Returns** dict

```
class airflow.contrib.hooks.aws_dynamodb_hook.AwsDynamoDBHook (table_keys=None,
                                                                ta-
                                                                ble_name=None,
                                                                re-
                                                                gion_name=None,
                                                                *args, **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with AWS DynamoDB.

**Parameters**

- **table\_keys** (*list*) – partition key and sort key
- **table\_name** (*str*) – target DynamoDB table
- **region\_name** (*str*) – aws region name (example: us-east-1)

**write\_batch\_data** (*items*)

Write batch items to dynamodb table with provisioned throughout capacity.

```
class airflow.contrib.hooks.aws_firehose_hook.AwsFirehoseHook (delivery_stream,
                                                                re-
                                                                gion_name=None,
                                                                *args, **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with AWS Kinesis Firehose. :param delivery\_stream: Name of the delivery stream :type delivery\_stream: str :param region\_name: AWS region name (example: us-east-1) :type region\_name: str

**get\_conn()**  
Returns AwsHook connection object.

**put\_records(records)**  
Write batch records to Kinesis Firehose

```
class airflow.contrib.hooks.aws_glue_catalog_hook.AwsGlueCatalogHook(aws_conn_id='aws_default',
                                                                    re-
                                                                    gion_name=None,
                                                                    *args,
                                                                    **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with AWS Glue Catalog

#### Parameters

- **aws\_conn\_id** (*str*) – ID of the Airflow connection where credentials and extra configuration are stored
- **region\_name** (*str*) – aws region name (example: us-east-1)

**check\_for\_partition** (*database\_name*, *table\_name*, *expression*)  
Checks whether a partition exists

#### Parameters

- **database\_name** (*str*) – Name of hive database (schema) @table belongs to
- **table\_name** (*str*) – Name of hive table @partition belongs to

**Expression** Expression that matches the partitions to check for (eg *a* = 'b' AND *c* = 'd')

**Return type** `bool`

```
>>> hook = AwsGlueCatalogHook()
>>> t = 'static_babynames_partitioned'
>>> hook.check_for_partition('airflow', t, "ds='2015-01-01'")
True
```

**get\_conn()**  
Returns glue connection object.

**get\_partitions** (*database\_name*, *table\_name*, *expression*=", *page\_size*=None, *max\_items*=None)  
Retrieves the partition values for a table.

#### Parameters

- **database\_name** (*str*) – The name of the catalog database where the partitions reside.
- **table\_name** (*str*) – The name of the partitions' table.
- **expression** (*str*) – An expression filtering the partitions to be returned. Please see official AWS documentation for further information. <https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-catalog-partitions.html#aws-glue-api-catalog-partitions-GetPartitions>
- **page\_size** (*int*) – pagination size
- **max\_items** (*int*) – maximum items to return

**Returns** set of partition values where each value is a tuple since a partition may be composed of multiple columns. For example: `{('2018-01-01', '1'), ('2018-01-01', '2')}`

```
class airflow.contrib.hooks.aws_hook.AwsHook(aws_conn_id='aws_default',          ver-
                                             ify=None)
```

Bases: `airflow.hooks.base_hook.BaseHook`

Interact with AWS. This class is a thin wrapper around the boto3 python library.

**expand\_role**(role)

If the IAM role is a role name, get the Amazon Resource Name (ARN) for the role. If IAM role is already an IAM role ARN, no change is made.

**Parameters** role – IAM role name or ARN

**Returns** IAM role ARN

**get\_credentials**(region\_name=None)

Get the underlying `botocore.Credentials` object.

This contains the following authentication attributes: access\_key, secret\_key and token.

**get\_session**(region\_name=None)

Get the underlying boto3.session.

```
class airflow.contrib.hooks.aws_lambda_hook.AwsLambdaHook(function_name,      re-
                                                            gion_name=None,
                                                            log_type='None', qual-
                                                            ifier='$LATEST',
                                                            invoca-
                                                            tion_type='RequestResponse',
                                                            *args, **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with AWS Lambda

**Parameters**

- **function\_name** (str) – AWS Lambda Function Name
- **region\_name** (str) – AWS Region Name (example: us-west-2)
- **log\_type** (str) – Tail Invocation Request
- **qualifier** (str) – AWS Lambda Function Version or Alias Name
- **invocation\_type** (str) – AWS Lambda Invocation Type (RequestResponse, Event etc)

**invoke\_lambda**(payload)

Invoke Lambda Function

```
class airflow.contrib.hooks.aws_sns_hook.AwsSnsHook(*args, **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with Amazon Simple Notification Service.

**get\_conn**()

Get an SNS connection

**publish\_to\_target**(target\_arn, message)

Publish a message to a topic or an endpoint.

**Parameters**



- **target\_arn** (*str*) – either a TopicArn or an EndpointArn
- **message** – the default message you want to send
- **message** – str

**class** airflow.contrib.hooks.azure\_container\_instance\_hook.**AzureContainerInstanceHook** (*conn\_id*)

Bases: *airflow.hooks.base\_hook.BaseHook*

A hook to communicate with Azure Container Instances.

This hook requires a service principal in order to work. After creating this service principal (Azure Active Directory/App Registrations), you need to fill in the client\_id (Application ID) as login, the generated password as password, and tenantId and subscriptionId in the extra's field as a json.

**Parameters** **conn\_id** (*str*) – connection id of a service principal which will be used to start the container instance

**create\_or\_update** (*resource\_group, name, container\_group*)

Create a new container group

#### Parameters

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group
- **container\_group** (*azure.mgmt.containerinstance.models.ContainerGroup*) – the properties of the container group

**delete** (*resource\_group, name*)

Delete a container group

#### Parameters

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group

**exists** (*resource\_group, name*)

Test if a container group exists

#### Parameters

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group

**get\_logs** (*resource\_group, name, tail=1000*)

Get the tail from logs of a container group

#### Parameters

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group
- **tail** (*int*) – the size of the tail

**Returns** A list of log messages

**Return type** *list[str]*

**get\_messages** (*resource\_group, name*)

Get the messages of a container group

#### Parameters

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group

**Returns** A list of the event messages

**Return type** `list[str]`

**get\_state\_exitcode\_details** (*resource\_group*, *name*)

Get the state and exitcode of a container group

**Parameters**

- **resource\_group** (*str*) – the name of the resource group
- **name** (*str*) – the name of the container group

**Returns** A tuple with the state, exitcode, and details. If the exitcode is unknown 0 is returned.

**Return type** `tuple(state,exitcode,details)`

**class** `airflow.contrib.hooks.azure_container_registry_hook.AzureContainerRegistryHook` (*conn\_id*)

Bases: `airflow.hooks.base_hook.BaseHook`

A hook to communicate with a Azure Container Registry.

**Parameters** **conn\_id** (*str*) – connection id of a service principal which will be used to start the container instance

**class** `airflow.contrib.hooks.azure_container_volume_hook.AzureContainerVolumeHook` (*wasb\_conn\_id*)

Bases: `airflow.hooks.base_hook.BaseHook`

A hook which wraps an Azure Volume.

**Parameters** **wasb\_conn\_id** (*str*) – connection id of a Azure storage account of which file shares should be mounted

**class** `airflow.contrib.hooks.azure_cosmos_hook.AzureCosmosDBHook` (*azure\_cosmos\_conn\_id*=*'azure\_cosmos'*)

Bases: `airflow.hooks.base_hook.BaseHook`

Interacts with Azure CosmosDB.

login should be the endpoint uri, password should be the master key optionally, you can use the following extras to default these values {"database\_name": "<DATABASE\_NAME>", "collection\_name": "COLLECTION\_NAME"}.

**Parameters** **azure\_cosmos\_conn\_id** (*str*) – Reference to the Azure CosmosDB connection.

**create\_collection** (*collection\_name*, *database\_name*=None)

Creates a new collection in the CosmosDB database.

**create\_database** (*database\_name*)

Creates a new database in CosmosDB.

**delete\_collection** (*collection\_name*, *database\_name*=None)

Deletes an existing collection in the CosmosDB database.

**delete\_database** (*database\_name*)

Deletes an existing database in CosmosDB.

**delete\_document** (*document\_id*, *database\_name*=None, *collection\_name*=None)

Delete an existing document out of a collection in the CosmosDB database.

**does\_collection\_exist** (*collection\_name*, *database\_name*=None)

Checks if a collection exists in CosmosDB.

**does\_database\_exist** (*database\_name*)  
Checks if a database exists in CosmosDB.

**get\_conn** ()  
Return a cosmos db client.

**get\_document** (*document\_id*, *database\_name=None*, *collection\_name=None*)  
Get a document from an existing collection in the CosmosDB database.

**get\_documents** (*sql\_string*, *database\_name=None*, *collection\_name=None*, *partition\_key=None*)  
Get a list of documents from an existing collection in the CosmosDB database via SQL query.

**insert\_documents** (*documents*, *database\_name=None*, *collection\_name=None*)  
Insert a list of new documents into an existing collection in the CosmosDB database.

**upsert\_document** (*document*, *database\_name=None*, *collection\_name=None*, *document\_id=None*)  
Inserts a new document (or updates an existing one) into an existing collection in the CosmosDB database.

**class** airflow.contrib.hooks.azure\_data\_lake\_hook.**AzureDataLakeHook** (*azure\_data\_lake\_conn\_id='azure\_data\_lake\_conn\_id'*)  
Bases: *airflow.hooks.base\_hook.BaseHook*

Interacts with Azure Data Lake.

Client ID and client secret should be in user and password parameters. Tenant and account name should be extra field as {"tenant": "<TENANT>", "account\_name": "ACCOUNT\_NAME"}.

**Parameters** **azure\_data\_lake\_conn\_id** (*str*) – Reference to the Azure Data Lake connection.

**check\_for\_file** (*file\_path*)  
Check if a file exists on Azure Data Lake.

**Parameters** **file\_path** (*str*) – Path and name of the file.

**Returns** True if the file exists, False otherwise.

**Return type** *bool*

**download\_file** (*local\_path*, *remote\_path*, *nthreads=64*, *overwrite=True*, *buffer\_size=4194304*, *block\_size=4194304*)

Download a file from Azure Blob Storage.

**Parameters**

- **local\_path** (*str*) – local path. If downloading a single file, will write to this specific file, unless it is an existing directory, in which case a file is created within it. If downloading multiple files, this is the root directory to write within. Will create directories as required.
- **remote\_path** (*str*) – remote path/globstring to use to find remote files. Recursive glob patterns using \*\* are not supported.
- **nthreads** (*int*) – Number of threads to use. If None, uses the number of cores.
- **overwrite** (*bool*) – Whether to forcibly overwrite existing files/directories. If False and remote path is a directory, will quit regardless if any files would be overwritten or not. If True, only matching filenames are actually overwritten.
- **buffer\_size** (*int*) – int [2\*\*22] Number of bytes for internal buffer. This block cannot be bigger than a chunk and cannot be smaller than a block.
- **block\_size** (*int*) – int [2\*\*22] Number of bytes for a block. Within each chunk, we write a smaller block for each API call. This block cannot be bigger than a chunk.

**get\_conn()**

Return a AzureDLFileSystem object.

**list(path)**

List files in Azure Data Lake Storage

**Parameters** *path* (*str*) – full path/globstring to use to list files in ADLS

**upload\_file** (*local\_path*, *remote\_path*, *nthreads*=64, *overwrite*=True, *buffer\_size*=4194304, *block\_size*=4194304)

Upload a file to Azure Data Lake.

**Parameters**

- **local\_path** (*str*) – local path. Can be single file, directory (in which case, upload recursively) or glob pattern. Recursive glob patterns using \*\* are not supported.
- **remote\_path** (*str*) – Remote path to upload to; if multiple files, this is the directory root to write within.
- **nthreads** (*int*) – Number of threads to use. If None, uses the number of cores.
- **overwrite** (*bool*) – Whether to forcibly overwrite existing files/directories. If False and remote path is a directory, will quit regardless if any files would be overwritten or not. If True, only matching filenames are actually overwritten.
- **buffer\_size** (*int*) – int [2\*\*22] Number of bytes for internal buffer. This block cannot be bigger than a chunk and cannot be smaller than a block.
- **block\_size** (*int*) – int [2\*\*22] Number of bytes for a block. Within each chunk, we write a smaller block for each API call. This block cannot be bigger than a chunk.

**class** airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook (*wasb\_conn\_id*='wasb\_default')

Bases: *airflow.hooks.base\_hook.BaseHook*

Interacts with Azure FileShare Storage.

Additional options passed in the 'extra' field of the connection will be passed to the *FileService()* constructor.

**Parameters** *wasb\_conn\_id* (*str*) – Reference to the wasb connection.

**check\_for\_directory** (*share\_name*, *directory\_name*, *\*\*kwargs*)

Check if a directory exists on Azure File Share.

**Parameters**

- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.exists()* takes.

**Returns** True if the file exists, False otherwise.

**Return type** *bool*

**check\_for\_file** (*share\_name*, *directory\_name*, *file\_name*, *\*\*kwargs*)

Check if a file exists on Azure File Share.

**Parameters**

- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.exists()* takes.

**Returns** True if the file exists, False otherwise.

**Return type** `bool`

**create\_directory** (*share\_name*, *directory\_name*, *\*\*kwargs*)

Create a new directory on a Azure File Share.

**Parameters**

- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.create\_directory()* takes.

**Returns** A list of files and directories

**Return type** `list`

**get\_conn** ()

Return the FileService object.

**get\_file** (*file\_path*, *share\_name*, *directory\_name*, *file\_name*, *\*\*kwargs*)

Download a file from Azure File Share.

**Parameters**

- **file\_path** (*str*) – Where to store the file.
- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.get\_file\_to\_path()* takes.

**get\_file\_to\_stream** (*stream*, *share\_name*, *directory\_name*, *file\_name*, *\*\*kwargs*)

Download a file from Azure File Share.

**Parameters**

- **stream** (*file-like object*) – A filehandle to store the file to.
- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.get\_file\_to\_stream()* takes.

**list\_directories\_and\_files** (*share\_name*, *directory\_name=None*, *\*\*kwargs*)

Return the list of directories and files stored on a Azure File Share.

**Parameters**

- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.list\_directories\_and\_files()* takes.

**Returns** A list of files and directories

Return type `list`

**load\_file** (*file\_path*, *share\_name*, *directory\_name*, *file\_name*, *\*\*kwargs*)

Upload a file to Azure File Share.

#### Parameters

- **file\_path** (*str*) – Path to the file to load.
- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.create\_file\_from\_path()* takes.

**load\_stream** (*stream*, *share\_name*, *directory\_name*, *file\_name*, *count*, *\*\*kwargs*)

Upload a stream to Azure File Share.

#### Parameters

- **stream** (*file-like*) – Opened file/stream to upload as the file content.
- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **count** (*int*) – Size of the stream in bytes
- **kwargs** (*object*) – Optional keyword arguments that *FileService.create\_file\_from\_stream()* takes.

**load\_string** (*string\_data*, *share\_name*, *directory\_name*, *file\_name*, *\*\*kwargs*)

Upload a string to Azure File Share.

#### Parameters

- **string\_data** (*str*) – String to load.
- **share\_name** (*str*) – Name of the share.
- **directory\_name** (*str*) – Name of the directory.
- **file\_name** (*str*) – Name of the file.
- **kwargs** (*object*) – Optional keyword arguments that *FileService.create\_file\_from\_text()* takes.

```
class airflow.contrib.hooks.bigquery_hook.BigQueryHook (bigquery_conn_id='bigquery_default',
                                                    delegate_to=None,
                                                    use_legacy_sql=True,
                                                    location=None)
```

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*, *airflow.hooks.dbapi\_hook.DbApiHook*

Interact with BigQuery. This hook uses the Google Cloud Platform connection.

**get\_conn** ()

Returns a BigQuery PEP 249 connection object.

**get\_pandas\_df** (*sql*, *parameters=None*, *dialect=None*)

Returns a Pandas DataFrame for the results produced by a BigQuery query. The DbApiHook method must be overridden because Pandas doesn't support PEP 249 connections, except for SQLite. See:

<https://github.com/pydata/pandas/blob/master/pandas/io/sql.py#L447> <https://github.com/pydata/pandas/issues/6900>

#### Parameters

- **sql** (*str*) – The BigQuery SQL to execute.
- **parameters** (*mapping or iterable*) – The parameters to render the SQL query with (not used, leave to override superclass method)
- **dialect** (*str in {'legacy', 'standard'}*) – Dialect of BigQuery SQL – legacy SQL or standard SQL defaults to use *self.use\_legacy\_sql* if not specified

#### **get\_service()**

Returns a BigQuery service object.

#### **insert\_rows** (*table, rows, target\_fields=None, commit\_every=1000*)

Insertion is currently unsupported. Theoretically, you could use BigQuery’s streaming API to insert rows into a table, but this hasn’t been implemented.

#### **table\_exists** (*project\_id, dataset\_id, table\_id*)

Checks for the existence of a table in Google BigQuery.

#### Parameters

- **project\_id** (*str*) – The Google cloud project in which to look for the table. The connection supplied to the hook must provide access to the specified project.
- **dataset\_id** (*str*) – The name of the dataset in which to look for the table.
- **table\_id** (*str*) – The name of the table to check the existence of.

**class** airflow.contrib.hooks.gcp\_bigtable\_hook.**BigtableHook** (*gcp\_conn\_id='google\_cloud\_default', delegate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for Google Cloud Bigtable APIs.

All the methods in the hook where *project\_id* is used must be called with keyword arguments rather than positional.

#### **create\_instance** (*\*args, \*\*kwargs*)

Creates new instance.

#### Parameters

- **instance\_id** (*str*) – The ID for the new instance.
- **main\_cluster\_id** (*str*) – The ID for main cluster for the new instance.
- **main\_cluster\_zone** (*str*) – The zone for main cluster. See <https://cloud.google.com/bigtable/docs/locations> for more details.
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the BigTable exists. If set to None or missing, the default *project\_id* from the GCP connection is used.
- **replica\_cluster\_id** (*str*) – (optional) The ID for replica cluster for the new instance.
- **replica\_cluster\_zone** (*str*) – (optional) The zone for replica cluster.
- **instance\_type** (*enums.Instance.Type*) – (optional) The type of the instance.
- **instance\_display\_name** (*str*) – (optional) Human-readable name of the instance. Defaults to *instance\_id*.

- **instance\_labels** (*dict*) – (optional) Dictionary of labels to associate with the instance.
- **cluster\_nodes** (*int*) – (optional) Number of nodes for cluster.
- **cluster\_storage\_type** (*enums.StorageType*) – (optional) The type of storage.
- **timeout** (*int*) – (optional) timeout (in seconds) for instance creation. If None is not specified, Operator will wait indefinitely.

**static create\_table** (*instance, table\_id, initial\_split\_keys=None, column\_families=None*)

Creates the specified Cloud Bigtable table. Raises `google.api_core.exceptions.AlreadyExists` if the table exists.

#### Parameters

- **instance** (*Instance*) – The Cloud Bigtable instance that owns the table.
- **table\_id** (*str*) – The ID of the table to create in Cloud Bigtable.
- **initial\_split\_keys** (*list*) – (Optional) A list of row keys in bytes to use to initially split the table.
- **column\_families** (*dict*) – (Optional) A map of columns to create. The key is the `column_id` str, and the value is a `google.cloud.bigtable.column_family.GarbageCollectionRule`.

**delete\_instance** (*\*args, \*\*kwargs*)

Deletes the specified Cloud Bigtable instance. Raises `google.api_core.exceptions.NotFound` if the Cloud Bigtable instance does not exist.

#### Parameters

- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the BigTable exists. If set to None or missing, the default `project_id` from the GCP connection is used.
- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.

**delete\_table** (*\*args, \*\*kwargs*)

Deletes the specified table in Cloud Bigtable. Raises `google.api_core.exceptions.NotFound` if the table does not exist.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.
- **table\_id** (*str*) – The ID of the table in Cloud Bigtable.
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the BigTable exists. If set to None or missing, the default `project_id` from the GCP connection is used.

**static get\_cluster\_states\_for\_table** (*instance, table\_id*)

Fetches Cluster States for the specified table in Cloud Bigtable. Raises `google.api_core.exceptions.NotFound` if the table does not exist.

#### Parameters

- **instance** (*Instance*) – The Cloud Bigtable instance that owns the table.
- **table\_id** (*str*) – The ID of the table in Cloud Bigtable to fetch Cluster States from.

**static get\_column\_families\_for\_table** (*instance, table\_id*)

Fetches Column Families for the specified table in Cloud Bigtable.

#### Parameters



- **instance** (*Instance*) – The Cloud Bigtable instance that owns the table.
- **table\_id** (*str*) – The ID of the table in Cloud Bigtable to fetch Column Families from.

**get\_instance** (\*args, \*\*kwargs)

Retrieves and returns the specified Cloud Bigtable instance if it exists. Otherwise, returns None.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Bigtable instance.
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the BigTable exists. If set to None or missing, the default project\_id from the GCP connection is used.

**static update\_cluster** (*instance, cluster\_id, nodes*)

Updates number of nodes in the specified Cloud Bigtable cluster. Raises google.api\_core.exceptions.NotFound if the cluster does not exist.

#### Parameters

- **instance** (*Instance*) – The Cloud Bigtable instance that owns the cluster.
- **cluster\_id** (*str*) – The ID of the cluster.
- **nodes** (*int*) – The desired number of nodes.

**class** airflow.contrib.hooks.cassandra\_hook.**CassandraHook** (*cassandra\_conn\_id='cassandra\_default'*)

Bases: [airflow.hooks.base\\_hook.BaseHook](#), [airflow.utils.log.logging\\_mixin.LoggingMixin](#)

Hook used to interact with Cassandra

Contact points can be specified as a comma-separated string in the 'hosts' field of the connection.

Port can be specified in the port field of the connection.

If SSL is enabled in Cassandra, pass in a dict in the extra field as kwargs for `ssl.wrap_socket()`. For example:

```
{
    'ssl_options' : {
        'ca_certs' : PATH_TO_CA_CERTS
    }
}
```

Default load balancing policy is RoundRobinPolicy. To specify a different LB policy:

```
- DCAwareRoundRobinPolicy
    {
        'load_balancing_policy': 'DCAwareRoundRobinPolicy',
        'load_balancing_policy_args': {
            'local_dc': LOCAL_DC_NAME,                // optional
            'used_hosts_per_remote_dc': SOME_INT_VALUE, // optional
        }
    }
- WhiteListRoundRobinPolicy
    {
        'load_balancing_policy': 'WhiteListRoundRobinPolicy',
        'load_balancing_policy_args': {
            'hosts': ['HOST1', 'HOST2', 'HOST3']
        }
    }
- TokenAwarePolicy
```

(continues on next page)

(continued from previous page)

```
{
    'load_balancing_policy': 'TokenAwarePolicy',
    'load_balancing_policy_args': {
        'child_load_balancing_policy': CHILD_POLICY_NAME, // optional
        'child_load_balancing_policy_args': { ... }         // optional
    }
}
```

For details of the Cluster config, see `cassandra.cluster`.

**get\_conn()**

Returns a cassandra Session object

**record\_exists(table, keys)**

Checks if a record exists in Cassandra

#### Parameters

- **table** (*str*) – Target Cassandra table. Use dot notation to target a specific keyspace.
- **keys** (*dict*) – The keys and their values to check the existence.

**shutdown\_cluster()**

Closes all sessions and connections associated with this Cluster.

**table\_exists(table)**

Checks if a table exists in Cassandra

**Parameters** **table** (*str*) – Target Cassandra table. Use dot notation to target a specific keyspace.

**class** `airflow.contrib.hooks.gcp_spanner_hook.CloudSpannerHook` (*gcp\_conn\_id='google\_cloud\_default', delete\_gate\_to=None*)

Bases: `airflow.contrib.hooks.gcp_api_base_hook.GoogleCloudBaseHook`

Hook for Google Cloud Spanner APIs.

All the methods in the hook where `project_id` is used must be called with keyword arguments rather than positional.

**create\_database(\*args, \*\*kwargs)**

Creates a new database in Cloud Spanner.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **database\_id** (*str*) – The ID of the database to create in Cloud Spanner.
- **ddl\_statements** (*list[str]*) – The string list containing DDL for the new database.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default `project_id` from the GCP connection is used.

**Returns** None

**create\_instance(\*args, \*\*kwargs)**

Creates a new Cloud Spanner instance.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **configuration\_name** (*str*) – The name of the instance configuration defining how the instance will be created. Possible configuration values can be retrieved via <https://cloud.google.com/spanner/docs/reference/rest/v1/projects.instanceConfigs/list>
- **node\_count** (*int*) – (Optional) The number of nodes allocated to the Cloud Spanner instance.
- **display\_name** (*str*) – (Optional) The display name for the instance in the GCP Console. Must be between 4 and 30 characters. If this value is not set in the constructor, the name falls back to the instance ID.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**delete\_database** (*\*args, \*\*kwargs*)

Drops a database in Cloud Spanner.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **database\_id** (*str*) – The ID of the database in Cloud Spanner.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** True if everything succeeded

**Return type** bool

**delete\_instance** (*\*args, \*\*kwargs*)

Deletes an existing Cloud Spanner instance.

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**execute\_dml** (*\*args, \*\*kwargs*)

Executes an arbitrary DML query (INSERT, UPDATE, DELETE).

#### Parameters

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **database\_id** (*str*) – The ID of the database in Cloud Spanner.
- **queries** (*str*) – The queries to execute.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**get\_database** (\*args, \*\*kwargs)

Retrieves a database in Cloud Spanner. If the database does not exist in the specified instance, it returns None.

**Parameters**

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **database\_id** (*str*) – The ID of the database in Cloud Spanner.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** Database object or None if database does not exist

**Return type** `google.cloud.spanner_v1.database.Database` or `None`

**get\_instance** (\*args, \*\*kwargs)

Gets information about a particular instance.

**Parameters**

- **project\_id** (*str*) – Optional, The ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.
- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.

**Returns** `google.cloud.spanner_v1.instance.Instance`

**Return type** `object`

**update\_database** (\*args, \*\*kwargs)

Updates DDL of a database in Cloud Spanner.

**Parameters**

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **database\_id** (*str*) – The ID of the database in Cloud Spanner.
- **ddl\_statements** (*list[str]*) – The string list containing DDL for the new database.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.
- **operation\_id** (*str*) – (Optional) The unique per database operation ID that can be specified to implement idempotency check.

**Returns** None

**update\_instance** (\*args, \*\*kwargs)

Updates an existing Cloud Spanner instance.

**Parameters**

- **instance\_id** (*str*) – The ID of the Cloud Spanner instance.
- **configuration\_name** (*str*) – The name of the instance configuration defining how the instance will be created. Possible configuration values can be retrieved via <https://cloud.google.com/spanner/docs/reference/rest/v1/projects.instanceConfigs/list>

- **node\_count** (*int*) – (Optional) The number of nodes allocated to the Cloud Spanner instance.
- **display\_name** (*str*) – (Optional) The display name for the instance in the GCP Console. Must be between 4 and 30 characters. If this value is not set in the constructor, the name falls back to the instance ID.
- **project\_id** (*str*) – Optional, the ID of the GCP project that owns the Cloud Spanner database. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

```
class airflow.contrib.hooks.gcp_sql_hook.CloudSqlDatabaseHook (gcp_cloudsql_conn_id='google_cloud_sql_de-  
fault_gcp_project_id=None)
```

Bases: *airflow.hooks.base\_hook.BaseHook*

Serves DB connection configuration for Google Cloud SQL (Connections of *gcpcloudsql://* type).

The hook is a “meta” one. It does not perform an actual connection. It is there to retrieve all the parameters configured in *gcpcloudsql://* connection, start/stop Cloud SQL Proxy if needed, dynamically generate Postgres or MySQL connection in the database and return an actual Postgres or MySQL hook. The returned Postgres/MySQL hooks are using direct connection or Cloud SQL Proxy socket/TCP as configured.

Main parameters of the hook are retrieved from the standard URI components:

- **user** - User name to authenticate to the database (from login of the URI).
- **password** - Password to authenticate to the database (from password of the URI).
- **public\_ip** - IP to connect to for public connection (from host of the URI).
- **public\_port** - Port to connect to for public connection (from port of the URI).
- **database** - Database to connect to (from schema of the URI).

Remaining parameters are retrieved from the extras (URI query parameters):

- **project\_id** - **Optional, Google Cloud Platform project where the Cloud SQL** instance exists. If missing, default project id passed is used.
- **instance** - Name of the instance of the Cloud SQL database instance.
- **location** - The location of the Cloud SQL instance (for example *europe-west1*).
- **database\_type** - The type of the database instance (MySQL or Postgres).
- **use\_proxy** - (default False) Whether SQL proxy should be used to connect to Cloud SQL DB.
- **use\_ssl** - (default False) Whether SSL should be used to connect to Cloud SQL DB. You cannot use proxy and SSL together.
- **sql\_proxy\_use\_tcp** - (default False) If set to true, TCP is used to connect via proxy, otherwise UNIX sockets are used.
- **sql\_proxy\_binary\_path** - Optional path to Cloud SQL Proxy binary. If the binary is not specified or the binary is not present, it is automatically downloaded.
- **sql\_proxy\_version** - Specific version of the proxy to download (for example *v1.13*). If not specified, the latest version is downloaded.
- **sslcert** - Path to client certificate to authenticate when SSL is used.
- **sslkey** - Path to client private key to authenticate when SSL is used.
- **sslrootcert** - Path to server’s certificate to authenticate when SSL is used.

**Parameters**

- **gcp\_cloudsql\_conn\_id** (*str*) – URL of the connection
- **default\_gcp\_project\_id** (*str*) – Default project id used if project\_id not specified in the connection URL

**cleanup\_database\_hook** ()

Clean up database hook after it was used.

**create\_connection** (\*\*kwargs)

Create connection in the Connection table, according to whether it uses proxy, TCP, UNIX sockets, SSL. Connection ID will be randomly generated.

**Parameters** **session** – Session of the SQL Alchemy ORM (automatically generated with decorator).

**delete\_connection** (\*\*kwargs)

Delete the dynamically created connection from the Connection table.

**Parameters** **session** – Session of the SQL Alchemy ORM (automatically generated with decorator).

**free\_reserved\_port** ()

Free TCP port. Makes it immediately ready to be used by Cloud SQL Proxy.

**get\_database\_hook** ()

Retrieve database hook. This is the actual Postgres or MySQL database hook that uses proxy or connects directly to the Google Cloud SQL database.

**get\_sqlproxy\_runner** ()

Retrieve Cloud SQL Proxy runner. It is used to manage the proxy lifecycle per task.

**Returns** The Cloud SQL Proxy runner.

**Return type** CloudSqlProxyRunner

**reserve\_free\_tcp\_port** ()

Reserve free TCP port to be used by Cloud SQL Proxy

**retrieve\_connection** (\*\*kwargs)

Retrieves the dynamically created connection from the Connection table.

**Parameters** **session** – Session of the SQL Alchemy ORM (automatically generated with decorator).

```
class airflow.contrib.hooks.gcp_sql_hook.CloudSqlHook (api_version,  
                                                    gcp_conn_id='google_cloud_default',  
                                                    delegate_to=None)
```

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for Google Cloud SQL APIs.

All the methods in the hook where project\_id is used must be called with keyword arguments rather than positional.

**create\_database** (\*args, \*\*kwargs)

Creates a new database inside a Cloud SQL instance.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/databases/insert#request-body>.

- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**create\_instance** (\*args, \*\*kwargs)

Creates a new Cloud SQL instance.

**Parameters**

- **body** (*dict*) – Body required by the Cloud SQL insert API, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/insert#request-body>.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**delete\_database** (\*args, \*\*kwargs)

Deletes a database from a Cloud SQL instance.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **database** (*str*) – Name of the database to be deleted in the instance.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**delete\_instance** (\*args, \*\*kwargs)

Deletes a Cloud SQL instance.

**Parameters**

- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.
- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.

**Returns** None

**export\_instance** (\*args, \*\*kwargs)

Exports data from a Cloud SQL instance to a Cloud Storage bucket as a SQL dump or CSV file.

**Parameters**

- **instance** (*str*) – Database instance ID of the Cloud SQL instance. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/export#request-body>
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**get\_conn** ()

Retrieves connection to Cloud SQL.

**Returns** Google Cloud SQL services object.

**Return type** *dict*

**get\_database** (\*args, \*\*kwargs)

Retrieves a database resource from a Cloud SQL instance.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **database** (*str*) – Name of the database in the instance.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** A Cloud SQL database resource, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/databases#resource>.

**Return type** dict

**get\_instance** (\*args, \*\*kwargs)

Retrieves a resource containing information about a Cloud SQL instance.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** A Cloud SQL instance resource.

**Return type** dict

**import\_instance** (\*args, \*\*kwargs)

Imports data into a Cloud SQL instance from a SQL dump or CSV file in Cloud Storage.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/export#request-body>
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**patch\_database** (\*args, \*\*kwargs)

Updates a database resource inside a Cloud SQL instance.

This method supports patch semantics. See <https://cloud.google.com/sql/docs/mysql/admin-api/how-tos/performance#patch>.

**Parameters**

- **instance** (*str*) – Database instance ID. This does not include the project ID.
- **database** (*str*) – Name of the database to be updated in the instance.
- **body** (*dict*) – The request body, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/databases/insert#request-body>.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None



**patch\_instance** (\*args, \*\*kwargs)  
Updates settings of a Cloud SQL instance.

Caution: This is not a partial update, so you must include values for all the settings that you want to retain.

#### Parameters

- **body** (*dict*) – Body required by the Cloud SQL patch API, as described in <https://cloud.google.com/sql/docs/mysql/admin-api/v1beta4/instances/patch#request-body>.
- **instance** (*str*) – Cloud SQL instance ID. This does not include the project ID.
- **project\_id** (*str*) – Project ID of the project that contains the instance. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**class** airflow.contrib.hooks.cloudant\_hook.**CloudantHook** (cloudant\_conn\_id='cloudant\_default')  
Bases: *airflow.hooks.base\_hook.BaseHook*

Interact with Cloudant.

This class is a thin wrapper around the cloudant python library. See the documentation [here](https://github.com/cloudant-labs/cloudant-python).

**db** ()  
Returns the Database object for this hook.

See the documentation for cloudant-python here <https://github.com/cloudant-labs/cloudant-python>.

**class** airflow.contrib.hooks.gcp\_dataflow\_hook.**DataFlowHook** (gcp\_conn\_id='google\_cloud\_default',  
delegate\_to=None,  
poll\_sleep=10)  
Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

**get\_conn** ()  
Returns a Google Cloud Dataflow service object.

**class** airflow.contrib.hooks.gcp\_dataproc\_hook.**DataProcHook** (gcp\_conn\_id='google\_cloud\_default',  
delegate\_to=None,  
api\_version='v1beta2')  
Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for Google Cloud Dataproc APIs.

**await** (operation)  
Awaits for Google Cloud Dataproc Operation to complete.

**get\_conn** ()  
Returns a Google Cloud Dataproc service object.

**wait** (operation)  
Awaits for Google Cloud Dataproc Operation to complete.

**class** airflow.contrib.hooks.databricks\_hook.**DatabricksHook** (databricks\_conn\_id='databricks\_default',  
time-  
out\_seconds=180,  
retry\_limit=3,  
retry\_delay=1.0)  
Bases: *airflow.hooks.base\_hook.BaseHook*

Interact with Databricks.

**run\_now** (json)  
Utility function to call the api/2.0/jobs/run-now endpoint.

**Parameters** `json(dict)` – The data used in the body of the request to the `run-now` endpoint.

**Returns** the `run_id` as a string

**Return type** `str`

**submit\_run** (*json*)

Utility function to call the `api/2.0/jobs/runs/submit` endpoint.

**Parameters** `json(dict)` – The data used in the body of the request to the `submit` endpoint.

**Returns** the `run_id` as a string

**Return type** `str`

**class** `airflow.contrib.hooks.datadog_hook.DatadogHook` (*datadog\_conn\_id='datadog\_default'*)  
 Bases: `airflow.hooks.base_hook.BaseHook`, `airflow.utils.log.logging_mixin.LoggingMixin`

Uses datadog API to send metrics of practically anything measurable, so it's possible to track # of db records inserted/deleted, records read from file and many other useful metrics.

Depends on the datadog API, which has to be deployed on the same server where Airflow runs.

**Parameters**

- **datadog\_conn\_id** – The connection to datadog, containing metadata for api keys.
- **datadog\_conn\_id** – str

**post\_event** (*title, text, aggregation\_key=None, alert\_type=None, date\_happened=None, handle=None, priority=None, related\_event\_id=None, tags=None, device\_name=None*)

Posts an event to datadog (processing finished, potentially alerts, other issues) Think about this as a means to maintain persistence of alerts, rather than alerting itself.

**Parameters**

- **title** (*str*) – The title of the event
- **text** (*str*) – The body of the event (more information)
- **aggregation\_key** (*str*) – Key that can be used to aggregate this event in a stream
- **alert\_type** (*str*) – The alert type for the event, one of ["error", "warning", "info", "success"]
- **date\_happened** (*int*) – POSIX timestamp of the event; defaults to now
- **handle** – str
- **priority** (*str*) – Priority to post the event as. ("normal" or "low", defaults to "normal")
- **related\_event\_id** (*id*) – Post event as a child of the given event
- **tags** (*list[str]*) – List of tags to apply to the event
- **device\_name** (*list*) – device\_name to post the event with

**Handle** User to post the event as; defaults to owner of the application key used to submit.

**query\_metric** (*query, from\_seconds\_ago, to\_seconds\_ago*)

Queries datadog for a specific metric, potentially with some function applied to it and returns the results.

**Parameters**

- **query** (*str*) – The datadog query to execute (see datadog docs)
- **from\_seconds\_ago** (*int*) – How many seconds ago to start querying for.

- **to\_seconds\_ago** (*int*) – Up to how many seconds ago to query for.

**send\_metric** (*metric\_name*, *datapoint*, *tags=None*, *type\_=None*, *interval=None*)

Sends a single datapoint metric to DataDog

#### Parameters

- **metric\_name** (*str*) – The name of the metric
- **datapoint** (*int or float*) – A single integer or float related to the metric
- **tags** (*list*) – A list of tags associated with the metric
- **type** (*str*) – Type of your metric: gauge, rate, or count
- **interval** (*int*) – If the type of the metric is rate or count, define the corresponding interval

**class** airflow.contrib.hooks.datastore\_hook.**DatastoreHook** (*datastore\_conn\_id='google\_cloud\_datastore\_def'*, *delegate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Interact with Google Cloud Datastore. This hook uses the Google Cloud Platform connection.

This object is not threads safe. If you want to make multiple requests simultaneously, you will need to create a hook per thread.

**allocate\_ids** (*partialKeys*)

Allocate IDs for incomplete keys. see <https://cloud.google.com/datastore/docs/reference/rest/v1/projects/allocateIds>

**Parameters** *partialKeys* – a list of partial keys

**Returns** a list of full keys.

**begin\_transaction** ()

Get a new transaction handle

**See also:**

<https://cloud.google.com/datastore/docs/reference/rest/v1/projects/beginTransaction>

**Returns** a transaction handle

**commit** (*body*)

Commit a transaction, optionally creating, deleting or modifying some entities.

**See also:**

<https://cloud.google.com/datastore/docs/reference/rest/v1/projects/commit>

**Parameters** *body* – the body of the commit request

**Returns** the response body of the commit request

**delete\_operation** (*name*)

Deletes the long-running operation

**Parameters** *name* – the name of the operation resource

**export\_to\_storage\_bucket** (*bucket*, *namespace=None*, *entity\_filter=None*, *labels=None*)

Export entities from Cloud Datastore to Cloud Storage for backup

**get\_conn** (*version='v1'*)

Returns a Google Cloud Datastore service object.

**get\_operation** (*name*)

Gets the latest state of a long-running operation

**Parameters** **name** – the name of the operation resource

**import\_from\_storage\_bucket** (*bucket, file, namespace=None, entity\_filter=None, labels=None*)

Import a backup from Cloud Storage to Cloud Datastore

**lookup** (*keys, read\_consistency=None, transaction=None*)

Lookup some entities by key

**See also:**

<https://cloud.google.com/datastore/docs/reference/rest/v1/projects/lookup>

**Parameters**

- **keys** – the keys to lookup
- **read\_consistency** – the read consistency to use. default, strong or eventual. Cannot be used with a transaction.
- **transaction** – the transaction to use, if any.

**Returns** the response body of the lookup request.

**poll\_operation\_until\_done** (*name, polling\_interval\_in\_seconds*)

Poll backup operation state until it's completed

**rollback** (*transaction*)

Roll back a transaction

**See also:**

<https://cloud.google.com/datastore/docs/reference/rest/v1/projects/rollback>

**Parameters** **transaction** – the transaction to roll back

**run\_query** (*body*)

Run a query for entities.

**See also:**

<https://cloud.google.com/datastore/docs/reference/rest/v1/projects/runQuery>

**Parameters** **body** – the body of the query request

**Returns** the batch of query results.

```
class airflow.contrib.hooks.discord_webhook_hook.DiscordWebhookHook (http_conn_id=None,
                                                                    web-
                                                                    hook_endpoint=None,
                                                                    mes-
                                                                    sage= "",
                                                                    user-
                                                                    name=None,
                                                                    avatar_url=None,
                                                                    tts=False,
                                                                    proxy=None,
                                                                    *args,
                                                                    **kwargs)
```

Bases: `airflow.hooks.http_hook.HttpHook`

This hook allows you to post messages to Discord using incoming webhooks. Takes a Discord connection ID with a default relative webhook endpoint. The default endpoint can be overridden using the `webhook_endpoint` parameter (<https://discordapp.com/developers/docs/resources/webhook>).

Each Discord webhook can be pre-configured to use a specific username and `avatar_url`. You can override these defaults in this hook.

#### Parameters

- **`http_conn_id`** (*str*) – Http connection ID with host as “<https://discord.com/api/>” and default webhook endpoint in the extra field in the form of {“webhook\_endpoint”: “webhooks/{webhook.id}/{webhook.token}”}
- **`webhook_endpoint`** (*str*) – Discord webhook endpoint in the form of “webhooks/{webhook.id}/{webhook.token}”
- **`message`** (*str*) – The message you want to send to your Discord channel (max 2000 characters)
- **`username`** (*str*) – Override the default username of the webhook
- **`avatar_url`** (*str*) – Override the default avatar of the webhook
- **`tts`** (*bool*) – Is a text-to-speech message
- **`proxy`** (*str*) – Proxy to use to make the Discord webhook call

#### **`execute()`**

Execute the Discord webhook call

```
class airflow.contrib.hooks.emr_hook.EmrHook (emr_conn_id=None, region_name=None,
                                             *args, **kwargs)
```

Bases: *airflow.contrib.hooks.aws\_hook.AwsHook*

Interact with AWS EMR. `emr_conn_id` is only necessary for using the `create_job_flow` method.

#### **`create_job_flow`** (*job\_flow\_overrides*)

Creates a job flow using the config from the EMR connection. Keys of the json extra hash may have the arguments of the boto3 `run_job_flow` method. Overrides for this config may be passed as the `job_flow_overrides`.

```
class airflow.contrib.hooks.fs_hook.FSHook (conn_id='fs_default')
```

Bases: *airflow.hooks.base\_hook.BaseHook*

Allows for interaction with an file server.

Connection should have a name and a path specified under extra:

example: Conn Id: `fs_test` Conn Type: File (path) Host, Schema, Login, Password, Port: empty Extra: {“path”: “/tmp”}

```
class airflow.contrib.hooks.ftp_hook.FTPHook (ftp_conn_id='ftp_default')
```

Bases: *airflow.hooks.base\_hook.BaseHook*

Interact with FTP.

Errors that may occur throughout but should be handled downstream. You can specify mode for data transfers in the extra field of your connection as {“passive”: “true”}.

#### **`close_conn()`**

Closes the connection. An error will occur if the connection wasn’t ever opened.

#### **`create_directory`** (*path*)

Creates a directory on the remote system.

**Parameters** `path` (*str*) – full path to the remote directory to create

**delete\_directory** (*path*)

Deletes a directory on the remote system.

**Parameters** **path** (*str*) – full path to the remote directory to delete

**delete\_file** (*path*)

Removes a file on the FTP Server.

**Parameters** **path** (*str*) – full path to the remote file

**describe\_directory** (*path*)

Returns a dictionary of {filename: {attributes}} for all files on the remote system (where the MLSD command is supported).

**Parameters** **path** (*str*) – full path to the remote directory

**get\_conn** ()

Returns a FTP connection object

**get\_mod\_time** (*path*)

Returns a datetime object representing the last time the file was modified

**Parameters** **path** (*string*) – remote file path

**get\_size** (*path*)

Returns the size of a file (in bytes)

**Parameters** **path** (*string*) – remote file path

**list\_directory** (*path, nlst=False*)

Returns a list of files on the remote system.

**Parameters** **path** (*str*) – full path to the remote directory to list

**rename** (*from\_name, to\_name*)

Rename a file.

**Parameters**

- **from\_name** – rename file from name
- **to\_name** – rename file to name

**retrieve\_file** (*remote\_full\_path, local\_full\_path\_or\_buffer, callback=None*)

Transfers the remote file to a local location.

If *local\_full\_path\_or\_buffer* is a string path, the file will be put at that location; if it is a file-like buffer, the file will be written to the buffer but not closed.

**Parameters**

- **remote\_full\_path** (*str*) – full path to the remote file
- **local\_full\_path\_or\_buffer** (*str or file-like buffer*) – full path to the local file or a file-like buffer
- **callback** (*callable*) – callback which is called each time a block of data is read. if you do not use a callback, these blocks will be written to the file or buffer passed in. if you do pass in a callback, note that writing to a file or buffer will need to be handled inside the callback. [default: `output_handle.write()`]

:Example:

```

hook = FTPHook(ftp_conn_id='my_conn')

remote_path = '/path/to/remote/file'
local_path = '/path/to/local/file'

# with a custom callback (in this case displaying progress on each read)
def print_progress(percent_progress):
    self.log.info('Percent Downloaded: %s%%' % percent_progress)

total_downloaded = 0
total_file_size = hook.get_size(remote_path)
output_handle = open(local_path, 'wb')
def write_to_file_with_progress(data):
    total_downloaded += len(data)
    output_handle.write(data)
    percent_progress = (total_downloaded / total_file_size) * 100
    print_progress(percent_progress)
hook.retrieve_file(remote_path, None, callback=write_to_file_with_progress)

# without a custom callback data is written to the local_path
hook.retrieve_file(remote_path, local_path)

```

**store\_file** (*remote\_full\_path*, *local\_full\_path\_or\_buffer*)

Transfers a local file to the remote location.

If *local\_full\_path\_or\_buffer* is a string path, the file will be read from that location; if it is a file-like buffer, the file will be read from the buffer but not closed.

#### Parameters

- **remote\_full\_path** (*str*) – full path to the remote file
- **local\_full\_path\_or\_buffer** (*str* or *file-like buffer*) – full path to the local file or a file-like buffer

**class** airflow.contrib.hooks.ftp\_hook.FTPSHook (*ftp\_conn\_id='ftp\_default'*)

Bases: *airflow.contrib.hooks.ftp\_hook.FTPHook*

**get\_conn**()

Returns a FTPS connection object.

**class** airflow.contrib.hooks.gcp\_transfer\_hook.GCPTransferServiceHook (*api\_version='v1', gcp\_conn\_id='google\_cloud\_dele-gate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for GCP Storage Transfer Service.

**get\_conn**()

Retrieves connection to Google Storage Transfer service.

**Returns** Google Storage Transfer service object

**Return type** dict

**class** airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook (*gcp\_conn\_id='google\_cloud\_default', dele-gate\_to=None, location=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

**create\_cluster** (*cluster*, *project\_id=None*, *retry=<sphinx.ext.autodoc.importer.\_MockObject object>*, *timeout=<sphinx.ext.autodoc.importer.\_MockObject object>*)

Creates a cluster, consisting of the specified number and type of Google Compute Engine instances.

#### Parameters

- **cluster** (*dict* or *google.cloud.container\_v1.types.Cluster*) – A Cluster protobuf or dict. If dict is provided, it must be of the same form as the protobuf message *google.cloud.container\_v1.types.Cluster*
- **project\_id** (*str*) – Google Cloud Platform project ID
- **retry** (*google.api\_core.retry.Retry*) – A retry object (*google.api\_core.retry.Retry*) used to retry requests. If None is specified, requests will not be retried.
- **timeout** (*float*) – The amount of time, in seconds, to wait for the request to complete. Note that if retry is specified, the timeout applies to each individual attempt.

**Returns** The full url to the new, or existing, cluster

**Raises** *ParseError*: On JSON parsing problems when trying to convert dict  
*AirflowException*: cluster is not dict type nor Cluster proto type

**delete\_cluster** (*name*, *project\_id=None*, *retry=<sphinx.ext.autodoc.importer.\_MockObject object>*, *timeout=<sphinx.ext.autodoc.importer.\_MockObject object>*)

Deletes the cluster, including the Kubernetes endpoint and all worker nodes. Firewalls and routes that were configured during cluster creation are also deleted. Other Google Compute Engine resources that might be in use by the cluster (e.g. load balancer resources) will not be deleted if they weren't present at the initial create time.

#### Parameters

- **name** (*str*) – The name of the cluster to delete
- **project\_id** (*str*) – Google Cloud Platform project ID
- **retry** (*google.api\_core.retry.Retry*) – Retry object used to determine when/if to retry requests. If None is specified, requests will not be retried.
- **timeout** (*float*) – The amount of time, in seconds, to wait for the request to complete. Note that if retry is specified, the timeout applies to each individual attempt.

**Returns** The full url to the delete operation if successful, else None

**get\_cluster** (*name*, *project\_id=None*, *retry=<sphinx.ext.autodoc.importer.\_MockObject object>*, *timeout=<sphinx.ext.autodoc.importer.\_MockObject object>*)

Gets details of specified cluster

#### Parameters

- **name** (*str*) – The name of the cluster to retrieve
- **project\_id** (*str*) – Google Cloud Platform project ID
- **retry** (*google.api\_core.retry.Retry*) – A retry object used to retry requests. If None is specified, requests will not be retried.
- **timeout** (*float*) – The amount of time, in seconds, to wait for the request to complete. Note that if retry is specified, the timeout applies to each individual attempt.

**Returns** *google.cloud.container\_v1.types.Cluster*

**get\_operation** (*operation\_name*, *project\_id=None*)

Fetches the operation from Google Cloud



**Parameters**

- **operation\_name** (*str*) – Name of operation to fetch
- **project\_id** (*str*) – Google Cloud Platform project ID

**Returns** The new, updated operation from Google Cloud

**wait\_for\_operation** (*operation*, *project\_id=None*)

Given an operation, continuously fetches the status from Google Cloud until either completion or an error occurring

**Parameters**

- **operation** (*google.cloud.container\_V1.gapic.enums.Operation*) – The Operation to wait for
- **project\_id** (*str*) – Google Cloud Platform project ID

**Returns** A new, updated operation fetched from Google Cloud

```
class airflow.contrib.hooks.gcp_compute_hook.GceHook (api_version='v1',  
                                                    gcp_conn_id='google_cloud_default',  
                                                    delegate_to=None)
```

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for Google Compute Engine APIs.

All the methods in the hook where *project\_id* is used must be called with keyword arguments rather than positional.

**get\_conn** ()

Retrieves connection to Google Compute Engine.

**Returns** Google Compute Engine services object

**Return type** *dict*

**get\_instance\_group\_manager** (*\*args, \*\*kwargs*)

Retrieves Instance Group Manager by *project\_id*, *zone* and *resource\_id*. Must be called with keyword arguments rather than positional.

**Parameters**

- **zone** (*str*) – Google Cloud Platform zone where the Instance Group Manager exists
- **resource\_id** (*str*) – Name of the Instance Group Manager
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to *None* or missing, the default *project\_id* from the GCP connection is used.

**Returns** Instance group manager representation as object according to <https://cloud.google.com/compute/docs/reference/rest/beta/instanceGroupManagers>

**Return type** *dict*

**get\_instance\_template** (*\*args, \*\*kwargs*)

Retrieves instance template by *project\_id* and *resource\_id*. Must be called with keyword arguments rather than positional.

**Parameters**

- **resource\_id** (*str*) – Name of the instance template

- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** Instance template representation as object according to <https://cloud.google.com/compute/docs/reference/rest/v1/instanceTemplates>

**Return type** dict

**insert\_instance\_template** (\*args, \*\*kwargs)

Inserts instance template using body specified. Must be called with keyword arguments rather than positional.

#### Parameters

- **body** (*dict*) – Instance template representation as object according to <https://cloud.google.com/compute/docs/reference/rest/v1/instanceTemplates>
- **request\_id** (*str*) – Optional, unique request\_id that you might add to achieve full idempotence (for example when client call times out repeating the request with the same request id will not create a new instance template again). It should be in UUID format as defined in RFC 4122
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**patch\_instance\_group\_manager** (\*args, \*\*kwargs)

Patches Instance Group Manager with the specified body. Must be called with keyword arguments rather than positional.

#### Parameters

- **zone** (*str*) – Google Cloud Platform zone where the Instance Group Manager exists
- **resource\_id** (*str*) – Name of the Instance Group Manager
- **body** (*dict*) – Instance Group Manager representation as json-merge-patch object according to <https://cloud.google.com/compute/docs/reference/rest/beta/instanceTemplates/patch>
- **request\_id** (*str*) – Optional, unique request\_id that you might add to achieve full idempotence (for example when client call times out repeating the request with the same request id will not create a new instance template again). It should be in UUID format as defined in RFC 4122
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**set\_machine\_type** (\*args, \*\*kwargs)

Sets machine type of an instance defined by project\_id, zone and resource\_id. Must be called with keyword arguments rather than positional.

#### Parameters

- **zone** (*str*) – Google Cloud Platform zone where the instance exists.
- **resource\_id** (*str*) – Name of the Compute Engine instance resource

- **body** (*dict*) – Body required by the Compute Engine setMachineType API, as described in <https://cloud.google.com/compute/docs/reference/rest/v1/instances/setMachineType>
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**start\_instance** (*\*args, \*\*kwargs*)

Starts an existing instance defined by project\_id, zone and resource\_id. Must be called with keyword arguments rather than positional.

**Parameters**

- **zone** (*str*) – Google Cloud Platform zone where the instance exists
- **resource\_id** (*str*) – Name of the Compute Engine instance resource
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**stop\_instance** (*\*args, \*\*kwargs*)

Stops an instance defined by project\_id, zone and resource\_id Must be called with keyword arguments rather than positional.

**Parameters**

- **zone** (*str*) – Google Cloud Platform zone where the instance exists
- **resource\_id** (*str*) – Name of the Compute Engine instance resource
- **project\_id** (*str*) – Optional, Google Cloud Platform project ID where the Compute Engine Instance exists. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

```
class airflow.contrib.hooks.gcp_function_hook.GcfHook (api_version,  
                                                    gcp_conn_id='google_cloud_default',  
                                                    delegate_to=None)
```

Bases: `airflow.contrib.hooks.gcp_api_base_hook.GoogleCloudBaseHook`

Hook for the Google Cloud Functions APIs.

All the methods in the hook where project\_id is used must be called with keyword arguments rather than positional.

**create\_new\_function** (*\*args, \*\*kwargs*)

Creates a new function in Cloud Function in the location specified in the body.

**Parameters**

- **location** (*str*) – The location of the function.
- **body** (*dict*) – The body required by the Cloud Functions insert API.
- **project\_id** (*str*) – Optional, Google Cloud Project project\_id where the function belongs. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** None

**delete\_function** (*name*)

Deletes the specified Cloud Function.

**Parameters** **name** (*str*) – The name of the function.

**Returns** None

**get\_conn** ()

Retrieves the connection to Cloud Functions.

**Returns** Google Cloud Function services object.

**Return type** dict

**get\_function** (*name*)

Returns the Cloud Function with the given name.

**Parameters** **name** (*str*) – Name of the function.

**Returns** A Cloud Functions object representing the function.

**Return type** dict

**update\_function** (*name, body, update\_mask*)

Updates Cloud Functions according to the specified update mask.

**Parameters**

- **name** (*str*) – The name of the function.
- **body** (*dict*) – The body required by the cloud function patch API.
- **update\_mask** (*[str]*) – The update mask - array of fields that should be patched.

**Returns** None

**upload\_function\_zip** (*\*args, \*\*kwargs*)

Uploads zip file with sources.

**Parameters**

- **location** (*str*) – The location where the function is created.
- **zip\_path** (*str*) – The path of the valid .zip file to upload.
- **project\_id** (*str*) – Optional, Google Cloud Project project\_id where the function belongs. If set to None or missing, the default project\_id from the GCP connection is used.

**Returns** The upload URL that was returned by generateUploadUrl method.

**class** airflow.contrib.hooks.gcp\_api\_base\_hook.**GoogleCloudBaseHook** (*gcp\_conn\_id='google\_cloud\_default', delete\_to=None*)

Bases: *airflow.hooks.base\_hook.BaseHook*

A base hook for Google cloud-related hooks. Google cloud has a shared REST API client that is built in the same way no matter which service you use. This class helps construct and authorize the credentials needed to then call googleapiclient.discovery.build() to actually discover and build a client for a Google cloud service.

The class also contains some miscellaneous helper functions.

All hook derived from this base hook use the ‘Google Cloud Platform’ connection type. Three ways of authentication are supported:

Default credentials: Only the ‘Project Id’ is required. You’ll need to have set up default credentials, such as by the `GOOGLE_APPLICATION_DEFAULT` environment variable or from the metadata server on Google Compute Engine.

JSON key file: Specify ‘Project Id’, ‘Keyfile Path’ and ‘Scope’.

Legacy P12 key files are not supported.

JSON data provided in the UI: Specify ‘Keyfile JSON’.

**static fallback\_to\_default\_project\_id** (*func*)

Decorator that provides fallback for Google Cloud Platform project id. If the project is None it will be replaced with the project\_id from the service account the Hook is authenticated with. Project id can be specified either via project\_id kwarg or via first parameter in positional args.

**Parameters** **func** – function to wrap

**Returns** result of the function call

**class** airflow.contrib.hooks.gcp\_kms\_hook.**GoogleCloudKMSHook** (*gcp\_conn\_id='google\_cloud\_default', delegate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Interact with Google Cloud KMS. This hook uses the Google Cloud Platform connection.

**decrypt** (*key\_name, ciphertext, authenticated\_data=None*)

Decrypts a ciphertext message using Google Cloud KMS.

**Parameters**

- **key\_name** (*str*) – The Resource Name for the key to be used for decryption. Of the form `projects/*/locations/*/keyRings/*/cryptoKeys/**`
- **ciphertext** (*str*) – The message to be decrypted.
- **authenticated\_data** (*bytes*) – Any additional authenticated data that was provided when encrypting the message.

**Returns** The original message.

**Return type** *bytes*

**encrypt** (*key\_name, plaintext, authenticated\_data=None*)

Encrypts a plaintext message using Google Cloud KMS.

**Parameters**

- **key\_name** (*str*) – The Resource Name for the key (or key version) to be used for encryption. Of the form `projects/*/locations/*/keyRings/*/cryptoKeys/**`
- **plaintext** (*bytes*) – The message to be encrypted.
- **authenticated\_data** (*bytes*) – Optional additional authenticated data that must also be provided to decrypt the message.

**Returns** The base 64 encoded ciphertext of the original message.

**Return type** *str*

**get\_conn** ()

Returns a KMS service object.

**Return type** *googleapiclient.discovery.Resource*

**class** airflow.contrib.hooks.gcs\_hook.**GoogleCloudStorageHook** (*google\_cloud\_storage\_conn\_id='google\_cloud\_storage\_default', delegate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Interact with Google Cloud Storage. This hook uses the Google Cloud Platform connection.

**copy** (*source\_bucket*, *source\_object*, *destination\_bucket=None*, *destination\_object=None*)

Copies an object from a bucket to another, with renaming if requested.

*destination\_bucket* or *destination\_object* can be omitted, in which case source bucket/object is used, but not both.

#### Parameters

- **source\_bucket** (*str*) – The bucket of the object to copy from.
- **source\_object** (*str*) – The object to copy.
- **destination\_bucket** (*str*) – The destination of the object to copied to. Can be omitted; then the same bucket is used.
- **destination\_object** (*str*) – The (renamed) path of the object if given. Can be omitted; then the same name is used.

**create\_bucket** (*bucket\_name*, *storage\_class='MULTI\_REGIONAL'*, *location='US'*, *project\_id=None*, *labels=None*)

Creates a new bucket. Google Cloud Storage uses a flat namespace, so you can't create a bucket with a name that is already in use.

#### See also:

For more information, see Bucket Naming Guidelines: <https://cloud.google.com/storage/docs/bucketnaming.html#requirements>

#### Parameters

- **bucket\_name** (*str*) – The name of the bucket.
- **storage\_class** (*str*) – This defines how objects in the bucket are stored and determines the SLA and the cost of storage. Values include
  - MULTI\_REGIONAL
  - REGIONAL
  - STANDARD
  - NEARLINE
  - COLDLINE.

If this value is not specified when the bucket is created, it will default to STANDARD.

- **location** (*str*) – The location of the bucket. Object data for objects in the bucket resides in physical storage within this region. Defaults to US.

#### See also:

<https://developers.google.com/storage/docs/bucket-locations>

- **project\_id** (*str*) – The ID of the GCP Project.
- **labels** (*dict*) – User-provided labels, in key/value pairs.

**Returns** If successful, it returns the `id` of the bucket.

**delete** (*bucket*, *object*, *generation=None*)

Delete an object if versioning is not enabled for the bucket, or if generation parameter is used.

#### Parameters

- **bucket** (*str*) – name of the bucket, where the object resides
- **object** (*str*) – name of the object to delete
- **generation** (*str*) – if present, permanently delete the object of this generation

**Returns** True if succeeded

**download** (*bucket, object, filename=None*)

Get a file from Google Cloud Storage.

**Parameters**

- **bucket** (*str*) – The bucket to fetch from.
- **object** (*str*) – The object to fetch.
- **filename** (*str*) – If set, a local file path where the file should be written to.

**exists** (*bucket, object*)

Checks for the existence of a file in Google Cloud Storage.

**Parameters**

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.

**get\_conn** ()

Returns a Google Cloud Storage service object.

**get\_crc32c** (*bucket, object*)

Gets the CRC32c checksum of an object in Google Cloud Storage.

**Parameters**

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.

**get\_md5hash** (*bucket, object*)

Gets the MD5 hash of an object in Google Cloud Storage.

**Parameters**

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.

**get\_size** (*bucket, object*)

Gets the size of a file in Google Cloud Storage.

**Parameters**

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.

**insert\_bucket\_acl** (*bucket, entity, role, user\_project*)

Creates a new ACL entry on the specified bucket. See: [https://cloud.google.com/storage/docs/json\\_api/v1/bucketAccessControls/insert](https://cloud.google.com/storage/docs/json_api/v1/bucketAccessControls/insert)

**Parameters**

- **bucket** (*str*) – Name of a bucket.

- **entity** (*str*) – The entity holding the permission, in one of the following forms: user-userId, user-email, group-groupId, group-email, domain-domain, project-team-projectId, allUsers, allAuthenticatedUsers. See: <https://cloud.google.com/storage/docs/access-control/lists#scopes>
- **role** (*str*) – The access permission for the entity. Acceptable values are: “OWNER”, “READER”, “WRITER”.
- **user\_project** (*str*) – (Optional) The project to be billed for this request. Required for Requester Pays buckets.

**insert\_object\_acl** (*bucket, object\_name, entity, role, generation, user\_project*)

Creates a new ACL entry on the specified object. See: [https://cloud.google.com/storage/docs/json\\_api/v1/objectAccessControls/insert](https://cloud.google.com/storage/docs/json_api/v1/objectAccessControls/insert)

#### Parameters

- **bucket** (*str*) – Name of a bucket.
- **object\_name** (*str*) – Name of the object. For information about how to URL encode object names to be path safe, see: [#encoding](https://cloud.google.com/storage/docs/json_api/#encoding)
- **entity** (*str*) – The entity holding the permission, in one of the following forms: user-userId, user-email, group-groupId, group-email, domain-domain, project-team-projectId, allUsers, allAuthenticatedUsers See: <https://cloud.google.com/storage/docs/access-control/lists#scopes>
- **role** (*str*) – The access permission for the entity. Acceptable values are: “OWNER”, “READER”.
- **generation** (*str*) – (Optional) If present, selects a specific revision of this object (as opposed to the latest version, the default).
- **user\_project** (*str*) – (Optional) The project to be billed for this request. Required for Requester Pays buckets.

**is\_updated\_after** (*bucket, object, ts*)

Checks if an object is updated in Google Cloud Storage.

#### Parameters

- **bucket** (*str*) – The Google cloud storage bucket where the object is.
- **object** (*str*) – The name of the object to check in the Google cloud storage bucket.
- **ts** (*datetime.datetime*) – The timestamp to check against.

**list** (*bucket, versions=None, maxResults=None, prefix=None, delimiter=None*)

List all objects from the bucket with the give string prefix in name

#### Parameters

- **bucket** (*str*) – bucket name
- **versions** (*bool*) – if true, list all versions of the objects
- **maxResults** (*int*) – max count of items to return in a single page of responses
- **prefix** (*str*) – prefix string which filters objects whose name begin with this prefix
- **delimiter** (*str*) – filters objects based on the delimiter (for e.g ‘.csv’)

**Returns** a stream of object names matching the filtering criteria



**rewrite** (*source\_bucket*, *source\_object*, *destination\_bucket*, *destination\_object=None*)

Has the same functionality as `copy`, except that will work on files over 5 TB, as well as when copying between locations and/or storage classes.

*destination\_object* can be omitted, in which case *source\_object* is used.

#### Parameters

- **source\_bucket** (*str*) – The bucket of the object to copy from.
- **source\_object** (*str*) – The object to copy.
- **destination\_bucket** (*str*) – The destination of the object to copied to.
- **destination\_object** (*str*) – The (renamed) path of the object if given. Can be omitted; then the same name is used.

**upload** (*bucket*, *object*, *filename*, *mime\_type='application/octet-stream'*, *gzip=False*, *multipart=False*, *num\_retries=0*)

Uploads a local file to Google Cloud Storage.

#### Parameters

- **bucket** (*str*) – The bucket to upload to.
- **object** (*str*) – The object name to set when uploading the local file.
- **filename** (*str*) – The local file path to the file to be uploaded.
- **mime\_type** (*str*) – The MIME type to set when uploading the file.
- **gzip** (*bool*) – Option to compress file for upload
- **multipart** (*bool or int*) – If True, the upload will be split into multiple HTTP requests. The default size is 256MiB per request. Pass a number instead of True to specify the request size, which must be a multiple of 262144 (256KiB).
- **num\_retries** (*int*) – The number of times to attempt to re-upload the file (or individual chunks, in the case of multipart uploads). Retries are attempted with exponential backoff.

**class** `airflow.contrib.hooks.imap_hook.ImapHook` (*imap\_conn\_id='imap\_default'*)

Bases: `airflow.hooks.base_hook.BaseHook`

This hook connects to a mail server by using the imap protocol.

**Parameters** **imap\_conn\_id** (*str*) – The connection id that contains the information used to authenticate the client.

**download\_mail\_attachments** (*name*, *local\_output\_directory*, *mail\_folder='INBOX'*, *check\_regex=False*, *latest\_only=False*, *not\_found\_mode='raise'*)

Downloads mail's attachments in the mail folder by its name to the local directory.

#### Parameters

- **name** (*str*) – The name of the attachment that will be downloaded.
- **local\_output\_directory** (*str*) – The output directory on the local machine where the files will be downloaded to.
- **mail\_folder** (*str*) – The mail folder where to look at.
- **check\_regex** (*bool*) – Checks the name for a regular expression.
- **latest\_only** (*bool*) – If set to True it will only download the first matched attachment.

- **not\_found\_mode** (*str*) – Specify what should happen if no attachment has been found. Supported values are ‘raise’, ‘warn’ and ‘ignore’. If it is set to ‘raise’ it will raise an exception, if set to ‘warn’ it will only print a warning and if set to ‘ignore’ it won’t notify you at all.

**has\_mail\_attachment** (*name*, *mail\_folder*=‘INBOX’, *check\_regex*=False)

Checks the mail folder for mails containing attachments with the given name.

#### Parameters

- **name** (*str*) – The name of the attachment that will be searched for.
- **mail\_folder** (*str*) – The mail folder where to look at.
- **check\_regex** (*bool*) – Checks the name for a regular expression.

**Returns** True if there is an attachment with the given name and False if not.

**Return type** *bool*

**retrieve\_mail\_attachments** (*name*, *mail\_folder*=‘INBOX’, *check\_regex*=False, *latest\_only*=False, *not\_found\_mode*=‘raise’)

Retrieves mail’s attachments in the mail folder by its name.

#### Parameters

- **name** (*str*) – The name of the attachment that will be downloaded.
- **mail\_folder** (*str*) – The mail folder where to look at.
- **check\_regex** (*bool*) – Checks the name for a regular expression.
- **latest\_only** (*bool*) – If set to True it will only retrieve the first matched attachment.
- **not\_found\_mode** (*str*) – Specify what should happen if no attachment has been found. Supported values are ‘raise’, ‘warn’ and ‘ignore’. If it is set to ‘raise’ it will raise an exception, if set to ‘warn’ it will only print a warning and if set to ‘ignore’ it won’t notify you at all.

**Returns** a list of tuple each containing the attachment filename and its payload.

**Return type** a list of tuple

**class** airflow.contrib.hooks.jenkins\_hook.**JenkinsHook** (*conn\_id*=‘jenkins\_default’)

Bases: *airflow.hooks.base\_hook.BaseHook*

Hook to manage connection to jenkins server

**class** airflow.contrib.hooks.jira\_hook.**JiraHook** (*jira\_conn\_id*=‘jira\_default’, *proxies*=None)

Bases: *airflow.hooks.base\_hook.BaseHook*

Jira interaction hook, a Wrapper around JIRA Python SDK.

**Parameters** **jira\_conn\_id** (*str*) – reference to a pre-defined Jira Connection

**class** airflow.contrib.hooks.gcp\_mlengine\_hook.**MLEngineHook** (*gcp\_conn\_id*=‘google\_cloud\_default’, *delegate\_to*=None)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

**create\_job** (*project\_id*, *job*, *use\_existing\_job\_fn*=None)

Launches a MLEngine job and wait for it to reach a terminal state.

#### Parameters

- **project\_id** (*str*) – The Google Cloud project id within which MLEngine job will be launched.

- **job** (*dict*) – MLEngine Job object that should be provided to the MLEngine API, such as:

```
{
  'jobId': 'my_job_id',
  'trainingInput': {
    'scaleTier': 'STANDARD_1',
    ...
  }
}
```

- **use\_existing\_job\_fn** (*function*) – In case that a MLEngine job with the same `job_id` already exist, this method (if provided) will decide whether we should use this existing job, continue waiting for it to finish and returning the job object. It should accepts a MLEngine job object, and returns a boolean value indicating whether it is OK to reuse the existing job. If 'use\_existing\_job\_fn' is not provided, we by default reuse the existing MLEngine job.

**Returns** The MLEngine job object if the job successfully reach a terminal state (which might be FAILED or CANCELLED state).

**Return type** `dict`

**create\_model** (*project\_id, model*)

Create a Model. Blocks until finished.

**create\_version** (*project\_id, model\_name, version\_spec*)

Creates the Version on Google Cloud ML Engine.

Returns the operation if the version was created successfully and raises an error otherwise.

**delete\_version** (*project\_id, model\_name, version\_name*)

Deletes the given version of a model. Blocks until finished.

**get\_conn** ()

Returns a Google MLEngine service object.

**get\_model** (*project\_id, model\_name*)

Gets a Model. Blocks until finished.

**list\_versions** (*project\_id, model\_name*)

Lists all available versions of a model. Blocks until finished.

**set\_default\_version** (*project\_id, model\_name, version\_name*)

Sets a version to be the default. Blocks until finished.

**class** `airflow.contrib.hooks.mongo_hook.MongoHook` (*conn\_id='mongo\_default', \*args, \*\*kwargs*)

Bases: `airflow.hooks.base_hook.BaseHook`

PyMongo Wrapper to Interact With Mongo Database Mongo Connection Documentation <https://docs.mongodb.com/manual/reference/connection-string/index.html> You can specify connection string options in extra field of your connection <https://docs.mongodb.com/manual/reference/connection-string/index.html#connection-string-options> ex. {replicaSet: test, ssl: True, connectTimeoutMS: 30000}

**aggregate** (*mongo\_collection, aggregate\_query, mongo\_db=None, \*\*kwargs*)

Runs an aggregation pipeline and returns the results <https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.aggregate> <https://api.mongodb.com/python/current/examples/aggregation.html>

**delete\_many** (*mongo\_collection*, *filter\_doc*, *mongo\_db=None*, *\*\*kwargs*)

Deletes one or more documents in a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.delete\\_many](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.delete_many)

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to delete from.
- **filter\_doc** (*dict*) – A query that matches the documents to delete.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.

**delete\_one** (*mongo\_collection*, *filter\_doc*, *mongo\_db=None*, *\*\*kwargs*)

Deletes a single document in a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.delete\\_one](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.delete_one)

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to delete from.
- **filter\_doc** (*dict*) – A query that matches the document to delete.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.

**find** (*mongo\_collection*, *query*, *find\_one=False*, *mongo\_db=None*, *\*\*kwargs*)

Runs a mongo find query and returns the results <https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.find>

**get\_collection** (*mongo\_collection*, *mongo\_db=None*)

Fetches a mongo collection object for querying.

Uses connection schema as DB unless specified.

**get\_conn** ()

Fetches PyMongo Client

**insert\_many** (*mongo\_collection*, *docs*, *mongo\_db=None*, *\*\*kwargs*)

Inserts many docs into a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert\\_many](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_many)

**insert\_one** (*mongo\_collection*, *doc*, *mongo\_db=None*, *\*\*kwargs*)

Inserts a single document into a mongo collection [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert\\_one](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_one)

**replace\_many** (*mongo\_collection*, *docs*, *filter\_docs=None*, *mongo\_db=None*, *upsert=False*, *collation=None*, *\*\*kwargs*)

Replaces many documents in a mongo collection.

Uses bulk\_write with multiple ReplaceOne operations [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.bulk\\_write](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.bulk_write)

---

**Note:** If no `filter_docs` are given, it is assumed that all replacement documents contain the `__id` field which are then used as filters.

---

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to update.
- **docs** (*list[dict]*) – The new documents.

- **filter\_docs** (*list[dict]*) – A list of queries that match the documents to replace. Can be omitted; then the `_id` fields from docs will be used.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.
- **upsert** (*bool*) – If `True`, perform an insert if no documents match the filters for the replace operation.
- **collation** (*pymongo.collation.Collation*) – An instance of `Collation`. This option is only supported on MongoDB 3.4 and above.

**replace\_one** (*mongo\_collection, doc, filter\_doc=None, mongo\_db=None, \*\*kwargs*)

Replaces a single document in a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.replace\\_one](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.replace_one)

---

**Note:** If no `filter_doc` is given, it is assumed that the replacement document contain the `_id` field which is then used as filters.

---

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to update.
- **doc** (*dict*) – The new document.
- **filter\_doc** (*dict*) – A query that matches the documents to replace. Can be omitted; then the `_id` field from doc will be used.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.

**update\_many** (*mongo\_collection, filter\_doc, update\_doc, mongo\_db=None, \*\*kwargs*)

Updates one or more documents in a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.update\\_many](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.update_many)

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to update.
- **filter\_doc** (*dict*) – A query that matches the documents to update.
- **update\_doc** (*dict*) – The modifications to apply.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.

**update\_one** (*mongo\_collection, filter\_doc, update\_doc, mongo\_db=None, \*\*kwargs*)

Updates a single document in a mongo collection. [https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.update\\_one](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.update_one)

#### Parameters

- **mongo\_collection** (*str*) – The name of the collection to update.
- **filter\_doc** (*dict*) – A query that matches the documents to update.
- **update\_doc** (*dict*) – The modifications to apply.
- **mongo\_db** (*str*) – The name of the database to use. Can be omitted; then the database from the connection string is used.

```
class airflow.contrib.hooks.openfaas_hook.OpenFaasHook (function_name=None,
                                                         conn_id='open_faas_default',
                                                         *args, **kwargs)
```

Bases: `airflow.hooks.base_hook.BaseHook`

Interact with Openfaas to query, deploy, invoke and update function

#### Parameters

- **function\_name** – Name of the function, Defaults to None
- **conn\_id** (*str*) – openfaas connection to use, Defaults to open\_faas\_default for example  
host : <http://openfaas.faaS.com>, Conn Type : Http

```
class airflow.contrib.hooks.pinot_hook.PinotDbApiHook (*args, **kwargs)
```

Bases: `airflow.hooks.dbapi_hook.DbApiHook`

Connect to pinot db(<https://github.com/linkedin/pinot>) to issue pql

```
get_conn ()
```

Establish a connection to pinot broker through pinot dbapi.

```
get_first (sql)
```

Executes the sql and returns the first resulting row.

**Parameters** **sql** (*str* or *list*) – the sql statement to be executed (str) or a list of sql statements to execute

```
get_pandas_df (sql, parameters=None)
```

Executes the sql and returns a pandas dataframe

#### Parameters

- **sql** (*str* or *list*) – the sql statement to be executed (str) or a list of sql statements to execute
- **parameters** (*mapping* or *iterable*) – The parameters to render the SQL query with.

```
get_records (sql)
```

Executes the sql and returns a set of records.

**Parameters** **sql** (*str*) – the sql statement to be executed (str) or a list of sql statements to execute

```
get_uri ()
```

Get the connection uri for pinot broker.

e.g: <http://localhost:9000/pql>

```
insert_rows (table, rows, target_fields=None, commit_every=1000)
```

A generic way to insert a set of tuples into a table, a new transaction is created every commit\_every rows

#### Parameters

- **table** (*str*) – Name of the target table
- **rows** (*iterable of tuples*) – The rows to insert into the table
- **target\_fields** (*iterable of strings*) – The names of the columns to fill in the table
- **commit\_every** (*int*) – The maximum number of rows to insert in one transaction. Set to 0 to insert all rows in one transaction.
- **replace** (*bool*) – Whether to replace instead of insert

**set\_autocommit** (*conn, autocommit*)

Sets the autocommit flag on the connection

**class** airflow.contrib.hooks.gcp\_pubsub\_hook.**PubSubHook** (*gcp\_conn\_id='google\_cloud\_default', delegate\_to=None*)

Bases: *airflow.contrib.hooks.gcp\_api\_base\_hook.GoogleCloudBaseHook*

Hook for accessing Google Pub/Sub.

The GCP project against which actions are applied is determined by the project embedded in the Connection referenced by *gcp\_conn\_id*.

**acknowledge** (*project, subscription, ack\_ids*)

Pulls up to *max\_messages* messages from Pub/Sub subscription.

#### Parameters

- **project** (*str*) – the GCP project name or ID in which to create the topic
- **subscription** (*str*) – the Pub/Sub subscription name to delete; do not include the 'projects/{project}/topics/' prefix.
- **ack\_ids** (*list*) – List of ReceivedMessage ackIds from a previous pull response

**create\_subscription** (*topic\_project, topic, subscription=None, subscription\_project=None, ack\_deadline\_secs=10, fail\_if\_exists=False*)

Creates a Pub/Sub subscription, if it does not already exist.

#### Parameters

- **topic\_project** (*str*) – the GCP project ID of the topic that the subscription will be bound to.
- **topic** (*str*) – the Pub/Sub topic name that the subscription will be bound to create; do not include the projects/{project}/subscriptions/ prefix.
- **subscription** (*str*) – the Pub/Sub subscription name. If empty, a random name will be generated using the uuid module
- **subscription\_project** (*str*) – the GCP project ID where the subscription will be created. If unspecified, *topic\_project* will be used.
- **ack\_deadline\_secs** (*int*) – Number of seconds that a subscriber has to acknowledge each message pulled from the subscription
- **fail\_if\_exists** (*bool*) – if set, raise an exception if the topic already exists

**Returns** subscription name which will be the system-generated value if the subscription parameter is not supplied

**Return type** *str*

**create\_topic** (*project, topic, fail\_if\_exists=False*)

Creates a Pub/Sub topic, if it does not already exist.

#### Parameters

- **project** (*str*) – the GCP project ID in which to create the topic
- **topic** (*str*) – the Pub/Sub topic name to create; do not include the projects/{project}/topics/ prefix.
- **fail\_if\_exists** (*bool*) – if set, raise an exception if the topic already exists

**delete\_subscription** (*project, subscription, fail\_if\_not\_exists=False*)

Deletes a Pub/Sub subscription, if it exists.

**Parameters**

- **project** (*str*) – the GCP project ID where the subscription exists
- **subscription** (*str*) – the Pub/Sub subscription name to delete; do not include the `projects/{project}/subscriptions/` prefix.
- **fail\_if\_not\_exists** (*bool*) – if set, raise an exception if the topic does not exist

**delete\_topic** (*project, topic, fail\_if\_not\_exists=False*)

Deletes a Pub/Sub topic if it exists.

**Parameters**

- **project** (*str*) – the GCP project ID in which to delete the topic
- **topic** (*str*) – the Pub/Sub topic name to delete; do not include the `projects/{project}/topics/` prefix.
- **fail\_if\_not\_exists** (*bool*) – if set, raise an exception if the topic does not exist

**get\_conn** ()

Returns a Pub/Sub service object.

**Return type** `googleapiclient.discovery.Resource`

**publish** (*project, topic, messages*)

Publishes messages to a Pub/Sub topic.

**Parameters**

- **project** (*str*) – the GCP project ID in which to publish
- **topic** (*str*) – the Pub/Sub topic to which to publish; do not include the `projects/{project}/topics/` prefix.
- **messages** (list of PubSub messages; see <http://cloud.google.com/pubsub/docs/reference/rest/v1/PubsubMessage>) – messages to publish; if the data field in a message is set, it should already be base64 encoded.

**pull** (*project, subscription, max\_messages, return\_immediately=False*)

Pulls up to `max_messages` messages from Pub/Sub subscription.

**Parameters**

- **project** (*str*) – the GCP project ID where the subscription exists
- **subscription** (*str*) – the Pub/Sub subscription name to pull from; do not include the `projects/{project}/topics/` prefix.
- **max\_messages** (*int*) – The maximum number of messages to return from the Pub/Sub API.
- **return\_immediately** (*bool*) – If set, the Pub/Sub API will immediately return if no messages are available. Otherwise, the request will block for an undisclosed, but bounded period of time

**Returns** A list of Pub/Sub `ReceivedMessage` objects each containing an `ackId` property and a `message` property, which includes the base64-encoded message content. See <https://cloud.google.com/pubsub/docs/reference/rest/v1/projects.subscriptions/pull#ReceivedMessage>

**class** `airflow.contrib.hooks.qubole_check_hook.QuboleCheckHook` (*context, \*args, \*\*kwargs*)

Bases: `airflow.contrib.hooks.qubole_hook.QuboleHook`



```
class airflow.contrib.hooks.qubole_hook.QuboleHook(*args, **kwargs)
    Bases: airflow.hooks.base_hook.BaseHook

    get_jobs_id(ti)
        Get jobs associated with a Qubole commands :param ti: Task Instance of the dag, used to determine the
        Quboles command id :return: Job informations associated with command

    get_log(ti)
        Get Logs of a command from Qubole :param ti: Task Instance of the dag, used to determine the Quboles
        command id :return: command log as text

    get_results(ti=None, fp=None, inline=True, delim=None, fetch=True)
        Get results (or just s3 locations) of a command from Qubole and save into a file :param ti: Task Instance of
        the dag, used to determine the Quboles command id :param fp: Optional file pointer, will create one and
        return if None passed :param inline: True to download actual results, False to get s3 locations only :param
        delim: Replaces the CTL-A chars with the given delim, defaults to ';' :param fetch: when inline is True,
        get results directly from s3 (if large) :return: file location containing actual results or s3 locations of results

    kill(ti)
        Kill (cancel) a Qubole command :param ti: Task Instance of the dag, used to determine the Quboles
        command id :return: response from Qubole
```

```
class airflow.contrib.hooks.redis_hook.RedisHook(redis_conn_id='redis_default')
    Bases: airflow.hooks.base_hook.BaseHook

    Wrapper for connection to interact with Redis in-memory data structure store

    get_conn()
        Returns a Redis connection.
```

```
class airflow.contrib.hooks.redshift_hook.RedshiftHook(aws_conn_id='aws_default',
                                                         verify=None)
    Bases: airflow.contrib.hooks.aws_hook.AwsHook

    Interact with AWS Redshift, using the boto3 library

    cluster_status(cluster_identifier)
        Return status of a cluster

        Parameters cluster_identifier (str) – unique identifier of a cluster

    create_cluster_snapshot(snapshot_identifier, cluster_identifier)
        Creates a snapshot of a cluster

        Parameters

        • snapshot_identifier (str) – unique identifier for a snapshot of a cluster

        • cluster_identifier (str) – unique identifier of a cluster

    delete_cluster(cluster_identifier, skip_final_cluster_snapshot=True, final_cluster_snapshot_identifier=")
        Delete a cluster and optionally create a snapshot

        Parameters

        • cluster_identifier (str) – unique identifier of a cluster

        • skip_final_cluster_snapshot (bool) – determines cluster snapshot creation

        • final_cluster_snapshot_identifier (str) – name of final cluster snapshot

    describe_cluster_snapshots(cluster_identifier)
        Gets a list of snapshots for a cluster
```

Parameters **cluster\_identifier** (*str*) – unique identifier of a cluster

**restore\_from\_cluster\_snapshot** (*cluster\_identifier, snapshot\_identifier*)

Restores a cluster from its snapshot

#### Parameters

- **cluster\_identifier** (*str*) – unique identifier of a cluster
- **snapshot\_identifier** (*str*) – unique identifier for a snapshot of a cluster

**class** airflow.contrib.hooks.sftp\_hook.**SFTPHook** (*ftp\_conn\_id='sftp\_default', \*args, \*\*kwargs*)

Bases: [airflow.contrib.hooks.ssh\\_hook.SSHHook](#)

This hook is inherited from SSH hook. Please refer to SSH hook for the input arguments.

Interact with SFTP. Aims to be interchangeable with FTPHook.

:Pitfalls:

- In contrast **with** FTPHook `describe_directory` only returns size, type **and** modify. It doesn't return `unix.owner, unix.mode, perm, unix.group` and `unique`.
- `retrieve_file` **and** `store_file` only take a local full path **and not** a buffer.
- If no mode **is** passed to `create_directory` it will be created **with** `777` permissions.

Errors that may occur throughout but should be handled downstream.

**close\_conn** ()

Closes the connection. An error will occur if the connection wasnt ever opened.

**create\_directory** (*path, mode=777*)

Creates a directory on the remote system. :param path: full path to the remote directory to create :type path: str :param mode: int representation of octal mode for directory

**delete\_directory** (*path*)

Deletes a directory on the remote system. :param path: full path to the remote directory to delete :type path: str

**delete\_file** (*path*)

Removes a file on the FTP Server :param path: full path to the remote file :type path: str

**describe\_directory** (*path*)

Returns a dictionary of {filename: {attributes}} for all files on the remote system (where the MLSD command is supported). :param path: full path to the remote directory :type path: str

**get\_conn** ()

Returns an SFTP connection object

**list\_directory** (*path*)

Returns a list of files on the remote system. :param path: full path to the remote directory to list :type path: str

**retrieve\_file** (*remote\_full\_path, local\_full\_path*)

Transfers the remote file to a local location. If `local_full_path` is a string path, the file will be put at that location :param remote\_full\_path: full path to the remote file :type remote\_full\_path: str :param local\_full\_path: full path to the local file :type local\_full\_path: str

**store\_file** (*remote\_full\_path, local\_full\_path*)

Transfers a local file to the remote location. If `local_full_path_or_buffer` is a string path, the file will be

read from that location :param remote\_full\_path: full path to the remote file :type remote\_full\_path: str  
 :param local\_full\_path: full path to the local file :type local\_full\_path: str

```
class airflow.contrib.hooks.ssh_hook.SSHHook (ssh_conn_id=None,    remote_host=None,
                                             username=None,      password=None,
                                             key_file=None,    port=None,    timeout=10,
                                             keepalive_interval=30)
```

Bases: `airflow.hooks.base_hook.BaseHook`

Hook for ssh remote execution using Paramiko. ref: <https://github.com/paramiko/paramiko> This hook also lets you create ssh tunnel and serve as basis for SFTP file transfer

#### Parameters

- **ssh\_conn\_id** (*str*) – connection id from airflow Connections from where all the required parameters can be fetched like username, password or key\_file. Thought the priority is given to the param passed during init
- **remote\_host** (*str*) – remote host to connect
- **username** (*str*) – username to connect to the remote\_host
- **password** (*str*) – password of the username to connect to the remote\_host
- **key\_file** (*str*) – key file to use to connect to the remote\_host.
- **port** (*int*) – port of remote host to connect (Default is paramiko SSH\_PORT)
- **timeout** (*int*) – timeout for the attempt to connect to the remote\_host.
- **keepalive\_interval** (*int*) – send a keepalive packet to remote host every keepalive\_interval seconds

**get\_conn** ()

Opens a ssh connection to the remote host.

**Return type** paramiko.client.SSHClient

**get\_tunnel** (remote\_port, remote\_host='localhost', local\_port=None)

Creates a tunnel between two hosts. Like ssh -L <LOCAL\_PORT>:host:<REMOTE\_PORT>.

#### Parameters

- **remote\_port** (*int*) – The remote port to create a tunnel to
- **remote\_host** (*str*) – The remote host to create a tunnel to (default localhost)
- **local\_port** (*int*) – The local port to attach the tunnel to

**Returns** sshtunnel.SSHTunnelForwarder object

```
class airflow.contrib.hooks.sagemaker_hook.SageMakerHook (*args, **kwargs)
```

Bases: `airflow.contrib.hooks.aws_hook.AwsHook`

Interact with Amazon SageMaker.

**check\_s3\_url** (s3url)

Check if an S3 URL exists

**Parameters** s3url (*str*) – S3 url

**Return type** bool

**check\_status** (job\_name, key, describe\_function, check\_interval, max\_ingestion\_time, non\_terminal\_states=None)

Check status of a SageMaker job

**Parameters**

- **job\_name** (*str*) – name of the job to check status
- **key** (*str*) – the key of the response dict that points to the state
- **describe\_function** (*python callable*) – the function used to retrieve the status
- **args** – the arguments for the function
- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.
- **non\_terminal\_states** (*set*) – the set of nonterminal states

**Returns** response of describe call after job is done

**check\_training\_config** (*training\_config*)

Check if a training configuration is valid

**Parameters** **training\_config** (*dict*) – training\_config

**Returns** None

**check\_training\_status\_with\_log** (*job\_name*, *non\_terminal\_states*, *failed\_states*,  
*wait\_for\_completion*, *check\_interval*,  
*max\_ingestion\_time*)

Display the logs for a given training job, optionally tailing them until the job is complete.

**Parameters**

- **job\_name** (*str*) – name of the training job to check status and display logs for
- **non\_terminal\_states** (*set*) – the set of non\_terminal states
- **failed\_states** (*set*) – the set of failed states
- **wait\_for\_completion** (*bool*) – Whether to keep looking for new log entries until the job completes
- **check\_interval** (*int*) – The interval in seconds between polling for new log entries and job completion
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** None

**check\_tuning\_config** (*tuning\_config*)

Check if a tuning configuration is valid

**Parameters** **tuning\_config** (*dict*) – tuning\_config

**Returns** None

**configure\_s3\_resources** (*config*)

Extract the S3 operations from the configuration and execute them.

**Parameters** **config** (*dict*) – config of SageMaker operation

**Return type** *dict*

**create\_endpoint** (*config*, *wait\_for\_completion=True*, *check\_interval=30*,  
*max\_ingestion\_time=None*)

Create an endpoint

#### Parameters

- **config** (*dict*) – the config for endpoint
- **wait\_for\_completion** (*bool*) – if the program should keep running until job finishes
- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** A response to endpoint creation

**create\_endpoint\_config** (*config*)

Create an endpoint config

**Parameters** **config** (*dict*) – the config for endpoint-config

**Returns** A response to endpoint config creation

**create\_model** (*config*)

Create a model job

**Parameters** **config** (*dict*) – the config for model

**Returns** A response to model creation

**create\_training\_job** (*config*, *wait\_for\_completion=True*, *print\_log=True*, *check\_interval=30*,  
*max\_ingestion\_time=None*)

Create a training job

#### Parameters

- **config** (*dict*) – the config for training
- **wait\_for\_completion** (*bool*) – if the program should keep running until job finishes
- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** A response to training job creation

**create\_transform\_job** (*config*, *wait\_for\_completion=True*, *check\_interval=30*,  
*max\_ingestion\_time=None*)

Create a transform job

#### Parameters

- **config** (*dict*) – the config for transform job
- **wait\_for\_completion** (*bool*) – if the program should keep running until job finishes

- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** A response to transform job creation

**create\_tuning\_job** (*config*, *wait\_for\_completion=True*, *check\_interval=30*,  
*max\_ingestion\_time=None*)

Create a tuning job

**Parameters**

- **config** (*dict*) – the config for tuning
- **wait\_for\_completion** (*bool*) – if the program should keep running until job finishes
- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** A response to tuning job creation

**describe\_endpoint** (*name*)

**Parameters** **name** (*string*) – the name of the endpoint

**Returns** A dict contains all the endpoint info

**describe\_endpoint\_config** (*name*)

Return the endpoint config info associated with the name

**Parameters** **name** (*string*) – the name of the endpoint config

**Returns** A dict contains all the endpoint config info

**describe\_model** (*name*)

Return the SageMaker model info associated with the name

**Parameters** **name** (*string*) – the name of the SageMaker model

**Returns** A dict contains all the model info

**describe\_training\_job** (*name*)

Return the training job info associated with the name

**Parameters** **name** (*str*) – the name of the training job

**Returns** A dict contains all the training job info

**describe\_training\_job\_with\_log** (*job\_name*, *positions*, *stream\_names*, *instance\_count*, *state*,  
*last\_description*, *last\_describe\_job\_call*)

Return the training job info associated with job\_name and print CloudWatch logs

**describe\_transform\_job** (*name*)

Return the transform job info associated with the name

**Parameters** **name** (*string*) – the name of the transform job

**Returns** A dict contains all the transform job info

**describe\_tuning\_job** (*name*)

Return the tuning job info associated with the name

**Parameters** **name** (*string*) – the name of the tuning job

**Returns** A dict contains all the tuning job info

**get\_conn** ()

Establish an AWS connection for SageMaker

**Return type** `SageMaker.Client`

**get\_log\_conn** ()

Establish an AWS connection for retrieving logs during training

**Return type** `CloudWatchLogs.Client`

**log\_stream** (*log\_group*, *stream\_name*, *start\_time=0*, *skip=0*)

A generator for log items in a single stream. This will yield all the items that are available at the current moment.

**Parameters**

- **log\_group** (*str*) – The name of the log group.
- **stream\_name** (*str*) – The name of the specific stream.
- **start\_time** (*int*) – The time stamp value to start reading the logs from (default: 0).
- **skip** (*int*) – The number of log entries to skip at the start (default: 0). This is for when there are multiple entries at the same timestamp.

**Return type** `dict`

**Returns**

A CloudWatch log event with the following key-value pairs:

‘timestamp’ (*int*): The time in milliseconds of the event.

‘message’ (*str*): The log event data.

‘ingestionTime’ (*int*): The time in milliseconds the event was ingested.

**multi\_stream\_iter** (*log\_group*, *streams*, *positions=None*)

Iterate over the available events coming from a set of log streams in a single log group interleaving the events from each stream so they’re yielded in timestamp order.

**Parameters**

- **log\_group** (*str*) – The name of the log group.
- **streams** (*list*) – A list of the log stream names. The position of the stream in this list is the stream number.
- **positions** (*list*) – A list of pairs of (timestamp, skip) which represents the last record read from each stream.

**Returns** A tuple of (stream number, cloudwatch log event).

**tar\_and\_s3\_upload** (*path*, *key*, *bucket*)

Tar the local file or directory and upload to s3

**Parameters**

- **path** (*str*) – local file or directory
- **key** (*str*) – s3 key
- **bucket** (*str*) – s3 bucket

**Returns** None

**update\_endpoint** (*config*, *wait\_for\_completion=True*, *check\_interval=30*,  
*max\_ingestion\_time=None*)

Update an endpoint

**Parameters**

- **config** (*dict*) – the config for endpoint
- **wait\_for\_completion** (*bool*) – if the program should keep running until job finishes
- **check\_interval** (*int*) – the time interval in seconds which the operator will check the status of any SageMaker job
- **max\_ingestion\_time** (*int*) – the maximum ingestion time in seconds. Any SageMaker jobs that run longer than this will fail. Setting this to None implies no timeout for any SageMaker job.

**Returns** A response to endpoint update

**class** airflow.contrib.hooks.salesforce\_hook.**SalesforceHook** (*conn\_id*, *\*args*,  
*\*\*kwargs*)

Bases: *airflow.hooks.base\_hook.BaseHook*

**describe\_object** (*obj*)

Get the description of an object from Salesforce.

This description is the object's schema and some extra metadata that Salesforce stores for each object

**Parameters** *obj* – Name of the Salesforce object that we are getting a description of.

**get\_available\_fields** (*obj*)

Get a list of all available fields for an object.

This only returns the names of the fields.

**get\_object\_from\_salesforce** (*obj*, *fields*)

Get all instances of the *object* from Salesforce. For each model, only get the fields specified in fields.

**All we really do underneath the hood is run:** SELECT <fields> FROM <obj>;

**make\_query** (*query*)

Make a query to Salesforce. Returns result in dictionary

**Parameters** *query* – The query to make to Salesforce

**sign\_in** ()

Sign into Salesforce.

If we have already signed it, this will just return the original object

**write\_object\_to\_file** (*query\_results*, *filename*, *fmt='csv'*, *coerce\_to\_timestamp=False*,  
*record\_time\_added=False*)

Write query results to file.

**Acceptable formats are:**

- **csv:** comma-separated-values file. This is the default format.



- **json:** JSON array. Each element in the array is a different row.
- **ndjson:** JSON array but each element is new-line delimited instead of comma delimited like in *json*

This requires a significant amount of cleanup. Pandas doesn't handle output to CSV and json in a uniform way. This is especially painful for datetime types. Pandas wants to write them as strings in CSV, but as millisecond Unix timestamps.

By default, this function will try and leave all values as they are represented in Salesforce. You use the *coerce\_to\_timestamp* flag to force all datetimes to become Unix timestamps (UTC). This is can be greatly beneficial as it will make all of your datetime fields look the same, and makes it easier to work with in other database environments

#### Parameters

- **query\_results** – the results from a SQL query
- **filename** – the name of the file where the data should be dumped to
- **fmt** – the format you want the output in. *Default: csv.*
- **coerce\_to\_timestamp** – True if you want all datetime fields to be converted into Unix timestamps. False if you want them to be left in the same format as they were in Salesforce. Leaving the value as False will result in datetimes being strings. *Defaults to False*
- **record\_time\_added** – (*optional*) True if you want to add a Unix timestamp field to the resulting data that marks when the data was fetched from Salesforce. *Default: False.*

```
class airflow.contrib.hooks.segment_hook.SegmentHook(segment_conn_id='segment_default',
                                                    segment_debug_mode=False,
                                                    *args, **kwargs)
```

Bases: *airflow.hooks.base\_hook.BaseHook*

**on\_error** (*error, items*)

Handles error callbacks when using Segment with segment\_debug\_mode set to True

```
class airflow.contrib.hooks.slack_webhook_hook.SlackWebhookHook(http_conn_id=None,
                                                                web-
                                                                hook_token=None,
                                                                message="",
                                                                attach-
                                                                ments=None,
                                                                chan-
                                                                nel=None,
                                                                user-
                                                                name=None,
                                                                icon_emoji=None,
                                                                link_names=False,
                                                                proxy=None,
                                                                *args,
                                                                **kwargs)
```

Bases: *airflow.hooks.http\_hook.HttpHook*

This hook allows you to post messages to Slack using incoming webhooks. Takes both Slack webhook token directly and connection that has Slack webhook token. If both supplied, Slack webhook token will be used.

Each Slack webhook token can be pre-configured to use a specific channel, username and icon. You can override these defaults in this hook.

#### Parameters

- **http\_conn\_id** (*str*) – connection that has Slack webhook token in the extra field
- **webhook\_token** (*str*) – Slack webhook token
- **message** (*str*) – The message you want to send on Slack
- **attachments** (*list*) – The attachments to send on Slack. Should be a list of dictionaries representing Slack attachments.
- **channel** (*str*) – The channel the message should be posted to
- **username** (*str*) – The username to post to slack with
- **icon\_emoji** (*str*) – The emoji to use as icon for the user posting to Slack
- **link\_names** (*bool*) – Whether or not to find and link channel and usernames in your message
- **proxy** (*str*) – Proxy to use to make the Slack webhook call

**execute** ()

Remote Popen (actually execute the slack webhook call)

#### Parameters

- **cmd** – command to remotely execute
- **kwargs** – extra arguments to Popen (see subprocess.Popen)

**class** airflow.contrib.hooks.snowflake\_hook.**SnowflakeHook** (\*args, \*\*kwargs)

Bases: *airflow.hooks.dbapi\_hook.DbApiHook*

Interact with Snowflake.

get\_sqlalchemy\_engine() depends on snowflake-sqlalchemy

**get\_conn** ()

Returns a snowflake.connection object

**get\_uri** ()

override DbApiHook get\_uri method for get\_sqlalchemy\_engine()

**set\_autocommit** (conn, autocommit)

Sets the autocommit flag on the connection

```

class airflow.contrib.hooks.spark_jdbc_hook.SparkJDBCHook (spark_app_name='airflow-
spark-jdbc',
spark_conn_id='spark-
default',
spark_conf=None,
spark_py_files=None,
spark_files=None,
spark_jars=None,
num_executors=None,
executor_cores=None,
execu-
tor_memory=None,
driver_memory=None,
verbose=False,
principal=None,
keytab=None,
cmd_type='spark_to_jdbc',
jdbc_table=None,
jdbc_conn_id='jdbc-
default',
jdbc_driver=None,
metastore_table=None,
jdbc_truncate=False,
save_mode=None,
save_format=None,
batch_size=None,
fetch_size=None,
num_partitions=None,
parti-
tion_column=None,
lower_bound=None,
upper_bound=None,
cre-
ate_table_column_types=None,
*args, **kwargs)

```

Bases: `airflow.contrib.hooks.spark_submit_hook.SparkSubmitHook`

This hook extends the SparkSubmitHook specifically for performing data transfers to/from JDBC-based databases with Apache Spark.

#### Parameters

- **spark\_app\_name** (*str*) – Name of the job (default airflow-spark-jdbc)
- **spark\_conn\_id** (*str*) – Connection id as configured in Airflow administration
- **spark\_conf** (*dict*) – Any additional Spark configuration properties
- **spark\_py\_files** (*str*) – Additional python files used (.zip, .egg, or .py)
- **spark\_files** (*str*) – Additional files to upload to the container running the job
- **spark\_jars** (*str*) – Additional jars to upload and add to the driver and executor class-path
- **num\_executors** (*int*) – number of executor to run. This should be set so as to manage the number of connections made with the JDBC database
- **executor\_cores** (*int*) – Number of cores per executor

- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G)
- **driver\_memory** (*str*) – Memory allocated to the driver (e.g. 1000M, 2G)
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-submit for debugging
- **keytab** (*str*) – Full path to the file that contains the keytab
- **principal** (*str*) – The name of the kerberos principal used for keytab
- **cmd\_type** (*str*) – Which way the data should flow. 2 possible values: spark\_to\_jdbc: data written by spark from metastore to jdbc jdbc\_to\_spark: data written by spark from jdbc to metastore
- **jdbc\_table** (*str*) – The name of the JDBC table
- **jdbc\_conn\_id** (*str*) – Connection id used for connection to JDBC database
- **jdbc\_driver** (*str*) – Name of the JDBC driver to use for the JDBC connection. This driver (usually a jar) should be passed in the ‘jars’ parameter
- **metastore\_table** (*str*) – The name of the metastore table,
- **jdbc\_truncate** (*bool*) – (spark\_to\_jdbc only) Whether or not Spark should truncate or drop and recreate the JDBC table. This only takes effect if ‘save\_mode’ is set to Overwrite. Also, if the schema is different, Spark cannot truncate, and will drop and recreate
- **save\_mode** (*str*) – The Spark save-mode to use (e.g. overwrite, append, etc.)
- **save\_format** (*str*) – (jdbc\_to\_spark-only) The Spark save-format to use (e.g. parquet)
- **batch\_size** (*int*) – (spark\_to\_jdbc only) The size of the batch to insert per round trip to the JDBC database. Defaults to 1000
- **fetch\_size** (*int*) – (jdbc\_to\_spark only) The size of the batch to fetch per round trip from the JDBC database. Default depends on the JDBC driver
- **num\_partitions** (*int*) – The maximum number of partitions that can be used by Spark simultaneously, both for spark\_to\_jdbc and jdbc\_to\_spark operations. This will also cap the number of JDBC connections that can be opened
- **partition\_column** (*str*) – (jdbc\_to\_spark-only) A numeric column to be used to partition the metastore table by. If specified, you must also specify: num\_partitions, lower\_bound, upper\_bound
- **lower\_bound** (*int*) – (jdbc\_to\_spark-only) Lower bound of the range of the numeric partition column to fetch. If specified, you must also specify: num\_partitions, partition\_column, upper\_bound
- **upper\_bound** (*int*) – (jdbc\_to\_spark-only) Upper bound of the range of the numeric partition column to fetch. If specified, you must also specify: num\_partitions, partition\_column, lower\_bound
- **create\_table\_column\_types** – (spark\_to\_jdbc-only) The database column data types to use instead of the defaults, when creating the table. Data type information should be specified in the same format as CREATE TABLE columns syntax (e.g: “name CHAR(64), comments VARCHAR(1024)”). The specified types should be valid spark sql data types.

```
class airflow.contrib.hooks.spark_sql_hook.SparkSqlHook (sql, conf=None,
                                                         conn_id='spark_sql_default',
                                                         to-
                                                         tal_executor_cores=None,
                                                         executor_cores=None,
                                                         executor_memory=None,
                                                         keytab=None,      princi-
                                                         pal=None, master='yarn',
                                                         name='default-name',
                                                         num_executors=None,
                                                         verbose=True,
                                                         yarn_queue='default')
```

Bases: `airflow.hooks.base_hook.BaseHook`

This hook is a wrapper around the spark-sql binary. It requires that the “spark-sql” binary is in the PATH.

#### Parameters

- **sql** (*str*) – The SQL query to execute
- **conf** (*str* (*format*: *PROP=VALUE*)) – arbitrary Spark configuration property
- **conn\_id** (*str*) – connection\_id string
- **total\_executor\_cores** (*int*) – (Standalone & Mesos only) Total cores for all ex-  
ecutors (Default: all the available cores on the worker)
- **executor\_cores** (*int*) – (Standalone & YARN only) Number of cores per executor  
(Default: 2)
- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G) (Default: 1G)
- **keytab** (*str*) – Full path to the file that contains the keytab
- **master** (*str*) – spark://host:port, mesos://host:port, yarn, or local
- **name** (*str*) – Name of the job.
- **num\_executors** (*int*) – Number of executors to launch
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-sql
- **yarn\_queue** (*str*) – The YARN queue to submit to (Default: “default”)

**run\_query** (*cmd*=”, *\*\*kwargs*)

Remote Popen (actually execute the Spark-sql query)

#### Parameters

- **cmd** – command to remotely execute
- **kwargs** – extra arguments to Popen (see subprocess.Popen)

```

class airflow.contrib.hooks.spark_submit_hook.SparkSubmitHook (conf=None,
                                                                conn_id='spark_default',
                                                                files=None,
                                                                py_files=None,
                                                                archives=None,
                                                                driver_classpath=None,
                                                                jars=None,
                                                                java_class=None,
                                                                packages=None,
                                                                exclude_packages=None,
                                                                repositories=None,
                                                                total_executor_cores=None,
                                                                executor_cores=None,
                                                                executor_memory=None,
                                                                driver_memory=None,
                                                                keytab=None,
                                                                principal=None,
                                                                name='default-name',
                                                                num_executors=None,
                                                                application_args=None,
                                                                env_vars=None,
                                                                verbose=False,
                                                                spark_binary='spark-submit')

```

Bases: `airflow.hooks.base_hook.BaseHook`, `airflow.utils.log.logging_mixin.LoggingMixin`

This hook is a wrapper around the spark-submit binary to kick off a spark-submit job. It requires that the “spark-submit” binary is in the PATH or the spark\_home to be supplied.

#### Parameters

- **conf** (*dict*) – Arbitrary Spark configuration properties
- **conn\_id** (*str*) – The connection id as configured in Airflow administration. When an invalid connection\_id is supplied, it will default to yarn.
- **files** (*str*) – Upload additional files to the executor running the job, separated by a comma. Files will be placed in the working directory of each executor. For example, serialized objects.
- **py\_files** (*str*) – Additional python files used by the job, can be .zip, .egg or .py.
- **driver\_classpath** (*str*) – Additional, driver-specific, classpath settings.
- **jars** (*str*) – Submit additional jars to upload and place them in executor classpath.
- **java\_class** (*str*) – the main class of the Java application
- **packages** (*str*) – Comma-separated list of maven coordinates of jars to include on the driver and executor classpaths
- **exclude\_packages** (*str*) – Comma-separated list of maven coordinates of jars to exclude while resolving the dependencies provided in ‘packages’

- **repositories** (*str*) – Comma-separated list of additional remote repositories to search for the maven coordinates given with ‘packages’
- **total\_executor\_cores** (*int*) – (Standalone & Mesos only) Total cores for all executors (Default: all the available cores on the worker)
- **executor\_cores** (*int*) – (Standalone, YARN and Kubernetes only) Number of cores per executor (Default: 2)
- **executor\_memory** (*str*) – Memory per executor (e.g. 1000M, 2G) (Default: 1G)
- **driver\_memory** (*str*) – Memory allocated to the driver (e.g. 1000M, 2G) (Default: 1G)
- **keytab** (*str*) – Full path to the file that contains the keytab
- **principal** (*str*) – The name of the kerberos principal used for keytab
- **name** (*str*) – Name of the job (default airflow-spark)
- **num\_executors** (*int*) – Number of executors to launch
- **application\_args** (*list*) – Arguments for the application being submitted
- **env\_vars** (*dict*) – Environment variables for spark-submit. It supports yarn and k8s mode too.
- **verbose** (*bool*) – Whether to pass the verbose flag to spark-submit process for debugging
- **spark\_binary** (*str*) – The command to use for spark submit. Some distros may use spark2-submit.

**Param** archives: Archives that spark should unzip (and possibly tag with #ALIAS) into the application working directory.

**submit** (*application=*”, *\*\*kwargs*)

Remote Popen to execute the spark-submit job

#### Parameters

- **application** (*str*) – Submitted application, jar or py file
- **kwargs** – extra arguments to Popen (see subprocess.Popen)

```
class airflow.contrib.hooks.sqoop_hook.SqoopHook (conn_id='sqoop_default', verbose=False, num_mappers=None, hcatalog_database=None, hcatalog_table=None, properties=None)
```

Bases: *airflow.hooks.base\_hook.BaseHook*

This hook is a wrapper around the sqoop 1 binary. To be able to use the hook it is required that “sqoop” is in the PATH.

Additional arguments that can be passed via the ‘extra’ JSON field of the sqoop connection:

- **job\_tracker**: Job tracker local/jobtracker:port.
- **namenode**: Namenode.
- **lib\_jars**: Comma separated jar files to include in the classpath.
- **files**: Comma separated files to be copied to the map reduce cluster.
- **archives**: Comma separated archives to be unarchived on the compute machines.
- **password\_file**: Path to file containing the password.

#### Parameters

- **conn\_id** (*str*) – Reference to the sqoop connection.
- **verbose** (*bool*) – Set sqoop to verbose.
- **num\_mappers** (*int*) – Number of map tasks to import in parallel.
- **properties** (*dict*) – Properties to set via the -D argument

**Popen** (*cmd*, *\*\*kwargs*)

Remote Popen

#### Parameters

- **cmd** – command to remotely execute
- **kwargs** – extra arguments to Popen (see subprocess.Popen)

**Returns** handle to subprocess

**export\_table** (*table*, *export\_dir*, *input\_null\_string*, *input\_null\_non\_string*, *staging\_table*, *clear\_staging\_table*, *enclosed\_by*, *escaped\_by*, *input\_fields\_terminated\_by*, *input\_lines\_terminated\_by*, *input\_optionally\_enclosed\_by*, *batch*, *relaxed\_isolation*, *extra\_export\_options=None*)

Exports Hive table to remote location. Arguments are copies of direct sqoop command line Arguments

#### Parameters

- **table** – Table remote destination
- **export\_dir** – Hive table to export
- **input\_null\_string** – The string to be interpreted as null for string columns
- **input\_null\_non\_string** – The string to be interpreted as null for non-string columns
- **staging\_table** – The table in which data will be staged before being inserted into the destination table
- **clear\_staging\_table** – Indicate that any data present in the staging table can be deleted
- **enclosed\_by** – Sets a required field enclosing character
- **escaped\_by** – Sets the escape character
- **input\_fields\_terminated\_by** – Sets the field separator character
- **input\_lines\_terminated\_by** – Sets the end-of-line character
- **input\_optionally\_enclosed\_by** – Sets a field enclosing character
- **batch** – Use batch mode for underlying statement execution
- **relaxed\_isolation** – Transaction isolation to read uncommitted for the mappers
- **extra\_export\_options** – Extra export options to pass as dict. If a key doesn't have a value, just pass an empty string to it. Don't include prefix of - for sqoop options.

**import\_query** (*query*, *target\_dir*, *append=False*, *file\_type='text'*, *split\_by=None*, *direct=None*, *driver=None*, *extra\_import\_options=None*)

Imports a specific query from the rdbms to hdfs

#### Parameters

- **query** – Free format query to run
- **target\_dir** – HDFS destination dir



- **append** – Append data to an existing dataset in HDFS
- **file\_type** – “avro”, “sequence”, “text” or “parquet” Imports data to hdfs into the specified format. Defaults to text.
- **split\_by** – Column of the table used to split work units
- **direct** – Use direct import fast path
- **driver** – Manually specify JDBC driver class to use
- **extra\_import\_options** – Extra import options to pass as dict. If a key doesn’t have a value, just pass an empty string to it. Don’t include prefix of – for sqoop options.

**import\_table** (*table*, *target\_dir=None*, *append=False*, *file\_type='text'*, *columns=None*, *split\_by=None*, *where=None*, *direct=False*, *driver=None*, *extra\_import\_options=None*)

Imports table from remote location to target dir. Arguments are copies of direct sqoop command line arguments

#### Parameters

- **table** – Table to read
- **target\_dir** – HDFS destination dir
- **append** – Append data to an existing dataset in HDFS
- **file\_type** – “avro”, “sequence”, “text” or “parquet”. Imports data to into the specified format. Defaults to text.
- **columns** – <col,col,col...> Columns to import from table
- **split\_by** – Column of the table used to split work units
- **where** – WHERE clause to use during import
- **direct** – Use direct connector if exists for the database
- **driver** – Manually specify JDBC driver class to use
- **extra\_import\_options** – Extra import options to pass as dict. If a key doesn’t have a value, just pass an empty string to it. Don’t include prefix of – for sqoop options.

**class** airflow.contrib.hooks.vertica\_hook.**VerticaHook** (\*args, \*\*kwargs)

Bases: *airflow.hooks.dbapi\_hook.DbApiHook*

Interact with Vertica.

**get\_conn** ()

Returns verticaql connection object

**class** airflow.contrib.hooks.wasb\_hook.**WasbHook** (wasb\_conn\_id='wasb\_default')

Bases: *airflow.hooks.base\_hook.BaseHook*

Interacts with Azure Blob Storage through the wasb:// protocol.

Additional options passed in the ‘extra’ field of the connection will be passed to the *BlockBlobService()* constructor. For example, authenticate using a SAS token by adding {“sas\_token”: “YOUR\_TOKEN”}.

**Parameters** **wasb\_conn\_id** (*str*) – Reference to the wasb connection.

**check\_for\_blob** (*container\_name*, *blob\_name*, \*\*kwargs)

Check if a blob exists on Azure Blob Storage.

#### Parameters

- **container\_name** (*str*) – Name of the container.

- **blob\_name** (*str*) – Name of the blob.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.exists()* takes.

**Returns** True if the blob exists, False otherwise.

**Return type** `bool`

**check\_for\_prefix** (*container\_name*, *prefix*, *\*\*kwargs*)

Check if a prefix exists on Azure Blob storage.

**Parameters**

- **container\_name** (*str*) – Name of the container.
- **prefix** (*str*) – Prefix of the blob.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.list\_blobs()* takes.

**Returns** True if blobs matching the prefix exist, False otherwise.

**Return type** `bool`

**delete\_file** (*container\_name*, *blob\_name*, *is\_prefix=False*, *ignore\_if\_missing=False*, *\*\*kwargs*)

Delete a file from Azure Blob Storage.

**Parameters**

- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.
- **is\_prefix** (*bool*) – If blob\_name is a prefix, delete all matching files
- **ignore\_if\_missing** (*bool*) – if True, then return success even if the blob does not exist.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.create\_blob\_from\_path()* takes.

**get\_conn** ()

Return the BlockBlobService object.

**get\_file** (*file\_path*, *container\_name*, *blob\_name*, *\*\*kwargs*)

Download a file from Azure Blob Storage.

**Parameters**

- **file\_path** (*str*) – Path to the file to download.
- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.create\_blob\_from\_path()* takes.

**load\_file** (*file\_path*, *container\_name*, *blob\_name*, *\*\*kwargs*)

Upload a file to Azure Blob Storage.

**Parameters**

- **file\_path** (*str*) – Path to the file to load.
- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.

- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.create\_blob\_from\_path()* takes.

**load\_string** (*string\_data*, *container\_name*, *blob\_name*, *\*\*kwargs*)

Upload a string to Azure Blob Storage.

#### Parameters

- **string\_data** (*str*) – String to load.
- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.create\_blob\_from\_text()* takes.

**read\_file** (*container\_name*, *blob\_name*, *\*\*kwargs*)

Read a file from Azure Blob Storage and return as a string.

#### Parameters

- **container\_name** (*str*) – Name of the container.
- **blob\_name** (*str*) – Name of the blob.
- **kwargs** (*object*) – Optional keyword arguments that *BlockBlobService.create\_blob\_from\_path()* takes.

```
class airflow.contrib.hooks.winrm_hook.WinRMHook (ssh_conn_id=None, end-
                                                    point=None, remote_host=None,
                                                    remote_port=5985, trans-
                                                    port='plaintext', username=None,
                                                    password=None, service='HTTP',
                                                    keytab=None, ca_trust_path=None,
                                                    cert_pem=None,
                                                    cert_key_pem=None,
                                                    server_cert_validation='validate',
                                                    kerberos_delegation=False,
                                                    read_timeout_sec=30, oper-
                                                    ation_timeout_sec=20, ker-
                                                    beros_hostname_override=None,
                                                    message_encryption='auto',
                                                    credssp_disable_tlsv1_2=False,
                                                    send_cbt=True)
```

Bases: *airflow.hooks.base\_hook.BaseHook*

Hook for winrm remote execution using pywinrm.

**Seealso** <https://github.com/diyan/pywinrm/blob/master/winrm/protocol.py>

#### Parameters

- **ssh\_conn\_id** (*str*) – connection id from airflow Connections from where all the required parameters can be fetched like username and password. Though the priority is given to the param passed during init
- **endpoint** (*str*) – When set to *None*, endpoint will be constructed like this: `'http://{remote_host}:{remote_port}/wsman'`
- **remote\_host** (*str*) – Remote host to connect to. Ignored if *endpoint* is not *None*.
- **remote\_port** (*int*) – Remote port to connect to. Ignored if *endpoint* is not *None*.

- **transport** (*str*) – transport type, one of ‘plaintext’ (default), ‘kerberos’, ‘ssl’, ‘ntlm’, ‘credssp’
- **username** (*str*) – username to connect to the remote\_host
- **password** (*str*) – password of the username to connect to the remote\_host
- **service** (*str*) – the service name, default is HTTP
- **keytab** (*str*) – the path to a keytab file if you are using one
- **ca\_trust\_path** (*str*) – Certification Authority trust path
- **cert\_pem** (*str*) – client authentication certificate file path in PEM format
- **cert\_key\_pem** (*str*) – client authentication certificate key file path in PEM format
- **server\_cert\_validation** (*str*) – whether server certificate should be validated on Python versions that support it; one of ‘validate’ (default), ‘ignore’
- **kerberos\_delegation** (*bool*) – if True, TGT is sent to target server to allow multiple hops
- **read\_timeout\_sec** (*int*) – maximum seconds to wait before an HTTP connect/read times out (default 30). This value should be slightly higher than operation\_timeout\_sec, as the server can block *at least* that long.
- **operation\_timeout\_sec** (*int*) – maximum allowed time in seconds for any single wsman HTTP operation (default 20). Note that operation timeouts while receiving output (the only wsman operation that should take any significant time, and where these timeouts are expected) will be silently retried indefinitely.
- **kerberos\_hostname\_override** (*str*) – the hostname to use for the kerberos exchange (defaults to the hostname in the endpoint URL)
- **message\_encryption\_enabled** (*bool*) – Will encrypt the WinRM messages if set to True and the transport auth supports message encryption (Default True).
- **credssp\_disable\_tlsv1\_2** (*bool*) – Whether to disable TLSv1.2 support and work with older protocols like TLSv1.0, default is False
- **send\_cbt** (*bool*) – Will send the channel bindings over a HTTPS channel (Default: True)

### 3.21.5 Executors

Executors are the mechanism by which task instances get run.

**class** airflow.executors.celery\_executor.CeleryExecutor

Bases: airflow.executors.base\_executor.BaseExecutor

CeleryExecutor is recommended for production use of Airflow. It allows distributing the execution of task instances to multiple worker nodes.

Celery is a simple, flexible and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system.

**end** (*synchronous=False*)

This method is called when the caller is done submitting job and wants to wait synchronously for the job submitted previously to be all done.

**start** ()

Executors may need to get things started. For example LocalExecutor starts N workers.

**sync()**

Sync will get called periodically by the heartbeat method. Executors should override this to perform gather statuses.

**class** airflow.executors.local\_executor.**LocalExecutor** (*parallelism=32*)

Bases: airflow.executors.base\_executor.BaseExecutor

LocalExecutor executes tasks locally in parallel. It uses the multiprocessing Python library and queues to parallelize the execution of tasks.

**end()**

This method is called when the caller is done submitting job and wants to wait synchronously for the job submitted previously to be all done.

**execute\_async** (*key, command, queue=None, executor\_config=None*)

This method will execute the command asynchronously.

**start()**

Executors may need to get things started. For example LocalExecutor starts N workers.

**sync()**

Sync will get called periodically by the heartbeat method. Executors should override this to perform gather statuses.

**class** airflow.executors.sequential\_executor.**SequentialExecutor**

Bases: airflow.executors.base\_executor.BaseExecutor

This executor will only run one task instance at a time, can be used for debugging. It is also the only executor that can be used with sqlite since sqlite doesn't support multiple connections.

Since we want airflow to work out of the box, it defaults to this SequentialExecutor alongside sqlite as you first install it.

**end()**

This method is called when the caller is done submitting job and wants to wait synchronously for the job submitted previously to be all done.

**execute\_async** (*key, command, queue=None, executor\_config=None*)

This method will execute the command asynchronously.

**sync()**

Sync will get called periodically by the heartbeat method. Executors should override this to perform gather statuses.

**class** airflow.executors.dask\_executor.**DaskExecutor** (*cluster\_address=None*)

Bases: airflow.executors.base\_executor.BaseExecutor

DaskExecutor submits tasks to a Dask Distributed cluster.

**end()**

This method is called when the caller is done submitting job and wants to wait synchronously for the job submitted previously to be all done.

**execute\_async** (*key, command, queue=None, executor\_config=None*)

This method will execute the command asynchronously.

**start()**

Executors may need to get things started. For example LocalExecutor starts N workers.

**sync()**

Sync will get called periodically by the heartbeat method. Executors should override this to perform gather statuses.

**terminate()**

This method is called when the daemon receives a SIGTERM

### 3.21.5.1 Community-contributed executors

**class** airflow.contrib.executors.kubernetes\_executor.**KubernetesExecutor**

Bases: airflow.executors.base\_executor.BaseExecutor, airflow.utils.log.logging\_mixin.LoggingMixin

**clear\_not\_launched\_queued\_tasks** (\*\*kwargs)

If the airflow scheduler restarts with pending “Queued” tasks, the tasks may or may not have been launched. Thus, on starting up the scheduler let’s check every “Queued” task to see if it has been launched (ie: if there is a corresponding pod on kubernetes)

If it has been launched then do nothing, otherwise reset the state to “None” so the task will be rescheduled

This will not be necessary in a future version of airflow in which there is proper support for State.LAUNCHED

**end()**

This method is called when the caller is done submitting job and wants to wait synchronously for the job submitted previously to be all done.

**execute\_async** (key, command, queue=None, executor\_config=None)

This method will execute the command asynchronously.

**start()**

Executors may need to get things started. For example LocalExecutor starts N workers.

**sync()**

Sync will get called periodically by the heartbeat method. Executors should override this to perform gather statuses.

---

## HTTP Routing Table

---

### /api

GET /api/experimental/dags/<DAG\_ID>/dag\_runs,  
134

GET /api/experimental/dags/<DAG\_ID>/dag\_runs/<string:execution\_date>/tasks/<TASK\_ID>,  
134

GET /api/experimental/dags/<DAG\_ID>/paused/<string:paused>,  
134

GET /api/experimental/dags/<DAG\_ID>/tasks/<TASK\_ID>,  
134

GET /api/experimental/dags/<string:dag\_id>/dag\_runs/<string:execution\_date>,  
134

GET /api/experimental/latest\_runs, 135

GET /api/experimental/pools, 135

GET /api/experimental/pools/<string:name>,  
135

GET /api/experimental/test, 134

POST /api/experimental/dags/<DAG\_ID>/dag\_runs,  
134

POST /api/experimental/pools, 135

DELETE /api/experimental/pools/<string:name>,  
135





### a

`airflow.macros`, [333](#)

`airflow.models`, [334](#)



## A

- `acknowledge()` (airflow.contrib.hooks.gcp\_pubsub\_hook.PubSubHook method), 407
- `add_task()` (airflow.models.DAG method), 339
- `add_tasks()` (airflow.models.DAG method), 339
- `AdlsToGoogleCloudStorageOperator` (class in airflow.contrib.operators.adls\_to\_gcs), 243
- `aggregate()` (airflow.contrib.hooks.mongo\_hook.MongoHook method), 403
- `airflow.macros` (module), 333
- `airflow.models` (module), 334
- `allocate_ids()` (airflow.contrib.hooks.datastore\_hook.DatastoreHook method), 387
- `are_dependencies_met()` (airflow.models.TaskInstance method), 346
- `are_dependents_done()` (airflow.models.TaskInstance method), 346
- `AthenaSensor` (class in airflow.contrib.sensors.aws\_athena\_sensor), 322
- `await()` (airflow.contrib.hooks.gcp\_dataproc\_hook.DataProcHook method), 385
- `AWSAthenaHook` (class in airflow.contrib.hooks.aws\_athena\_hook), 365
- `AWSAthenaOperator` (class in airflow.contrib.operators.aws\_athena\_operator), 243
- `AWSBatchOperator` (class in airflow.contrib.operators.awsbatch\_operator), 243
- `AwsDynamoDBHook` (class in airflow.contrib.hooks.aws\_dynamodb\_hook), 366
- `AwsFirehoseHook` (class in airflow.contrib.hooks.aws\_firehose\_hook), 366
- `AwsGlueCatalogHook` (class in airflow.contrib.hooks.aws\_glue\_catalog\_hook), 367
- `AwsGlueCatalogPartitionSensor` (class in airflow.contrib.sensors.aws\_glue\_catalog\_partition\_sensor), 322
- `AwsHook` (class in airflow.contrib.hooks.aws\_hook), 368
- `AwsLambdaHook` (class in airflow.contrib.hooks.aws\_lambda\_hook), 368
- `AwsRedshiftClusterSensor` (class in airflow.contrib.sensors.aws\_redshift\_cluster\_sensor), 323
- `AwsSnsHook` (class in airflow.contrib.hooks.aws\_sns\_hook), 368
- `AzureContainerInstanceHook` (class in airflow.contrib.hooks.azure\_container\_instance\_hook), 369
- `AzureContainerInstancesOperator` (class in airflow.contrib.operators.azure\_container\_instances\_operator), 244
- `AzureContainerRegistryHook` (class in airflow.contrib.hooks.azure\_container\_registry\_hook), 370
- `AzureContainerVolumeHook` (class in airflow.contrib.hooks.azure\_container\_volume\_hook), 370
- `AzureCosmosDBHook` (class in airflow.contrib.hooks.azure\_cosmos\_hook), 370
- `AzureCosmosDocumentSensor` (class in airflow.contrib.sensors.azure\_cosmos\_sensor), 323
- `AzureCosmosInsertDocumentOperator` (class in airflow.contrib.operators.azure\_cosmos\_operator), 245
- `AzureDataLakeHook` (class in airflow.contrib.hooks.azure\_data\_lake\_hook), 371
- `AzureDataLakeStorageListOperator` (class in airflow.contrib.operators.adls\_list\_operator), 371

245  
 AzureFileShareHook (class in *airflow.contrib.hooks.azure\_fileshare\_hook*), 372

## B

bag\_dag() (*airflow.models.DagBag* method), 343  
 BaseHook (class in *airflow.hooks.base\_hook*), 349  
 BaseOperator (class in *airflow.models*), 217, 334  
 BaseSensorOperator (class in *airflow.sensors.base\_sensor\_operator*), 221  
 BashOperator (class in *airflow.operators.bash\_operator*), 222  
 BashSensor (class in *airflow.contrib.sensors.bash\_sensor*), 323  
 begin\_transaction() (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* method), 387  
 BigQueryCheckOperator (class in *airflow.contrib.operators.bigquery\_check\_operator*), 246  
 BigQueryCreateEmptyDatasetOperator (class in *airflow.contrib.operators.bigquery\_operator*), 246  
 BigQueryCreateEmptyTableOperator (class in *airflow.contrib.operators.bigquery\_operator*), 247  
 BigQueryCreateExternalTableOperator (class in *airflow.contrib.operators.bigquery\_operator*), 248  
 BigQueryDeleteDatasetOperator (class in *airflow.contrib.operators.bigquery\_operator*), 249  
 BigQueryGetDataOperator (class in *airflow.contrib.operators.bigquery\_get\_data*), 249  
 BigQueryHook (class in *airflow.contrib.hooks.bigquery\_hook*), 374  
 BigQueryIntervalCheckOperator (class in *airflow.contrib.operators.bigquery\_check\_operator*), 250  
 BigQueryOperator (class in *airflow.contrib.operators.bigquery\_operator*), 251  
 BigQueryTableDeleteOperator (class in *airflow.contrib.operators.bigquery\_table\_delete\_operator*), 252  
 BigQueryTableSensor (class in *airflow.contrib.sensors.bigquery\_sensor*), 323  
 BigQueryToBigQueryOperator (class in *airflow.contrib.operators.bigquery\_to\_bigquery*), 252  
 BigQueryToCloudStorageOperator (class in

*airflow.contrib.operators.bigquery\_to\_gcs*), 252  
 BigQueryValueCheckOperator (class in *airflow.contrib.operators.bigquery\_check\_operator*), 253  
 BigtableClusterUpdateOperator (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 253  
 BigtableHook (class in *airflow.contrib.hooks.gcp\_bigtable\_hook*), 375  
 BigtableInstanceCreateOperator (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 253  
 BigtableInstanceDeleteOperator (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 254  
 BigtableTableCreateOperator (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 255  
 BigtableTableDeleteOperator (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 255  
 BigtableTableWaitForReplicationSensor (class in *airflow.contrib.operators.gcp\_bigtable\_operator*), 255  
 BranchPythonOperator (class in *airflow.operators.python\_operator*), 222  
 build\_job() (*airflow.contrib.operators.jenkins\_job\_trigger\_operator.JenkinsJobTriggerOperator* method), 295  
 bulk\_dump() (*airflow.hooks.dbapi\_hook.DbApiHook* method), 349  
 bulk\_dump() (*airflow.hooks.mysql\_hook.MySqlHook* method), 358  
 bulk\_dump() (*airflow.hooks.postgres\_hook.PostgresHook* method), 359  
 bulk\_insert\_rows() (*airflow.hooks.oracle\_hook.OracleHook* method), 359  
 bulk\_load() (*airflow.hooks.dbapi\_hook.DbApiHook* method), 349  
 bulk\_load() (*airflow.hooks.mysql\_hook.MySqlHook* method), 358  
 bulk\_load() (*airflow.hooks.postgres\_hook.PostgresHook* method), 359

## C

call() (*airflow.hooks.zendesk\_hook.ZendeskHook* method), 365  
 CassandraHook (class in *airflow.contrib.hooks.cassandra\_hook*), 377

CassandraRecordSensor (class in *airflow.contrib.sensors.cassandra\_record\_sensor*), 324  
 CassandraTableSensor (class in *airflow.contrib.sensors.cassandra\_table\_sensor*), 324  
 CassandraToGoogleCloudStorageOperator (class in *airflow.contrib.operators.cassandra\_to\_gcs*), 256  
 CeleryExecutor (class in *airflow.executors.celery\_executor*), 428  
 CeleryQueueSensor (class in *airflow.contrib.sensors.celery\_queue\_sensor*), 324  
 check\_for\_blob() (*airflow.contrib.hooks.wasb\_hook.WasbHook* method), 425  
 check\_for\_bucket() (*airflow.hooks.S3\_hook.S3Hook* method), 360  
 check\_for\_directory() (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 372  
 check\_for\_file() (*airflow.contrib.hooks.azure\_data\_lake\_hook.AzureDataLakeHook* method), 371  
 check\_for\_file() (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 372  
 check\_for\_key() (*airflow.hooks.S3\_hook.S3Hook* method), 360  
 check\_for\_named\_partition() (*airflow.hooks.hive\_hooks.HiveMetastoreHook* method), 353  
 check\_for\_partition() (*airflow.contrib.hooks.aws\_glue\_catalog\_hook.AwsGlueCatalogHook* method), 367  
 check\_for\_partition() (*airflow.hooks.hive\_hooks.HiveMetastoreHook* method), 354  
 check\_for\_path() (*airflow.hooks.webhdfs\_hook.WebHDFSHook* method), 364  
 check\_for\_prefix() (*airflow.contrib.hooks.wasb\_hook.WasbHook* method), 426  
 check\_for\_prefix() (*airflow.hooks.S3\_hook.S3Hook* method), 361  
 check\_for\_wildcard\_key() (*airflow.hooks.S3\_hook.S3Hook* method), 361  
 check\_query\_status() (*airflow.contrib.hooks.aws\_athena\_hook.AWSAthenaHook* method), 365  
 check\_response() (*airflow.hooks.http\_hook.HttpHook* method), 357  
 check\_s3\_url() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 411  
 check\_status() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 411  
 check\_training\_config() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 412  
 check\_training\_status\_with\_log() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 412  
 check\_tuning\_config() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 412  
 CheckOperator (class in *airflow.operators.check\_operator*), 223  
 cleanup\_database\_hook() (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* method), 382  
 clear() (*airflow.models.BaseOperator* method), 220  
 clear() (*airflow.models.DAG* method), 339  
 clear\_not\_launched\_queued\_tasks() (*airflow.contrib.executors.kubernetes\_executor.KubernetesExecutor* method), 430  
 clear\_task\_instances() (in module *airflow.models*), 349  
 clear\_xcom\_data() (*airflow.models.TaskInstance* method), 346  
 cli() (*airflow.models.DAG* method), 339  
 close\_conn() (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 389  
 close\_conn() (*airflow.contrib.hooks.sftp\_hook.SFTPHook* method), 410  
 closest\_ds\_partition() (in module *airflow.macros.hive*), 333  
 CloudantHook (class in *airflow.contrib.hooks.cloudant\_hook*), 385  
 CloudSpannerHook (class in *airflow.contrib.hooks.gcp\_spanner\_hook*), 378  
 CloudSpannerInstanceDatabaseDeleteOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 256  
 CloudSpannerInstanceDatabaseDeployOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 256  
 CloudSpannerInstanceDatabaseQueryOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 256

- 257
- CloudSpannerInstanceDatabaseUpdateOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 257
- CloudSpannerInstanceDeleteOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 258
- CloudSpannerInstanceDeployOperator (class in *airflow.contrib.operators.gcp\_spanner\_operator*), 258
- CloudSqlBaseOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 258
- CloudSqlDatabaseHook (class in *airflow.contrib.hooks.gcp\_sql\_hook*), 381
- CloudSqlHook (class in *airflow.contrib.hooks.gcp\_sql\_hook*), 382
- CloudSqlInstanceCreateOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 259
- CloudSqlInstanceDatabaseCreateOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 259
- CloudSqlInstanceDatabaseDeleteOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 259
- CloudSqlInstanceDatabasePatchOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 260
- CloudSqlInstanceDeleteOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 260
- CloudSqlInstanceExportOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 260
- CloudSqlInstanceImportOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 261
- CloudSqlInstancePatchOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 261
- CloudSqlQueryOperator (class in *airflow.contrib.operators.gcp\_sql\_operator*), 262
- cluster\_status() (*airflow.contrib.hooks.redshift\_hook.RedshiftHook* method), 409
- collect\_dags() (*airflow.models.DagBag* method), 343
- command() (*airflow.models.TaskInstance* method), 346
- command\_as\_list() (*airflow.models.TaskInstance* method), 346
- commit() (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* method), 387
- concurrency\_reached (*airflow.models.DAG* attribute), 339
- configure\_s3\_resources() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 412
- construct\_api\_call\_params() (*airflow.operators.slack\_operator.SlackAPIOperator* method), 237
- construct\_api\_call\_params() (*airflow.operators.slack\_operator.SlackAPIPostOperator* method), 238
- construct\_ingest\_query() (*airflow.operators.hive\_to\_druid.HiveToDruidTransfer* method), 227
- convert\_map\_type() (*airflow.contrib.operators.cassandra\_to\_gcs.CassandraToGoogleCloudStorageOperator* class method), 256
- convert\_tuple\_type() (*airflow.contrib.operators.cassandra\_to\_gcs.CassandraToGoogleCloudStorageOperator* class method), 256
- convert\_types() (*airflow.contrib.operators.postgres\_to\_gcs\_operator.PostgresToGoogleCloudStorageOperator* class method), 302
- convert\_user\_type() (*airflow.contrib.operators.cassandra\_to\_gcs.CassandraToGoogleCloudStorageOperator* class method), 256
- copy() (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 398
- copy\_expert() (*airflow.hooks.postgres\_hook.PostgresHook* method), 360
- copy\_object() (*airflow.hooks.S3\_hook.S3Hook* method), 361
- create\_bucket() (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 398
- create\_bucket() (*airflow.hooks.S3\_hook.S3Hook* method), 361
- create\_cluster() (*airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook* method), 391
- create\_cluster\_snapshot() (*airflow.contrib.hooks.redshift\_hook.RedshiftHook* method), 409
- create\_collection() (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 370
- create\_connection() (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* method), 346

[method](#)), 382  
[create\\_dagrun\(\)](#) ([airflow.models.DAG](#) [method](#)), 339  
[create\\_dagrun\(\)](#) ([airflow.models.DagModel](#) [method](#)), 344  
[create\\_database\(\)](#) ([air-flow.contrib.hooks.azure\\_cosmos\\_hook.AzureCosmosDBHook](#) [method](#)), 370  
[create\\_database\(\)](#) ([air-flow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook](#) [method](#)), 378  
[create\\_database\(\)](#) ([air-flow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook](#) [method](#)), 382  
[create\\_directory\(\)](#) ([air-flow.contrib.hooks.azure\\_fileshare\\_hook.AzureFileShareHook](#) [method](#)), 373  
[create\\_directory\(\)](#) ([air-flow.contrib.hooks.ftp\\_hook.FTPHook](#) [method](#)), 389  
[create\\_directory\(\)](#) ([air-flow.contrib.hooks.sftp\\_hook.SFTPHook](#) [method](#)), 410  
[create\\_endpoint\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 412  
[create\\_endpoint\\_config\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 413  
[create\\_instance\(\)](#) ([air-flow.contrib.hooks.gcp\\_bigtable\\_hook.BigtableHook](#) [method](#)), 375  
[create\\_instance\(\)](#) ([air-flow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook](#) [method](#)), 378  
[create\\_instance\(\)](#) ([air-flow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook](#) [method](#)), 383  
[create\\_job\(\)](#) ([airflow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook](#) [method](#)), 402  
[create\\_job\\_flow\(\)](#) ([air-flow.contrib.hooks.emr\\_hook.EmrHook](#) [method](#)), 389  
[create\\_model\(\)](#) ([air-flow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook](#) [method](#)), 403  
[create\\_model\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 413  
[create\\_new\\_function\(\)](#) ([air-flow.contrib.hooks.gcp\\_function\\_hook.GcfHook](#) [method](#)), 395  
[create\\_or\\_update\(\)](#) ([air-flow.contrib.hooks.azure\\_container\\_instance\\_hook.AzureContainerInstanceHook](#) [method](#)), 369  
[create\\_subscription\(\)](#) ([air-flow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 407  
[create\\_table\(\)](#) ([air-flow.contrib.hooks.gcp\\_bigtable\\_hook.BigtableHook](#) [method](#)), 376  
[create\\_topic\(\)](#) ([air-flow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 407  
[create\\_training\\_job\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 413  
[create\\_transform\\_job\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 413  
[create\\_tuning\\_job\(\)](#) ([air-flow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 414  
[create\\_version\(\)](#) ([air-flow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook](#) [method](#)), 403  
[current\\_state\(\)](#) ([airflow.models.TaskInstance](#) [method](#)), 346

## D

[dag](#) ([airflow.models.BaseOperator](#) [attribute](#)), 220, 336  
[DAG](#) (class in [airflow.models](#)), 338  
[DagBag](#) (class in [airflow.models](#)), 343  
[dagbag\\_report\(\)](#) ([airflow.models.DagBag](#) [method](#)), 343  
[DagModel](#) (class in [airflow.models](#)), 344  
[DagRun](#) (class in [airflow.models](#)), 344  
[DaskExecutor](#) (class in [air-flow.executors.dask\\_executor](#)), 429  
[DatabricksHook](#) (class in [air-flow.contrib.hooks.databricks\\_hook](#)), 385  
[DatabricksRunNowOperator](#) (class in [airflow.contrib.operators.databricks\\_operator](#)), 269  
[DatabricksSubmitRunOperator](#) (class in [air-flow.contrib.operators.databricks\\_operator](#)), 271  
[DatadogHook](#) (class in [air-flow.contrib.hooks.datadog\\_hook](#)), 386  
[DatadogSensor](#) (class in [air-flow.contrib.sensors.datadog\\_sensor](#)), 324  
[DataFlowHook](#) (class in [air-flow.contrib.hooks.gcp\\_dataflow\\_hook](#)), 385  
[DataFlowJavaOperator](#) (class in [air-flow.contrib.operators.dataflow\\_operator](#)), 262  
[DataFlowPythonOperator](#) (class in [air-flow.contrib.operators.dataflow\\_operator](#)), 262



- 264
- DataflowTemplateOperator (class in *airflow.contrib.operators.dataflow\_operator*), 273
- DataproClusterCreateOperator (class in *airflow.contrib.operators.dataproc\_operator*), 274
- DataproClusterDeleteOperator (class in *airflow.contrib.operators.dataproc\_operator*), 276
- DataproClusterScaleOperator (class in *airflow.contrib.operators.dataproc\_operator*), 276
- DataProcHadoopOperator (class in *airflow.contrib.operators.dataproc\_operator*), 265
- DataProcHiveOperator (class in *airflow.contrib.operators.dataproc\_operator*), 265
- DataProcHook (class in *airflow.contrib.hooks.gcp\_dataproc\_hook*), 385
- DataProcPigOperator (class in *airflow.contrib.operators.dataproc\_operator*), 266
- DataProcPySparkOperator (class in *airflow.contrib.operators.dataproc\_operator*), 267
- DataProcSparkOperator (class in *airflow.contrib.operators.dataproc\_operator*), 268
- DataProcSparkSqlOperator (class in *airflow.contrib.operators.dataproc\_operator*), 269
- DataproWorkflowTemplateBaseOperator (class in *airflow.contrib.operators.dataproc\_operator*), 277
- DataproWorkflowTemplateInstantiateInlineOperator (class in *airflow.contrib.operators.dataproc\_operator*), 277
- DataproWorkflowTemplateInstantiateOperator (class in *airflow.contrib.operators.dataproc\_operator*), 277
- DatastoreExportOperator (class in *airflow.contrib.operators.datastore\_export\_operator*), 278
- DatastoreHook (class in *airflow.contrib.hooks.datastore\_hook*), 387
- DatastoreImportOperator (class in *airflow.contrib.operators.datastore\_import\_operator*), 278
- DayOfWeekSensor (class in *airflow.contrib.sensors.weekday\_sensor*), 325
- db () (*airflow.contrib.hooks.cloudant\_hook.CloudantHook* method), 385
- DbApiHook (class in *airflow.hooks.dbapi\_hook*), 349
- deactivate\_stale\_dags () (*airflow.models.DAG* static method), 340
- deactivate\_unknown\_dags () (*airflow.models.DAG* static method), 340
- decrypt () (*airflow.contrib.hooks.gcp\_kms\_hook.GoogleCloudKMSHook* method), 397
- delete () (*airflow.contrib.hooks.azure\_container\_instance\_hook.AzureContainerInstanceHook* method), 369
- delete () (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 398
- delete\_cluster () (*airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook* method), 392
- delete\_cluster () (*airflow.contrib.hooks.redshift\_hook.RedshiftHook* method), 409
- delete\_collection () (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 370
- delete\_connection () (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* method), 382
- delete\_database () (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 370
- delete\_database () (*airflow.contrib.hooks.gcp\_spanner\_hook.CloudSpannerHook* method), 379
- delete\_database () (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlHook* method), 383
- delete\_directory () (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 390
- delete\_directory () (*airflow.contrib.hooks.sftp\_hook.SFTPHook* method), 410
- delete\_document () (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 370
- delete\_file () (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 390
- delete\_file () (*airflow.contrib.hooks.sftp\_hook.SFTPHook* method), 410
- delete\_file () (*airflow.contrib.hooks.wasb\_hook.WasbHook* method), 426
- delete\_function () (*airflow.contrib.hooks.gcp\_function\_hook.GcfHook* method), 395



[delete\\_instance\(\)](#) ([airflow.contrib.hooks.gcp\\_bigtable\\_hook.BigtableHook](#) [method](#)), 416  
[describe\\_training\\_job\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 376  
[delete\\_instance\(\)](#) ([airflow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook](#) [method](#)), 414  
[describe\\_training\\_job\\_with\\_log\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 379  
[delete\\_instance\(\)](#) ([airflow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook](#) [method](#)), 414  
[describe\\_transform\\_job\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 383  
[delete\\_many\(\)](#) ([airflow.contrib.hooks.mongo\\_hook.MongoHook](#) [method](#)), 414  
[describe\\_tuning\\_job\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 403  
[delete\\_objects\(\)](#) ([airflow.hooks.S3\\_hook.S3Hook](#) [method](#)), 414  
[DiscordWebhookHook](#) (class in [airflow.contrib.hooks.discord\\_webhook\\_hook](#)), 388  
[delete\\_one\(\)](#) ([airflow.contrib.hooks.mongo\\_hook.MongoHook](#) [method](#)), 404  
[DiscordWebhookOperator](#) (class in [airflow.contrib.operators.discord\\_webhook\\_operator](#)), 279  
[delete\\_operation\(\)](#) ([airflow.contrib.hooks.datastore\\_hook.DatastoreHook](#) [method](#)), 387  
[DockerHook](#) (class in [airflow.hooks.docker\\_hook](#)), 351  
[delete\\_subscription\(\)](#) ([airflow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 407  
[DockerOperator](#) (class in [airflow.operators.docker\\_operator](#)), 223  
[delete\\_table\(\)](#) ([airflow.contrib.hooks.gcp\\_bigtable\\_hook.BigtableHook](#) [method](#)), 376  
[does\\_collection\\_exist\(\)](#) ([airflow.contrib.hooks.azure\\_cosmos\\_hook.AzureCosmosDBHook](#) [method](#)), 370  
[delete\\_topic\(\)](#) ([airflow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 408  
[does\\_database\\_exist\(\)](#) ([airflow.contrib.hooks.azure\\_cosmos\\_hook.AzureCosmosDBHook](#) [method](#)), 370  
[delete\\_version\(\)](#) ([airflow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook](#) [method](#)), 403  
[download\(\)](#) ([airflow.contrib.hooks.gcs\\_hook.GoogleCloudStorageHook](#) [method](#)), 399  
[download\\_file\(\)](#) ([airflow.contrib.hooks.azure\\_data\\_lake\\_hook.AzureDataLakeHook](#) [method](#)), 371  
[deps](#) ([airflow.models.BaseOperator](#) attribute), 220, 337  
[download\\_mail\\_attachments\(\)](#) ([airflow.contrib.hooks.imap\\_hook.ImapHook](#) [method](#)), 401  
[describe\\_cluster\\_snapshots\(\)](#) ([airflow.contrib.hooks.redshift\\_hook.RedshiftHook](#) [method](#)), 409  
[downstream\\_list](#) ([airflow.models.BaseOperator](#) attribute), 220, 337  
[describe\\_directory\(\)](#) ([airflow.contrib.hooks.ftp\\_hook.FTPHook](#) [method](#)), 390  
[DruidCheckOperator](#) (class in [airflow.operators.druid\\_check\\_operator](#)), 224  
[describe\\_directory\(\)](#) ([airflow.contrib.hooks.sftp\\_hook.SFTPHook](#) [method](#)), 410  
[DruidDbApiHook](#) (class in [airflow.hooks.druid\\_hook](#)), 351  
[describe\\_endpoint\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 414  
[DruidHook](#) (class in [airflow.hooks.druid\\_hook](#)), 352  
[DruidOperator](#) (class in [airflow.operators.druid\\_operator](#)), 279  
[describe\\_endpoint\\_config\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 414  
[ds\\_add\(\)](#) (in module [airflow.macros](#)), 333  
[ds\\_format\(\)](#) (in module [airflow.macros](#)), 333  
[DummyOperator](#) (class in [airflow.operators.dummy\\_operator](#)), 225  
[describe\\_model\(\)](#) ([airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook](#) [method](#)), 414  
[ECSOperator](#) (class in [airflow.contrib.operators.ecs\\_operator](#)), 279  
[describe\\_object\(\)](#) ([airflow.contrib.hooks.salesforce\\_hook.SalesforceHook](#) [method](#)), 414

[EmailOperator](#) (class in [airflow.flow.operators.email\\_operator](#)), 225  
[EmrAddStepsOperator](#) (class in [airflow.contrib.operators.emr\\_add\\_steps\\_operator](#)), 280  
[EmrBaseSensor](#) (class in [airflow.contrib.sensors.emr\\_base\\_sensor](#)), 326  
[EmrCreateJobFlowOperator](#) (class in [airflow.contrib.operators.emr\\_create\\_job\\_flow\\_operator](#)), 280  
[EmrHook](#) (class in [airflow.contrib.hooks.emr\\_hook](#)), 389  
[EmrJobFlowSensor](#) (class in [airflow.contrib.sensors.emr\\_job\\_flow\\_sensor](#)), 326  
[EmrStepSensor](#) (class in [airflow.contrib.sensors.emr\\_step\\_sensor](#)), 326  
[EmrTerminateJobFlowOperator](#) (class in [airflow.contrib.operators.emr\\_terminate\\_job\\_flow\\_operator](#)), 280  
[encrypt](#) () ([airflow.contrib.hooks.gcp\\_kms\\_hook.GoogleCloudKMSHook](#) method), 397  
[end](#) () ([airflow.contrib.executors.kubernetes\\_executor.KubernetesExecutor](#) method), 430  
[end](#) () ([airflow.executors.celery\\_executor.CeleryExecutor](#) method), 428  
[end](#) () ([airflow.executors.dask\\_executor.DaskExecutor](#) method), 429  
[end](#) () ([airflow.executors.local\\_executor.LocalExecutor](#) method), 429  
[end](#) () ([airflow.executors.sequential\\_executor.SequentialExecutor](#) method), 429  
[error](#) () ([airflow.models.TaskInstance](#) method), 346  
[execute](#) () ([airflow.contrib.hooks.discord\\_webhook\\_hook.DiscordWebhookHook](#) method), 389  
[execute](#) () ([airflow.contrib.hooks.slack\\_webhook\\_hook.SlackWebhookHook](#) method), 418  
[execute](#) () ([airflow.contrib.operators.aws\\_athena\\_operator.AWSAthenaOperator](#) method), 243  
[execute](#) () ([airflow.contrib.operators.dataflow\\_operator.DataFlowPythonOperator](#) method), 265  
[execute](#) () ([airflow.contrib.operators.discord\\_webhook\\_operator.DiscordWebhookOperator](#) method), 279  
[execute](#) () ([airflow.contrib.operators.file\\_to\\_gcs.FileToGoogleCloudStorageOperator](#) method), 281  
[execute](#) () ([airflow.contrib.operators.file\\_to\\_wasb.FileToWasbOperator](#) method), 281  
[execute](#) () ([airflow.contrib.operators.imap\\_attachment\\_to\\_s3\\_operator.ImapAttachmentToS3Operator](#) method), 295  
[execute](#) () ([airflow.contrib.operators.mongo\\_to\\_s3.MongoToS3Operator](#) method), 300  
[execute](#) () ([airflow.contrib.operators.slack\\_webhook\\_operator.SlackWebhookOperator](#) method), 317  
[execute](#) () ([airflow.contrib.operators.spark\\_jdbc\\_operator.SparkJDBCOperator](#) method), 318  
[execute](#) () ([airflow.contrib.operators.spark\\_sql\\_operator.SparkSqlOperator](#) method), 319  
[execute](#) () ([airflow.contrib.operators.spark\\_submit\\_operator.SparkSubmitOperator](#) method), 320  
[execute](#) () ([airflow.contrib.operators.sqoop\\_operator.SqoopOperator](#) method), 320  
[execute](#) () ([airflow.contrib.sensors.pubsub\\_sensor.PubSubPullSensor](#) method), 329  
[execute](#) () ([airflow.models.BaseOperator](#) method), 220, 337  
[execute](#) () ([airflow.operators.bash\\_operator.BashOperator](#) method), 222  
[execute](#) () ([airflow.operators.slack\\_operator.SlackAPIOperator](#) method), 237  
[execute\\_async](#) () ([airflow.contrib.executors.kubernetes\\_executor.KubernetesExecutor](#) method), 430  
[execute\\_async](#) () ([airflow.executors.dask\\_executor.DaskExecutor](#) method), 429  
[execute\\_async](#) () ([airflow.executors.local\\_executor.LocalExecutor](#) method), 429  
[execute\\_async](#) () ([airflow.executors.sequential\\_executor.SequentialExecutor](#) method), 429  
[execute\\_dml](#) () ([airflow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook](#) method), 379  
[execute\\_if\\_exists](#) () ([airflow.contrib.hooks.azure\\_container\\_instance\\_hook.AzureContainerInstanceHook](#) method), 369  
[exists](#) () ([airflow.contrib.hooks.gcs\\_hook.GoogleCloudStorageHook](#) method), 369  
[expand\\_role](#) () ([airflow.contrib.hooks.aws\\_hook.AwsHook](#) method), 368  
[export\\_table](#) () ([airflow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook](#) method), 384  
[export\\_table](#) () ([airflow.contrib.hooks.sqoop\\_hook.SqoopHook](#) method), 424  
[extract\\_to\\_gcs\\_bucket](#) () ([airflow.contrib.hooks.datastore\\_hook.DatastoreHook](#) method), 387  
[ExternalTaskSensor](#) (class in [airflow.contrib.sensors.external\\_task\\_sensor](#)), 239  
[fallback](#) ([airflow.models.DAG](#) attribute), 340

- FileSensor (class in *airflow.contrib.sensors.file\_sensor*), 326
- FileToGoogleCloudStorageOperator (class in *airflow.contrib.operators.file\_to\_gcs*), 281
- FileToWasbOperator (class in *airflow.contrib.operators.file\_to\_wasb*), 281
- find() (*airflow.contrib.hooks.mongo\_hook.MongoHook* method), 404
- find() (*airflow.models.DagRun* static method), 344
- folder (*airflow.models.DAG* attribute), 340
- following\_schedule() (*airflow.models.DAG* method), 340
- free\_reserved\_port() (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* method), 382
- FSHook (class in *airflow.contrib.hooks.fs\_hook*), 389
- FTPHook (class in *airflow.contrib.hooks.ftp\_hook*), 389
- FTPSensor (class in *airflow.contrib.sensors.ftp\_sensor*), 326
- FTPSHook (class in *airflow.contrib.hooks.ftp\_hook*), 391
- FTPSSensor (class in *airflow.contrib.sensors.ftp\_sensor*), 326
- ## G
- GceBaseOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 283
- GceHook (class in *airflow.contrib.hooks.gcp\_compute\_hook*), 393
- GceInstanceGroupManagerUpdateTemplateOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 283
- GceInstanceStartOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 284
- GceInstanceStopOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 284
- GceInstanceTemplateCopyOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 285
- GceSetMachineTypeOperator (class in *airflow.contrib.operators.gcp\_compute\_operator*), 285
- GcfFunctionDeleteOperator (class in *airflow.contrib.operators.gcp\_function\_operator*), 286
- GcfFunctionDeployOperator (class in *airflow.contrib.operators.gcp\_function\_operator*), 286
- GcfHook (class in *airflow.contrib.hooks.gcp\_function\_hook*), 395
- GCPTransferServiceHook (class in *airflow.contrib.hooks.gcp\_transfer\_hook*), 391
- generate\_command() (*airflow.models.TaskInstance* static method), 346
- GenericTransfer (class in *airflow.operators.generic\_transfer*), 225
- get\_active\_runs() (*airflow.models.DAG* method), 340
- get\_autocommit() (*airflow.hooks.dbapi\_hook.DbApiHook* method), 349
- get\_autocommit() (*airflow.hooks.mssql\_hook.MsSqlHook* method), 358
- get\_autocommit() (*airflow.hooks.mysql\_hook.MySqlHook* method), 358
- get\_available\_fields() (*airflow.contrib.hooks.salesforce\_hook.SalesforceHook* method), 416
- get\_bucket() (*airflow.hooks.S3\_hook.S3Hook* method), 362
- get\_cluster() (*airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook* method), 392
- get\_cluster\_states\_for\_table() (*airflow.contrib.hooks.gcp\_bigtable\_hook.BigtableHook* static method), 376
- get\_collection() (*airflow.contrib.hooks.mongo\_hook.MongoHook* method), 404
- get\_column\_families\_for\_table() (*airflow.contrib.hooks.gcp\_bigtable\_hook.BigtableHook* static method), 376
- get\_conn() (*airflow.contrib.hooks.aws\_athena\_hook.AWSAthenaHook* method), 365
- get\_conn() (*airflow.contrib.hooks.aws\_firehose\_hook.AwsFirehoseHook* method), 367
- get\_conn() (*airflow.contrib.hooks.aws\_glue\_catalog\_hook.AwsGlueCatalogHook* method), 367
- get\_conn() (*airflow.contrib.hooks.aws\_sns\_hook.AwsSnsHook* method), 368
- get\_conn() (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 371
- get\_conn() (*airflow.contrib.hooks.azure\_data\_lake\_hook.AzureDataLakeHook* method), 371
- get\_conn() (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 373
- get\_conn() (*airflow.contrib.hooks.bigquery\_hook.BigQueryHook* method), 374
- get\_conn() (*airflow.contrib.hooks.cassandra\_hook.CassandraHook* method), 378
- get\_conn() (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* method), 387

`get_conn()` (`airflow.contrib.hooks.ftp_hook.FTPHook` `method`), 390  
`get_conn()` (`airflow.contrib.hooks.ftp_hook.FTPSHook` `method`), 391  
`get_conn()` (`airflow.contrib.hooks.gcp_compute_hook.GoogleComputeHook` `method`), 393  
`get_conn()` (`airflow.contrib.hooks.gcp_dataflow_hook.DataflowHook` `method`), 385  
`get_conn()` (`airflow.contrib.hooks.gcp_dataproc_hook.DataprocHook` `method`), 385  
`get_conn()` (`airflow.contrib.hooks.gcp_function_hook.GcpFunctionHook` `method`), 396  
`get_conn()` (`airflow.contrib.hooks.gcp_kms_hook.GoogleCloudKMSHook` `method`), 397  
`get_conn()` (`airflow.contrib.hooks.gcp_mlengine_hook.MLEngineHook` `method`), 403  
`get_conn()` (`airflow.contrib.hooks.gcp_pubsub_hook.PubSubHook` `method`), 408  
`get_conn()` (`airflow.contrib.hooks.gcp_sql_hook.CloudSqlHook` `method`), 383  
`get_conn()` (`airflow.contrib.hooks.gcp_transfer_hook.GCPTransferServiceHook` `method`), 391  
`get_conn()` (`airflow.contrib.hooks.gcs_hook.GoogleCloudStorageHook` `method`), 399  
`get_conn()` (`airflow.contrib.hooks.mongo_hook.MongoHook` `method`), 404  
`get_conn()` (`airflow.contrib.hooks.pinot_hook.PinotDbApiHook` `method`), 406  
`get_conn()` (`airflow.contrib.hooks.redis_hook.RedisHook` `method`), 409  
`get_conn()` (`airflow.contrib.hooks.sagemaker_hook.SageMakerHook` `method`), 415  
`get_conn()` (`airflow.contrib.hooks.sftp_hook.SFTPHook` `method`), 410  
`get_conn()` (`airflow.contrib.hooks.snowflake_hook.SnowflakeHook` `method`), 418  
`get_conn()` (`airflow.contrib.hooks.ssh_hook.SSHHook` `method`), 411  
`get_conn()` (`airflow.contrib.hooks.vertica_hook.VerticaHook` `method`), 425  
`get_conn()` (`airflow.contrib.hooks.wasb_hook.WasbHook` `method`), 426  
`get_conn()` (`airflow.hooks.dbapi_hook.DbApiHook` `method`), 350  
`get_conn()` (`airflow.hooks.druid_hook.DruidDbApiHook` `method`), 351  
`get_conn()` (`airflow.hooks.hive_hooks.HiveServer2Hook` `method`), 355  
`get_conn()` (`airflow.hooks.http_hook.HttpHook` `method`), 357  
`get_conn()` (`airflow.hooks.jdbc_hook.JdbcHook` `method`), 358  
`get_conn()` (`airflow.hooks.mssql_hook.MsSqlHook` `method`), 358  
`get_conn()` (`airflow.hooks.mysql_hook.MySqlHook` `method`), 358  
`get_conn()` (`airflow.hooks.oracle_hook.OracleHook` `method`), 359  
`get_conn()` (`airflow.hooks.postgres_hook.PostgresHook` `method`), 360  
`get_conn()` (`airflow.hooks.presto_hook.PrestoHook` `method`), 360  
`get_conn()` (`airflow.hooks.sqlite_hook.SQLiteHook` `method`), 364  
`get_conn()` (`airflow.hooks.webhdfs_hook.WebHDFSHook` `method`), 364  
`get_conn()` (`airflow.contrib.hooks.gcs_hook.GoogleCloudStorageHook` `method`), 399  
`get_cursor()` (`airflow.contrib.hooks.aws_hook.AwsHook` `method`), 368  
`get_cursor()` (`airflow.hooks.dbapi_hook.DbApiHook` `method`), 350  
`get_dag()` (`airflow.models.DagBag` `method`), 343  
`get_dagrun()` (`airflow.models.DAG` `method`), 344  
`get_dagrun()` (`airflow.models.DAG` `method`), 340  
`get_dagrun()` (`airflow.models.TaskInstance` `method`), 347  
`get_database()` (`airflow.contrib.hooks.gcp_spanner_hook.CloudSpannerHook` `method`), 379  
`get_database()` (`airflow.contrib.hooks.gcp_sql_hook.CloudSqlHook` `method`), 383  
`get_database_hook()` (`airflow.contrib.hooks.gcp_sql_hook.CloudSqlDatabaseHook` `method`), 382  
`get_databases()` (`airflow.hooks.hive_hooks.HiveMetastoreHook` `method`), 354  
`get_db_hook()` (`airflow.operators.druid_check_operator.DruidCheckOperator` `method`), 225  
`get_default_view()` (`airflow.models.DAG` `method`), 340  
`get_direct_relative_ids()` (`airflow.models.BaseOperator` `method`), 220, 337  
`get_direct_relatives()` (`airflow.models.BaseOperator` `method`), 220, 337  
`get_document()` (`airflow.contrib.hooks.azure_cosmos_hook.AzureCosmosDBHook` `method`), 371  
`get_documents()` (`airflow.contrib.hooks.azure_cosmos_hook.AzureCosmosDBHook` `method`), 371  
`get_fernet()` (`in module airflow.models`), 349



[get\\_file\(\) \(airflow.contrib.hooks.azure\\_fileshare\\_hook.AzureFileShareHook method\), 373](#)  
[get\\_file\(\) \(airflow.contrib.hooks.azure\\_container\\_instance\\_hook.AzureContainerInstanceHook method\), 369](#)  
[get\\_file\(\) \(airflow.contrib.hooks.wasb\\_hook.WasbHook method\), 426](#)  
[get\\_file\\_to\\_stream\(\) \(airflow.contrib.hooks.gcs\\_hook.GoogleCloudStorageHook method\), 399](#)  
[get\\_file\\_to\\_stream\(\) \(airflow.contrib.hooks.azure\\_fileshare\\_hook.AzureFileShareHook method\), 373](#)  
[get\\_file\\_to\\_stream\(\) \(airflow.contrib.hooks.azure\\_container\\_instance\\_hook.AzureContainerInstanceHook method\), 369](#)  
[get\\_first\(\) \(airflow.contrib.hooks.pinot\\_hook.PinotDbApiHook method\), 406](#)  
[get\\_first\(\) \(airflow.hooks.dbapi\\_hook.DbApiHook method\), 350](#)  
[get\\_first\(\) \(airflow.hooks.presto\\_hook.PrestoHook method\), 360](#)  
[get\\_first\(\) \(airflow.operators.druid\\_check\\_operator.DruidCheckOperator method\), 225](#)  
[get\\_flat\\_relative\\_ids\(\) \(airflow.models.BaseOperator method\), 220, 337](#)  
[get\\_flat\\_relative\\_ids\(\) \(airflow.models.BaseOperator method\), 220, 337](#)  
[get\\_flat\\_relatives\(\) \(airflow.models.BaseOperator method\), 220, 337](#)  
[get\\_function\(\) \(airflow.contrib.hooks.gcp\\_function\\_hook.GcfHook method\), 396](#)  
[get\\_hook\(\) \(airflow.contrib.sensors.aws\\_glue\\_catalog\\_partition\\_sensor.AwsGlueCatalogPartitionSensor method\), 322](#)  
[get\\_hook\(\) \(airflow.contrib.sensors.aws\\_glue\\_catalog\\_start\\_sensor.AwsGlueCatalogStartSensor method\), 387](#)  
[get\\_instance\(\) \(airflow.contrib.hooks.gcp\\_bigtable\\_hook.BigtableHook method\), 377](#)  
[get\\_instance\(\) \(airflow.contrib.hooks.gcp\\_container\\_hook.GKEClusterHook method\), 392](#)  
[get\\_instance\(\) \(airflow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook method\), 380](#)  
[get\\_instance\(\) \(airflow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook method\), 384](#)  
[get\\_instance\\_group\\_manager\(\) \(airflow.contrib.hooks.gcp\\_compute\\_hook.GceHook method\), 393](#)  
[get\\_instance\\_template\(\) \(airflow.contrib.hooks.gcp\\_compute\\_hook.GceHook method\), 393](#)  
[get\\_jobs\\_id\(\) \(airflow.contrib.hooks.qubole\\_hook.QuboleHook method\), 409](#)  
[get\\_key\(\) \(airflow.hooks.S3\\_hook.S3Hook method\), 362](#)  
[get\\_last\\_dagrun\(\) \(in module airflow.models\), 349](#)  
[get\\_latest\\_runs\(\) \(airflow.models.DagRun class method\), 344](#)  
[get\\_log\(\) \(airflow.contrib.hooks.qubole\\_hook.QuboleHook method\), 409](#)  
[get\\_log\\_conn\(\) \(airflow.contrib.hooks.sagemaker\\_hook.SageMakerHook method\), 415](#)  
[get\\_md5hash\(\) \(airflow.contrib.hooks.gcs\\_hook.GoogleCloudStorageHook method\), 399](#)  
[get\\_messages\(\) \(airflow.contrib.hooks.azure\\_container\\_instance\\_hook.AzureContainerInstanceHook method\), 369](#)  
[get\\_metastore\\_client\(\) \(airflow.hooks.hive\\_hooks.HiveMetastoreHook method\), 354](#)  
[get\\_mod\\_time\(\) \(airflow.contrib.hooks.ftp\\_hook.FTPHook method\), 399](#)  
[get\\_model\(\) \(airflow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook method\), 403](#)  
[get\\_num\\_active\\_runs\(\) \(airflow.models.DAG method\), 340](#)  
[get\\_num\\_task\\_instances\(\) \(airflow.models.DAG static method\), 341](#)  
[get\\_object\\_from\\_salesforce\(\) \(airflow.contrib.hooks.salesforce\\_hook.SalesforceHook method\), 416](#)  
[get\\_operation\(\) \(airflow.contrib.hooks.gcp\\_container\\_hook.GKEClusterHook method\), 392](#)  
[get\\_operation\(\) \(airflow.contrib.hooks.gcp\\_spanner\\_hook.CloudSpannerHook method\), 380](#)  
[get\\_operation\(\) \(airflow.contrib.hooks.gcp\\_sql\\_hook.CloudSqlHook method\), 384](#)  
[get\\_operation\(\) \(airflow.contrib.hooks.gcp\\_compute\\_hook.GceHook method\), 393](#)  
[get\\_operation\(\) \(airflow.contrib.hooks.gcp\\_mlengine\\_hook.MLEngineHook method\), 403](#)  
[get\\_pandas\\_df\(\) \(airflow.contrib.hooks.bigquery\\_hook.BigQueryHook method\), 374](#)  
[get\\_pandas\\_df\(\) \(airflow.contrib.hooks.pinot\\_hook.PinotDbApiHook method\), 406](#)  
[get\\_pandas\\_df\(\) \(airflow.hooks.dbapi\\_hook.DbApiHook method\), 350](#)  
[get\\_pandas\\_df\(\) \(airflow.hooks.druid\\_hook.DruidDbApiHook method\), 351](#)  
[get\\_pandas\\_df\(\) \(airflow.hooks.hive\\_hooks.HiveServer2Hook method\), 355](#)  
[get\\_pandas\\_df\(\) \(airflow.hooks.presto\\_hook.PrestoHook method\), 360](#)  
[get\\_partitions\(\) \(airflow.contrib.hooks.aws\\_glue\\_catalog\\_hook.AwsGlueCatalogHook method\), 367](#)  
[get\\_partitions\(\) \(airflow.hooks.hive\\_hooks.HiveMetastoreHook method\), 354](#)  
[get\\_previous\\_dagrun\(\) \(airflow.models.DagRun](#)

- `method`), 344
- `get_previous_scheduled_dagrun()` (*airflow.models.DagRun* method), 344
- `get_query_results()` (*airflow.contrib.hooks.aws\_athena\_hook.AWSAthenaHook* method), 365
- `get_records()` (*airflow.contrib.hooks.pinot\_hook.PinotDbApiHook* method), 406
- `get_records()` (*airflow.hooks.dbapi\_hook.DbApiHook* method), 350
- `get_records()` (*airflow.hooks.hive\_hooks.HiveServer2Hook* method), 356
- `get_records()` (*airflow.hooks.presto\_hook.PrestoHook* method), 360
- `get_results()` (*airflow.contrib.hooks.qubole\_hook.QuboleHook* method), 409
- `get_results()` (*airflow.hooks.hive\_hooks.HiveServer2Hook* method), 356
- `get_run()` (*airflow.models.DagRun* static method), 344
- `get_run_dates()` (*airflow.models.DAG* method), 341
- `get_s3_key()` (*airflow.contrib.operators.s3\_to\_sftp\_operator.S3ToSFTPOperator* static method), 311
- `get_s3_key()` (*airflow.contrib.operators.sftp\_to\_s3\_operator.SFTPToS3Operator* static method), 313
- `get_service()` (*airflow.contrib.hooks.bigquery\_hook.BigQueryHook* method), 375
- `get_session()` (*airflow.contrib.hooks.aws\_hook.AwsHook* method), 368
- `get_size()` (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 390
- `get_size()` (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 399
- `get_sqlproxy_runner()` (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* method), 382
- `get_state_exitcode_details()` (*airflow.contrib.hooks.azure\_container\_instance\_hook.AzureContainerInstanceHook* method), 370
- `get_table()` (*airflow.hooks.hive\_hooks.HiveMetastoreHook* method), 354
- `get_tables()` (*airflow.hooks.hive\_hooks.HiveMetastoreHook* method), 354
- `get_task_instance()` (*airflow.models.DagRun* method), 345
- `get_task_instances()` (*airflow.models.BaseOperator* method), 220, 337
- `get_task_instances()` (*airflow.models.DagRun* method), 345
- `get_template_env()` (*airflow.models.DAG* method), 341
- `get_tunnel()` (*airflow.contrib.hooks.ssh\_hook.SSHHook* method), 411
- `get_uri()` (*airflow.contrib.hooks.pinot\_hook.PinotDbApiHook* method), 406
- `get_uri()` (*airflow.contrib.hooks.snowflake\_hook.SnowflakeHook* method), 418
- `get_uri()` (*airflow.hooks.druid\_hook.DruidDbApiHook* method), 351
- `get_wildcard_key()` (*airflow.hooks.S3\_hook.S3Hook* method), 362
- `GKEClusterCreateOperator` (class in *airflow.contrib.operators.gcp\_container\_operator*), 281
- `GKEClusterDeleteOperator` (class in *airflow.contrib.operators.gcp\_container\_operator*), 282
- `GKEClusterHook` (class in *airflow.contrib.hooks.gcp\_container\_hook*), 391
- `GKEPodOperator` (class in *airflow.contrib.operators.gcp\_container\_operator*), 287
- `GoogleCloudBaseHook` (class in *airflow.contrib.hooks.gcp\_api\_base\_hook*), 396
- `GoogleCloudKMSHook` (class in *airflow.contrib.hooks.gcp\_kms\_hook*), 397
- `GoogleCloudStorageBucketCreateAclEntryOperator` (class in *airflow.contrib.operators.gcs\_acl\_operator*), 287
- `GoogleCloudStorageCreateBucketOperator` (class in *airflow.contrib.operators.gcs\_operator*), 287
- `GoogleCloudStorageDownloadOperator` (class in *airflow.contrib.operators.gcs\_download\_operator*), 288
- `GoogleCloudStorageHook` (class in *airflow.contrib.hooks.gcp\_storage\_hook*), 397
- `GoogleCloudStorageListOperator` (class in *airflow.contrib.operators.gcs\_list\_operator*), 288
- `GoogleCloudStorageObjectCreateAclEntryOperator` (class in *airflow.contrib.operators.gcs\_acl\_operator*), 289

- GoogleCloudStorageObjectSensor (class in *airflow.contrib.sensors.gcs\_sensor*), 326
- GoogleCloudStorageObjectUpdatedSensor (class in *airflow.contrib.sensors.gcs\_sensor*), 327
- GoogleCloudStoragePrefixSensor (class in *airflow.contrib.sensors.gcs\_sensor*), 327
- GoogleCloudStorageToBigQueryOperator (class in *airflow.contrib.operators.gcs\_to\_bq*), 289
- GoogleCloudStorageToGoogleCloudStorageOperator (class in *airflow.contrib.operators.gcs\_to\_gcs*), 291
- GoogleCloudStorageToGoogleCloudStorageTransferOperator (class in *airflow.contrib.operators.gcs\_to\_gcs\_transfer\_operator*), 292
- GoogleCloudStorageToS3Operator (class in *airflow.contrib.operators.gcs\_to\_s3*), 293
- ## H
- handle\_callback() (*airflow.models.DAG* method), 341
- has\_dag() (*airflow.models.BaseOperator* method), 220, 337
- has\_mail\_attachment() (*airflow.contrib.hooks.imap\_hook.ImapHook* method), 402
- HipChatAPIOperator (class in *airflow.contrib.operators.hipchat\_operator*), 293
- HipChatAPISendRoomNotificationOperator (class in *airflow.contrib.operators.hipchat\_operator*), 294
- Hive2SambaOperator (class in *airflow.operators.hive\_to\_samba\_operator*), 226
- HiveCliHook (class in *airflow.hooks.hive\_hooks*), 352
- HiveMetastoreHook (class in *airflow.hooks.hive\_hooks*), 353
- HiveOperator (class in *airflow.operators.hive\_operator*), 226
- HivePartitionSensor (class in *airflow.sensors.hive\_partition\_sensor*), 239
- HiveServer2Hook (class in *airflow.hooks.hive\_hooks*), 355
- HiveStatsCollectionOperator (class in *airflow.operators.hive\_stats\_operator*), 226
- HiveToDruidTransfer (class in *airflow.operators.hive\_to\_druid*), 227
- HiveToDynamoDBTransferOperator (class in *airflow.contrib.operators.hive\_to\_dynamodb*), 294
- HiveToMySQLTransfer (class in *airflow.operators.hive\_to\_mysql*), 228
- HttpHook (class in *airflow.hooks.http\_hook*), 356
- HttpSensor (class in *airflow.sensors.http\_sensor*), 240
- ## I
- ImapAttachmentSensor (class in *airflow.contrib.sensors.imap\_attachment\_sensor*), 327
- ImapAttachmentToS3Operator (class in *airflow.contrib.operators.imap\_attachment\_to\_s3\_operator*), 295
- ImapHook (class in *airflow.contrib.hooks.imap\_hook*), 402
- import\_from\_storage\_bucket() (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* method), 388
- import\_instance() (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlHook* method), 384
- import\_query() (*airflow.contrib.hooks.sqoop\_hook.SqoopHook* method), 424
- import\_table() (*airflow.contrib.hooks.sqoop\_hook.SqoopHook* method), 425
- init\_on\_load() (*airflow.models.TaskInstance* method), 347
- init\_run\_context() (*airflow.models.TaskInstance* method), 347
- insert\_bucket\_acl() (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 399
- insert\_documents() (*airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook* method), 371
- insert\_instance\_template() (*airflow.contrib.hooks.gcp\_compute\_hook.GceHook* method), 394
- insert\_many() (*airflow.contrib.hooks.mongo\_hook.MongoHook* method), 404
- insert\_object\_acl() (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 400
- insert\_one() (*airflow.contrib.hooks.mongo\_hook.MongoHook* method), 404
- insert\_rows() (*airflow.contrib.hooks.bigquery\_hook.BigQueryHook* method), 375
- insert\_rows() (*airflow.contrib.hooks.pinot\_hook.PinotDbApiHook* method), 406

`insert_rows()` (*airflow.hooks.dbapi\_hook.DbApiHook* method), 350  
`insert_rows()` (*airflow.hooks.druid\_hook.DruidDbApiHook* method), 351  
`insert_rows()` (*airflow.hooks.oracle\_hook.OracleHook* method), 359  
`insert_rows()` (*airflow.hooks.presto\_hook.PrestoHook* method), 360  
`IntervalCheckOperator` (class in *airflow.operators.check\_operator*), 228  
`InvalidFernetToken`, 345  
`invoke_lambda()` (*airflow.contrib.hooks.aws\_lambda\_hook.AwsLambdaHook* method), 368  
`is_eligible_to_retry()` (*airflow.models.TaskInstance* method), 347  
`is_fixed_time_schedule()` (*airflow.models.DAG* method), 341  
`is_paused` (*airflow.models.DAG* attribute), 341  
`is_premature` (*airflow.models.TaskInstance* attribute), 347  
`is_updated_after()` (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 400

## J

`JdbcHook` (class in *airflow.hooks.jdbc\_hook*), 357  
`JdbcOperator` (class in *airflow.operators.jdbc\_operator*), 228  
`JenkinsHook` (class in *airflow.contrib.hooks.jenkins\_hook*), 402  
`JenkinsJobTriggerOperator` (class in *airflow.contrib.operators.jenkins\_job\_trigger\_operator*), 295  
`JiraHook` (class in *airflow.contrib.hooks.jira\_hook*), 402  
`JiraOperator` (class in *airflow.contrib.operators.jira\_operator*), 296  
`JiraSensor` (class in *airflow.contrib.sensors.jira\_sensor*), 328  
`JiraTicketSensor` (class in *airflow.contrib.sensors.jira\_sensor*), 328

## K

`key` (*airflow.models.TaskInstance* attribute), 347  
`kill()` (*airflow.contrib.hooks.qubole\_hook.QuboleHook* method), 409  
`kill_zombies()` (*airflow.models.DagBag* method), 343

`KubernetesExecutor` (class in *airflow.contrib.executors.kubernetes\_executor*), 430  
`KubernetesPodOperator` (class in *airflow.contrib.operators.kubernetes\_pod\_operator*), 296

## L

`latest_execution_date` (*airflow.models.DAG* attribute), 341  
`LatestOnlyOperator` (class in *airflow.operators.latest\_only\_operator*), 229  
`list()` (*airflow.contrib.hooks.azure\_data\_lake\_hook.AzureDataLakeHook* method), 372  
`list()` (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* method), 400  
`list_directories_and_files()` (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 373  
`list_directory()` (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 390  
`list_directory()` (*airflow.contrib.hooks.sftp\_hook.SFTPHook* method), 410  
`list_keys()` (*airflow.hooks.S3\_hook.S3Hook* method), 362  
`list_prefixes()` (*airflow.hooks.S3\_hook.S3Hook* method), 362  
`list_versions()` (*airflow.contrib.hooks.gcp\_mlengine\_hook.MLEngineHook* method), 403  
`load_bytes()` (*airflow.hooks.S3\_hook.S3Hook* method), 362  
`load_df()` (*airflow.hooks.hive\_hooks.HiveCliHook* method), 352  
`load_file()` (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 374  
`load_file()` (*airflow.contrib.hooks.wasb\_hook.WasbHook* method), 426  
`load_file()` (*airflow.hooks.hive\_hooks.HiveCliHook* method), 353  
`load_file()` (*airflow.hooks.S3\_hook.S3Hook* method), 363  
`load_file()` (*airflow.hooks.webhdfs\_hook.WebHDFSHook* method), 364  
`load_file_obj()` (*airflow.hooks.S3\_hook.S3Hook* method), 363  
`load_stream()` (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 374  
`load_string()` (*airflow.contrib.hooks.azure\_fileshare\_hook.AzureFileShareHook* method), 374



load\_string() (airflow.contrib.hooks.wasb\_hook.WasbHook method), 427

load\_string() (airflow.hooks.S3\_hook.S3Hook method), 363

LocalExecutor (class in airflow.executors.local\_executor), 429

log\_stream() (airflow.contrib.hooks.sagemaker\_hook.SageMakerHook method), 415

lookup() (airflow.contrib.hooks.datastore\_hook.DatastoreHook method), 388

## M

make\_query() (airflow.contrib.hooks.salesforce\_hook.SalesforceHook method), 416

max\_partition() (airflow.hooks.hive\_hooks.HiveMetastoreHook method), 355

max\_partition() (in module airflow.macros.hive), 334

MetastorePartitionSensor (class in airflow.sensors.metastore\_partition\_sensor), 240

MLEngineBatchPredictionOperator (class in airflow.contrib.operators.mlengine\_operator), 297

MLEngineHook (class in airflow.contrib.hooks.gcp\_mlengine\_hook), 402

MLEngineModelOperator (class in airflow.contrib.operators.mlengine\_operator), 298

MLEngineTrainingOperator (class in airflow.contrib.operators.mlengine\_operator), 299

MLEngineVersionOperator (class in airflow.contrib.operators.mlengine\_operator), 299

MongoHook (class in airflow.contrib.hooks.mongo\_hook), 403

MongoSensor (class in airflow.contrib.sensors.mongo\_sensor), 328

MongoToS3Operator (class in airflow.contrib.operators.mongo\_to\_s3), 300

MssqlHook (class in airflow.hooks.mssql\_hook), 358

MssqlOperator (class in airflow.operators.mssql\_operator), 229

MssqlToHiveTransfer (class in airflow.operators.mssql\_to\_hive), 229

multi\_stream\_iter() (airflow.contrib.hooks.sagemaker\_hook.SageMakerHook method), 415

MySQLHook (class in airflow.hooks.mysql\_hook), 358

MySQLOperator (class in airflow.operators.mysql\_operator), 229

MySQLToGoogleCloudStorageOperator (class in airflow.contrib.operators.mysql\_to\_gcs), 300

MySQLToHiveTransfer (class in airflow.operators.mysql\_to\_hive), 230

NamedHivePartitionSensor (class in airflow.sensors.named\_hive\_partition\_sensor), 240

next\_retry\_datetime() (airflow.models.TaskInstance method), 347

on\_schedule() (airflow.models.DAG method), 341

NullFernet (class in airflow.models), 345

## O

on\_error() (airflow.contrib.hooks.segment\_hook.SegmentHook method), 417

on\_kill() (airflow.contrib.operators.aws\_athena\_operator.AWSAthenaOperator method), 243

on\_kill() (airflow.models.BaseOperator method), 220, 337

open\_slots() (airflow.models.Pool method), 345

OpenFaasHook (class in airflow.contrib.hooks.openfaas\_hook), 405

OracleHook (class in airflow.hooks.oracle\_hook), 359

OracleOperator (class in airflow.operators.oracle\_operator), 230

OracleToAzureDataLakeTransfer (class in airflow.contrib.operators.oracle\_to\_azure\_data\_lake\_transfer), 301

OracleToOracleTransfer (class in airflow.contrib.operators.oracle\_to\_oracle\_transfer), 301

## P

patch\_database() (airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlHook method), 384

patch\_instance() (airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlHook method), 384

patch\_instance\_group\_manager() (airflow.contrib.hooks.gcp\_compute\_hook.GceHook method), 394

PigCliHook (class in airflow.hooks.pig\_hook), 359

PigOperator (class in airflow.operators.pig\_operator), 231

PinotDbApiHook (class in airflow.contrib.hooks.pinot\_hook), 406

poke() (airflow.contrib.operators.gcp\_bigtable\_operator.BigtableTableWrapper method), 256

poke () (airflow.contrib.sensors.aws\_athena\_sensor.AthenaSensor method), 322

poke () (airflow.contrib.sensors.aws\_glue\_catalog\_partition\_sensor.AirflowGlueCatalogPartitionSensor method), 323

poke () (airflow.contrib.sensors.aws\_redshift\_cluster\_sensor.AwsRedshiftClusterSensor method), 323

poke () (airflow.contrib.sensors.azure\_cosmos\_sensor.AzureCosmosDocumentSensor method), 323

poke () (airflow.contrib.sensors.bash\_sensor.BashSensor method), 323

poke () (airflow.contrib.sensors.bigquery\_sensor.BigQueryTableSensor method), 324

poke () (airflow.contrib.sensors.cassandra\_record\_sensor.CassandraRecordSensor method), 324

poke () (airflow.contrib.sensors.cassandra\_table\_sensor.CassandraTableSensor method), 324

poke () (airflow.contrib.sensors.celery\_queue\_sensor.CeleryQueueSensor method), 324

poke () (airflow.contrib.sensors.datadog\_sensor.DatadogSensor method), 325

poke () (airflow.contrib.sensors.emr\_base\_sensor.EmrBaseSensor method), 326

poke () (airflow.contrib.sensors.file\_sensor.FileSensor method), 326

poke () (airflow.contrib.sensors.ftp\_sensor.FTPSensor method), 326

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStorageObjectSensor method), 327

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStorageObjectUpdateSensor method), 327

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStoragePrefixSensor method), 327

poke () (airflow.contrib.sensors.imap\_attachment\_sensor.ImapAttachmentSensor method), 328

poke () (airflow.contrib.sensors.jira\_sensor.JiraSensor method), 328

poke () (airflow.contrib.sensors.jira\_sensor.JiraTicketSensor method), 328

poke () (airflow.contrib.sensors.mongo\_sensor.MongoSensor method), 329

poke () (airflow.contrib.sensors.pubsub\_sensor.PubSubPullSensor method), 329

poke () (airflow.contrib.sensors.python\_sensor.PythonSensor method), 329

poke () (airflow.contrib.sensors.qubole\_sensor.QuboleSensor method), 330

poke () (airflow.contrib.sensors.redis\_key\_sensor.RedisKeySensor method), 330

poke () (airflow.contrib.sensors.sagemaker\_base\_sensor.SageMakerBaseSensor method), 330

poke () (airflow.contrib.sensors.sftp\_sensor.SFTPSensor method), 330

poke () (airflow.contrib.sensors.wasb\_sensor.WasbBlobSensor method), 331

poke () (airflow.contrib.sensors.wasb\_sensor.WasbPrefixSensor method), 332

poke () (airflow.contrib.sensors.aws\_glue\_catalog\_partition\_sensor.AirflowGlueCatalogPartitionSensor method), 325

poke () (airflow.contrib.sensors.aws\_redshift\_cluster\_sensor.AwsRedshiftClusterSensor method), 222

poke () (airflow.contrib.sensors.azure\_cosmos\_sensor.AzureCosmosDocumentSensor method), 239

poke () (airflow.contrib.sensors.bash\_sensor.BashSensor method), 239

poke () (airflow.contrib.sensors.bigquery\_sensor.BigQueryTableSensor method), 240

poke () (airflow.contrib.sensors.cassandra\_record\_sensor.CassandraRecordSensor method), 240

poke () (airflow.contrib.sensors.cassandra\_table\_sensor.CassandraTableSensor method), 241

poke () (airflow.contrib.sensors.celery\_queue\_sensor.CeleryQueueSensor method), 241

poke () (airflow.contrib.sensors.datadog\_sensor.DatadogSensor method), 242

poke () (airflow.contrib.sensors.emr\_base\_sensor.EmrBaseSensor method), 242

poke () (airflow.contrib.sensors.file\_sensor.FileSensor method), 242

poke () (airflow.contrib.sensors.ftp\_sensor.FTPSensor method), 242

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStorageObjectSensor method), 242

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStorageObjectUpdateSensor method), 295

poke () (airflow.contrib.sensors.gcs\_sensor.GoogleCloudStoragePrefixSensor method), 295

poke () (airflow.contrib.sensors.imap\_attachment\_sensor.ImapAttachmentSensor method), 388

poke () (airflow.contrib.sensors.jira\_sensor.JiraSensor method), 366

poke () (airflow.contrib.sensors.jira\_sensor.JiraTicketSensor method), 366

poke () (airflow.contrib.sensors.mongo\_sensor.MongoSensor method), 347

poke () (airflow.contrib.sensors.pubsub\_sensor.PubSubPullSensor method), 424

poke () (airflow.contrib.sensors.python\_sensor.PythonSensor method), 386

poke () (airflow.contrib.sensors.qubole\_sensor.QuboleSensor method), 220, 337

poke () (airflow.contrib.sensors.redis\_key\_sensor.RedisKeySensor method), 359

poke () (airflow.contrib.sensors.sagemaker\_base\_sensor.SageMakerBaseSensor method), 231

poke () (airflow.contrib.sensors.sftp\_sensor.SFTPSensor method), 302

poke () (airflow.contrib.sensors.wasb\_sensor.WasbBlobSensor method), 302

[pre\\_execute\(\)](#) ([airflow.models.BaseOperator](#) [method](#)), 367  
[prepare\\_request\(\)](#) ([airflow.contrib.operators.hipchat\\_operator.HipChatAPISendRequestNotificationOperator](#) [method](#)), 294  
[prepare\\_request\(\)](#) ([airflow.contrib.operators.hipchat\\_operator.HipChatAPISendRequestNotificationOperator](#) [method](#)), 294  
[prepare\\_template\(\)](#) ([airflow.models.BaseOperator](#) [method](#)), 221, 337  
[PrestoCheckOperator](#) (class in [airflow.operators.presto\\_check\\_operator](#)), 231  
[PrestoHook](#) (class in [airflow.hooks.presto\\_hook](#)), 360  
[PrestoIntervalCheckOperator](#) (class in [airflow.operators.presto\\_check\\_operator](#)), 232  
[PrestoToMySQLTransfer](#) (class in [airflow.operators.presto\\_to\\_mysql](#)), 232  
[PrestoValueCheckOperator](#) (class in [airflow.operators.presto\\_check\\_operator](#)), 232  
[previous\\_schedule\(\)](#) ([airflow.models.DAG](#) [method](#)), 342  
[previous\\_ti](#) ([airflow.models.TaskInstance](#) attribute), 347  
[process\\_file\(\)](#) ([airflow.models.DagBag](#) [method](#)), 343  
[publish\(\)](#) ([airflow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 408  
[publish\\_to\\_target\(\)](#) ([airflow.contrib.hooks.aws\\_sns\\_hook.AwsSnsHook](#) [method](#)), 368  
[PubSubHook](#) (class in [airflow.contrib.hooks.gcp\\_pubsub\\_hook](#)), 407  
[PubSubPublishOperator](#) (class in [airflow.contrib.operators.pubsub\\_operator](#)), 302  
[PubSubPullSensor](#) (class in [airflow.contrib.sensors.pubsub\\_sensor](#)), 329  
[PubSubSubscriptionCreateOperator](#) (class in [airflow.contrib.operators.pubsub\\_operator](#)), 302  
[PubSubSubscriptionDeleteOperator](#) (class in [airflow.contrib.operators.pubsub\\_operator](#)), 303  
[PubSubTopicCreateOperator](#) (class in [airflow.contrib.operators.pubsub\\_operator](#)), 303  
[PubSubTopicDeleteOperator](#) (class in [airflow.contrib.operators.pubsub\\_operator](#)), 304  
[pull\(\)](#) ([airflow.contrib.hooks.gcp\\_pubsub\\_hook.PubSubHook](#) [method](#)), 408  
[put\\_records\(\)](#) ([airflow.contrib.hooks.aws\\_firehose\\_hook.AwsFirehoseHook](#) [method](#)), 367  
[PythonOperator](#) (class in [airflow.operators.python\\_operator](#)), 232  
[PythonSensor](#) (class in [airflow.contrib.sensors.python\\_sensor](#)), 329  
[PythonVirtualenvOperator](#) (class in [airflow.operators.python\\_virtualenv\\_operator](#)), 233

## Q

[QuboleCheckHook](#) (class in [airflow.contrib.hooks.qubole\\_check\\_hook](#)), 408  
[QuboleCheckOperator](#) (class in [airflow.contrib.operators.qubole\\_check\\_operator](#)), 304  
[QuboleFileSensor](#) (class in [airflow.contrib.sensors.qubole\\_sensor](#)), 329  
[QuboleHook](#) (class in [airflow.contrib.hooks.qubole\\_hook](#)), 408  
[QuboleOperator](#) (class in [airflow.contrib.operators.qubole\\_operator](#)), 305  
[QubolePartitionSensor](#) (class in [airflow.contrib.sensors.qubole\\_sensor](#)), 330  
[QuboleSensor](#) (class in [airflow.contrib.sensors.qubole\\_sensor](#)), 330  
[QuboleValueCheckOperator](#) (class in [airflow.contrib.operators.qubole\\_check\\_operator](#)), 307  
[query\\_metric\(\)](#) ([airflow.contrib.hooks.datadog\\_hook.DatadogHook](#) [method](#)), 386  
[queued\\_slots\(\)](#) ([airflow.models.Pool](#) [method](#)), 345

## R

[random\(\)](#) (in module [airflow.macros](#)), 333  
[read\\_file\(\)](#) ([airflow.contrib.hooks.wasb\\_hook.WasbHook](#) [method](#)), 427  
[read\\_key\(\)](#) ([airflow.hooks.S3\\_hook.S3Hook](#) [method](#)), 363  
[ready\\_for\\_retry\(\)](#) ([airflow.models.TaskInstance](#) [method](#)), 347  
[record\\_exists\(\)](#) ([airflow.contrib.hooks.cassandra\\_hook.CassandraHook](#) [method](#)), 378  
[RedisHook](#) (class in [airflow.contrib.hooks.redis\\_hook](#)), 409  
[RedisKeySensor](#) (class in [airflow.contrib.sensors.redis\\_key\\_sensor](#)), 330  
[RedshiftHook](#) (class in [airflow.contrib.hooks.redshift\\_hook](#)), 409  
[RedshiftToS3Transfer](#) (class in [airflow.operators.redshift\\_to\\_s3\\_operator](#)), 234

`refresh_from_db()` (*airflow.models.DagRun* *method*), 345  
`refresh_from_db()` (*airflow.models.TaskInstance* *method*), 347  
`rename()` (*airflow.contrib.hooks.ftp\_hook.FTPHook* *method*), 390  
`render_template()` (*airflow.models.BaseOperator* *method*), 221, 337  
`render_template_from_field()` (*airflow.models.BaseOperator* *method*), 221, 337  
`replace_many()` (*airflow.contrib.hooks.mongo\_hook.MongoHook* *method*), 404  
`replace_one()` (*airflow.contrib.hooks.mongo\_hook.MongoHook* *method*), 405  
`reserve_free_tcp_port()` (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* *method*), 382  
`restore_from_cluster_snapshot()` (*airflow.contrib.hooks.redshift\_hook.RedshiftHook* *method*), 410  
`retrieve_connection()` (*airflow.contrib.hooks.gcp\_sql\_hook.CloudSqlDatabaseHook* *method*), 382  
`retrieve_file()` (*airflow.contrib.hooks.ftp\_hook.FTPHook* *method*), 390  
`retrieve_file()` (*airflow.contrib.hooks.sftp\_hook.SFTPHook* *method*), 410  
`retrieve_mail_attachments()` (*airflow.contrib.hooks.imap\_hook.ImapHook* *method*), 402  
`rewrite()` (*airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook* *method*), 400  
`rollback()` (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* *method*), 388  
`run()` (*airflow.hooks.dbapi\_hook.DbApiHook* *method*), 350  
`run()` (*airflow.hooks.http\_hook.HttpHook* *method*), 357  
`run()` (*airflow.hooks.presto\_hook.PrestoHook* *method*), 360  
`run()` (*airflow.models.BaseOperator* *method*), 221, 337  
`run()` (*airflow.models.DAG* *method*), 342  
`run_and_check()` (*airflow.hooks.http\_hook.HttpHook* *method*), 357  
`run_cli()` (*airflow.hooks.hive\_hooks.HiveCliHook* *method*), 353  
`run_cli()` (*airflow.hooks.pig\_hook.PigCliHook* *method*), 359  
`run_now()` (*airflow.contrib.hooks.databricks\_hook.DatabricksHook* *method*), 385  
`run_query()` (*airflow.contrib.hooks.aws\_athena\_hook.AWSAthenaHook* *method*), 366  
`run_query()` (*airflow.contrib.hooks.datastore\_hook.DatastoreHook* *method*), 388  
`run_query()` (*airflow.contrib.hooks.spark\_sql\_hook.SparkSqlHook* *method*), 421  
`run_with_advanced_retry()` (*airflow.hooks.http\_hook.HttpHook* *method*), 357

## S

`S3CopyObjectOperator` (*class* in *airflow.contrib.operators.s3\_copy\_object\_operator*), 308  
`S3DeleteObjectsOperator` (*class* in *airflow.contrib.operators.s3\_delete\_objects\_operator*), 308  
`S3FileTransformOperator` (*class* in *airflow.operators.s3\_file\_transform\_operator*), 234  
`S3Hook` (*class* in *airflow.hooks.S3\_hook*), 360  
`S3KeySensor` (*class* in *airflow.sensors.s3\_key\_sensor*), 241  
`S3ListOperator` (*class* in *airflow.contrib.operators.s3\_list\_operator*), 309  
`S3PrefixSensor` (*class* in *airflow.sensors.s3\_prefix\_sensor*), 241  
`S3ToGoogleCloudStorageOperator` (*class* in *airflow.contrib.operators.s3\_to\_gcs\_operator*), 310  
`S3ToGoogleCloudStorageTransferOperator` (*class* in *airflow.contrib.operators.s3\_to\_gcs\_transfer\_operator*), 310  
`S3ToHiveTransfer` (*class* in *airflow.operators.s3\_to\_hive\_operator*), 235  
`S3ToRedshiftTransfer` (*class* in *airflow.operators.s3\_to\_redshift\_operator*), 236  
`S3ToSFTPOperator` (*class* in *airflow.contrib.operators.s3\_to\_sftp\_operator*), 311  
`SageMakerBaseOperator` (*class* in *airflow.contrib.operators.sagemaker\_base\_operator*), 313  
`SageMakerBaseSensor` (*class* in *airflow.contrib.sensors.sagemaker\_base\_sensor*), 330  
`SageMakerEndpointConfigOperator` (*class* in *airflow.contrib.operators.sagemaker\_endpoint\_config\_operator*), 313

[SageMakerEndpointOperator](#) (class in [airflow.contrib.operators.sagemaker\\_endpoint\\_operator](#)), 351  
[set\\_autocommit\(\)](#) (airflow.contrib.operators.sagemaker\_endpoint\_operator method), 352  
[SageMakerEndpointSensor](#) (class in [airflow.contrib.sensors.sagemaker\\_endpoint\\_sensor](#)), 331  
[set\\_autocommit\(\)](#) (airflow.contrib.sensors.sagemaker\_endpoint\_sensor method), 358  
[SageMakerHook](#) (class in [airflow.contrib.hooks.sagemaker\\_hook](#)), 411  
[set\\_autocommit\(\)](#) (airflow.contrib.hooks.sagemaker\_hook method), 358  
[SageMakerModelOperator](#) (class in [airflow.contrib.operators.sagemaker\\_model\\_operator](#)), 314  
[set\\_autocommit\(\)](#) (airflow.contrib.operators.sagemaker\_model\_operator method), 358  
[SageMakerTrainingOperator](#) (class in [airflow.contrib.operators.sagemaker\\_training\\_operator](#)), 314  
[set\\_default\\_version\(\)](#) (airflow.contrib.operators.sagemaker\_training\_operator method), 358  
[SageMakerTrainingSensor](#) (class in [airflow.contrib.sensors.sagemaker\\_training\\_sensor](#)), 331  
[set\\_dependency\(\)](#) (airflow.models.DAG method), 403  
[SageMakerTransformOperator](#) (class in [airflow.contrib.operators.sagemaker\\_transform\\_operator](#)), 315  
[downstream\(\)](#) (airflow.models.BaseOperator method), 221, 338  
[SageMakerTransformSensor](#) (class in [airflow.contrib.sensors.sagemaker\\_transform\\_sensor](#)), 331  
[set\\_machine\\_type\(\)](#) (airflow.contrib.sensors.sagemaker\_transform\_sensor method), 394  
[SageMakerTuningOperator](#) (class in [airflow.contrib.operators.sagemaker\\_tuning\\_operator](#)), 315  
[set\\_upstream\(\)](#) (airflow.models.BaseOperator method), 221, 338  
[setdefault\(\)](#) (airflow.models.Variable class method), 348  
[SageMakerTuningSensor](#) (class in [airflow.contrib.sensors.sagemaker\\_tuning\\_sensor](#)), 331  
[SFTPHook](#) (class in [airflow.contrib.hooks.sftp\\_hook](#)), 410  
[SalesforceHook](#) (class in [airflow.contrib.hooks.salesforce\\_hook](#)), 416  
[SFTPOperator](#) (class in [airflow.contrib.operators.sftp\\_operator](#)), 311  
[SambaHook](#) (class in [airflow.hooks.samba\\_hook](#)), 364  
[SFTPSensor](#) (class in [airflow.contrib.sensors.sftp\\_sensor](#)), 330  
[schedule\\_interval](#) (airflow.models.BaseOperator attribute), 221, 337  
[SFTPToS3Operator](#) (class in [airflow.contrib.operators.sftp\\_to\\_s3\\_operator](#)), 312  
[SegmentHook](#) (class in [airflow.contrib.hooks.segment\\_hook](#)), 417  
[SegmentTrackEventOperator](#) (class in [airflow.contrib.operators.segment\\_track\\_event\\_operator](#)), 316  
[ShortCircuitOperator](#) (class in [airflow.operators.python\\_operator](#)), 236  
[select\\_key\(\)](#) (airflow.hooks.S3\_hook.S3Hook method), 364  
[shutdown\\_cluster\(\)](#) (airflow.contrib.hooks.cassandra\_hook.CassandraHook method), 378  
[send\\_metric\(\)](#) (airflow.contrib.hooks.datadog\_hook.DatadogHook method), 387  
[sign\\_in\(\)](#) (airflow.contrib.hooks.salesforce\_hook.SalesforceHook method), 416  
[SequentialExecutor](#) (class in [airflow.executors.sequential\\_executor](#)), 429  
[SimpleHttpOperator](#) (class in [airflow.operators.http\\_operator](#)), 236  
[size\(\)](#) (airflow.models.DagBag method), 344  
[set\\_autocommit\(\)](#) (airflow.contrib.hooks.pinot\_hook.PinotDbApiHook method), 406  
[SlackAPIOperator](#) (class in [airflow.operators.slack\\_operator](#)), 237  
[set\\_autocommit\(\)](#) (airflow.contrib.hooks.snowflake\_hook.SnowflakeHook method), 418  
[SlackAPIPostOperator](#) (class in [airflow.operators.slack\\_operator](#)), 237  
[SlackHook](#) (class in [airflow.hooks.slack\\_hook](#)), 364  
[set\\_autocommit\(\)](#) (airflow.hooks.slack\_webhook\_hook.SlackWebhookHook method), 417  
[SlackWebhookHook](#) (class in [airflow.contrib.hooks.slack\\_webhook\\_hook](#)), 417



SlackWebhookOperator (class in *airflow.contrib.operators.slack\_webhook\_operator*), 316  
 SnowflakeHook (class in *airflow.contrib.hooks.snowflake\_hook*), 418  
 SnowflakeOperator (class in *airflow.contrib.operators.snowflake\_operator*), 317  
 SnsPublishOperator (class in *airflow.contrib.operators.sns\_publish\_operator*), 317  
 SparkJDBCHook (class in *airflow.contrib.hooks.spark\_jdbc\_hook*), 418  
 SparkJDBCOperator (class in *airflow.contrib.operators.spark\_jdbc\_operator*), 317  
 SparkSqlHook (class in *airflow.contrib.hooks.spark\_sql\_hook*), 420  
 SparkSqlOperator (class in *airflow.contrib.operators.spark\_sql\_operator*), 318  
 SparkSubmitHook (class in *airflow.contrib.hooks.spark\_submit\_hook*), 421  
 SparkSubmitOperator (class in *airflow.contrib.operators.spark\_submit\_operator*), 319  
 SqliteHook (class in *airflow.hooks.sqlite\_hook*), 364  
 SqliteOperator (class in *airflow.operators.sqlite\_operator*), 238  
 SqlSensor (class in *airflow.sensors.sql\_sensor*), 242  
 SqoopHook (class in *airflow.contrib.hooks.sqoop\_hook*), 423  
 SqoopOperator (class in *airflow.contrib.operators.sqoop\_operator*), 320  
 SSHHook (class in *airflow.contrib.hooks.ssh\_hook*), 411  
 SSHOperator (class in *airflow.contrib.operators.ssh\_operator*), 313  
 start() (*airflow.contrib.executors.kubernetes\_executor.KubernetesExecutor* method), 430  
 start() (*airflow.executors.celery\_executor.CeleryExecutor* method), 428  
 start() (*airflow.executors.dask\_executor.DaskExecutor* method), 429  
 start() (*airflow.executors.local\_executor.LocalExecutor* method), 429  
 start\_instance() (*airflow.contrib.hooks.gcp\_compute\_hook.GceHook* method), 395  
 stop\_instance() (*airflow.contrib.hooks.gcp\_compute\_hook.GceHook* method), 395  
 stop\_query() (*airflow.contrib.hooks.aws\_athena\_hook.AWSAthenaHook* method), 366  
 store\_file() (*airflow.contrib.hooks.ftp\_hook.FTPHook* method), 391  
 store\_file() (*airflow.contrib.hooks.sftp\_hook.SFTPHook* method), 410  
 sub\_dag() (*airflow.models.DAG* method), 342  
 SubDagOperator (class in *airflow.operators.subdag\_operator*), 238  
 subdags (*airflow.models.DAG* attribute), 342  
 submit() (*airflow.contrib.hooks.spark\_submit\_hook.SparkSubmitHook* method), 423  
 submit\_run() (*airflow.contrib.hooks.databricks\_hook.DatabricksHook* method), 386  
 sync() (*airflow.contrib.executors.kubernetes\_executor.KubernetesExecutor* method), 430  
 sync() (*airflow.executors.celery\_executor.CeleryExecutor* method), 428  
 sync() (*airflow.executors.dask\_executor.DaskExecutor* method), 429  
 sync() (*airflow.executors.local\_executor.LocalExecutor* method), 429  
 sync() (*airflow.executors.sequential\_executor.SequentialExecutor* method), 429  
 sync\_to\_db() (*airflow.models.DAG* method), 342  
**T**  
 table\_exists() (*airflow.contrib.hooks.bigquery\_hook.BigQueryHook* method), 375  
 table\_exists() (*airflow.contrib.hooks.cassandra\_hook.CassandraHook* method), 378  
 table\_exists() (*airflow.hooks.hive\_hooks.HiveMetastoreHook* method), 355  
 tar\_and\_s3\_upload() (*airflow.contrib.hooks.sagemaker\_hook.SageMakerHook* method), 415  
 TaskInstance (class in *airflow.models*), 345  
 template\_fields (*airflow.contrib.sensors.ftp\_sensor.FTPSensor* attribute), 326  
 terminate() (*airflow.executors.dask\_executor.DaskExecutor* method), 429  
 test\_cycle() (*airflow.models.DAG* method), 342  
 test\_hql() (*airflow.hooks.hive\_hooks.HiveCliHook* method), 353  
 TimeDeltaSensor (class in *airflow.sensors.time\_delta\_sensor*), 242  
 TimeSensor (class in *airflow.sensors.time\_sensor*), 242  
 to\_csv() (*airflow.hooks.hive\_hooks.HiveServer2Hook* method), 356  
 topological\_sort() (*airflow.models.DAG* method), 343

transform() (airflow.contrib.operators.mongo\_to\_s3.MongoToS3Operator static method), 300  
 tree\_view() (airflow.models.DAG method), 343  
 TriggerDagRunOperator (class in airflow.contrib.operators.dagrun\_operator), 238  
 try\_number (airflow.models.TaskInstance attribute), 347  
 type\_map() (airflow.contrib.operators.mysql\_to\_gcs.MySQLToGoogleCloudStorageOperator class method), 301  
 type\_map() (airflow.contrib.operators.postgres\_to\_gcs.PostgresToGoogleCloudStorageOperator class method), 302

## U

update\_cluster() (airflow.contrib.hooks.gcp\_bigtable\_hook.BigtableHook static method), 377  
 update\_database() (airflow.contrib.hooks.gcp\_spanner\_hook.CloudSpannerHook method), 380  
 update\_endpoint() (airflow.contrib.hooks.sagemaker\_hook.SageMakerHook method), 416  
 update\_function() (airflow.contrib.hooks.gcp\_function\_hook.GcfHook method), 396  
 update\_instance() (airflow.contrib.hooks.gcp\_spanner\_hook.CloudSpannerHook method), 380  
 update\_many() (airflow.contrib.hooks.mongo\_hook.MongoHook method), 405  
 update\_one() (airflow.contrib.hooks.mongo\_hook.MongoHook method), 405  
 update\_state() (airflow.models.DagRun method), 345  
 upload() (airflow.contrib.hooks.gcs\_hook.GoogleCloudStorageHook method), 401  
 upload\_file() (airflow.contrib.hooks.azure\_data\_lake\_hook.AzureDataLakeHook method), 372  
 upload\_function\_zip() (airflow.contrib.hooks.gcp\_function\_hook.GcfHook method), 396  
 upsert\_document() (airflow.contrib.hooks.azure\_cosmos\_hook.AzureCosmosDBHook method), 371  
 upstream\_list (airflow.models.BaseOperator attribute), 221, 338  
 used\_slots() (airflow.models.Pool method), 345

## V

ValueCheckOperator (class in airflow.operators.check\_operator), 238  
 Variable (class in airflow.models), 348

## W

wait\_for\_operation() (airflow.contrib.hooks.gcp\_dataproc\_hook.DataProcHook method), 385  
 wait\_for\_operation() (airflow.contrib.hooks.gcp\_container\_hook.GKEClusterHook method), 393  
 WasbBlobSensor (class in airflow.contrib.sensors.wasb\_sensor), 331  
 WasbDeleteBlobOperator (class in airflow.contrib.operators.wasb\_delete\_blob\_operator), 321  
 WasbHook (class in airflow.contrib.hooks.wasb\_hook), 425  
 WasbPrefixSensor (class in airflow.contrib.sensors.wasb\_sensor), 331  
 WebHDFSHook (class in airflow.hooks.webhdfs\_hook), 364  
 WebHdfsSensor (class in airflow.sensors.web\_hdfs\_sensor), 242  
 WinRMHook (class in airflow.contrib.hooks.winrm\_hook), 427  
 WinRMOperator (class in airflow.contrib.operators.winrm\_operator), 321  
 write\_batch\_data() (airflow.contrib.hooks.aws\_dynamodb\_hook.AwsDynamoDBHook method), 366  
 write\_object\_to\_file() (airflow.contrib.hooks.salesforce\_hook.SalesforceHook method), 416

## X

xcom\_pull() (airflow.models.BaseOperator method), 221, 338  
 xcom\_pull() (airflow.models.TaskInstance method), 348  
 xcom\_push() (airflow.models.BaseOperator method), 221, 338  
 xcom\_push() (airflow.models.TaskInstance method), 348

## Z

ZendeskHook (*class in airflow.hooks.zendesk\_hook*),  
[365](#)