AIP-21: Changes in import paths

Status

State	Completed
Discussion Thread	[VOTE] Changes in import paths
JIRA	AIRFLOW-4733 - Jira project doesn't exist or you don't have permission to view it.
Created	$action.dateFormatter.formatGivenString("yyyy-MM-dd", $content.getCreationDate())
In Release	2.0.0

Motivation

During the 4-year project development, the community made many decisions about the structure and principles of module creation. Some decisions have been made by Airbnb and they no longer apply. Other recommendations were rejected because of the low recognition of the community.

The first chapter discusses the problems that concern modules. This chapter is written from a general perspective. Imagine that there are only packages and modules in the project. Specific lines of code do not matter. The next chapter discusses the implementation details and ways of introducing proposed improvements from the perspective of the source code. Here is the important full content of the files. Chapter “Executive considerations” discusses how to introduce these rules into our codebase. You should look at this chapter from the perspective of the repository. At the end of this document, there are conclusions and a summary.

Chapters containing considerations are divided into specific cases. Each case has a possible solution discussed. Included examples serve to show specific cases and solutions In the real world, many problems overlap, so the solutions and examples also overlap. The examples try to be readable for this purpose are limited only to a given case.

This document assumes you are already familiar with Airflow codebase and may change over time based on feedback.

Every time I write resources, I mean operators, hooks, sensors or another piece of code that is specific to an integration.

This AIP has gone through many changes

This AIP has gone through many changes and it might be confusing trying to get the final conclusions from the sequence of events (it has original voting + set of subsequent updates to it).

However the final agreement to the proposal has been nicely captured in https://github.com/apache/airflow/blob/master/CONTRIBUTING.rst#naming-conventions-for-provider-packages and you should treat it as the current status.

All proposed solutions are backwards compatible.

- Status
- Motivation
- Final results (see voting below)
- Target groups in the providers packages
- Update to the original point D. (2019-10-11).
- Update to include 1.10.* backportability and details about non-cloud-providers package (2019-11-16).
- Architectural considerations
  - Case #1 airflow.contrib.{resources}
  - Case #2 git *_{operator/sensor}{/s}.py
  - Case #3 {aws/azure/gcp}_*
  - Case #4 Separate namespace for resources
  - Case #5 *Operator
  - Case #6 Other isolated cases
- Implementation considerations
  - Case #7
- Summary of the proposal

- Voting
- Reference

Final results (see voting below)

Case 1 **What to do with the " contrib" folder**	**Case 2** **Drop modules *_operator suffix**	**Case 3 + Case 4** **Grouping Cloud Providers operators/sensors/hooks**	**Case 5** ***Operator *Sensor *Hook in class name**	**Case 6** **Isolated cases**	**Case 7** **Deprecation method**
A. Move everything "contrib" "airflow"	**A.** Drop the suffix. Example: • ***airflow.operator.gcp_bigtable_operator.py*** becomes ***airflow.operator.gcp_bigtable.py***.	**D.** Group operators/sensors/hooks in **airflow/providers/<PROVIDER>**/operators (sensors, hooks). Each provider can define its own internal structure of that package. For example in case of "google" provider the packages will be further grouped by "gcp", "gsuite", "core" sub-packages. In case of transfer operators where two providers are involved, the transfer operators will be moved to "source" of the transfer. When there is only one provider as target but source is a database or another non-provider source, the operator is put to the target provider. ***NOTE**! The above decision has been updated during AIP-8 Split Providers into Separate Packages for Airflow 2.0. The rule we apply is "maintainability" rule - i.e. transfer operators are put in the provider, where stakeholders are more likely to have interest in maintaining them.* Non-cloud provider ones are moved to airflow/operators(sensors/hooks). **Drop the prefix.** Examples: AWS operator: • ***airflow/contrib/operators/sns_publish_operator.py*** *becomes* ***airflow/providers/amazon/aws/operators/sns_publish.py*** *Google GCP operator:* • ***airflow/contrib/operators/dataproc_operator.py*** becomes ***airflow/providers/gooogle/cloud/operators/dataproc.py*** Previously GCP-prefixed operator: • **airflow/contrib/operators/*gcp_bigtable_operator.py*** becomes **airflow/providers/google/cloud/operators/*bigtable.py*** *Transfer from GCP:* • ***airflow/contrib/operators/gcs_to_s3_operator.py** becomes **airflow/providers/google/cloud/operators/gcs_to_s3.py*** *MySQL to GCS:* • ***airflow/contrib/operators/mysql_to_gcs_operator.py** becomes **airflow/providers/google/cloud/operators/mysql_to_gcs.py*** *SSH operator:* • **airflow/contrib/operators/*ssh_operator.py*** becomes **airflow/operators/ssh.*py***	**B.** No change - keep Operator /Sensor suffix in class name.	**A.** Make individual decisions of renames for operators that do not follow common conventions used for other operators. Consistency trumps compatibility. Examples: ***DataProcHadoopOperator*** renamed to: ***DataprocHadoopOperator***	**A.** Native python method (with better IDE support and more flexible but a bit more verbose)

Target groups in the providers packages

			Service	**Transfer**
Fundamentals (no change)			airflow.hooks.base_hook airflow.hooks.dbapi_hook airflow.models.baseoperator airflow.sensors.base_sensor_operator airflow.operators.branch_operator airflow.operators.check_operator airflow.operators.dagrun_operator airflow.operators.dummy_operator airflow.operators.generic_transfer airflow.operators.latest_only_operator airflow.operators.subdag_operator airflow.sensors.external_task_sensor airflow.sensors.sql_sensor airflow.sensors.time_delta_sensor airflow.sensors.time_sensor airflow.contrib.sensors.weekday_sensor move to airflow.sensors airflow.operators.bash_operator airflow.contrib.sensors.bash_sensor airflow.operators.python_operator airflow.contrib.sensors.python_sensor	
providers				
	google			

		cloud	airflow.gcp.hooks.automl, airflow.gcp.operators.automl airflow.gcp.hooks.bigquery airflow.gcp.operators.bigquery airflow.gcp.hooks.bigquery_dts airflow.gcp.operators.bigquery_dts airflow.gcp.hooks.bigtable airflow.gcp.operators.bigtable airflow.gcp.hooks.cloud_build airflow.gcp.operators.cloud_build airflow.gcp.hooks.compute airflow.gcp.operators.compute airflow.gcp.hooks.dlp airflow.gcp.operators.dlp airflow.gcp.hooks.dataflow airflow.gcp.operators.dataflow airflow.gcp.hooks.dataproc airflow.gcp.operators.dataproc airflow.gcp.hooks.datastore airflow.gcp.operators.datastore airflow.gcp.hooks.functions airflow.gcp.operators.functions airflow.gcp.hooks.kms airflow.gcp.hooks.kubernetes_engine airflow.gcp.operators.kubernetes_engine airflow.gcp.hooks.mlengine airflow.gcp.operators.mlengine airflow.gcp.hooks.cloud_memorystore airflow.gcp.operators.cloud_memorystore airflow.providers.google.cloud.hooks.natural_language airflow.providers.google.cloud. operators.natural_language airflow.providers.google.cloud..hooks.pubsub airflow.providers.google.cloud..operators. pubsub airflow.gcp.hooks.spanner airflow.gcp.operators.spanner airflow.gcp.hooks.speech_to_text airflow.gcp.operators.speech_to_text airflow.gcp.hooks.cloud_sql airflow.gcp.operators.cloud_sql airflow.gcp.hooks.gcs airflow.gcp.operators.gcs airflow.gcp.hooks.cloud_storage_transfer_service airflow.gcp.operators. cloud_storage_transfer_service airflow.gcp.hooks.tasks airflow.gcp.operators.tasks airflow.gcp.hooks.text_to_speech airflow.gcp.operators.text_to_speech airflow.gcp.hooks.translate airflow.gcp.operators.translate airflow.gcp.hooks.video_intelligence airflow.gcp.operators.video_intelligence airflow.providers.google.cloud.hooks.vision	airflow.operators.cassandra_to_gcs, airflow.operators.adls_to_gcs, airflow.contrib. operators.s3_to_gcs_operator, airflow.gcp.operators.cloud_storage_transfer_service, airflow.operators.adls_to_gcs airflow.contrib.operators.s3_to_gcs_operator,airflow.gcp.operators. cloud_storage_transfer_service airflow.operators.bigquery_to_bigquery airflow.operators.bigquery_to_gcs airflow.operators.bigquery_to_mysql airflow.operators.cassandra_to_gcs airflow.operators.gcs_to_bq airflow.operators.gcs_to_gcs,airflow.gcp.operators.cloud_storage_transfer_service airflow.operators.local_to_gcs airflow.operators.mssql_to_gcs airflow.operators.mysql_to_gcs airflow.operators.postgres_to_gcs airflow.operators.sql_to_gcs airflow.operators.gcs_to_sftp
		gsuite	airflow.contrib.hooks.gdrive_hook airflow.gcp.hooks.gsheets	airflow.contrib.operators.gcs_to_gdrive_operator
		marketing_platform	airflow.providers.google.marketing_platform.hooks.campaign_manager airflow.providers. google.marketing_platform.operators.campaign_manager airflow.providers.google. marketing_platform.sensors.campaign_manager airflow.providers.google. marketing_platform.hooks.display_video airflow.providers.google.marketing_platform. operators.display_video airflow.providers.google.marketing_platform.sensors.display_video airflow.providers.google.marketing_platform.hooks.search_ads airflow.providers.google. marketing_platform.operators.search_ads airflow.providers.google.marketing_platform. sensors.search_ads	
	amazon			
		aws	airflow.contrib.hooks.aws_dynamodb_hook airflow.contrib.hooks.aws_glue_catalog_hook airflow.contrib.hooks.aws_logs_hook airflow.contrib.hooks.emr_hook airflow.contrib.hooks.sagemaker_hook airflow.contrib.operators.ecs_operator airflow.contrib.operators.emr_add_steps_operator airflow.contrib.operators.emr_create_job_flow_operator airflow.contrib.operators.emr_terminate_job_flow_operator airflow.contrib.operators.s3_copy_object_operator airflow.contrib.operators.s3_delete_objects_operator airflow.contrib.operators.s3_list_operator airflow.contrib.operators.sagemaker_base_operator airflow.contrib.operators.sagemaker_endpoint_config_operator airflow.contrib.operators.sagemaker_endpoint_operator airflow.contrib.operators.sagemaker_model_operator airflow.contrib.operators.sagemaker_training_operator airflow.contrib.operators.sagemaker_transform_operator airflow.contrib.operators.sagemaker_tuning_operator airflow.contrib.sensors.aws_glue_catalog_partition_sensor airflow.contrib.sensors.emr_base_sensor airflow.contrib.sensors.emr_job_flow_sensor airflow.contrib.sensors.emr_step_sensor airflow.contrib.sensors.sagemaker_base_sensor airflow.contrib.sensors.sagemaker_endpoint_sensor airflow.contrib.sensors.sagemaker_training_sensor airflow.contrib.sensors.sagemaker_transform_sensor airflow.contrib.sensors.sagemaker_tuning_sensor airflow.operators.s3_file_transform_operator airflow.providers.amazon.aws.hooks.athena airflow.providers.amazon.aws.hooks.datasync airflow.providers.amazon.aws.hooks.kinesis airflow.providers.amazon.aws.hooks.lambda_function airflow.providers.amazon.aws.hooks.redshift airflow.providers.amazon.aws.hooks.s3 airflow.providers.amazon.aws.hooks.sns airflow.providers.amazon.aws.hooks.sqs airflow.providers.amazon.aws.operators.athena airflow.providers.amazon.aws.operators.batch airflow.providers.amazon.aws.operators.datasync airflow.providers.amazon.aws.operators.sns airflow.providers.amazon.aws.operators.sqs airflow.providers.amazon.aws.sensors.athena airflow.providers.amazon.aws.sensors.redshift airflow.providers.amazon.aws.sensors.sqs airflow.sensors.s3_key_sensor airflow.sensors.s3_prefix_sensor	airflow.contrib.operators.hive_to_dynamodb, airflow.operators. google_api_to_s3_transfer, airflow.contrib.operators.hive_to_dynamodb, airflow. operators.redshift_to_s3_operator, airflow.operators.s3_to_hive_operator, airflow. operators.s3_to_redshift_operator, airflow.contrib.operators.dynamodb_to_s3, airflow. contrib.operators.s3_to_sftp_operator, airflow.contrib.operators.sftp_to_s3_operator, airflow.operators.gcs_to_s3, airflow.contrib.operators.imap_attachment_to_s3_operator, airflow.contrib.operators.mongo_to_s3, airflow.operators.google_api_to_s3_transfer, airflow.operators.gcs_to_s3
	microsoft			
		azure	airflow.contrib.hooks.wasb_hook airflow.contrib.operators.wasb_delete_blob_operator airflow.contrib.sensors.wasb_sensor airflow.contrib.hooks.azure_container_instance_hook, airflow.contrib.hooks.azure_container_registry_hook, airflow.contrib.hooks. azure_container_volume_hook airflow.contrib.operators. azure_container_instances_operator airflow.contrib.hooks.azure_cosmos_hook airflow. contrib.operators.azure_cosmos_operator airflow.contrib.sensors.azure_cosmos_sensor airflow.contrib.hooks.azure_data_lake_hook airflow.contrib.operators.adls_list_operator airflow.contrib.hooks.azure_fileshare_hook,	airflow.contrib.operators.file_to_wasb, airflow.contrib.operators. oracle_to_azure_data_lake_transfer
	apache			
		cassandra	airflow.contrib.hooks.cassandra_hook airflow.contrib.sensors.cassandra_record_sensor, airflow.contrib.sensors.cassandra_table_sensor	
		druid	airflow.hooks.druid_hook airflow.contrib.operators.druid_operator,airflow.operators. druid_check_operator	airflow.operators.hive_to_druid
		hadoop	airflow.hooks.hdfs_hook airflow.sensors.hdfs_sensor, airflow.contrib.sensors.hdfs_sensor, airflow.hooks.webhdfs_hook airflow.sensors.web_hdfs_sensor	

		hive	airflow.hooks.hive_hooks airflow.operators.hive_operator,airflow.operators.hive_stats_operator airflow.sensors.named_hive_partition_sensor, airflow.sensors.hive_partition_sensor,airflow.sensors.metastore_partition_sensor	airflow.operators.mssql_to_hive, airflow.operators.s3_to_hive_operator, airflow.contrib.operators.vertica_to_hive
		pig	airflow.hooks.pig_hook airflow.operators.pig_operator	
		pinot	airflow.contrib.hooks.pinot_hook	
		spark	airflow.contrib.hooks.spark_jdbc_hook, airflow.contrib.hooks.spark_jdbc_script,airflow.contrib.hooks.spark_sql_hook, airflow.contrib.hooks.spark_submit_hook airflow.contrib.operators.spark_jdbc_operator, airflow.contrib.operators.spark_sql_operator,airflow.contrib.operators.spark_submit_operator	
		sqoop	airflow.contrib.hooks.sqoop_hook airflow.contrib.operators.sqoop_operator	
	mysql			airflow.operators.hive_to_mysql, airflow.contrib.operators.presto_to_mysql
	jira		airflow.contrib.hooks.jira_hook airflow.contrib.operators.jira_operator airflow.contrib.sensors.jira_sensor	
	databricks		airflow.contrib.hooks.databricks_hook airflow.contrib.operators.databricks_operator	
	datadog		airflow.contrib.hooks.datadog_hook airflow.contrib.sensors.datadog_sensor	
	dingding		airflow.contrib.hooks.dingding_hook airflow.contrib.operators.dingding_operator	
	discord		airflow.contrib.hooks.discord_webhook_hook airflow.contrib.operators.discord_webhook_operator	
	cloudant		airflow.contrib.hooks.cloudant_hook	
	jenkins		airflow.contrib.hooks.jenkins_hook airflow.contrib.operators.jenkins_job_trigger_operator	
	opsgenie		airflow.contrib.hooks.opsgenie_alert_hook airflow.contrib.operators.opsgenie_alert_operator	
	qubole		airflow.contrib.hooks.qubole_hook,airflow.contrib.hooks.qubole_check_hook airflow.contrib.operators.qubole_operator,airflow.contrib.operators.qubole_check_operator airflow.contrib.sensors.qubole_sensor	
	salesforce		airflow.contrib.hooks.salesforce_hook	
	segment		airflow.contrib.hooks.segment_hook airflow.contrib.operators.segment_track_event_operator	
	slack		airflow.hooks.slack_hook,airflow.contrib.hooks.slack_webhook_hook airflow.operators.slack_operator,airflow.contrib.operators.slack_webhook_operator	
	snowflake		airflow.contrib.hooks.snowflake_hook airflow.contrib.operators.snowflake_operator	
	vertica		airflow.contrib.hooks.vertica_hook airflow.contrib.operators.vertica_operator	airflow.contrib.operators.vertica_to_mysql
	zendesk		airflow.hooks.zendesk_hook	
	celery		airflow.contrib.sensors.celery_queue_sensor	
	docker		airflow.hooks.docker_hook airflow.operators.docker_operator, airflow.contrib.operators.docker_swarm_operator	
	kubernetes		airflow.contrib.operators.kubernetes_pod_operator	
	mssql		airflow.hooks.mssql_hook airflow.operators.mssql_operator	
	mongodb		airflow.contrib.hooks.mongo_hook airflow.contrib.sensors.mongo_sensor	
	mysql		airflow.hooks.mysql_hook airflow.operators.mysql_operator	
	openfaas		airflow.contrib.hooks.openfaas_hook	
	oracle		airflow.hooks.oracle_hook airflow.operators.oracle_operator	airflow.contrib.operators.oracle_to_oracle_transfer
	papermill		airflow.operators.papermill_operator	
	postgres		airflow.hooks.postgres_hook airflow.operators.postgres_operator	
	presto		airflow.hooks.presto_hook airflow.operators.presto_check_operator	
	redis		airflow.contrib.hooks.redis_hook airflow.contrib.operators.redis_publish_operator airflow.contrib.sensors.redis_pub_sub_sensor,airflow.contrib.sensors.redis_key_sensor	
	samba		airflow.hooks.samba_hook	airflow.operators.hive_to_samba_operator
	sqlite		airflow.hooks.sqlite_hook	airflow.operators.sqlite_operator
	imap		airflow.contrib.hooks.imap_hook airflow.contrib.sensors.imap_attachment_sensor	
	ssh		airflow.contrib.hooks.ssh_hook airflow.contrib.operators.ssh_operator	
	filesystem		airflow.contrib.hooks.fs_hook airflow.contrib.sensors.file_sensor	
	sftp		airflow.contrib.hooks.sftp_hook airflow.contrib.operators.sftp_operator airflow.contrib.sensors.sftp_sensor	
	ftp		airflow.contrib.hooks.ftp_hook airflow.contrib.sensors.ftp_sensor	
	http		airflow.hooks.http_hook airflow.operators.http_operator airflow.sensors.http_sensor	
	grpc		airflow.contrib.hooks.grpc_hook airflow.contrib.operators.grpc_operator	
	smtp		airflow.operators.email_operator	
	jdbc		airflow.hooks.jdbc_hook airflow.operators.jdbc_operator	
	winrm		airflow.contrib.hooks.winrm_hook airflow.contrib.operators.winrm_operator	

Update to the original point D. (2019-10-11).

During implementation of AIP-23 we found that the original decision about grouping operators was not the best and did not cover all the scenarios. Therefore we updated the rules as follows:

- Grouping by cloud provider should be done in "airflow/**providers**" package (previously it was directly in "airflow"
- Each provider can have different internal structure, potentially grouping the operators in sub-packages. For example in case of "google" provider the packages will be further grouped by "gcp", "gsuite", "core" sub-packages.
- In case of transfer operators where two providers are involved, the transfer operators will be moved to ~~"source"~~ (NOTE it's been changed to "target" in subsequent Update) of the transfer. When there is only one provider as target but source is a database or another non-provider source, the operator is put to the target provider.

Update to include 1.10.* backportability and details about non-cloud-providers package (2019-11-16).

In the light of coming Airflow 2.0 release the community decided there is a need to make it easier to migrate from Airflow 1.10 to the upcoming 2.0 release. Airflow 2.0 is - by definition - not backwards-compatible with 1.10.* series. There are a number of incompatibilities that are introduced - in core, database, concepts but also in parameters of a number of operators integrating with external services/software. DAGs written for Airflow 1.10.* might not work out-of-the-box in Airflow 2.0. We have not yet figured out if we are going to provide some automated migration, but we can provide a mechanism to switch to Airflow 2.0 "provider" set of operators and hooks even when still running Airflow 1.10. That can make migration process easier as organisation doing the migration might do it in steps. There are some organisations that still use python 2 even though Airflow 2.0 supports only python 3.5+ (possibly 3.6+ in the final 2.0 release). We figured out that most of the new/updated operators to be released in airflow 2.0 which were not cherry-picked to Airflow 1.10 are still runnable in Airflow 1.10. and you should be able to start using those operators in Airflow 1.10 in parallel to old operators. We decided to move most of the non-core operators to new packages (all inside "providers" package and release them as separate packages that will be installable in Python 3.5+ Airflow 1.10* releases). POC for that is available here:

https://github.com/apache/airflow/pull/6507

Therefore the migration process might look as follows.

(1) Python 2.7 + Airflow 1.10.* (2) Python 3.6 + Airflow 1.10.* (3) Python 3.6 + Airflow 1.10.* + switch to using "providers" operators (4) Python 3.6 + Airflow 2.0

Switching to the new "providers" operators can be mostly automated and it can be done incrementally for the DAGs a company has (we can provide some scripts for that). Each of the steps can be done separately in it's own pace.

This will make it easier for companies to move to Airflow 2.0 as well as it might provide an early testing ground for all the operators/hooks/sensors which are only present in Airflow 2.0 and have incompatible changes.

The list of all Airflow operators/sensors/hooks is above in target_groups

Architectural considerations

It is based on widely accepted rules, and also shows cases when these rules are not followed. I will also show ideas for improving these principles.

Case #1 airflow.contrib.{resources}

There should be one-- and preferably only one --obvious way to do it.

Tim Peters, The Zen of Python

Currently, resources are located in two places:

airflow.{resource_type}

airflow.contrib.{resource_type}

In the first place are resources that were originally maintained by Airbnb. However, they have been transferred to Apache and Airbnb is not responsible for their maintenance. The community is responsible for maintaining them. In the second place are operators that are maintained by the community from the beginning until now. Currently, all new resources are added only to the second place. The changes and development of the first place are strictly limited.

There is currently no reason for this type of division. All resources should be in one place.

Solution A:

We should move all the resources from the first place to the second. All resources will be located in airflow.{hooks/operators/sensors/example_dags}.

Advantages	Disadvantages

- resources are **located in one place** (and one place only). No need to check multiple locations for docs for example. - **no confusion for new contributors** whether their work needs to be managed differently. (New contributors shouldn't wonder if there is a difference between their work and non-contrib work. Because there shouldn't be one!)	- resources moved from contrib to core has to be tested before moved. Outdated hooks/operators need to be updated or removed. Unit tests for all need to be added if it doesn't already exists.

~~**Solution B:**~~

~~Move all the well-tested and maintained resources to the core for e.g GCP resources are well-tested with good documentation. All the new resources need to be first added to contrib folder and once they reach "maturity" they can be moved to core. We need to define what is that maturity. Contrib resources would be analogous to beta features in a product. We should also consider changing the words "contrib" to "incubator" in this situation.~~

~~Advantages~~	~~Disadvantages~~
~~- resources in core can be trusted by people and contributors take full responsibility of those resources.~~ ~~- resources in contrib are subject to change.~~	~~- resources needs to be maintained at 2 places and can be a source of confusion for new contributors.~~

Solution C:

No change.

Case #2 git *_{operator/sensor}{/s}.py

Currently, the import takes the following format:

```
airflow{.contrib/}.operators.*_operator
```

There is information redundancy here. There is no need to use the word "operator" twice

It is worth mentioning that the word "operator" also appears in the class name

Solution A:

The import should take the following format:

```
airflow{.contrib/}.operators.*
```

Suffix "_operator" should be dropped

Example:
File *airflow/contrib/operators/gcp_bigtable_operator.py* should be moved to *airflow/contrib/operators/gcp_bigtable.py*.

Advantages	Disadvantages
- Shorter name, but still focussing on the essential task of the class (no information loss)	-

~~**Solution B:**~~

~~No change~~

~~Advantages and disadvantages are analogous to solution A.~~

Case #3 {aws/azure/gcp}_*

With the development of integration for the largest cloud providers, a large part of new files received a prefix, which is assigned to each of them. For example, for Google Cloud Platform it is "gcp". Google mentioned the practice even in official recommendations[1]. Not all files follow this rule. Ansible also uses similar structure.

~~**Solution A:**~~

~~The prefix can be completely dropped. Major provider can get a separate sub-module for each type of resource.~~

~~Operators that integrate with two services will not change.~~

~~Example:~~
~~File *airflow/contrib/operators/gcp_bigtable_operator.py* should be moved to *airflow/contrib/operators/gcp/bigtable_operator.py*.~~

~~The package format will look like this:~~
~~*airflow/{contrib/}{resource_type}/{supplier/}bigtable_operator.py*~~

~~Advantages~~	~~Disadvantages~~
~~- shorter name, but still focussing on the essential task of the class (no information loss)~~ ~~- users only need to look at their _supplier_ package instead of lot of other _supplier_'s services at once. (Most users probably use only one supplier at a time)~~ ~~(This could also speed up navigating through the documentation for users depending on how the documentation is structured)~~	~~• it's a bit easier to find files when the file name contains relevant gcp_* for example in most IDE's. This is however very weak argument as most of the IDEs will also support gcp/* as prefix when looking for a file~~

~~**Solution B:**~~

~~The prefix will be completed for incompatible files~~

~~Example:~~
~~File */airflow/contrib/operators/sns_publish_operator.py* should be moved to */airflow/contrib/operators/aws_sns_publish_operator.py*~~
~~File */airflow/contrib/operators/dataproc_operator.py* should be moved to */airflow/contrib/operators/gcp_dataproc_operator.py*~~

~~Operators that integrate with two services will not change.~~

Solution C:

This solution has been reported by ashb

The prefix can be completely dropped. Major provider will get their own sub-module, which will contain all types of resources.

This change forces the adoption of a solution A from Case #1 airflow.contrib.{resources} at the same time.

The package format will look like this:.
airflow/integration/{supplier}/{resource_type}/bigtable_operator.py

Advantages	Disadvantages
This way the integration package contains everything from a supplier and you won't have multiple same supplier packages for hooks, operators, macros, etc. Moreover it would be simpler to move such an integration to a separate repository. (See AIP-8)	-

~~**Solution D:**~~

~~The prefix can be completely dropped. Major provider will get their own sub-module, which will contain all types of resources. Other operators will be moved to one module (ex. core).~~

~~This change forces the adoption of a solution A from Case #1 airflow.contrib.{resources} at the same time.~~

~~The package format will look like this:.~~
~~*airflow_integration/{resource_type}/gcp_bigtable_operator.py*~~

~~Example:~~
~~File */airflow/contrib/operators/sns_publish_operator.py* should be moved to */airflow_integration/aws/operators/aws_sns_publish_operator.py*~~
~~File */airflow/operators/bash_operator.py* should be moved to */airflow_integration/core/bash_operator.py*~~

~~Case #4 Separate namespace for resources~~

~~*Namespaces are one honking great idea -- let's do more of those!*~~

~~*Tim Peters, The Zen of Python*~~

~~Note - we do not move the namespaces out. It's mereWe can create a new namespace for all resources. We will not take advantage of all the possibilities that it offers, but in the future it will be possible to easily switch to a separate package for group of resource.~~

~~This solution should also be considered taking into account the acceptance of solution D from Case #3 {aws/azure/gcp}_*~~

~~Example of a project that uses a separate namespace: https://github.com/googleapis/google-cloud-python~~

~~Note: This change does not introduce separated packages for group of resources. It tries to retain only compatibility. Details are available: AIP-8 Split Hooks/Operators into Separate Packages by Tim Swast.~~

~~The package format will look like this:.~~
~~*airflow_resources/{category}/{resource_type}/bigtable_operator.py*~~

~~**Solution #A:**~~

~~We can introduce namespaces.~~

~~Advantages~~	~~Disadvantages~~
~~We will avoid changing import paths in the future~~	~~-~~

Solution #B:

We reject introduction namespaces.

Advantages and disadvantages are analogous to solution A

Note that grouping remains as if in namespaces (but this is merely a package not a separate namespace),

Case #5 *Operator

Class name does not need suffix "Operator"

~~**Solution A:**~~

~~We can delete the suffix “Operator” from the class name~~

~~Example:~~
~~Class GcpTransferServiceJobDeleteOperator should be renamed to GcpTransferServiceJobDelete.~~

~~Advantages~~	~~Disadvantages~~
~~- Shorter name, but still focussing on the essential task of the class (no information loss)~~	~~-~~

Solution B:

No change

Advantages and disadvantages are analogous to solution A.

Case #6 Other isolated cases

There are other random cases of inconsistencies in the naming of classes. It is necessary to review the list of all classes and prepare a plan of change. Support from major cloud service providers will be useful.

For example:
Google Dataproc operators:

Current name	Proposition of new name
DataProcHadoopOperator	DataprocHadoopOperator
DataProcHiveOperator	DataprocHiveOperator
DataProcPigOperator	DataprocPigOperator
DataProcPySparkOperator	DataprocPySparkOperator
DataProcSparkOperator	DataprocSparkOperator
DataProcSparkSqlOperator	DataprocSparkSqlOperator
DataprocClusterCreateOperator	No change
DataprocClusterDeleteOperator	No change

DataprocClusterScaleOperator	No change
DataprocWorkflowTemplateBaseOperator	No change
DataprocWorkflowTemplateInstantiateInlineOperator	No change
DataprocWorkflowTemplateInstantiateOperator	No change
GoogleCloudStorageToS3Operator	GcsToS3Operator

This document does not analyze such cases. It can be one area of analysis by other groups of people ex. employees of the largest cloud service providers.

Any such change must be considered individually when accepting pull requests. Each change must be consistent with the recommendations made after voting on the changes in this document.

Implementation considerations

Case #7

Developer perspective - source code, and console view from both methods is available: https://imgur.com/a/mRaWpQm

Repository with samples: https://github.com/mik-laj/airflow-deprecation-sample

Solution #A native python

Advantages	Disadvantages
Its supported by IDE. More flexible - we can add a link to the documentation	Files must exist in the project - temporary mess. More code in the project (226 characters. vs 78 character = +189%).

Sample PR: https://github.com/apache/airflow/pull/4667

~~**Solution #B zope.deprecation/sys.modules hack**~~

~~Solution proposed by @ashb~~

~~Advantages~~	~~Disadvantages~~
~~Less boilerplate code.~~	~~It is NOT supported by IDE.~~ ~~Files must exist in the project - temporary mess.~~ ~~No configuration options~~

Executive considerations

We can introduce the proposed changes in two ways:

1. as one commit;
2. as many commits for each group of operators;

The first method will be faster to perform, but finding one bug (if it would appear) in such a patch will be very difficult. The introduced change should, therefore, be made a series of corrections.

Each change should contain one commit. Each PR and commit should be described in the format: “[AIRFLOW-XXX]”

Summary of the proposal

Green are the voted options

Choice	Case 1 **What to do with the " contrib" folder**	Case 2 **Drop modules *_operator suffix**	Case 3 + Case 4 **Grouping Cloud Providers operators /sensors/hooks**	Case 5 ***Operator *Sensor *Hook in class name**	Case 6 **Isolated cases**	Case 7 **Deprecation method**
A	Move everything "co ntrib" "airflow"	Drop the suffix. Example: • ***airflow.operator.gcp_bigtable_operator.py*** becomes ***airflow.operator.gcp_bigtable.py***.	Keep operators/sensors/hooks in airflow /operators(sensors, hooks) and keep/add prefixes in file names. • ***airflow/contrib/operators/sns_publish_operator.py*** *becomes* ***airflow/operators/aws_sns_publish_operator.py*** • ***airflow/contrib/operators/dataproc_operator.py*** becomes ***airflow/operators/gcp_dataproc_operator.py*** • **airflow/*contrib*/hooks/*gcp_bigtable_hook.py*** becomes **airflow/hooks/gcp_*bigtable_hook.py*** • **airflow/*contrib*/operators/*ssh_operator.py*** becomes **airflow/operators/ssh_operator*.py***	Remove the Operator suffix from class name. Examples: • ***GcpTransferServiceJobDeleteOperator*** rename to ***GcpTransferServiceJobDelete*** • ***BashOperator*** rename to **Bash**	Make individual decisions of renames for operators that do not follow common conventions used for other operators. Consistency trumps compatibility. Examples: ***DataProcHadoopOperator*** renamed to: ***DataprocHadoopOperator***	Native python method (with better IDE support and more flexible but a bit more verbose)
B	Move well tested code "c ontrib" "airflow" Rename "contrib" to "inc ubator" for less-well tested code.	No change. Example: • ***gcp_bigtable_operator.py*** stays ***gcp_bigtable_operator.py***	Group operators/sensors/hooks in airflow /operators(sensors, hooks)/**<PROVIDER>.** Non-cloud provider ones are moved to airflow /operators(sensors/hooks). **Drop the prefix.** • ***airflow/contrib/operators/sns_publish_operator.py*** *becomes* ***airflow/operators/aws/sns_publish_operator.py*** • ***airflow/contrib/operators/dataproc_operator.py*** becomes ***airflow/operators/gcp/dataproc_operator.py*** • **airflow/*contrib*/operators/*gcp_bigtable_operator.py*** becomes **airflow/operators/gcp/*bigtable_operator.py*** • **airflow/*contrib*/operators/*ssh_operator.py*** becomes **airflow/operators/ssh_operator*.py***	No change - keep Operator/Sensor suffix in class name.	Avoid renaming operators due to better backwards compatibility.	Use zope.deprecation (less IDE support, less verbose, less flexibility)
C	No change		Group operators/sensors/hooks in airflow /operators(sensors, hooks)**/<PROVIDER>.** Non-cloud provider ones are moved to airflow /operators(sensors/hooks). **Keep the prefix.** • ***airflow/contrib/operators/sns_publish_operator.py*** *becomes* ***airflow/operators/aws/aws_sns_publish_operator.py*** • ***airflow/contrib/operators/dataproc_operator.py*** becomes ***airflow/operators/gcp/gcp_dataproc_operator.py*** • **airflow/*contrib*/operators/*gcp_bigtable_operator.py*** becomes **airflow/operators/gcp/gcp_*bigtable_operator.py*** • **airflow/*contrib*/operators/*ssh_operator.py*** becomes **airflow/operators/ssh_operator*.py***			

D			Group operators/sensors/hooks in **airflow /<PROVIDER>**/operators(sensors, hooks). Non-cloud provider ones are moved to airflow /operators(sensors/hooks). **Drop the prefix.** • ***airflow/contrib/operators /sns_publish_operator.py*** *becomes* ***airflow/aws/operators/sns_publish_operator.py*** • ***airflow/contrib/operators /dataproc_operator.py*** becomes ***airflow/gcp/operators /dataproc_operator.py*** • **airflow/*contrib*/operators/*gcp_bigtable_operator.py*** becomes **airflow/*gcp*/operators/*bigtable_operator.py*** • **airflow/*contrib*/operators/*ssh_operator.py*** becomes **airflow/operators/ssh_operator.py**			
E			Group operators/sensors/hooks in **airflow /<PROVIDER>**/operators(sensors, hooks). Non-cloud provider ones are moved to airflow /operators(sensors/hooks). **Keep the prefix**. • ***airflow/contrib/operators /sns_publish_operator.py*** *becomes* ***airflow/aws/operators/aws_sns_publish_operator.py*** • ***airflow/contrib/operators /dataproc_operator.py*** becomes ***airflow/gcp/operators/gcp_dataproc_operator.py*** • **airflow/*contrib*/operators/*gcp_bigtable_operator.py*** becomes **airflow/*gcp*/operators/gcp_*bigtable_operator.py*** • **airflow/*contrib*/operators/*ssh_operator.py*** becomes **airflow/operators/ssh_operator.py**			

Voting

Feel free to add your votes below:

Person	Binding	Case 1 What to do with the " contrib" folder	Case 2 Drop modules *_operator suffix	Case 3 + Case 4 Grouping Cloud Providers operators /sensors/hooks	Case 5 *Operator *Sensor *Hook in class name	Case 6 Isolated cases	Case 7 Deprecation method
Jarek Potiuk	Yes	**A**: Move everything "contrib" "airflow"	**A**. gcp_bigtable_operator.py gcp_bigtable.py	**D**. airflow/contrib/operators/gcp_bigtable_operator.py airflow/gcp/operators/bigtable_operator.py	**B**. No changes. Keep *Operator *Sensor *Hook in class name	**A**. Rename isolated cases for consistency.	**A**. Native python with better IDE integration.
Ash Berlin-Taylor	Yes	**A**	**A**	**D**	**B** - it's clearer at call site (`task = XOperator()` vs `task = X()`)	No opinion	No strong opionin
Fokko Driesprong	Yes	**A**	**A**	**A**	**B**	**A**	**A**
Felix Uellendall	No	**A**	**A**	**D**	**B**	**A**	No opinion
Kamil Bregula	Yes	**A**	**A**	**D**	**B**	**A**	**A**
Kaxil Naik	Yes	**A**	**A**	**D**	**B**	**A**	**A**
Bas Harenslak	Yes	**A**	**A**	**A**	**B**	**A**	**A**

Philippe Gagnon	No	**A**	**A**	**D**	**B**	**A**	**A**

Any strong "vetos" on any of the answers please record it here with justification:

Person	Binding	Case 1 What to do with the " contrib" folder	Case 2 Drop modules *_operator suffix	Case 3 Separate out module's Cloud Provider prefixes (gcp/aws /azure) to packages	Case 4 Introduce separate namespaces for different resources	Case 5 *Operator *Sensor *Hook in class name	Case 6 Isolated cases	Case 7 Deprecation method
Ash Berlin-Taylor					Vetoing A with the prefix of `airflow_resources` , but don't object to `airflow.gcp.operator.x` grouping.			

Original votes on Case 3/Case 4:

	Case 3 Separate out module's Cloud Provider prefixes (gcp/aws /azure) to packages	Case 4 Introduce separate namespaces for different resources
Jarek Potiuk	**C**. gcp_bigtable_operator.py gcp /operators/bigtable.py	**B**. No namespaces introduction.
Ash Berlin-Taylor	No opinion	
Fokko Driesprong	**B**	**B**
Felix Uellendall	**C**	**B**
Kamil Bregula	**C**	**B**
Daniel Standish		
Kaxil Naik	**C**	**A** - `airflow.gcp.operator.*` `It will make it more organised. Anyone willing to find out Airflow support for a particular cloud provider would just need to look into a single folder.`
Chen Tong		
Bas Harenslak	**B**	**B**

The Voting mechanism:

Voting will take place till **Tuesday** 30 Jul 2019 6pm CEST (5 pm BST, 9am PST) .

After the choice, the final consistent set of choices will be announced (taking into account majority of binding vote, also including potential vetos and consistency between the choices. Non-binding votes will be taken into account in case there is a draw. The final set of choices will be announced at de vlist thread after the voting completes.

Unless there is a veto raised to the final proposal, the final proposal is accepted by Lazy Consensus on **Friday** 02 Aug 2019 at 6pm CEST (5 pm BST, 9am PST).

Reference

fenglu@google.com. 2018. GCP Service Airflow Integration Guide. [ONLINE] Available at: https://lists.apache.org/thread.html/e8534d82be611ae7bcb21ba371546a4278aad117d5e50361fd8f14fe@%3Cdev.airflow.apache.org%3E. [Accessed 8 February 2019].