[Creating a custom Operator](https://airflow.apache.org/docs/stable/howto/howto/custom-operator.html)[¶](https://airflow.apache.org/docs/stable/howto/custom-operator.html" \l "creating-a-custom-operator" \o "Permalink to this headline)

Airflow allows you to create new operators to suit the requirements of you or your team. The extensibility is one of the many reasons which makes Apache Airflow powerful.

You can create any operator you want by extending the [**airflow.models.baseoperator.BaseOperator**](https://airflow.apache.org/docs/stable/_api/airflow/models/baseoperator/index.html#airflow.models.baseoperator.BaseOperator)

There are two methods that you need to override in a derived class:

* Constructor - Define the parameters required for the operator. You only need to specify the arguments specific to your operator. Use **@apply\_defaults** decorator function to fill unspecified arguments with **default\_args**. You can specify the **default\_args** in the dag file. See [Default args](https://airflow.apache.org/docs/stable/concepts.html#default-args) for more details.
* Execute - The code to execute when the runner calls the operator. The method contains the airflow context as a parameter that can be used to read config values.

Let’s implement an example **HelloOperator** in a new file **hello\_operator.py**:

from airflow.models.baseoperator import BaseOperator

from airflow.utils.decorators import apply\_defaults

class HelloOperator(BaseOperator):

@apply\_defaults

def \_\_init\_\_(

self,

name: str,

\*args, \*\*kwargs) -> None:

super().\_\_init\_\_(\*args, \*\*kwargs)

self.name = name

def execute(self, context):

message = "Hello {}".format(self.name)

print(message)

return message

Note

For imports to work, you should place the file in a directory that is present in the **PYTHONPATH** env. Airflow adds **dags/**, **plugins/**, and **config/** directories in the Airflow home to **PYTHONPATH** by default. e.g., In our example, the file is placed in the **custom\_operator** directory.

You can now use the derived custom operator as follows:

from custom\_operator.hello\_operator import HelloOperator

with dag:

hello\_task = HelloOperator(task\_id='sample-task', name='foo\_bar')

Hooks[¶](https://airflow.apache.org/docs/stable/howto/custom-operator.html#hooks)

Hooks act as an interface to communicate with the external shared resources in a DAG. For example, multiple tasks in a DAG can require access to a MySQL database. Instead of creating a connection per task, you can retrieve a connection from the hook and utilize it. Hook also helps to avoid storing connection auth parameters in a DAG. See [Managing Connections](https://airflow.apache.org/docs/stable/howto/connection/index.html) for how to create and manage connections.

Let’s extend our previous example to fetch name from MySQL:

class HelloDBOperator(BaseOperator):

@apply\_defaults

def \_\_init\_\_(

self,

name: str,

mysql\_conn\_id: str,

database: str,

\*args, \*\*kwargs) -> None:

super().\_\_init\_\_(\*args, \*\*kwargs)

self.name = name

self.mysql\_conn\_id = mysql\_conn\_id

self.database = database

def execute(self, context):

hook = MySqlHook(mysql\_conn\_id=self.mysql\_conn\_id,

schema=self.database)

sql = "select name from user"

result = hook.get\_first(sql)

message = "Hello {}".format(result['name'])

print(message)

return message

When the operator invokes the query on the hook object, a new connection gets created if it doesn’t exist. The hook retrieves the auth parameters such as username and password from Airflow backend and passes the params to the [**airflow.hooks.base\_hook.BaseHook.get\_connection()**](https://airflow.apache.org/docs/stable/_api/airflow/hooks/base_hook/index.html#airflow.hooks.base_hook.BaseHook.get_connection). You should create hook only in the **execute** method or any method which is called from **execute**. The constructor gets called whenever Airflow parses a DAG which happens frequently. The **execute** gets called only during a DAG run.

User interface[¶](https://airflow.apache.org/docs/stable/howto/custom-operator.html#user-interface)

Airflow also allows the developer to control how the operator shows up in the DAG UI. Override **ui\_color** to change the background color of the operator in UI. Override **ui\_fgcolor** to change the color of the label.

class HelloOperator(BaseOperator):

ui\_color = '#ff0000'

ui\_fgcolor = '#000000'

....

Templating[¶](https://airflow.apache.org/docs/stable/howto/custom-operator.html#templating)

You can use [Jinja templates](https://airflow.apache.org/docs/stable/concepts.html#id1) to parameterize your operator. Airflow considers the field names present in **template\_fields** for templating while rendering the operator.

class HelloOperator(BaseOperator):

template\_fields = ['name']

@apply\_defaults

def \_\_init\_\_(

self,

name: str,

\*args, \*\*kwargs) -> None:

super().\_\_init\_\_(\*args, \*\*kwargs)

self.name = name

def execute(self, context):

message = "Hello from {}".format(name)

print(message)

return message

You can use the template as follows:

with dag:

hello\_task = HelloOperator(task\_id='task\_id\_1', dag=dag, name='{{ task\_id }}')

In this example, Jinja looks for the **name** parameter and substitutes **{{ task\_id }}** with **task\_id\_1**.

The parameter can also contain a file name, for example, a bash script or a SQL file. You need to add the extension of your file in **template\_ext**. If a **template\_field** contains a string ending with the extension mentioned in **template\_ext**, Jinja reads the content of the file and replace the templates with actual value. Note that Jinja substitutes the operator attributes and not the args.

class HelloOperator(BaseOperator):

template\_fields = ['guest\_name']

@apply\_defaults

def \_\_init\_\_(

self,

name: str,

\*args, \*\*kwargs) -> None:

super().\_\_init\_\_(\*args, \*\*kwargs)

self.guest\_name = name

In the example, the **template\_fields** should be **['guest\_name']** and not **['name']**

Define an operator extra link[¶](https://airflow.apache.org/docs/stable/howto/custom-operator.html#define-an-operator-extra-link)

For your operator, you can [Define an extra link](https://airflow.apache.org/docs/stable/howto/define_extra_link.html) that can redirect users to external systems. For example, you can add a link that redirects the user to the operator’s manual.