El siguiente código define la clase MyBasicMathOperator. Este operador hereda del BaseOperator y puede realizar aritmética cuando le proporciona dos números y una operación. Este código se guarda en la carpeta include de un archivo llamado basic_math_operator.py.

```
from airflow.models.baseoperator import BaseOperator

class MyBasicMathOperator(BaseOperator):
    """
    Example Operator that does basic arithmetic.
    :param first_number: first number to put into an equation
    :param second_number: second number to put into an equation
    :param operation: mathematical operation to perform
    """

    # provide a list of valid operations
    valid_operations = ("+", "-", "*", "/")
    # define which fields can use Jinja templating
    template_fields = ("first_number", "second_number")

    def __init__(
        self,
        first_number: float,
        second_number: float,
        operation: str,
        *args,
        **kwargs,
    ):
```

```
        super().__init__(*args, **kwargs)
        self.first_number = first_number
        self.second_number = second_number
        self.operation = operation

        # raise an import error if the operation provided is not valid
        if self.operation not in self.valid_operations:
            raise ValueError(
                f"{self.operation} is not a valid operation. Choose one of {self.valid_operations}"
            )

    def execute(self, context):
        self.log.info(
            f"Equation: {self.first_number} {self.operation} {self.second_number}"
        )
        if self.operation == "+":
            res = self.first_number + self.second_number
            self.log.info(f"Result: {res}")
            return res
        if self.operation == "-":
            res = self.first_number - self.second_number
            self.log.info(f"Result: {res}")
            return res
        if self.operation == "*":
            res = self.first_number * self.second_number
```

```
            self.log.info(f"Result: {res}")
            return res
        if self.operation == "/":
            try:
                res = self.first_number / self.second_number
            except ZeroDivisionError as err:
                self.log.critical(
                    "If you have set up an equation where you are trying to divide by zero, you have done something WRONG. - Randall Munroe, 2006"
                )
                raise ZeroDivisionError

            self.log.info(f"Result: {res}")
            return res
```

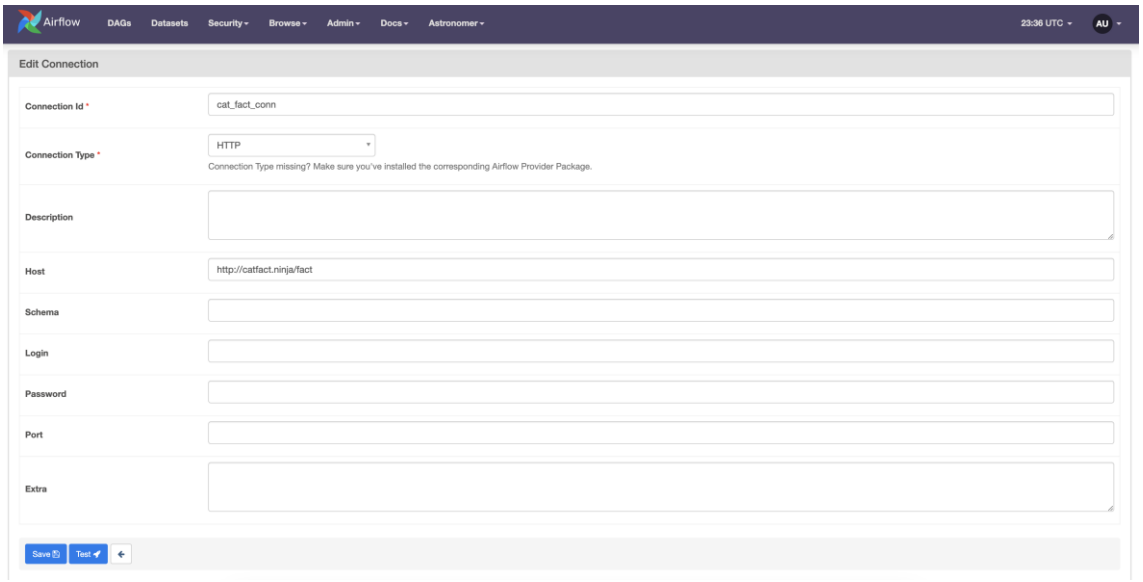
Además del operador personalizado, el DAG de ejemplo utiliza un enlace personalizado para conectarse a CatFactAPI. Este enlace resume la recuperación de la URL de la API desde una conexión de Airflow y realiza varias llamadas a la API en un bucle. Este código también debe colocarse en el directorio include en un archivo llamado cat_fact_hook.py.

```
"""This module allows you to connect to the CatFactAPI."""

from airflow.hooks.base import BaseHook

import requests as re



class CatFactHook(BaseHook):
    """
    Interact with the CatFactAPI.
```

```
Performs a connection to the CatFactAPI and retrieves a cat fact client.

:cat_fact_conn_id: Connection ID to retrieve the CatFactAPI url.
"""

conn_name_attr = "cat_conn_id"
default_conn_name = "cat_conn_default"
conn_type = "http"
hook_name = "CatFact"

def __init__(
    self, cat_fact_conn_id: str = default_conn_name, *args, **kwargs
) -> None:
    super().__init__(*args, **kwargs)
    self.cat_fact_conn_id = cat_fact_conn_id
    self.get_conn()

def get_conn(self):
    """Function that initiates a new connection to the CatFactAPI."""

    # get the connection object from the Airflow connection
    conn = self.get_connection(self.cat_fact_conn_id)

    # return the host URL
    return conn.host
```

```
    def log_cat_facts(self, number_of_cat_facts_needed: int = 1):
        """Function that logs between 1 to 10 catfacts depending on its input."""
        if number_of_cat_facts_needed < 1:
            self.log.info(
                "You will need at least one catfact! Setting request number to 1."
            )
            number_of_cat_facts_needed = 1
        if number_of_cat_facts_needed > 10:
            self.log.info(
                f"{number_of_cat_facts_needed} are a bit many. Setting request number to 10."
            )
            number_of_cat_facts_needed = 10

        cat_fact_connection = self.get_conn()

        # log several cat facts using the connection retrieved
        for i in range(number_of_cat_facts_needed):
            cat_fact = re.get(cat_fact_connection).json()
            self.log.info(cat_fact["fact"])
        return f"{i} catfacts written to the logs!"
```

Para utilizar este enlace personalizado, debe crear una conexión Airflow con el ID de conexión cat_fact_conn, el tipo de conexión HTTPy el Host http://catfact.ninja/fact.



Luego puede importar el operador personalizado y el hook personalizado a su DAG.

```
from pendulum import datetime

from airflow import DAG

from airflow.operators.python import PythonOperator

from include.basic_math_operator import MyBasicMathOperator

from include.cat_fact_hook import CatFactHook



def use_cat_fact_hook(number):

    num_catfacts_needed = round(number)

    # instatiating a CatFactHook at runtime of this task

    hook = CatFactHook("cat_fact_conn")

    hook.log_cat_facts(num_catfacts_needed)
```

```
with DAG(
    dag_id="my_math_cat_dag",
    schedule_interval="@daily",
    start_date=datetime(2021, 1, 1),
    # render Jinja template as native Python object
    render_template_as_native_obj=True,
    catchup=False,
):
    add = MyBasicMathOperator(
        task_id="add",
        first_number=23,
        second_number=19,
        operation="+",
        # any BaseOperator arguments can be used with the custom operator too
        doc_md="Addition Task.",
    )

    multiply = MyBasicMathOperator(
        task_id="multiply",
        # use the return value from the add task as the first_number, pulling from XCom
        first_number="{{ ti.xcom_pull(task_ids='add', key='return_value') }}",
        second_number=35,
        operation="-",
    )
```

```
use_cat_fact_hook_task = PythonOperator(
    task_id="use_cat_fact_hook",
    python_callable=use_cat_fact_hook,
    op_args=[multiply.output],
)

add >> multiply >> use_cat_fact_hook_task
```