PRÁCTICA COMPOSER

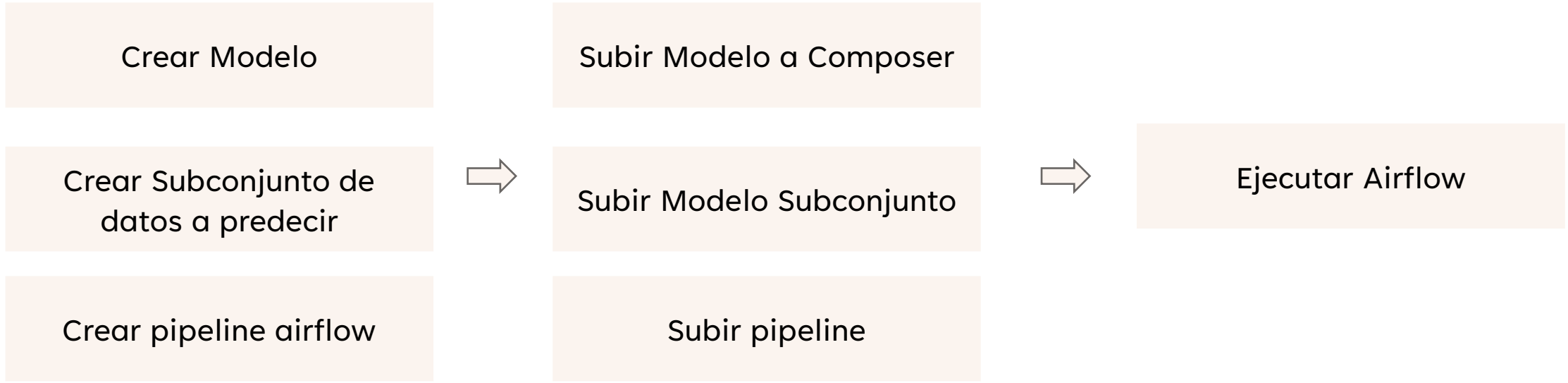
Carlos Moragón y Paloma Pérez de Madrid

ÍNDICE

ÍNDICE

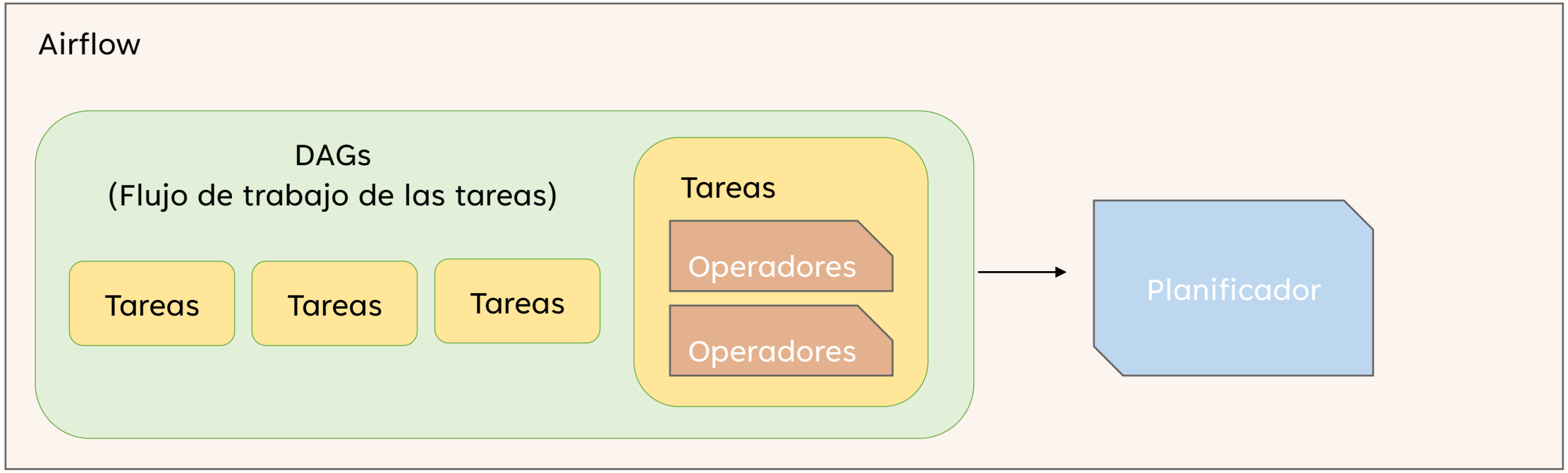
1. OBJETIVO

Crear un pipline con Airflow mediante el servicio de Google Composer para predecir un subconjunto de datos en base a un modelo ya generado



1. OBJETIVO

Crear un pipline con Airflow mediante el servicio de Google Composer para predecir un subconjunto de datos en base a un modelo ya generado



1. OBJETIVO

Crear un pipline con Airflow mediante el servicio de Google Composer para predecir un subconjunto de datos en base a un modelo ya generado de h2o

¿Qué vamos a hacer?

```
########################################################################## ESTABLECEMOS DEPENDENCIAS #####


t_begin  >> t1_fichero_existe>>t_condicion
t_condicion>>t_hub_start
t_condicion>>t_end
t_hub_start>>t_2_subir_scoring>>t_3_flag>>t_end

#########################################################################################################
```

Operadores

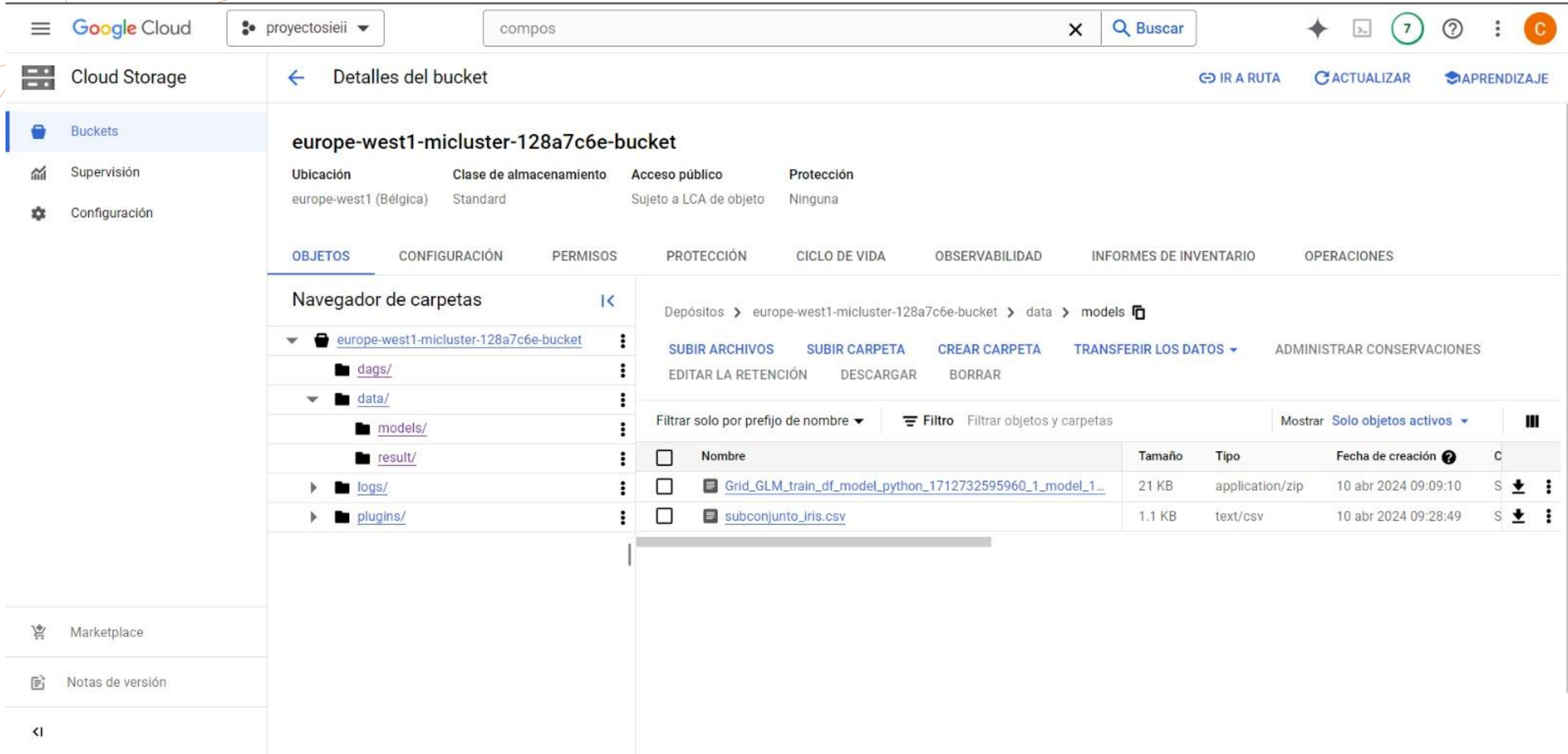
ÍNDICE

METODOLOGÍA

1. Generar modelo con el dataset Iris y h2o → “modelo_iris_ipynb”
2. Guardar el modelo → En el archivo modelo_iris,ipynb, lo guarderemos con model.save_mojo(‘./ruta’) y se generará el siguiente archivo: “Grid_GLM_train_df_model_python_1712732595960_1_model_10.zip”
3. Crear un subconjunto de Iris y guardarlo en el bucket de composer → “subconjunto_iris.csv”

Ahora subiremos el modelo y el subconjunto al bucket de Composer

Composer >> micluster >> Dags


Google Cloud
proyectosieii
compos
Buscar
Cloud Storage
Buckets
Supervisión
Configuración
Marketplace
Notas de versión
Detalles del bucket
IR A RUTA
ACTUALIZAR
APRENDIZAJE
europe-west1-micluster-128a7c6e-bucket
Ubicación
europe-west1 (Bélgica)
Clase de almacenamiento
Standard
Acceso público
Sujeto a LCA de objeto
Protección
Ninguna
OBJETOS
CONFIGURACIÓN
PERMISOS
PROTECCIÓN
CICLO DE VIDA
OBSERVABILIDAD
INFORMES DE INVENTARIO
OPERACIONES
Navegador de carpetas
europe-west1-micluster-128a7c6e-bucket
dags/
data/
models/
result/
logs/
plugins/
Depósitos > europe-west1-micluster-128a7c6e-bucket > data > models
SUBIR ARCHIVOS
SUBIR CARPETA
CREAR CARPETA
TRANSFERIR LOS DATOS
ADMINISTRAR CONSERVACIONES
EDITAR LA RETENCIÓN
DESCARGAR
BORRAR
Filtrar solo por prefijo de nombre
Filtro
Filtrar objetos y carpetas
Mostrar
Solo objetos activos
Nombre
Tamaño
Tipo
Fecha de creación
Grid_GLM_train_df_model_python_1712732595960_1_model_1...
21 KB
application/zip
10 abr 2024 09:09:10
subconjunto_iris.csv
1.1 KB
text/csv
10 abr 2024 09:28:49

METODOLOGÍA

4. Crear archivo del pipeline de airflow

```
import os.path
from airflow import models
from airflow.utils.dates import days_ago
from datetime import datetime, timedelta, date
from airflow.contrib.operators import gcs_to_bq
from airflow.contrib.operators import bigquery_to_gcs
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator, BranchPythonOperator
import pandas as pd
from airflow.contrib.sensors.file_sensor import FileSensor
```

METODOLOGÍA

4. Crear archivo del pipeline de airflow

```
## Nuestro proyecto
gcs_project = 'dark-yen-414408'
dataset = 'clase'
## Bucket de GCS. La ruta /home/airflow/gcs/ está mapeada al bucket que se va a generar cuando creamos nuestro Composer
gcs_airflow = 'europe-west1-micluster-128a7c6e-bucket'

tabla_resultado = 'elscoring'


def scoring_modelo():
    # Inicializa H2O
    h2o.init()
    # Carga el subconjunto de datos desde el archivo CSV
    sub_iris_h2o = h2o.import_file("europe-west1-micluster-128a7c6e-bucket/data/models/subconjunto_iris.csv")
    # europe-west1-micluster-128a7c6e-bucket/data/models/subconjunto_iris.csv
    modelo = h2o.upload_mojo("europe-west1-micluster-128a7c6e-bucket/data/models/Grid_GLM_train_df_model_python_1712732595960_1_model_10.zip")
    predictions = modelo.predict(sub_iris_h2o)

    # Guarda las predicciones en un DataFrame de pandas
    predictions_df = predictions.as_data_frame()

    # Guarda las predicciones en un archivo CSV
    predictions_df.to_csv('europe-west1-micluster-128a7c6e-bucket/data/result/predicciones.csv', index=False)
```

METODOLOGÍA

4. Crear archivo del pipeline de airflow

```
def flag_file (flagfile):
    file = open(flagfile, "w+")
    file.write("Ejecutado \n")
    file.close()

def get_path (filepath):
    existe = os.path.exists(filepath)
    if existe:
        return 'end'
    else:
        return 'start'


default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(3),
    'schedule_interval': "5 * * * *",
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 2,
    'catchup': True,
    'retry_delay': timedelta(minutes=2),
    'project_id': gcs_project
}
```

METODOLOGÍA

4. Crear archivo del pipeline de airflow

```
## Bucket de GCS. La ruta /home/airflow/gcs/ está mapeada al bucket que se va a generar cuando creamos nuestro Composer

# Define DAG: Set ID and assign default args and schedule interval
with models.DAG('NadiePasaDeEstaEsquinaAquiMandanLasDivinasPorQueSomosGasolinaGasolinaDeVerdad',
        default_args=default_args
) as mydag:
    start_time = datetime.now()
    year, week_num, day_of_week = start_time.isocalendar()

    semana_actual = 'W-{0}-{1}'.format(year, week_num)

    t_begin = DummyOperator(task_id="begin", dag=mydag)
    t_end = DummyOperator(task_id = 'end', dag=mydag)
    t_hub_start = DummyOperator(task_id="start", dag=mydag)

    t_condicion = BranchPythonOperator(task_id='continue_branch', dag=mydag, python_callable=get_path,
                                       op_kwargs={
                                           'filepath': "/home/airflow/gcs/data/flag_file_" + semana_actual + ".txt"})

    t1_fichero_existe = FileSensor(task_id="comprobar_scoring_ejecutado",
                                   poke_interval=30,
                                   filepath='europe-west1-micluster-128a7c6e-bucket/data/result/predicciones.csv', dag=mydag)

    ## Las rutas se podían haber cargado de un fichero de configuración. Al ser un ejercicio conceptual, lo dejo así.
```

METODOLOGÍA

4. Crear archivo del pipeline de airflow

```
          t_3_flag = PythonOperator(
           task_id='create_flag',
           python_callable=flag_file,
           op_kwargs={'flagfile': "/home/airflow/gcs/data/flag_file_"+semana_actual+".txt"
                   },
           dag=mydag
     )

        t_2_subir_scoring = PythonOperator(
            task_id='scoring_modelo',
            python_callable=scoring_modelo,
            op_kwargs={
            },
            dag=mydag

        )

################################################ ESTABLECEMOS DEPENDENCIAS ##########################################################

    t_begin  >> t1_fichero_existe>>t_condicion
    t_condicion>>t_hub_start
    t_condicion>>t_end
    t_hub_start>>t_2_subir_scoring>>t_3_flag>>t_end

  ###################################################################################################################################
```

METODOLOGÍA

- Importante: antes de subir el archivo del pipe del airflow tenemos que instalar h2o en el cluster (micluster>>paquetesPyPi>>h20=3.36.0.3 en nuestro caso)