

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

Alumno: Oswaldo Gael Cervantes Castoño

Código: 219747468

Computación Tolerante a Fallas

Sección D06

Profesor: Michel Emanuel López Franco



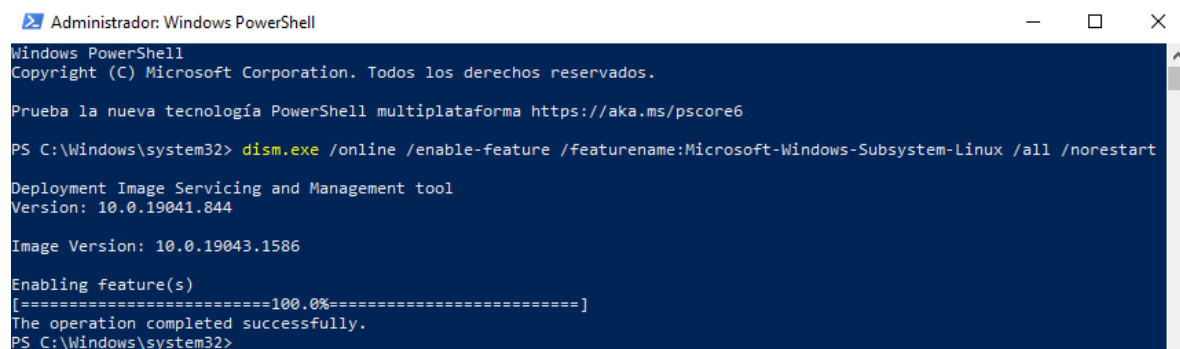
Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Objetivo:

Reporte en el que muestres un ejemplo utilizando Airflow.

Desarrollo:

El primer paso para el desarrollo de esta actividad es habilitar WSL (Windows Subsystem for Linux) ejecutando Powershell como administrador y utilizando el siguiente comando y para observar los cambios aplicados debemos reiniciar el equipo una vez terminado el proceso:



```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

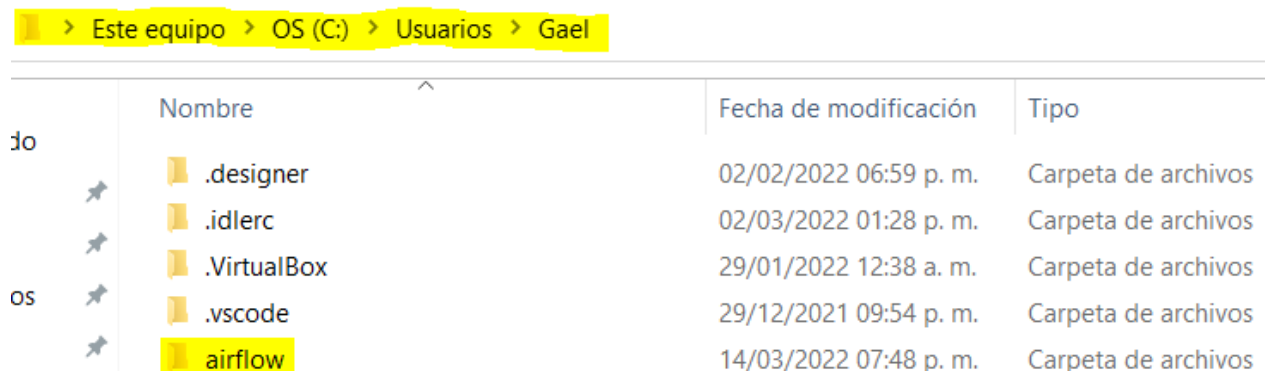
PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

Deployment Image Servicing and Management tool
Version: 10.0.19041.844

Image Version: 10.0.19043.1586

Enabling feature(s)
[=====100.0%=====]
The operation completed successfully.
PS C:\Windows\system32>
```

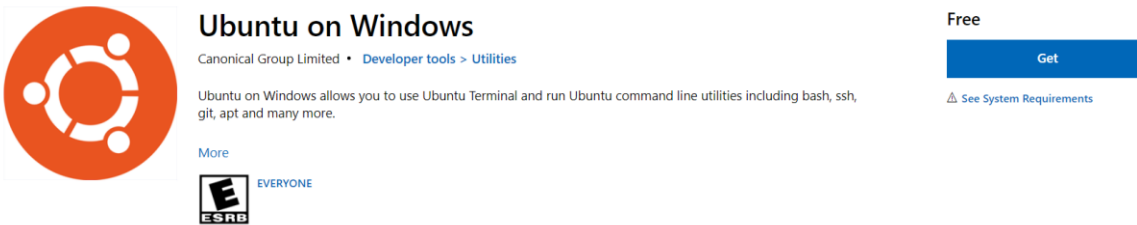
Lo siguiente es crear una carpeta en la siguiente ruta: C:/Users/Gael/ llamada “airflow” para instalar Apache Airflow y guardar los DAGs creados después de la instalación:



Este equipo > OS (C:) > Usuarios > Gael			
	Nombre	Fecha de modificación	Tipo
do	.designer	02/02/2022 06:59 p. m.	Carpeta de archivos
	.idlerc	02/03/2022 01:28 p. m.	Carpeta de archivos
os	.VirtualBox	29/01/2022 12:38 a. m.	Carpeta de archivos
	.vscode	29/12/2021 09:54 p. m.	Carpeta de archivos
	airflow	14/03/2022 07:48 p. m.	Carpeta de archivos

Tenemos que instalar una distribución de Linux para poder instalar Apache Airflow, en este caso yo la voy a instalar desde la Microsoft Store:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.



Una vez que lo instalamos procedemos a ejecutarlo y una vez que se inicia después de unos minutos creamos un usuario y contraseña:

```
gael@LAPTOP-37AFC7C7: ~  
Installing, this may take a few minutes...  
Please create a default UNIX user account. The username does not need to match your Windows username.  
For more information visit: https://aka.ms/wslusers  
Enter new UNIX username: gael  
New password:  
Retype new password:  
passwd: password updated successfully  
Installation successful!  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 4.4.0-19041-Microsoft x86_64)
```

Ejecutamos los comandos “sudo apt update && sudo apt upgrade” para asegurarnos de que todo se encuentra actualizado al momento; para proceder necesitamos introducir la contraseña:

```
gael@LAPTOP-37AFC7C7:~$ sudo apt update && sudo apt upgrade  
[sudo] password for gael:  
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]  
Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease  
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]  
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]  
Get:5 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]  
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [1316 kB]  
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [230 kB]  
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [9772 B]  
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [799 kB]  
25% [5 Packages 1501 kB/8628 kB 17%] [9 Packages 83.3 kB/799 kB 10%]
```

Con el comando que se encuentra en la siguiente imagen podemos utilizar el editor de nano para modificar un archivo de configuración que se sugiere en el proceso de instalación. Con CTRL+S guardamos y salimos con CTRL+X:

```
airflow@LAPTOP-37AFC7C7:~$ sudo nano /etc/wsl.conf
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
airflow@LAPTOP-37AFC7C7: ~  
GNU nano 4.8 /etc/wsl.conf  
[automount]  
root = /  
options = "metadata" _
```

Para que los cambios que realizamos en el punto anterior tengan efecto, tenemos que cerrar sesión en nuestro usuario actual de Windows y volver a ingresar para posteriormente ejecutar Ubuntu de nuevo.

Ahora procedemos a revisar que Python se encuentre instalado y lo podemos corroborar con la siguiente imagen posterior a ejecutar el comando:

```
airflow@LAPTOP-37AFC7C7:~$ sudo apt install python3  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python3 is already the newest version (3.8.2-0ubuntu2).  
python3 set to manually installed.  
The following package was automatically installed and is no longer required:  
  libfwupdplugin1  
Use 'sudo apt autoremove' to remove it.  
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.  
airflow@LAPTOP-37AFC7C7:~$ python3 --version  
Python 3.8.10  
airflow@LAPTOP-37AFC7C7:~$
```

Ahora procedemos a ejecutar de nuevo el comando “sudo apt update” para poder actualizar el entorno al momento:

```
gael@LAPTOP-37AFC7C7:~$ sudo apt update  
[sudo] password for gael:  
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease  
Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease  
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]  
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]  
Get:5 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [14.8 kB]  
Get:6 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [20.3 kB]  
Fetched 257 kB in 3s (78.3 kB/s)  
_
```

Lo siguiente es instalar pip, lo cual podemos hacer utilizando el comando que se muestra en la siguiente imagen:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
airflow@LAPTOP-37AFC7C7:~$ sudo apt install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfwupdplugin1
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cpp cpp-9 dpkg-dev fakeroot g++ g++-9 gcc gcc-9
  gcc-9-base libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libatomic1 libbinutils
  libc-dev-bin libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libdpkg-perl libexpat1-dev libfakeroot
  libfile-fcntllock-perl libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0 libmpc3 libpython3-dev libpython3.8-dev
  libquadmath0 libstdc++-9-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev python-pip-whl python3-dev
  python3-wheel python3.8-dev zlib1g-dev
```

Instalamos Apache Airflow:

```
airflow@LAPTOP-37AFC7C7:~$ pip3 install apache-airflow
Collecting apache-airflow
  Downloading apache_airflow-2.2.4-py3-none-any.whl (5.3 MB)
    | 5.3 MB 2.6 MB/s
Collecting flask-caching<2.0.0,>=1.5.0
  Downloading Flask_Caching-1.10.1-py3-none-any.whl (34 kB)
Collecting apache-airflow-providers-http
  Downloading apache_airflow_providers_http-2.1.0-py3-none-any.whl (21 kB)
Collecting pendulum~=2.0
  Downloading pendulum-2.1.2-cp38-cp38-manylinux1_x86_64.whl (155 kB)
    | 155 kB 2.5 MB/s
Collecting python-daemon>=2.2.4
  Downloading python_daemon-2.3.0-py2.py3-none-any.whl (35 kB)
Collecting flask-appbuilder==3.4.4
  Downloading Flask_AppBuilder-3.4.4-py3-none-any.whl (1.9 MB)
    | 1.9 MB 887 kB/s
Collecting setproctitle<2,>=1.1.8
```

El siguiente paso es muy importante ya que consiste en definir una variable de entorno que ayudará a Airflow a encontrar recursos tales como los DAGs y el archivo de configuración de Airflow, así que definimos dicha variable por medio del siguiente comando:

```
airflow@LAPTOP-37AFC7C7:~$ export AIRFLOW_HOME=/c/Users/Gael/airflow
airflow@LAPTOP-37AFC7C7:~$
```

Ahora tenemos que agregar esta misma variable de entorno de forma permanente en el sistema para no tener que definirla todo el tiempo como en el paso anterior, para lo cual tenemos que hacer uso del comando **nano ~/.bashrc** y una vez abierto el archivo de configuración tenemos que agregar la línea mencionada en el punto anterior en la parte superior del fichero.

Una vez hecho lo anterior, cerramos Ubuntu y lo volvemos a ejecutar.

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Para confirmar que lo que hemos hecho hasta ahora funciona correctamente, verificamos la versión de Airflow con el siguiente comando:

```
gael@LAPTOP-37AFC7C7: ~  
gael@LAPTOP-37AFC7C7:~$ airflow version  
2.2.4  
gael@LAPTOP-37AFC7C7:~$
```

Verificamos el valor de la variable de entorno definida anteriormente con respecto al directorio HOME de Airflow como se hace en la siguiente imagen:

```
gael@LAPTOP-37AFC7C7:~$ echo $AIRFLOW_HOME  
/c/Users/Gael/airflow  
gael@LAPTOP-37AFC7C7:~$
```

Ya estamos listos para ejecutar Airflow y el primer paso es utilizar el comando “airflow db init” para inicializar la base de datos de Apache Airflow:

```
gael@LAPTOP-37AFC7C7:~$ airflow db init  
DB: sqlite:///c/Users/Gael/airflow/airflow.db  
[2022-03-14 13:27:23,183] {db.py:919} INFO - Creating tables  
INFO [alembic.runtime.migration] Context impl SQLiteImpl.  
INFO [alembic.runtime.migration] Will assume non-transactional DDL.  
INFO [alembic.runtime.migration] Running upgrade -> e3a246e0dc1, current schema  
INFO [alembic.runtime.migration] Running upgrade e3a246e0dc1 -> 1507a7289a2f, create is_encrypted  
/home/gael/.local/lib/python3.8/site-packages/alembic/ddl/sqlite.py:74 UserWarning: Skipping unsupported ALTER for creat  
ion of implicit constraint. Please refer to the batch mode feature which allows for SQLite migrations using a copy-and-m  
ove strategy.  
INFO [alembic.runtime.migration] Running upgrade 1507a7289a2f -> 13eb55f81627, maintain history for compatibility with  
earlier migrations  
INFO [alembic.runtime.migration] Running upgrade 13eb55f81627 -> 338e90f54d61, More logging into task_instance  
INFO [alembic.runtime.migration] Running upgrade 338e90f54d61 -> 52d714495f0, job_id indices  
INFO [alembic.runtime.migration] Running upgrade 52d714495f0 -> 502898887f84, Adding extra to Log  
INFO [alembic.runtime.migration] Running upgrade 502898887f84 -> 1b38cef5b76e, add dagrun  
INFO [alembic.runtime.migration] Running upgrade 1b38cef5b76e -> 2e541a1dcfed, task_duration  
INFO [alembic.runtime.migration] Running upgrade 2e541a1dcfed -> 40e67319e3a9, dagrun_config  
INFO [alembic.runtime.migration] Running upgrade 40e67319e3a9 -> 561833c1c74b, add password column to user  
INFO [alembic.runtime.migration] Running upgrade 561833c1c74b -> 4446e08588, dagrun start end  
INFO [alembic.runtime.migration] Running upgrade 4446e08588 -> bbc73705a13e, Add notification_sent column to sla_miss  
INFO [alembic.runtime.migration] Running upgrade bbc73705a13e -> bba5a7cfc896, Add a column to track the encryption sta  
te of the 'Extra' field in connection  
INFO [alembic.runtime.migration] Running upgrade bba5a7cfc896 -> 1968acfc09e3, add is_encrypted column to variable tabl  
e  
INFO [alembic.runtime.migration] Running upgrade 1968acfc09e3 -> 2e82aab8ef20, rename user table
```

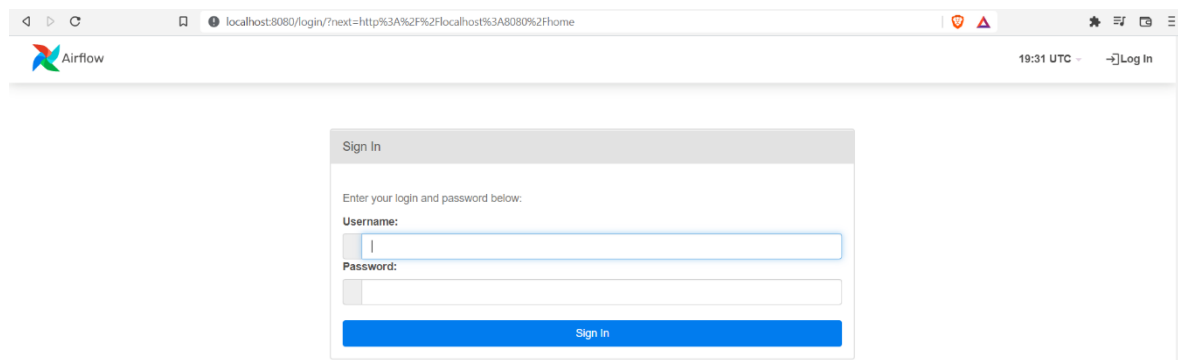
Ahora utilizamos el comando “airflow webserver” para levantar el servidor de Apache Airflow y poder visualizar la interfaz desde nuestro localhost en el puerto 8080:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
gael@LAPTOP-37AFC7C7:~$ airflow webserver

[2022-03-14 13:29:20,325] {dagbag.py:500} INFO - Filling up the DagBag from /dev/null
[2022-03-14 13:29:20,377] {manager.py:779} WARNING - No user yet created, use flask fab command to do it.
[2022-03-14 13:29:20,488] {manager.py:496} INFO - Created Permission View: menu access on List Users
[2022-03-14 13:29:20,501] {manager.py:558} INFO - Added Permission menu access on List Users to role Admin
[2022-03-14 13:29:20,526] {manager.py:496} INFO - Created Permission View: menu access on Security
[2022-03-14 13:29:20,540] {manager.py:558} INFO - Added Permission menu access on Security to role Admin
[2022-03-14 13:29:20,582] {manager.py:496} INFO - Created Permission View: menu access on List Roles
[2022-03-14 13:29:20,597] {manager.py:558} INFO - Added Permission menu access on List Roles to role Admin
[2022-03-14 13:29:20,629] {manager.py:496} INFO - Created Permission View: can read on User Stats Chart
[2022-03-14 13:29:20,643] {manager.py:558} INFO - Added Permission can read on User Stats Chart to role Admin
[2022-03-14 13:29:20,667] {manager.py:496} INFO - Created Permission View: menu access on User's Statistics
[2022-03-14 13:29:20,680] {manager.py:558} INFO - Added Permission menu access on User's Statistics to role Admin
[2022-03-14 13:29:20,725] {manager.py:496} INFO - Created Permission View: menu access on Base Permissions
[2022-03-14 13:29:20,736] {manager.py:558} INFO - Added Permission menu access on Base Permissions to role Admin
[2022-03-14 13:29:20,776] {manager.py:496} INFO - Created Permission View: can read on View Menus
[2022-03-14 13:29:20,790] {manager.py:558} INFO - Added Permission can read on View Menus to role Admin
[2022-03-14 13:29:20,815] {manager.py:496} INFO - Created Permission View: menu access on Views/Menus
[2022-03-14 13:29:20,829] {manager.py:558} INFO - Added Permission menu access on Views/Menus to role Admin
[2022-03-14 13:29:20,867] {manager.py:496} INFO - Created Permission View: can read on Permission Views
[2022-03-14 13:29:20,880] {manager.py:558} INFO - Added Permission can read on Permission Views to role Admin
[2022-03-14 13:29:20,906] {manager.py:496} INFO - Created Permission View: menu access on Permission on Views/Menus
[2022-03-14 13:29:20,918] {manager.py:558} INFO - Added Permission menu access on Permission on Views/Menus to role Admin
```

Accedemos a localhost:8080 y podemos ver que se nos solicita la creación de un usuario para poder continuar:

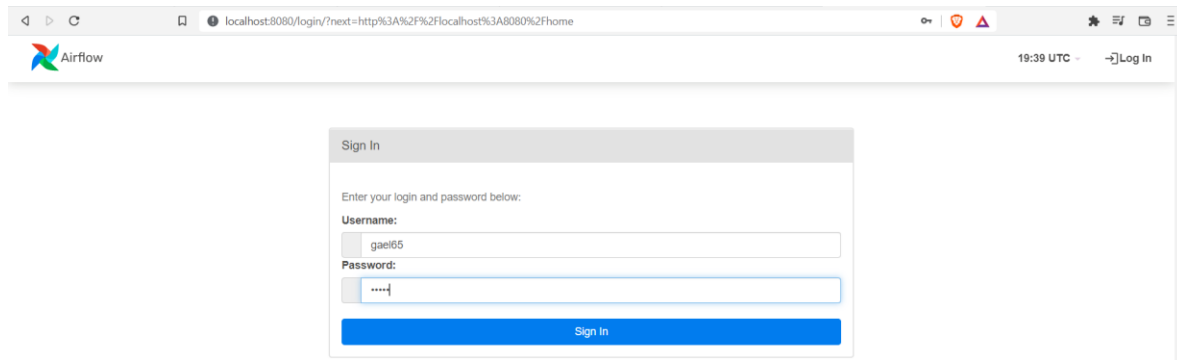


Utilizamos la siguiente secuencia de comandos para la creación de nuestro usuario (para lo cual primero tuvimos que cancelar el servicio de apache webserver)

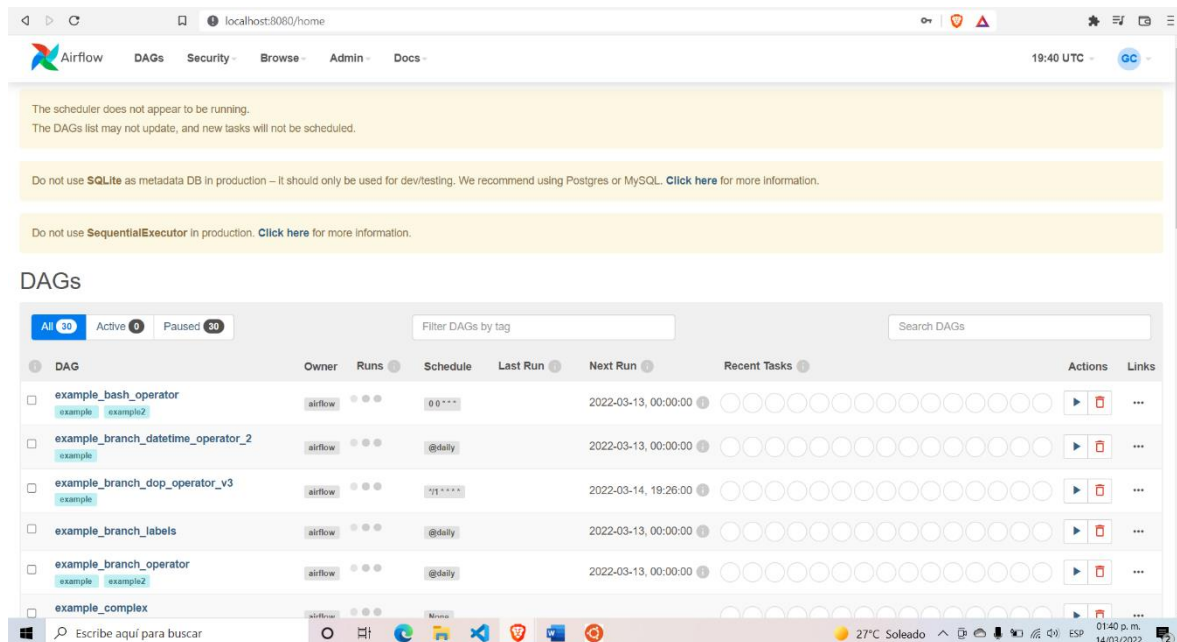
```
gael@LAPTOP-37AFC7C7:~$ airflow users create \
> --email oswaldogael10@gmail.com --firstname gael \
> --lastname cervantes --password 12345 \
> --role Admin --username gael65
[2022-03-14 13:38:40,580] {manager.py:779} WARNING - No user yet created, use flask fab command to do it.
[2022-03-14 13:38:40,839] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.
can_create Admin
[2022-03-14 13:38:42,068] {manager.py:214} INFO - Added user gael65
User "gael65" created with role "Admin"
gael@LAPTOP-37AFC7C7:~$
```

Ingresamos en la interfaz las credenciales de nuestro usuario creado:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.



Observamos que ya podemos acceder a la interfaz de Apache Airflow, pero tenemos una serie de errores; el más significativo hace referencia a que no ha sido iniciado el “scheduler” de Airflow:

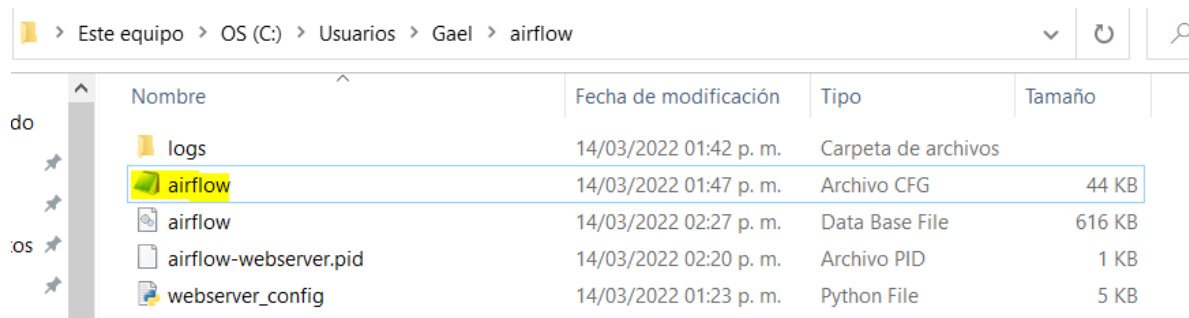


Para inicializar el scheduler es necesario abrir otra terminal de Ubuntu, para lo cual presionamos Shift + Clic izquierdo sobre el ícono de Ubuntu. Una vez abierta la nueva terminal ingresamos el comando “airflow scheduler”:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
gael@LAPTOP-37AFC7C7: ~  
gael@LAPTOP-37AFC7C7:~$ airflow scheduler  
  
[2022-03-14 13:42:35,423] {scheduler_job.py:619} INFO - Starting the scheduler  
[2022-03-14 13:42:35,423] {scheduler_job.py:624} INFO - Processing each file at most -1 times  
[2022-03-14 13:42:35 -0600] [5662] [INFO] Starting gunicorn 20.1.0  
[2022-03-14 13:42:35 -0600] [5662] [INFO] Listening at: http://0.0.0.0:8793 (5662)  
[2022-03-14 13:42:35 -0600] [5662] [INFO] Using worker: sync  
[2022-03-14 13:42:35,441] {manager.py:163} INFO - Launched DagFileProcessorManager with pid: 5663  
[2022-03-14 13:42:35,442] {scheduler_job.py:1137} INFO - Resetting orphaned tasks for active dag runs  
[2022-03-14 13:42:35,450] {settings.py:55} INFO - Configured default timezone Timezone('UTC')  
[2022-03-14 13:42:35,465] {manager.py:441} WARNING - Because we cannot use more than 1 thread (parsing_processes = 2) wh  
en using sqlite. So we set parallelism to 1.  
[2022-03-14 13:42:35 -0600] [5664] [INFO] Booting worker with pid: 5664  
[2022-03-14 13:42:35 -0600] [5666] [INFO] Booting worker with pid: 5666
```

Ahora es necesario realizar una modificación en el archivo de configuración de airflow, mismo que podemos encontrar en el directorio definido al inicio del proceso de instalación:



Este equipo > OS (C:) > Usuarios > Gael > airflow				
	Nombre	Fecha de modificación	Tipo	Tamaño
	logs	14/03/2022 01:42 p. m.	Carpeta de archivos	
	airflow	14/03/2022 01:47 p. m.	Archivo CFG	44 KB
	airflow	14/03/2022 02:27 p. m.	Data Base File	616 KB
	airflow-webserver.pid	14/03/2022 02:20 p. m.	Archivo PID	1 KB
	webserver_config	14/03/2022 01:23 p. m.	Python File	5 KB

Tenemos que ubicar el parámetro llamado “load_examples” que inicialmente está definido como “True”:

```
# Whether to load the DAG examples that ship with Airflow. It's good to  
# get started, but you probably want to set this to ``False`` in a production  
# environment  
load_examples = True
```

Redefinimos el parámetro con “False”:

```
# Whether to load the DAG examples that ship with Airflow. It's good to  
# get started, but you probably want to set this to ``False`` in a production  
# environment  
load_examples = False
```

Ejecutamos de nueva cuenta el comando de “airflow db init”:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
gael@LAPTOP-37AFC7C7:~$ airflow db init
DB: sqlite:///c/Users/Gael/airflow/airflow.db
[2022-03-14 13:47:58,995] {db.py:919} INFO - Creating tables
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
Initialization done
gael@LAPTOP-37AFC7C7:~$
```

Del mismo modo “airflow webserver”:

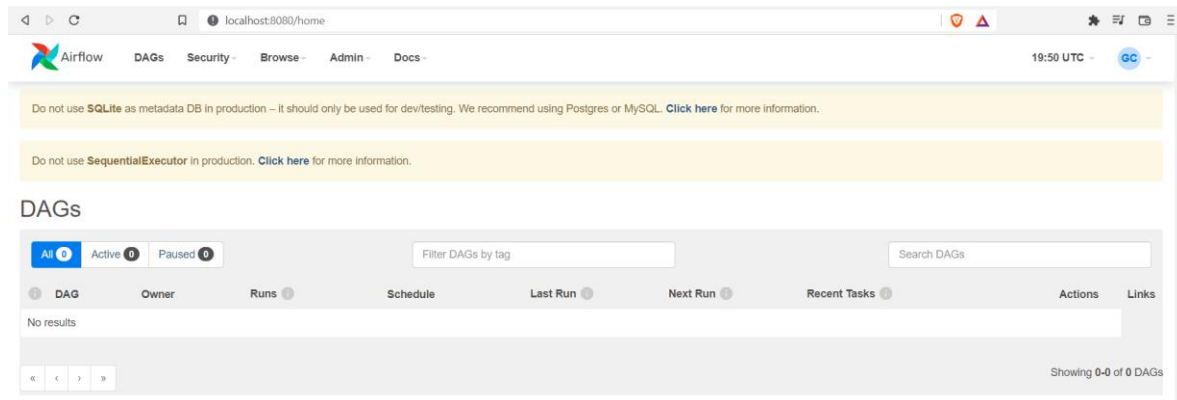
[illegible]

Finalmente “airflow scheduler”:

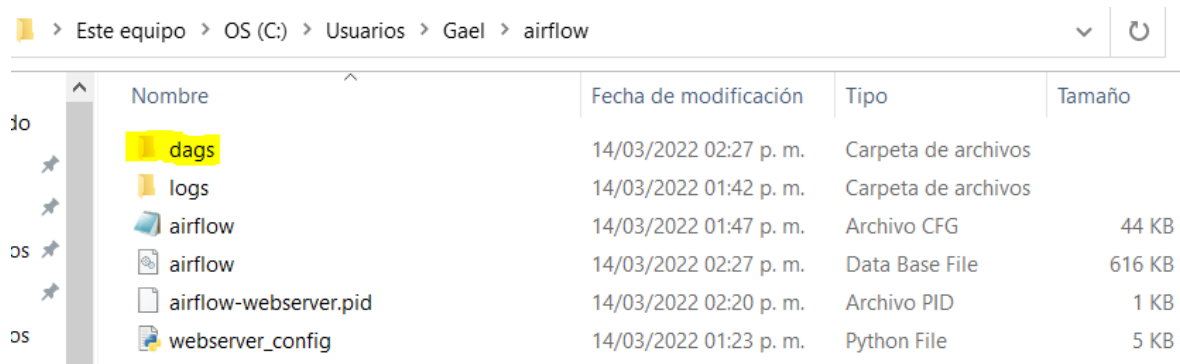
[illegible]

Regresamos a la interfaz que abrimos en nuestro navegador web y refrescamos:

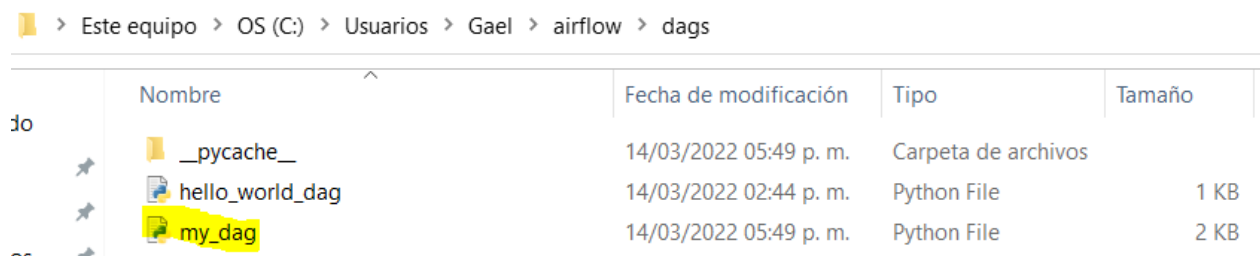
Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.



Para poder comenzar con la creación de DAGs tenemos que crear una carpeta llamada “dags” en el directorio “airflow” que creamos al inicio de este tutorial:

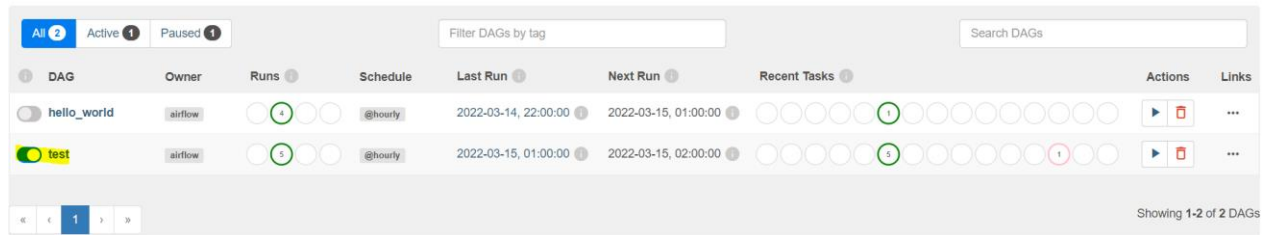


Dentro de esta carpeta vamos a crear nuestros DAGs. Previamente definí uno llamado “hello_world_dag” pero para este tutorial voy a crear uno nuevo llamado “my_dag”:



Refrescamos nuestra interfaz en el navegador y podemos observar que se muestran ambos DAGs.

DAGs



The screenshot shows the Apache Airflow web interface. At the top, there are tabs for 'All', 'Active', and 'Paused'. Below this is a search bar and a table of DAGs. The table has columns for DAG, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. Two DAGs are listed: 'hello_world' and 'test'. The 'test' DAG is highlighted in yellow. The 'Recent Tasks' column shows a sequence of task IDs (1, 2, 3, 4, 5) with green circles indicating successful completion.

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
hello_world	airflow	4	@hourly	2022-03-14, 22:00:00	2022-03-15, 01:00:00	1 2 3 4 5	[Play] [Stop] [Refresh]	...
test	airflow	5	@hourly	2022-03-15, 01:00:00	2022-03-15, 02:00:00	1 2 3 4 5	[Play] [Stop] [Refresh]	...

Podemos observar que se visualizan algunas cosas interesantes en la ventana principal de la interfaz como los DAGs que tenemos definidos y su estado, las veces que se ha ejecutado el DAG y su estado, fechas de última y siguiente ejecución, así como el estado actual de sus tareas.

Para el archivo de nuestro DAG el primer paso es importar algunos módulos u “operadores” desde airflow como por ejemplo DAG, PythonOperator (para poder ejecutar una función de Python desde Airflow), BranchPythonOperator (para poder tener un flujo de ramas en cuanto a ejecución de funciones de Python que a su vez retornan un valor de id de tarea o “task_id”), BashOperator (para poder ejecutar comandos desde Airflow) y otras librerías como “datetime” para poder definir algunos parámetros asociados con fechas de calendarización para la ejecución de nuestras tareas y “randint” que conforme se avance en la explicación del código se mencionará su utilidad.

```
from airflow import DAG
from airflow.operators.python import PythonOperator, BranchPythonOperator
from airflow.operators.bash import BashOperator

from datetime import datetime
from random import randint
```

El siguiente paso es definir el “context manager” por medio de “with DAG() as dag”. Dentro de los parámetros primero definimos el identificador del DAG, después la fecha en la que se comenzará a programar nuestra tarea (para esto fue que se importó el módulo “datetime”), después en intervalo de reprogramación de la tarea que en este caso fue definido para reprogramarse cada hora y finalmente definimos el parámetro “catchup”

como False, esto con el fin de que en Airflow solamente se visualice la información más reciente obtenida desde la última ejecución del DAG.

Dentro del cuerpo del “context manager”, en primer lugar, definimos 3 tareas llamadas “training_model_A/B/C” por medio de una especie de “foreach” que itera sobre la lista ['A', 'B', 'C'] para crear operadores de tipo PythonOperator que en este caso reciben como “task_id” el valor que devuelve el “foreach” en cada iteración y se asocia este operador con una función por medio de “python_callable=” seguido de “_training_model”, el cual es el nombre de la función asociada con el operador:

```
with DAG("test",
        start_date=datetime(2022, 3, 14),
        schedule_interval="@hourly",
        catchup=False) as dag:

    training_model_tasks = [
        PythonOperator(
            task_id=f"training_model_{model_id}",
            python_callable=_training_model,
            op_kwargs={
                "model": model_id
            }
        ) for model_id in ['A', 'B', 'C']
    ]
```

Ahora definimos la función que se asocia con estos 3 operadores que hemos definido en el paso anterior, la cual simplemente genera un número aleatorio entre 1 y 10:

```
def _training_model(model):
    return randint(1, 10)
```

El siguiente paso es definir otro operador, el cual en este caso se trata de un operador de tipo BranchPythonOperator para poder elegir la ejecución de una de dos opciones que

tendremos en el futuro. Por el momento definimos el id de esta tarea y la asociamos con la función “_choosing_best_model”:

```
choosing_best_model = BranchPythonOperator(  
    task_id="choosing_best_model",  
    python_callable=_choosing_best_model  
)
```

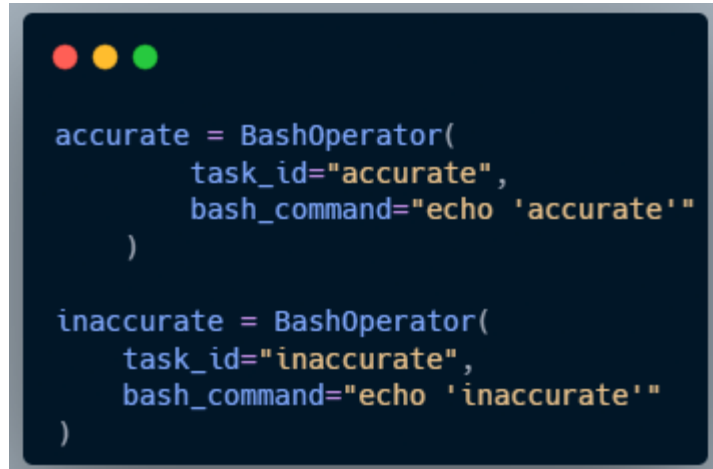
Para la función “_choosing_best_model”, necesitamos compartir información entre nuestras tareas puesto que requerimos de la mejor “efectividad” entre los diferentes operadores “training_model” definidos anteriormente para lo cual tenemos que hacer uso de XCOM (Cross Communication Message) para poder recuperar dicha información desde la base de datos de Airflow.

Primero definimos el parámetro “ti” (Task Instance) como argumento de la función, mismo que dentro de la función podemos acceder a su método “xcom_pull()” definiendo los id’s de las tareas que queremos recuperar desde la base de datos; misma información se almacena en una variable puesto que la información recuperada se retorna como una estructura de datos iterable.

Por medio de la función “max()” enviando como parámetro la estructura de datos que contiene la información de los operadores “training_model” para obtener la efectividad más grande y según la sentencia selectiva o “if” retornaremos un id de tarea:

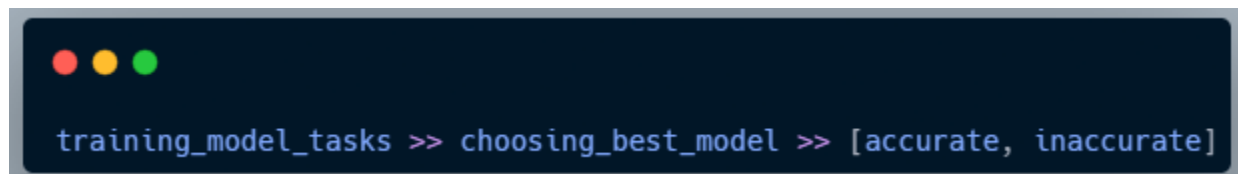
```
def _choosing_best_model(ti):  
    accuracies = ti.xcom_pull(task_ids=[  
        "training_model_A",  
        "training_model_B",  
        "training_model_C"  
    ])   
    if max(accuracies) > 8:  
        return "accurate"  
    return "inaccurate"
```

El siguiente paso es definir un último tipo de operador, en este caso son dos operadores de tipo BashOperator para ejecutar un comando según el id de tarea que haya sido proporcionado por la tarea anterior:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. It contains two lines of Python code defining BashOperator tasks.

```
accurate = BashOperator(  
    task_id="accurate",  
    bash_command="echo 'accurate'"  
)  
  
inaccurate = BashOperator(  
    task_id="inaccurate",  
    bash_command="echo 'inaccurate'"  
)
```

El último paso es definir la dependencia entre nuestras tareas. En primer lugar, tienen que generarse los operadores o “training_models” para tener información a evaluar en la función que se llama a continuación o en segunda instancia “choosing_best_model” la cual retornará un id de tarea que puede ser “accurate” o “inaccurate” y según sea este se ejecutará alguna de las dos funciones que llevan el mismo nombre para imprimir en la salida estándar un mensaje por medio de los comandos de bash:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. It shows a dependency chain between tasks.

```
training_model_tasks >> choosing_best_model >> [accurate, inaccurate]
```

Con lo anterior ya tenemos lista nuestra DAG y podemos utilizar algunas de las herramientas que nos ofrece Apache Airflow para poder visualizar información relevante con respecto a la ejecución de nuestras tareas.

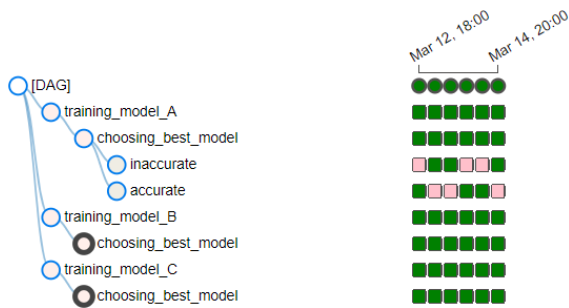
Lo primero que haremos es visualizar la información de secuencia de ejecución de nuestras tareas así como el estado en el que finalizaron las mismas en diferentes puntos en el tiempo, remarcados con fechas y colores para el estado de finalización:

DAG: test

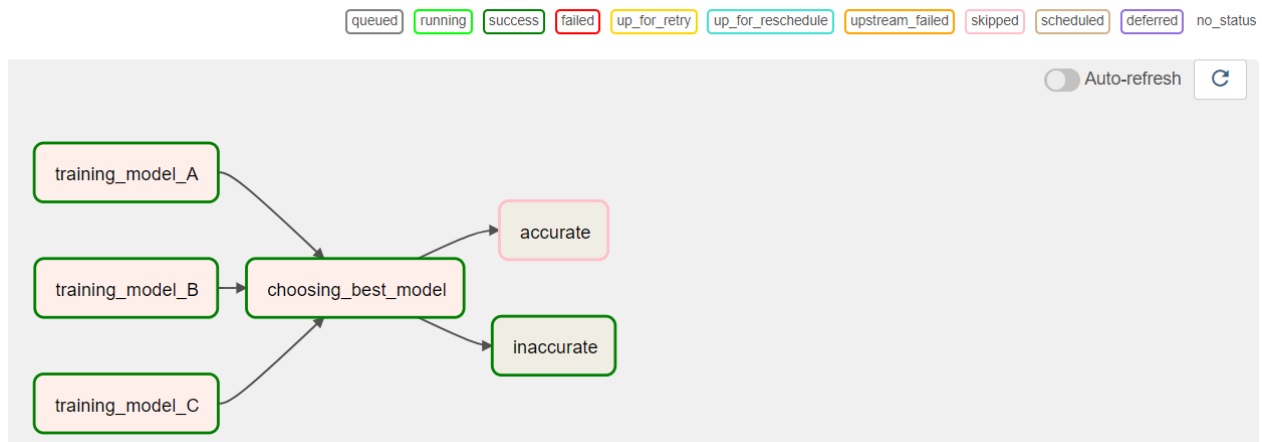
Tree Graph Calendar Task Duration Task Tries

2022-03-15T02:00:00Z Runs 25 Update

☐ BashOperator ☐ BranchPythonOperator ☐ PythonOperator



Si seleccionamos la opción de grafo podemos visualizar las dependencias de las tareas que definimos, así como el estado de finalización de las tareas. En este punto cabe señalar que tenemos 5 tareas que finalizaron con éxito y una que terminó en estado “skipped”: esto se debe a que definimos un operador de “Branch” o rama y eso hizo que en el flujo se haya tomado un camino (en esta ejecución fue “inaccurate”) y se omitió el otro operador:



Si hacemos clic sobre el último estado de finalización de nuestro grafo podemos tener acceso a más funciones como por ejemplo el “LOG”:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.









```
*** Reading local file: /c/Users/Gael/airflow/logs/test/inaccurate/2022-03-15T02:00:00+00:00/1.log
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: test.inaccurate scheduled__2022-03-15T02:00:00+00:00 [queued]>
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: test.inaccurate scheduled__2022-03-15T02:00:00+00:00 [queued]>
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1243} INFO -
-----
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1244} INFO - Starting attempt 1 of 1
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1245} INFO -
-----
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1264} INFO - Executing <Task(BashOperator): inaccurate> on 2022-03-15 02:00:00+00:00
[2022-03-14, 21:00:10 UTC] {standard_task_runner.py:52} INFO - Started process 7390 to run task
[2022-03-14, 21:00:10 UTC] {standard_task_runner.py:76} INFO - Running: ['airflow', 'tasks', 'run', 'test', 'inaccurate', 'scheduled__2022-03-15T02:00:00+00:00', '--job-id', '39', '--raw', '--subdir', 'DAGS',
[2022-03-14, 21:00:10 UTC] {standard_task_runner.py:77} INFO - Job 39: Subtask inaccurate
[2022-03-14, 21:00:10 UTC] {logging_mixin.py:109} INFO - Running <TaskInstance: test.inaccurate scheduled__2022-03-15T02:00:00+00:00 [running]> on host LAPTOP-37AFC7C7.localdomain
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1429} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=test
AIRFLOW_CTX_TASK_ID=inaccurate
AIRFLOW_CTX_EXECUTION_DATE=2022-03-15T02:00:00+00:00
AIRFLOW_CTX_DAG_RUN_ID=scheduled__2022-03-15T02:00:00+00:00
[2022-03-14, 21:00:10 UTC] {subprocess.py:62} INFO - Temp dir root location:
/tmp
[2022-03-14, 21:00:10 UTC] {subprocess.py:74} INFO - Running command: ['bash', '-c', "echo 'inaccurate'"]
[2022-03-14, 21:00:10 UTC] {subprocess.py:85} INFO - Output:
[2022-03-14, 21:00:10 UTC] {subprocess.py:89} INFO - inaccurate
[2022-03-14, 21:00:10 UTC] {subprocess.py:93} INFO - Command exited with return code 0
[2022-03-14, 21:00:10 UTC] {taskinstance.py:1272} INFO - Marking task as SUCCESS. dag_id=test, task_id=inaccurate, execution_date=20220315T020000, start_date=20220315T030010, end_date=20220315T030010
[2022-03-14, 21:00:10 UTC] {local_task_job.py:154} INFO - Task exited with return code 0
[2022-03-14, 21:00:10 UTC] {local_task_job.py:264} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

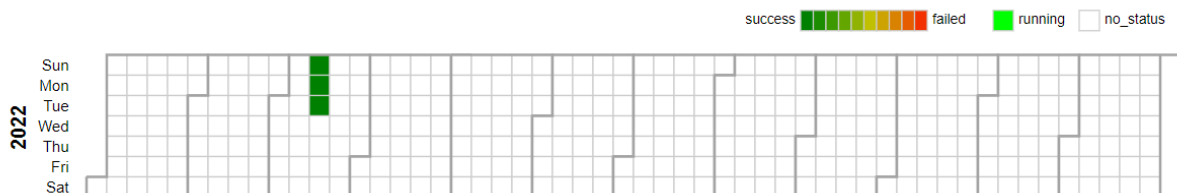
Podemos observar información específica como por ejemplo en la siguiente imagen, observamos que según el estado de finalización de la última tarea se muestra un mensaje por medio de un comando bash:

```
[2022-03-14, 21:00:10 UTC] {subprocess.py:74} INFO - Running command: ['bash', '-c', "echo 'inaccurate'"]
[2022-03-14, 21:00:10 UTC] {subprocess.py:85} INFO - Output:
[2022-03-14, 21:00:10 UTC] {subprocess.py:89} INFO - inaccurate
[2022-03-14, 21:00:10 UTC] {subprocess.py:93} INFO - Command exited with return code 0
```

También tenemos una opción para mostrar un calendario parecido al mostrado en la vista de árbol:

DAG: test

 Tree  Graph  Calendar  Task Duration  Task Tries  Landing Times  Gantt  Details  Code



Podemos visualizar la información que se compartió entre tareas por medio de XCOM:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Task Instance: inaccurate at 2022-03-15, 02:00:00

[Task Instance Details](#)

[< > Rendered Template](#)

[Log](#)

[XCom](#)

XCom

Key	Value
return_value	inaccurate

Incluso podemos visualizar el código completo en la interfaz:



The screenshot shows the Apache Airflow web interface with the DAG 'test' selected. The code is displayed in a light gray box with line numbers from 1 to 51. The code defines two functions, `_choosing_best_model` and `_training_model`, and sets up a DAG with three tasks: `training_model_tasks` (a list of `PythonOperator` tasks), `choosing_best_model` (a `BranchPythonOperator`), and `accurate` and `inaccurate` (both `BashOperator` tasks). The DAG is configured with a start date of 2022-03-14 and a schedule interval of '@hourly'.

```
1 from airflow import DAG
2 from airflow.operators.python import PythonOperator, BranchPythonOperator
3 from airflow.operators.bash import BashOperator
4
5 from datetime import datetime
6 from random import randint
7
8 def _choosing_best_model(ti):
9     accuracies = ti.xcom_pull(task_ids=[
10         "training_model_A",
11         "training_model_B",
12         "training_model_C"
13     ])
14     if max(accuracies) > 8:
15         return "accurate"
16     return "inaccurate"
17
18 def _training_model(model):
19     return randint(1, 10)
20
21 with DAG("test",
22     start_date=datetime(2022, 3, 14),
23     schedule_interval="@hourly",
24     catchup=False) as dag:
25
26     training_model_tasks = [
27         PythonOperator(
28             task_id=f"training_model_{model_id}",
29             python_callable=_training_model,
30             op_kwargs={
31                 "model": model_id
32             }
33         ) for model_id in ['A', 'B', 'C']
34     ]
35
36     choosing_best_model = BranchPythonOperator(
37         task_id="choosing_best_model",
38         python_callable=_choosing_best_model
39     )
40
41     accurate = BashOperator(
42         task_id="accurate",
43         bash_command="echo 'accurate'"
44     )
45
46     inaccurate = BashOperator(
47         task_id="inaccurate",
48         bash_command="echo 'inaccurate'"
49     )
50
51     training_model_tasks >> choosing_best_model >> [accurate, inaccurate]
```

Enlace al repositorio: <https://github.com/gaeltcervantes65/Airflow.git>

Conclusiones

Como conclusiones para la realización de esta actividad puedo mencionar que pude aprender mucho con todo el proceso que involucró trabajar un ejemplo en Apache Airflow. En primer lugar, con la instalación pude aprender algunas cosas básicas sobre Linux como uso de comandos y editores de archivos; aunque tuve muchas complicaciones en esta parte considero que las pude resolver a tiempo y con buenos resultados.

En cuanto a la codificación del ejemplo, en un principio considero que fue complicado entender un poco cómo es que funciona la ejecución de los scripts de Python puesto que en este caso no se utiliza la terminal sino que Airflow los ejecuta por medio de “operadores” tales como PythonOperator para ejecutar funciones.

Si tuviera que realizar una comparativa entre Airflow y Prefect considero que me quedo con Apache Airflow puesto que a pesar de que se me complicó mucho la instalación, la interfaz me dejó encantado puesto que es muy intuitiva además de que ofrece muchas funciones que facilitan las tareas para los desarrolladores (incluso puedes visualizar tu código fuente en la interfaz de Airflow) además de que pude comprender de mejor manera cómo funciona la dependencia entre tareas y la transferencia de información entre las mismas (cosa que con Prefect no pude entender del todo).

Fuentes de información

- datastacktv. (2020, 24 agosto). Install Apache Airflow on Windows using Windows Subsystem for Linux (WSL) [Video]. YouTube. <https://www.youtube.com/watch?v=M521KLHGaZc>
- Marc Lamberti. (2021, 4 marzo). *Airflow DAG: Coding your first DAG for Beginners* [Video]. YouTube. <https://www.youtube.com/watch?v=IH1-0hwFZRQ>
- I. (2022, 5 enero). *Hello World using Apache-Airflow - INSAID*. Medium. <https://insaid.medium.com/hello-world-using-apache-airflow-91859e3bbfd5>