Computación Tolerante a Fallas

Sección D06

Apache Airflow

Fernández Venegas David Guadalupe

Código: 216437352

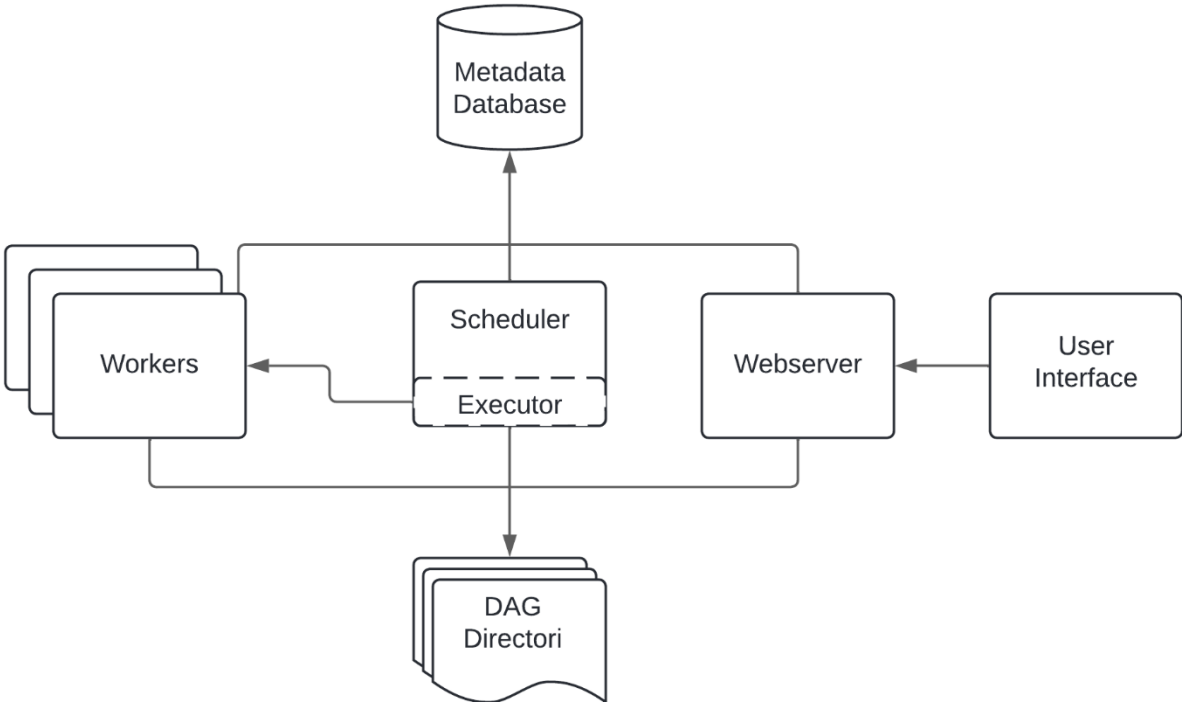
Profesor: López Franco Michel Emanuel

Introducción

Airflow es una plataforma la cual te permite crear programáticamente, agendar y monitorear flujos de trabajo. Airflow permite crear grafos dirigidos acíclicos (DAGs) en donde cada uno de los nodos del grafo representa una tarea. Además, airflow cuenta con un ejecutor el cual ejecuta las tareas para mantener las dependencias entre ellas, permitiendo agendar scripts con funcionalidades extra.

Esta herramienta es parte aplicación y parte biblioteca. Es aplicación ya que esta tiene un componente servidor que está permanentemente activo y se encarga de agendar y ejecutar las tareas, además de tener un componente de aplicación web que permite monitorear el estado de nuestras tareas. Pero también es biblioteca ya que se utiliza dentro de Python para trabajar por medio de las clases y elementos que esta incluye.

Arquitectura de airflow:



- En la zona central se encuentra un scheduler el cual se encargar de agendar y llamar al ejecutor.
- El ejecutor envía las tareas a los trabajadores para que estos las ejecuten.
- En la parte inferior se tiene un directorio en donde se guardan los archivos de configuración de los DAGs.
- En la parte superior se encuentra una base de datos en donde se almacena el estado de los trabajos que se han ejecutado.
- Del lado derecho se tiene un servidor webcon una interfaz que permite a los usuarios consultar el estado de los trabajos de forma amigable y remota.

Instalación

Para instalar apache airflow simplemente usamos el siguiente comando en la terminal:

```
$ pip install apache-airflow
```

Una vez instalado, se berá inicializar la base de datos:

```
$ airflow db init
```

El siguiente paso es crear un usuario:

```
$ Airflow users create \
      --username admin \
      --firstname David \
      --lastname Fernandez \
      --role Admin \
      --password ******** \
--email >   --email davidf@gmail.com
```

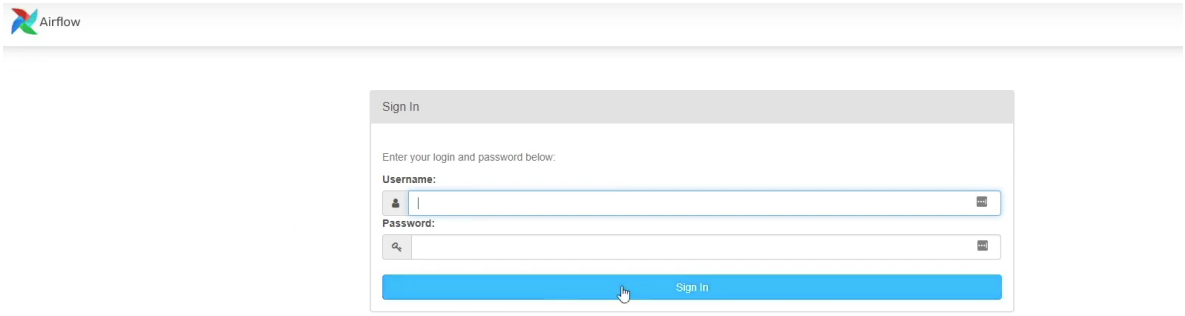
Se inicializar el servidor web

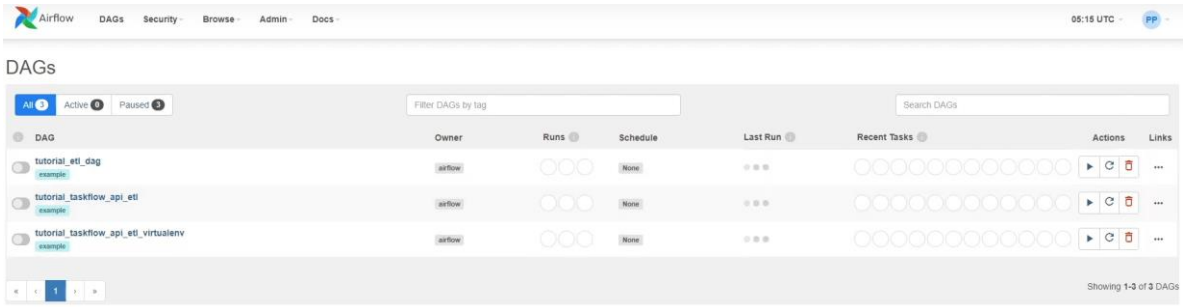
```
$ airflow webserver --port 8080
```

Y por ultimo inicializamos el scheduler

```
$ airflow scheduler
```

Ahora en un navegador podemos entrar a localhost:8080 para entrar a la interfaz de airflow en donde debemos ingresar nuestro usuario y contraseña





Programa

- **from datetime import timedelta**: Importa la clase **timedelta** de la librería **datetime**. se usa para representar una duración de tiempo.
- **from airflow import DAG**: Importa la clase **DAG** de Airflow, que representa un flujo de trabajo.
- **from airflow.operators.python import PythonOperator**: Importa el operador **PythonOperator**, que permite ejecutar funciones de Python como tareas en Airflow.
- **from airflow.utils.dates import days_ago**: Importa **days_ago**, una función de Airflow para obtener la fecha actual menos un número específico de días.
- **import logging**: Importa el módulo **logging** de Python para registrar mensajes durante la ejecución.

```
from datetime import timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
```

Define un diccionario que contiene los argumentos predeterminados para las tareas dentro del DAG. Esto incluye el propietario del DAG, la configuración de notificaciones por correo electrónico y la configuración de reintento en caso de fallos.

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

Se definen tres funciones (scrape, process, save), que representan las operaciones que se realizarán como tareas en el flujo de trabajo.

```
def scrape():
    logging.info("performing scraping")

def process():
    logging.info("performing processing")

def save():
    logging.info("performing saving")
```

- Se define un nuevo objeto **DAG** llamado **'first'**, con varios parámetros:
    - Los argumentos predeterminados definidos anteriormente.
    - Descripción del DAG.
    - Frecuencia con la que se ejecutará el DAG, en este caso, cada día.
    - Fecha de inicio del DAG, dos días antes de la fecha actual.
    - Etiquetas asociadas al DAG.
- Se definen tres tareas (**scrape_task**, **process_task**, **save_task**) utilizando **PythonOperator**, cada una ejecutando las funciones **scrape**, **process** y **save** respectivamente.
- Se define el flujo de dependencias entre las tareas utilizando el operador **>>**.

```
with DAG(
    'first',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(2),
    tags=['example']
)as dag:
    scrape_task = PythonOperator(task_id="scrape", python_callable=scrape)
    process_task = PythonOperator(task_id="process",
python_callable=process)
    save_task = PythonOperator(task_id="save", python_callable=save)

    scrape_task >> process_task >> save_task
```

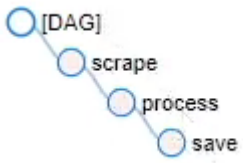
Resultados

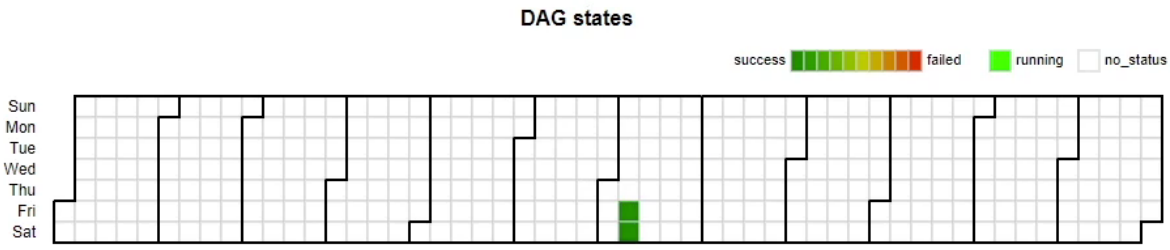
Utilizando la interfaz gráfica proporcionada por Prefect, podemos ver los resultados de nuetros flujos.

Al actualizar nuestra página de airflow, ahora podremos ver nuetro DAG:

Podemos quitar la pausa de nuestro DAG para observar los resultados.

Ahora podemos analisar nuestro DAG de formas distintas, incluyendo la vista de arbol, vista de gráfico o la vista de calendario:





Conclusión

Airflow es una herramienta poderosa para el manejo de flujos de trabajo, especialmente útil en entornos donde se necesitan ejecutar tareas programadas de manera eficiente y confiable. La principal utilidad de Airflow radica en su capacidad para definir, programar y monitorear flujos de trabajo complejos mediante el uso de DAGs, que permiten representar visualmente las dependencias entre tareas y la lógica de ejecución.

El ejemplo elaborado se demuestra cómo utilizar Airflow para definir un DAG simple que consta de tres tareas (**scrape**, **process**, **save**). Este DAG está configurado para ejecutarse diariamente a partir de dos días antes de la fecha actual.

El uso de esta herramienta nos da una gran cantidad de posibilidades con las cuales poder elaborar un programa robusto y confiable, como por ejemplo:

- **Automatización de Procesos**: permite automatizar procesos complejos mediante la definición de flujos de trabajo estructurados.
- **Gestión de Dependencias**: es posible gestionar las dependencias entre tareas de forma intuitiva, garantizando que las tareas se ejecuten en el orden correcto.
- **Monitoreo y Registro**: proporciona un panel de control para monitorear el estado y el rendimiento de los flujos de trabajo, además de facilitar el registro de eventos y errores durante la ejecución.
- **Flexibilidad y Escalabilidad**: es altamente configurable y escalable, lo que lo hace adecuado para una amplia variedad de casos de uso, desde tareas simples hasta flujos de trabajo complejos distribuidos en varios nodos.