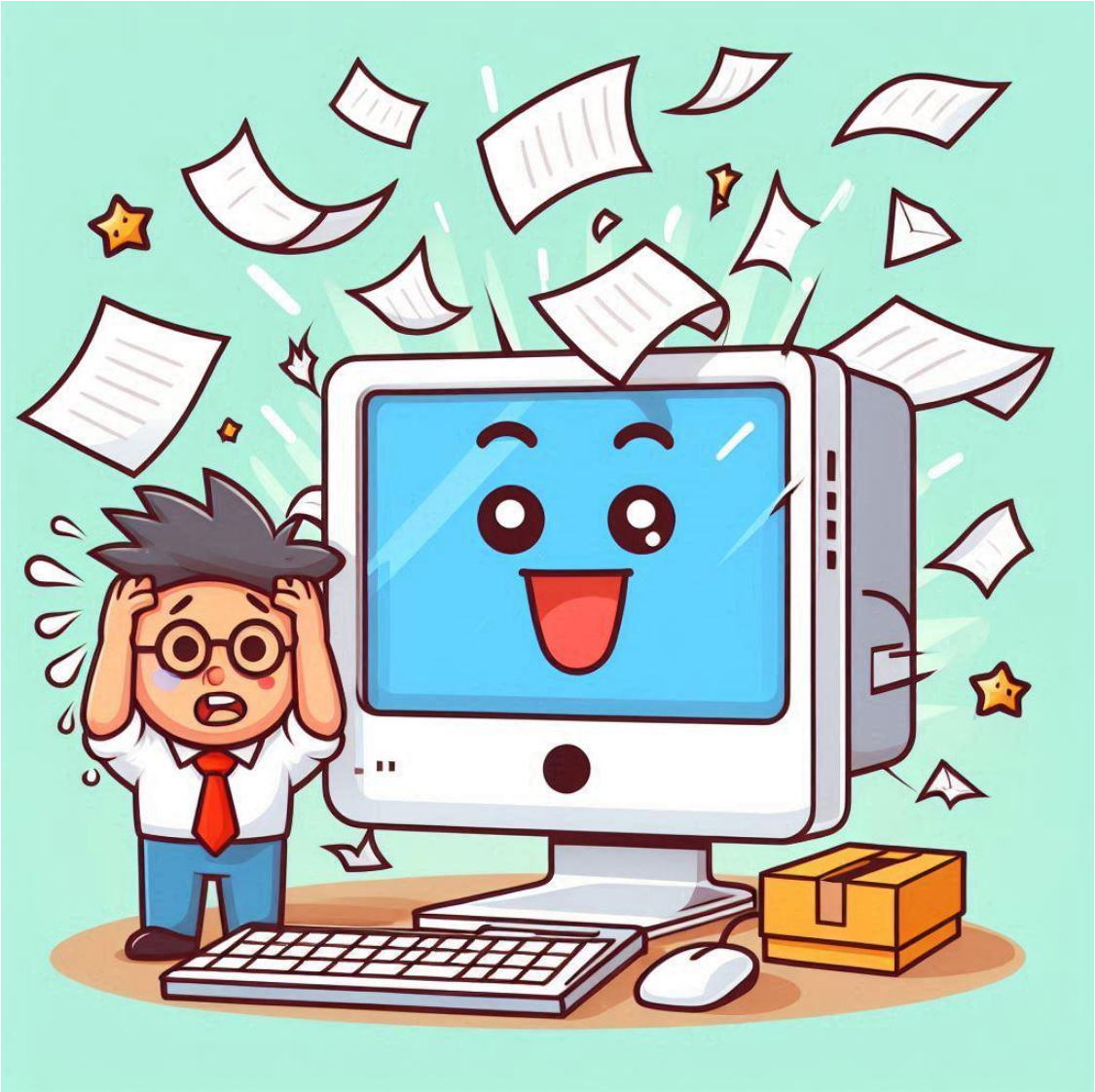
Airflow is a platform to programmatically author, schedule and monitor workflows.



- **Nombre:** Jesús Eduardo Jiménez Covarrubias
- **Materia:** Computación Tolerante a Fallas
- **Profesor:** Michel Emanuel López Franco
- **NRC:** 179961
- **Sección:** D06
- **Ciclo:** 2025-A

Instrucciones:

¿Qué es Apache Airflow?

Use Airflow to author workflows as Directed Acyclic Graphs (DAGs) of tasks. The Airflow scheduler executes your tasks on an array of workers while following the specified dependencies. Rich command line utilities make performing complex surgeries on DAGs a snap. The rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed.

https://airflow.apache.org/docs/apache-airflow/stable/python-api-ref.html

https://www.youtube.com/watch?v=ewK4KszmeTI

https://aprenderbigdata.com/apache-airflow

Desarrollo:

Código (Python):

```
from datetime import datetime
import subprocess
import time

class Task:
    def __init__(self, task_id):
        self.task_id = task_id
        self.next_tasks = []

    def set_next(self, *tasks):
        self.next_tasks.extend(tasks)
        return tasks[-1]

    def execute(self):
        print(f"Ejecutando tarea: {self.task_id}")


class BashTask(Task):
    def __init__(self, task_id, bash_command):
        super().__init__(task_id)
```

```
        self.bash_command = bash_command

    def execute(self):
        super().execute()
        subprocess.run(self.bash_command, shell=True)


class Workflow:
    def __init__(self, name, schedule="@daily"):
        self.name = name
        self.schedule = schedule
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)
        return task

    def run(self):
        print(f"Iniciando workflow: {self.name}")
        for task in self.tasks:
            task.execute()
            time.sleep(1)  # Pequeña pausa entre tareas
        print(f"Workflow {self.name} completado")


# Crear el workflow
workflow = Workflow('mi_primer_workflow')

# Definir las tareas
inicio = workflow.add_task(Task('inicio'))
procesamiento =
workflow.add_task(BashTask('procesamiento_datos', 'echo
"Procesando datos!"'))
notificacion =
workflow.add_task(BashTask('envio_notificacion', 'echo
"Notificación enviada!"'))
fin = workflow.add_task(Task('fin'))

# Definir el flujo de trabajo
inicio.set_next(procesamiento)
procesamiento.set_next(notificacion)
```

```
notificacion.set_next(fin)

# Ejecutar el workflow
if __name__ == "__main__":
    workflow.run()

    # Tarea paralela alternativa
    # inicio >> [procesamiento, notificacion] >> fin

# Documentación adicional
"""
Este DAG demuestra:
1. Creación de tareas simples con DummyOperator
2. Tareas que ejecutan comandos Bash con BashOperator
3. Definición de dependencias entre tareas
4. Programación diaria

Recursos útiles:
- Documentación oficial: https://airflow.apache.org
- Tutorial en video:
https://www.youtube.com/watch?v=ewK4KszmeTI
- Guía en español: https://aprenderbigdata.com/apache-airflow
"""
```

Funcionamiento del Código:

1. Estructura de Clases

class Task:

  def __init__(self, task_id):

    self.task_id = task_id

    self.next_tasks = []

- Esta es la clase base `Task` que representa una tarea básica

- Cada tarea tiene un `task_id` (identificador) y una lista de (tareas siguientes) `next_tasks`

- Es como un nodo en una cadena de tareas

class BashTask(Task):

  def __init__(self, task_id, bash_command):

super().__init__(task_id)

self.bash_command = bash_command

- es una clase especializada que hereda de `Task` `BashTask`

- Permite ejecutar comandos de bash/terminal

- Añade la capacidad de ejecutar comandos del sistema

class Workflow:

def __init__(self, name, schedule="@daily"):

self.name = name

self.schedule = schedule

self.tasks = []

- La clase `Workflow` es el contenedor principal

- Gestiona toda la secuencia de tareas

- Mantiene un registro de todas las tareas y su orden de ejecución

2. Flujo de Ejecución

1. Creación del Workflow:**

workflow = Workflow('mi_primer_workflow')

- Se crea una instancia de `Workflow` con un nombre específico

1. Definición de Tareas:**

inicio = workflow.add_task(Task('inicio'))

procesamiento = workflow.add_task(BashTask('procesamiento_datos', 'echo "Procesando datos!"'))

notificacion = workflow.add_task(BashTask('envio_notificacion', 'echo "Notificación enviada!"'))

fin = workflow.add_task(Task('fin'))

- Se crean cuatro tareas:

- : Una tarea simple de inicio `inicio`

- : Una tarea bash que muestra "Procesando datos!" `procesamiento`

- : Una tarea bash que muestra "Notificación enviada!" `notificacion`

- `fin`: Una tarea simple de finalización

1. **Definición del Flujo:**

inicio.set_next(procesamiento)

procesamiento.set_next(notificacion)

notificacion.set_next(fin)

- Establece el orden de ejecución: inicio → procesamiento → notificación → fin

- Es como crear una cadena donde cada tarea sabe cuál es la siguiente

1. **Ejecución del Workflow:**

if __name__ == "__main__":

workflow.run()

- Cuando se ejecuta el script, el método `run()` del workflow:

1. Muestra "Iniciando workflow: mi_primer_workflow"
2. Ejecuta cada tarea en orden
3. Espera 1 segundo entre tareas (time.sleep(1))
4. Muestra "Workflow mi_primer_workflow completado"

3. Resultado de Ejecución

Cuando ejecutas el código, verás esta secuencia:

Iniciando workflow: mi_primer_workflow

Ejecutando tarea: inicio

Ejecutando tarea: procesamiento_datos

Procesando datos!

Ejecutando tarea: envio_notificacion

Notificación enviada!

Ejecutando tarea: fin

Workflow mi_primer_workflow completado

4. Ventajas del Diseño

1. **Modularidad**: Cada tarea es independiente y puede modificarse sin afectar a las otras

2. **Extensibilidad**: Puedes crear nuevos tipos de tareas heredando de la clase `Task`

3. **Flexibilidad**: Puedes cambiar fácilmente el orden de las tareas

4. **Claridad**: El flujo de trabajo es fácil de entender y seguir

5. Uso Práctico

Este código es útil para:

- Automatizar secuencias de tareas

- Crear flujos de trabajo simples

- Ejecutar comandos del sistema en un orden específico

Pantallazos del Código:

Ejecutar en terminal:

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

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

(.venv) PS D:\Septimo Semeste\COMPUTACION TOLERANTE A FALLAS\Ejercicios\Ejercicio 10> python Airflow.py
Iniciando workflow: mi_primer_workflow
Ejecutando tarea: inicio
Ejecutando tarea: procesamiento_datos
"Procesando datos!"
Ejecutando tarea: envio_notificacion
"Notificación enviada!"
Ejecutando tarea: fin
Workflow mi_primer_workflow completado
(.venv) PS D:\Septimo Semeste\COMPUTACION TOLERANTE A FALLAS\Ejercicios\Ejercicio 10>
```

Conclusión:

El código presentado representa una implementación simplificada pero efectiva de un sistema de flujo de trabajo (workflow), que ofrece varias ventajas y aprendizajes importantes:

1. Aspectos Positivos

- **Simplicidad**: El código elimina la necesidad de dependencias externas como Apache Airflow, haciéndolo más accesible y fácil de ejecutar.

- **Funcionalidad**: A pesar de su simplicidad, mantiene las funcionalidades básicas esenciales:

  - Ejecución secuencial de tareas

  - Manejo de comandos bash

  - Organización clara del flujo de trabajo

2. Aplicaciones Prácticas

- Ideal para:

  - Proyectos pequeños y medianos

  - Automatización de tareas locales

  - Prototipado rápido de flujos de trabajo

  - Aprendizaje de conceptos de programación orientada a objetos

3. Limitaciones

- No incluye características avanzadas como:

  - Programación distribuida

  - Interface web

  - Manejo de errores avanzado

  - Paralelismo complejo

4. Valor Educativo

El código sirve como excelente ejemplo para entender:

- Principios de programación orientada a objetos

- Patrones de diseño básicos

- Gestión de flujos de trabajo

- Integración con comandos del sistema

5. Mejoras Potenciales

Para futuros desarrollos se podría:

- Añadir manejo de errores más robusto

- Implementar paralelismo

- Agregar logging más detallado

- Incluir opciones de persistencia

En resumen, este código representa una solución práctica y educativa para la gestión de flujos de trabajo simples, proporcionando un balance entre funcionalidad y simplicidad, ideal para aprendizaje y proyectos de pequeña a mediana escala.