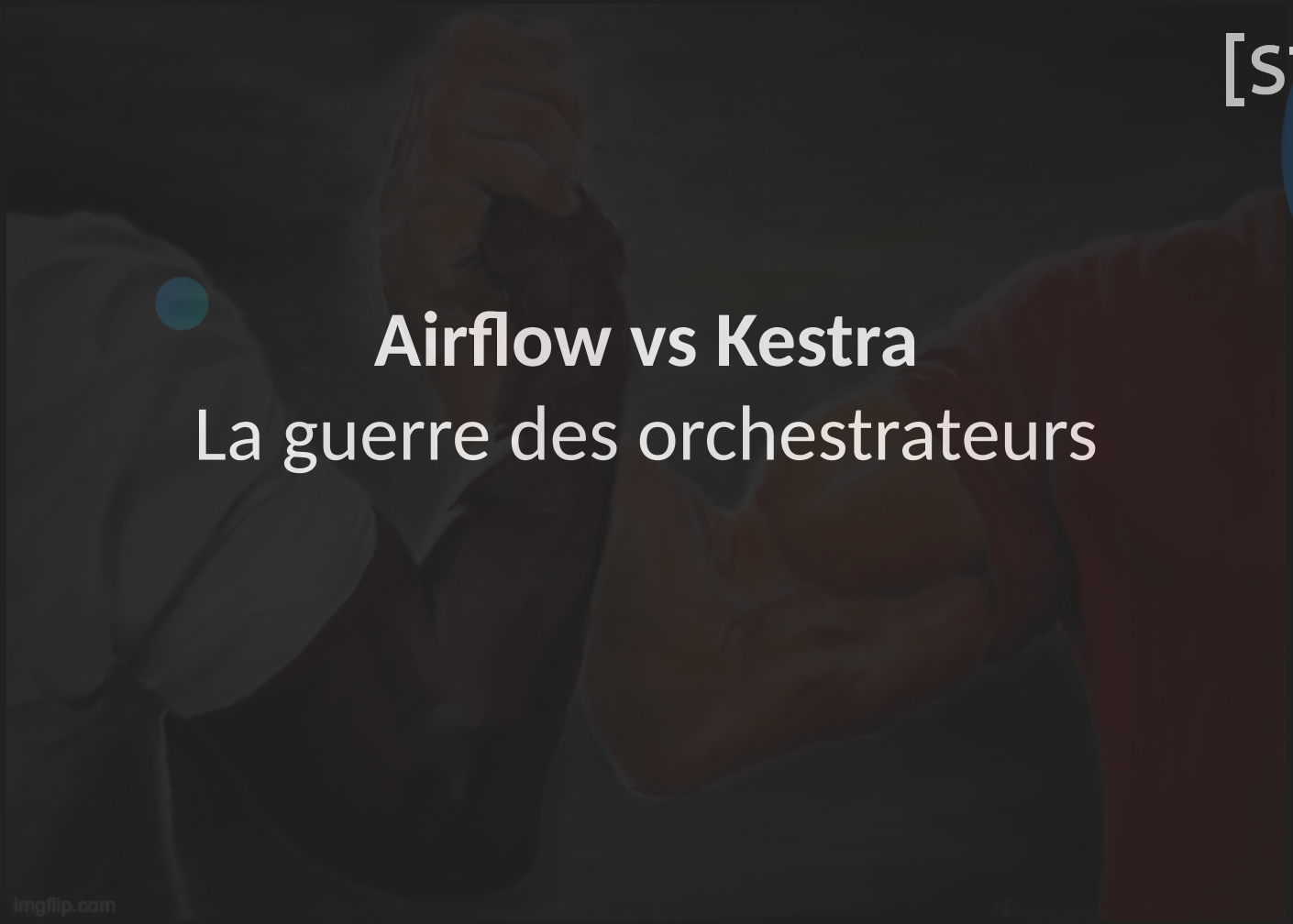
[sfeir]



Airflow vs Kestra

La guerre des orchestrateurs

[sfeir]

Florian
DEZE

Data / ML Engineer

deze.f@sfeir.com

Guillaume
FAUVERGUE

Data Engineer

fauvergue.g@sfeir.com

Un peu de vécu

[sfeir]

La page prend du temps à refresh

Un flux à fait planté les autres

Pourquoi ma nouvelle version ne se charge pas ?

Pourquoi mon Dag ne se lance plus ?

Il est pas encore apparu ? ça fait 5 minutes

[sfeir]

Un Orchestrateur c'est quoi ?

- Bien plus qu'un cron
- Bien plus qu'une simple chaîne linéaire de tâches

Apache Airflow

Créé par Airbnb
Release open source en 2016
Dernière version: 3.1

kestra.

Créé par Ludovic Dehon à Leroy Merlin
Release open source 2022
Dernière version: 1.3

[sf≡ir]

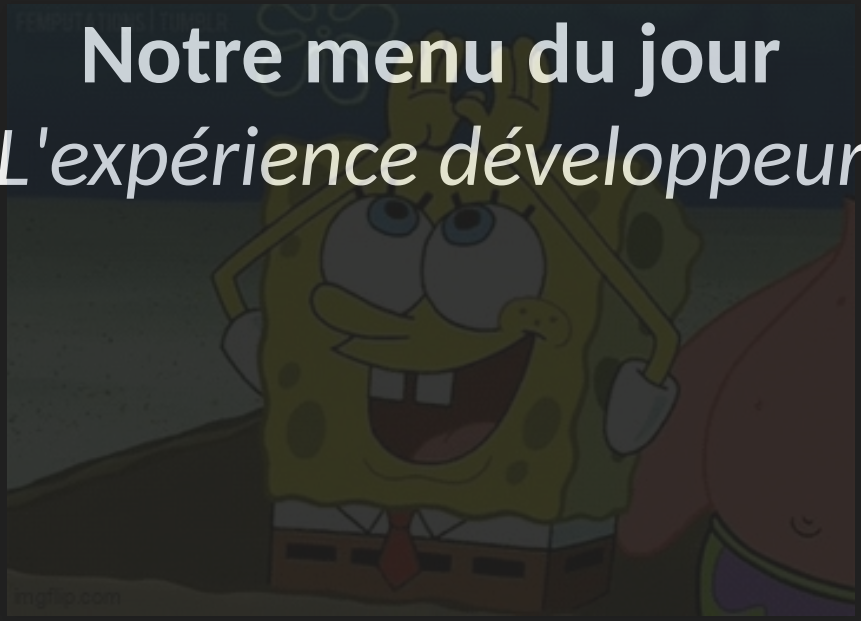
DISCLAIMER

La performance n’est pas le seul critère

[sf≡ir]

Notre menu du jour

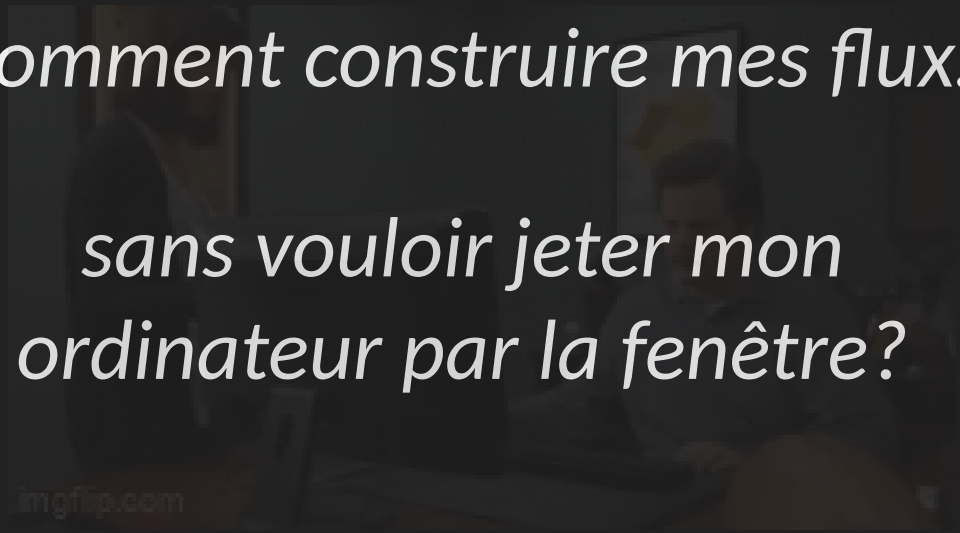
L'expérience développeur



BUILD

Comment construire mes flux…

sans vouloir jeter mon ordinateur par la fenêtre?



DEMO : *nous allons présenter une flux basique*

- **Détection d’un nouveau fichier dans un stockage BLOB**
- **Ingestion du fichier dans un Datawarehouse**

Un flux sur Airflow

```python
from airflow import DAG
from airflow.providers.google.cloud.sensors.gcs import GCSObjectExistenceSensor
from airflow.providers.google.cloud.transfers.gcs_to_bigquery import
GCSToBigQueryOperator
from datetime import datetime

with DAG(
    dag_id="gcs_to_bigquery_flow",
    start_date=datetime(2026, 1, 1),
    schedule=None, # Le DAG tourne en continu ou est déclenché
    catchup=False,
) as dag:

    # 1. Le "Sensor" asynchrone qui attend le fichier
    wait_for_file = GCSObjectExistenceSensor(
        task_id="wait_for_file",
        bucket="mon-bucket-inbound",
        object="data_inbound.csv",
        deferrable=True, # CRUCIAL : Libère le worker Airflow pendant l'attente !
    )

    # 2. Le chargement vers BigQuery
    load_to_bq = GCSToBigQueryOperator(
        task_id="load_csv_to_bq",
        bucket="mon-bucket-inbound",
        source_objects=["data_inbound.csv"],
        destination_project_dataset_table="mon-projet-gcp.ma_dataset.ma_table",
        source_format="CSV",
        skip_leading_rows=1,
    )

    # 3. La définition explicite de la dépendance
    wait_for_file >> load_to_bq
```

- **Flexibilité du code**
- **L'importance de deferrable=True**
- **Dépendance explicite**

Un flux sur Kestra

```yaml
id: gcs_to_bigquery_flow
namespace: dev.data

# 1. Le déclencheur (Trigger)
triggers:
  - id: watch_gcs_bucket
    type: io.kestra.plugin.gcp.gcs.Trigger
    projectId: mon-projet-gcp
    bucket: mon-bucket-inbound
    action: MOVE # Déplace le fichier après traitement pour éviter les doublons
    moveTo:
      bucket: mon-bucket-archive

# 2. L'exécution (Tasks)
tasks:
  - id: load_csv_to_bq
    type: io.kestra.plugin.gcp.bigquery.Load
    # Kestra récupère l'URI du fichier qui a déclenché le flow !
    from: "{{ trigger.uri }}"
    destinationTable: mon-projet-gcp.ma_dataset.ma_table
    format: CSV
    csvOptions:
      fieldDelimiter: ","
      headerRows: 1
```

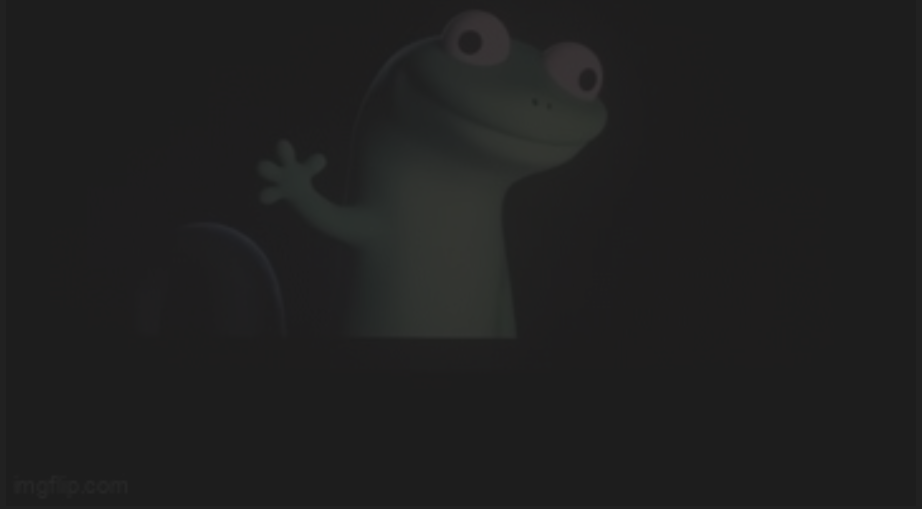
- Evenementiel natif
- Passage de contexte automatique
- Nettoyage intégré

Réponse finale au débat syntaxique

```
load_csv_to_bq = io.kestra.plugin.gcp.bigquery.Load(
    from = "{{ trigger.uri }}"
    destinationTable = "mon-projet-gcp.ma_dataset.ma_table"
    format = "CSV"
    csvOptions= {
        "fieldDelimiter": ",",
        "headerRows": 1
    }
)
```

TRIGGER & DÉPENDANCE

Comment déclencher mes flux ?



Airflow

On peut déclencher un DAG de plusieurs manières :

- Appel API
- Cron
- Assets
- Mixe Cron et Assets
- MessageQueueTrigger

Kestra

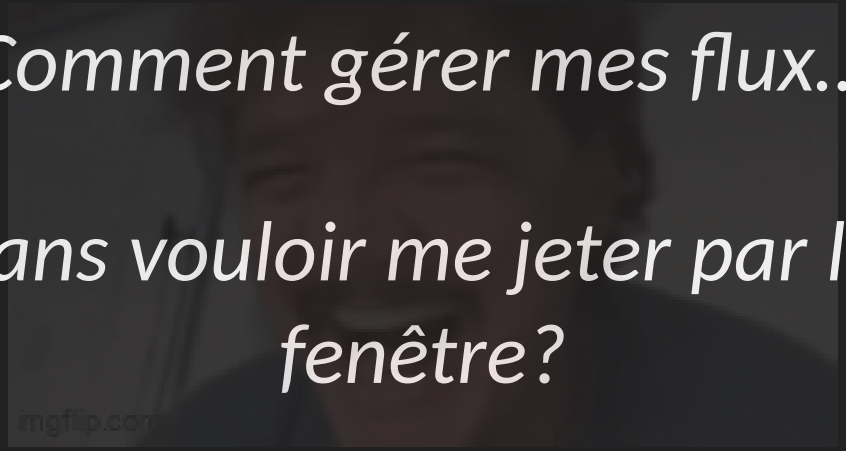
On peut déclencher un DAG de plusieurs manières :

- Appel API
- Cron
- Abonnement à des flux
- Trigger de sous-flux

RUN

Comment gérer mes flux…

sans vouloir me jeter par la fenêtre?



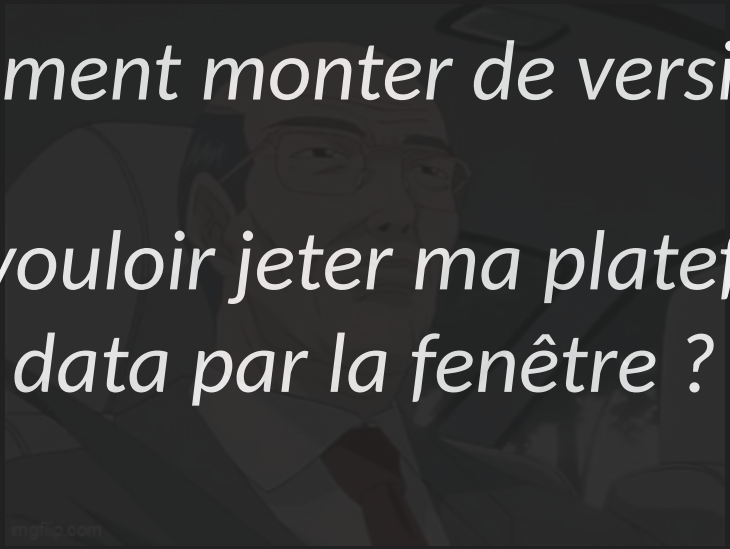
DEMO : *comment trouver les anomalies dans Airflow et Kestra*

DEMO : *comment relancer les flux dans Airflow et Kestra ?*

MAINTENABILITÉ

Comment monter de version…

sans vouloir jeter ma plateforme data par la fenêtre ?



Contexte

“ La version d’un orchestrateur n’est pas la version d’un opérateur ”

- 500 Dags utilisant l’opérateur X
- Migration de mon orchestrateur sur une version supérieur

=> L’opérateur X a un Breaking Change, je dois modifier mes 500 dags !

Sur Airflow

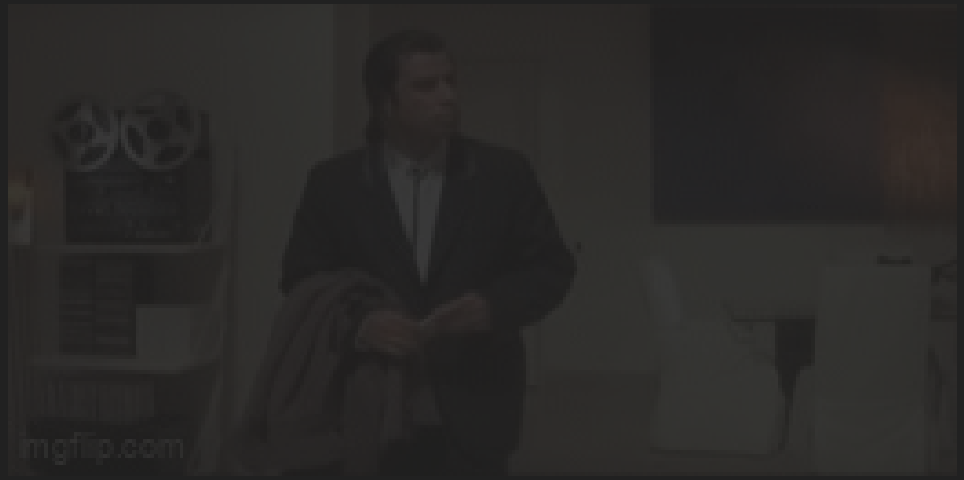
- Ne jamais utiliser les opérateurs natifs directement
  - Créer une surcouche
  - Possibilité de forcer des valeurs sur certains arguments
- Privilégier les KubernetesPodOperator et les images docker aux bashOperator et pythonOperator
- Appliquer des Policies

Sur Kestra

- Utiliser des subflows pour les tâches redondantes
- Mettre des valeurs par défauts sur des argument
- Créer son propre opérateur en JAVA via micronaute

CONCLUSION

C’est qui qui faut choisir ?



Apache Airflow

kestra.

- Personnalisation simple des opérateurs
- Définition normé des DAGs via les policies
- Versionning par run
- Montée de version plus simple (si bonnes pratiques appliquées)

- Type de Trigger événementiel plus complets
- CI/CD: Terraform et update des flux avec faible latence
- Syntaxe plus lisible sur des flux simples

Merci de votre attention

Des questions ?