INFO5 VTS
Apache Airflow

29 Novembre 2021

CAMBUS Quentin
JULIENNE Malone

Vue d'ensemble

1. Kezako ?
2. Fonctionnement
3. Avantages & Inconvénients
4. Démonstration
5. Cas d’utilisation
6. Concurrence

Kezako ?

Kezako ?

Qui ?	• Apache
Quand ?	• 2014
Quoi ?	• Gestion de flux de travail open-source • 100% Python

Flux de travail

Un workflow, flux de travaux ou encore flux opérationnel, est la représentation d'une suite de tâches ou d'opérations effectuées par une personne, un groupe de personnes, un organisme, etc. Le terme flow (« flux ») renvoie au passage du produit, du document, de l'information, etc., d'une étape à l'autre.

Wikipedia

Kezako ?

Qui ?	• Apache
Quand ?	• 2014
Quoi ?	• Gestion de flux de travail open-source • 100% Python
Pourquoi ?	• Faciliter et automatiser certaines tâches • Gestion et surveillance des flux

Fonctionnement

Fonctionnement - Installation

Localement

Docker

Fonctionnement - Dossiers & Fichiers

Dossiers

- dags
  - L'ensemble des workflow
- plugins
  - Plugin personnalisé
    - Ajout de fonctionnalités
    - Ajout de page au web serveur
- logs
  - Ensemble des logs des tâches exécutées

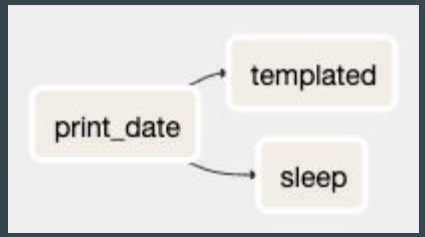
Fichiers

- P
- Y
- T
- H
- O
- N

Fonctionnement - Dags

- (Directed Acyclic Graph) est un pipeline de données défini en code Python.

- Une suite de tâches à exécuter et organisées



Fonctionnement - Et niveau code alors ?

Les imports

```
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG

# Operators; we need this to operate!
from airflow.operators.bash import BashOperator
```

Fonctionnement - Et niveau code alors ? (suite)

Les arguments d'un DAG

```
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
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),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}
```

Fonctionnement - Et niveau code alors ? (suite)

Définition d'un DAG

```
with DAG(
    'tutorial',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval=timedelta(days=1),
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=['example'],
) as dag:
```

Fonctionnement - Et niveau code alors ? (suite)

Définition de tâches

```
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    retries=3,
)
```

Fonctionnement - Et niveau code alors ? (suite)

Orchestration des tâches

```
t1.set_downstream(t2)

# This means that t2 will depend on t1
# running successfully to run.
# It is equivalent to:
t2.set_upstream(t1)

# The bit shift operator can also be
# used to chain operations:
t1 >> t2

# And the upstream dependency with the
# bit shift operator:
t2 << t1

# Chaining multiple dependencies becomes
# concise with the bit shift operator:
t1 >> t2 >> t3

# A list of tasks can also be set as
# dependencies. These operations
# all have the same effect:
t1.set_downstream([t2, t3])
t1 >> [t2, t3]
[t2, t3] << t1
```

Fonctionnement - Les plus

- Gestion des clés

- Providers
  - Google
  - Postgres

- Sécurité
  - Ajout de rôles
  - Gestion des permissions

Avantages & Inconvénients

Avantages

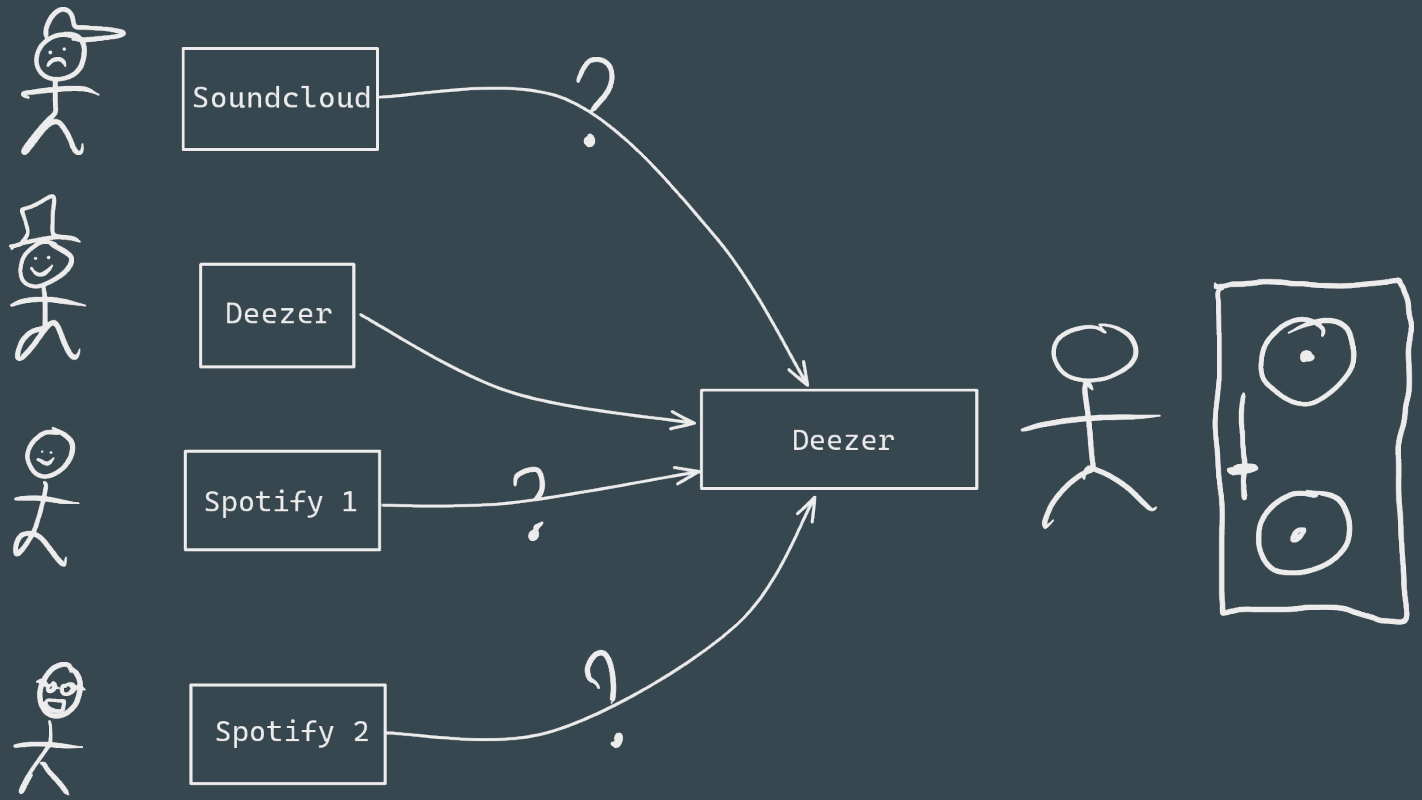
- Open source
- Gratuit
- 100% Python
- Interface Graphique élégante
- Quantité de flux
- Scalabilité

Inconvénients

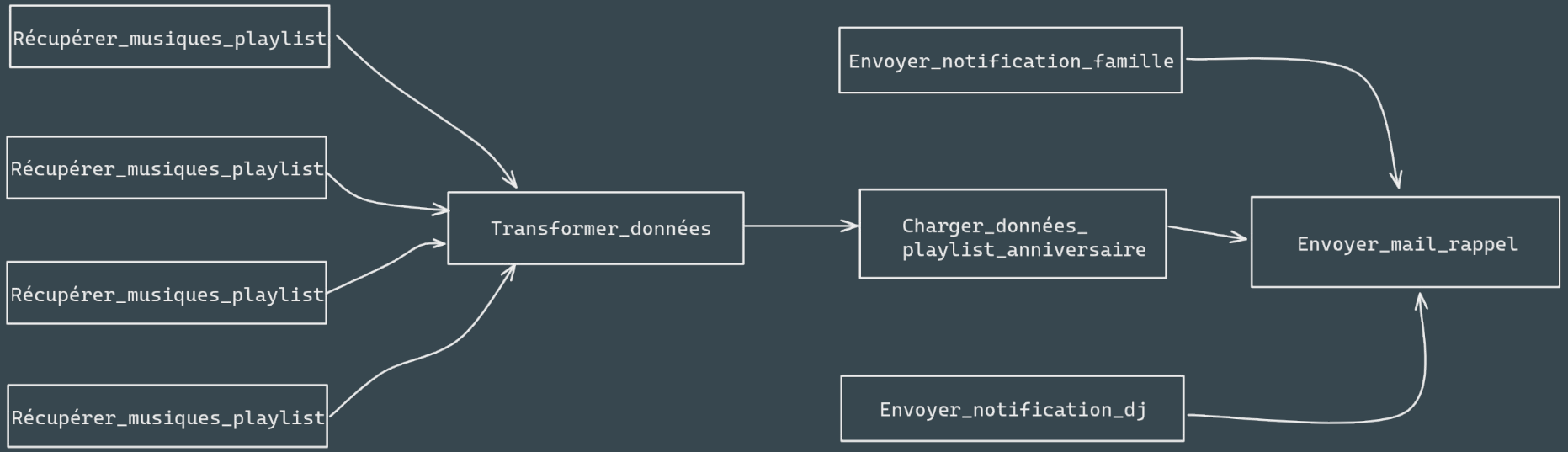
- Utilisation dynamique des tâches
- Modularité des DAG
- Besoin d’une infrastructure

Démonstration

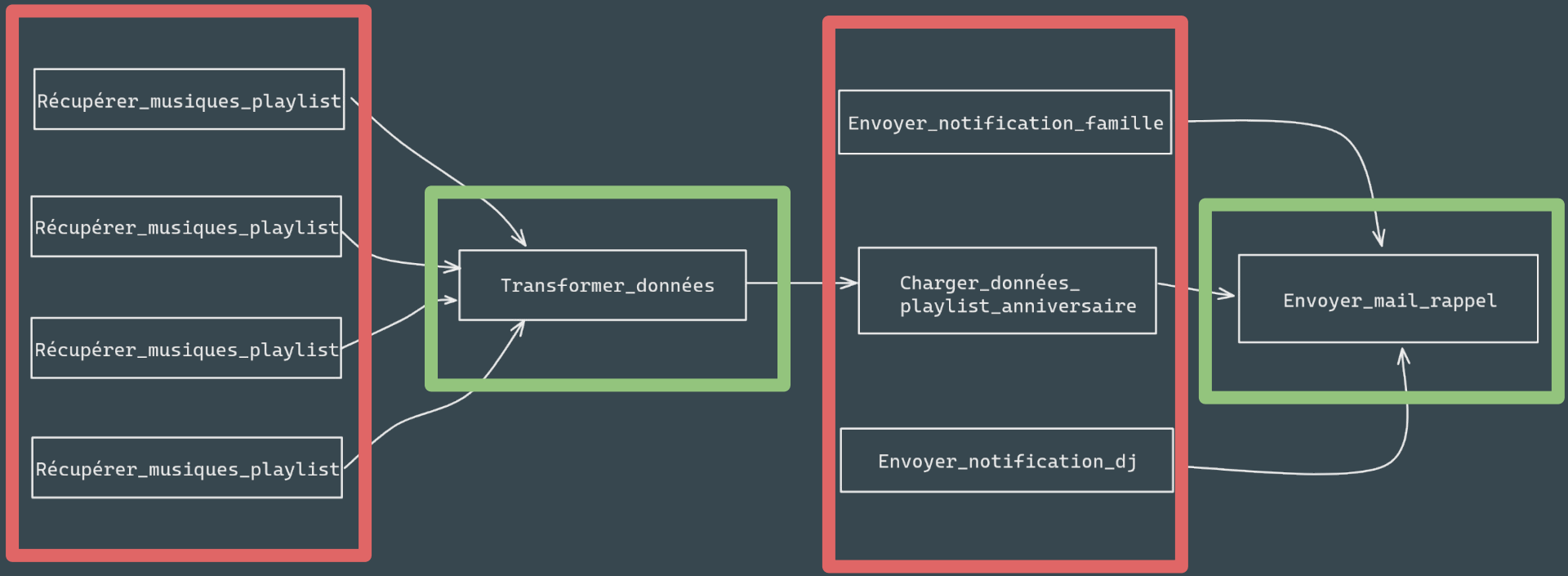
Démonstration - Vue générale



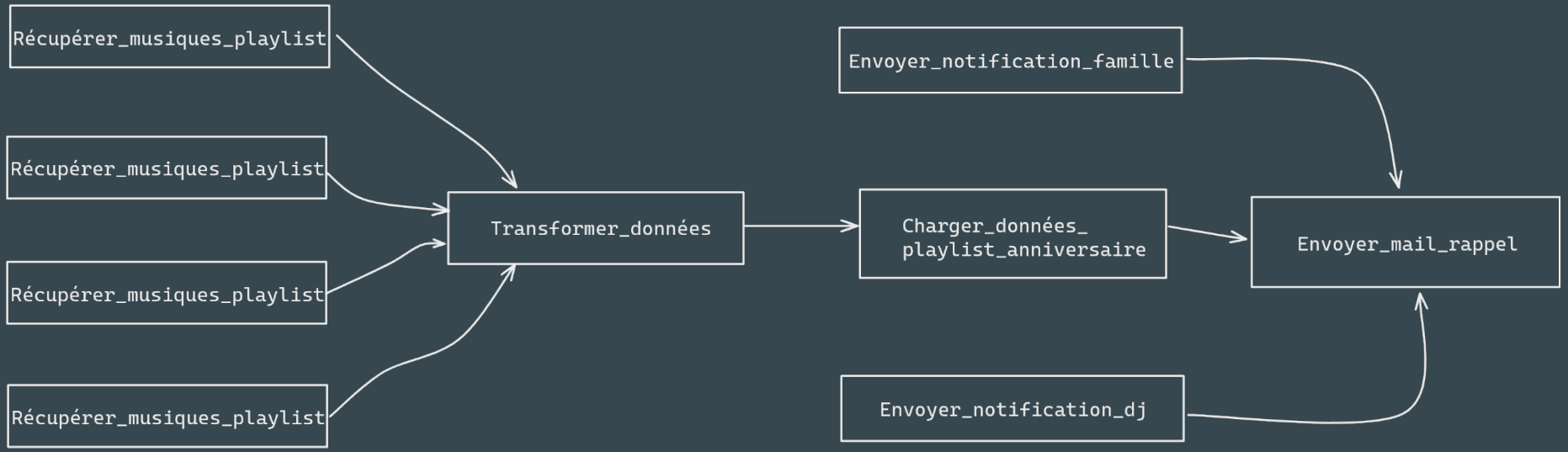
Démonstration - Vue générale (suite)



Démonstration - Vue générale (suite)



Démonstration - Dags



Cas d’utilisation

Cas d’utilisation

- ETL pipelines extrayant des données depuis plusieurs sources, lance des jobs où d’autres transformations de données

- Entrainement de modèle de machine learning

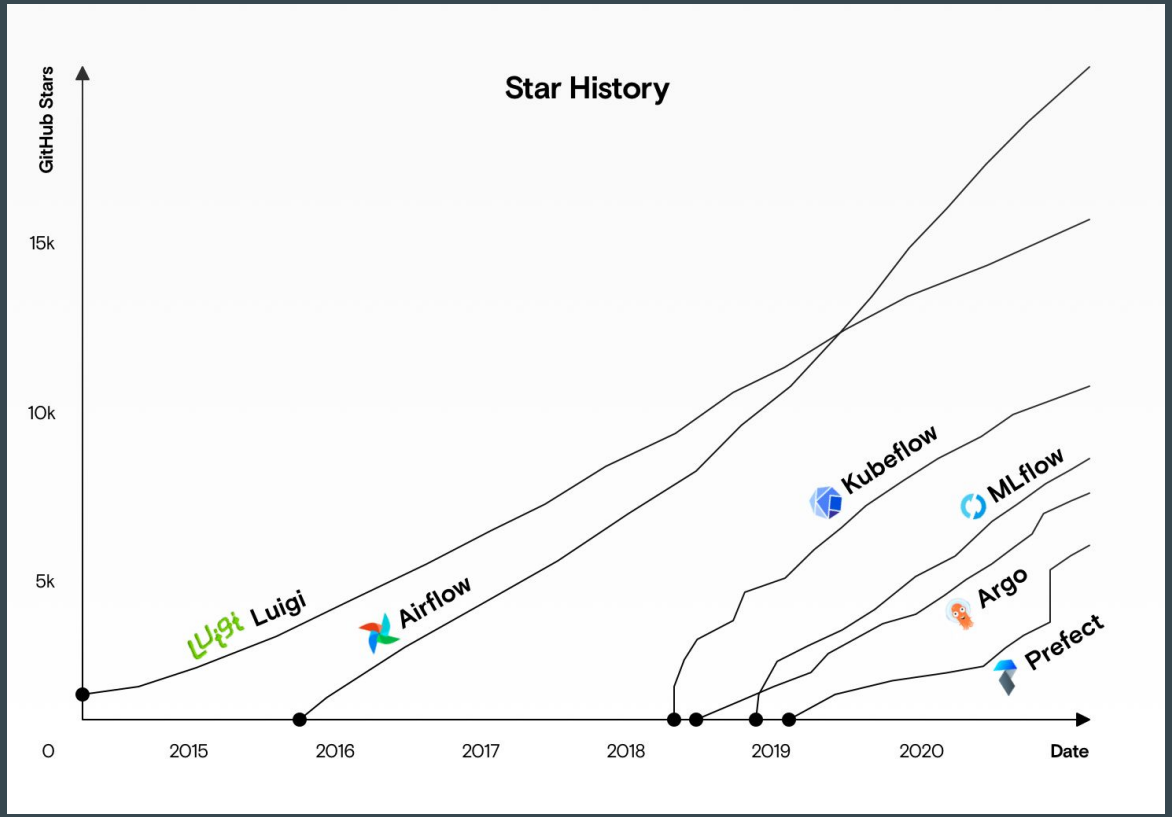
- Génération automatisée de rapports

- Backups et tâches DevOps

Concurrence

Concurrence

Concurrence



https://www.datarevenue.com/en-blog/airflow-vs-luigi-vs-argo-vs-mlflow-vs-kubeflow#:~:text=Airflow%20is%20an%20Apache%20project,learning%20how%20to%20use%20it.

Conclusion & Discussion

Soutenez l'open source

Merci de votre écoute