Apache Airflow

Por Alexandre Castanheira

Requisitos para essa apresentação

CONHECER ALGUMA
LINGUAGEM DE PROGRAMAÇÃO

ESTAR SE PERGUNTANDO:

O QUE DIACHO É O AIRFLOW?”

Cronograma

- O que é o Airflow
- Airflow na engenharia de Dados
- Principais componentes do Airflow
- Recomendações para uso da ferramenta
- Hands on
- Dúvidas

O que é o Airflow?

O que é o Airflow

- Surgiu em outubro de 2014 dentro do **Airbnb**
- Incubado pela fundação **Apache** em janeiro 2019
- Criado para gerenciar fluxos de trabalho complexos
- Escrito em Python (mas executa “qualquer” coisa)
- “Configuration as code”
- É um “orquestrador de workflow”

O que é um orquestrador?

Resposta longa:

- Ele coordena o fluxo de trabalho, sem necessariamente processar nada diretamente.
- Ele agenda a execução de cada processo e tem um controle sobre as dependências entre eles.
- Sua visualização acaba funcionando como uma documentação do processo.

O que é um orquestrador?

Resposta curta:

É uma CRONTAB mais bonita e inteligente.

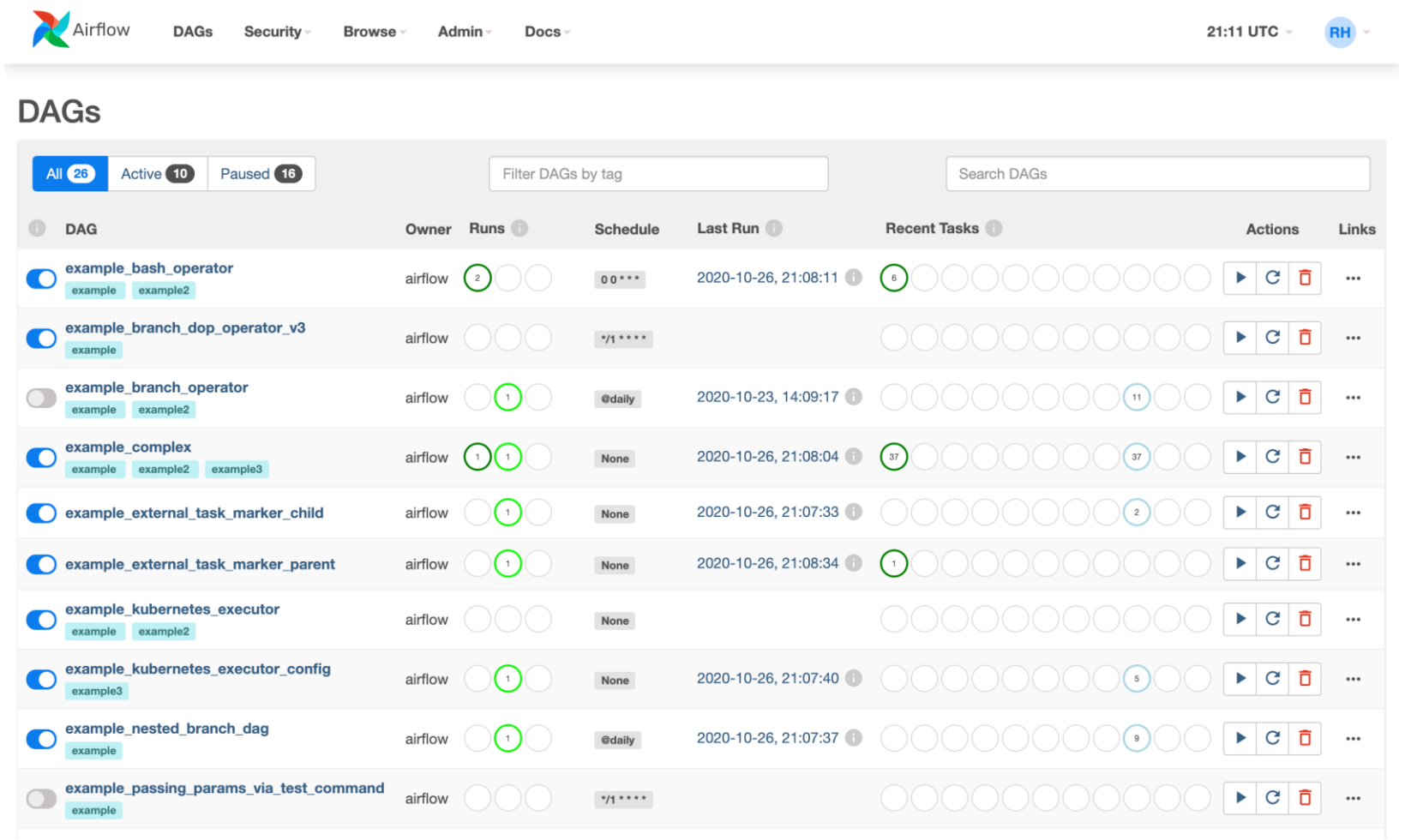
Exemplo de uma crontab

```
devconnected@debian-10:/etc/cron.d$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
```

Exemplo da interface do Airflow



Por que usar um orquestrador?

- Workflows complexos
- Monitoramento de execução
- Fácil reprocessamento de tasks
- Divisão de responsabilidades
- Facilidade para adicionar novos jobs
- CRONTAB ☹

Airflow na Engenharia de Dados

- Sem um orquestrador o Engenheiro de Dados provavelmente enlouqueceria.
- Possui built-in diversas integrações prontas para processos comuns de ETL, como conexão com diversos bancos de dados e serviços na cloud.
- É fácil o suficiente para pessoas de fora da engenharia de dados acompanharem sua interface.
- Suporta operações idempotentes.

Idempotência

"Em matemática e ciência da computação, a idempotência é a propriedade que algumas operações têm de poderem ser aplicadas várias vezes sem que o valor do resultado se altere após a aplicação inicial."

Idempotência

- Um UPDATE **é** uma operação idempotente, um INSERT, **não**.

- Outro exemplo:

```
1   # isso NÃO É idempotente
2   from datetime import datetime
3   print(datetime.now())
4
5   #isso É idempotente
6   print("2022-05-05 14:18:35.977027")
```

Idempotência no Airflow

- Fácil de reproduzir algum erro.
- É possível reprocessar uma task que rodou em um dia específico sem precisar alterar o código.
- Sem risco de duplicação de dados. (se todos fizerem sua parte =])

Principais Componentes

Tasks

Uma task é a representação de uma única tarefa (sério?) no fluxo da DAG.

- Ex.:
  - Obter um Dataset do banco de dados
  - Consultar um valor em uma API
  - Salvar dados no Data Lake

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/concepts/tasks.html

Operators

- É um objeto Python que vai te ajudar a criar sua Task
- É possível construir operators personalizados se necessário.
- O Airflow possui Operators prontos para tarefas comuns, exemplos:

  - **PythonOperator**: Permite executar códigos Python
  - **BashOperator**: Permite executar um código bash (ou qualquer coisa que pudesse ser executado em um terminal, incluindo scripts de outras linguagens)
  - **S3ListOperator**: Faz uma listagem de objetos em um repositório S3.
  - **SFTPOperator**: Operador para transferência de arquivos entre um host local e remoto (e vice versa)

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/concepts/operators.html

XCom

- É a forma com que uma task se comunica com outra.
- Os dados ficam armazenados em banco de dados.
- Os dados podem ser recuperados mesmo ao reprocessar uma tarefa.

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/concepts/xcoms.html

Templates Jinja

No código, podemos usar diversos templates pré-definidos pelo Airflow para obtermos mais informações em relação ao contexto da execução.

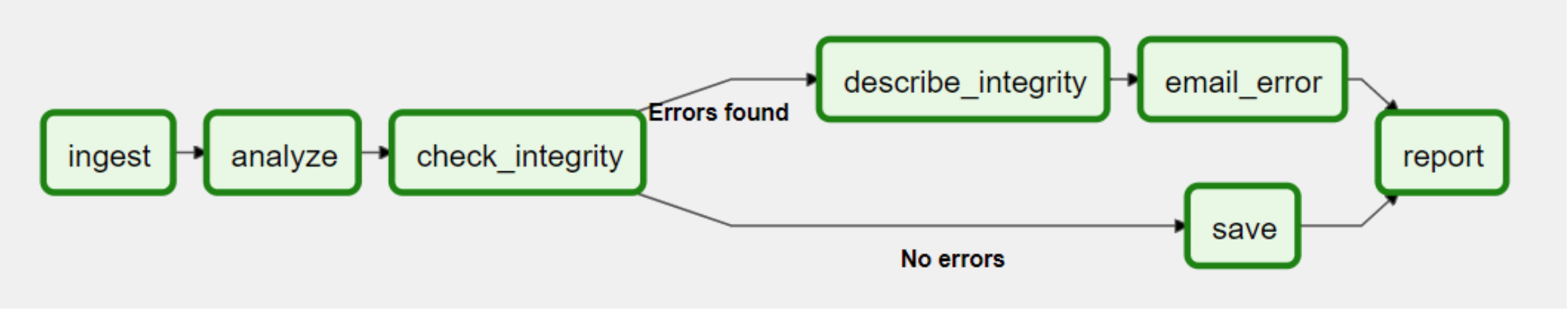
Exemplos:

- {{ data_interval_start }}: Início do intervalo de execução
- {{ params }}: Parâmetros que podem ser passados na ativação da DAG.
- {{ next_execution_date }}: Próxima data de execução.
- {{ prev_start_date_success }}: Última vez que essa DAG foi bem sucedida na execução.

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/templates-ref.html

DAG

- É composta por um conjunto de Tasks que podem ou não possuir uma dependência entre elas.
- Directed Acyclic Graph ou Gráfico Acíclico Dirigido é um gráfico parecido com isso:



Referência:
https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html

Connections e Variables

- Ambos são cadastrados na interface do Airflow e ficam armazenados no banco de dados
- Podem estar criptografados na interface
- Elas existem para não ter chaves e outros dados sensíveis ou mutáveis demais no código.
- As **variables** são como variáveis de ambiente, pode ser armazenado desde um único texto, até um JSON complexo.
- Uma **connection** possui todas as informações necessárias a respeito de uma única conexão, ex: dados de um banco de dados (host, user, password, etc), conta AWS (access e secret Keys), etc.

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/howto/variable.html
https://airflow.apache.org/docs/apache-airflow/stable/howto/connection.html

Webserver e Shedule

São os dois principais **processos** do airflow.

- Webserver
  - É a interface gráfica do Airflow
  - No Webserver podemos monitorar e executar Jobs

- Scheduler
  - O Scheduler é o Job responsável por agendar as execuções de tarefas

Referência:
https://airflow.apache.org/docs/apache-airflow/stable/concepts/overview.html

O que eu gostaria que tivessem me dito quando aprendi Airflow?

Pontos de atenção

- Documentação.
- Muitas formas diferentes de se fazer a mesma coisa.
- É necessário atenção com o banco de dados e armazenamento da máquina em que o airflow está rodando.
- Funcionamento do agendamento de tarefas não é intuitivo.

Como funciona o agendamento de tarefas?

Ao criar a DAG, é necessário definir:

- start_date (obrigatório): Data em que o job poderá ser agendado.
- end_date (opcional): Após esta data, o job não será mais agendado.
- interval (obrigatório): A frequencia com que será executado.
- catchup (opcional): Se irá fazer um agendamento retroativo.

Importante: A primeira execução da DAG é sempre start_date + Interval

Como funciona o agendamento de tarefas?

Exemplo 1: DAG que pega métricas do Google Analytics do dia.

Levando em conta que:

- start_date: 2022-01-01
- interval: @daily
- **catchup: True**

- Por causa do **catchup True**, assim que você ligar a DAG, ele irá agendar uma execução para cada dia, desde o dia 01/01/2022.
- O primeiro dia que ele irá pegar dados será o dia 02/01/2022.

- Caso o catchup fosse **False**, ele iria levar em consideração o momento em que você **ligou** a DAG e executaria **imediatamente** após a DAG ser ativada (afinal a condição do start_date + Interval já foi atingida).
- E a execução seguinte nesse cenário seria: hora que a DAG foi ligada + interval.

Como evitar dores de cabeça com agendamentos?

- Caso o **catchup seja False**, coloque um **start_date** no **futuro**, pois você terá um controle maior de quando será executada.

- Caso o **catchup seja True**, tire proveito dos templates {{ execution_date }} e {{ next_execution_date }} para garantir a idempotência da execução.

Cola básica para o uso do catchup:

- Ingestão histórica? Catchup = True
- Geração de relatório? Catchup = False
- Dispara e-mails? Catchup = False

Recomendações para projetos

- Armazenar logs remotamente
- Use o airflow apenas para orquestração
- Cuidado com as “super dags”
- Evite reinventar a roda
- Se possível use serviços gerenciados

Hands On

https://github.com/aleCastanheira/brownbag_airflow

Dúvidas

Obrigado!

Contato:

www.linkedin.com/in/alexandre-castanheira