
Apache
Airflow
Data Pipelines
com Apache Airflow
Stack Academy

Stack

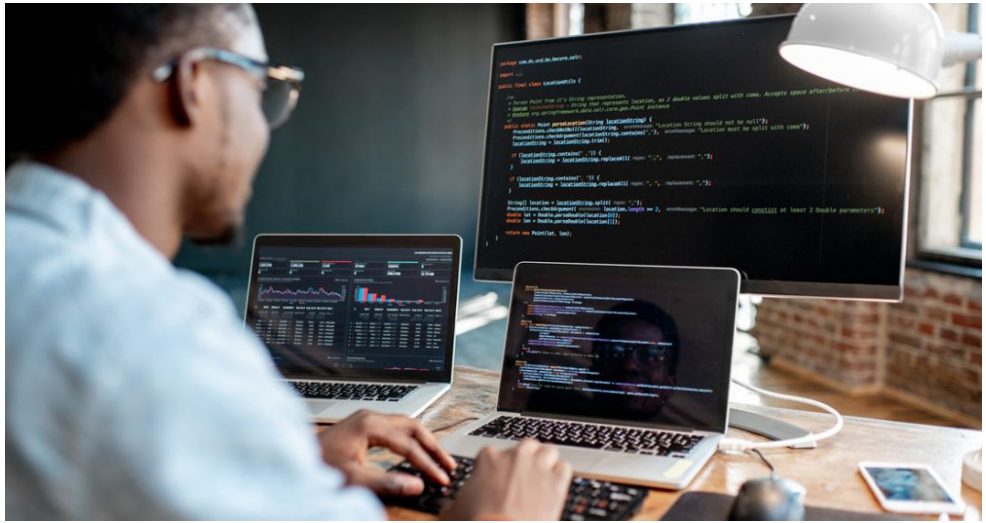
Objetivo

- Entender o que é um **Data Pipeline.**
- Desenvolver **pipelines e automatizar processos** utilizando o Apache Airflow.
- Ter mais autonomia e independência.
- Saber se **comunicar e trabalhar em equipe.**
- Estar pronto para **enfrentar qualquer desafio.**

Stack

Pra quem é esse curso?

- Analistas, Cientistas e Engenheiros de Dados.
- Gerentes e Heads de Tecnologia.



Stack

Responsabilidades dos Profissionais

- **Analistas e Cientistas de Dados.**
  - Especificar regras de negócio do projeto.
    - Níveis de acordo de serviço.
    - Disponibilidade dos dados e informações.
  - Desenvolver e manter os Pipelines de Dados.
    - Definir fontes de dados.
    - Especificar e desenvolver tarefas.
  - Monitoramento e análise.
    - Estados das tarefas.
    - Tempo de execução.
    - Agendamento.

Stack

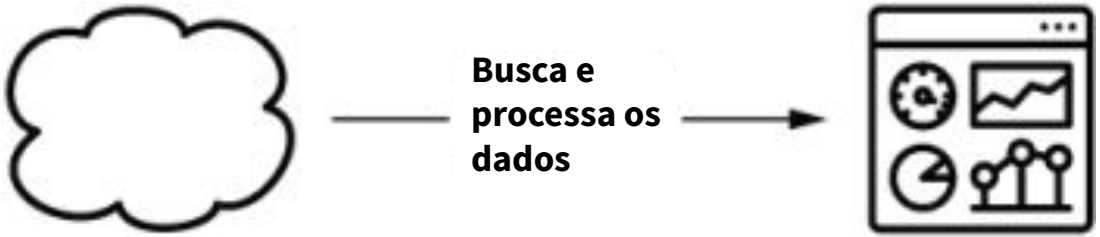
Responsabilidades dos Profissionais

- **Engenheiro de Dados**
  - Desenvolver e manter os Pipelines de Dados
    - Definir ambiente e arquitetura.
      - On Premisse, On Cloud
    - Especificar tecnologias para armazenamento.
      - Data Lakes, Data Warehouse.
    - Definir formas de processamento.
      - Streaming, Clusters.
    - Definir e implementar padrões e segurança e integração com outros sistemas.
    - Monitoramento.

Stack

Introdução aos Data Pipelines

Data Pipelines basicamente consiste de algumas tarefas ou ações que devem ser executadas para atingir um resultado.



Stack

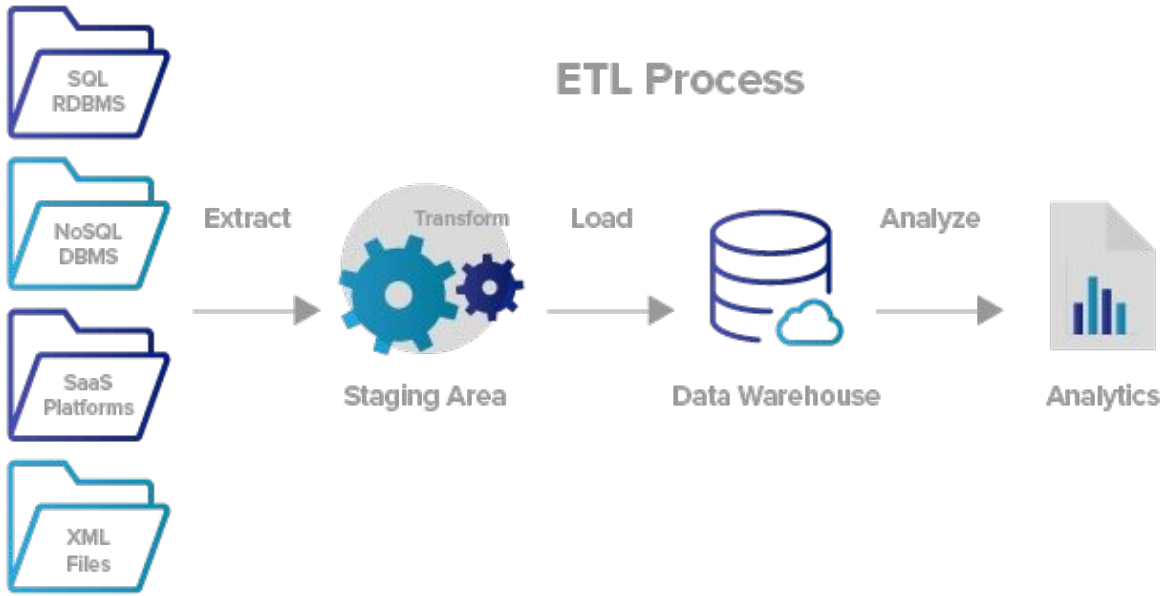
Introdução aos Data Pipelines

1. Busca os dados de temperatura de uma API.
2. Limpa e transforma os dados.
3. Insere os dados transformados em uma base de dados para ser exibido no Dashboard.

Stack

Alguns Pipelines

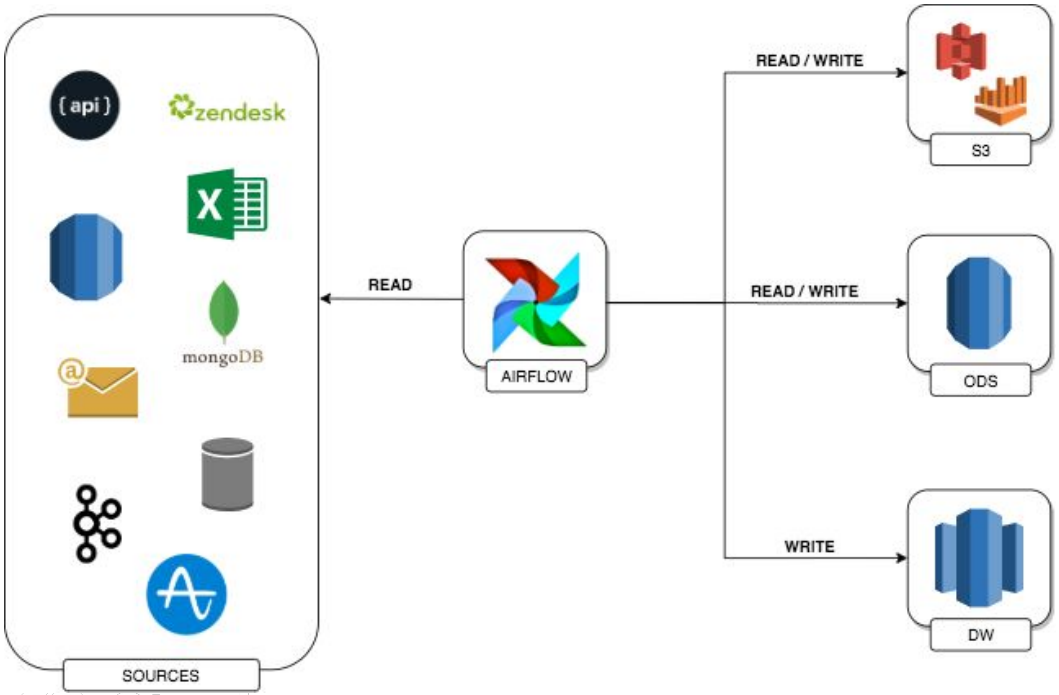
ETL (Extração, Transformação e Carga).



Stack

Alguns Pipelines

ETL com Airflow.

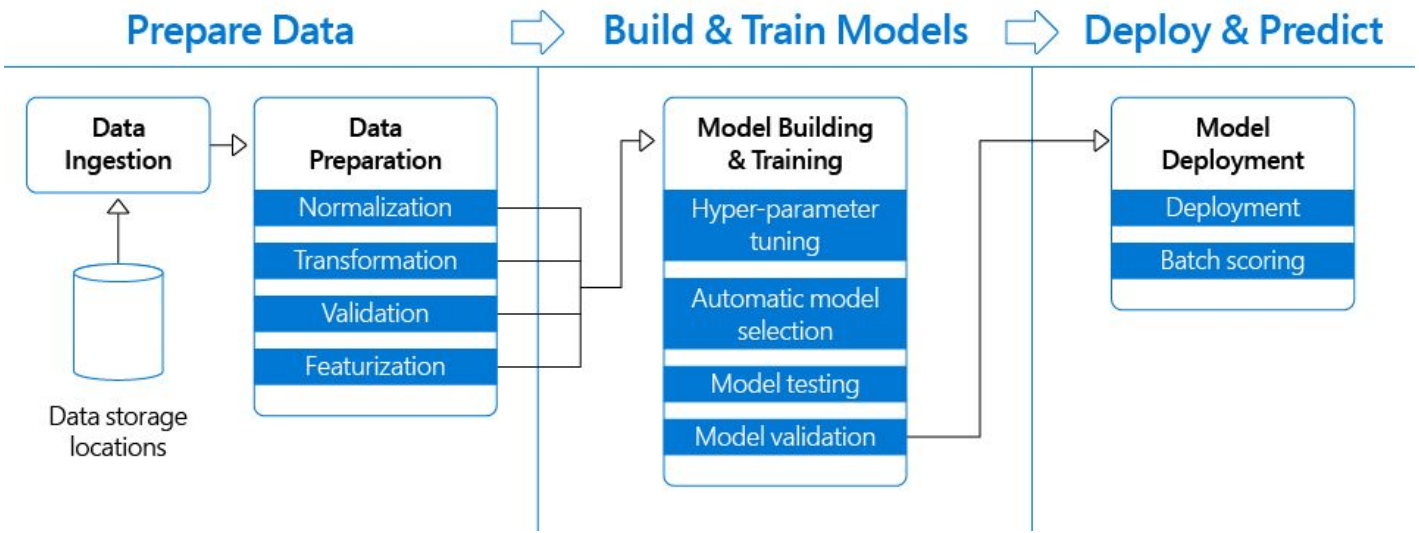


Stack

Stack

Alguns Pipelines

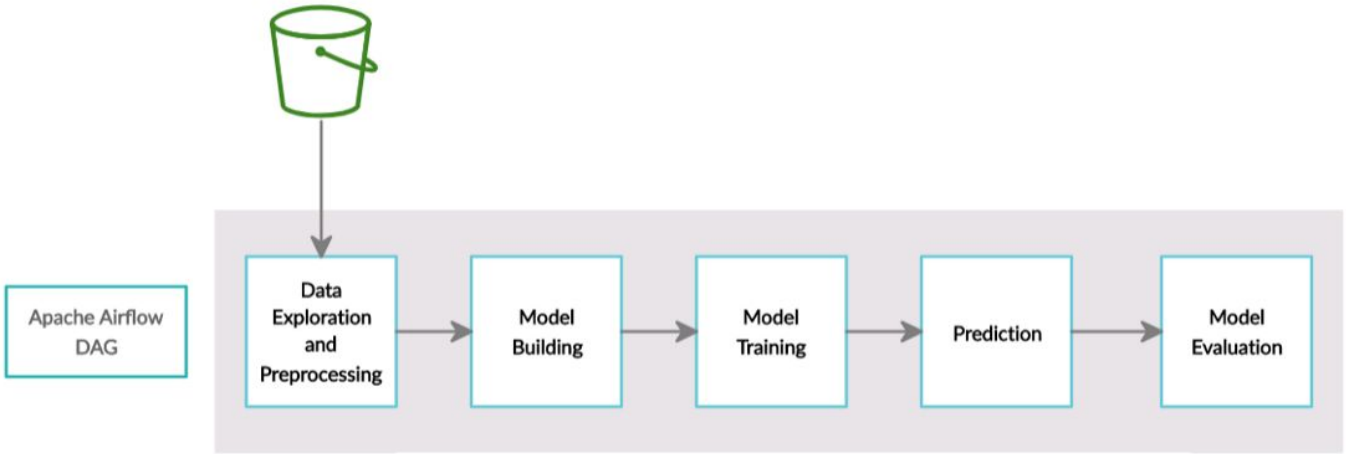
Machine Learning Pipeline.



Stack

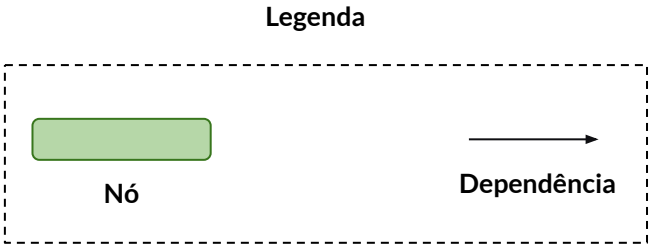
Alguns Pipelines

Machine Learning Pipeline.



Stack

Representação em grafos do Pipeline de exemplo

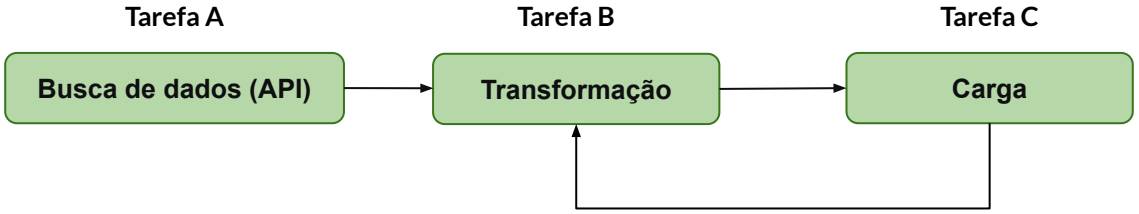


Stack

Representação em grafos do Pipeline de exemplo

Grafo direcionado acíclico.

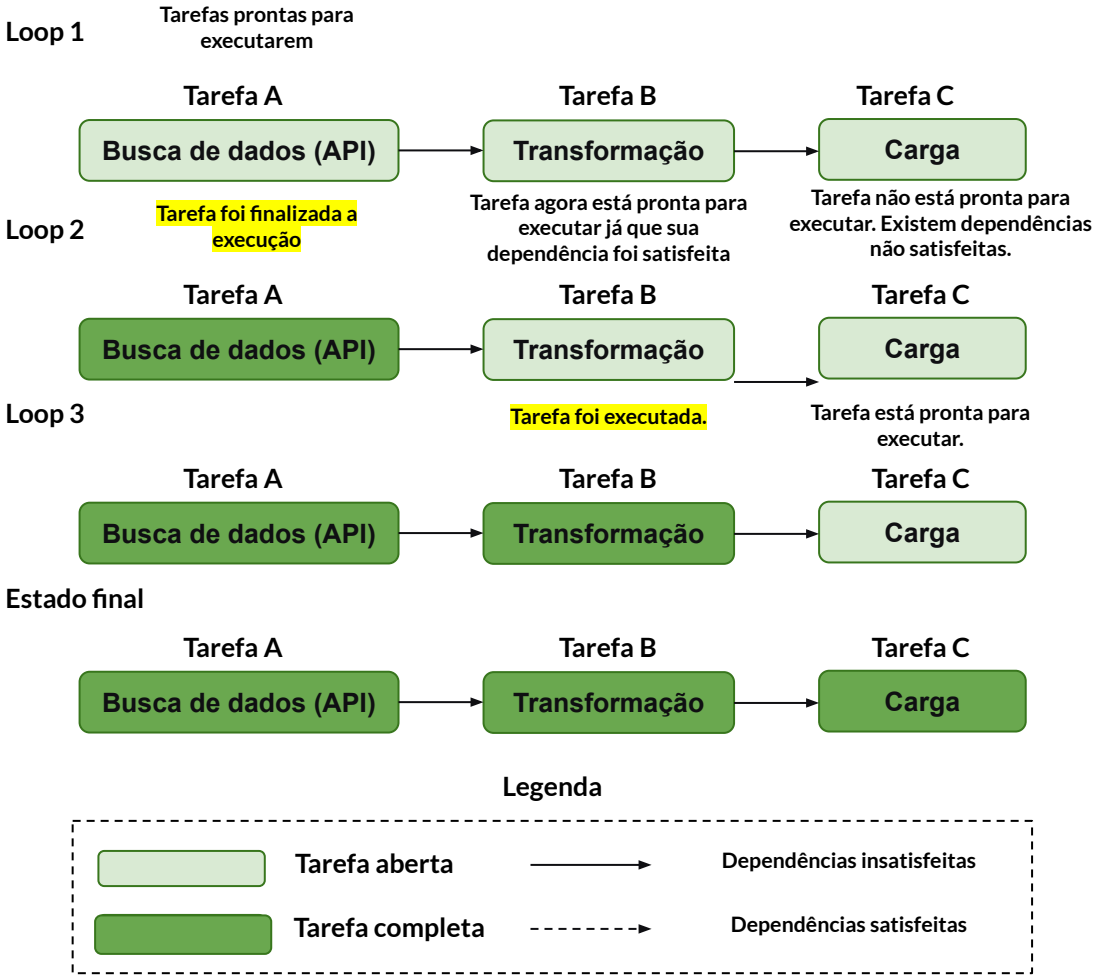
Grafo direcionado cíclico.



A Tarefa B não será executada pois precisa da execução da tarefa C. A tarefa C também não pode executar pois é dependente da tarefa B (Deadlock)

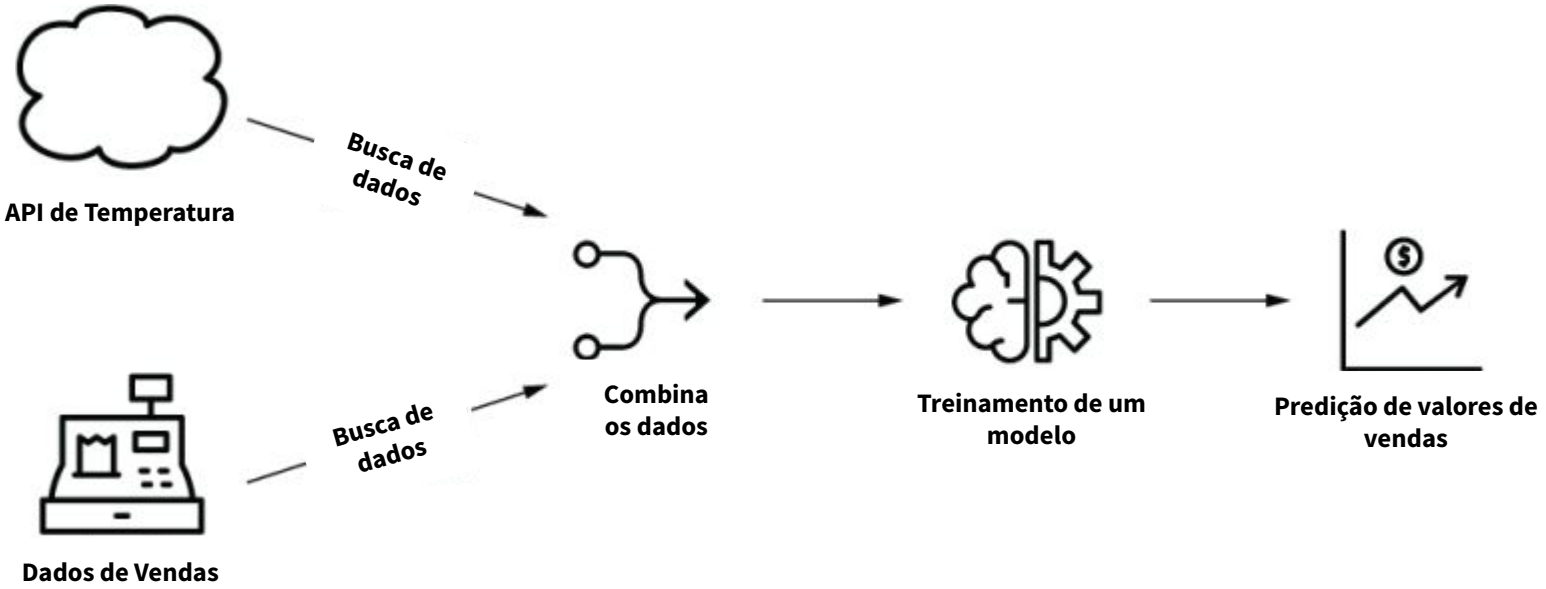
Stack

Loops e Estados das Tarefas



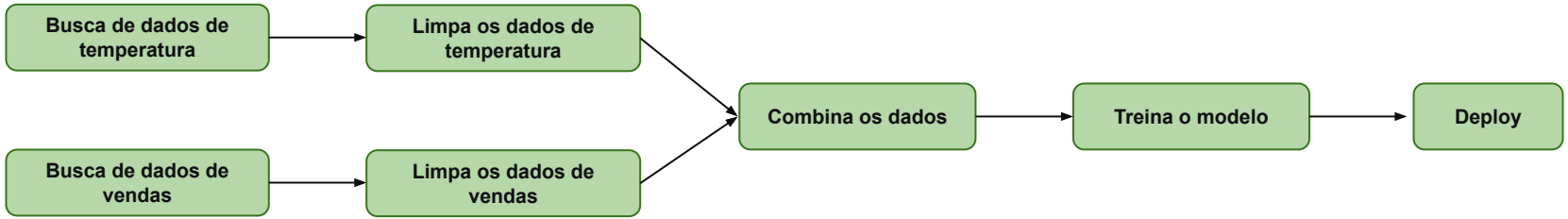
Stack

Pipelines em grafos vs scripts sequenciais



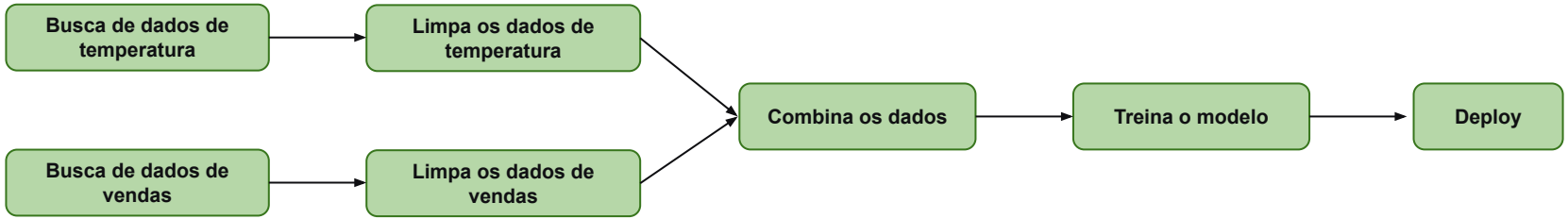
Stack

Pipelines em grafos vs scripts sequenciais



Stack

Pipelines em grafos vs scripts sequenciais



Stack

Apache
Airflow

Stack

O que é o Apache Airflow ?

- Tecnologia open source para o desenvolvimento e monitoramento de Workflows.

Porque escolher o Airflow?

- Implementação de completos e complexos Pipelines utilizando código Python.
  - Versionamento.
- Fácil integração com outros sistemas por conta da comunidade.
- Ricas opções para trabalhar com agendamento.
  - Processamento incremental.
  - Redução de custos.
- Backfilling para reprocessar dados históricos.
- Interface Web para Monitoramento e Debugging.
- Open Source e API's

Stack

Airflow vs Scripts Bash

- Operadores
  - Fácil e Rápido desenvolvimento.
  - Integração com outros sistemas.
  - Sensores.
- Gerenciamento de erro, notificações e log.
- Controle de fluxo de execução.
- Sistema de agendamento avançado.
  - Backfilling.
- Executores para processamento distribuído.
- Interface Web amigável.
- Implementação em ambientes Cloud.

Stack

Airflow vs outros Workloads Managers

Table 1.1 Overview of several well-known workflow managers and their key characteristics.

Name	Originated at[a]	Workflows defined in	Written in	Scheduling	Backfilling	User interface[b]	Installation platform	Horizontally scalable
Airflow	Airbnb	Python	Python	Yes	Yes	Yes	Anywhere	Yes
Argo	Applatix	YAML	Go	Third party[c]		Yes	Kubernetes	Yes
Azkaban	LinkedIn	YAML	Java	Yes	No	Yes	Anywhere	
Conductor	Netflix	JSON	Java	No		Yes	Anywhere	Yes
Luigi	Spotify	Python	Python	No	Yes	Yes	Anywhere	Yes
Make		Custom DSL	C	No	No	No	Anywhere	No
Metaflow	Netflix	Python	Python	No		No	Anywhere	Yes
Nifi	NSA	UI	Java	Yes	No	Yes	Anywhere	Yes
Oozie		XML	Java	Yes	Yes	Yes	Hadoop	Yes

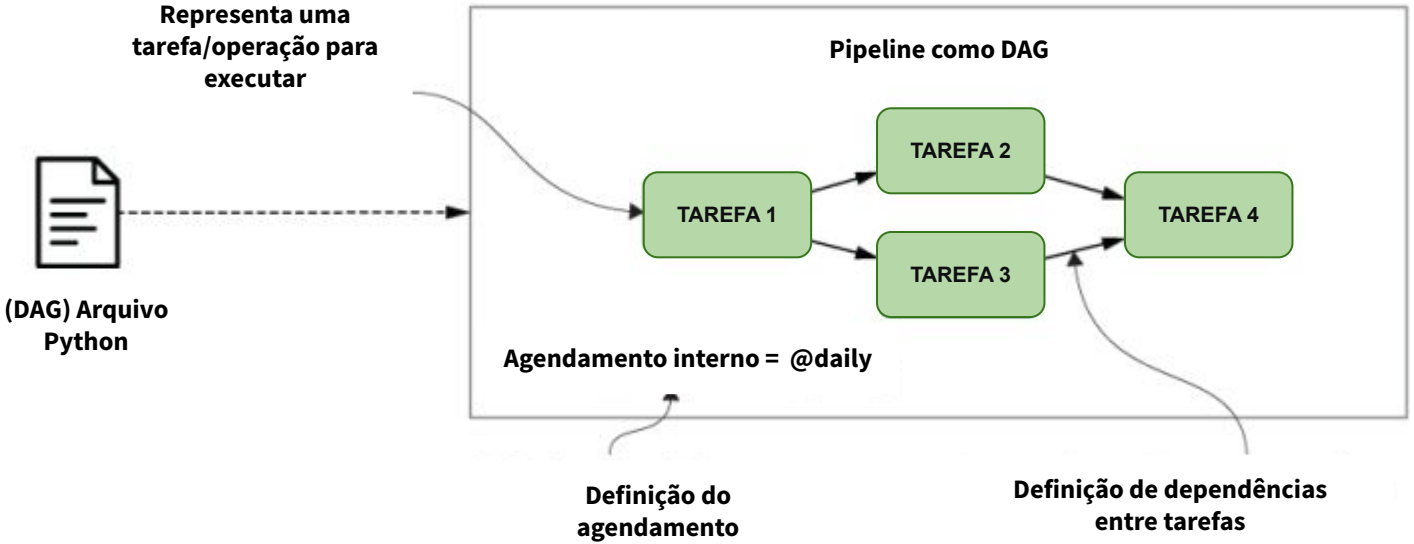
Stack

Quando NÃO utilizar Airflow?

- Necessidade de manipular pipelines em Streaming.
- Equipe com pouco ou nenhum conhecimento em Python.
  - Alternativas Azure Data Factory, NIFI, Pentaho.
- Pipelines grandes e complexos podem exigir uma grande organização.

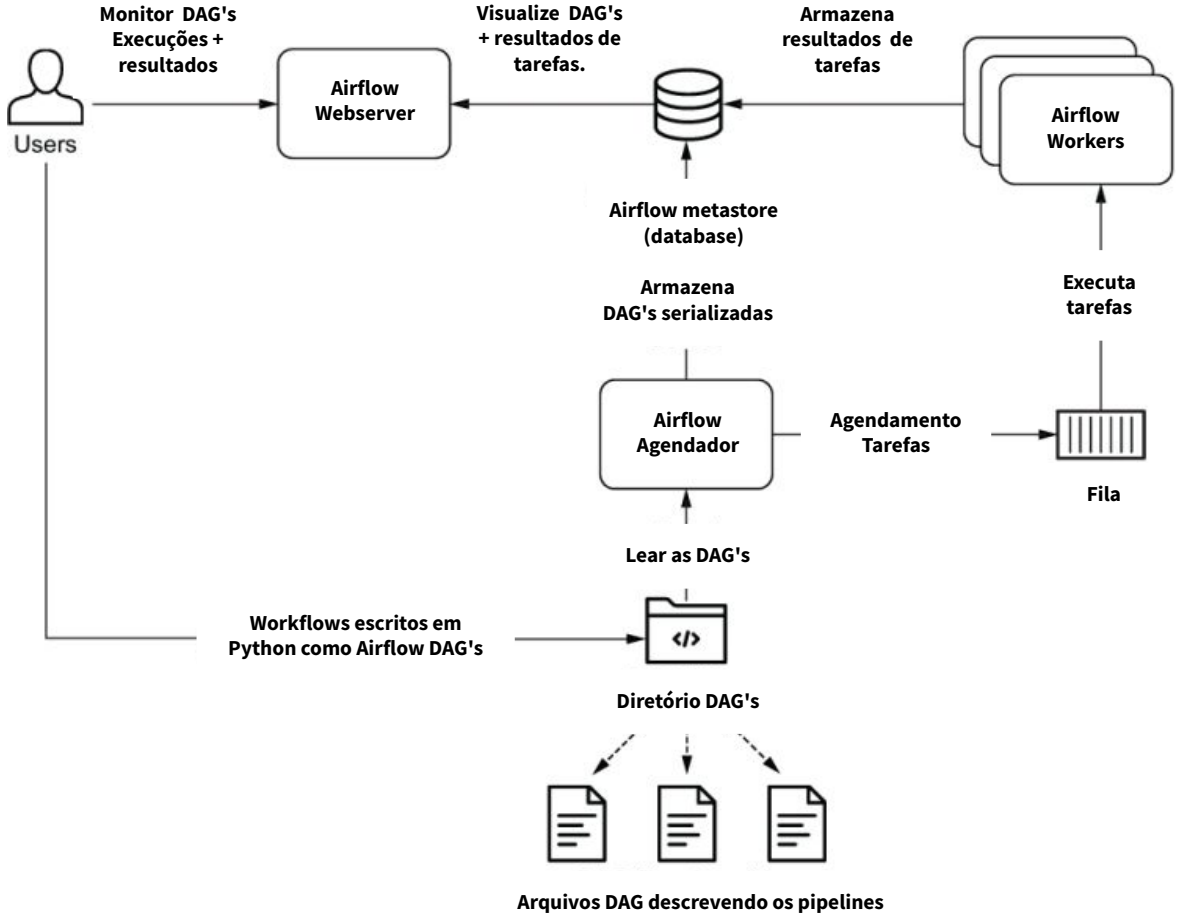
Stack

Definição de Pipelines com a Flexibilidade do Código Python



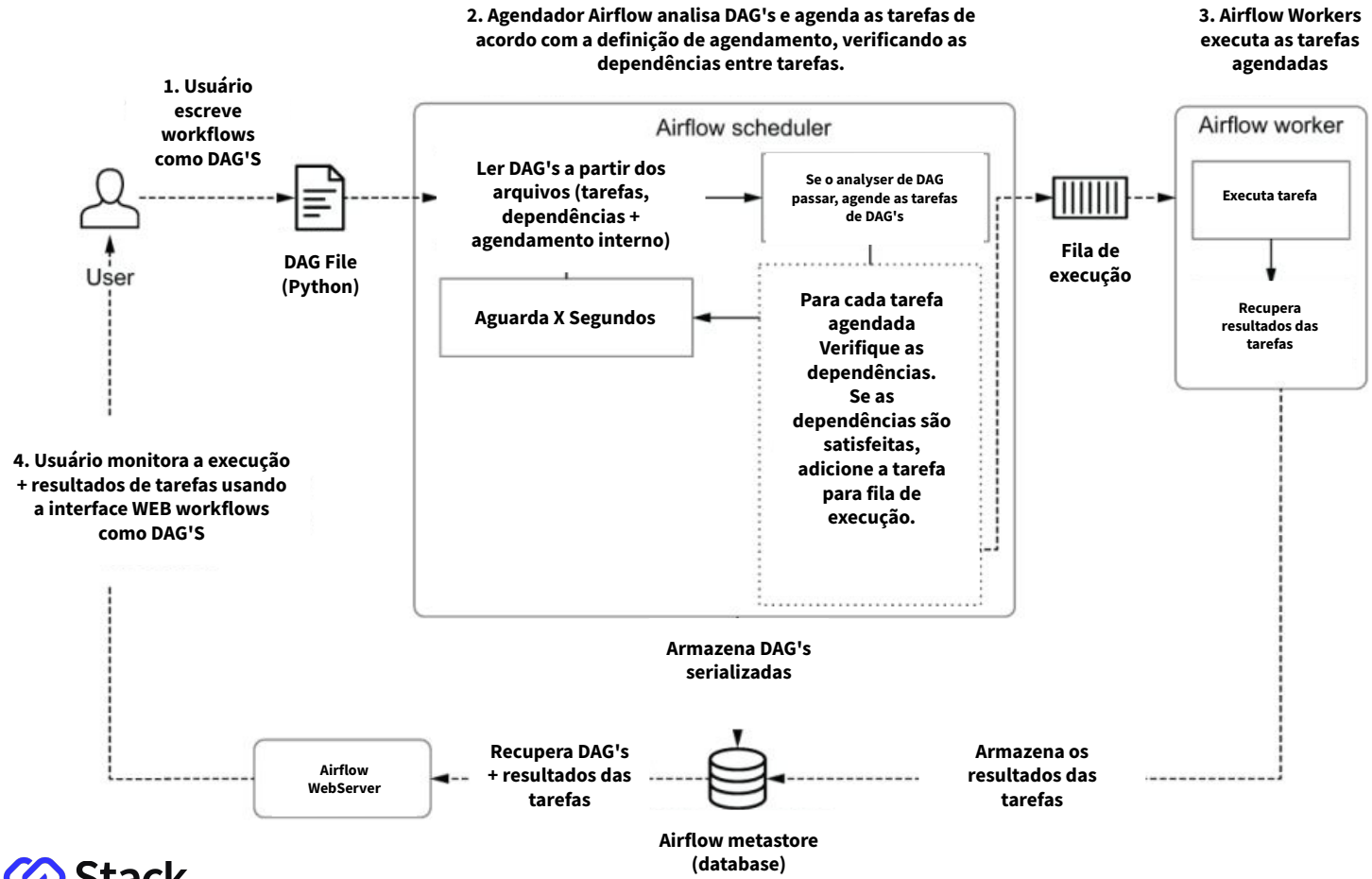
Stack

Componentes/Arquitetura do Airflow



Stack

Componentes/Arquitetura do Airflow

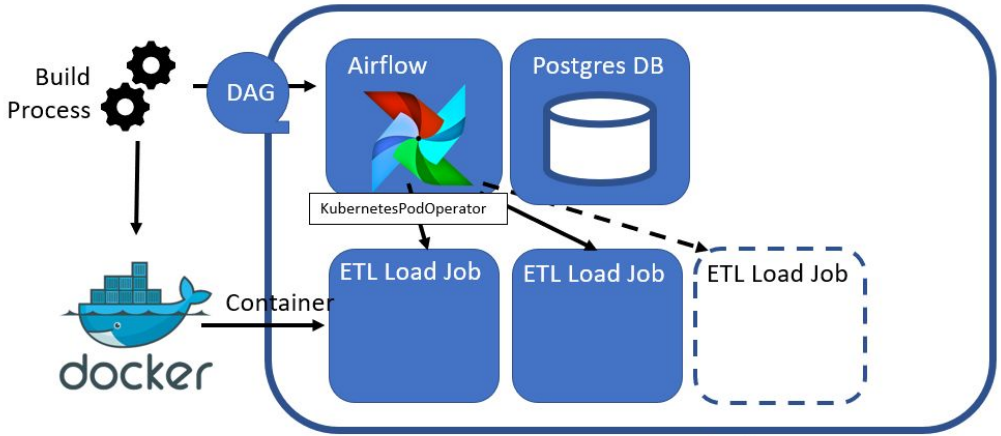


Stack

Vamos colocar a mão na massa.

Stack

Instalando o Airflow utilizando o Docker



Stack

Instalando o Airflow utilizando o Docker

```
docker run -d -p 8080:8080 -v "$PWD/dags:/opt/airflow/dags/" \
--entrypoint=/bin/bash \
--name airflow apache/airflow:2.1.1-python3.8 \
-c '(airflow db init && \
    airflow users create --username admin --password abc123 --firstname Felipe --lastname
Lastname --role Admin --email admin@example.org
    ); \
airflow webserver & \
airflow scheduler \
'
```

Stack

DAG Hello World

```python
from airflow import DAG
import airflow.utils.dates
from airflow.operators.bash import BashOperator

dag = DAG(
    dag_id="hello_world",
    description="A primeira DAG de teste do airflow",
    start_date=airflow.utils.dates.days_ago(14),
    schedule_interval="@daily",
)

task_echo_message = BashOperator(
    task_id="echo_message",
    bash_command="echo Hello World!",
    dag=dag,
)


task_echo_message
```

Stack

Airflow Command Line Interface

Conectando ao container.
docker container exec -it airflow bash

Listando os comandos disponíveis.
airflow --help

Verificando as informações do ambiente.
airflow info

Verificando as configurações.
airflow config list

Stack

Airflow Command Line Interface

Verificando as conexões do ambiente.
airflow connections list

Listando as dags disponíveis.
airflow dags list

Listando as tasks de uma determinada dag.
airflow tasks list <dag-id>
airflow tasks list hello_world

Testando a execução de uma determinada task.
airflow <tasks> <test> <dag_id> <task_id> <execution-date>
airflow tasks test hello_world echo_message 2021-07-01

Stack

Conceitos

Workloads

Uma DAG é composta por tarefas que podem ser de três tipos.

- **Operadores** que são tarefas predefinidas que agrupadas formam a execução da DAG.
- **Sensores** que é uma subclasse especial de Operadores que trabalham aguardando um evento externo acontecer.
- **Taskflow** que é uma função Python personalizada empacotada como uma Tarefa.

Conceitos

DAGs - Core do Airflow.

- Reuni tarefas juntas.
- Especifica suas dependências e relacionamentos.
- Define como e quando serão executadas.

Declarando uma DAG.
Através de um Context manager.

- **with DAG("etl-db-producao") as dag:**
  op = DummyOperator(task_id="task")

Através de um construtor padrão.

- **my_dag = DAG("etl-db-producao")**
  op = DummyOperator(task_id="task", dag=my_dag)

Stack

Conceitos

Através de um decorator - transforma uma função em uma dag.

- ```
  @dag(start_date=days_ago(2))
  def generate_dag():
      op = DummyOperator(task_id="task")
  dag = generate_dag()
  ```

Stack
  ```

Conceitos

Executando DAG's

DAG's são executadas de duas formas.

- Acionamento manual ou por API.
- Intervalo de agendamento.
  - `with DAG("etl-db-producao", schedule_interval="@daily"):`
  - `with DAG("etl-db-producao", schedule_interval="0 * * * *"):`

Argumentos padrão

```
default_args = {
  'start_date': datetime(2016, 1, 1),
  'owner': 'airflow'
}

with DAG('etl-db-producao', default_args = default_args) as dag:
    op = DummyOperator(task_id='dummy')
    print(op.owner)
```

Stack

Conceitos

Control Flow

Tarefas têm dependências entre elas

- **extracao >> [transformacao, carga]**
- **carga << notificacao**

Podemos definir as dependências através dos métodos.

- **extracao.set_downstream([transformacao, carga])**
- **carga.set_upstream(notificacao)**

Stack

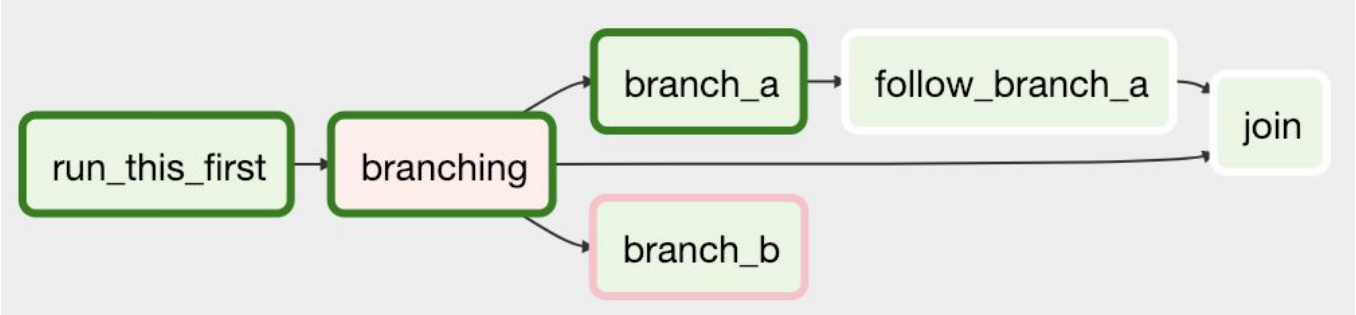
Control Flow

Formas de controlar a execução de tarefas.

- **Branching**
  - Determina qual tarefa mover a partir de uma condição.
- **Latest Only**
  - Só é executada em DAGs em execução no presente.
- **Depends on Past**
  - Tarefas podem depender de si mesmas de uma execução anterior.
- **Trigger Rules**
  - Permite definir as condições para um DAG executar uma tarefa.

Control Flow

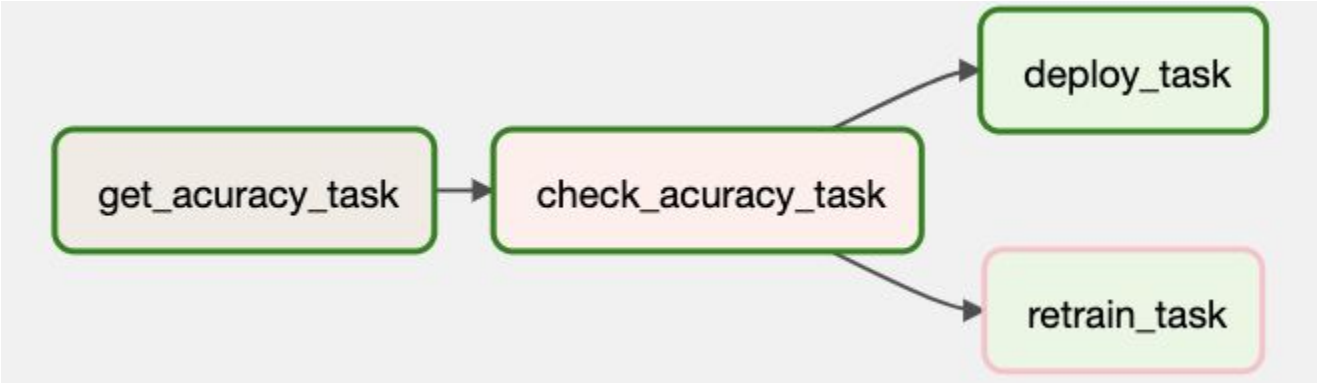
Branching



Stack

Control Flow

Branching



Stack

Control Flow

DAG - Branching

```python
from airflow import DAG
import airflow.utils.dates
from airflow.operators.bash import BashOperator
from airflow.operators.python import BranchPythonOperator
from airflow.operators.dummy import DummyOperator

dag = DAG(
    dag_id="branching-teste",
    description="DAG de exemplo para o conceito de Branching",
    start_date=airflow.utils.dates.days_ago(14),
    schedule_interval=None,
)

def check_acuracy(ti):
    acuracy_value = int(ti.xcom_pull(key="model_acuracy"))
    if acuracy_value >= 90:
        return 'deploy_task'
    else:
        return 'retrain_task'

get_acuracy_op = BashOperator(
    task_id='get_acuracy_task',
    bash_command='echo "{{ ti.xcom_push(key="model_acuracy", value=90) }}"',
    dag=dag,
)

check_acuracy_op = BranchPythonOperator(
    task_id='check_acuracy_task',
    python_callable=check_acuracy,
    dag=dag,
)

deploy_op = DummyOperator(task_id='deploy_task', dag=dag)
retrain_op = DummyOperator(task_id='retrain_task', dag=dag)

get_acuracy_op >> check_acuracy_op >> [deploy_op, retrain_op]
```

Stack

Control Flow

Latest Only

Este Operador especial ignora todas as tarefas posteriores de si mesmo se você não estiver na "última" execução do DAG.

Stack

Control Flow

Depends On Past
Uma tarefa só pode ser executada se a execução anterior da tarefa no DAG Run anterior foi bem-sucedida.

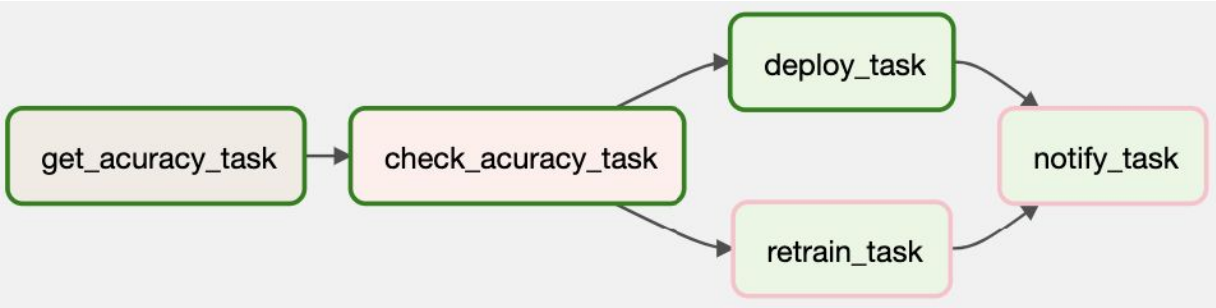
Trigger Rules
Nos permite controlar o comportamento de acionamento das tarefas.

- **all_success (padrão):** Todas as tarefas anteriores foram bem-sucedidas.
- **all_failed:** Todas as tarefas anteriores estão em um estado de falha ou upstream_failed.
- **all_done:** todas as tarefas upstream são feitas com sua execução.
- **one_failed:** pelo menos uma tarefa anterior falhou (não espera que todas as tarefas anteriores já tenham executado).
- **one_success:** pelo menos uma tarefa anterior foi bem-sucedida (não espera que todas as tarefas anteriores tenham executado).
- **none_failed:** Todas as tarefas anteriores não falharam ou **upstream_failed** - ou seja, todas as tarefas anteriores foram bem-sucedidas ou foram ignoradas.
- **none_failed_or_skipped:** Todas as tarefas anteriores não falharam ou **upstream_failed**, e pelo menos uma tarefa anterior foi bem-sucedida.
- **none_skipped:** nenhuma tarefa anterior está em um estado ignorado - ou seja, todas as tarefas anteriores estão em um estado de sucesso, falha ou upstream_failed.
- **dummy:** sem dependências, execute esta tarefa a qualquer momento.

Stack

Control Flow

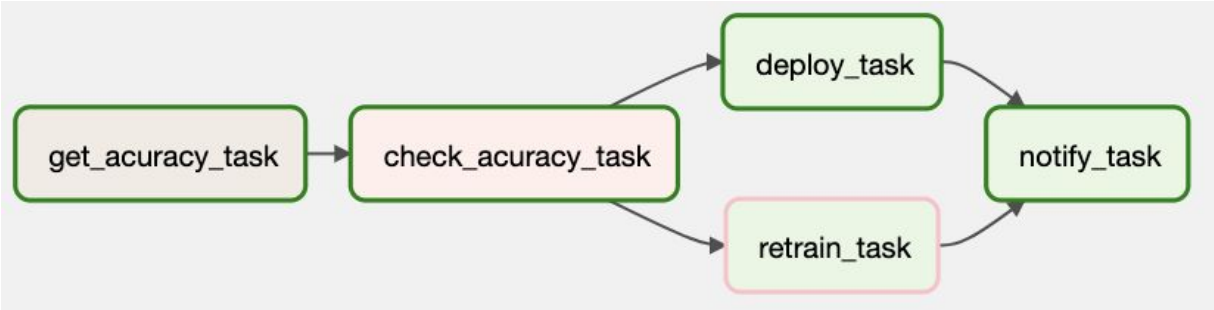
Exemplo da aplicação de trigger rules.



Stack

Control Flow

Exemplo da aplicação de trigger rules.
Alterando a sua trigger rule para NONE_FAILED.

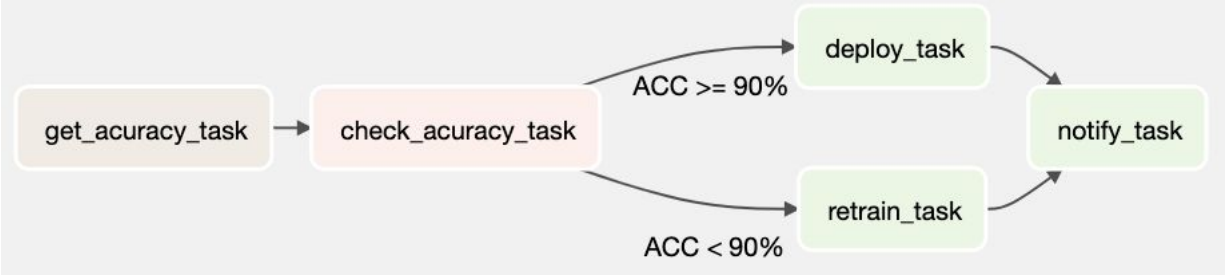


Stack

Edges Labels

Podemos definir labels para documentar as relações e dependências entre tarefas.

- **get_acuracy_op >> check_acuracy_op >> Label("Limit 90%") >> [deploy_op, retrain_op] >> notify_op**
- **get_acuracy_op.set_downstream(check_acuracy, Label("Métrica ACC"))**



Stack

Tasks

A unidade de execução mais básica do Airflow.
Podem ser de três tipos.

- **Operadores , Sensores e Taskflow.**

O relacionamento é definido através das dependências (upstream e downstreams).

Estados de cada tarefa.

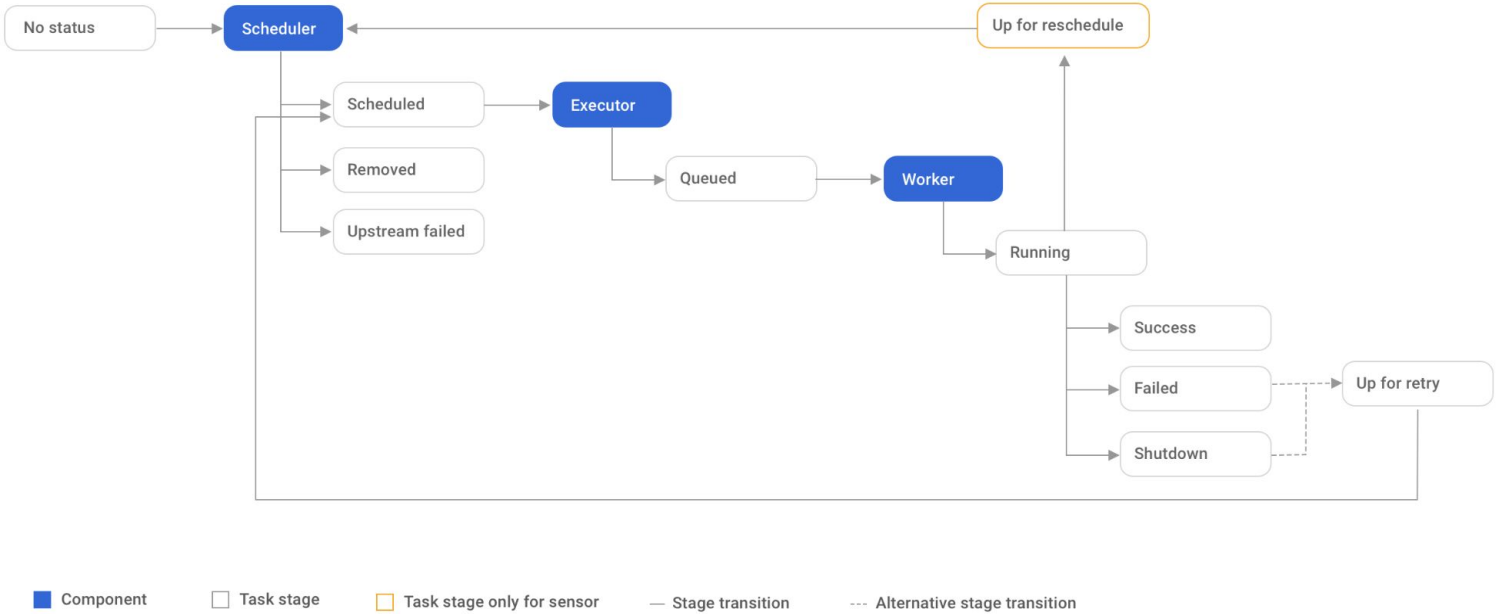
Stack

Tasks

Estados de cada tarefa.

- **none**: a tarefa ainda não foi enfileirada para execução (suas dependências ainda não foram atendidas).
- **scheduled**: O agendador determinou que as dependências da Tarefa são atendidas e deve ser executado.
- **queued**: A tarefa foi atribuída a um executor e está aguardando um trabalhador.
- **running**: a tarefa está sendo executada em um trabalhador (ou em um executor local / síncrono).
- **success**: a tarefa terminou em execução sem erro.
- **failed**: a tarefa teve um erro durante a execução e falhou ao executar.
- **skipped**: a tarefa foi ignorada devido a ramificação, LatestOnly ou semelhante.
- **upstream_failed**: uma tarefa upstream falhou e a regra de acionamento diz que precisávamos dela.
- **up_for_retry**: A tarefa falhou, mas ainda restam novas tentativas e será reprogramada.
- **up_for_reschedule**: A tarefa é um Sensor que está em modo de reprogramação.
- **sensing**: a tarefa é um sensor inteligente
- **removed**: a tarefa desapareceu do DAG desde o início da execução.

Tasks


No status
Scheduler
Up for reschedule
Scheduled
Executor
Removed
Queued
Worker
Upstream failed
Running
Success
Failed
Up for retry
Shutdown
Component
Task stage
Task stage only for sensor
Stage transition
Alternative stage transition

Stack

Conceitos

Transferência de dados entre tarefas.

- **X-coms (Cross-communications) - push e pull metadados.**
- **Uploading e Downloading em serviços de armazenamento (Bancos de dados, serviços, arquivos)**

Stack

Operators

Um operador é um template para uma tarefa pré-definida.

Exemplos:

- BashOperator - executa um comando bash.
- PythonOperator - chama uma função Python.
- EmailOperator - envia um e-mail.
- SimpleHttpOperator - envia uma requisição http.
- SqliteOperator - operador para trabalhar com sqlite.

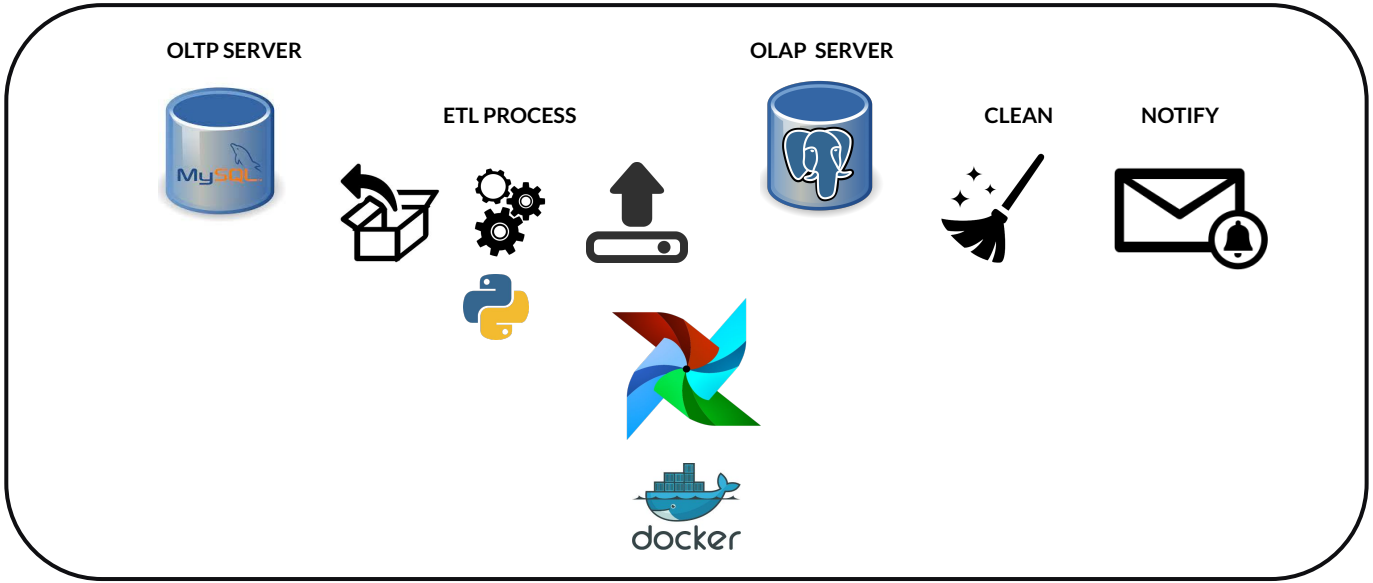
Outros operadores disponíveis...

- MySqlOperator
- PostgresOperator
- MsSqlOperator
- OracleOperator
- JdbcOperator
- DockerOperator

Vamos colocar a mão na massa.

Stack

Data Pipeline 01



Stack

Data Pipeline 01

Requisitos do Pipeline.

- Necessidade de separar os dados do ambiente OLTP para o ambiente OLAP.
- Todas as tarefas devem ser executadas diariamente.
- Todo o processo deve ser notificado.
- O ambiente de stage deve ser limpo.
- Todos os logs devem ser mantidos.

Stack

Montando o Ambiente OLTP

MySQL (OLTP)

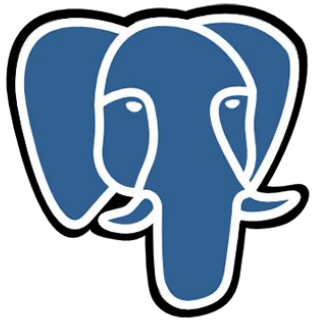
- Criar o container.
- Mapear volumes.
- Instalar e configurar os softwares (apt-get, vim)
- Carregar os dados.
- Realizar Queries.

Stack

Montando o Ambiente OLAP

Postgresql (OLAP)

- Criar o container.
- Mapear volumes.
- Instalar e configurar os softwares (apt-get, vim)
- Criar o banco de dados.



Stack

PostgreSQL

Configurando o Airflow para enviar E-mails

Servidor SMTP

- Criar uma conta no Mailtrap.
- Configurar o arquivo de configurações do Airflow.

Stack

Agendamento no Airflow

Agendamento em intervalos

- scheduler_interval= None - significa que a dag não terá um agendamento.
- Podemos definir execuções uma vez em uma hora, em um dia, um mês com o parâmetro **scheduler_interval.**

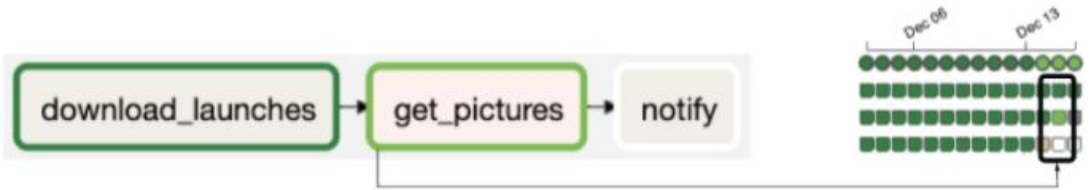
Stack

Agendamento no Airflow

Agendamento em intervalos

- scheduler_interval= None - significa que a dag não terá um agendamento.
- Podemos definir execuções uma vez em uma hora, em um dia, um mês com o parâmetro **scheduler_interval.**

Figure 2.14 Relationship between graph view and tree view

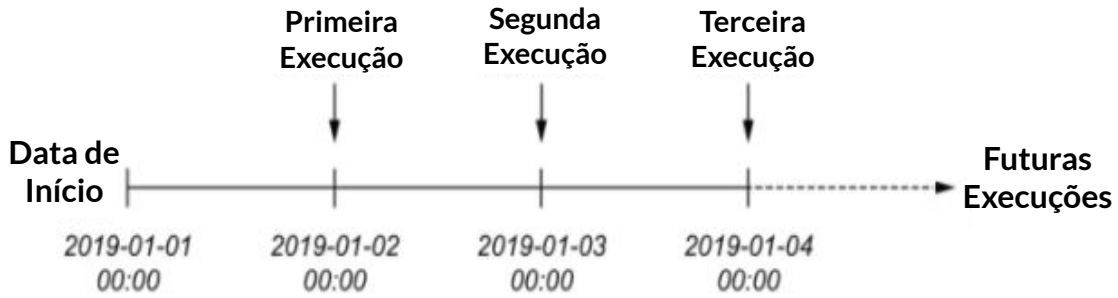


Stack

Agendamento no Airflow

Esquema de execução no agendamento diário do Airflow.

- **"@daily"** - Especifica a execução uma vez no dia.
- end_date - Especifica a data fim para a execução das dags.



Stack

Agendamento no Airflow

Exemplo.

```
dag = DAG (
    dag_id="teste_date",
    schedule_interval="@daily",
    start_date=dt.datetime(year=2019, month=1, day=1),
    end_date=dt.datetime(year=2019, month=1, day=5),
)
```

Stack

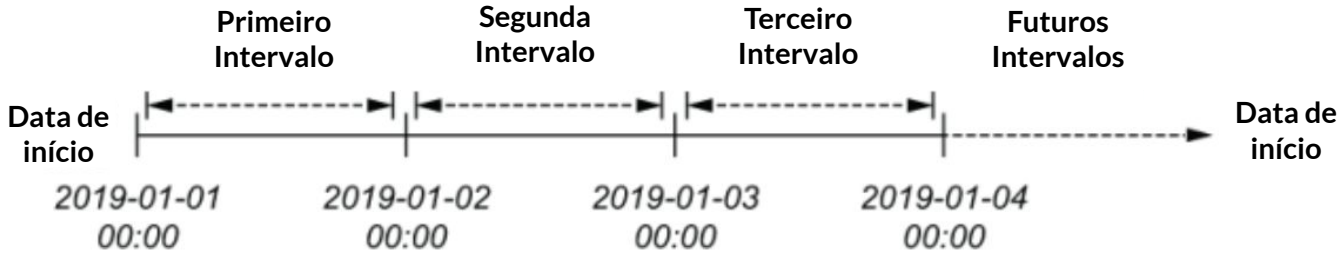
Dag Schedule Daily

```python
from datetime import datetime

from airflow import DAG
from airflow.operators.bash import BashOperator

dag = DAG(
    dag_id="scheduler-daily",
    schedule_interval="@daily",
    start_date=datetime(2021, 7, 25),
)

print_date_task = BashOperator(
    task_id="print_date",
    bash_command=(
        "echo Iniciando a tarefa: &&"
        "date"
    ),
    dag=dag,
)

processing_task = BashOperator(
    task_id="processing_data",
    bash_command=(
        "echo Processando os dados... &&"
        "sleep 50"
    ),
    dag=dag,
)

print_date_task >> processing_task
```

Stack

Execution Date

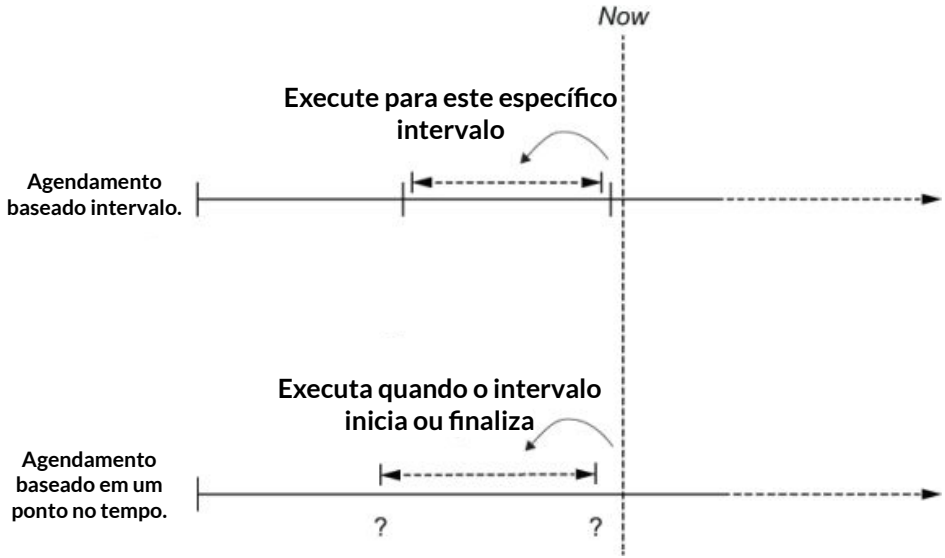
Entendo a execução baseada em intervalos.



Stack

Execution Date

Entendo a execução baseada em intervalos.



Stack

Cron-based Scheduler

Agendamento em intervalos de tempo mais complexos.

- Que tal todos os dias às 23:30 ? Todos os sábados as 23:45?

```
# ┌──────── minute (0 - 59)
# │ ┌────── hour (0 - 23)
# │ │ ┌──── day of the month (1 - 31)
# │ │ │ ┌── month (1 - 12)
# │ │ │ │ ┌ day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │  7 is also Sunday on some systems)
# * * * * *
```

Stack

Cron-based Scheduler

Alguns exemplos de expressões cron.

- **0 * * * *** = hourly (executando em uma hora)
- **0 0 * * *** = daily (executando todos os dias à meia noite)
- **0 0 * * 0** = weekly (executando a meia noite do domingo)

Um pouco mais complicado.

- **0 0 1 * *** = meia noite no primeiro dia de cada mês.
- **45 23 * * SAT** = às 23:45 de cada sábado.
- **0 0 * * MON-FRI** = executa de segunda à sexta à meia noite.
- **0 0,12 * * *** = executa todos os dias a meia noite e ao meio dia.

Stack

Macros para Agendamento

Algumas macros fornecidas pelo Airflow.

Macro	Significado.
`@once`	Executa uma vez e apenas uma.
`@hourly`	Executa uma vez a cada hora e no início da hora.
`@daily`	Executa uma vez no dia à meia noite.
`@weekly`	Executa uma vez por semana à meia noite.
`@monthly`	Executa uma vez por mês à meia noite no primeiro dia do mês.
`@yearly`	Executa uma vez por ano à meia noite iniciando em primeiro de Janeiro.

Cron-based Scheduler

Crontab Guru
https://crontab.guru/

Frequency-based Scheduler

Através do agendamento baseado no cron é difícil definir intervalos baseados em frequência.

Como agendar a execução da Dag uma vez a cada três dias?
Podemos utilizar o objeto timedelta.

```
dag = DAG(
    dag_id="teste",
    schedule_interval=dt.timedelta(days=3),
    start_date=dt.datetime(year=2019, month=1, day=1),
    end_date=dt.datetime(year=2019, month=1, day=5),
)
```

Para executar a cada 10 minutos
timedelta(minutes=10)

A cada duas horas.
timedelta(hours=2)

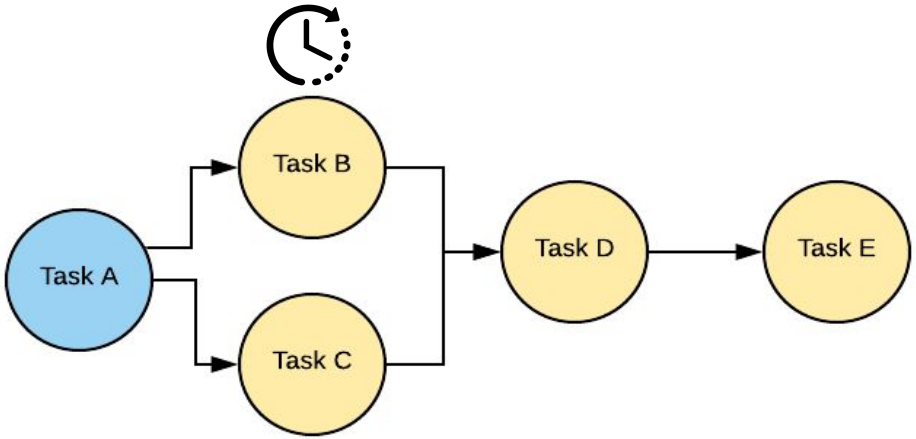
Stack

Frequency-based Scheduler

Exemplo

Re-execução de Tarefas

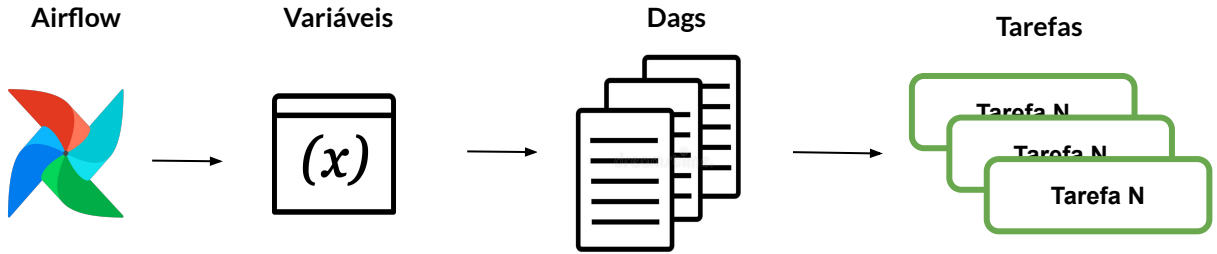
- Existem fatores externos que influenciam na performance da tarefas.
- Continuar o Pipeline de onde parou.



Stack

Variáveis, Templates e Macros

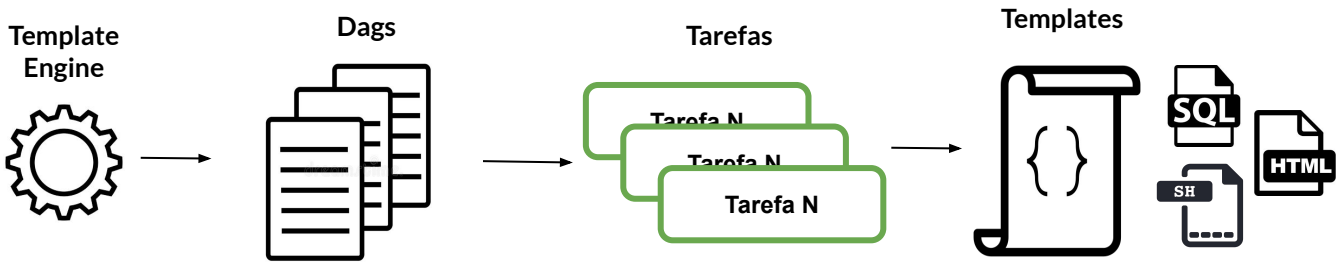
- Atribuição de valores e parâmetros comuns para várias dags e tarefas.
- Atualização e manutenção centralizada.
- Dados independente do código.
- Melhor legibilidade e organização.



Stack

Variáveis, Templates e Macros

- Atribuição de valores e parâmetros comuns para várias dags e tarefas.
- Atualização e manutenção centralizada.
- Dados independente do código.
- Melhor legibilidade e organização.



Stack

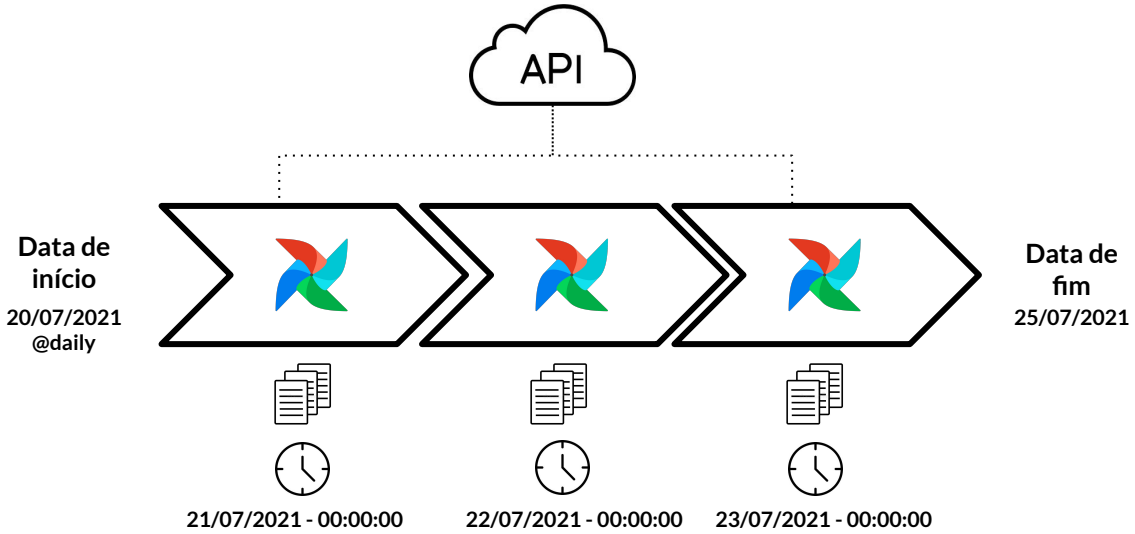
Variáveis, Templates e Macros

- **Macros padrão definidas pelo Airflow e que são utilizáveis em todo template.**
- **Documentação:**
  https://airflow.apache.org/docs/apache-airflow/stable/macros-ref.html

{{ ds }}	A data de execução da dag no formato YYYY-MM-DD.
{{ ti }}	equivale a macro {{ task_instance }}
{{ params }}	Uma referência para parâmetros definidos pelo usuário.
{{ var.value.my_var }}	Acesso a variáveis globais definidas e representadas como um dicionário.

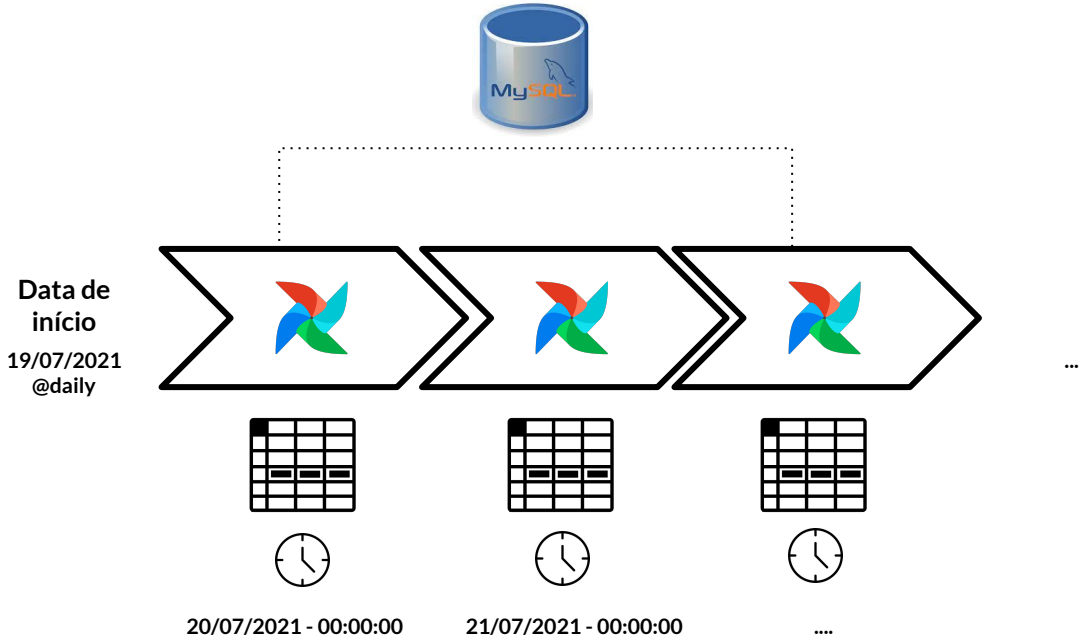
Stack

Exemplo Coleta de Dados de API do Mercado Financeiro



Stack

Coletando e Carregando Dados Incrementais



Stack

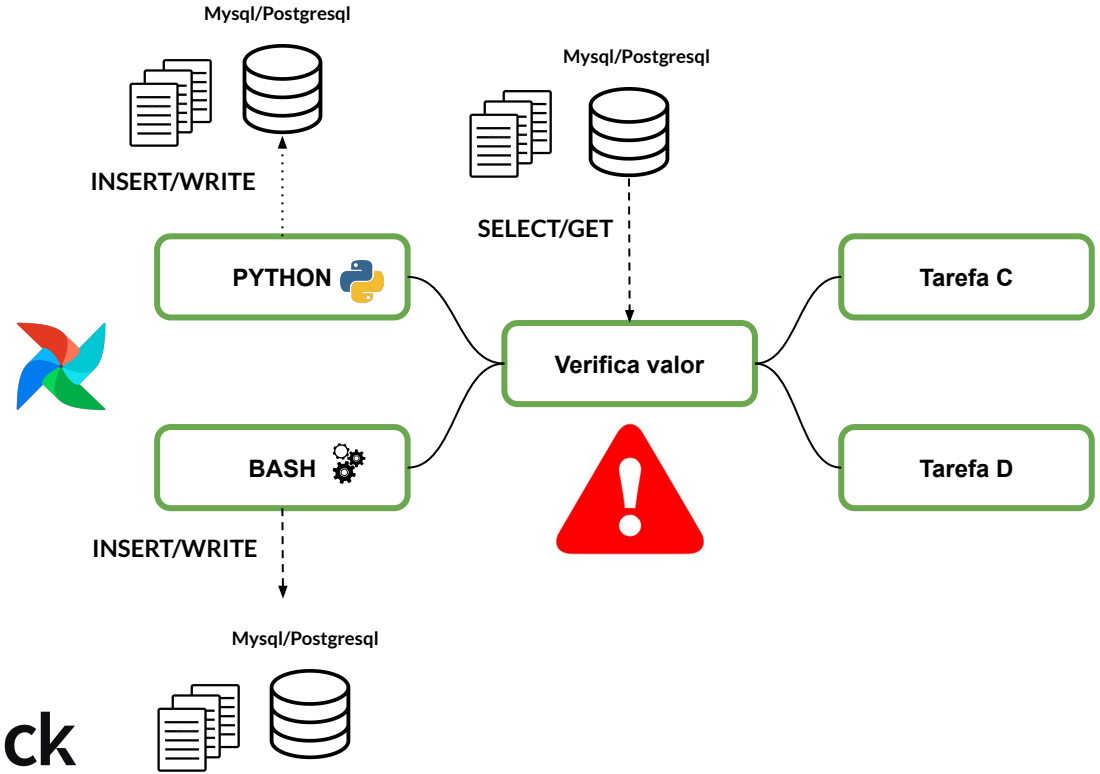
Comunicação entre Tarefas

- Como permitir a comunicação entre tarefas?
- Como compartilhar dados entre cada tarefa?
- Utilizar arquivos ou bancos de dados como armazenamento temporário?
    - Cada tarefa/operador implementar sua persistência?
    - Qual será o formato de armazenamento e recuperação dos dados?

Stack

Comunicação entre Tarefas

- Análise de cenário.



Stack

XComs

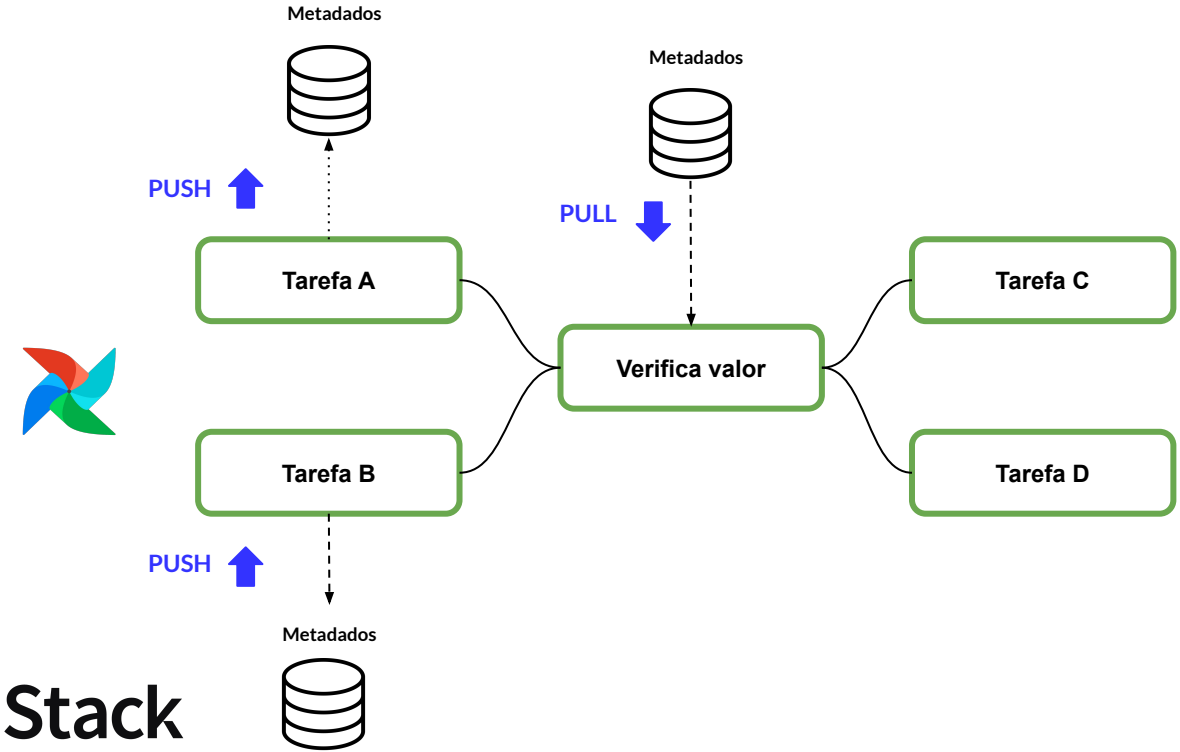
Cross Communications

- Permite fácil comunicação entre as tarefas.
- Métodos de armazenamento e recuperação dos dados padronizados.
- Formato nativo do Airflow.
- Armazenamento de metadados e logs importantes.

Stack

XComs

Cenário de comunicação entre tarefas utilizando XComs.



Stack

XComs

Atributo	Descrição
Key	O identificador do registro. É utilizado para recuperar o valor associado.
Value	O valor que será compartilhado e que será associado a key.
Timestamp	A data e horário que o xcom foi criada.
Execution Date	A data de execução que a DAG gerou a xcom.
Task ID	O ID da tarefa onde a xcom foi criada.
Dag ID	O ID da dag onde a xcom foi criada.

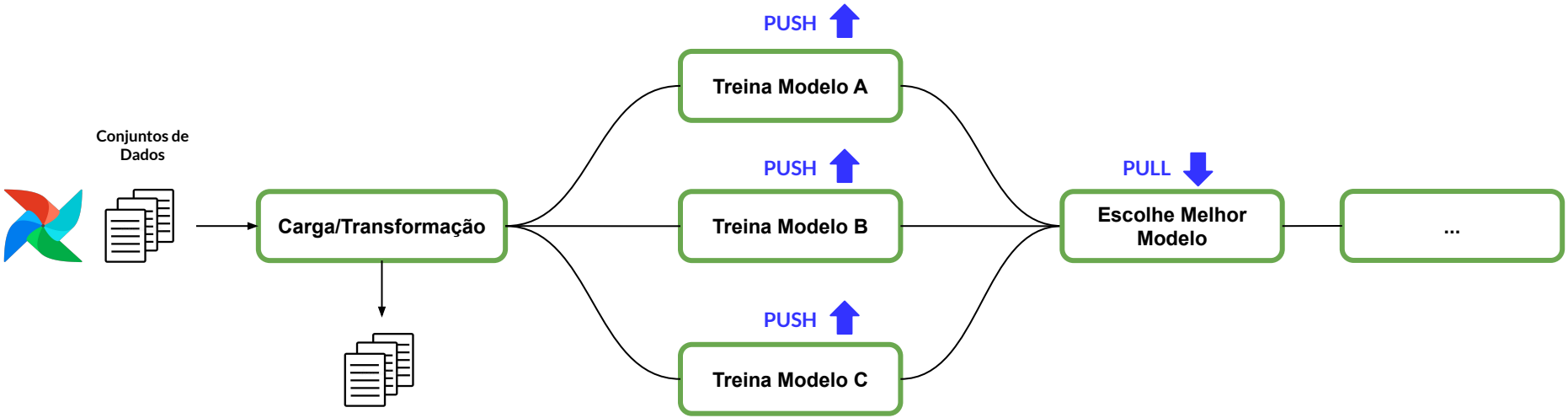
Cross Communications

- Atributos no modelo de dados.
- **Limitação**
  - SQLite: 2 GB
  - Postgres: 1 GB
  - MySQL: 64 KB

Stack

XComs Prática

Cenário de comunicação entre tarefas utilizando XCom através de um Pipeline para carregar e transformar os dados para testar três diferentes modelos de Machine Learning.



Stack

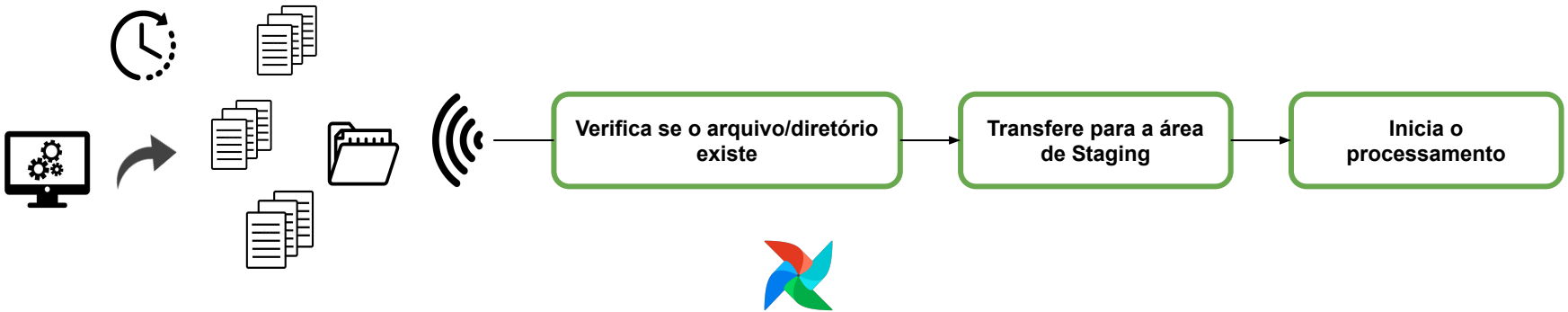
Sensor Operators

Tipo especial de operadores que nos permitem esperar para um evento acontecer ou uma condição ser satisfeita.

- Aguardar um arquivo estar disponível em um diretório/volume.
- Aguardar dados serem retornados a partir de uma consulta SQL.
- Receber um status de uma requisição http.
- Aguardar uma partição no hive está disponível.
- Aguardar por um arquivo está presente em um bucket S3.

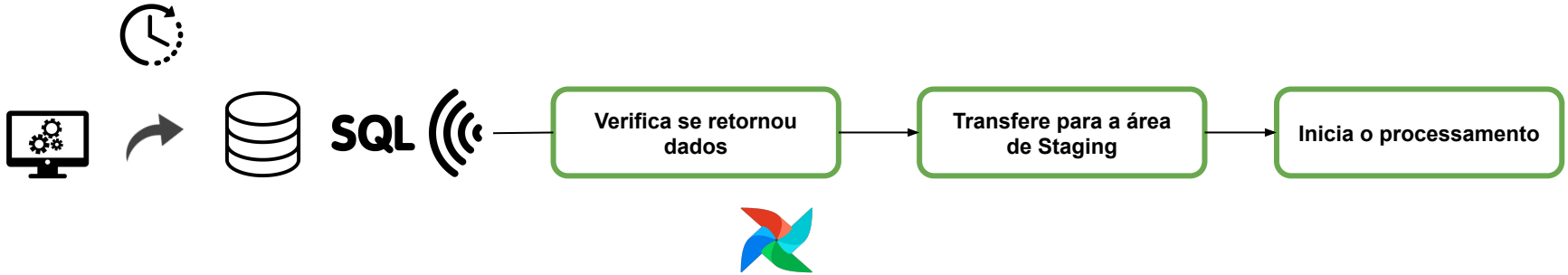
Stack

Sensor Operators



Stack

Sensor Operators

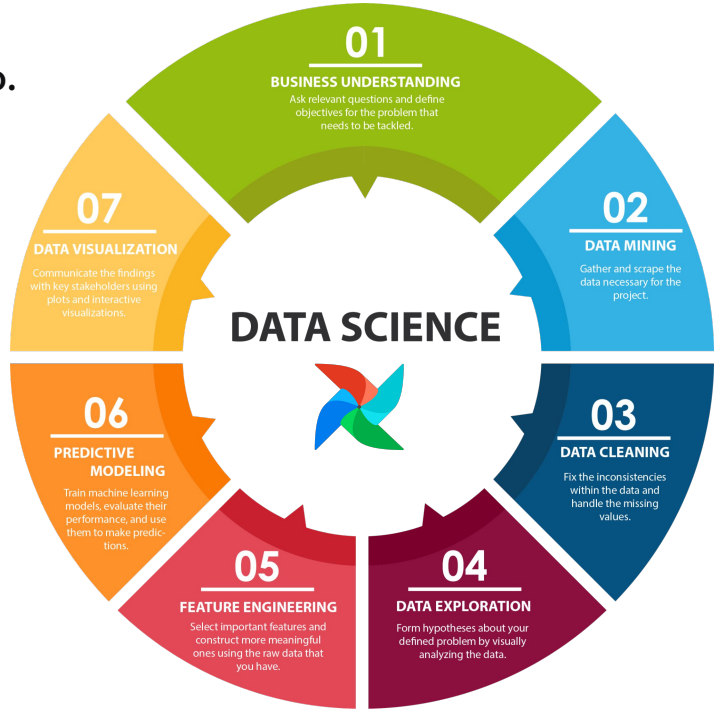


Stack

Data Science Project Automation

Automação das Etapas do Projeto.

- Carga dos dados.
- Limpeza.
- Análise e Exploração.
- Pré-processamento e transformação.
- Feature Engineering.
- Modelagem.
- Ajustes e Avaliação.
- Deploy.



Stack

Ferramenta para **parametrização e execução** de Jupyter Notebooks.
https://github.com/nteract/papermill

Stack

Algumas alternativas

Vantagens

- Alternativa mais rápida e simples.
- Ideal para etapas de prototipação.
- Desenvolvimento simples.
- Sem aumentar a curva de aprendizado.

Desvantagens

- Difícil manutenção e controle de erros.
- Todo o processo é atômico. Difícil dividir as operações.
- Todo o processamento é feito no nível do Kernel.

Data Science Project Automation

Apache Airflow

Stack

Algumas alternativas

Vantagens

- Integração com outros componentes.
- Tolerância a falhas.
- Alertas personalizados.
- Operações independentes.
  - Redução do tempo de processamento e custos.
- Melhor versionamento e log.
- Fácil manutenção e controle de erros.
- Melhor controle de fluxos.
- Fácil integração entre equipes de Engenharia e Ciência de Dados.

Desvantagens

- Maior curva de aprendizado.
- Maior tempo de desenvolvimento.

Stack