# Relatório LAMIA 18
# Prática: Pipelines de Dados II - Airflow (II)

**Kaique Medeiros Lima**

## 1 Introdução

Neste relatório, será discutido as outras aplicações do Apache Airflow, como o uso de executores locais, executores Celery e executores Kubernetes. Além disso, será abordado o uso de variáveis, macros e templates, compartilhamento de dados entre tarefas, controle de DAGs e dependências entre DAGs. Por fim, será discutido o sistema de login do Airflow, o uso do StatsD e a stack TIG.

## 2 Descrição da atividade

### 2.1 O que é SQLite?

O SQLite é um banco de dados relacional embarcado no programa, o que significa que não é baseado em servidor. Ele é compatível com as propriedades ACID e é implementado pelo padrão SQL. Sua configuração é mínima, permitindo que seja facilmente executado sem grandes ajustes. Embora suporte um número ilimitado de leitores, apenas um processo pode escrever por vez. O banco de dados tem um tamanho máximo limitado de 140 TB e é armazenado inteiramente dentro de um único arquivo, tornando-o bastante simples de gerenciar.

### 2.2 Como a tarefa é executada?

1. Quando uma **DAG** é agendada, cada tarefa tem seu registro com status (Na fila, Agendada, Running, Sucesso, Falha) armazenado dentro do banco de dados de metadados.

2. O agendador lê periodicamente do banco de dados de metadados para checar se há alguma tarefa para rodar.

3. O executor pega as tarefas da fila interna e especifica como executá-las (Sequencial, Local, Celery e K8s).

#### 2.2.1 O que é o Executor Sequencial?

O Executor Sequencial é o mais básico, permitindo que apenas uma tarefa seja executada de cada vez. Ele é o único executor compatível com o SQLite e é a configuração padrão do Airflow. No entanto, vale ressaltar que não deve ser utilizado em ambientes de produção, sendo recomendado apenas para depuração no contexto do Airflow.

## 2.3 Executor Local com PostgreSQL

### 2.3.1 O que é o Executor Local?

O Executor Local permite rodar múltiplas tarefas em paralelo, utilizando o módulo Python `multiprocessing` para criar processos que executam as tarefas. Ele é fácil de configurar e eficiente no uso de recursos, sendo uma opção leve para ambientes que exigem escalonamento vertical. No entanto, como apresenta um único ponto de falha, sua implementação deve ser cuidadosamente planejada.

### 2.3.2 O que é PostgreSQL?

O PostgreSQL é um banco de dados relacional compatível com ACID, operando no modelo cliente-servidor. Ele lida com conexões simultâneas e implementa o padrão SQL, oferecendo recursos avançados para gestão de dados. Além disso, o PostgreSQL é altamente escalável e extensível, o que o torna uma escolha robusta para aplicações que exigem grande capacidade de processamento e flexibilidade.

## 2.4 Estratégias do Executor Local

### 2.4.1 Paralelismo

Caso o paralelismo seja equivalente a 0, é ilimitado. Caso seja maior que 0, é limitado. Como uma boa prática, é melhor configurar o paralelismo para:

$$\text{number\_of\_cores} - 1$$

## 2.5 Escalonamento do Apache Airflow com executores Celery

### 2.5.1 O que é o Executor Celery?

O Executor Celery é responsável por escalonar o Apache Airflow, sendo reforçado pelo Celery, uma fila de tarefas assíncronas distribuídas. Ele distribui as tarefas entre os nodos em execução e utiliza o `airflow worker` para criar workers que executam as tarefas. Com escalonamento horizontal, o Celery oferece alta disponibilidade, permitindo que o Airflow continue agendando tarefas mesmo se um worker parar de funcionar. Para seu funcionamento, é necessário um intermediário de mensagens, como RabbitMQ ou Redis, e permite organizar diferentes filas de tarefas.

### 2.5.2 Pontos negativos

- Precisa de um componente extra, sendo este o intermediário de mensagens.
- Exige tempo e trabalho para organizar o sistema.
- Manutenção dos workers.

## 2.6 Relembrando sobre Kubernetes

O Kubernetes é uma plataforma open source poderosa, fundada pela Google em 2014, que atua como controlador de containers. Ele automatiza diversas tarefas de administração de sistemas, como atualização de servidores, instalação de patches de segurança, configuração

de redes, backups, monitoramento e failover. O Kubernetes também permite o deploy instantâneo de atualizações e oferece métodos eficientes para implementar práticas de Continuous Delivery (CD), como os modelos Canary e Blue-green. Além disso, suporta auto-escalonamento e já vem com sistemas de redundância e failover implementados, tornando as aplicações mais confiáveis e resilientes.

### 2.6.1 Pontos negativos

- A configuração correta do sistema Kubernetes é extremamente complexa.
- Difícil de manter.
- A curva de aprendizado é íngreme.

### 2.6.2 Objetos importantes do Kubernetes

**Pod:** Grupo de um ou mais containers com armazenamento/rede comp

## 2.7 Escalando Airflow com Executores Kubernetes

### 2.7.1 Pontos negativos do Celery Executor

- Precisa de tempo para configurar aplicações como RabbitMQ, Celery e Flower.
- Lida com dependências faltantes.
- Perda de recursos: os workers do Airflow ficam ociosos enquanto não possuem trabalho.
- Nodos dos workers não são tão resilientes.
- Apesar disso, ainda é amplamente utilizado em produção.

### 2.7.2 O que é o Executor Kubernetes?

O Executor Kubernetes executa as tasks no Kubernetes, onde cada task é representada por um Pod. A configuração das tasks é feita diretamente no Pod, e o cluster pode ser expandido ou encolhido conforme a carga de trabalho. Os Pods são interrompidos apenas quando completam sua execução, e o scheduler do Kubernetes sobrescreve o fluxo de eventos do sistema para gerenciar o processo de execução das tasks.

**Distribuição DAG:** O processo de distribuição da DAG no Kubernetes começa com um Git clone, utilizando o *init-container* para cada Pod. Em seguida, é montado um volume contendo as DAGs e, por fim, a imagem é construída com os códigos das DAGs para que possam ser executadas no cluster.

## 2.8 Evitar implementar o código diretamente com Variáveis, Macros e Templates

### 2.8.1 Casos de uso

```
SELECT * FROM my_table WHERE dt = "2019-01-01"
```

ou

```
SELECT * FROM my_table WHERE dt="{{ execution_date }}"
```

Idempotência.

### 2.8.2 Variáveis

As variáveis são armazenadas no banco de dados de metadados e podem ser definidas tanto pela CLI quanto pela UI. Elas possuem três colunas principais: chave, valor e um campo que indica se estão criptografadas. Além disso, as variáveis podem ser armazenadas no formato JSON.

### 2.8.3 Templates

- Permite substituir os valores dos *placeholders* a qualquer momento.
- Baseado no Jinja Templating.
- *Placeholders:* {{}}.
- Pode ser usado com parâmetros de template.

### 2.8.4 Macros

- {{ ds }}: Data de execução da DAG no formato `YYYY-MM-DD`.
- {{ prev_ds }}: Data de execução anterior da DAG no formato `YYYY-MM-DD`.
- {{ next_ds }}: Próxima data de execução da DAG no formato `YYYY-MM-DD`.
- {{ dag }}: O objeto atual da DAG.
- {{ params }}: Dados acessíveis pelos operadores.
- {{ var.value.chave_da_variavel }}: Acesso às variáveis armazenadas nos metadados.

## 2.9 Como compartilhar dados entre suas tasks com XCOM

### 2.9.1 Definição

Possibilita múltiplas tasks trocarem mensagens entre si.

KEY: "my_key"  VALUE: "my_value"  TIMESTAMP: 2019-01-01

- task_id.
- dag_id.

Como boa prática, mantenha os valores leves. Caso contrário, utilize um sistema externo.

## 2.10 TriggerDagRunOperator: Quando sua DAG controla outra DAG

Ele ativa um diagrama de um DAG ID especificado quando certa condição é atingida.

### 2.10.1 Como funciona?

Parâmetros do `TriggerDagRunOperator`:

- `trigger_dag_id`
- `python_callable`
- `params`

### 2.10.2 Pontos importantes

- O controlador não espera pelo alvo finalizar.
- O controlador e o alvo são independentes.
- Sem visualização do controlador nem de seu histórico.
- Ambas as DAGs devem estar agendadas.
- Intervalo de agendamento da DAG alvo deve ser `None`.

## 2.11 Dependências entre DAGs com o ExternalTaskSensor

O `ExternalTaskSensor` espera pela task externa (em outra DAG) terminar para rodar.

### 2.11.1 Pontos importantes

- Mantém a mesma agenda entre as DAGs.
- Pode usar os parâmetros `execution_delta` ou `execution_date_fn`.
- O `ExternalTaskSensor` é diferente do `TriggerDagRunOperator`.
  - Usar ambos normalmente leva ao sensor travar.
- Modo "cutucar" usado por padrão; pode usar "reagendar" se necessário.

## 2.12 Observações sobre o AWS EKS

**AWS EKS** (Elastic Kubernetes Service) é o Serviço Elástico Kubernetes da Amazon. Ele facilita o deploy, manutenção e escalonamento de aplicações em containers.

### 2.12.1 Características principais

- Orquestra seus containers.
- Ainda precisa configurar subredes, VPS, cargos IAM, chaves SSH, etc.
- Integrado com serviços AWS (ECR, ELB, IAM, etc.).

### 2.12.2 Como acessar externamente

**ClusterIP:**

- Impossibilita mudanças enquanto está rodando.
- Permite que aplicativos conversem internamente, mas não oferece acesso externo.
- Não recomendado.

**NodePort:**

- Abre portas específicas em nodos mais velhos.
- Não recomendado para produção.

**Load Balancer:**

- Oferece um IP público para acessar serviços.
- Serviços expostos ao Load Balancer recebem um IP e são cobrados.

**Ingress:**

- Não é um serviço, mas um ponto de entrada.
- Trafega de forma eficiente para os serviços internos.
- Melhor e mais eficiente, mas difícil de implementar.

## 2.13 Como o sistema de login funciona no Airflow

### 2.13.1 O básico

- Baseado no módulo de logging.
- Logs escritos em arquivos.
- Relatórios por níveis: `INFO`, `ERROR`, `DEBUG`, `WARNING`, `CRITICAL`.
- Usam `formatters`.
- `Handlers` para saídas:
  - `FileHandler`
  - `StreamHandler`
  - `NullHandler`

### 2.13.2 Como o logger é estabelecido

1. `getLogger()`
2. `FileHandler`
3. `Formatter(settings.SIMPLE_LOG_FORMAT)`
4. `setFormatter`
5. `addHandler`
6. `setLevel(settings.LOGGING_LEVEL)`

```
def setup_logging(filename):
    # Cria o relatório do file handler para o processo daemon
    root = logging.getLogger()
    handler = logging.FileHandler(filename)
    formatter = logging.Formatter(settings.SIMPLE_LOG_FORMAT)
    handler.setFormatter(formatter)
    root.addHandler(handler)
    root.setLevel(settings.LOGGING_LEVEL)

    return handler.system
```

### 2.13.3 Como o sistema funciona?

O objeto `logger` é criado e inicializado. Quando um componente deseja gerar um log, ele chama o objeto `logger` com o método correspondente ao nível daquele log.

## 2.14 ELK Stack

### 2.14.1 O que é o Elasticsearch?

O Elasticsearch é uma ferramenta de busca e análise de dados. Ele permite indexar, armazenar e consultar grandes volumes de dados de forma rápida e eficiente. É amplamente utilizado para buscas full-text, monitoramento em tempo real, análise de logs e visualização de dados.

### 2.14.2 Kibana

O Kibana é uma ferramenta open source poderosa e fácil de usar para criar dashboards e visualizar dados armazenados no Elasticsearch.

### 2.14.3 LogStash

O LogStash lida com o processamento do pipeline de dados no servidor. Ele consegue lidar com múltiplas fontes de dados simultaneamente, permitindo o envio de vários dados processados ao mesmo tempo para diversos sistemas.

### 2.14.4 Filebeat

O Filebeat é uma ferramenta utilizada para envio e encaminhamento de logs e dados para o Elasticsearch ou LogStash.

## 2.15 Introdução ao StatsD

- O Airflow envia métricas ao StatsD.
- Utiliza o Daemon para agregar e resumir as métricas da aplicação.
- É extremamente rápido (baseado em UDP) e consome poucos recursos.
- Encaminha as métricas para outras aplicações.

### 2.15.1 Métricas

Todas as métricas são baseadas em três tipos principais:

- **Counters:** Contadores para eventos.
- **Gauges:** Indicadores de valores em determinado momento.
- **Timers:** Métricas temporais para monitorar a duração de eventos.

### 2.15.2 Stack TIG

### 2.15.3 Telegraf:

- Agente para coleta, processamento e agregação de métricas.

### 2.15.4 InfluxDB:

- Banco de dados especializado em séries temporais (Time Series).

### 2.15.5 Grafana:

- Ferramenta para visualização de dados e monitoramento de aplicações.

### 2.15.6 Pontos importantes

- O pacote `StatsD` deve ser instalado.
- Métricas podem ser filtradas utilizando o `statsd_allow_list`.

# 3 Conclusão

Esse card trabalhou com as configurações do Apache Airflow, o qual não houve com alterações diretamente nas DAGs, mas sim em dockerfiles e airflow.cfg. Além disso, foi abordado o uso do SQLite no Airflow.