Executor Celery

Quando utilizamos o executor Local, ficamos limitados aos recursos de uma única máquina para executar nossas tarefas, o que não é indicado em casos que queremos colocar nosso DAG em produção, justamente porque, em caso de falha na máquina, toda a execução das tarefas também falhará. No entanto, o Airflow nos dá uma alternativa para lidar com este problema: o executor Celery.

Este executor nos permite trabalhar com fila de tarefas, mecanismo que admite a distribuição dessas tarefas em processos distintos, sendo cada um executado por diferentes workers ("trabalhadores" em português). Os workers funcionam, basicamente, como máquinas individuais que aguardam as tarefas chegarem na fila para executá-las.

O Celery nos permite criar workers em máquinas diferentes, gerando o processamento distribuído de tarefas, o que resolve o problema de depender unicamente de um ponto de falha. Sendo assim, caso uma máquina venha a falhar, somente um worker também falhará, e as tarefas executadas por ele serão enviadas para um worker que permanece em funcionamento, impedindo uma quebra no fluxo de execução dessas tarefas.

Agora que entendemos como o executor Celery distribui a execução das tarefas, entenderemos como ele se integra aos demais componentes do Airflow.

Após configurarmos o executor, o processamento se fecha em uma espécie de ciclo. Este ciclo inicia com o Scheduler, responsável por enviar as tarefas agendadas para a fila. O Worker aguarda a chegada dessas tarefas na fila para que possa executá-las e, ao final da execução, reporta ao banco de dados. O banco de dados, por sua vez, registra a tarefa e seus status. Por fim, o Scheduler lê esse banco de dados e atualiza o status da tarefa no dag. Esse processo é o mesmo independentemente da quantidade de workers.

Vale ressaltar, no entanto, que para o funcionamento desse processo, o Celery precisa de um broker ("mediador" em português), responsável por armazenar as tarefas que estão sendo executadas. Este mediador é utilizado, basicamente, para guardar a fila de tarefas. Na documentação do Celery, há duas indicações de banco de dados não relacionais que podem ser adotados para essa função. Aqui, utilizaremos o Redis.

As principais vantagens do CeleryExecutor é que ele nos oferece paralelismo (execução de várias tarefas ao mesmo tempo), alta disponibilidade (capacidade de utilizar várias máquinas para que não fiquemos à mercê de um único ponto de falha), e processamento distribuído das nossas tarefas. As desvantagens ficam por conta da configuração mais trabalhosa e da manutenção do trabalhador, porque, havendo mais de um worker, faz-se necessário configurar todas as máquinas. Por sua performance, este executor é utilizado principalmente em cargas de trabalho mais pesadas e na colocação de DAGs em produção.

Configuração

Sabemos que para funcionar, o executor Celery precisa de um mediador que será responsável por armazenar a nossa fila de tarefas. Então, agora faremos a instalação deste mediador, que será o banco de dados Redis.

Primeiramente, baixe o arquivo presente neste link e salve-o na pasta "Documents". Trata-se de um arquivo comprimido, então vamos descompactá-lo!

Abra o terminal e navegue até a pasta "Documents". Para isso, você tem duas opções: abrir manualmente o terminal e executar o comando cd Documents ou, na pasta "Documents", clicar com o lado direito do mouse e selecionar "Open in Terminal".

Em seguida, vamos executar o comando tar -xf, seguido do nome do arquivo a ser descompactado, redis-7.0.4.tar.gz.

```
cd Documents
tar -xf redis-7.0.4.tar.gz
```

Para conferir se o arquivo foi descompactado, vá até a pasta "Documents" e verifique se há uma pasta descomprimida do arquivo em questão. De volta ao terminal, acesse-a com o comando cd redis-7.0.4/

```
cd redis-7.0.4/
```

O Redis possui muitos módulos que precisam ser instalados, para isso utilizaremos o comando make.

```
make
```

Execute-o e aguarde. Esta etapa pode levar um tempo!

Após a execução do comando anterior, finalizaremos a instalação com o comando a seguir:

```
sudo make install
```

Ao executá-lo, sua senha será requerida, digite-a e tecle "Enter". Verifique se a instalação foi concluída com sucesso executando o comando redis-server. Observe que o log nos retorna uma série de informações, entre elas a porta (que deve aparecer como "Port") onde o Redis está rodando. Salve esta informação, pois ela será necessária.

```
redis-server
```

Pronto! Finalizamos a instalação do Redis. Agora voltaremos ao arquivo de configuração do Airflow para definir o executor como Celery, conectar o Airflow ao Redis e alterar algumas outras configurações.

Abra o arquivo de configuração do Airflow ("airflow.cfg" em "airflow") no VS Code e tecle "Ctrl + F" para pesquisar por "executor". Nas ocorrências do resultado, busque pela variável "executor", que está definida como "LocalExecutor", e altere para "CeleryExecutor". Tecle "Ctrl + S" para salvar.

```
executor = CeleryExecutor
```

```
Em seguida, busque por "result_backend". Perceba que se trata de uma
variável que recebe uma url correspondente ao endereço do banco de dados
responsável por armazenar o status das tarefas, neste caso, o Postgres. Por
ser semelhante ao que passamos para a variável "sql_alchemy", deixaremos o
trecho inicial da url, "db+postgresql:", e altareramos o restante pelo nome
do usuário ("airflow_user"), senha ("airflow_pass") e nome do banco
("airflow_db") que criamos no Postgres. Ficará assim:
```

```
result_backend =
db+postgresql://airflow_user:airflow_pass@localhost/airflow_db
```

Tecle "Ctrl + S" para salvar essa alteração.

Busque, agora, pelo parâmetro "broker_url". Passaremos para ele o endereço do Redis, ou seja, onde está sendo executado. Para isso, colocaremos o localhost, uma vez que está rodando localmente na nossa máquina, e a porta informada anteriormente no log. Ficará assim:

```
broker_url = redis://0.0.0.0:6379/0
```

Salve o arquivo.

A partir de agora, para utilizar o Airflow, precisaremos de um terminal executando o banco de dados Redis a fim de que o Airflow consiga se conectar a ele em tempo de execução.

Broker

O Redis é um banco de dados não relacional do tipo chave-valor, que é uma forma de armazenarmos um valor vinculado a uma chave. Ele é um sistema voltado para armazenamento e processamento mais dinâmico e ágil. Além disso, esse banco de dados pode ser utilizado em diferentes casos de uso como armazenamento de cache, streaming de mídia, dentre outros.

O Celery utiliza o Redis como mediador (broker), pois ele é responsável por "mediar" as tarefas que são enviadas pelo scheduler até o worker. Sendo assim, ele recebe as tarefas do Scheduler, armazena elas na fila de tarefas e entrega essas tarefas para um worker que será responsável por executá-las

Para entender mais sobre o funcionamento do Celery por baixo dos panos, acesse a documentação:

- Documentação do Celery
- Celery Executor

Celery Flower

No terminal da máquina, vamos executar o comando redis-server. Isso é necessário para que, ao rodar o Airflow, ele consiga se conectar ao nosso mediador, o banco de dados Redis.

```
redis-server
```

Agora, abra outra aba do terminal, clicando no símbolo de + envolvido por uma janela, no canto superior esquerdo deste terminal.

Não usaremos o comando airflow standalone para executar o Airflow, porque este só é utilizado em situações que empregamos o executor sequencial ou local. Se o executarmos, as configurações assumirão a formatação anterior, correspondente a estes executores. Sendo assim, precisaremos executar cada um dos componentes separados do Airflow: um terminal será para o Airflow Scheduler e outro para o Airflow web server.

Acesse a pasta "airflow", ative o ambiente e importe a variável de ambiente Airflow Home. Os comandos destas etapas estão listados, respectivamente, a seguir:

Precisaremos realizar essas etapas sempre que abrirmos uma nova aba do terminal.

```
source venv/bin/activate
```

```
export AIRFLOW_HOME=~Documents/airflow
```

Em seguida, execute o comando airflow scheduler para iniciar o Scheduler do Airflow.

```
airflow scheduler
```

Abra uma terceira aba do terminal e perceba que já se encontra em "airflow", então podemos partir para a ativação do ambiente e importação da variável Airflow Home:

```
source venv/bin/activate
```

```
export AIRFLOW_HOME=~Documents/airflow
```

Em seguida, execute airflow webserver e aguarde a inicialização.

```
airflow webserver
```

Observe que, no log de execução do webserver, há mensagens em vermelho que configuram um erro proveniente da versão do Airflow que estamos usando. O Airflow está em constante evolução e, por isso,

podem surgir alguns bugs. Mas não se preocupe, eles não irão interferir no andamento do nosso projeto. Uma das atividades disponibilizadas neste curso, explica os motivos que podem levar ao aparecimento deste erro, além de como evitá-lo.

Após a execução do webserver, podemos acessar o Airflow no navegador através de "localhost8080". Na interface, busque pelo DAG "get_stocks_dag", abra-o e exclua o histórico de execução clicando no símbolo da lixeira, no canto superior direito. Em seguida, retorne à listagem da interface do Airflow e atualize até que este DAG volte a aparecer.

Quando "get_stocks_dag" reaparecer, abra-o e ative-o. Lembre-se de ativar, também, o "Auto-refresh". Quando a execução começar, observe que temos 4 DAG runs (indicados pelas 4 colunas que se assemelham a um gráfico), várias tarefas com status de agendadas (representadas pelos cubos na cor marrom, abaixo das colunas), e duas na fila de execução (caracterizadas pelos dois cubos na cor verde escuro, também abaixo das colunas).

Note, ainda, que as tarefas na fila de execução permanecem no aguardo. Isso ocorre porque ainda não iniciamos um worker que pegue essas tarefas e as execute.

Antes de inicializar um worker, veremos como essas informações estão aparecendo na nossa fila de tarefas. O Celery possui uma ferramenta chamada Flower que nos permite visualizar as tarefas que estão na fila, aguardando os workers pegá-las e executá-las. Vamos acessá-la!

Abra uma quarta aba no terminal, ative o ambiente e importe a variável Airflow Home.

```
source venv/bin/activate
```

```
export AIRFLOW_HOME=~Documents/airflow
```

Em seguida, execute o comando airflow celery flower para que possamos acessar a ferramenta em questão.

```
airflow celery flower
```

O log de execução deste comando trará várias informações, entre elas o endereço de acesso "http://0.0.0.0:5555", que deve vir antecedido pelo texto "Visit me at". Copie o link e acesse-o no navegador.

No **docker**, para acessar observe a tag flower do docker-compose.yaml, na tag test do healthcheck há a url de acesso http://localhost:5555/.

Na interface principal do Flower há uma aba chamada "Dashboard" que traz informações sobre os workers que estão rodando e as tarefas que estão sendo executadas por eles. Como não temos nenhum worker em execução, não há informações nesta aba.

A terceira aba, chamada "Broker", diz respeito ao número de tarefas que estão na fila para serem executadas. Observe que temos 2 mensagens de espera - ou seja, duas tarefas - então, para que possamos

vê-las finalmente sendo executadas, acionaremos um worker.

Volte ao terminal e abra uma nova aba. Neste novo terminal, realize o processo de ativar o ambiente e importar a variável.

```
source venv/bin/activate
```

```
export AIRFLOW_HOME=~Documents/airflow
```

Em seguida, execute o comando airflow celery worker para acionar o worker.

```
airflow celery worker
```

Volte à interface do Airflow e observe que os cubos que representavam as tarefas na fila, antes verde escuro, agora indicam que essas tarefas estão em execução, assumindo a cor verde claro. Perceba, também, que as tasks estão sendo executadas duas por vez, respeitando o parâmetro "max_active_tasks_per_dag", definido como "2".

Atualize a página do Flower e veja que as mensagens da aba "Broker" foram zeradas, não há mais nenhuma tarefa aguardando execução porque o worker já está a executá-las. Agora, clique na aba "Dashboard" e observe que consta um worker ativo, com status online, seguido da quantidade de tarefas em execução - número que varia entre 0 e 2.

No VS Code, a pasta "stocks" foi criada em "airflow", bem como suas respectivas subpastas e os arquivos correspondentes a elas. Nossas tarefas estão sendo executadas!

É importante ressaltar que, neste curso, utilizaremos apenas um worker a fim de simplificar nosso trabalho. Mas, para que possamos aproveitar todas as vantagens do executor Celery, é necessário que tenhamos várias máquinas executando diferentes workers, assim atingiríamos a alta disponibilidade e o processamento distribuído.

Até esta etapa, passamos por 3 diferentes executores: o sequencial (executor padrão do Airflow), o local e o Celery, sendo os dois últimos capazes de realizar tarefas em paralelo. Além disso também vimos os parâmetros "max_active_tasks_per_dag" e "max_active_runs_per_dag", responsáveis por controlar a quantidade de tarefas e DAGs, respectivamente, a serem executados simultaneamente. Por último, entendemos o conceito de fila de tarefas, e o significado e funcionalidade dos workers.

Erro no log de execução

Como o Airflow é um projeto open source ele está em constante evolução e bugs podem aparecer de vez em quando, por isso manter a versão do Airflow atualizada com patches é sempre importante (patches é o terceiro número da versão, por exemplo: 2.3.2 - Major.Minor.Patches).

Quando executamos o Airflow utilizando o comando airflow webserver nos deparamos com uma mensagem de erro no log de execução. O motivo para esse erro é que o serviço do webserver roda em 4

processos em paralelo e, quando executamos esse comando, no banco de dados todos os processos tentam atualizar essas permissões ao mesmo tempo, causando esse erro de duplicação de chave na tabela.

No arquivo airflow.cfg temos uma variável que pode nos ajudar a evitar que esse erro apareça. Podemos pesquisar pela variável update_fab_perms e setar o valor dela como False:

```
update_fab_perms = False
```

Dessa forma, o webserver não irá atualizar as permissões, o que irá prevenir o erro.

Mas vale ressaltar que esse erro não afeta no desenvolvimento do nosso projeto, então ele não prejudica o desenvolvimento do curso. De todo modo, ele deve ser resolvido nas próximas versões do Airflow que forem lançadas.