Ian Stoianov Loureiro

# Data Lake - Ifood - AWS

- Considerações iniciais

As linguagens disponíveis para desenvolvimento da solução de conhecimento próprio do autor, eram SQL e Python.

O desenvolvimento em SQL poderia ser feito inteiramente em uma ferramenta somente, utilizando AWS Athena.

Para efetuar o desenvolvimento em Python, duas opções disponíveis dentro da AWS seriam um Jupyter Notebook dentro do SageMaker, que poderia acarretar custos, devido ser uma ferramenta de treino de modelos de *Machine Learning*, ou uma função Lambda.

Foi escolhido efetuar todo o processo em SQL, dentro do AWS Athena, pois o volume de dados, a depender de como fosse efetuado as consultas e tratamentos, poderia extrapolar o limite de 3gb de memória por instância de Lambda disponibilizado pela Amazon.

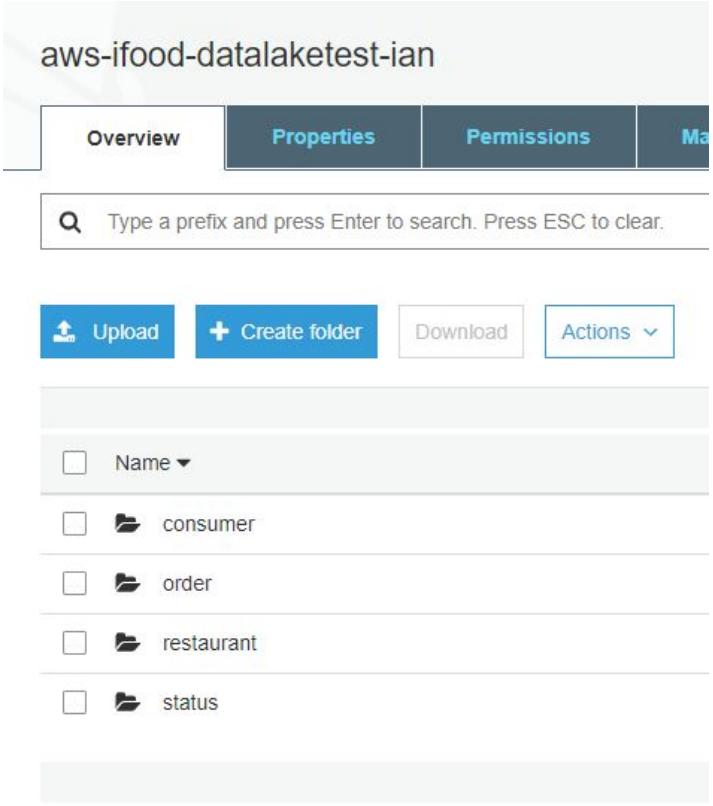
Com algumas otimizações, não há impedimentos para o uso de consultas no Lambda, com armazenamento em *dataframes* Pandas, e comandos de join entre *dataframes*. Inclusive, um desafio encontrado na montagem da consulta de itens por pedido, poderia ter sido mais facilmente resolvido com o uso de Python, mas, naquela etapa, foi julgado que o tempo para desenvolver nova solução poderia extrapolar o prazo para entrega.

- Carregamento dos dados

Foi criado um bucket no S3 com o nome ‘aws-ifood-datalaketest-ian’, na região sa-east-1, onde, usando o cliente de linha de comando da AWS, foi executado o seguinte comando para efetuar a cópia dos arquivos raw para o bucket indicado:

aws s3 sync s3://ifood-data-architect-test-source/ s3://aws-ifood-datalaketest-ian/

Para facilitar a organização, os arquivos foram separados em pastas, gerando a seguinte estrutura:




Com os dados disponíveis num bucket, foi utilizado o AWS Glue, com um crawler por arquivo, para conectar o bucket ao AWS Athena, tornando-o pesquisável.

| Name | Schedule | Status | Logs | Last runtime |
|---|---|---|---|---|
| ifood_consumer | | Ready | Logs | 47 secs |
| ifood_orders | | Ready | Logs | 37 secs |
| ifood_restaurant | | Ready | Logs | 46 secs |
| ifood_status | | Ready | Logs | 38 secs |

Os quatro crawlers foram executados e a camada raw ficou disponível no banco demo_db.

- Validação de dados para camada trusted

Foram utilizadas uma variedade de consultas para identificar anomalias dos dados das tabelas fornecidas, que tiveram que ser tratadas a fim de garantir qualidade dos dados gravados no nível trusted.

-Validação de unicidade de pedido:

Utilizada a seguinte consulta (1):

```
SELECT order_id,
count(*)
FROM "demo_db"."ifood_orderorder"
group by order_id
having count(*) > 1
limit 10;
```

Com essa consulta, foi identificado existência de duplicidade para vários pedidos, num total de 1.241.965 entradas com order_id aparecendo 2 ou mais vezes, sendo as diferenças visíveis somente nos campos cpf e created_at, com uma entrada em 2018 e outra em 2019. A fim de determinar o tratamento adequado, foi utilizado a seguinte consulta (2):

```
SELECT year(from_iso8601_timestamp(pedido.order_created_at)),
         count(*)
FROM "demo_db"."ifood_statusstatus" status
INNER JOIN "demo_db"."ifood_orderorder" pedido
    ON pedido.order_id = status.order_id
        AND pedido.order_created_at = status.created_at
GROUP BY  year(from_iso8601_timestamp(pedido.order_created_at))
```

que indicou que todas entradas da tabela de pedidos que possuem pelo menos uma equivalência de data e hora de criação com a tabela de status, são na verdade de 2019:

| _col0 | _col1 |
|---|---|
| 2019 | 3634939 |

Sendo assim, o tratamento dessas duplicidades poderia seguramente ignorar as entradas em 2018.

Como solução para este problema, foi efetuado um segundo join com a tabela de status, com este utilizando, além do order_id, a data de criação do pedido e do status, de forma a garantir que o pedido duplicado não fizesse parte do resultado final.

Usando consulta análoga à (1) na tabela de status, foi identificado que há duplicidade de várias entradas, com repetição dos valores de todos os campos

Visto o que foi identificado, a utilização de um row_number em join com a tabela de pedidos garante que essa repetição não cause duplicidade no resultado final.

Análogo da consulta (1) foi efetuado para as tabelas de consumidor e de restaurante, e não foi identificado duplicidade de chave primária em nenhuma das duas.

- Consultas para inserção em camada trusted.

-Tabela de pedidos com dados de último status, restaurante, e cliente:

A consulta montada para a criação de camada Trusted das informações de pedidos foi a seguinte (comentários no código removidos) (3):

```
SELECT pedidos.order_id,
cast(null as varchar) as cpf,
pedidos.customer_id,
pedidos.delivery_address_city,
pedidos.delivery_address_country,
pedidos.delivery_address_district,
cast(null as varchar) as delivery_address_external_id,
cast(null as varchar) as delivery_address_latitude,
cast(null as varchar) as delivery_address_longitude,
pedidos.delivery_address_state,
cast(null as varchar) as delivery_address_zip_code,
pedidos.items,
pedidos.merchant_id,
pedidos.merchant_latitude,
pedidos.merchant_longitude,
pedidos.merchant_timezone,
pedidos.horapedido,
pedidos.horapedidoconvertido,
pedidos.order_scheduled,
pedidos.order_total_amount,
pedidos.origin_platform,
pedidos.horastatus,
```

```
pedidos.horastatusconvertido,
pedidos.status_id,
pedidos.ultimostatus,
pedidos.language,
pedidos.datacadastrocliente,
pedidos.cadastroativo,
cast(null as varchar) as customer_name,
pedidos.customer_phone_area,
cast(null as bigint) as customer_phone_number,
pedidos.datacadastrorestaurante,
pedidos.restauranteativo,
pedidos.price_range,
pedidos.average_ticket,
pedidos.takeout_time,
pedidos.delivery_time,
pedidos.minimum_order_value,
pedidos.merchant_city,
pedidos.merchant_state,
pedidos.merchant_country,
cast(pedidos.horapedidoconvertido as date) datapedidolocal
FROM
    (SELECT pedido.order_id,
         pedido.customer_id,
         pedido.delivery_address_city,
         pedido.delivery_address_country,
         pedido.delivery_address_district,
         pedido.delivery_address_state,
         pedido.items,
         pedido.merchant_id,
         pedido.merchant_latitude,
         pedido.merchant_longitude,
         pedido.merchant_timezone,
         cast(from_iso8601_timestamp(pedido.order_created_at) as
timestamp) as horapedido,

cast(at_timezone(from_iso8601_timestamp(pedido.order_created_at),pedido
.merchant_timezone) as timestamp) AS horapedidoconvertido,
         pedido.order_scheduled,
         pedido.order_total_amount,
         pedido.origin_platform,
```

```
        cast(from_iso8601_timestamp(status.created_at) as timestamp)
AS horastatus,
        cast(at_timezone(from_iso8601_timestamp(status.created_at),
pedido.merchant_timezone) as timestamp) AS horastatusconvertido,
        status.status_id,
        status.value AS ultimostatus,
        consumer.language,
        cast(from_iso8601_timestamp(consumer.created_at) AS date) AS
datacadastrocliente,
        consumer.active AS cadastroativo,
        consumer.customer_phone_area,
        cast(from_iso8601_timestamp(restaurante.created_at) as date)
AS datacadastrorestaurante,
        restaurante.enabled restauranteativo,
        restaurante.price_range,
        restaurante.average_ticket,
        restaurante.takeout_time,
        restaurante.delivery_time,
        restaurante.minimum_order_value,
        restaurante.merchant_city,
        restaurante.merchant_state,
        restaurante.merchant_country,
        row_number() over(partition by pedido.order_id
             ORDER BY  from_iso8601_timestamp(pedido.order_created_at)
DESC,
                       from_iso8601_timestamp(status.created_at) desc
                  ) linha
    FROM "demo_db"."ifood_restaurantrestaurant" restaurante
INNER JOIN "demo_db"."ifood_orderorder" pedido
ON pedido.merchant_id = restaurante.id
INNER JOIN "demo_db"."ifood_statusstatus" status
        ON pedido.order_id = status.order_id
INNER JOIN "demo_db"."ifood_statusstatus" statusf
        ON pedido.order_id = statusf.order_id
     AND from_iso8601_timestamp(pedido.order_created_at) =
from_iso8601_timestamp(statusf.created_at)
INNER JOIN "demo_db"."ifood_consumerconsumer" consumer
        ON pedido.customer_id = consumer.customer_id
    ) pedidos
WHERE pedidos.linha = 1;
```

Para a anonimização de dados pessoais, foi escolhido nulificar os campos, a fim de evitar que pudessem ser rastreados para quaisquer dados identificáveis. Desses, somente foi mantido o campo customer_id,, para que seja possível identificar pedidos por usuário ou contagem de usuários no período, por exemplo, assim como a data de criação da conta do usuário.

A criação da tabela trusted usou o seguinte comando (4):

```
CREATE TABLE ifood_trusted.pedidos_completo_anonimo
WITH (
  format='PARQUET',
  partitioned_by=array['datapedidolocal'],
  external_location='s3://aws-ifood-ian-triusted/Pedidos/'
) AS
SELECT ...
```

Que determinou particionar a tabela pela data de pedido local do restaurante, assim como salvar na tabela e bucket criados para propósito de armazenar a camada confiável

Após executada a criação da tabela, foi executada nova verificação, e não foi identificado duplicidade do campo order_id

-Tabela de status por pedido

Como verificado na etapa de validação, foi identificado duplicidade de status_id em múltiplas entradas, foi utilizado um row_number, particionando pelo order_id e status_id, a fim de garantir que somente uma entrada de cada status fosse armazenada (5).

```
select pedidos.order_id,
       pedidos.status_id,
       pedidos.status,
       pedidos.criacaopedido,
       pedidos.horaevento
from (
SELECT
         pedido.order_id,
         status.status_id,
         status.value status,
         cast(from_iso8601_timestamp(pedido.order_created_at) as
timestamp) criacaopedido,
         cast(from_iso8601_timestamp(status.created_at) as timestamp)
horaevento,
        row_number() over(partition by pedido.order_id,
status.status_id /*particao no row_number garante pegar só uma entrada
por status*/
```

```
ORDER BY  from_iso8601_timestamp(pedido.order_created_at) DESC,
/*decrescrente para recuperar o valor mais recente por partição*/
          from_iso8601_timestamp(status.created_at) DESC)
          linha
FROM "demo_db"."ifood_orderorder" pedido
INNER JOIN "demo_db"."ifood_statusstatus" status
    ON pedido.order_id = status.order_id
  ) pedidos
  where pedidos.linha = 1;
```

A tabela na camada confiável foi criada com o seguinte comando (6):

```
CREATE TABLE ifood_trusted.status_por_pedido
WITH (
  format='PARQUET',
  external_location='s3://aws-ifood-ian-triusted/Status/'
) AS
select ...
```

Não foi identificado duplicidade do campo status_id após a criação da tabela

-Tabela de *items* por pedido:

Campo *items* possui um array de JSON, mas devido a forma que foi armazenado esse campo na tabela de pedidos (string), não foi possível usar mapeamento do array para geração de um item por linha. Mesmo armazenando somente os campo order_id e *items*, e criando tabela com mapeamento do JSON, ainda foi encontrado erro de tipo de dados.

Por haver um número variável de entradas no array *items*, para gerar o resultado de um item por linha seria necessário uso do método UNNEST, que não funcionou da forma esperada com campo do tipo string.

Como detalhado nas considerações iniciais, esse problema poderia ter sido resolvido com o uso de AWS Lambda e Python.

Portanto, foi avaliado o comprimento máximo do campo *items*, e gerado uma tabela, ‘indices’, contendo uma sequência de números de 1 até o comprimento máximo, que no caso, era 100. Usar essa tabela de forma programática dentro da consulta aumentou o tempo de execução, de forma a estourar o tempo máximo de 30 minutos.Geração dessa tabela, utilizada somente para este propósito, foi feita com o seguinte código (7):

```
      CREATE TABLE ifood_trusted.indices
      WITH (
        format='PARQUET'
      ) AS
      select number from (
      select sequence (1,100) seq
```

```
        )
        cross join unnest(seq) as t(number)
```

Por questão de velocidade de execução da consulta, também foi utilizada a tabela de pedidos anonimizados no nível trusted, por ter sido armazenada em parquet (redução de ~20 minutos para ~8 minutos).

Com isso, foi efetuado um cross join da tabela de pedidos com a tabela de índices, filtrando de forma que cada linha fosse multiplicada quantas vezes tivesse itens, e com isso, foi possível usar esse índice para buscar uma posição por vez do array.

O campo garnishItems possui outro array JSON, mas não foi separado um por linha, a fim de manter a relação 1 pedido para n itens com a tabela de pedidos, e o JSON foi disponibilizado para caso seja necessário análise desse campo.

Com isso, o código para a seleção foi o seguinte (8):

```
      SELECT pedidos.order_id,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.name') as nomeitem,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.unitPrice.value') as valorunitario,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.addition.value') as adicionalunitario,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.discount.value') as descontounitario,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.quantity') as quantidade,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.totalValue.value') as valoritem,

json_array_length(json_extract(json_array_get(pedidos.items,
indices.number-1), '$.garnishItems')) as numeroguarnicoes,
                json_format(json_extract(json_array_get(pedidos.items,
indices.number-1), '$.garnishItems')) as guarnicoes,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.totalAddition.value') as adicionaltotal,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.totalDiscount.value') as descontototal,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.externalId') as externalId,
                json_extract_scalar(json_array_get(pedidos.items,
indices.number-1), '$.integrationId') as integrationId,
                cast(pedidos.items as varchar) items
      FROM "ifood_trusted"."pedidos_completo_anonimo" pedidos
```

```
CROSS JOIN "ifood_trusted"."indices"
    WHERE json_array_length(pedidos.items)>= indices.number;
```

E o bloco para criação da tabela (9):

```
CREATE TABLE ifood_trusted.items_por_pedido
WITH (
  format='PARQUET',
  external_location='s3://aws-ifood-ian-triusted/Items/'
) AS
SELECT ...
```

Com a execução dos CTAS das 3 tabelas, o banco ifood_trusted se encontra da seguinte forma:


O bucket com os dados do data lake de nível trusted está disponível em:

s3://aws-ifood-ian-triusted/