# Estudo de caso - Criação de Pipeline de ingestão com Retorno de API pública.

Sumário

[Estudo de caso - Criação de Pipeline de ingestão com Retorno de API pública. 1](#_Toc177370461)

[Objetivo 3](#_Toc177370462)

[Descrição da Solução Alto nível: 3](#_Toc177370463)

[Descrição da Solução (detalhada): 5](#_Toc177370464)

[jb\_ing\_api\_openbrewerydb\_breweries\_to\_bronze 5](#_Toc177370465)

[jb\_ing\_api\_openbrewerydb\_breweries\_to\_silver 8](#_Toc177370466)

[jb\_gold\_openbrewerydb\_breweries 10](#_Toc177370467)

[wkf\_openbrewerydb\_breweries 11](#_Toc177370468)

[jb\_pipe\_openbrewerydb\_breweries 13](#_Toc177370469)

[As ferramentas utilizadas ( AWS Glue / Athena / S3 / SNS) 18](#_Toc177370470)

[Configuração do Ambiente. 18](#_Toc177370471)

## Objetivo

Objetivo deste documento é detalhar a criação de um pipeline de ingestão e tratamento de dados a partir de uma API Pública.

<https://api.openbrewerydb.org/breweries>

A ideia é criar algumas componentes na cloud para que os dados dessa API possam ser ingeridos em um ambiente cloud e com isso retratar conceitos / Problemas e possíveis soluções utilizadas em ambientes analíticos e explicar como as Decisões foram tomadas.

O pipeline criado tem objetivo de exemplificar funcionalidades gerais na criação de uma plataforma de dados.

## Descrição da Solução Alto nível:

Diagrama

Descrição gerada automaticamente

* Para Essa solução foram consideradas 3 camadas distintas para armazenamento dos dados no Data Lake.
  + **Bronze** 🡪 Responsável pelos dados brutos, sem alteração, armazenados de forma isolada para garantir reprocessamento em caso de falha do pipeline ou indisponibilidade da fonte de dados.
  + **Silver** 🡪 Responsável Pela transformação e padronização dos dados em formato colunar (parquet), também é nessa camada que serão realizadas normalizações técnicas baseado em melhores práticas da plataforma como transformação de data-Type, normalização de nomes de colunas, de-duplicação de dados, padronização de dados etc. (Aplicação de regras técnicas).
  + **Gold** 🡪 Responsável Pela transformação dos dados baseados em regras / necessidades de negócio, como criação de tabelas agregadas, visões etc.

A execução do pipeline Segue o Seguinte fluxo:

1. Um Glue Job é executado e faz a conexão com a API , coleta as informações brutas da API e salva os dados no bucket do s3 para garantir isolamento e se necessário reprocessamento dos dados com base em uma data passada. Para garantir mais visibilidade dos dados que estão sendo trafegados no Data Lake, também é criada uma tabela no catálogo do glue que pode ser acessada via Athena. Uma coluna é adicionada no final do dataset com a data de processamento no formato YYYYMMDD, assim é possível verificar o retorno da Api ao longo do tempo.
2. Um Glue job é executado na sequência para carregar os dados da camada Bronze para camada Silver. Na camada bronze os dados são carregados diariamente e controlados pela coluna adicionada no final da tabela chamada ANOMESDIA, logo considerando que os dados de retorno da API são uma “tabela de domínio”, na camada bronze temos uma foto diária desses dados para análise e consumo. Em contrapartida, na camada silver vamos manter somente a última posição dos dados, ou seja, a última foto.
3. Já na terceira para do fluxo, temos outros job do Glue responsável por ler essa ultima foto dos dados existentes na camada silver e gerar uma visão agregada.

Utilizando bibliotecas especificas, todos os 3 jobs das 3 camadas já criam suas respectivas tabelas no database bronze, silver e gold do datalake, facilitando o consumo dos dados e validação do pipeline.

Para orquestrar todo o fluxo foram propostas duas soluções simples:

Foi criado um worklfow, que é uma função do próprio AWS para iniciar os Jobs em determinada cadeia de eventos a partir de triggers de inicialização.

E uma outra proposta foi a criação de um pipeline customizado utilizando a própria ferramenta para demonstrar a versatilidade de criar estruturas complexas de controle de forma bem simples utilizando python e com baixo consumo de recurso computacional e financeiro.

Neste último exemplo também foi adicionado um exemplo de alerta de envio de e-mail automático ao final da execução do pipeline com os detalhes e métricas da execução.

## Descrição da Solução (detalhada):

### jb\_ing\_api\_openbrewerydb\_breweries\_to\_bronze

**Caminho**: scripts\jb\_ing\_api\_openbrewerydb\_breweries\_to\_bronze.py

**Partes Importantes do Código:**

Chama Api e verifica status de retorno, se status 200, converte para Json e cria dataframe pandas.

Adiciona data no formato YYYYMMDD ao dataframe para controle da ingestão

Texto

Descrição gerada automaticamente

Na sequencia, forço a mudança dos datatypes para string e escrevo os dados no s3 utilizando a biblioteca aws Wrangler, com ela já consigo escrever a tabela particionada no formato parquet e adicionar a referencia da tabela no catalogo do glue.

Texto

Descrição gerada automaticamente

Depois disso aproveitando que temos as informações do retorno da api em memória, criamos um arquivo no formato json para salvar no s3 com a mascara da data de processamento. O arquivo deve ser útil caso façamos uma analise de todo histórico via sage maker ou seja necessário reprocessar a tabela construída na camada bronze.

Texto

Descrição gerada automaticamente

Abaixo conseguimos observar o arquivo json sendo escrito uma vez ao dia no bucket.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

E no athena, é possível verificar esse histórico e consumir os dados já estruturados.

Dessa forma temos flexibilidade no uso das informações:

* Para quem deseja consumir os dados estruturados.
* Para quem deseja utilizar o arquivo json em outras analises / plataforma etc.

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

### [jb\_ing\_api\_openbrewerydb\_breweries\_to\_silver](https://us-east-1.console.aws.amazon.com/gluestudio/home?region=us-east-1#/editor/job/jb_ing_api_openbrewerydb_breweries_to_silver)

**Caminho**: scripts\jb\_ing\_api\_openbrewerydb\_breweries\_to\_silver.py

**Partes Importantes do Código:**

Faço a leitura dos dados na camada bronze utilizando a biblioteca aws Wrangler. Como não temos detalhes de como funciona o processo de update de dados na fonte de dados, e aparentemente é uma fonte de dados fixa, pegamos somente sempre a ultima “foto” dos dados salvos na camada bronze e consideramos como sendo o ultimo dado valido.

Porem, poderia ser feito outras formas de manter integridade dos dados para uso na camada GOLD, como um merge / rankig das informações para pegar somente os últimos registros atualizados, uma SCD e etc.

Essa decisão vai depender basicamente de como os dados são atualizados na origem para que criemos a mesma informação de-duplicada e pronta para uso na camada silver.

**Texto

Descrição gerada automaticamente**

Depois disso, verificamos se a tabela já existe no ambiente, como vamos criar a tabela no formato iceberg, fazemos as tratativas necessárias.

**Texto

Descrição gerada automaticamente**

Na sequência, criamos a tabela no formato iceberg no catalogo do glue / s3.

Na camada silver os dados estão particionados pelo campo state\_province

**Texto

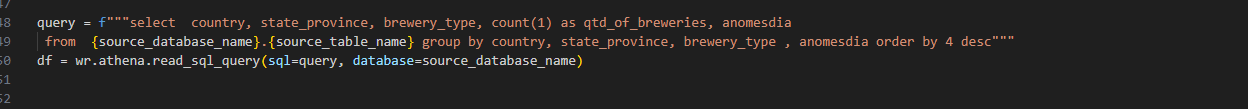
Descrição gerada automaticamente**

### [jb\_gold\_openbrewerydb\_breweries](https://us-east-1.console.aws.amazon.com/gluestudio/home?region=us-east-1#/editor/job/jb_gold_openbrewerydb_breweries)

**Caminho**: scripts\jb\_ing\_api\_openbrewerydb\_breweries\_to\_silver.py

**Partes Importantes do Código:**

Cria o dataframe utilizando uma regra de sumarização passada:



Faz as verificações de integridade caso a tabela já exista no ambiente:

**Texto

Descrição gerada automaticamente**

Cria a tabela no catálogo do glue, escrevendo o dataframe no formato iceberg no s3 particionado pela coluna state province.

**Texto

Descrição gerada automaticamente**

### [wkf\_openbrewerydb\_breweries](https://us-east-1.console.aws.amazon.com/glue/home?region=us-east-1#/v2/etl-configuration/workflows/view/wkf_openbrewerydb_breweries)

Como Exemplo de orquestrador foi criado o worklfow abaixo:

Interface gráfica do usuário, Gráfico de dispersão

Descrição gerada automaticamente

A logica é simples, primeiro temos um trigger baseado em schedule agendado para iniciar automaticamente todos os dias as 08:00 da manha. Esse trigger também poderia ser iniciado baseado em evento ou iniciado por um lambda por exemplo.

Gráfico

Descrição gerada automaticamente com confiança baixa

O trigger vai iniciar o primeiro job do pipeline, a ingestão para a camada bronze.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Após o termino do primeiro job do pipeline, existe outro trigger que espera que este job finalize com sucesso, quando ele finalizar ele inicia automaticamente o próximo job do pipeline, a ingestão na camada silver... esse processo se seque ate o fim do pipeline.

Gráfico de dispersão

Descrição gerada automaticamente

È possível monitorar e verificar a execução de todo fluxo na interface do glue:

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

Também existe uma aba para verificar os detalhes da execução em tempo real:

**Interface gráfica do usuário

Descrição gerada automaticamente**

### [jb\_pipe\_openbrewerydb\_breweries](https://us-east-1.console.aws.amazon.com/gluestudio/home?region=us-east-1#/editor/job/jb_pipe_openbrewerydb_breweries)

**Caminho**: scripts\jb\_pipe\_openbrewerydb\_breweries.py

**Partes Importantes do Código:**

O codigo é bem simples, acho que o objetivo aqui é mostrar como utilizandos funções simples de chamdas de API da própria aws conseguimos criar fluxos e controles mais complexos e logicas de tratativas dentro do pipeline , controle de qualidade e etc de forma direta e programável no python.

Primeiro temos as declarações das funções:

Uma função para envio de e-mail utililizando o serviço SNS

E outra função para verificar a existência da tabela.

**Texto

Descrição gerada automaticamente**

Também temos uma função para iniciar e controlar a execução dos Jobs do glue via boto3.

Texto

Descrição gerada automaticamente

Aqui a logica do pipeline fica simples.

Primeiro eu chamo todos os Jobs do pipeline na ordem, e na sequência faço as verificações de qualidade.

O objetivo aqui é demonstrar que programaticamente dá para fazer qualquer fluxo complexo no ETL

Texto

Descrição gerada automaticamente com confiança média

Na sequencia eu faco algumas verificações adicionais como por exemplo a quantidade de registros inseridos em cada tabela.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

No final do job eu criei também um alerta que chega por e-mail avisando os status do processamento do pipeline.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Quando job finalizar ele vai colocar essas strings formatadas em um tópico SNS que vai disparar as informações para meu e-mail. Isso é útil para acompanhar métricas de processamento de uma malha complexa ou critica por exemplo.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

[send\_api\_openbrewerydb\_breweries\_pipeline\_status](https://us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/topic/arn:aws:sns:us-east-1:373249867868:send_api_openbrewerydb_breweries_pipeline_status)

Para que o envio de e-mail do job acima funcione eu criei um tópico simples no SNS (simple Notification Service) da AWS. Basicamente é um serviço da aws que cria um tópico que vai mandar uma mensagem para todos os assinantes do tópico assim que ele receber essa mensagem, uma espécie de broadcast.

Basicamente para replicar esse comportamento você precisa criar um tópico simples

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

E adicionar uma assinatura para este tópico com seu e-mail:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

A aws vai mandar um e-mail de confirmação para seu e-mail para que vc possa receber mensagens toda vez que esse tópico for acionado.

Ao confirmar a subscrição, conforme imagem abaixo, toda mensagem que for direcionada para esse tópico vai chegar em forma de e-mail para sua assinatura.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

## As ferramentas utilizadas ( AWS Glue / Athena / S3 / SNS)

Glue, ferramenta de integração de dados da aws , serverless, permite criação de Jobs Spark para paralelismo (MPP), ou criação de Jobs python shell para execução em apenas um nó de carga de dados mais simples. Para este pipeline foi utilizado somente os Jobs do tipo python shell para minimizar o custo de processamento, visto que o dataset é muito pequeno.

<https://aws.amazon.com/pt/glue/>

Athena, serviço de consultar interativas da aws. Utilizado neste caso para validar as informações escritas no s3 no formato parquet e as tabelas criadas no catalogo do glue. O Athena tem a facilidade de utilizar consultas SQL normais para verificação e validação das informações além se poder ser fonte de dados para outros serviços da aws como QuickSight e etc.

<https://aws.amazon.com/pt/athena/>

S3, serviço de armazenamento de dados na nuvem, neste case utilizado para fazer a permanência dos dados no formato parquet para que possa sem consumidos posteriormente pelo Athena.

<https://aws.amazon.com/pt/s3/>

SNS, Simple notification service. Serviço de mensageria utilizado neste case para exemplificar a funcionalidade de alerta ao termino de pipeline de dados . Como foi executando de dentro de um código python, pode suportar logicas complexas para envio das informações como controle de fluxo mais elaborados.

<https://aws.amazon.com/pt/sns/>

## Configuração do Ambiente.

Para criação deste pipeline foi utilizado a maioria dos serviços de forma padrão, sem configurações adicionais ou configurações adicionais mínimas.

Para replicar os código aqui exemplificados na aws além do código é necessário:

Edição da Role **AWSGlueServiceRoleDefault** ou criação de uma role especifica para ser utilizada no Glue com acesso a escrita dos dados no S3, execução dos Jobs do glue e chamada de API do athena.

Para manter a logica mais simples, editei a role **AWSGlueServiceRoleDefault** e adicionais politicas gerenciadas pela aws dando acesso FULL nos demais servicos utilizados neste estudo.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Foram Adicionados:

* AmazonAthenaFullAccess, para as chamadas envolvendo API’s do athena.
* AmazonS3FullAccess, para as chamadas envolvendo escrita de dados no S3
* AWSGlueCOnsoleFullAccess, para as chamadas envolvendo o próprio Glue.
* AWSGlueServiceRole, para gerenciamento do Servico. (padrão)

Todos os Jobs foram criados na configuração Python Shell , utilizando somente 1/16 DPU’s, Essa escolha foi feita tendo em vista minimizar o custo do pipeline como um todo, sendo essa opção a mais barata disponível na ferramenta.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Nos Jobs em que é utilizado a biblioteca AWSwrangler é adicionado a ultima versão da biblioteca na parametrizações do glue.

--additional-python-modules awswrangler==3.2.1

Com isso na inicialização a maquina serverless vai automaticamente instalar a versão apontada para ser utilizada na execução do job.

<https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-libraries.html>

Isso da a ferramenta a possibilidade de usar praticamente qualquer biblioteca dentro na execução do código.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente