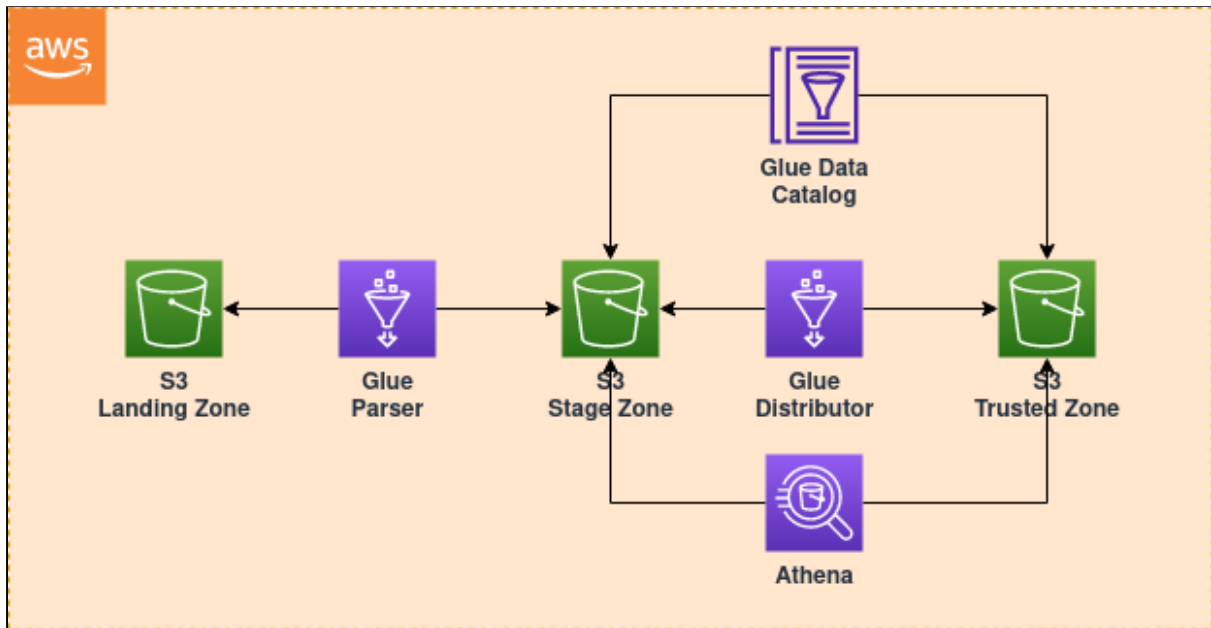


# **EVENT DISTRIBUTOR**

## SUMÁRIO

ARQUITETURA	3
PROVISIONAMENTO	4
ARQUIVOS E MÉTODOS	4
DEPLOY	6
EXECUÇÃO DO FLUXO	6
RESULTADOS	6
MELHORIAS	6

## 1. ARQUITETURA



As amostras de dados são disponibilizadas em um bucket chamado pismo-landing-zone. O upload pode ser feito manualmente ou utilizando AWS CLI. Dentro da landing zone há dois diretórios: pending e processed.

- pending: responsável por armazenar os arquivos a serem processados
- processed: responsável por armazenar os arquivos já processados

Para o processamento dos dados, foi utilizado o serviço AWS Glue (PySpark). A primeira job, Parser, é responsável por ler da landing zone (pending), “parsear” os dados e salvá-los no bucket pismo-stage-zone/events. Os detalhes de processamento das jobs são explicados no capítulo 3 (Arquivos e Métodos) deste documento.

Assim que a job Parser é finalizada com sucesso, entra em execução a segunda etapa do processo, a job Distributor. Esta etapa possui a responsabilidade de ler os dados da stage zone e salvar no bucket pismo-trusted-zone, aplicando as regras e premissas do desafio.

O Glue Data Catalog foi utilizado para criar o catálogo dos dados salvos no S3 (stage e trusted). Foram utilizados crawlers para a criação das tabelas.

O Athena foi utilizado para explorar os dados processados via ANSI SQL.

## 2. PROVISIONAMENTO

Para prover a arquitetura da AWS usada no desafio foi utilizado o terraform, versão 0.14.9.

O arquivo `terraform/main.tf` contém os detalhes de cada recurso utilizado na infraestrutura (roles, policieis, buckets e etc).

Para provisionar a infraestrutura é necessário ter o terraform instalado e as credenciais da AWS configuradas. Depois disso, basta executar: `terraform apply`.

## 3. ARQUIVOS E MÉTODOS

Para solucionar o desafio, foram criados dois arquivos principais utilizando a linguagem Python: `source/parser.py` e `source/distributor.py`. Este capítulo visa explicar os métodos presentes em cada arquivo e a sequência de execução de cada etapa.

### Métodos Parser:

#### `get_pending_files`

Responsável por retornar um array contendo todos os caminhos de objetos (amostras) armazenados na landing zone pending/.

#### `parser_to_s3`

Recebe como parâmetro uma lista de caminhos de objetos armazenados no S3 (landing/pending). Para cada caminho de arquivo de amostra é realizado o parser do conteúdo (json) utilizando o método `replace`, depois disso, é criado o pandas dataframe tendo como conteúdo as linhas presentes no arquivo. Por fim, o arquivo é salvo no path: `stage-zone/events` no formato `parquet`.

#### `move_pending_files_to_processed`

Recebe como parâmetro uma lista de caminhos de objetos armazenados no S3 (landing-zone/pending). Para cada caminho de arquivo de amostra, os arquivos

são movidos para a pasta processed do bucket landing tendo como novo diretório a data atual de execução do processo, exemplo:

**pismo-landing-zone/processed/date=2022-01-20/test.json**

Por fim, o diretório pending é zerado.

### **main**

Método principal responsável por executar outros métodos na seguinte sequência:

- get\_pending\_files
- parser\_to\_s3
- move\_pending\_files\_to\_processed

### **Métodos Distributor:**

#### **define\_event\_type**

Recebe como parâmetro um Dynamic Frame do Glue. É responsável por converter o parâmetro recebido em um Data Frame do spark e adicionar a coluna **domain\_event\_type** concatenando as colunas **domain** e **event\_type** separando-as por "-". A concatenação é feita utilizando o método **concat\_ws**. O método retorna o Data Frame.

#### **define\_date\_columns**

O método recebe como parâmetro um Data Frame e adiciona as colunas **year**, **month**, **day** extraídas do campo **timestamp**. O método retorna o Data Frame.

#### **deduplicate**

Recebe como parâmetro um Data Frame. Possui a responsabilidade de "deduplicar" os dados que se repetem com base nos campos **event\_id** e **timestamp**. É utilizado SparkSQL para aplicar a "deduplicação". O método retorna um Data Frame.

### **save\_to\_trusted**

O método recebe como parâmetro um Data Frame. É responsável por salvar os dados na trusted zone no formato parquet, separados pelas colunas domain\_event\_type, year, month e day.

### **erase\_stage**

Possui a responsabilidade de zerar a stage zone. Como resultado, o diretório **events/** fica vazio, aguardando o próximo processamento/carga.

### **main**

O método tem a função de criar um Dynamic Frame com base na tabela externa events (criada via crawler) que pertence ao database pismo-stage-zone. Em sequência os seguintes métodos são acionados:

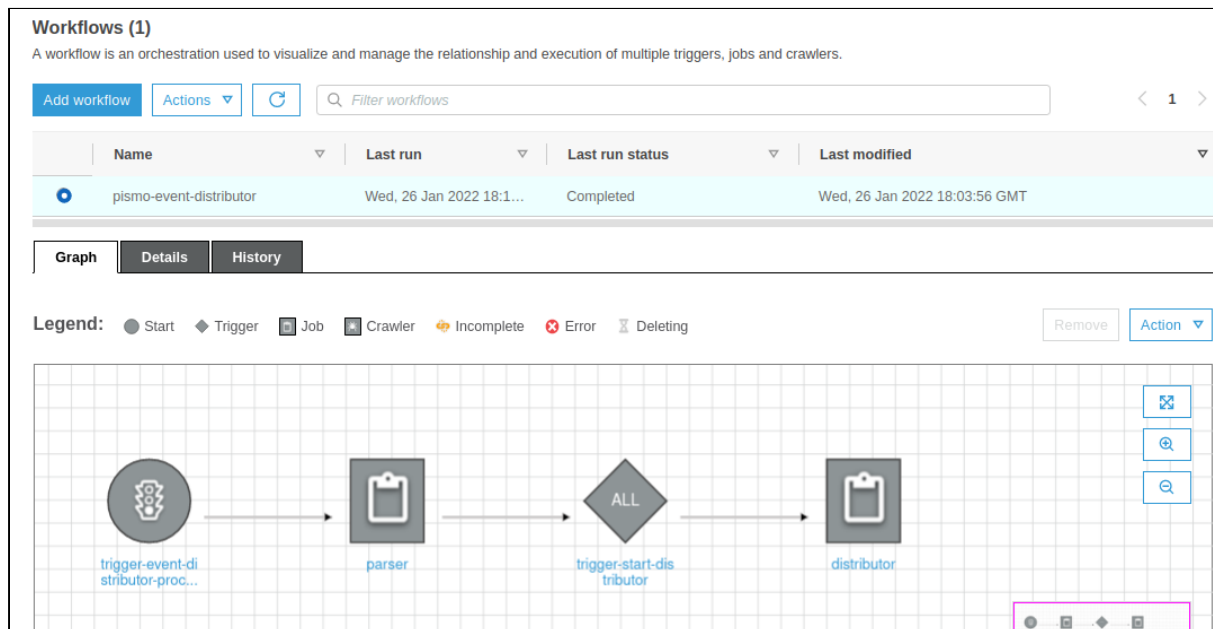
- define\_event\_type
- define\_date\_columns
- deduplicate
- save\_to\_trusted
- erase\_stage

## **4. DEPLOY**

Para realizar o deploy das jobs **source/parser.py** e **source/distributor.py** foi criado um arquivo chamado **deploy.sh**. O arquivo tem como objetivo copiar os scripts para o path: **s3://pismo-glue-scripts/** (bucket destinado a ser usado como fonte de código das jobs do Glue). A cópia é realizada via AWS CLI.

## 5. EXECUÇÃO DO FLUXO

Para executar o fluxo foi criado um Workflow do Glue:



O Workflow permite que as etapas (parser e distributor) sejam acionadas em sequência. Porém, a segunda parte só começa a rodar se a primeira finalizar com sucesso.

Os testes foram feitos acionando o fluxo manualmente, mas pode ser alterado para rodar em um horário específico ou a partir de um evento,

## 6. RESULTADOS

Depois que as jobs parser e distributor finalizam a execução, chega-se ao seguinte resultado:




**pismo-trusted-zone/events:**

Particionamento por domain\_event\_type, year, month e day:

Amazon S3 > pismo-trusted-zone > events/ > domain\_event\_type=account-creation/ > year=2021/ > month=1/ > day=10/

Amazon S3 > pismo-trusted-zone > events/ > domain\_event\_type=transaction-creation/ > year=2021/ > month=1/ > day=9/

## Exemplos de arquivos parquet:







<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class ▼
<input type="checkbox"/>	 <a href="#">part-00000-bf06e659-29f6-4c38-9d13-1646260bfcf9.c000.snappy.parquet</a>	parquet	January 25, 2022, 16:17:58 (UTC-03:00)	2.2 KB	Standard
<input type="checkbox"/>	 <a href="#">part-00001-bf06e659-29f6-4c38-9d13-1646260bfcf9.c000.snappy.parquet</a>	parquet	January 25, 2022, 16:17:58 (UTC-03:00)	2.2 KB	Standard
<input type="checkbox"/>	 <a href="#">part-00002-bf06e659-29f6-4c38-9d13-1646260bfcf9.c000.snappy.parquet</a>	parquet	January 25, 2022, 16:17:58 (UTC-03:00)	2.2 KB	Standard


## pismo-landing-zone/processed:



Dados já processados que poderão ser encontrados na landing zone, organizados por data, para um possível “reprocessamento” ou histórico.




**Objects (3)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 Inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

  Copy S3 URI  Copy URL  Download  Open  Delete **Actions ▼** **Create folder**

 Upload

 Show versions < 1 > 

<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class ▼
<input type="checkbox"/>	 <a href="#">date=2022-01-20/</a>	Folder	-	-	-
<input type="checkbox"/>	 <a href="#">date=2022-01-25/</a>	Folder	-	-	-
<input type="checkbox"/>	 <a href="#">date=2022-01-26/</a>	Folder	-	-	-



Consumo via Athena (trusted zone):

Query 1 × | ✔ Query 2 ×

1 | SELECT \* FROM "pismo-trusted-zone"."events" limit 10;

SQL Ln 1, Col 1

Run again

Cancel

Save as

Clear

Create ▼

✔ Completed

Time in queue: 0.126 sec

Run time: 1.089 sec

Data scanned: 3.94 KB

Results (10)

Copy

Download results

Search rows

< 1 > ⚙

# ▼	timestamp ▼	event_id ▼	domain ▼	event_type ▼	data
1	2021-01-12 00:08:28.000	f77dbf32-e1d9-4905-8b4d-94b23be91371	account	creation	{addresses=null, person
2	2021-01-10 11:20:19.000	0a811052-91aa-4123-a36d-9c2af6b2b050	account	creation	{person=null, company
3	2021-01-10 12:41:17.000	30766600-806e-4f60-a6a0-974f9153ea71	account	creation	{person=null, company
4	2021-01-10 05:48:03.000	51cc2d30-e546-4d7e-b25b-512e195bb29f	account	creation	{person=null, id=77190
5	2021-01-10 20:52:29.000	75414f30-0997-4dcf-8601-742ad961237b	account	creation	{addresses=null, person

## Estrutura de tabela (trusted zone):

Database

pismo-trusted-zone ▼

Tables and views Create ▼ ⚙️

🔍 *Filter tables and views*

▼ **Tables (1)** < 1 >

events	⋮
Partitioned	
timestamp	⋮
timestamp	
event_id	⋮
string	
domain	⋮
string	
event_type	⋮
string	
data	⋮
map<string,bigint>	
domain_event_type	⋮
string (Partitioned)	
year	⋮
string (Partitioned)	
month	⋮
day	⋮
string (Partitioned)	

## Performance das jobs do Glue considerando o processamento de um arquivo de 2MB (5556 registros):

### Parser:

Média de execução: 46 sec.

### Distributor:

Média de execução: 1 min.

Observação: é possível que mesmo aumentando o tamanho da amostra ainda se mantenha o mesmo tempo de execução das jobs, pois foi testado com uma massa menor de dados e o tempo médio se manteve o mesmo.

## 7. MELHORIAS

### **Pontos importantes:**

- Criar parametrização do terraform e organização de variáveis;
- Desenvolver testes unitários sobre as jobs do Glue;
- Criar rotina de deploy a partir de push no repositório e execução de testes;
- Criar arquivo de bibliotecas utilizadas nos scripts Python necessárias para rodar o projeto (Dependências);
- Monitoramento de execução do Workflow.