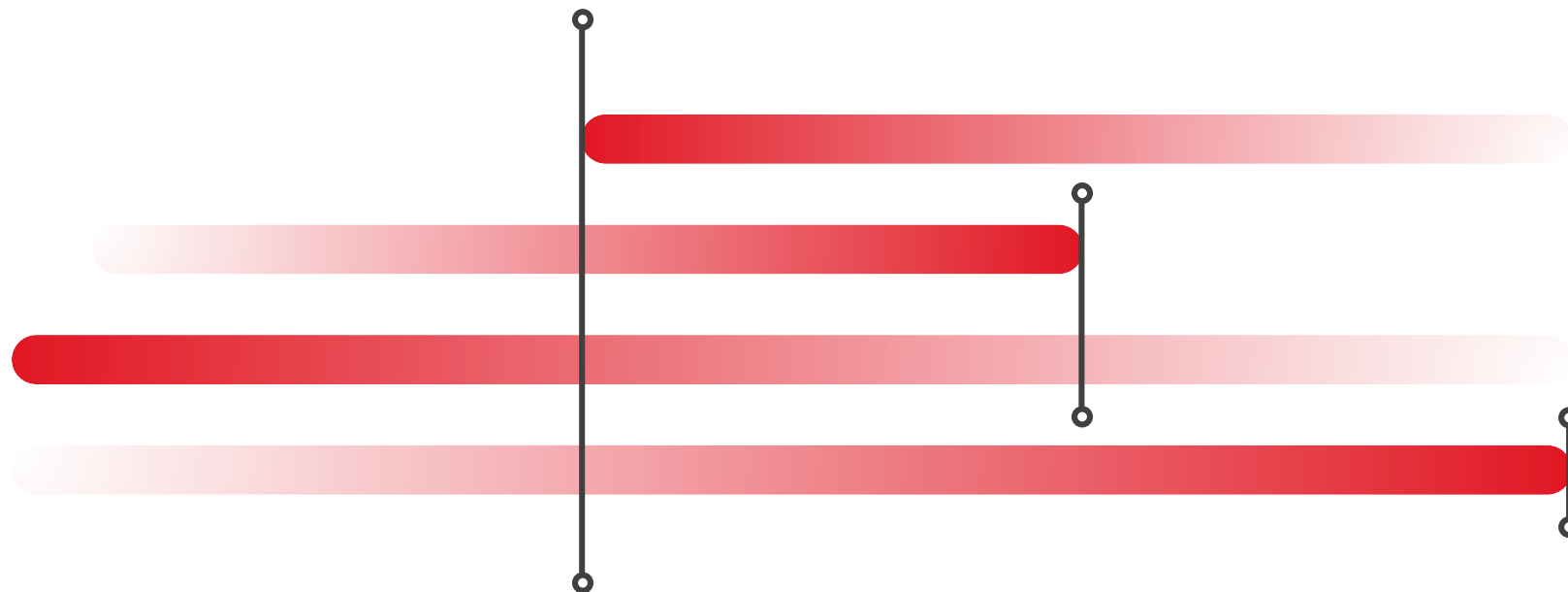




uniesp

Centro Universitário



ENGENHARIA
DE DADOS

| **LAYSA**
BELICHI




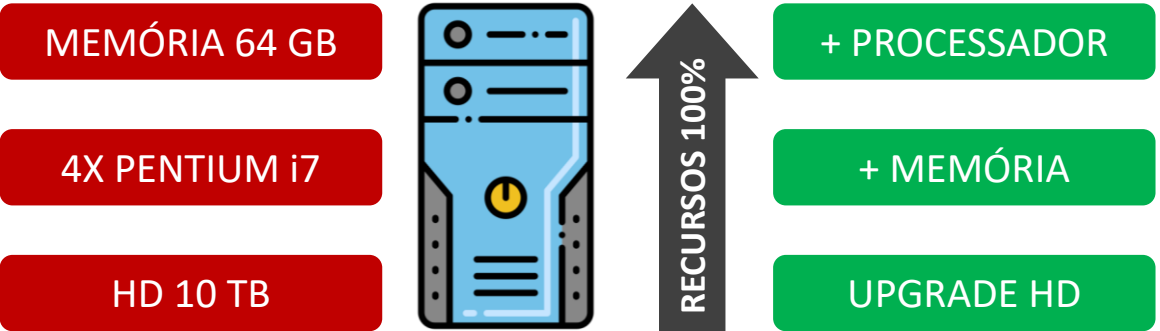


ARQUITETURAS

VERTICAL E HORIZONTAL

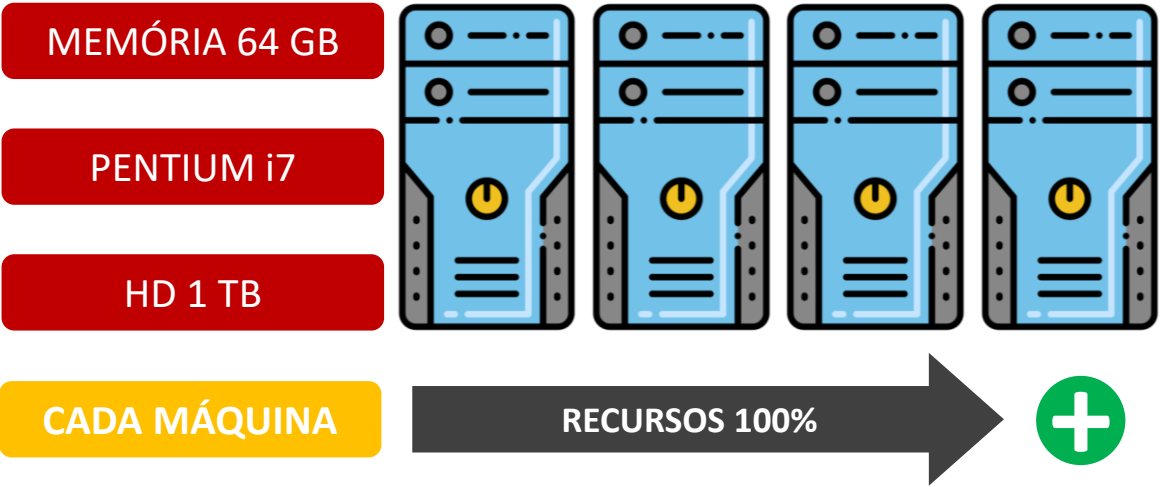
VERTICAL

 VOLTADO PARA USO DOMÉSTICO.
É OBSOLETO EM MEIO CORPORATIVO.

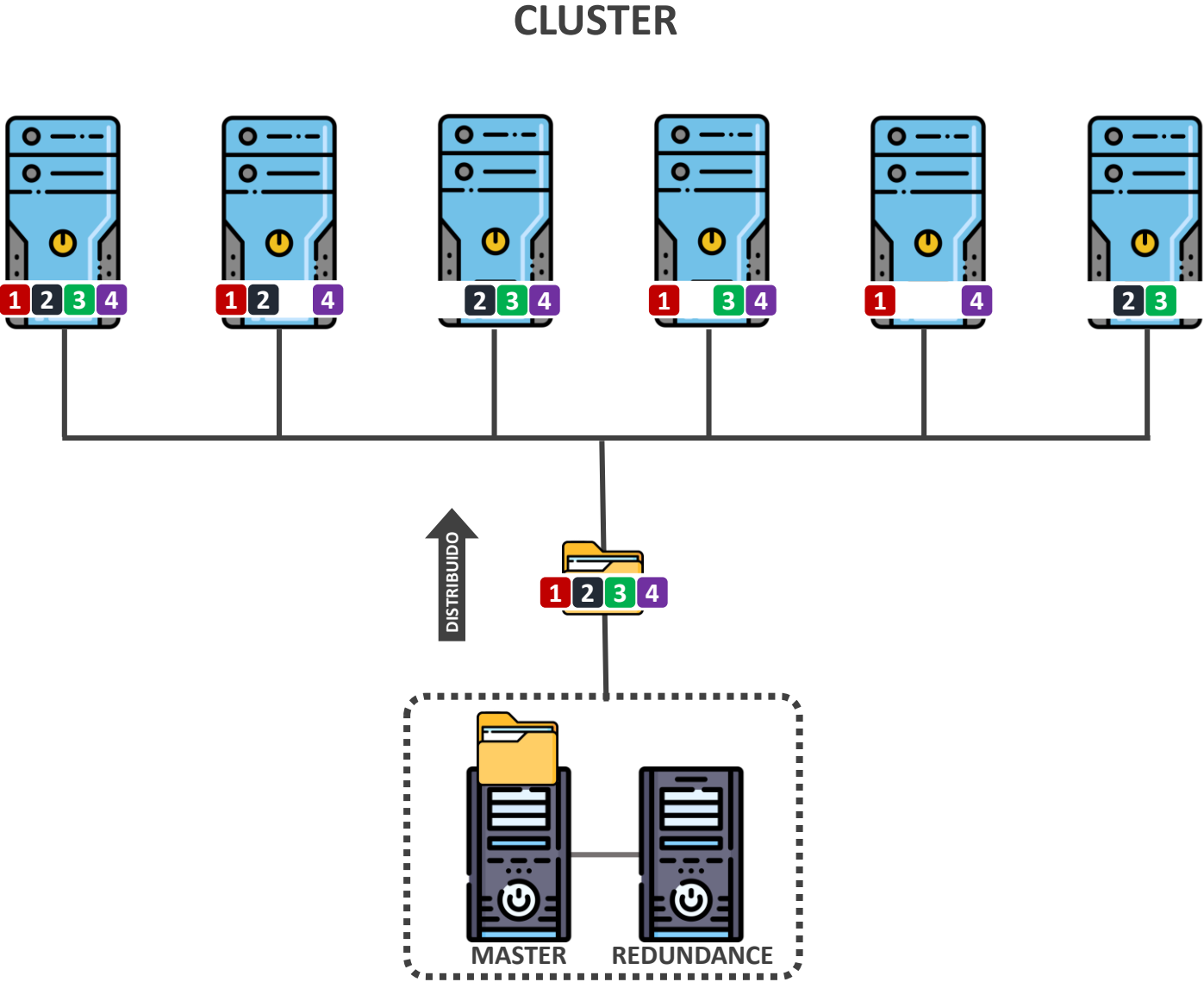


HORIZONTAL

AUMENTA A ESCALABILIDADE E DIMINUI O CUSTO.
HARDWARE UTILIZADO PODE SER MAIS BARATO E MAIS LEVE.



DISTRIBUIÇÃO DE PROCESSAMENTO
FILE SYSTEM



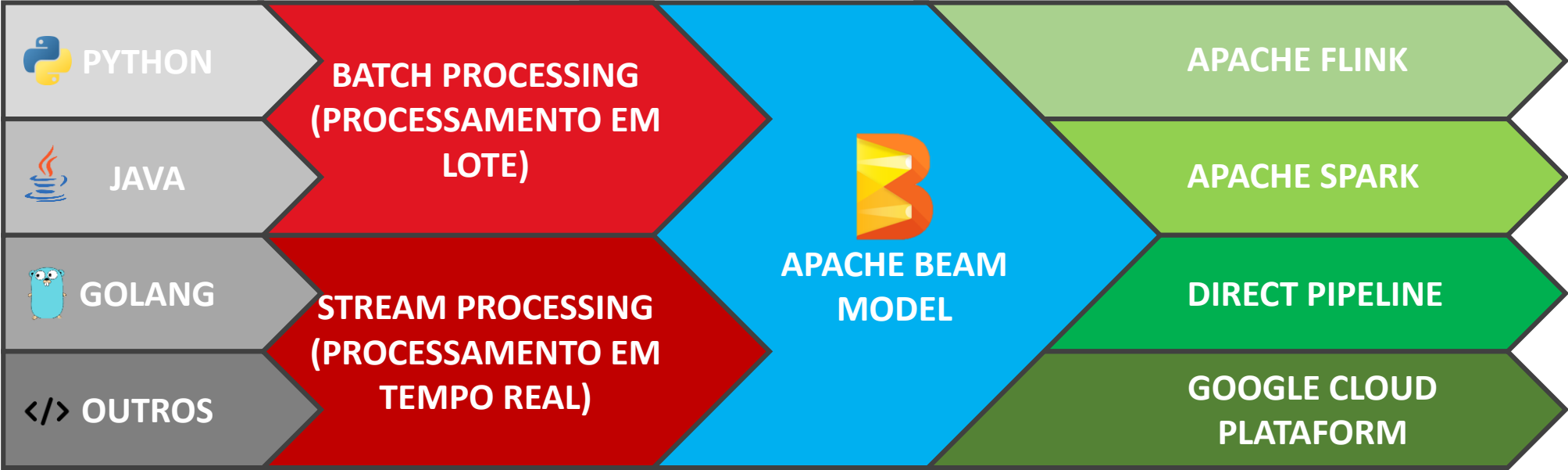


O APACHE BEAM É UM MODELO DE PROGRAMAÇÃO DE CÓDIGO ABERTO E UMA BIBLIOTECA PARA A CONSTRUÇÃO DE PIPELINES DE PROCESSAMENTO DE DADOS PARALELOS E DISTRIBUÍDOS.

ELE FOI DESENVOLVIDO PARA FORNECER UM MODELO UNIFICADO PARA EXPRESSAR TRANSFORMAÇÕES DE DADOS E EXECUTÁ-LAS EM VÁRIAS PLATAFORMAS DE PROCESSAMENTO DE DADOS, COMO:

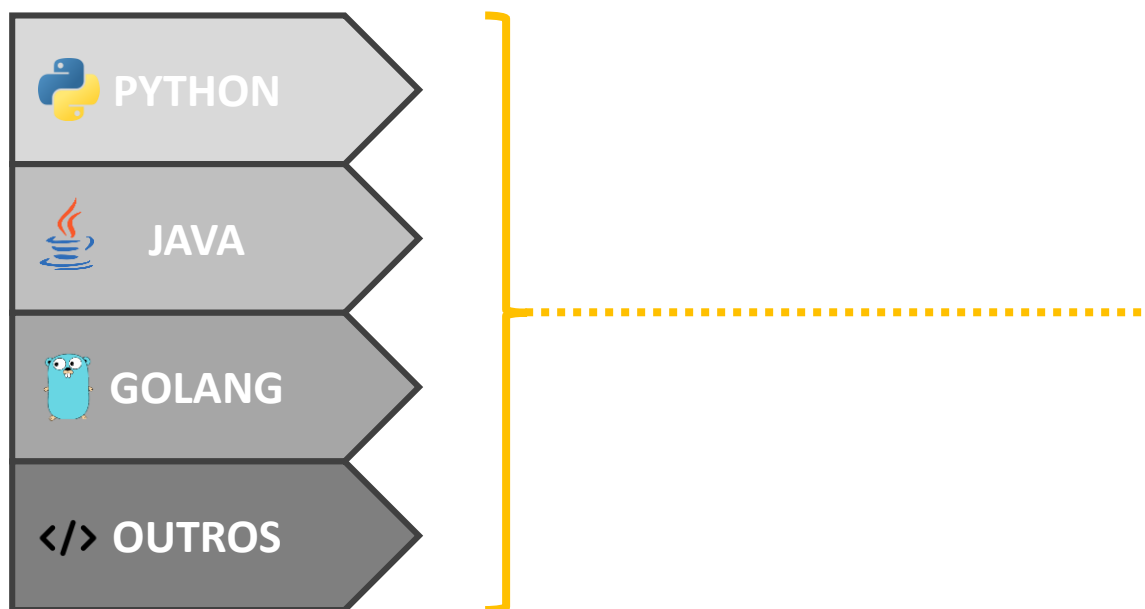
- APACHE FLINK
- APACHE SPARK
- **GOOGLE CLOUD DATAFLOW**
- OUTRAS

ARQUITETURA
APACHE BEAM



ARQUITETURA

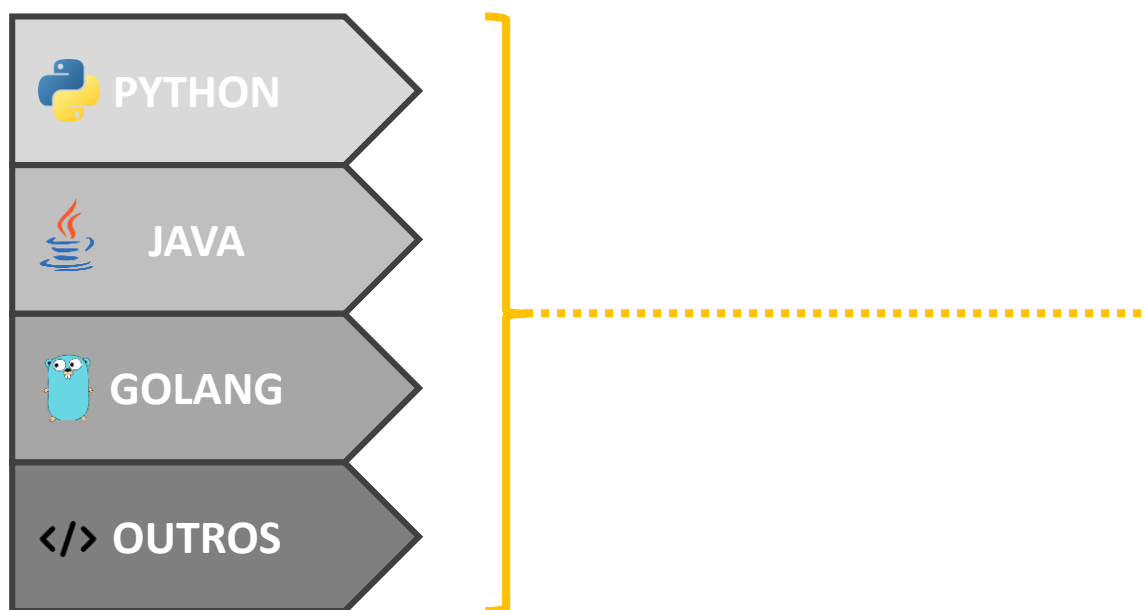
APACHE BEAM



O APACHE BEAM FOI PROJETADO PARA SER UMA PLATAFORMA DE PROCESSAMENTO DE DADOS PORTÁTIL E FLEXÍVEL, E UMA DAS MANEIRAS DE ATINGIR ESSA PORTABILIDADE É SUPORTAR VÁRIAS LINGUAGENS DE PROGRAMAÇÃO.

O SUPORTE A VÁRIAS LINGUAGENS É ÚTIL PORQUE DIFERENTES EQUIPES OU DESENVOLVEDORES PODEM TER PREFERÊNCIAS DIFERENTES OU EXPERTISE EM LINGUAGENS ESPECÍFICAS. ALÉM DISSO, PERMITE A INTEGRAÇÃO DE CÓDIGO EXISTENTE EM DIFERENTES LINGUAGENS EM UM PIPELINE DE DADOS.

O APACHE BEAM SUPORTA VÁRIAS LINGUAGENS DE PROGRAMAÇÃO, ATRAVÉS DA SUA API UNIFICADA. ISSO É POSSÍVEL DEVIDO AO FATO DE QUE OS PIPELINES SÃO DEFINIDOS USANDO UMA DSL (LINGUAGEM DE DOMÍNIO ESPECÍFICO) QUE É INDEPENDENTE DA LINGUAGEM DE PROGRAMAÇÃO.



1.EXPERIÊNCIA E PREFERÊNCIA DO DESENVOLVEDOR: SE A EQUIPE DE DESENVOLVIMENTO TEM MAIS EXPERIÊNCIA/PREFERÊNCIA POR UMA LINGUAGEM.

2.ECOSSISTEMA E BIBLIOTECAS DISPONÍVEIS: LINGUAGENS PODEM TER UM ECOSSISTEMA MAIS RICO OU BIBLIOTECAS ESPECÍFICAS QUE SÃO MELHORES PARA TAREFAS ESPECÍFICAS.

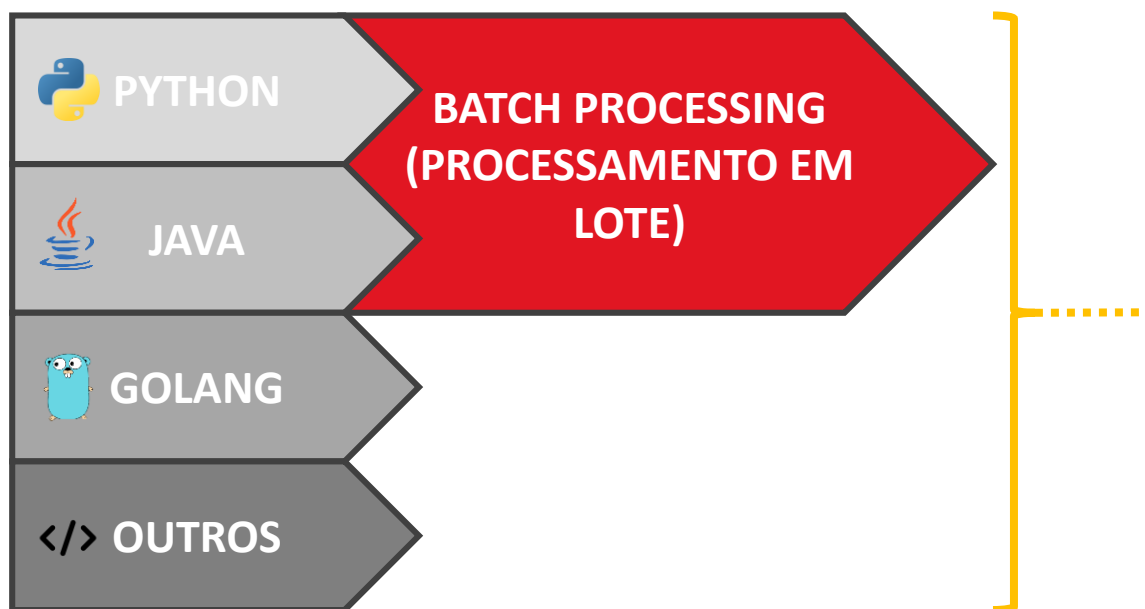
3.INTEGRAÇÃO COM OUTROS SISTEMAS: SE VOCÊ ESTIVER INTEGRANDO O APACHE BEAM COM OUTROS SISTEMAS, PODE SER LÓGICO ESCOLHER AQUELA LINGUAGEM PARA FACILITAR A INTEGRAÇÃO.

4.DESEMPENHO E ESCALABILIDADE: ALGUMAS LINGUAGENS PODEM TER VANTAGENS DE DESEMPENHO OU ESCALABILIDADE SOBRE OUTRAS EM DETERMINADOS CENÁRIOS.

5.SUORTE À PLATAFORMA DE EXECUÇÃO: ALGUNS RUNNERS (MOTORES DE EXECUÇÃO) PODEM TER UM MELHOR SUPORTE PARA DETERMINADAS LINGUAGENS.

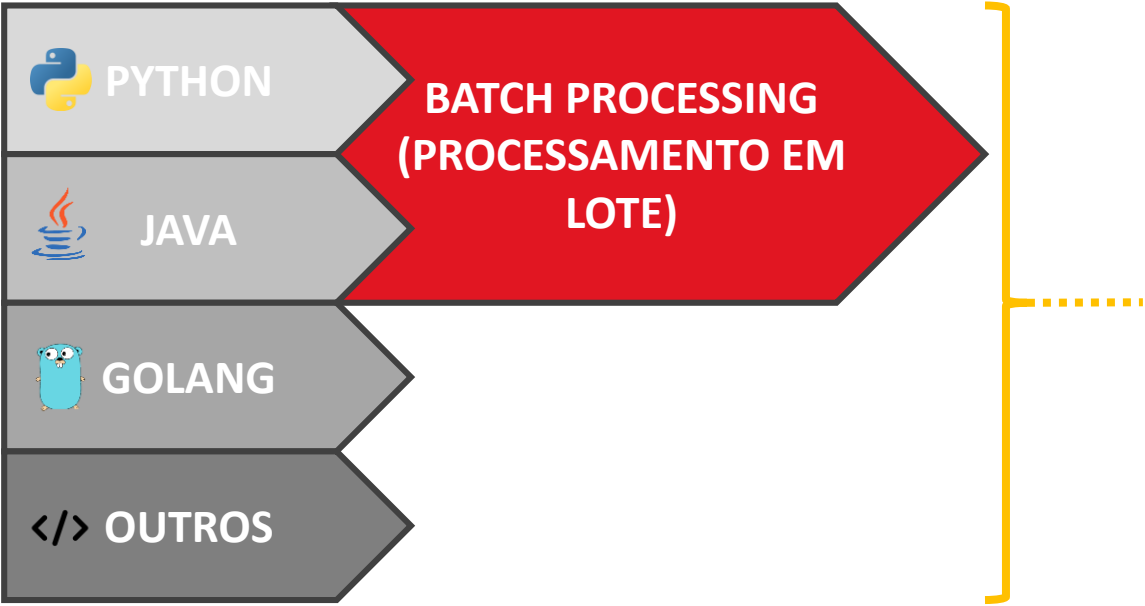
ARQUITETURA

APACHE BEAM



O APACHE BEAM É UM MODELO UNIFICADO DE PROCESSAMENTO DE DADOS DESENVOLVIDO PARA SIMPLIFICAR A CRIAÇÃO DE PIPELINES DE DADOS PARALELOS E DISTRIBUÍDOS. O BATCH PROCESSING, OU PROCESSAMENTO EM LOTE, É UM ESTILO DE PROCESSAMENTO DE DADOS NO QUAL UM CONJUNTO FIXO DE DADOS É PROCESSADO DE UMA VEZ.

QUANDO VOCÊ USA O APACHE BEAM PARA PROCESSAMENTO EM LOTE, VOCÊ ESTÁ LIDANDO COM DADOS QUE SÃO COLETADOS E PROCESSADOS EM GRANDES VOLUMES, GERALMENTE EM INTERVALOS AGENDADOS OU EM RESPOSTA A EVENTOS ESPECÍFICOS.



1. PIPELINES EM LOTE (BATCH PIPELINES): SÃO FLUXOS DE TRABALHO DE DADOS QUE PROCESSAM UM CONJUNTO FIXO DE DADOS EM LOTE.

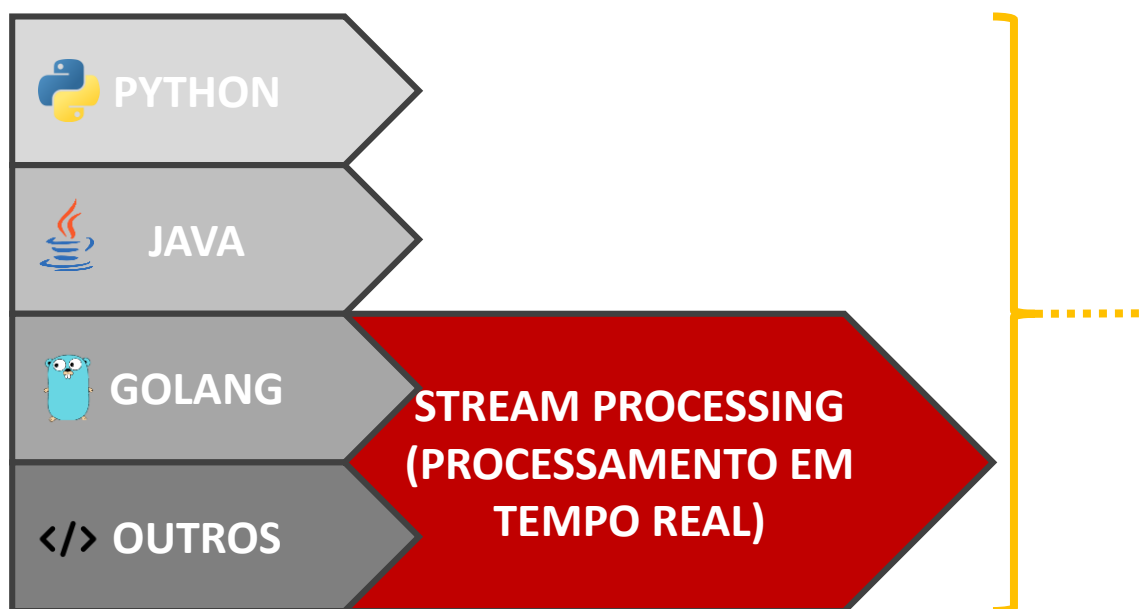
2. PCOLLECTIONS (COLEÇÕES DE PROCESSAMENTO): REPRESENTAM OS DADOS QUE SÃO PROCESSADOS PELO PIPELINE. NO CONTEXTO DE PROCESSAMENTO EM LOTE, PCOLLECTIONS GERALMENTE CONTÊM CONJUNTOS ESTÁTICOS DE DADOS.

3. TRANSFORMAÇÕES DE PARDO: AS TRANSFORMAÇÕES PARDO SÃO USADAS PARA APLICAR FUNÇÕES A CADA ELEMENTO DE UMA PCOLLECTION. ISSO PERMITE A EXECUÇÃO DE LÓGICA PERSONALIZADA EM CADA ELEMENTO DOS DADOS DURANTE O PROCESSAMENTO EM LOTE.

4. FONTES E DESTINOS: NO CONTEXTO DE PROCESSAMENTO EM LOTE, VOCÊ DEFINE DE ONDE OS DADOS SERÃO LIDOS (FONTES) E PARA ONDE OS RESULTADOS FINAIS SERÃO GRAVADOS (DESTINOS).

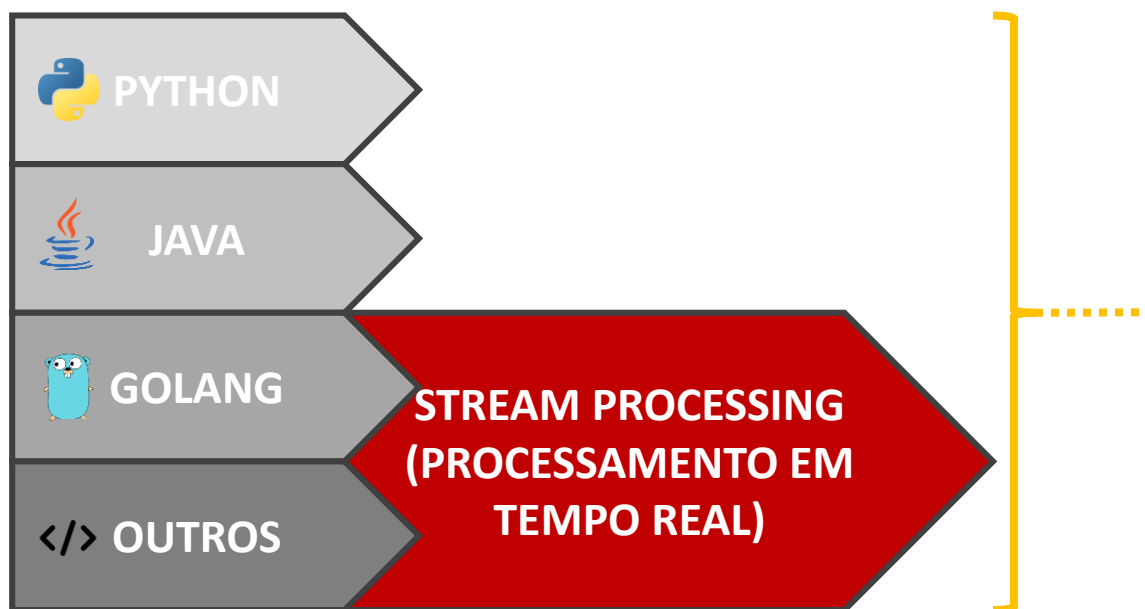
ARQUITETURA

APACHE BEAM



AO USAR O APACHE BEAM PARA STREAM PROCESSING, OS DESENVOLVEDORES PODEM CONSTRUIR PIPELINES DE DADOS QUE PROCESSAM EVENTOS À MEDIDA QUE ELES OCORREM, PERMITINDO ANÁLISES EM TEMPO REAL E TOMADA DE DECISÕES EM RESPOSTA A MUDANÇAS DINÂMICAS NOS DADOS.

O SUPORTE INTEGRADO PARA AMBOS PROCESSAMENTO EM LOTE E EM TEMPO REAL FAZ DO APACHE BEAM UMA ESCOLHA PODEROSA E FLEXÍVEL PARA DIVERSAS NECESSIDADES DE PROCESSAMENTO DE DADOS.



1. PCOLLECTIONS (COLEÇÕES DE PROCESSAMENTO):

ASSIM COMO NO PROCESSAMENTO EM LOTE, O APACHE BEAM MODELA OS DADOS COMO FLUXOS (STREAMS) DE DADOS NO PROCESSAMENTO EM TEMPO REAL.

2. TRANSFORMAÇÕES DE JANELA:

NO STREAM PROCESSING, A JANELA DE TEMPO É UM CONCEITO CRUCIAL. O APACHE BEAM PERMITE QUE VOCÊ DEFINA JANELAS DE TEMPO SOBRE AS QUAIS AS TRANSFORMAÇÕES SÃO APLICADAS. ISSO É ÚTIL PARA PROCESSAR DADOS EM INTERVALOS ESPECÍFICOS DE TEMPO.

3. EVENT TIME E PROCESS TIME:

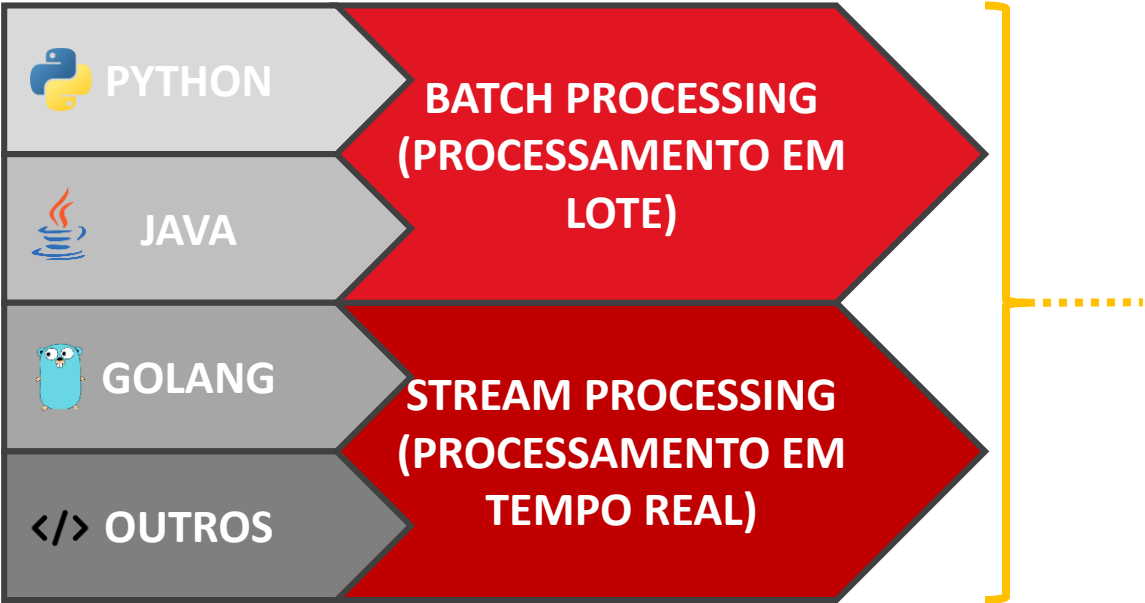
O APACHE BEAM OFERECE SUPORTE AO PROCESSAMENTO BASEADO EM EVENT TIME, QUE É O MOMENTO EM QUE UM EVENTO OCORREU NO MUNDO REAL, EM OPOSIÇÃO AO PROCESS TIME, QUE É O MOMENTO EM QUE O EVENTO É PROCESSADO. ISSO É IMPORTANTE PARA LIDAR COM ATRASOS E DADOS FORA DE ORDEM EM PIPELINES DE STREAM PROCESSING.



5. PARDO EM MODO CONTÍNUO: A TRANSFORMAÇÃO PARDO, QUE PERMITE A APLICAÇÃO DE FUNÇÕES A ELEMENTOS INDIVIDUAIS DE UMA PCOLLECTION, É USADA TAMBÉM NO STREAM PROCESSING. NO ENTANTO, NO CONTEXTO DE STREAM PROCESSING, O PARDO OPERA DE FORMA CONTÍNUA, PROCESSANDO EVENTOS À MEDIDA QUE CHEGAM.

6. TRIGGERS E ACUMULAÇÃO DE RESULTADOS: TRIGGERS PERMITEM QUE VOCÊ CONTROLE QUANDO OS RESULTADOS SÃO EMITIDOS EM UM PIPELINE DE STREAM PROCESSING. ISSO É ÚTIL PARA DEFINIR A LÓGICA DE ACUMULAÇÃO E SAÍDA DE RESULTADOS COM BASE EM CONDIÇÕES ESPECÍFICAS.

7. RUNNERS DE STREAM PROCESSING: O APACHE BEAM SUPORTA RUNNERS QUE SÃO OTIMIZADOS PARA O PROCESSAMENTO EM TEMPO REAL, COMO O GOOGLE CLOUD DATAFLOW. ESSES RUNNERS SÃO PROJETADOS PARA LIDAR COM A NATUREZA CONTÍNUA E EM TEMPO REAL DO PROCESSAMENTO DE STREAMING.



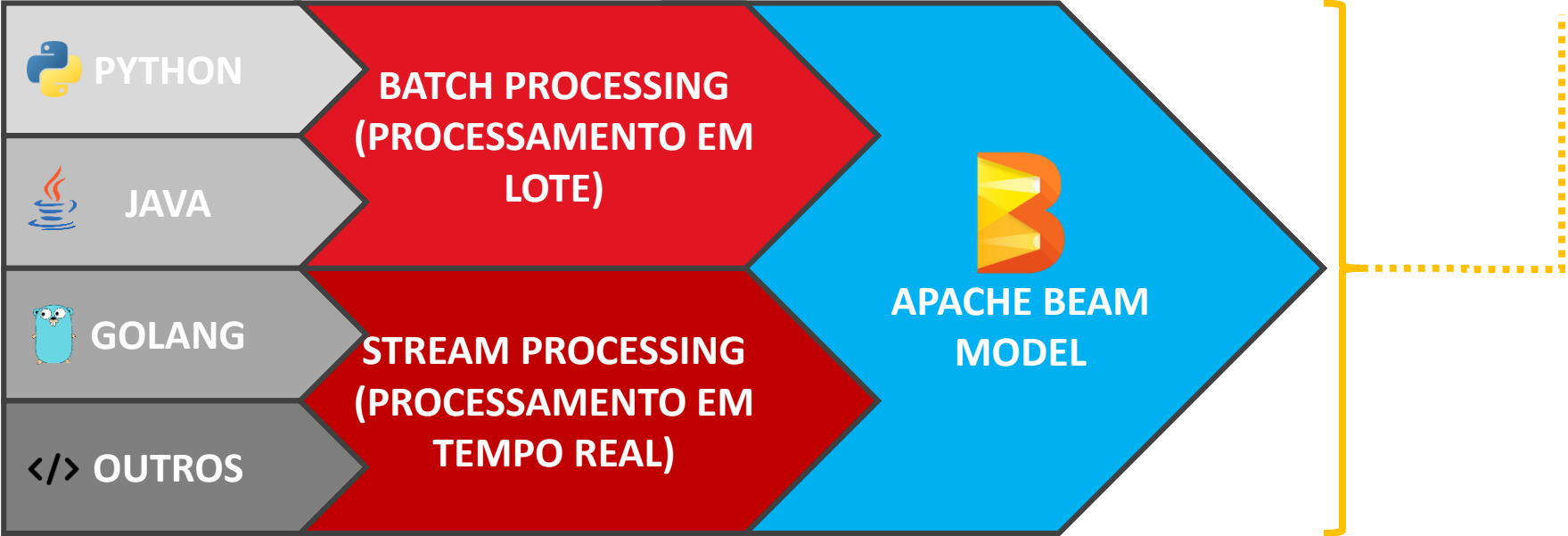
BATCH PROCESSING

- 1.Relatórios Analíticos Periódicos
- 2.Preparação de Dados para Machine Learning
- 3.ETL (Extração, Transformação e Carga) de Dados
- 4.Análise de Dados de Vendas
- 5.Geração de Relatórios Financeiros
- 6.Histórico de Comportamento do Usuário
- 7.Contagem de Palavras em Grandes Conjuntos de Texto

STREAM PROCESSING

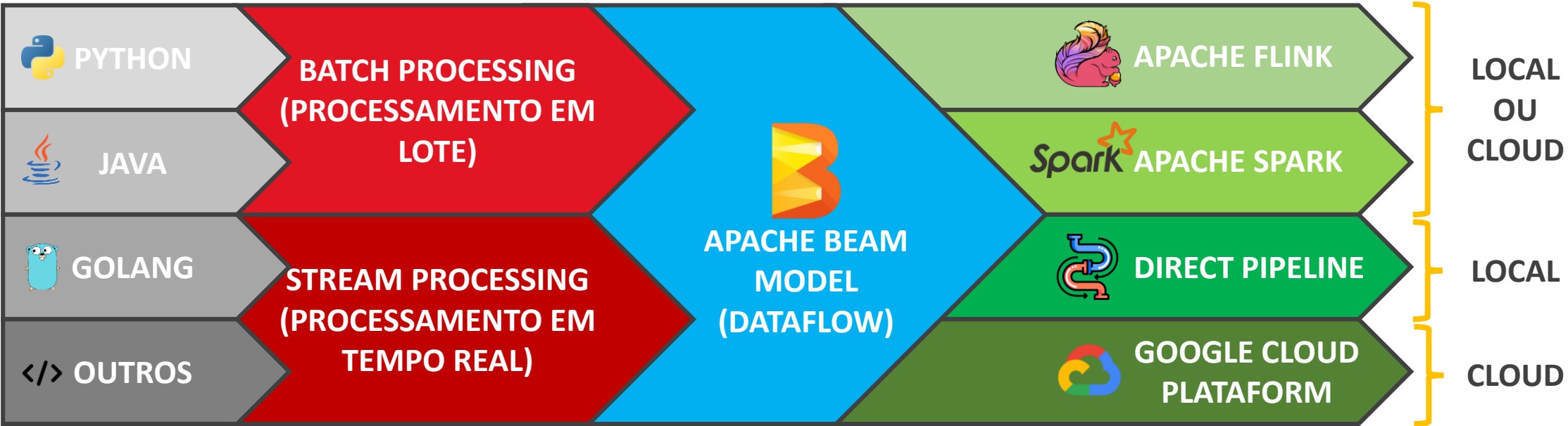
- 1.Monitoramento de Redes e Sistemas
- 2.Análise de Mídias Sociais em Tempo Real
- 3.Processamento de Eventos de IoT (Internet das Coisas)
- 4.Detecção de Fraudes Financeiras:
- 5.Sistemas de Recomendação em Tempo Real
- 6.Saúde em Tempo Real
- 7.Análise de Logs de Aplicações em Tempo Real

ARQUITETURA
APACHE BEAM

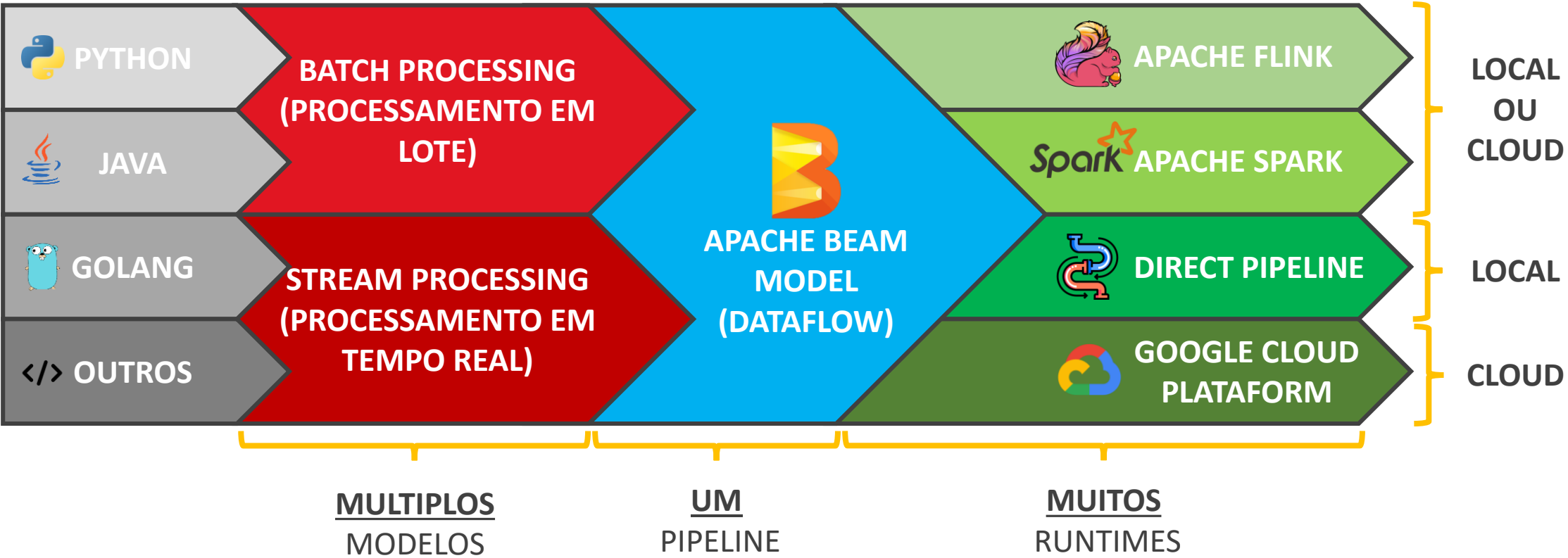


É UM MODELO DE PROGRAMAÇÃO UNIFICADO PARA
CONSTRUIR PIPELINES DE PROCESSAMENTO DE DADOS
EM LARGA ESCALA.

ARQUITETURA
APACHE BEAM



ARQUITETURA
APACHE BEAM






<https://colab.research.google.com/>



<https://github.com/laysabelici/apache-beam/tree/main>

- Utilizando Apache Beam no Google Colab
- Read Inputs
- Write Inputs
- `beam.Map/beam.FlatMap`
- `beam.Filter`
- `beam.Flatten`
- `beam.CombinePerKey`
- `beam.combiners.Count.PerKey()`
- `beam.CoGroupByKey`
- ParDo – Funções Customizadas

 Open in Colab

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
pip install apache_beam[interactive]
```


```
In [5]: #IMPORT BIBLIOTECA
import apache_beam as beam
```

```
In [15]: #DEFINIR A PIPELINE
p1 = beam.Pipeline()

voos = (
    p1
    #LER ARQUIVO E EXCLUIR CABEÇALHO
    #AS PIPES >> SIGNIFICAM QUE UM COMANDO É USADO COMO INPUT DO OUTRO
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv', skip_header_lines = 1)
    #aplicando a função lambda a cada elemento do PCollection (coleção de dados)
    #representada por record.
    #A função lambda record.split(',') está dividindo cada registro por vírgulas, criando assim uma lista de valores.
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    | "Mostrar resultados" >> beam.Map(print)
)

p1.run()
```

UTILIZAR ARQUIVO VOO_SAMPLE.CSV

 Open in Colab


```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]  
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA  
import apache_beam as beam
```

```
In [19]: #DEFINIR A PIPELINE  
p1 = beam.Pipeline()  
  
voos = (  
    p1  
    #LER ARQUIVO E EXCLUIR CABEÇALHO  
    #AS PIPES >> SIGNIFICAM QUE UM COMANDO É USADO COMO INPUT DO OUTRO  
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv', skip_header_lines = 1)  
    #aplicando a função lambda a cada elemento do PCollection (coleção de dados)  
    #representada por record.  
    #A função lambda record.split(',') está dividindo cada registro por vírgulas, criando assim uma lista de valores.  
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))  
    #Gravar de dados  
    | "Gravar dados" >> beam.io.WriteToText('/content/voos.txt')  
)  
  
p1.run()
```

BEAM

MAP

 Open in Colab

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA
import apache_beam as beam
```


```
In [19]: #DEFINIR A PIPELINE
p1 = beam.Pipeline()

voos = (
    p1
    #LER ARQUIVO E EXCLUIR CABEÇALHO
    #AS PIPES >> SIGNIFICAM QUE UM COMANDO É USADO COMO INPUT DO OUTRO
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv', skip_header_lines = 1)
    #Aplicando a função lambda a cada elemento do PCollection (coleção de dados)
    #representada por record.
    #A função lambda record.split(',') está dividindo cada registro por vírgulas, criando assim uma lista de valores.
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    #Usada para aplicar uma função a cada elemento de uma coleção de dados.
    #No entanto, nesse caso, a função é print.
    #Isso significa que a função print será aplicada a cada elemento da coleção.
    | "Mostrar resultados" >> beam.Map(print)
)

p1.run()
```

BEAM

FlatMap

 [Open in Colab](#)

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
        pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA
        import apache_beam as beam
```

```
n [32]: p1 = beam.Pipeline()


        Collection = (
            p1
            | beam.io.ReadFromText('/content/poema.txt')
            #O FlatMap é uma transformação que permite gerar zero ou mais elementos de saída
            #para cada elemento de entrada
            | beam.FlatMap(lambda record: record.split(' '))
            | beam.Map(print)
        )

        p1.run()
```

UTILIZAR ARQUIVO POEMA.TXT

BEAM

FILTER


 Open in Colab

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]  
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA  
import apache_beam as beam
```

```
In [44]: p1 = beam.Pipeline()  
  
palavras=['quatro', 'um']  
  
#FUNCTION  
def encontrarPalavras(i):  
    if i in palavras:  
        return True  
  
Collection = (  
    p1  
    | "Importar dados" >> beam.io.ReadFromText('/content/poema.txt')  
    | "Separar por vírgula" >> beam.FlatMap(lambda record: record.split(' '))  
    | "Filtrar palavras" >> beam.Filter(encontrarPalavras)  
    | "Mostrar dados" >> beam.Map(print)  
)  
  
p1.run()
```

BEAM FILTER

 Open in Colab

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA
import apache_beam as beam
```

```
In [39]: p1 = beam.Pipeline()

voos = (
    p1
    #LER ARQUIVO E EXCLUIR CABEÇALHO
    #AS PIPES >> SIGNIFICAM QUE UM COMANDO É USADO COMO INPUT DO OUTRO
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',
                                              skip_header_lines = 1)

    #aplicando a função lambda a cada elemento do PCollection (coleção de dados)
    #representada por record.
    #A função lambda record.split(',') está dividindo cada registro por vírgulas, criando assim uma lista de valores.
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    #Realiza uma separação estilo booleano, também foi indicado a coluna para a ação.
    | "Pegar voos de Los Angeles" >> beam.Filter(lambda record: record[3] == "LAX")
    #Gravar de dados
    | "Mostrar resultados" >> beam.Map(print)
)

p1.run()
```

In [46]:

```
p1 = beam.Pipeline()

negros = ('João', 'Maria', 'José')
brancos = ('Pedro', 'Ana', 'Paulo')
indios = ('Alice', 'Lucas', 'Bruna')

negros_pc = (
    p1
    | "Criar PCollection negros" >> beam.Create(negros)
)
brancos_pc = (
    p1
    | "Criar PCollection brancos" >> beam.Create(brancos)
)
indios_pc = (
    p1
    | "Criar PCollection indios" >> beam.Create(indios)
)

pessoas = (
    (negros_pc, brancos_pc, indios_pc)
    #é usada para unir várias PCollection em uma única PCollection.
    | "Unir PCollections" >> beam.Flatten()
    | "Mostrar dados" >> beam.Map(print)
)

p1.run()
```

BEAM

CombinePerKey

 Open in Colab


```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA
import apache_beam as beam
```

```
In [51]: p1 = beam.Pipeline()

Tempo_Atrasos = (
    p1
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',
                                              skip_header_lines = 1)
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    | "Realiza uma separação estilo booleano, também foi indicado a coluna para a ação."
    | "Agora vamos pegar todos os itens maior que 0 da coluna 8, referentes a atrasos."
    | "temmos que indicar o tipo de dado da coluna, pois ele trata sempre como str."
    | "Pegar voos atrasados" >> beam.Filter(lambda record: int(record[8]) > 0)
    | "Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso."
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))
    | "Soma todas as keys que são iguais, para um value unico"
    | "Somar por key" >> beam.CombinePerKey(sum)
    | "Mostra os dados"
    | "Mostrar resultados" >> beam.Map(print)
)

p1.run()
```

 Open in Colab

```
In [ ]: #INSTALADOR PARA MÁQUINA VIRTUAL, EM MÁQUINA LOCAL NÃO COLOQUE [INTERACTIVE]
pip install apache_beam[interactive]
```

```
In [5]: #IMPORT BIBLIOTECA
import apache_beam as beam
```

```
n [54]: p1 = beam.Pipeline()

Qtd_Atrasos = (
    p1
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',
                                              skip_header_lines = 1)
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    #Realiza uma separação estilo booleano, também foi indicado a coluna para a ação.
    #Agora vamos pegar todos os itens maior que 0 da coluna 8, referentes a atrasos.
    #temmos que indicar o tipo de dado da coluna, pois ele trata sempre como str.
    | "Pegar voos atrasados" >> beam.Filter(lambda record: int(record[8]) > 0)
    #Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso.
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))
    # Soma todas as keys que são iguais, para um value unico
    #Fornece a quantidade de atrasos por aeroporto
    | "Somar por key" >> beam.combiners.Count.PerKey(sum)
    #Mostra os dados
    | "Mostrar resultados" >> beam.Map(print)
)

p1.run()
```

```
In [55]: p1 = beam.Pipeline()

Tempo_Atrasos = (
    p1
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',
                                              skip_header_lines = 1)
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    | #Realiza uma separação estilo booleano, também foi indicado a coluna para a ação.
    | #Agora vamos pegar todos os itens maior que 0 da coluna 8, referentes a atrasos.
    | #temmos que indicar o tipo de dado da coluna, pois ele trata sempre como str.
    | "Pegar voos atrasados" >> beam.Filter(lambda record: int(record[8]) > 0)
    | #Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso.
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))
    | # Soma todas as keys que são iguais, para um value unico
    | "Somar por key" >> beam.CombinePerKey(sum)
)
```

```
In [58]: Qtd_Atrasos = (
    p1
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',
                                              skip_header_lines = 1)
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))
    | #Realiza uma separação estilo booleano, também foi indicado a coluna para a ação.
    | #Agora vamos pegar todos os itens maior que 0 da coluna 8, referentes a atrasos.
    | #temmos que indicar o tipo de dado da coluna, pois ele trata sempre como str.
    | "Pegar voos atrasados" >> beam.Filter(lambda record: int(record[8]) > 0)
    | #Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso.
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))
    | # Soma todas as keys que são iguais, para um value unico
    | #Fornece a quantidade de atrasos por aeroporto
    | "Somar por key" >> beam.combiners.Count.PerKey(sum)
)
```

```
In [59]: tabela_atrasos = (
    #Criando tabela sendo dicionário.
    #Essa estrutura pode ser útil em determinados casos,
    #especialmente se você estiver agrupando ou processando dados
    #que têm uma relação específica entre eles.
    {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
    | #É usada para agrupar esses dados por chave.
    | #útil quando precisamos realizar operações de agregação ou processamento
    | #em grupos específicos de dados.
    | "Group by" >> beam.CoGroupByKey()
    | "Mostrar dados" >> beam.Map(print)
)

p1.run()
```

```
In [60]: #beam.DoFn é usado para definir funções de transformação.  
#e também herda funcionalidades dessa classe para métodos específicos.  
class filtro(beam.DoFn):  
    #process é obrigatório quando usamos DoFn, ele ativa o processamento.  
    #self é uma convenção em Python para se referir à instância da própria classe.  
    #record é uma variável que representa um elemento da PCollection  
    #que está sendo processado pelo método process.  
    def process(self, record):  
        if int(record[8])>0:  
            #retorna em uma lista  
            return [record]
```

```
In [61]: p1 = beam.Pipeline()  
  
Tempo_Atrasos = (  
    p1  
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',  
                                                skip_header_lines = 1)  
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))  
    #permite aplicar uma função personalizada (uma classe que herda de beam.DoFn)  
    #a cada elemento da PCollection.  
    | "Pegar voos atrasados" >> beam.ParDo(filtro())  
    #Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso.  
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))  
    # Soma todas as keys que são iguais, para um value unico  
    | "Somar por key" >> beam.CombinePerKey(sum)  
)
```



- ~~Utilizando Apache Beam no Google Colab~~
- Read Inputs
- Write Inputs
- `beam.Map/beam.FlatMap`
- `beam.Filter`
- `beam.Flatten`
- `beam.CombinePerKey`
- `beam.combiners.Count.PerKey()`
- `beam.CoGroupByKey`
- ParDo — Funções Customizadas


```
In [62]: Qtd_Atrasos = (  
    p1  
    | "Importar dados" >> beam.io.ReadFromText('/content/voos_sample.csv',  
                                                skip_header_lines = 1)  
    | "Separar por vírgula" >> beam.Map(lambda record: record.split(','))  
    #permite aplicar uma função personalizada (uma classe que herda de beam.DoFn)  
    #a cada elemento da PCollection.  
    | "Pegar voos atrasados" >> beam.ParDo(filtro())  
    #Criando uma Key e um Value, utilizando a coluna 4 (destino) e o atraso.  
    | "Criar conjunto par" >> beam.Map(lambda record: (record[4], int(record[8])))  
    # Soma todas as keys que são iguais, para um value unico  
    #Fornece a quantidade de atrasos por aeroporto  
    | "Somar por key" >> beam.combiners.Count.PerKey(sum)  
    )
```

```
In [63]: tabela_atrasos = (  
    #Criando tabela sendo dicionário.  
    #Essa estrutura pode ser útil em determinados casos,  
    #especialmente se você estiver agrupando ou processando dados  
    #que têm uma relação específica entre eles.  
    {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}  
    #É usada para agrupar esses dados por chave.  
    #útil quando precisamos realizar operações de agregação ou processamento  
    #em grupos específicos de dados.  
    | "Group by" >> beam.CoGroupByKey()  
    | "Mostrar dados" >> beam.Map(print)  
    )  
  
p1.run()
```

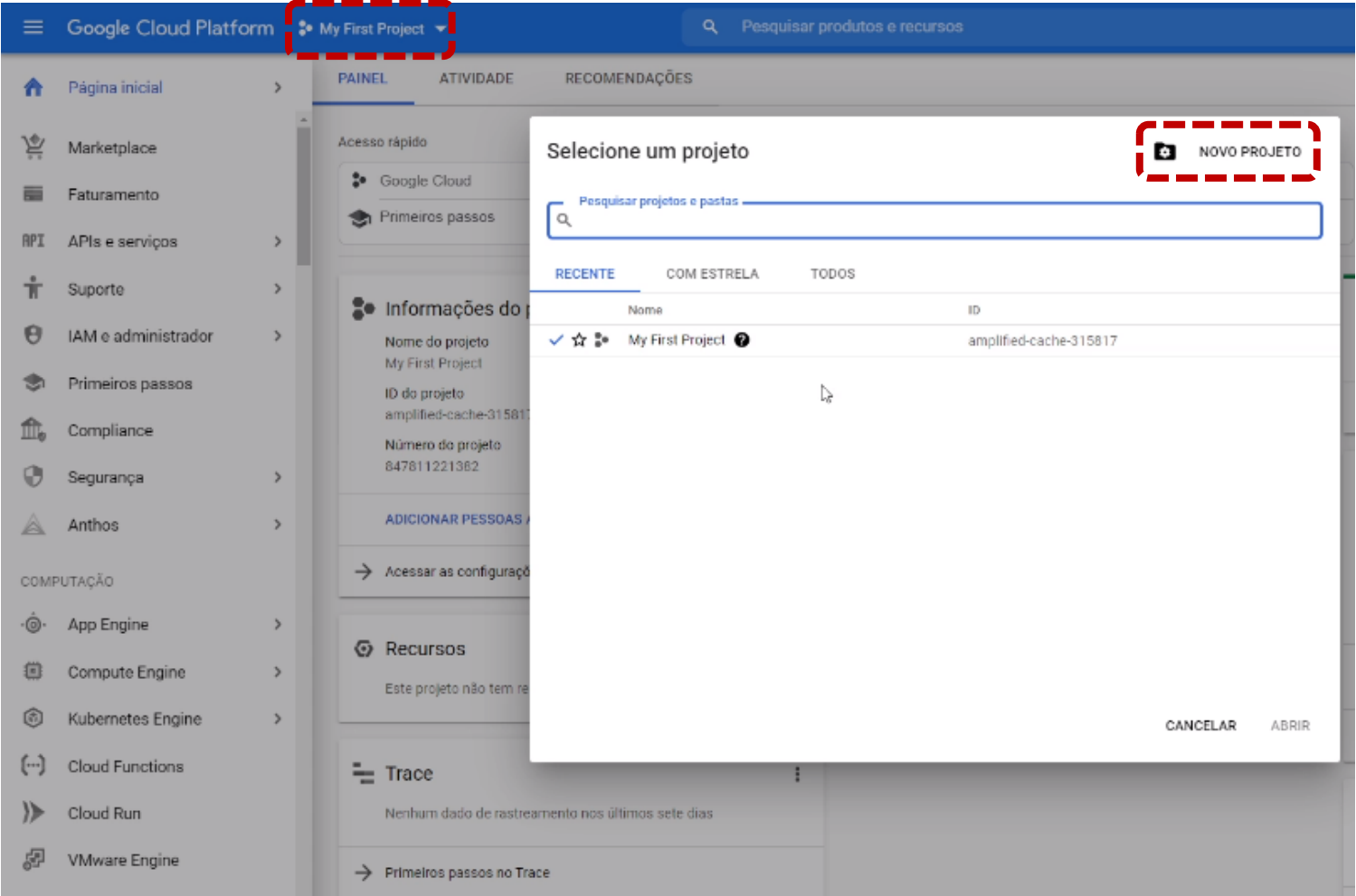
APACHE BEAM

GCP & DATAFLOW

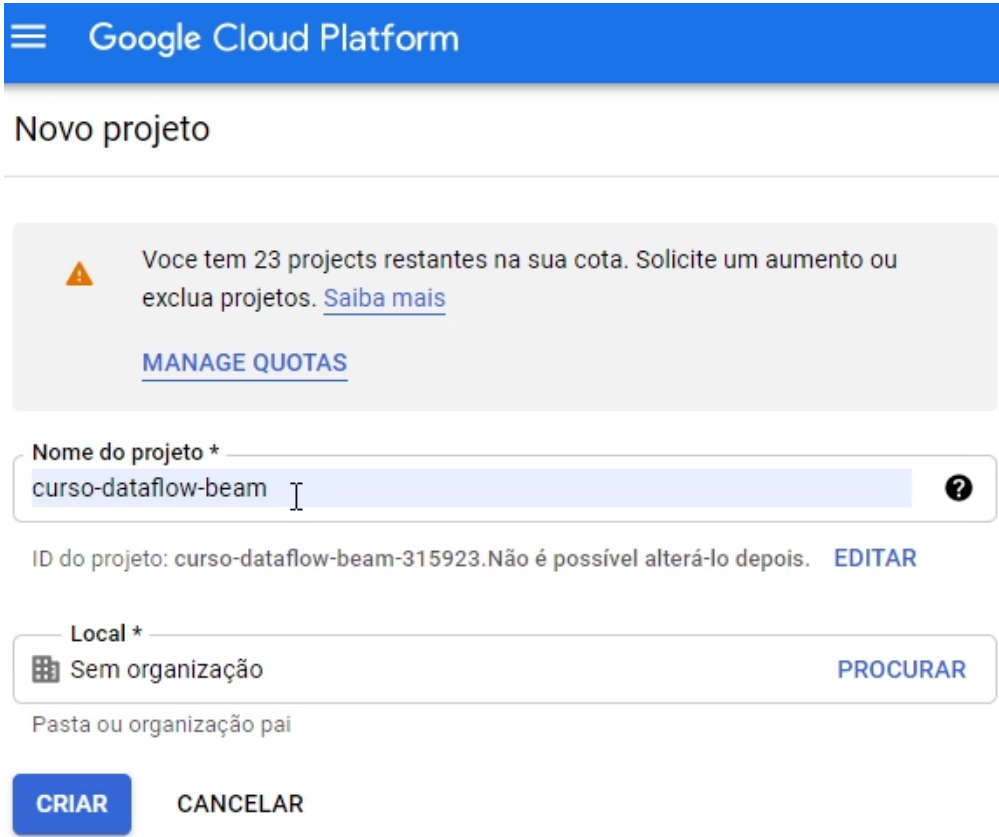
- 
- 
- Setup Ambiente GCP
 - Criando Service Account e Bucket
 - Setup Apache Beam Local (SDK)
 - Executando Direct Runner e Salvando na Nuvem GCP
 - Criando Template DataFlow
 - Executando Job Batch no DataFlow
 - ESTUDEM: Criando Template para gravar dados em BQ
 - ESTUDEM: Executando Job Batch para gravar dados no BQ

GCP

SETUP AMBIENTE GCP



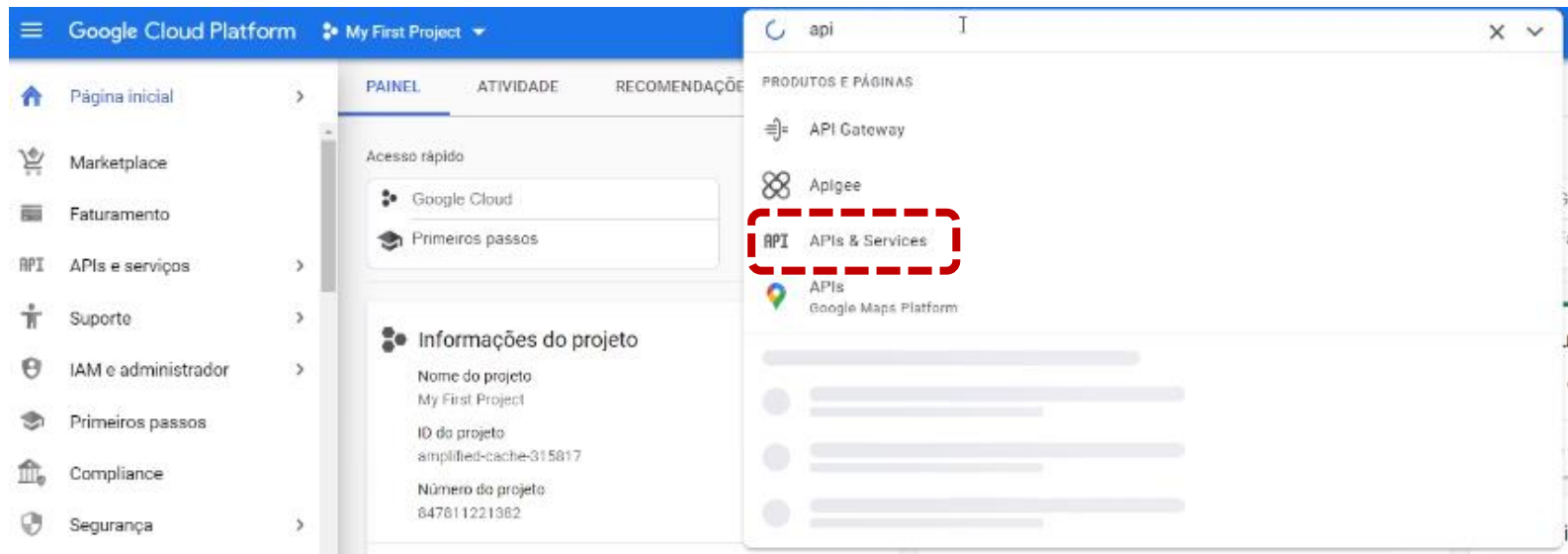
VAMOS INICIAR CRIANDO UM NOVO PROJETO, CLICAMOS NO ALTO DA TELA EM MYFIRST PROJECT, E DEPOIS EM NOVO PROJETO.



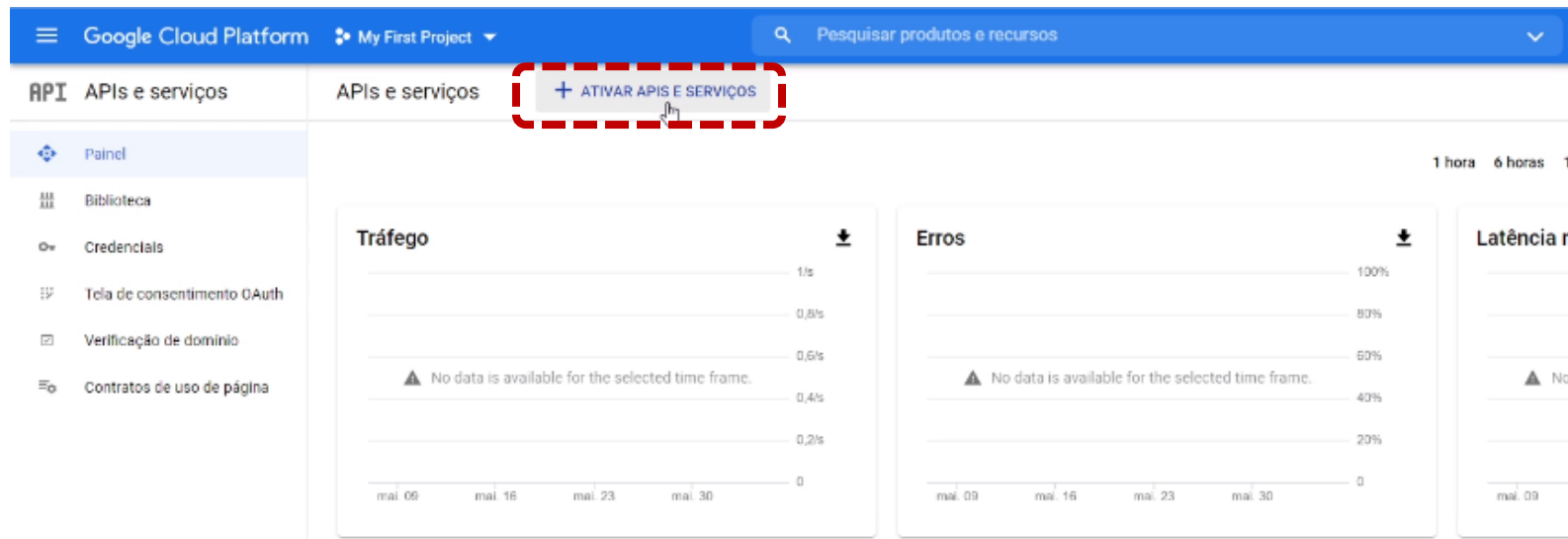
NOMENCLATURA DE SUA PREFERÊNCIA MAS SEM ESPAÇO. VAMOS UTILIZA-LÁ MAIS A FRENTE. APÓS ESSE PAÇO VAMOS HABILITAR APIS PARA COMUNICAÇÃO EXTERNA.

GCP

SETUP AMBIENTE GCP



PRECISAMOS ATIVAR O SERVIÇO DE APIS PARA CONSEGUIRMOS CONEXÃO COM AMBIENTES EXTERNOS AO GCP E PARA UTILIZAÇÃO DE FERRAMENTAS.




AS APIS A SEREM ATIVADAS:

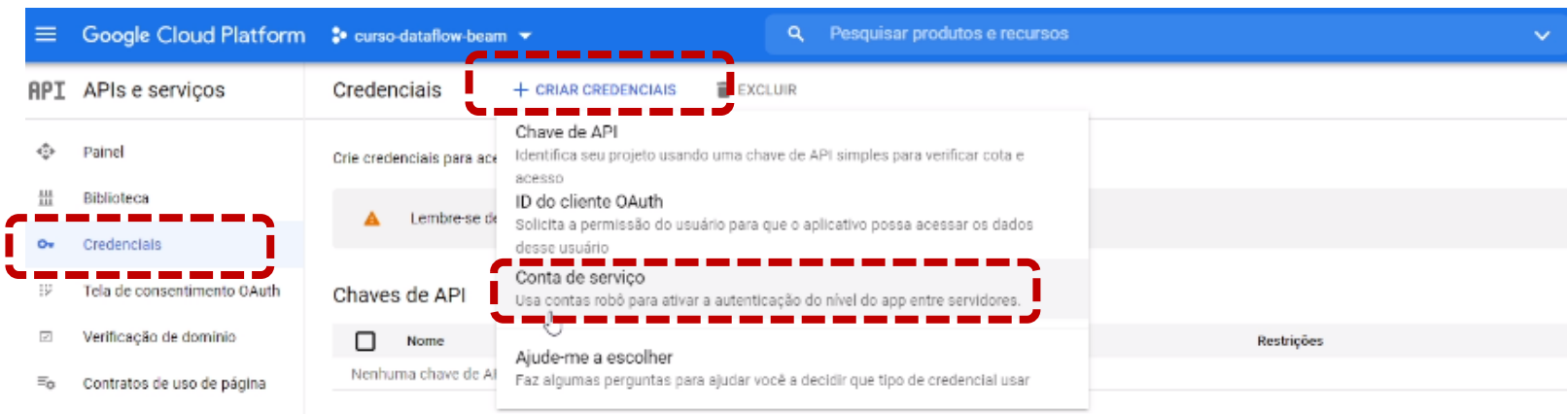
DATAFLOW
COMPUTE_COMPONENT
LOGGING
STORAGE_COMPONENT
STORAGE_API
BIGQUERY
PUBSUB
DATASTORE.GOOGLEAPIS.COM
CLOUDRESOURCEMANAGER.GOOGLEAPIS.COM

APACHE BEAM

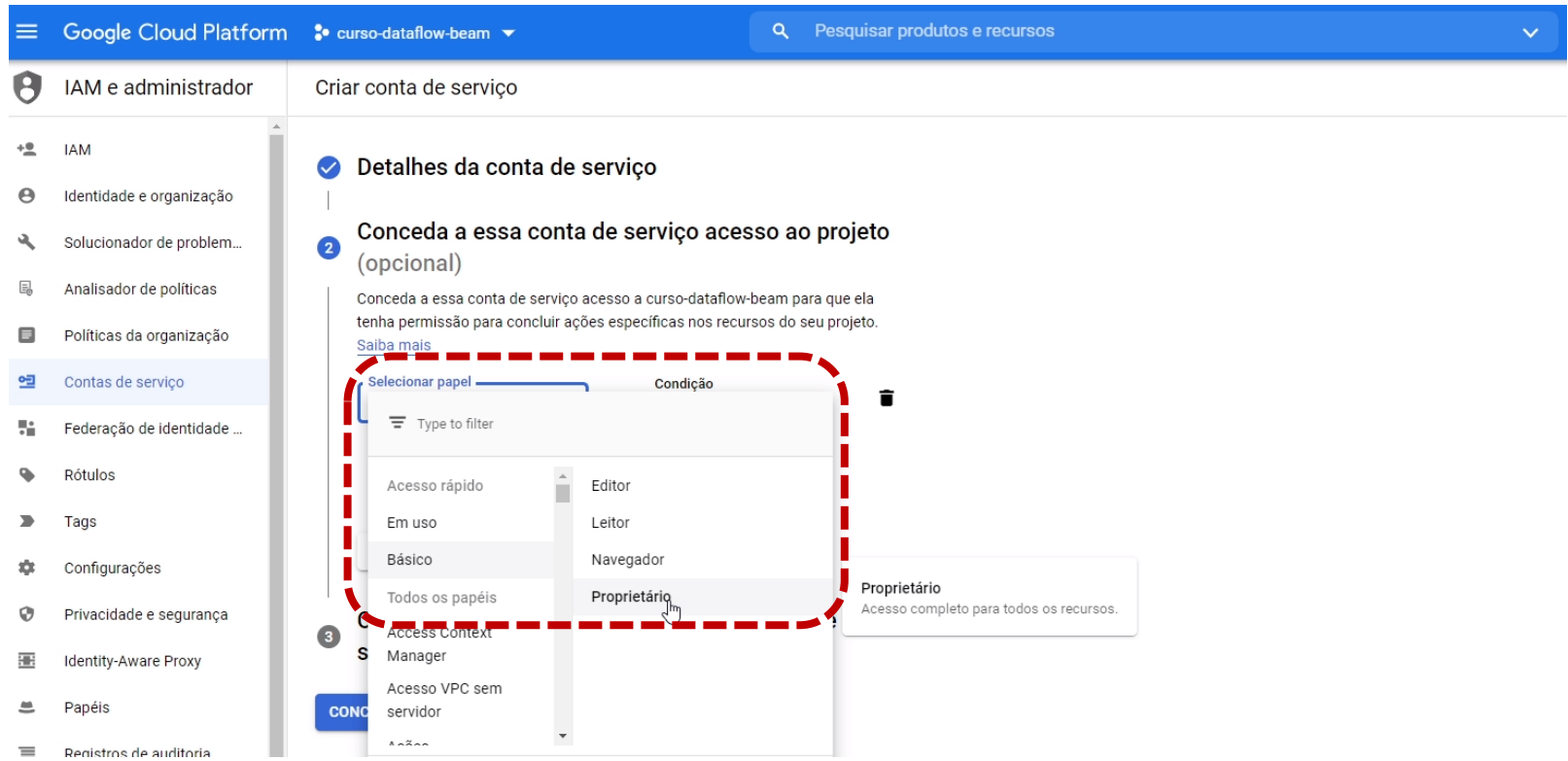
GCP & DATAFLOW

- 
- ~~Setup Ambiente GCP~~
 - Criando Service Account e Bucket
 - Setup Apache Beam Local (SDK)
 - Executando Direct Runner e Salvando na Nuvem GCP
 - Criando Template DataFlow
 - Executando Job Batch no DataFlow
 - ESTUDEM: Criando Template para gravar dados em BQ
 - ESTUDEM: Executando Job Batch para gravar dados no BQ

CRIANDO SERVICE ACCOUNT E BUCKET



A CONTA DE SERVIÇO É UM USUÁRIO ESPECIFICO QUE TERÁ AUTORIZAÇÕES E ACESSOS, DESSA FORMA SE MANTEM A SEGURANÇA POIS AO INVÉS DE ACESSOS GENERALISTAS, ELES FICAM RESTRITOS AS ATRIBUIÇÕES.



1 Detalhes da conta de serviço

Nome da conta de serviço

curso-apache-beam

Nome de exibição para esta conta de serviço

ID da conta de se...

curso-apache-beam @curso-dataflow-beam-315923.iam.gserviceacc

Descrição da conta de serviço

Conta criada para dar acesso local a GCP

Descreva como a conta de serviço será usada

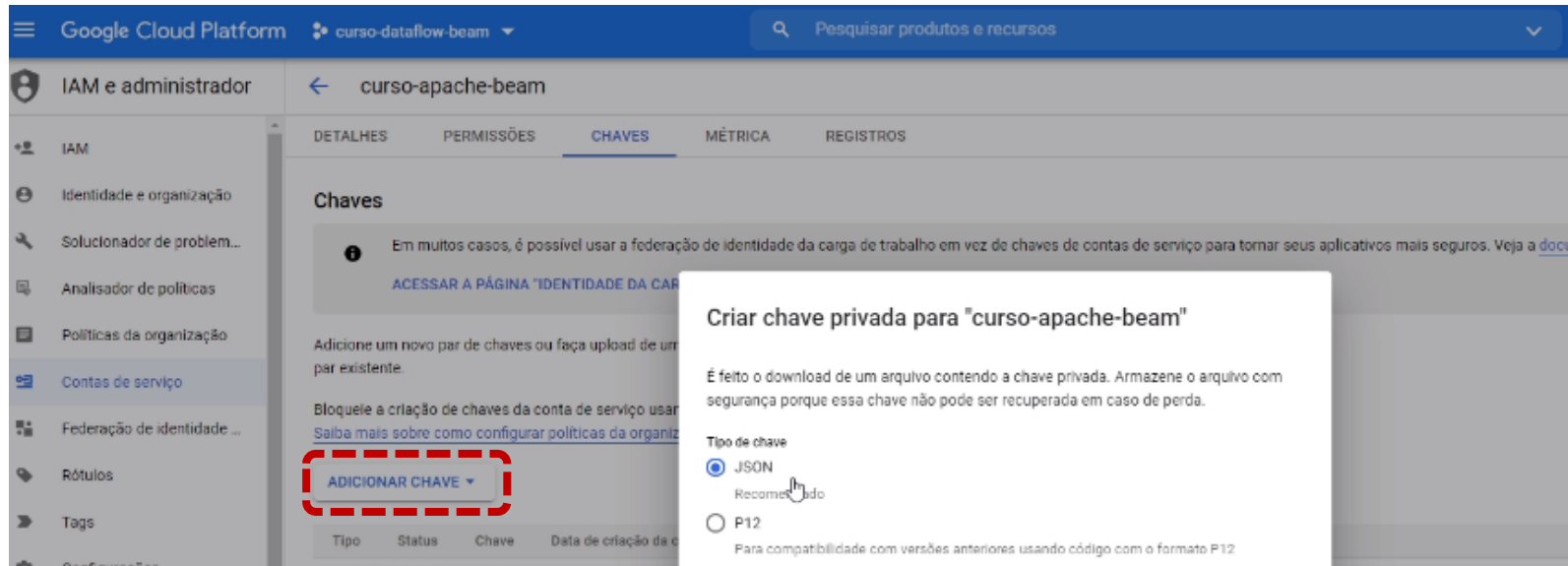
PRIMEIRO ÉE PRECISO NOMEAR A CONTA DE SERVIÇO. DEPOIS SELECIONAR O PAPEL QUE AQUELA CONTA TERÁ DE ATRIBUIÇÃO, SELECIONAREMOS BÁSICO E PROPRIETÁRIO.

DEPOIS APENAS SEGUIR COM CONCLUIR.

CRIANDO SERVICE ACCOUNT E BUCKET



AGORA PRECISAMOS PERMITIR QUE OS RECURSOS ATIVOS RECONHEÇAM A CONTA DE SERVIÇO CRIADA. E PARA ISSO VAMOS CRIAR UMA CHAVE.



A CHAVE SERÁ DO TIPO JSON. ENTÃO QUANDO REALIZARMOS EXECUÇÕES DENTRO DA FERRAMENTA GCP, PASSAREMOS O CAMINHO DA CHAVE JSON E OS RECURSOS ENTENDERÃO QUE TEMOS NIVEL DE ACESSO E AUTORIZAÇÃO PARA REALIZAR ATIVIDADES.

AO CLICAR EM CRIAR, O DOWNLOAD DE UM ARQUIVO SERÁ FEITO. E É SUA CHAVE.



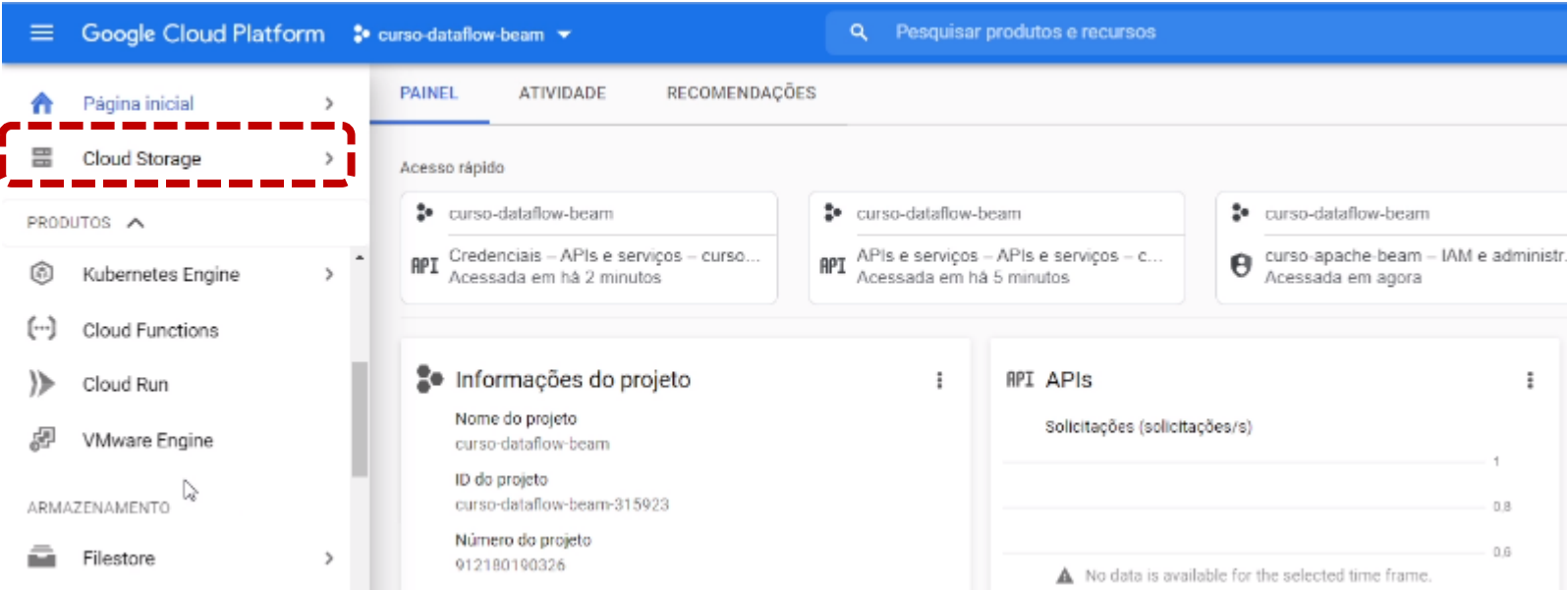
Google
Cloud Storage

O GOOGLE CLOUD STORAGE É UM SERVIÇO DE ARMAZENAMENTO EM NUVEM OFERECIDO PELO GOOGLE CLOUD PLATFORM (GCP). ELE PERMITE QUE OS USUÁRIOS ARMAZENEM E RECUPEREM DADOS DE MANEIRA ESCALÁVEL, SEGURA E DURÁVEL NA INFRAESTRUTURA GLOBAL DA GOOGLE.

EXISTEM DIFERENTES CLASSES DE ARMAZENAMENTO DISPONÍVEIS NO GOOGLE CLOUD STORAGE, COMO STANDARD, NEARLINE E COLDLINE, PERMITINDO OTIMIZAR OS CUSTOS COM BASE NOS REQUISITOS DE ACESSO AOS DADOS.

O GOOGLE CLOUD STORAGE É AMPLAMENTE UTILIZADO PARA ARMAZENAMENTO DE BACKUP, DISTRIBUIÇÃO DE CONTEÚDO, ARMAZENAMENTO DE DADOS PARA ANÁLISE, HOSPEDAGEM DE MÍDIA E MUITO MAIS.

CRIANDO SERVICE ACCOUNT E BUCKET



VAMOS ACESSAR O STORAGE NA TELA PRINCIPAL DO PAINEL DO GCP.



VAMOS CRIAR O NOVO BUCKET, ELE SOLICITARÁ UM NOME.



DEPOIS O LOCAL DE ARMAZENAMENTO, O IDEAL É COLOCA-LO NA REGIÃO MAIS PRÓXIMA AO SEU CENTRO DE USO, POR EXEMPLO NO SERVIDOR SÃO PAULO.

DEPOIS PODE PROSSEGUIR COM AS CONFIGURAÇÕES ORIGINAIS OFERECIDAS POIS ELAS SÃO OTIMIZADAS PARA GASTAR O MENOR RECURSO POSSIVEL.

UM BUCKET FUNCIONA COMO UMA PASTA PARA ARMAZENAR ARQUIVOS.

APACHE BEAM

GCP & DATAFLOW

- 
- 
- ~~Setup Ambiente GCP~~
 - ~~Criando Service Account e Bucket~~
 - Setup Apache Beam Local (SDK)
 - Executando Direct Runner e Salvando na Nuvem GCP
 - Criando Template DataFlow
 - Executando Job Batch no DataFlow
 - ESTUDEM: Criando Template para gravar dados em BQ
 - ESTUDEM: Executando Job Batch para gravar dados no BQ



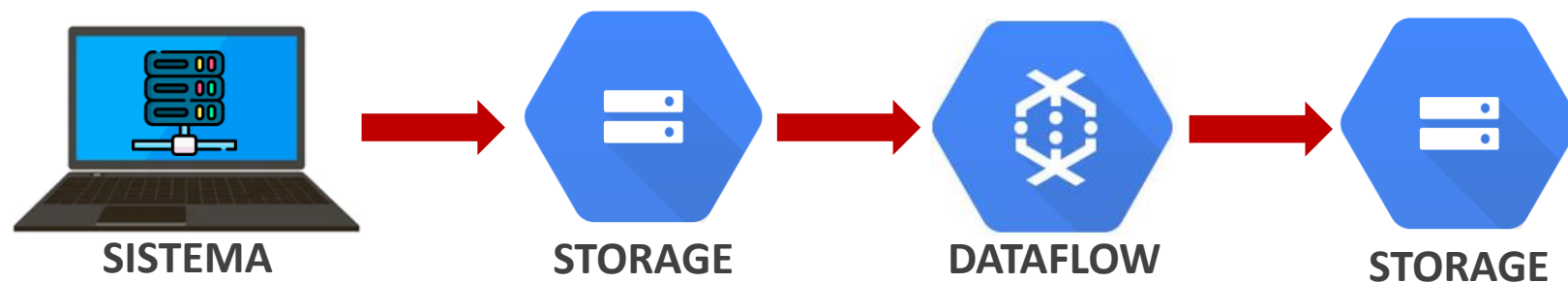
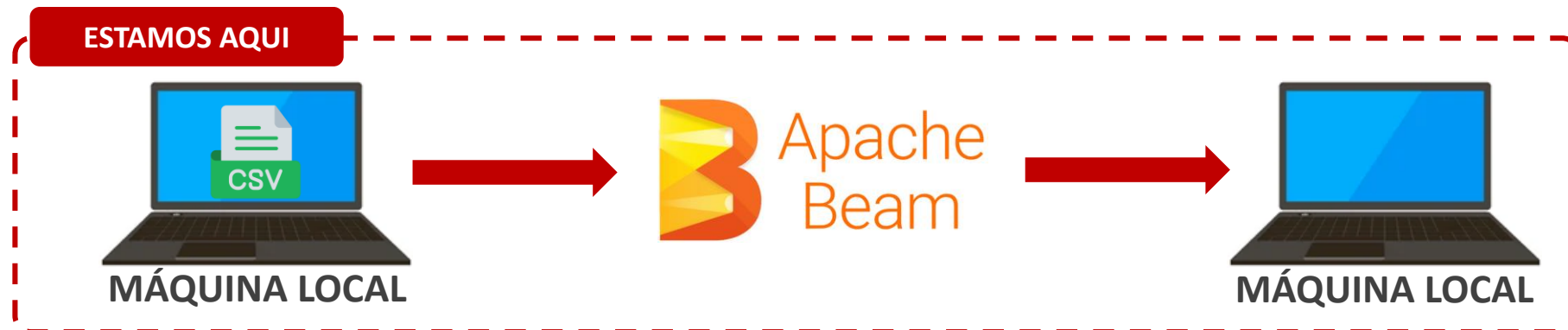
AGORA VAMOS FAZER COM QUE NOSSOS DADOS LOCAIS SEJAM TRANSPOSTOS PARA O AMBIENTE CLOUD.

COM PYTHON JÁ RODANDO NO VSCODE, VAMOS INSTALAR O APACHE BEAM:

PIP INSTALL APACHE-BEAM

PIP INSTALL APACHE-BEAM[GCP]

USAREMOS ESSAS DUAS BIBLIOTECAS POIS UMA É O APACHE-BEAM PURO E OUTRA PARA A CLOUD ESPECIFICA, SE ESTIVESSEMOS EM OUTRA CLOUD APENAS SUBSTUÍRIAMOS O NOME DELA.



SETUP APACHE BEAM LOCAL (SDK)

3.10- ParDo.py X

C:\Users\cassi\Google Drive\GCP\Dataflow Course\Meu_Curso\Seção 3 - Principais Transformações\3.10- ParDo.py >

```

1  import apache_beam as beam
2
3  p1 = beam.Pipeline()
4
5  class filtro(beam.DoFn):
6      def process(self, record):
7          if int(record[8]) > 0:
8              return [record]
9
10 Tempo_Atrasos = (
11     p1
12     | "Importar Dados Atraso" >> beam.io.ReadFromText("voos_sample.csv", skip_header_lines = 1)
13     | "Separar por Vírgulas Atraso" >> beam.Map(lambda record: record.split(','))
14     | "Pegar voos com atraso" >> beam.ParDo(filtro())
15     | "Criar par atraso" >> beam.Map(lambda record: (record[4], int(record[8])))
16     | "Somar por key" >> beam.CombinePerKey(sum)
17     # | "Mostrar Resultados" >> beam.Map(print)
18 )
19
20 Qtd_Atrasos = (
21     p1
22     | "Importar Dados" >> beam.io.ReadFromText("voos_sample.csv", skip_header_lines = 1)
23     | "Separar por Vírgulas Qtd" >> beam.Map(lambda record: record.split(','))
24     | "Pegar voos com Qtd" >> beam.ParDo(filtro())
25     | "Criar par Qtd" >> beam.Map(lambda record: (record[4], int(record[8])))
26     | "Contar por key" >> beam.combiners.Count.PerKey()
27     # | "Mostrar Resultados QTD" >> beam.Map(print)
28 )
29
30 tabela_atrasos = (
31     {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
32     | "Group By" >> beam.CoGroupByKey()
33     | beam.Map(print)
34 )
35
36 p1.run()

```

NO VSCODE, CASO QUEIRA EXECUTAR O ÚLTIMO CÓDIGO FEITO NO COLAB, NOS DEPARARIAMOS COM ERRO, POIS O LOCAL DE CONSUMO DE DADOS NÃO ESTÁ APONTADO CORRETAMENTE EM NOSSA MÁQUINA LOCAL.

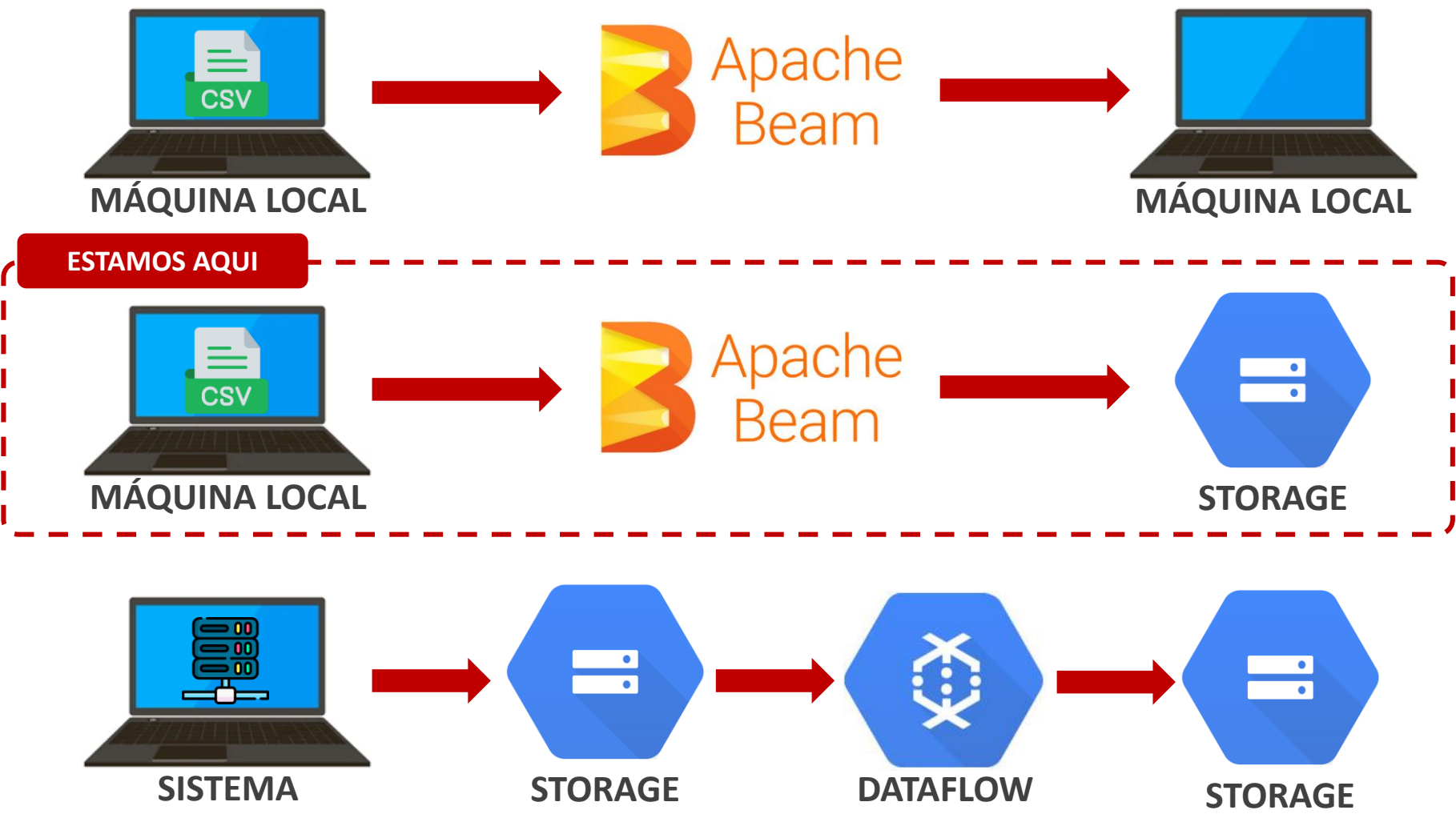
ENTÃO VAMOS SUBSTITUIR PELO CAMINHO DE ONDE O ARQUIVO ESTA EM NOSSA MÁQUINA LOCAL.

CASO O CAMINHO FIQUE “COLORIDO” COM ERRO, ADICIONE DUAS BARRAS //.

E TAMBÉM ADICIONE R ANTES DE ‘CAMINHO’ PARA QUE A EXECUÇÃO SEJA COMO RAW TEXT. ASSIM ELE ENTENDE QUE TODO O CONTEÚDO DEVE SER TRATADO COMO STRING.

```
r'C:\\Users\\laybe\\OneDrive\\Area de Trabalho\\apache-beam\\dados\\voos_sample.csv'
```

DEVEMOS VISUALIZAR AO FINAL A TABELA NO TERMINAL DO VSCODE.

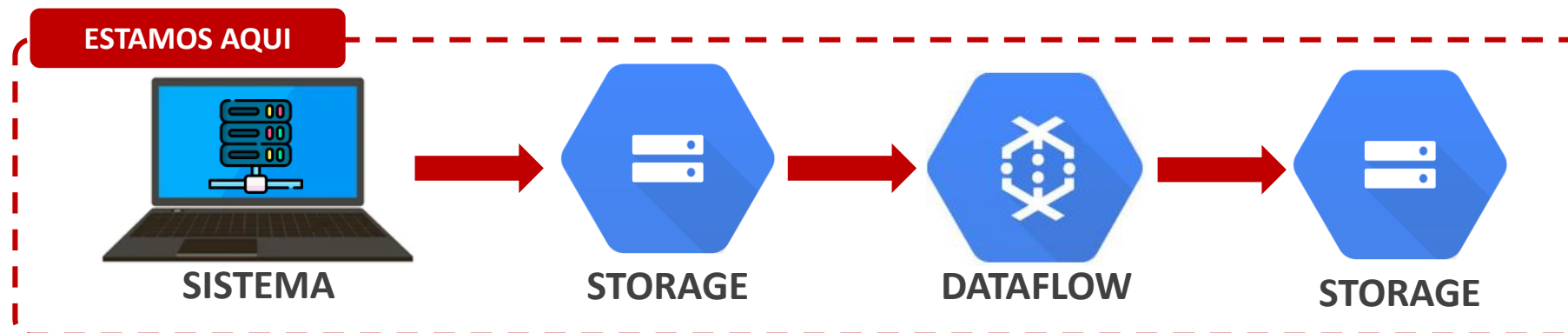


EXECUTANDO DIRECT RUNNER E SALVANDO NA NUVEM GCP

```
voos_batch_chave_serviceAccount.py X
C: > Users > lfbs3 > Downloads > voos_batch_chave_serviceAccount.py
1  import apache_beam as beam
2  import os
3
4  serviceAccount = r'C:\Users\cassi\Google Drive\GCP\Dataflow Course\Meu_Curso\curso-dataflow-beam-315923-4d903d955091.json'
5  os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = serviceAccount
6
7  p1 = beam.Pipeline()
8
9  class filtro(beam.DoFn):
10     def process(self, record):
11         if int(record[8]) > 0:
12             return [record]
13
14  Tempo_Atrasos = (
15     p1
16     | "Importar Dados Atraso" >> beam.io.ReadFromText(r'C:\\Users\\Laybe\\OneDrive\\Área de Trabalho\\apache-beam\\dados\\voos_sample.csv', skip_header_lines = 1)
17     | "Separar por Vírgulas Atraso" >> beam.Map(lambda record: record.split(','))
18     | "Pegar voos com atraso" >> beam.ParDo(filtro())
19     | "Criar par atraso" >> beam.Map(lambda record: (record[4], int(record[8])))
20     | "Somar por key" >> beam.CombinePerKey(sum)
21 )
22
23  Qtd_Atrasos = (
24     p1
25     | "Importar Dados" >> beam.io.ReadFromText(r'C:\\Users\\Laybe\\OneDrive\\Área de Trabalho\\apache-beam\\dados\\voos_sample.csv', skip_header_lines = 1)
26     | "Separar por Vírgulas Qtd" >> beam.Map(lambda record: record.split(','))
27     | "Pegar voos com Qtd" >> beam.ParDo(filtro())
28     | "Criar par Qtd" >> beam.Map(lambda record: (record[4], int(record[8])))
29     | "Contar por key" >> beam.combiners.Count.PerKey()
30 )
31
32  tabela_atrasos = (
33     {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
34     | "Group By" >> beam.CoGroupByKey()
35     | "Saida Para GCP" >> beam.io.WriteToText(r"gs://curso-apache-beam/Voos_atrados_qtd.csv")
36 )
37
38  p1.run()
```

VAMOS CRIAR UMA VARIÁVEL PARA ARMAZENAR A CHAVE JSON ATRAVÉS DE SEU CAMINHO.

VAMOS USAR A BIBLIOTECA OS, QUE SERVE PARA INTERAÇÃO ENTRE SISTEMA OPERACIONAIS SUBJACENTES E VAMOS REFERENCIAR A CHAVE PARA QUE FAÇA A CONEXÃO CORRETAMENTE, AO FINAL DO CÓDIGO A SAIDA SERÁ NO BUCKET ATRAVÉS DO CAMINHO ONDE QUEREMOS SALVAR O DATAFRAME, gs://NOME-QUE-DEMOS.





Google
DataFlow

O GOOGLE CLOUD DATAFLOW É UM SERVIÇO TOTALMENTE GERENCIADO PARA PROCESSAMENTO DE DADOS EM TEMPO REAL E EM LOTE. ELE É PROJETADO PARA FACILITAR A CRIAÇÃO, EXECUÇÃO E MONITORAMENTO DE PIPELINES DE DADOS ESCALÁVEIS E EFICIENTES.

O GOOGLE CLOUD DATAFLOW SUPORTA TANTO O PROCESSAMENTO EM LOTE (BATCH) QUANTO O PROCESSAMENTO EM TEMPO REAL (STREAMING).

AGORA O STORAGE RECEBERÁ NOSSO ARQUIVO DE DADOS DE ENTRADA, TERÁ UMA ROTINA DE ATRIBUIÇÕES EXECUTADO PELO DATAFLOW E SUA SAIDA NOVAMENTE NO STORAGE.

ENTRADA: SOURCE

SAIDA: TARGET

CRIANDO TEMPLATE DATAFLOW

```

voos_batch_dataflow.py X
C:\Users> ifbs3 > Downloads > voos_batch_dataflow.py
1 import apache_beam as beam
2 import os
3 from apache_beam.options.pipeline_options import PipelineOptions
4
5 pipeline_options = {
6     'project': 'curso-dataflow-beam-315923',
7     'runner': 'DataflowRunner',
8     'region': 'southamerica-east1',
9     'staging_location': 'gs://curso-apache-beam/temp',
10    'temp_location': 'gs://curso-apache-beam/temp',
11    'template_location': 'gs://curso-apache-beam/template/batch_job_df_gcs_voos' }
12
13 pipeline_options = PipelineOptions.from_dictionary(pipeline_options)
14 p1 = beam.Pipeline(options=pipeline_options)
15
16 serviceAccount = r'C:\Users\cassi\Google Drive\GCP\Dataflow Course\Meu_Curso\curso-dataflow-beam-315923-4d903d955091.json'
17 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = serviceAccount
18
19 class filtro(beam.DoFn):
20     def process(self, record):
21         if int(record[8]) > 0:
22             return [record]
23
24 Tempo_Atrasos = (
25     p1
26     | "Importar Dados Atraso" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
27     | "Separar por Vírgulas Atraso" >> beam.Map(Lambda record: record.split(','))
28     | "Pegar voos com atraso" >> beam.ParDo(filtro())
29     | "Criar par atraso" >> beam.Map(Lambda record: (record[4], int(record[8])))
30     | "Somar por key" >> beam.CombinePerKey(sum)
31 )
32
33 Qtd_Atrasos = (
34     p1
35     | "Importar Dados" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
36     | "Separar por Vírgulas Qtd" >> beam.Map(Lambda record: record.split(','))
37     | "Pegar voos com Qtd" >> beam.ParDo(filtro())
38     | "Criar par Qtd" >> beam.Map(Lambda record: (record[4], int(record[8])))
39     | "Contar por key" >> beam.combiners.Count.PerKey()
40 )
41
42 tabela_atrasos = (
43     {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
44     | beam.CoGroupByKey()
45     | beam.io.WriteToText(r"gs://curso-apache-beam/saida/Voos_atrados_qtd.csv")
46 )
47
48 p1.run()

```

VAMOS DECLARAR A VARIÁVEL PIPELINE_OPTIONS, PARA ATRAVÉS DELA TRANSCREVERMOS AS CONFIGURAÇÕES DE COMO SERÁ O NOSSO PIPELINE. ESSAS INFORMAÇÕES DESCRITAS NO NOSSO CÓDIGO SÃO BÁSICAS E ATRAVÉS DELAS O FLUXO DE DADOS JÁ É FUNCIONAL.

O ID DO PROJETO ESTÁ NO PAINEL INICIAL:

The screenshot shows the Google Cloud Platform console interface. At the top, there's a navigation bar with 'Google Cloud Platform' and a dropdown menu showing 'curso-dataflow-beam'. Below this, there's a 'PAINEL' (Dashboard) tab selected. The dashboard displays 'Acesso rápido' (Quick access) with three cards: 'curso-dataflow-beam', 'Navegador - Dataflow - curso-dataflow-beam', and 'Navegador - Cloud Storage - curso-d...'. Below these, there's a section titled 'Informações do projeto' (Project information) which is highlighted with a red dashed box. This section shows the project name 'curso-dataflow-beam', the project ID 'curso-dataflow-beam-315923' (highlighted with a blue box), and the project number '912180190326'. To the right of this section, there's a graph titled 'API APIs' showing 'Solicitações (solicitações/s)' (Requests (requests/s)) over time, with a peak around 9:45.

CRIANDO TEMPLATE DATAFLOW

```

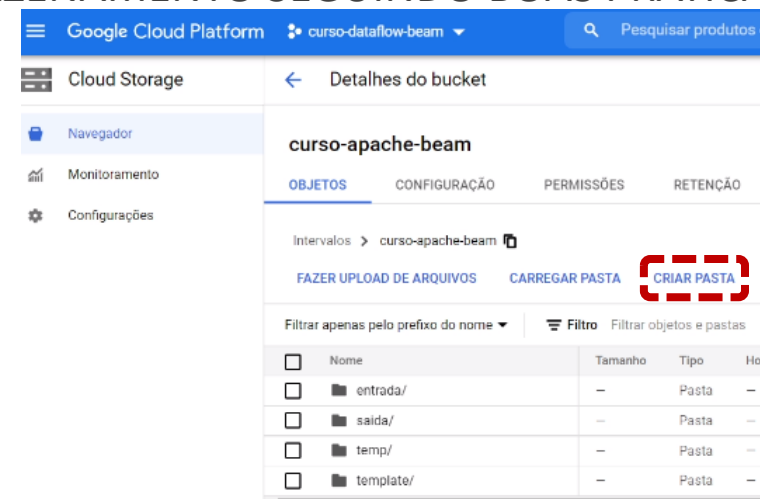
voos_batch_dataflow.py X
C:\Users\lfbs3 > Downloads > voos_batch_dataflow.py
1 import apache_beam as beam
2 import os
3 from apache_beam.options.pipeline_options import PipelineOptions
4
5 pipeline_options = {
6     'project': 'curso-dataflow-beam-315923',
7     'runner': 'DataflowRunner',
8     'region': 'southamerica-east1',
9     'staging_location': 'gs://curso-apache-beam/temp',
10    'temp_location': 'gs://curso-apache-beam/temp',
11    'template_location': 'gs://curso-apache-beam/template/batch_job_df_gcs_voos' }
12
13 pipeline_options = PipelineOptions.from_dictionary(pipeline_options)
14 p1 = beam.Pipeline(options=pipeline_options)
15
16 serviceAccount = r'C:\Users\cassi\Google Drive\GCP\Dataflow Course\Meu_Curso\curso-dataflow-beam-315923-4d903d955091.json'
17 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = serviceAccount
18
19 class filtro(beam.DoFn):
20     def process(self, record):
21         if int(record[8]) > 0:
22             return [record]
23
24 Tempo_Atrasos = (
25     p1
26     | "Importar Dados Atraso" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
27     | "Separar por Vírgulas Atraso" >> beam.Map(lambda record: record.split(','))
28     | "Pegar voos com atraso" >> beam.ParDo(filtro())
29     | "Criar par atraso" >> beam.Map(lambda record: (record[4], int(record[8])))
30     | "Somar por key" >> beam.CombinePerKey(sum)
31 )
32
33 Qtd_Atrasos = (
34     p1
35     | "Importar Dados" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
36     | "Separar por Vírgulas Qtd" >> beam.Map(lambda record: record.split(','))
37     | "Pegar voos com Qtd" >> beam.ParDo(filtro())
38     | "Criar par Qtd" >> beam.Map(lambda record: (record[4], int(record[8])))
39     | "Contar por key" >> beam.combiners.Count.PerKey()
40 )
41
42 tabela_atrasos = (
43     {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
44     | beam.CoGroupByKey()
45     | beam.io.WriteToText(r"gs://curso-apache-beam/saida/Voos_atrados_qtd.csv")
46 )
47
48 p1.run()

```

STAGING_LOCATION É NOSSA ÁREA DE STAGING, VAMOS CRIAR A PASTE TEMPO NO BUCKET PARA O ARMAZENADOR.

TEMPLATE_LOCATION: ARMAZENARÁ O TEMPLATE OU ROTINA DE EXECUÇÃO PARA O DATAFLOW, TAMBÉM IREMOS CRIAR O REPOSITÓRIO NO BUCKET.

TAMBÉM TEREMOS QUE CRIAR O DIRETÓRIO DE ENTRADA E SAÍDA, PARA QUE EXISTA LOCAIS DE ARMAZENAMENTO SEGUINDO BOAS PRÁTICAS.



CRIANDO TEMPLATE DATAFLOW

```

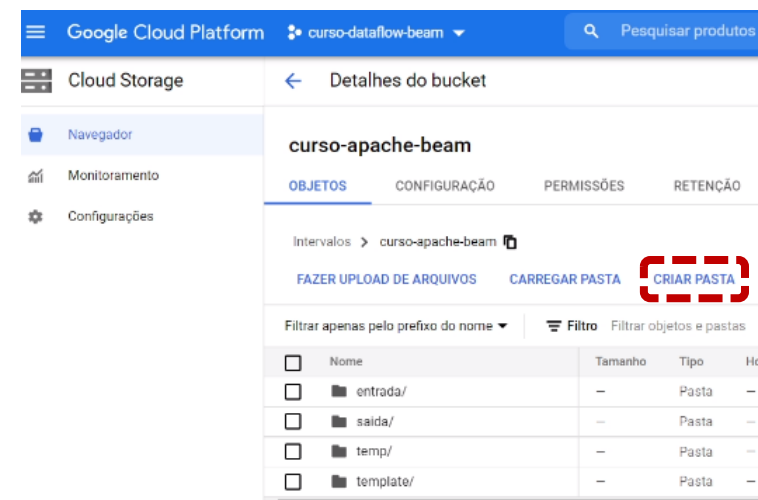
voos_batch_dataflow.py X
C:\Users> ifbs3 > Downloads > voos_batch_dataflow.py
1 import apache_beam as beam
2 import os
3 from apache_beam.options.pipeline_options import PipelineOptions
4
5 pipeline_options = {
6     'project': 'curso-dataflow-beam-315923',
7     'runner': 'DataflowRunner',
8     'region': 'southamerica-east1',
9     'staging_location': 'gs://curso-apache-beam/temp',
10    'temp_location': 'gs://curso-apache-beam/temp',
11    'template_location': 'gs://curso-apache-beam/template/batch_job_df_gcs_voos' }
12
13 pipeline_options = PipelineOptions.from_dictionary(pipeline_options)
14 p1 = beam.Pipeline(options=pipeline_options)
15
16 serviceAccount = r'C:\Users\cassi\Google Drive\GCP\Dataflow Course\Meu_Curso\curso-dataflow-beam-315923-4d903d955091.json'
17 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = serviceAccount
18
19 class filtro(beam.DoFn):
20     def process(self, record):
21         if int(record[8]) > 0:
22             return [record]
23
24 Tempo_Atrasos = (
25     p1
26     | "Importar Dados Atraso" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
27     | "Separar por Vírgulas Atraso" >> beam.Map(lambda record: record.split(','))
28     | "Pegar voos com atraso" >> beam.ParDo(filtro())
29     | "Criar par atraso" >> beam.Map(lambda record: (record[4], int(record[8])))
30     | "Somar por key" >> beam.CombinePerKey(sum)
31 )
32
33 Qtd_Atrasos = (
34     p1
35     | "Importar Dados" >> beam.io.ReadFromText(r"gs://curso-apache-beam/entrada/voos_sample.csv", skip_header_lines = 1)
36     | "Separar por Vírgulas Qtd" >> beam.Map(lambda record: record.split(','))
37     | "Pegar voos com Qtd" >> beam.ParDo(filtro())
38     | "Criar par Qtd" >> beam.Map(lambda record: (record[4], int(record[8])))
39     | "Contar por key" >> beam.combiners.Count.PerKey()
40 )
41
42 tabela_atrasos = (
43     {'Qtd_Atrasos': Qtd_Atrasos, 'Tempo_Atrasos': Tempo_Atrasos}
44     | beam.CoGroupByKey()
45     | beam.io.WriteToText(r"gs://curso-apache-beam/saida/Voos_atrados_qtd.csv")
46 )
47
48 p1.run()

```

PODEMOS DAR QUALQUER NOME AO ARQUIVO QUE SERÁ ARMAZENADO NO TEMPLATE_LOCATION.

A PASTA ENTRADA PRECISA TER UM ARQUIVO DE REFERÊNCIA, PODEMOS APENAS FAZER O UPLOAD DO CSV VOOS_SAMPLE.

PODEMOS DAR QUALQUER NOME AO ARQUIVO QUE SERÁ ARMAZENADO NA PASTA DE SAIDA.

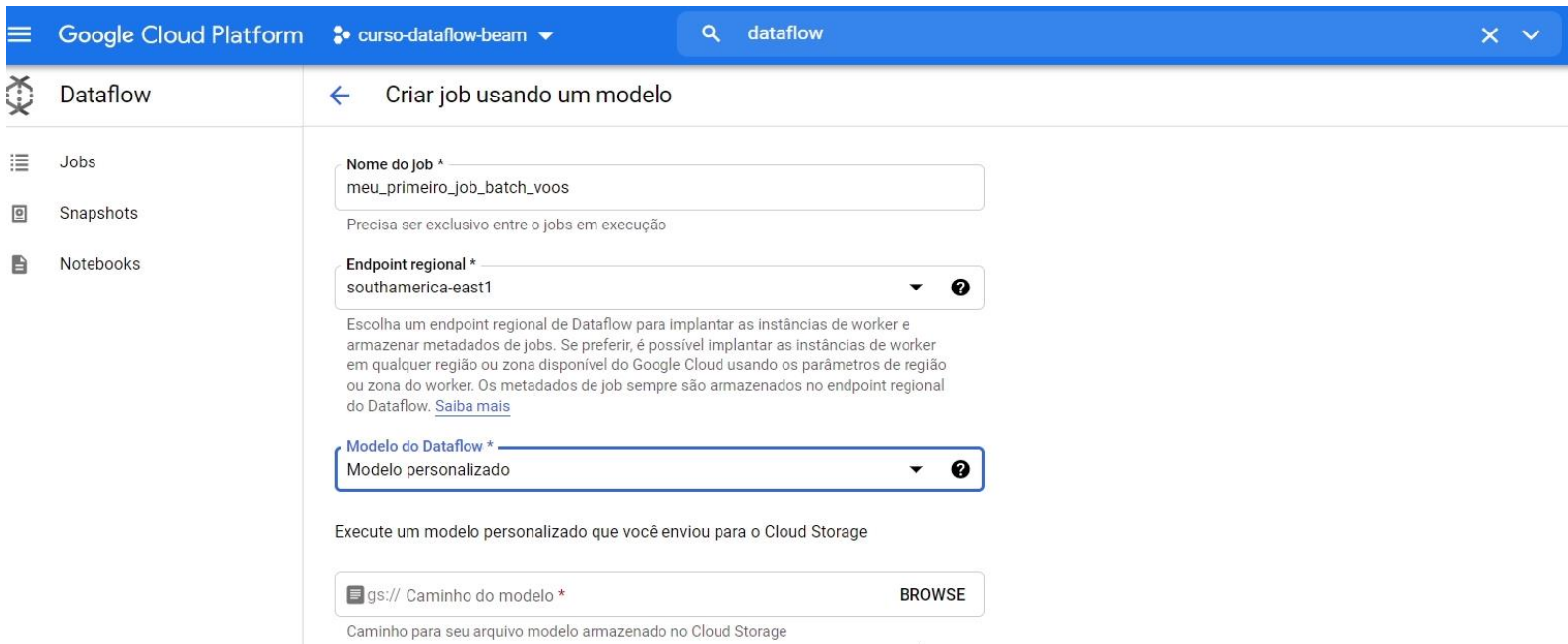
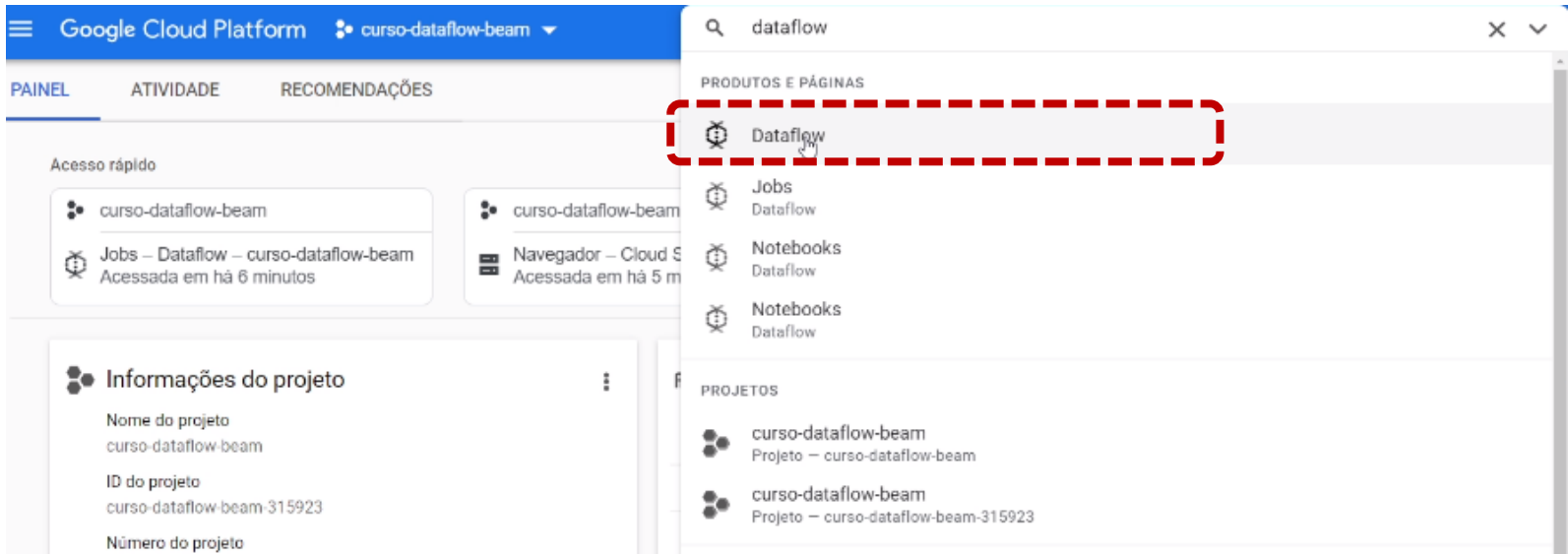


APACHE BEAM

GCP & DATAFLOW

- ~~Setup Ambiente GCP~~
- ~~Criando Service Account e Bucket~~
- ~~Setup Apache Beam Local (SDK)~~
- ~~Executando Direct Runner e Salvando na Nuvem GCP~~
- ~~Criando Template DataFlow~~
- Executando Job Batch no DataFlow
- ESTUDEM: Criando Template para gravar dados em BQ
- ESTUDEM: Executando Job Batch para gravar dados no BQ

EXECUTANDO JOB BATCH NO DATAFLOW

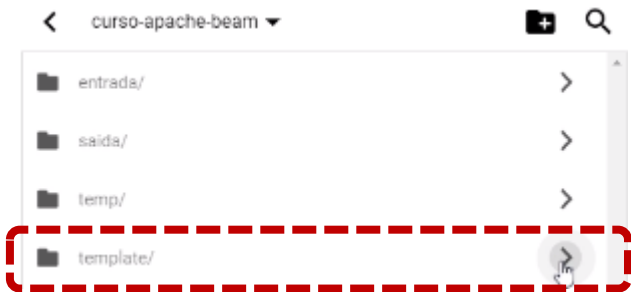


AGORA VAMOS CRIAR A ROTINA DE EXECUÇÕES NO DATAFLOW.

VAMOS CRIAR UM NOVO JOB USANDO MODELO, HÁ TAMBÉM A OPÇÃO DE CRIAR UM JOB USANDO SQL MAS NESSE CASO IRIAMOS UTILIZAR CASO OS DADOS ESTIVESSEM NO BANCO BIGQUERY DO GCP.

VAMOS NOMEAR O JOB DE FORMA QUE SABEMOS IDENTIFICAR SOBRE O QUE ELE TRATA E SELECIONAR A ZONA MAIS PRÓXIMA PARA EXECUÇÃO.

EM MODELO DE DATAFLOW HÁ INÚMERAS OPÇÕES PRÉ-CONFIGURADAS, MAS VAMOS CONFIGURAR UMA NOVA NOSSA ATRAVÉS DA PERSONALIZAÇÃO. DEPOIS VAMOS LOCALIZAR O TEMPLATE GERADO ATRAVÉS DE BROWSE:



DEPOIS PASSAREMOS O LOCAL TEMPORÁRIO, QUE É NOSSO TEMP_LOCATION: 'GS://CURSO-APACHE-BEAM/TEMP'. EXECUTE.

GCP

EXECUTANDO JOB BATCH NO DATAFLOW

Dataflow

Jobs

Snapshots

Notebooks

Google Cloud Platform

curso-dataflow-beam

dataflow

meu_primeiro...

CLONAR

INTERROMPER

REGISTROS

COMPARTILHAR

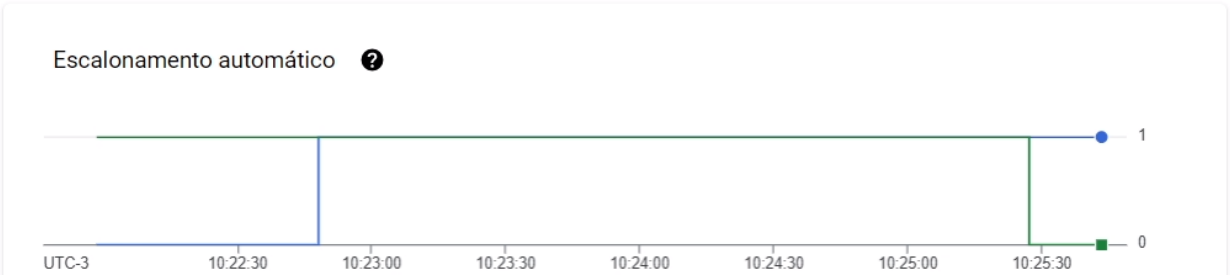
HORÁRIO MÁX

GRÁFICO DE JOB

DETALHES DA EXECUÇÃO

MÉTRICAS DO JOB

Escalonamento automático



Registros

REGISTROS DO JOB

REGISTROS DE WORKER

DIAGNÓSTICOS

Mostrando 49 mensagens

Gravidade

Informações

Filtro

Filtrar registros

2021-06-06T13:25:23.976921681Z

Finished operation WriteToText/Write/WriteImpl/PreFinalize/_UnpickledSideInput(Extr...

2021-06-06T13:25:24.013855436Z

Executing operation WriteToText/Write/WriteImpl/PreFinalize/PreFinalize

2021-06-06T13:25:25.488773206Z

Finished operation WriteToText/Write/WriteImpl/PreFinalize/PreFinalize

2021-06-06T13:25:25.525830586Z

Executing operation WriteToText/Write/WriteImpl/FinalizeWrite/_UnpickledSideText(0

2021-06-06T13:25:25.548113686Z

Finished operation WriteToText/Write/WriteImpl/FinalizeWrite/_Unpic

2021-06-06T13:25:25.585556428Z

Executing operation WriteToText/Write/WriteImpl/FinalizeWrite/Final

2021-06-06T13:25:26.960361617Z

Finished operation WriteToText/Write/WriteImpl/FinalizeWrite/Finali

2021-06-06T13:25:27.036659647Z

Stopping worker pool...

Informações do job

Tipo de job

Lote

Status do job

Em execução

Versão do SDK

Apache Beam Python 3.8 SDK 2.28.0

Região do job

southamerica-east1

Local do worker

southamerica-east1-c

Quantidade atual de workers

1

Status mais recente do worker

Stopping worker pool.

Horário de início

6 de junho de 2021 10:21:46 GMT-3

Tempo decorrido

4 min 2 s

Tipo de criptografia

Chave gerenciada pelo Google

Métricas de recursos

vCPUs atuais

1

Tempo total de vCPU

0,045 vCPU hr

Tempo total de PD HDD

1,129 GB hr

Notas da versão

O tempo total de execução de todas as vCPUs associadas ao seu job em vCPU-horas. Por exemplo, se os workers do job executaram 3 vCPUs por 4 horas cada, o tempo total da vCPU será igual a 12 vCPU-horas.

GCP

EXECUTANDO JOB BATCH NO DATAFLOW

Google Cloud Platform

curso-dataflow-beam

Pesquisar produtos e recursos

5

Cloud Storage

Navegador

Monitoramento

Configurações

Detalhes do bucket

REFRESH

SAIBA MAIS

curso-apache-beam

OBJETOS

CONFIGURAÇÃO

PERMISSÕES

RETENÇÃO

CICLO DE VIDA

Intervalos

curso-apache-beam

saida

FAZER UPLOAD DE ARQUIVOS

CARREGAR PASTA

CRIAR PASTA

GERENCIAR RETENÇÕES










FAZER O DOWNLOAD

EXCLUIR

Filtrar apenas pelo prefixo do nome

Filtro

Filtrar objetos e pastas

<input type="checkbox"/>	Nome	Tamanho	Tipo	Horário da criação	Classe de armazenamento	Última modificação	Acesso público	Criptografia	
<input type="checkbox"/>	 Voos_atrados_qtd.csv-00000-of-00	106 B	text/plain	6 de jun. de 2021 1...	Standard	6 de jun. de 202...	Não público	Google-managed key	 
<input type="checkbox"/>	 Voos_atrados_qtd.csv-00001-of-00	54 B	text/plain	6 de jun. de 2021 1...	Standard	6 de jun. de 202...	Não público	Google-managed key	 
<input type="checkbox"/>	 Voos_atrados_qtd.csv-00002-of-00	107 B	text/plain	6 de jun. de 2021 1...	Standard	6 de jun. de 202...	Não público	Google-managed key	 

Notas da versão