

---

# Apache Beam na Vida Real

Abordagem prática do uso do Apache Beam

Diego Marcello

---

---

# Introdução

**Diego Marcello, engenheiro de dados na SulAmérica**

Apaixonado por plataformas de dados e entusiasta em tecnologia...



- BigData e BI
- Desenvolvimento em cloud para Azure, AWS e GCP
- Python, Java, C# e NodeJS...
- SQL na veia...

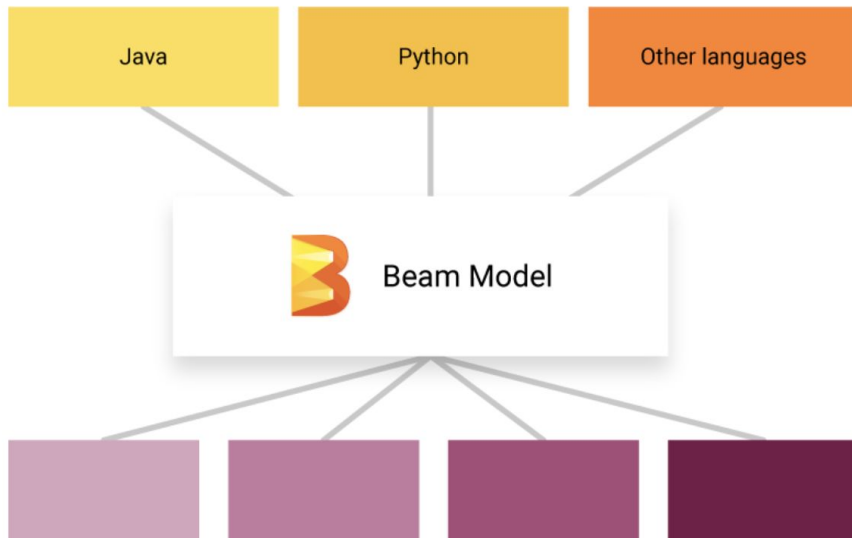
**LinkedIn:** <https://www.linkedin.com/in/diegomarcello/>

---

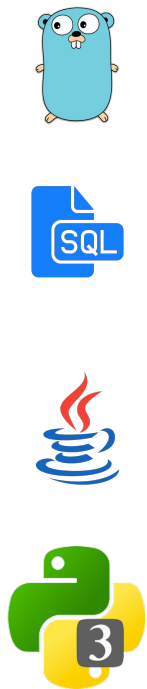
# O que é o Apache Beam?

Modelo unificado de código aberto para construção de pipeline de processamento em streaming ou lotes.

Desenvolvimento fácil, aplicação simples e baixo tempo de aprendizado. Esta é uma das plataformas mais interessantes para se aprender.



# Apache Beam - SDK



# Apache Beam - Pipeline



```
Pipeline p = Pipeline.create()
```

```
PCollection pCol1 = p.apply(transform).apply(...)
```

```
PCollection pCol2 = pCol1.apply(transform)
```

```
p.run()
```



# Apache Beam - Transformações

Podemos usar, na maioria dos casos, o que é chamado de transformações primitivas:

- ParDo / GroupByKey / AssignWindows / Flatten

pipeline

```
.apply(Create.of("to", "be", "or", "not", "to", "be"))

.apply(MapElements.via(

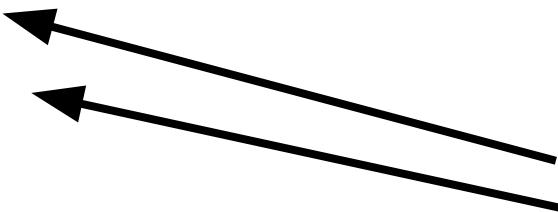
    new SimpleFunction<String, KV<String, Integer>>() {

        @Override public KV<String, Integer> apply(String input) {

            return KV.of(input, 1);}

    })

).apply(Sum.integersPerKey());
```



Transformação  
Composta



---

---

# Apache Beam SDK's - Python

Por mais que possamos usar muito mais Apache Beam com Java, dependendo do que precisa ser feito, usar o python é mais produtivo.

---

# Apache Beam - Save the python

Mais simples e mais intuitivo, com python temos o mesmo resultado.



pipeline

```
.apply(Create.of("to", "be", "or", "not",  
"to", "be"))  
  
.apply(MapElements.via(new  
SimpleFunction<String, KV<String,  
Integer>>() {  
    @Override  
    public KV<String, Integer> apply(String  
input) {  
        return KV.of(input, 1);  
    }  
}))  
  
.apply(Sum.integersPerKey());
```



pipeline

```
| beam.Create(['to', 'be', 'or', 'not',  
'to', 'be'])  
  
| beam.Map(lambda word: (word, 1))  
  
| beam.GroupByKey()  
  
| beam.Map(lambda kv: (kv[0], sum(kv[1])))
```



# Apache Beam - IO Transforms

Dentro da plataforma temos diversos modos de conexões já prontos para uso no Java e Python.

No Python temos um número menor de conectores, mas são suficientes para a maioria dos trabalhos.

Language	File-based	Messaging	Database
Java	Beam Java supports Apache HDFS, Amazon S3, Google Cloud Storage, and local filesystems. <a href="#">FileIO</a> (general-purpose reading, writing, and matching of files) <a href="#">AvroIO</a> <a href="#">TextIO</a> <a href="#">TFRecordIO</a> <a href="#">XmlIO</a> <a href="#">TikaIO</a> <a href="#">ParquetIO</a>	<a href="#">Amazon Kinesis</a> <a href="#">AMQP</a> <a href="#">Apache Kafka</a> <a href="#">Google Cloud Pub/Sub</a> <a href="#">JMS</a> <a href="#">MQTT</a> <a href="#">RabbitMQIO</a> <a href="#">SqsIO</a>	<a href="#">Apache Cassandra</a> <a href="#">Apache Hadoop Input/Output Format</a> <a href="#">Apache HBase</a> <a href="#">Apache Hive (HCatalog)</a> <a href="#">Apache Kudu</a> <a href="#">Apache Solr</a> <a href="#">Elasticsearch (v2.x, v5.x, v6.x)</a> <a href="#">Google BigQuery</a> <a href="#">Google Cloud Bigtable</a> <a href="#">Google Cloud Datastore</a> <a href="#">Google Cloud Spanner</a> <a href="#">JDBC</a> <a href="#">MongoDB</a> <a href="#">Redis</a>
Python/Batch	Beam Python supports Apache HDFS, Google Cloud Storage, and local filesystems. <a href="#">avroio</a> <a href="#">parquetio.py</a> <a href="#">textio</a> <a href="#">tfrecordio</a> <a href="#">vcfio</a>		<a href="#">Google BigQuery</a> <a href="#">Google Cloud Datastore</a> <a href="#">Google Cloud Bigtable (Write)</a> <a href="#">MongoDB</a>
Python/Streaming		<a href="#">Google Cloud Pub/Sub</a>	<a href="#">Google BigQuery (sink only)</a>

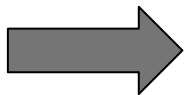
# Apache Beam - SDK Atualmente



beam



Flink



beam



# Apache Beam - Executor portátil

Em versões mais recentes, uma funcionalidade interessante foi implementada, onde pode ser trocado facilmente o executor.

1. Pipeline desenvolvido
2. Informe o executor, via parâmetro ou internamente:
  - a. `options.setRunner(FlinkRunner.class)`
  - b. `--runner=DirectRunner / --runner=SparkRunner`
3. `p.run()`

# Apache Beam - Executor portátil

## Dicas para executar no Spark

1. Crie um ambiente virtual
  - a. `virtualenv venv`
  - b. `source venv/bin/activate`
2. Instale o Apache Beam SDK
  - a. `pip install apache_beam`
3. Inicie o endpoint JobService
  - a. `./gradlew :runners:spark:job-server:runShadow`

4. Exemplo dentro do pipeline:

```
options = PipelineOptions([  
    "--runner=PortableRunner",  
    "--job_endpoint=localhost:8099",  
    "--environment_type=LOOPBACK"  
])
```

# Apache Beam - Usando pandas

Digamos que você precise processar vários dataframe's gerados através do pandas. Você pode usar isso em escala catalogando outras informações usando Apache Beam.

```
[...]
```

```
def exibindo_item(item):
```

```
    print(item)
```

```
def procurando_item(item):
```

```
    item2 = catalogo.find(item)
```

```
    return item2
```

```
df = pd.DataFrame({'letras' : ['a', 'b', 'c', 'd', 'e'], 'numeros' : [1, 2, 3, 4, 5],})
```

```
[...]
```

```
(p | 'Lendo DataFrame do Pandas' >> beam.Create(df.values.tolist()))
```

```
| 'Procura item' >> beam.Map(procurando_item)
```

```
| 'Mostrando resultado' >> beam.Map(exibindo_item))
```

# Vale ler...

<https://s.apache.org/beam-runner-api>

<https://s.apache.org/beam-runner-api-combine-model>

<https://s.apache.org/beam-fn-api>

<https://s.apache.org/beam-fn-api-processing-a-bundle>

<https://s.apache.org/beam-fn-state-api-and-bundle-processing>

<https://s.apache.org/beam-fn-api-send-and-receive-data>

<https://s.apache.org/beam-fn-api-container-contract>

<https://s.apache.org/beam-portability-timers>

## Referências

---

# Obrigado!

Email: [diego.marcello@sulamerica.com.br](mailto:diego.marcello@sulamerica.com.br)

---