

Big data



A evolução da gestão de dados

Bancos de dados relacionais

Um banco de dados relacional é um conjunto de informações que organiza dados em relações predefinidas, em que os dados são armazenados em uma ou mais tabelas (ou "relações") de colunas e linhas, facilitando a visualização e a compreensão de como as diferentes estruturas de dados se relacionam.

Os relacionamentos são uma conexão lógica entre diferentes tabelas, estabelecidas com base na interação entre estas.



A evolução da gestão de dados

Data Warehouses

Um data warehouse é um repositório centralizado que armazena dados estruturados (tabelas de banco de dados, planilhas do Excel) e dados semiestruturados (arquivos XML) para geração de relatórios e análises.

Os dados fluem de várias fontes, sendo limpos e padronizados antes de chegarem ao warehouse.

Como o data warehouse pode armazenar grandes quantidades de informações, ele fornece aos usuários acesso fácil a uma abundância de dados históricos que podem ser usados para mineração e visualização de dados, entre outros formatos de relatórios de business intelligence.



O que é big data?



O que é big data?

Big Data nada mais é do que o processamento de quantidades muito grandes de dados;

Algumas pessoas começam a falar em Big Data quando bancos de dados tradicionais deixam de atender as volumetrias desejadas, mas temos algumas características que nos ajudam a definir a fronteira.

O que é big data?

1. Volume

2. Velocidade



O que é big data?

3. Variedade

Data View - Employee (Drawing1.dwg)

— New Link Template — — New Label Te

Emp_Id	Last_Name	First_Name	Gender	Title
1000	Torbati	Yolanda	F	Programmer
1001	Kleinn	Joel	M	Programmer
1002	Ginsburg	Laura	F	President
1003	Cox	Jennifer	F	Programmer
1005	Ziada	Mauri	M	Product Designer
1006	Keyser	Cara	F	Account Executive
1010	Smith	Roxie	M	Programmer
1011	Nelson	Robert	M	Programmer
1012	Sachsen	Lars	M	Support Technician
1013	Shannon	Don	M	Product Designer

Record 1



```
[
  {
    "description": "quarter",
    "mode": "REQUIRED",
    "name": "qtr",
    "type": "STRING"
  },
  {
    "description": "sales representative",
    "mode": "NULLABLE",
    "name": "rep",
    "type": "STRING"
  },
  {
    "description": "total sales",
    "mode": "NULLABLE",
    "name": "sales",
    "type": "INTEGER"
  }
]
```

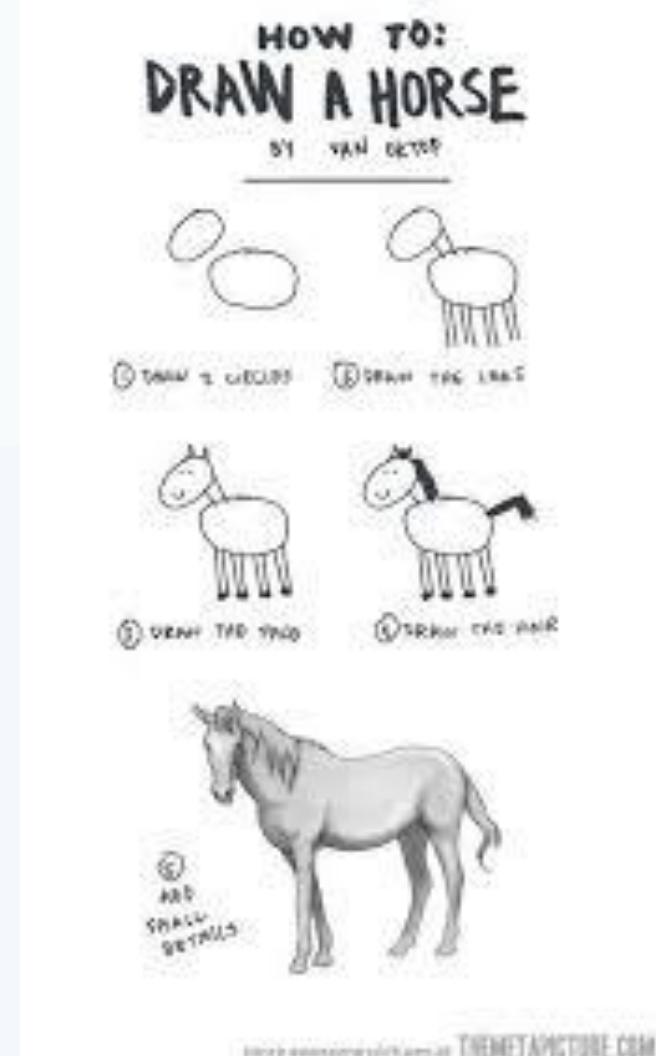
O que é big data?

4. Veracidade

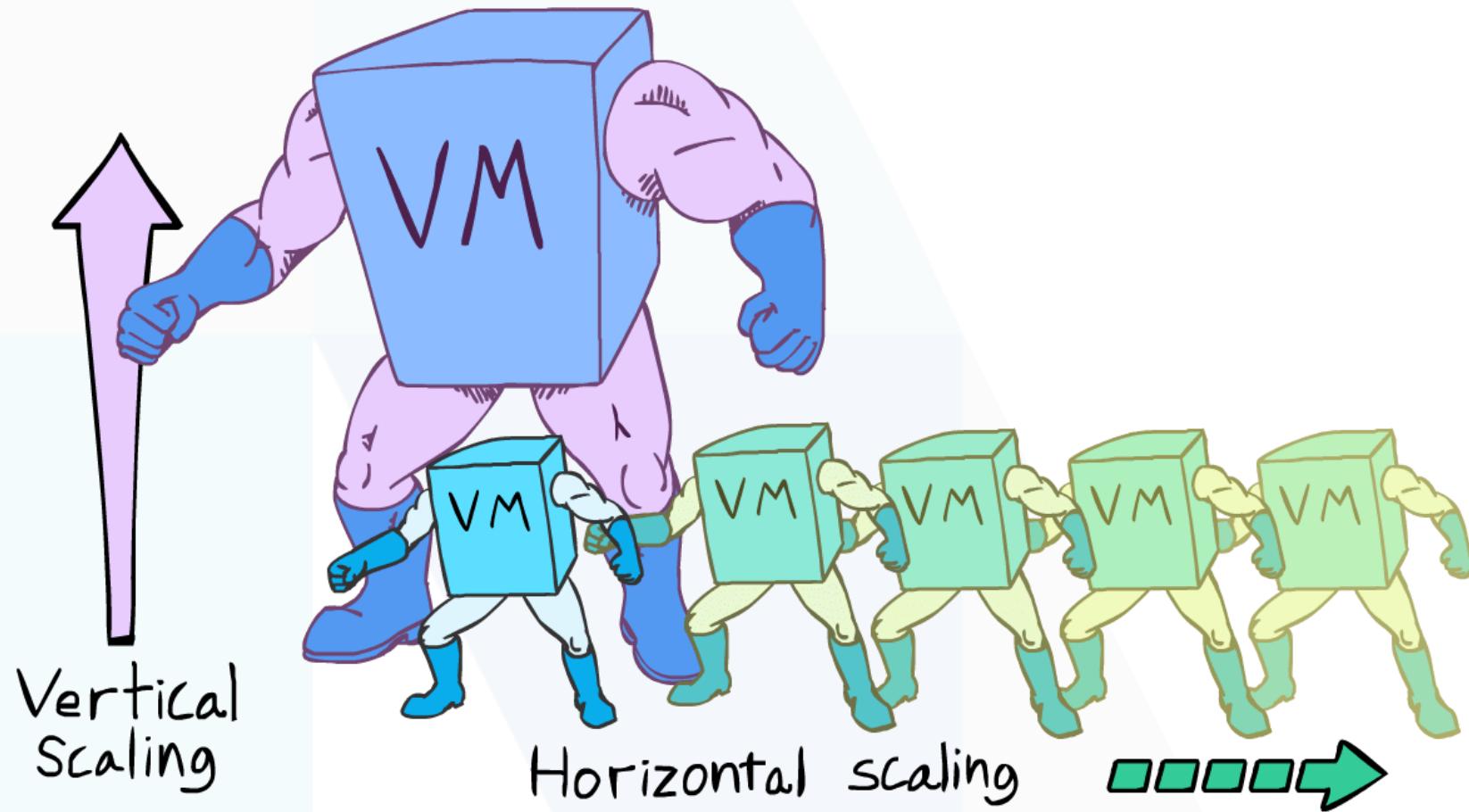


O que é big data?

Mas como?



Dividir e conquistar (Crescimento horizontal)



Particionamento dos dados

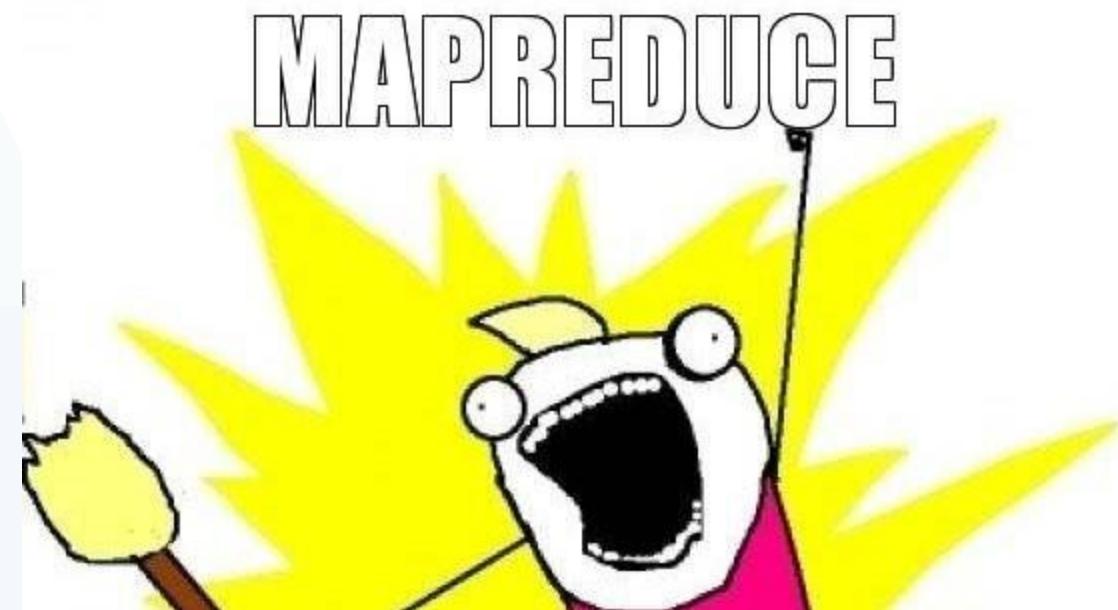




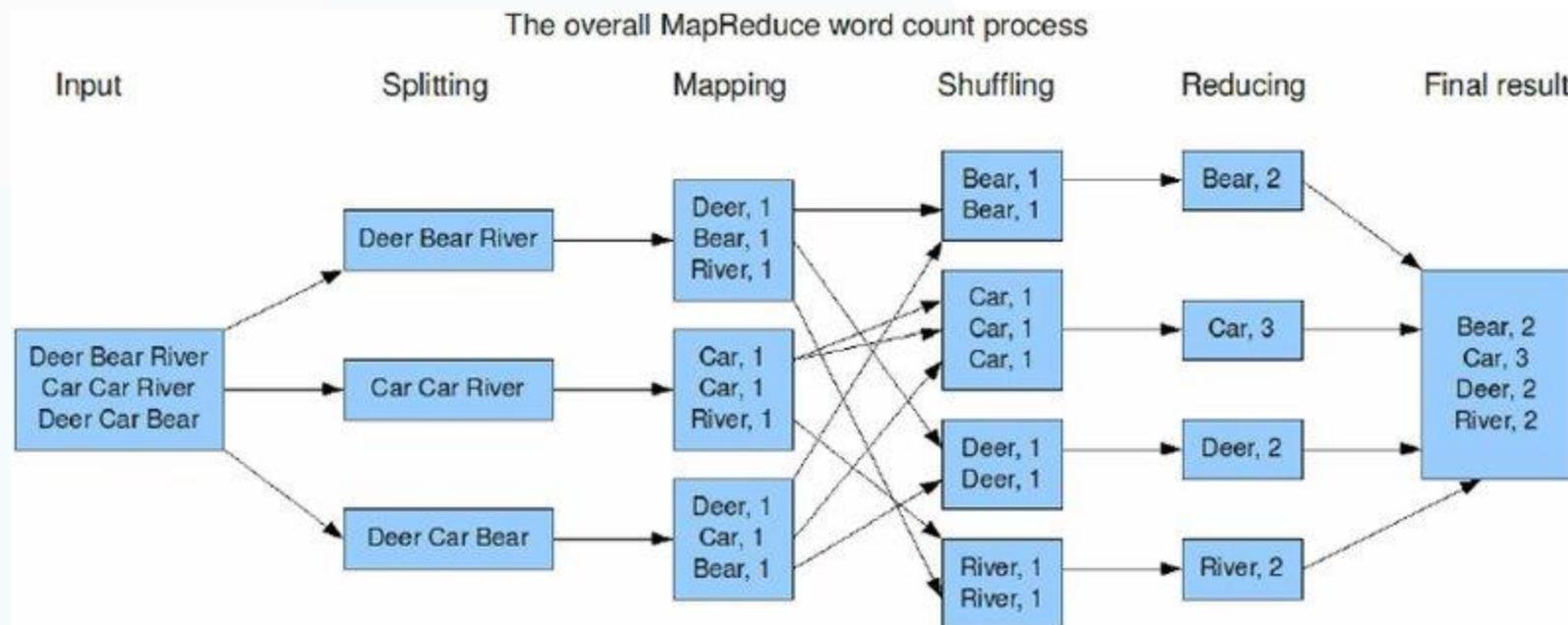
Mapreduce

Sobre o mapreduce:

- Mapreduce é um modelo de programação em que cada tarefa é especificada em termos de funções de mapeamento e redução;
- Tarefas de map e reduce executam paralelamente no cluster;



Mapreduce



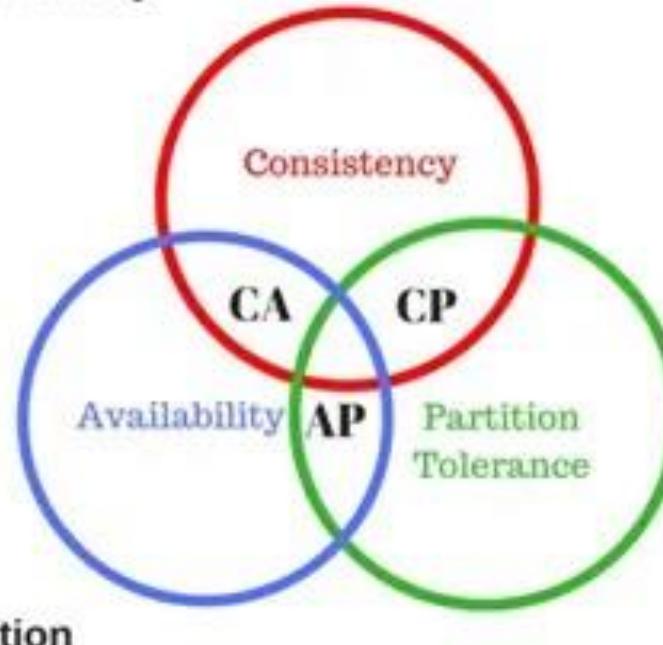
SQL vs NoSQL?



O diagrama CAP

Consistency & Availability

- MySQL
- PostGreSQL



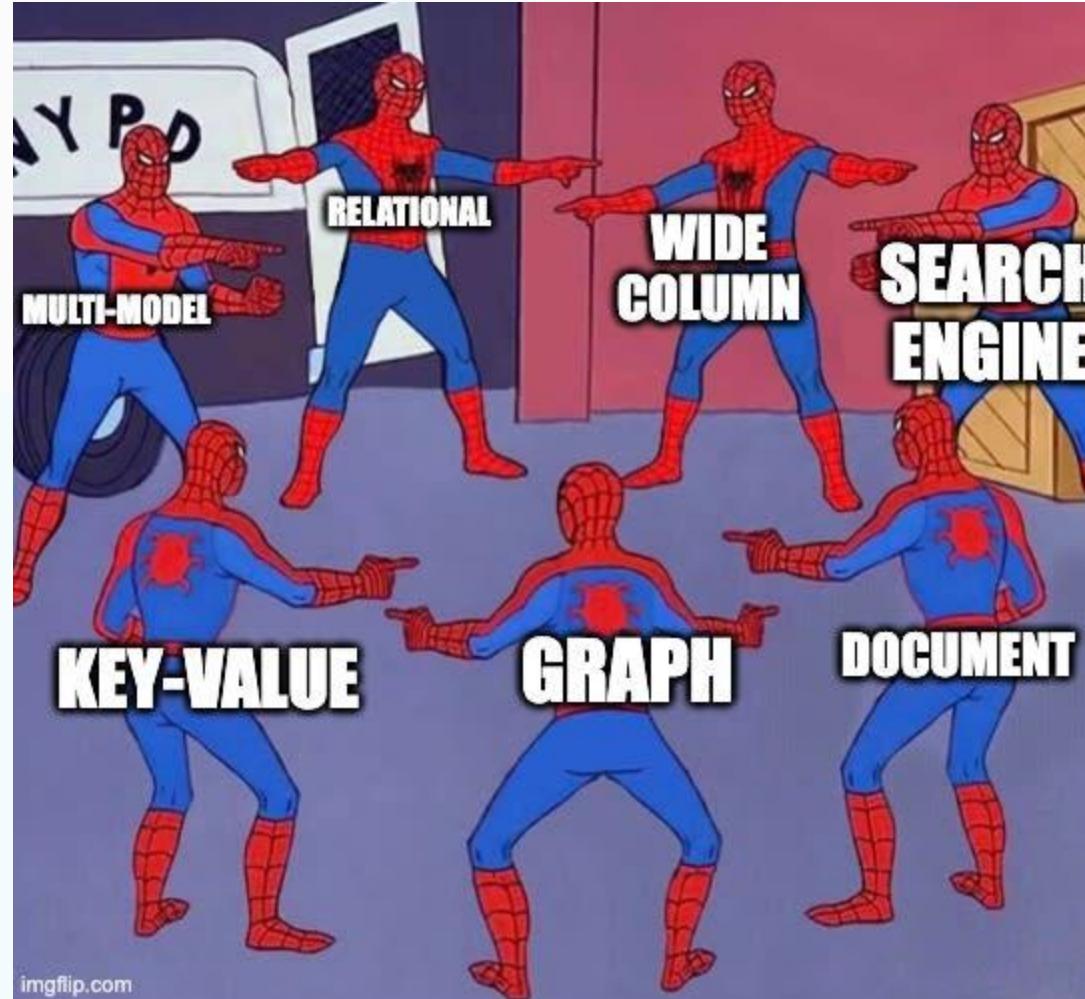
Availability & Partition Tolerance

- Riak
- Cassandra
- CouchDB
- DynamoDB

Consistency & Partition Tolerance

- HBase
- MongoDB
- Redis
- MemCache

SQL vs NoSQL?



A evolução da gestão de dados (parte 2)

Data Lakes

Um data lake é um repositório centralizado que ingere e armazena grandes volumes de dados em sua forma original.

Os dados podem ser processados e usados como base para uma variedade de necessidades analíticas.

Devido à sua arquitetura aberta e escalonável, um data lake pode acomodar todos os tipos de dados de qualquer fonte, desde dados estruturados (tabelas de banco de dados, planilhas do Excel) até semiestruturados (arquivos XML, páginas da Web) e não estruturados (imagens, arquivos de áudio, tweets), tudo sem sacrificar a fidelidade.



A evolução da gestão de dados (parte 2)

Lakehouses

O objetivo dos data lakehouses é centralizar data centers distintos e simplificar os esforços de engenharia para que todos na organização possam ser usuários de dados.

Um data lakehouse usa o mesmo armazenamento de objetos em nuvem de baixo custo dos data lakes para fornecer armazenamento sob demanda para facilitar o provisionamento e o escalonamento.

Ele também integra camadas de metadados para fornecer recursos semelhantes aos do armazenamento, como esquemas estruturados, suporte para transações ACID, governança de dados e outros recursos de otimização e gerenciamento de dados.



A evolução da gestão de dados (parte 2)

Data mesh

- Disponibilização de dados como produto: Trata-se da ideia de que os dados devem ser entregues como produtos, acessíveis por outros domínios através de APIs, com a premissa de que sejam dados limpos, atualizados e completos.
- Governança federada de dados: Este princípio visa permitir a interoperabilidade entre os domínios, através de políticas, padrões e responsabilidades, garantindo a conformidade e qualidade dos dados.



**Data
Mess**

**Data
Mesh**

A evolução da gestão de dados (parte 2)

Data mesh

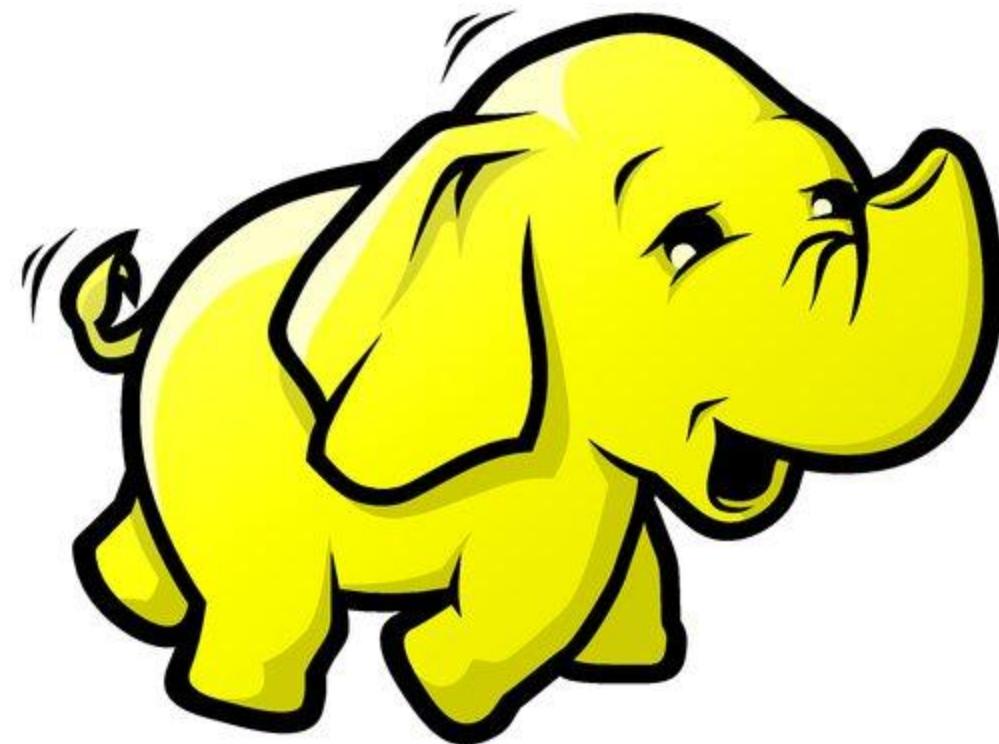
- Infraestrutura para disponibilização dos dados como self-service: refere-se ao uso de tecnologias que facilitam a descentralização dos dados, permitindo que equipes de diferentes domínios possam criar e gerenciar seus dados sem depender de uma equipe centralizada.
- Arquitetura de dados descentralizada: este princípio destaca que as equipes devem ter propriedade sobre o ciclo de vida dos dados, assegurando sua qualidade e entregando valor. As equipes tornam-se responsáveis por seus próprios dados analíticos e operacionais.



**Data
Mess**

**Data
Mesh**

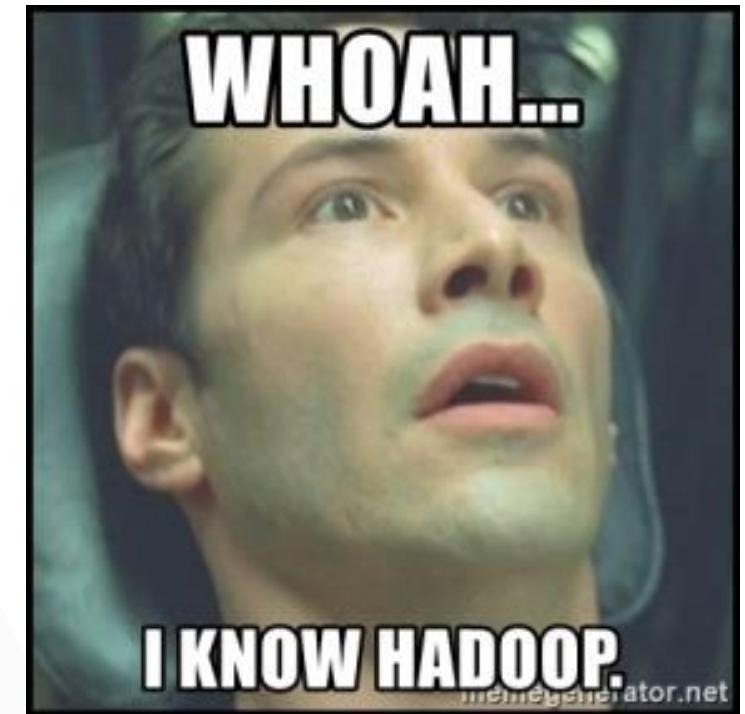
O que é Hadoop?



O que é Hadoop?

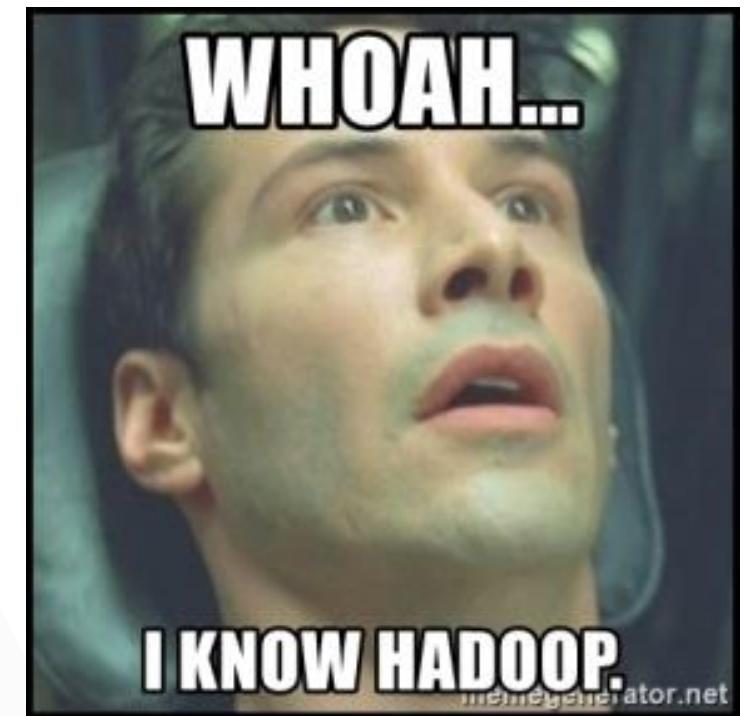
O Hadoop é antes de mais nada um sistema de arquivos distribuído. Ele é composto de dois componentes principais:

- HDFS (Hadoop Distributed File System), responsável por manipular o armazenamento de dados entre todos os nós do cluster;
- Map-Reduce, manipula o processamento dos dados dentro dos arquivos do cluster.



O que é Hadoop?

- O HDFS é um sistema de arquivos escalável e distribuído, cujo desenho é baseado fortemente no GFS (Google File System).
- Um sistema de arquivos distribuído divide o arquivo em partes menores, distribuindo-o entre os diversos nós do cluster.
- Dessa maneira, podemos não só armazenar arquivos muito maiores, como também aceleramos o processamento, visto que cada nó processará, em geral, apenas a sua parte.



O que é Hadoop?

Em relação aos nós do cluster, podemos separá-los em dois grupos:

- Namenode: administra o namespace do sistema de arquivos, armazenando os metadados destes. Em outras palavras, ele mapeia os arquivos e os blocos nos quais estes se encontram armazenados.
- Datanode: armazena as partes dos arquivos em forma de blocos. Os datanodes reportam aos namenodes o estado dos arquivos que possuem, de modo que este último esteja ciente do que pode ser processado.
- Uma terceira estrutura possível é a figura do namenode secundário, responsável por assumir as atividades do namenode primário, quando este estiver indisponível.



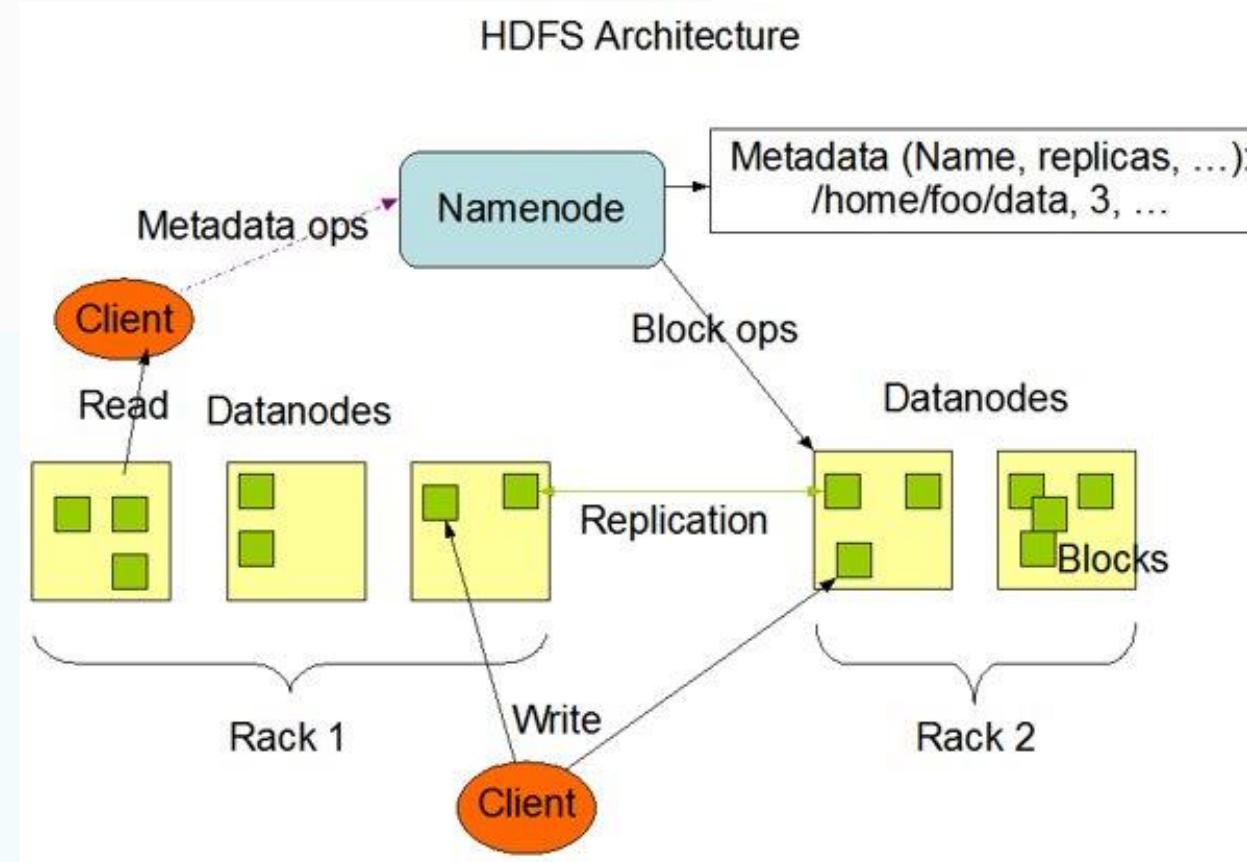
O que é Hadoop?

Sobre os arquivos:

- Arquivos no hadoop são “quebrados” em partes menores, por padrão 64MB (podendo ser configurado no sistema de arquivos);
- Cada uma dessas partes é multiplicada pelo número de réplicas configuradas (por padrão 3) e distribuída em diferentes datanodes. Por essa razão, quando fazemos as análises de capacidade do cluster, devemos sempre considerar o tamanho estimado dos arquivos multiplicado pelo número de réplicas;
- A localização de cada uma das partes é armazenada no namenode para processos futuros.



O que é Hadoop?



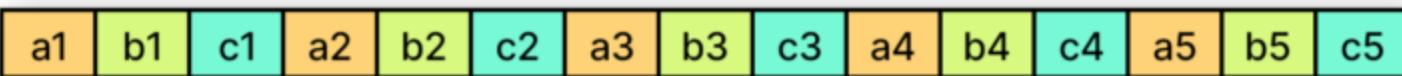
Uma pequena pausa para os formatos de arquivo

NTT DATA

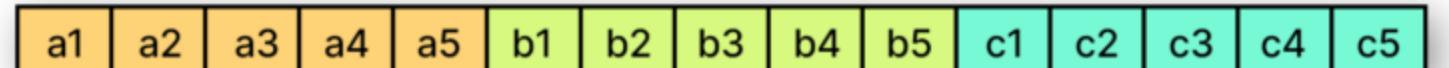
Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

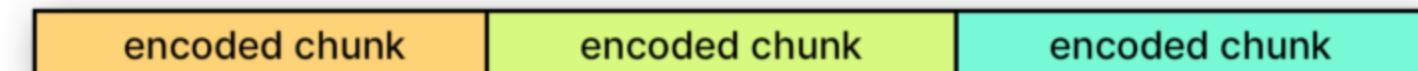
Row Layout



Column Layout



↓ ↓ ↓ encoding



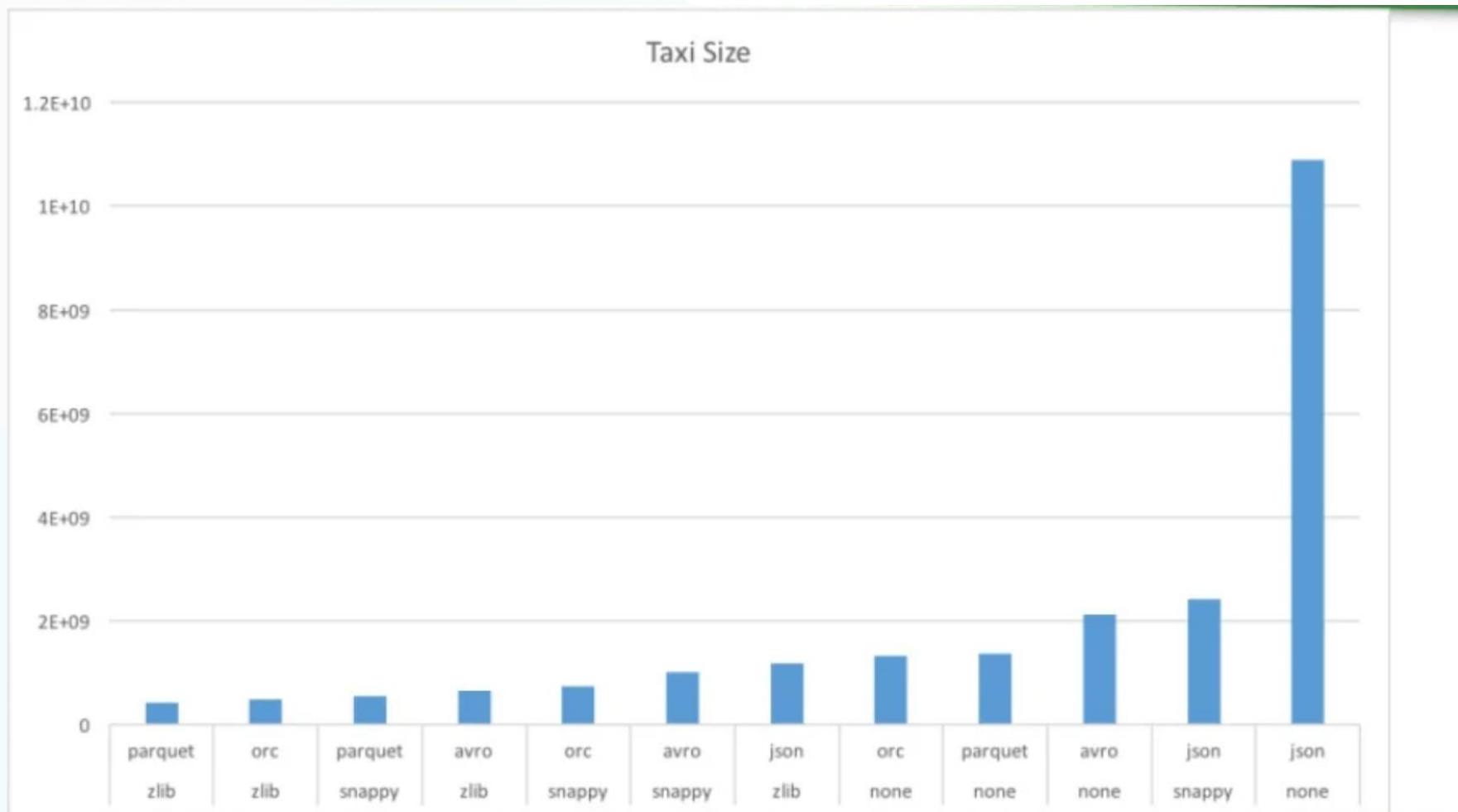


Apache
ORC™



Uma pequena pausa para os formatos de arquivo

NTT DATA



© Hortonworks Inc. 2011 – 2016. All Rights Reserved



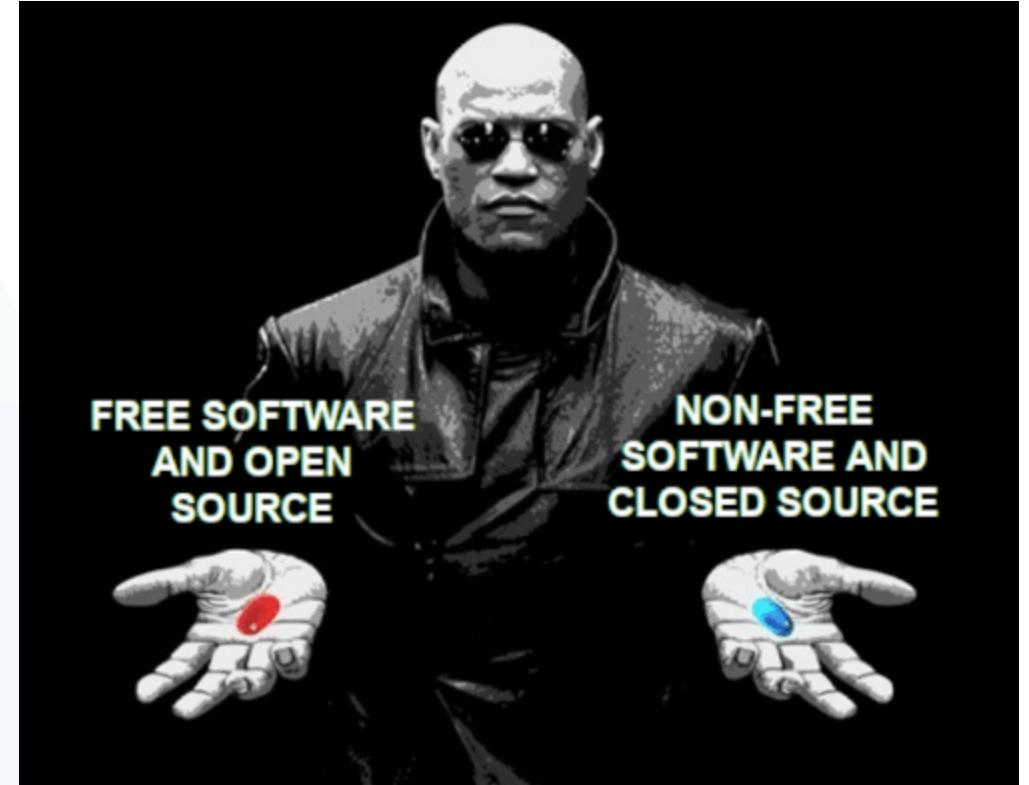
<https://www.slideshare.net/HadoopSummit/file-format-benchmark-avro-json-orc-parquet-65740483>

Por que open source?

O software open source é colaborativo, conta com a produção comunitária e revisão por colegas para usar, mudar e compartilhar código-fonte.

Desenvolvedores compartilham insights, ideias e códigos para criar soluções de software mais inovadoras coletivamente e individualmente.

O Open source software opera com os princípios básicos de produção em pares e colaboração em massa, criando um desenvolvimento de software mais sustentável para usuários finais.



File storages



Processamento massivo

NTT DATA

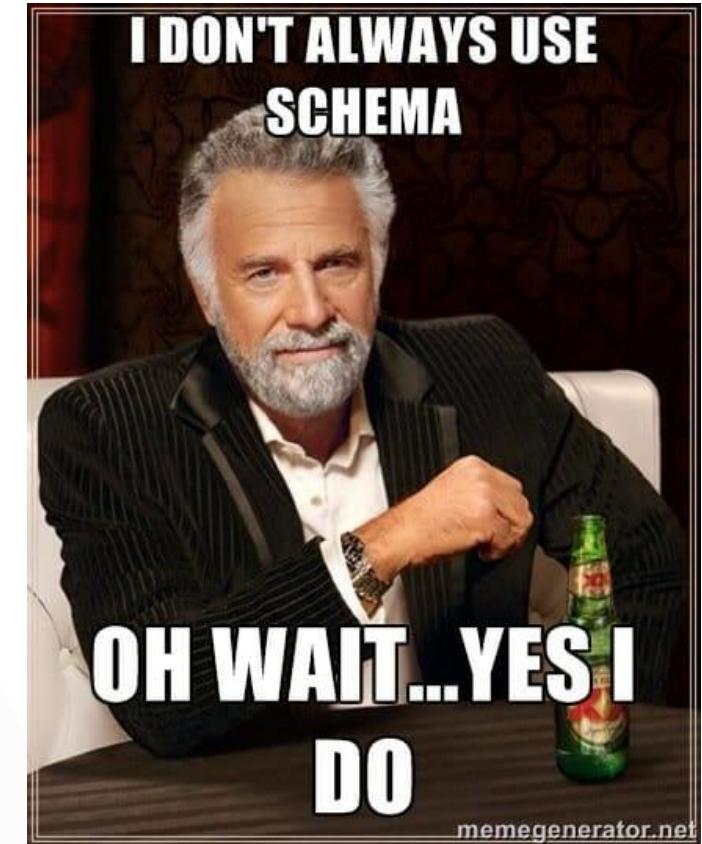


 RabbitMQ

 kafka

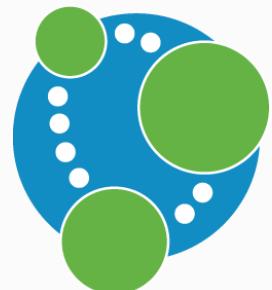


Bancos de dados de documentos

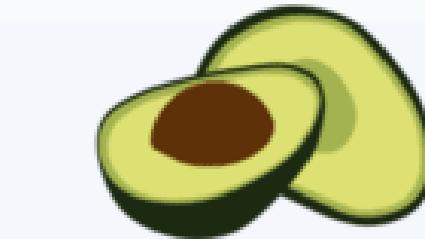


Bancos de dados de grafos

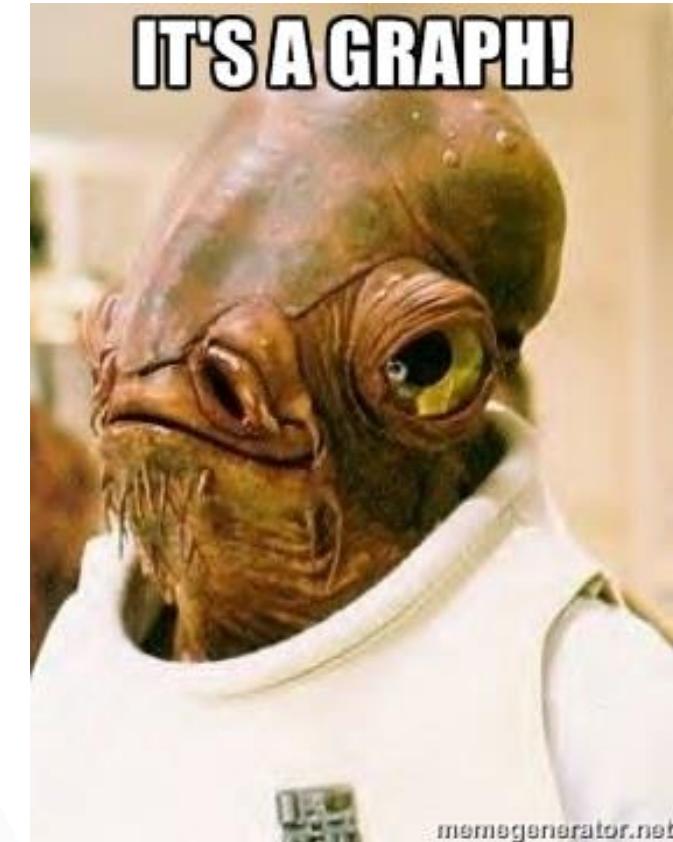
NTT DATA



neo4j

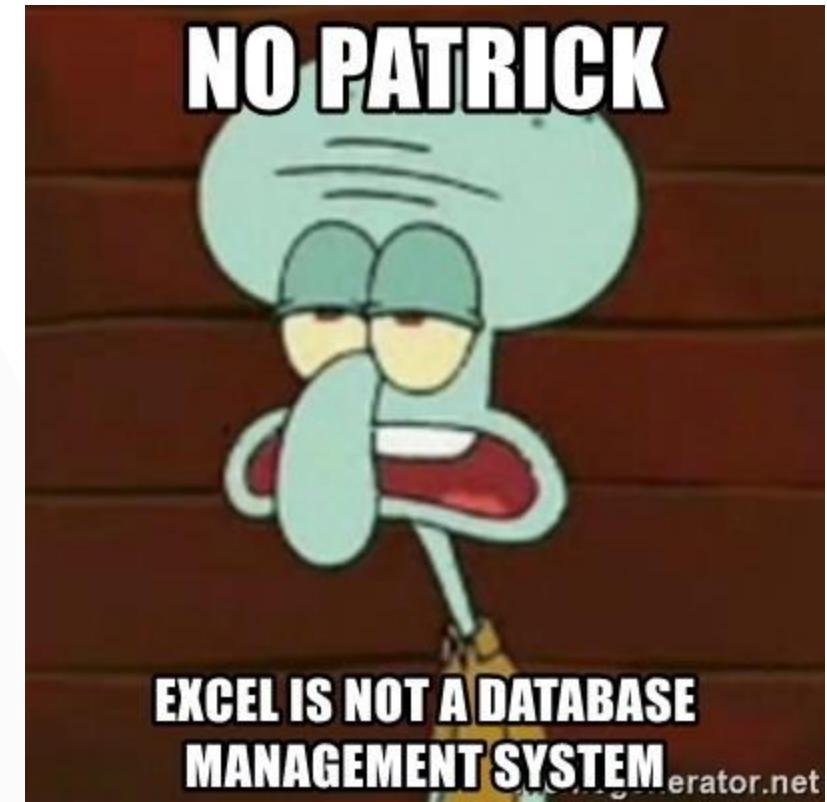


ArangoDB

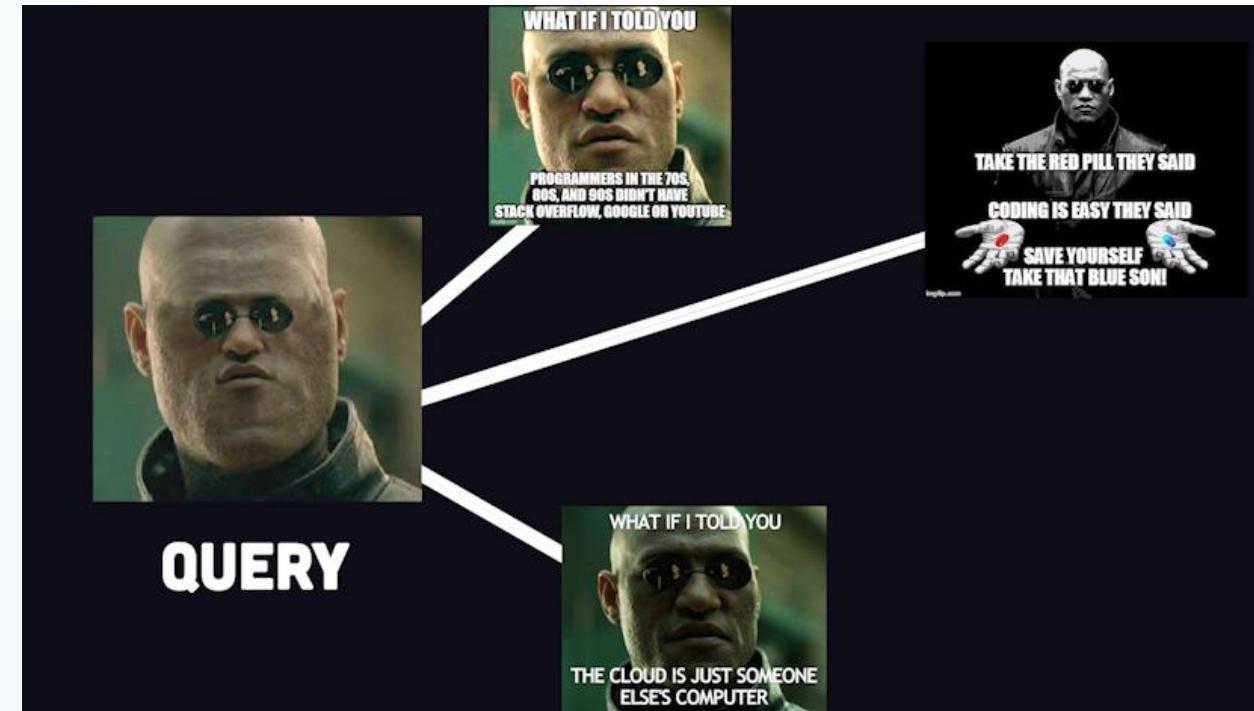


Bancos de dados NOSQL colunares

NTT DATA



Vector search



Distributed Query Engine

NTT DATA



prestoDB



APACHE
DRILL

APACHE
Spark™



Bancos de dados de busca textual

NTT DATA



elasticsearch



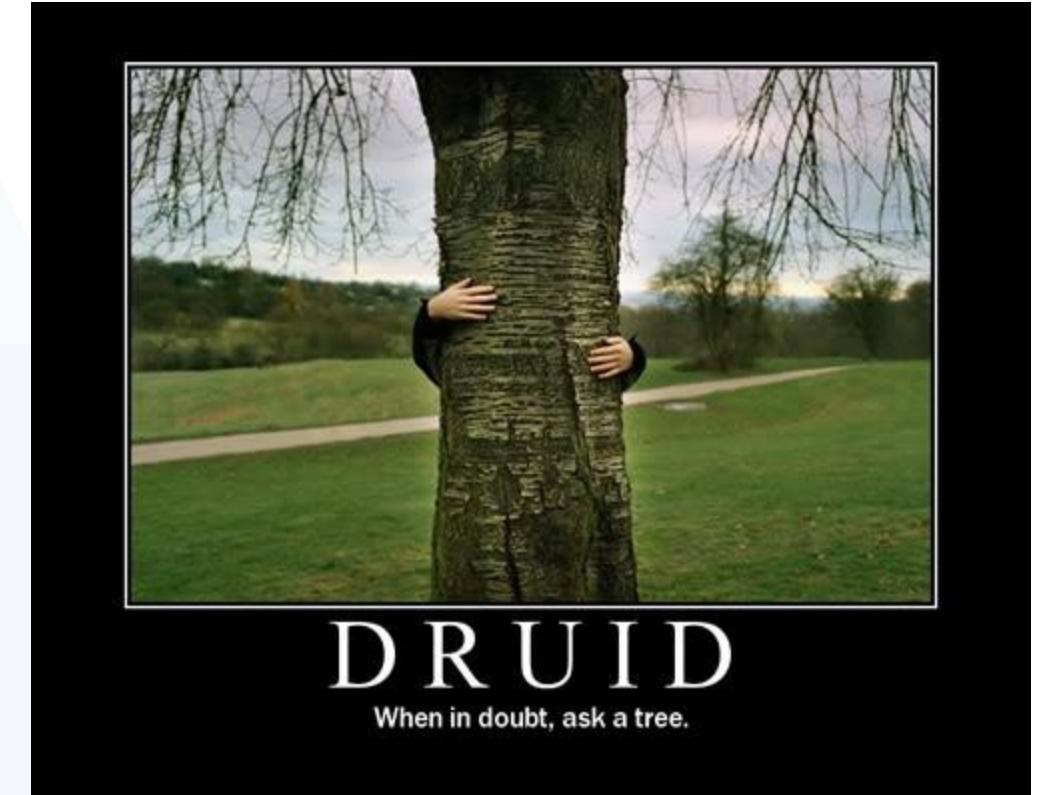
Bancos de dados em memória

NTT DATA

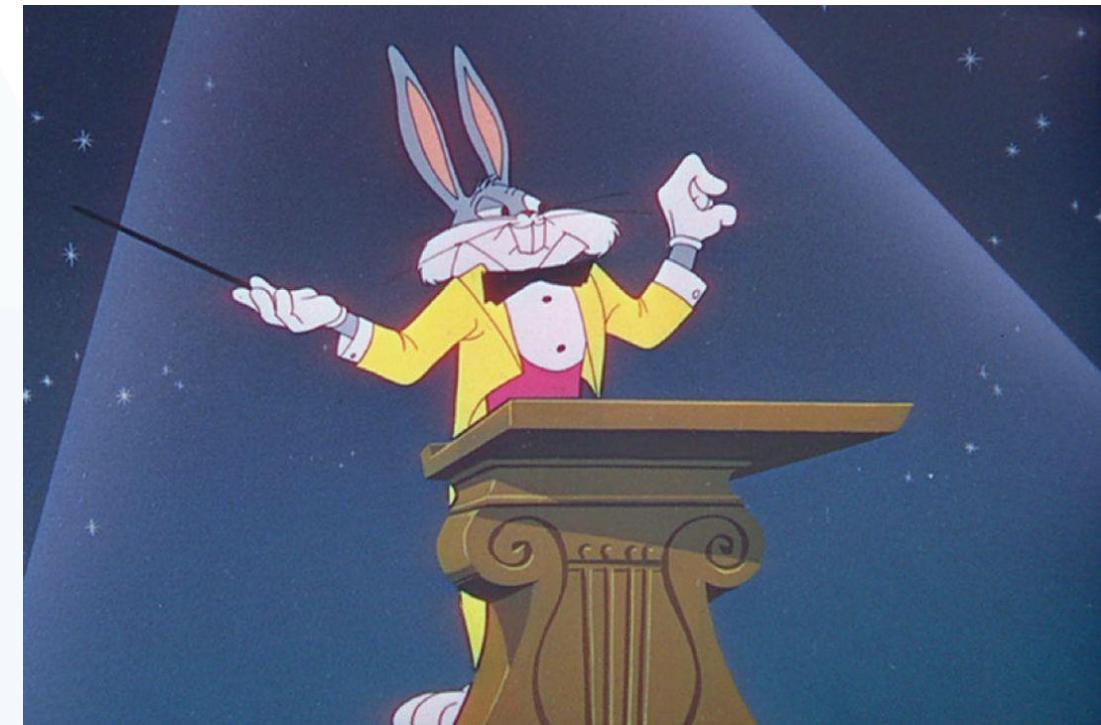


Bancos de dados OLAP (Online Analytical Processing)

NTT DATA



Orquestração



Transformação de dados (visual)



Timeseries

NTT DATA



OLTP (Online Transaction Processing)

NTT DATA



Dataviz

NTT DATA



Mão na massa!

Brincando com docker

Alguns passos importantes que vamos utilizar:

Fazer o download do kubectl:

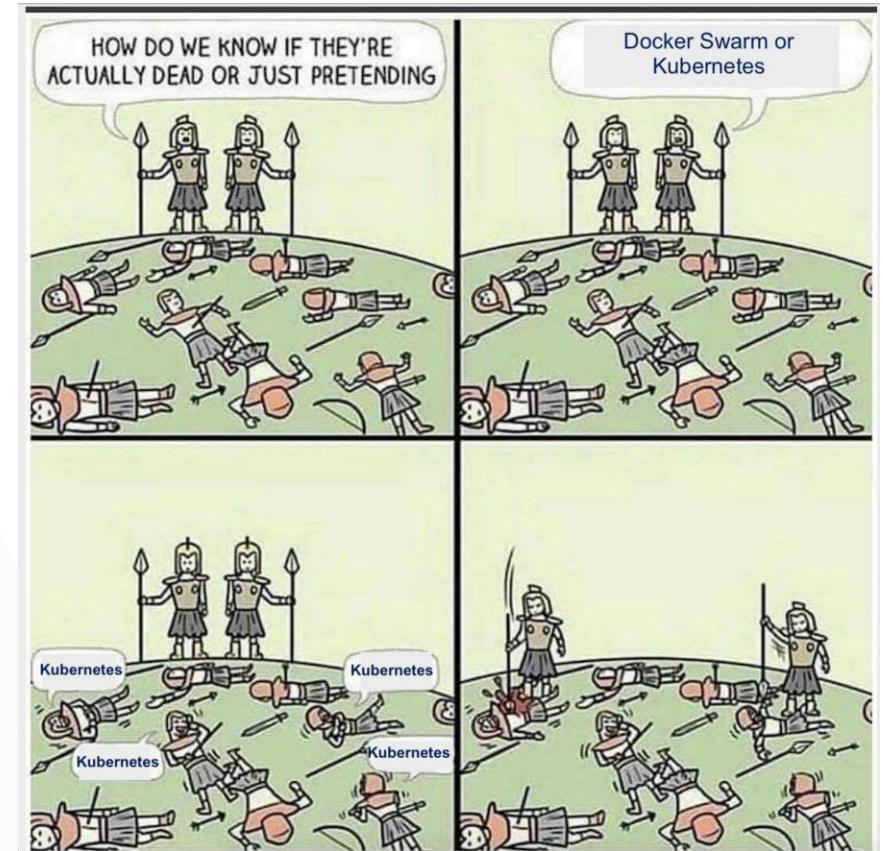
<https://dl.k8s.io/release/v1.29.1/bin/windows/amd64/kubectl.exe>

Copiar o config do Kubernetes para:

C:\Users\<usuário>\.kube\config

Para checar se o cluster foi configurado:

kubectl config view



Brincando com docker

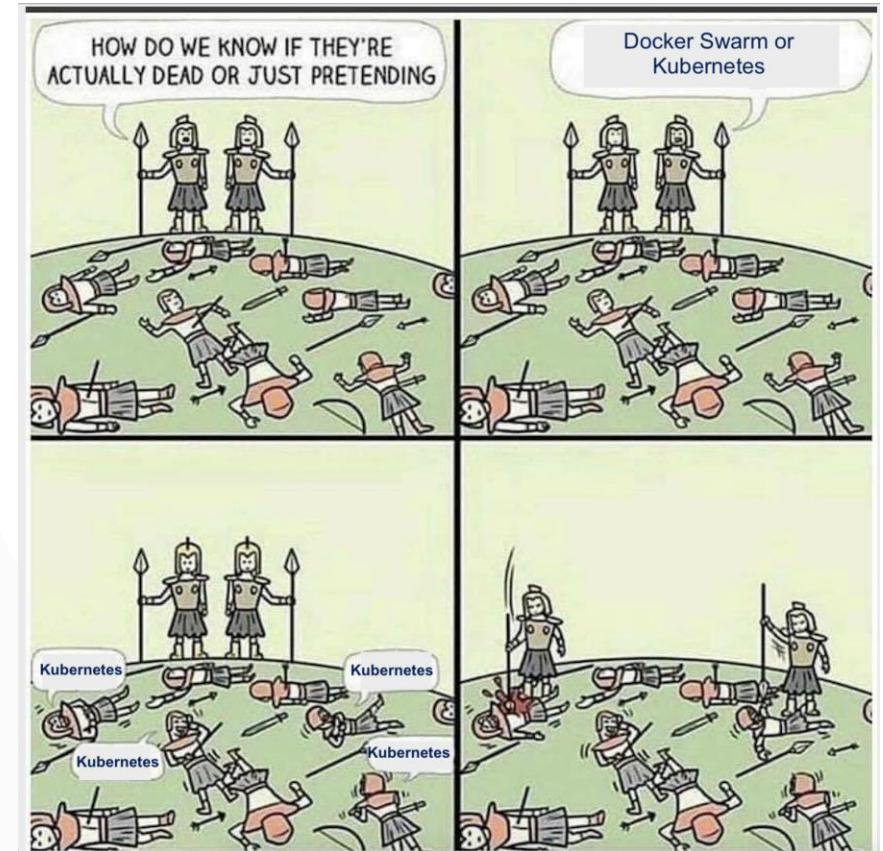
Alguns passos importantes que vamos utilizar:

Para logar no namenode:

```
kubectl exec -it simple-hdfs-namenode-default-0 --  
/bin/bash
```

Para fazer um port-forward:

```
kubectl port-forward spark-jupyter-{id de vcs} --  
/bin/bash
```



Entendendo o ambiente

Você tem duas maneiras de executar comandos hadoop:

- hdfs dfs -<command>
- hadoop fs -<command>



Entendendo o ambiente

Comandos principais:

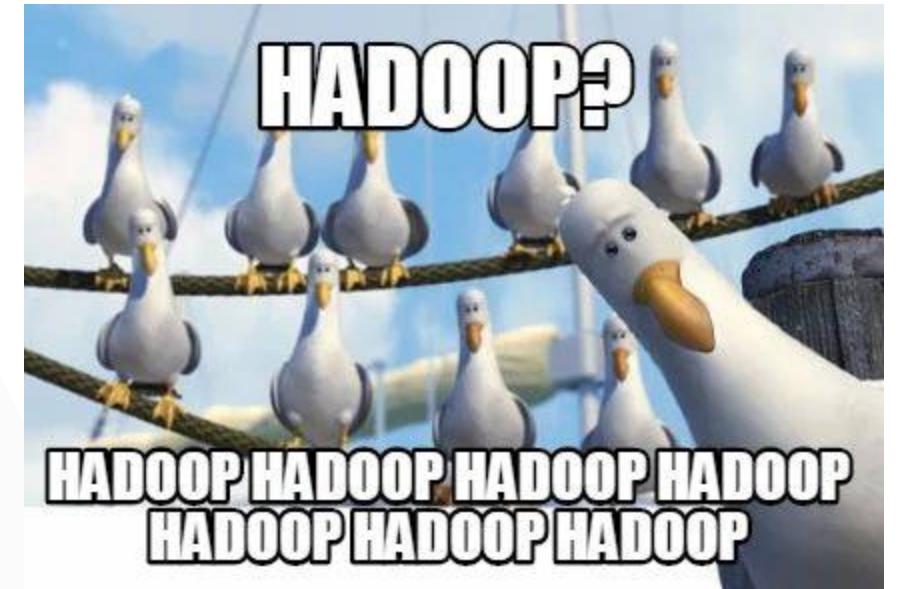
- Listar arquivos:
hdfs dfs -ls
- Versão recursiva do ls:
hdfs dfs -lsr
- Criar um diretório:
hdfs dfs -mkdir meuDiretorio
- Conta o número de arquivos de um diretório:
hdfs dfs -count meuDiretorio



Entendendo o ambiente

Comandos principais:

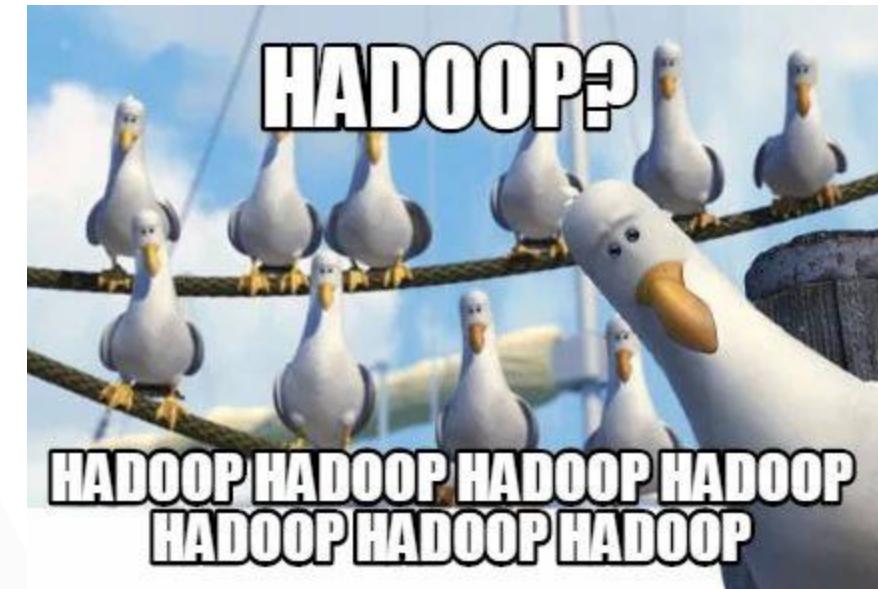
- Visualizar dados de um arquivo:
`hdfs dfs -cat meuDiretorio/arquivo.txt`
- Alterar permissões da minha pasta ou arquivo:
`hdfs dfs -chmod 777 meuDiretorio`
- Remover arquivo:
`hdfs dfs -rm pasta/config.sh`
- Remover pasta:
`hdfs dfs -rm -r pasta/config.sh`



Entendendo o ambiente

Comandos principais:

- Copiar um arquivo de um diretório:
hdfs dfs -cp meuDiretorio/arquivo.txt novoDiretorio/
- Copiar arquivos de um diretório:
hdfs dfs –cp -r meuDiretorio/ novoDiretorio/
- Retorna o espaço total utilizado pelos arquivos:
hdfs dfs -du meuDiretorio



Entendendo o ambiente

Comandos principais:

- Mostra o espaço livre em disco:
`hdfs dfs -df meuDiretorio/`
- Realiza o merge dos arquivos de uma pasta em uma determinada saída:
`hdfs dfs -getmerge -nl /src /opt/output.txt`
- Adicionar arquivos no hadoop:
`hdfs dfs -put arquivo.txt meuDiretorio`
- Obter arquivo do hadoop:
`hdfs dfs -get meuDiretorio/arquivo.txt`



Spark RDD

- O RDD ou Resilient Distributed Dataset é a principal abstração do Spark.
- Ele nada mais é do que uma coleção de elementos particionados entre os diversos nós de um cluster.
- RDDs contém algumas características principais:
 - Ele, como o próprio nome diz, é resiliente a falhas, ou seja, caso haja alguma falha durante o processo, ele é capaz de se recuperar e continuar a atividade;
 - RDDs são estruturados para serem naturalmente distribuídos, sendo capazes de existir entre diversos nós de um cluster;
 - Eles são imutáveis. Um RDD gera outro RDD, jamais ele poderá ser modificado. Seu conteúdo poderá ser transformado, resultando em outro RDD.

Transformando o RDD

Um dos pontos mais importantes quando trabalhamos com RDD é a capacidade de transformar os dados em algo novo. Para isso temos algumas operações que utilizaremos com frequência. A primeira e mais utilizada é a função map:

```
wget https://raw.githubusercontent.com/wess/iotr/master/lotr.txt  
curl -O https://raw.githubusercontent.com/wess/iotr/master/lotr.txt
```

```
lines = sc.textFile("hdfs:///diretorio/lotr.txt")  
lines.take(10)  
lineLengths = lines.map(lambda s: len(s))
```

Transformando o RDD

Outra transformação extremamente útil é o reduce:

```
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda valAcumulado, valSeguinte: valAcumulado +
valSeguinte)
```

Transformando o RDD

O flatMap é uma transformação que converte uma lista de elementos em linhas do meu RDD (um RDD de lista de strings para um RDD de strings, por exemplo):

```
lineLengths = lines.flatMap(lambda s: s.split(";"))
```

Transformando o RDD

O `reduceByKey`, realiza uma operação semelhante ao `reduce`, porém agrupa os valores por uma chave:

```
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

Outras transformações

sortBy:

```
words = (lines.flatMap(lambda line: line.split(" "))  
         .map(lambda word: word.lower())  
         .map(lambda word: (word, 1))  
         .reduceByKey(lambda a, b: a + b)  
         .sortBy(lambda word_count: word_count[1], ascending=False))
```

Outras transformações

Count:

```
count = lines.flatMap(lambda line: line.split(" ")) \  
    .map(lambda words: words.lower()) \  
    .count()
```

Outras transformações

Distinct:

```
count = (lines.flatMap(lambda line: line.split(" "))  
         .map(lambda word: word.lower())  
         .distinct()  
         .count())
```

Outras transformações

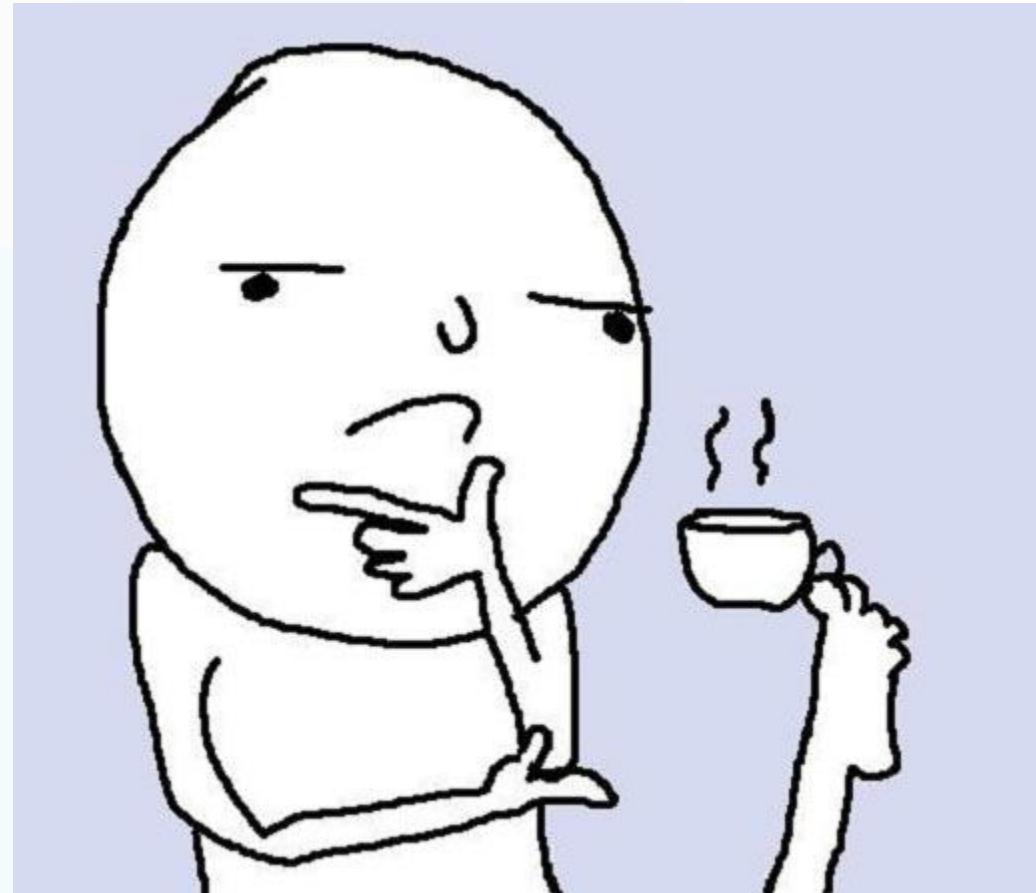
Joins:

```
conjunto1 = sc.parallelize([("Abacaxi", 22), ("Laranja", 13), ("Melancia", 15)])
conjunto2 = sc.parallelize([("Abacaxi", 65), ("Abacaxi", 2), ("Laranja", 23), ("Melancia", 5)])
```

```
joined = conjunto1.join(conjunto2).take(1000)
for item in joined:
    print(item)
```

Exercícios

E se quisermos saber o total de frutas?



Outras transformações

countByKey

```
conjunto1 = sc.parallelize([("Abacaxi", 22), ("Laranja", 13), ("Melancia", 15)])
conjunto2 = sc.parallelize([("Abacaxi", 65), ("Abacaxi", 2), ("Laranja", 23), ("Melancia", 5)])
```

```
result = (conjunto1.join(conjunto2)
          .map(lambda x: (x[0], x[1][0] + x[1][1]))
          .countByKey())
```

```
for key, count in result.items():
    print(f'{key}: {count}')
```

Exercício

Agora que sabemos as operações básicas envolvendo um RDD, vamos a um pequeno exercício:

- Tokenizar o livro lotr.txt;
- Deixar as palavras em minúsculo;
- Realizar a contagem de palavras;