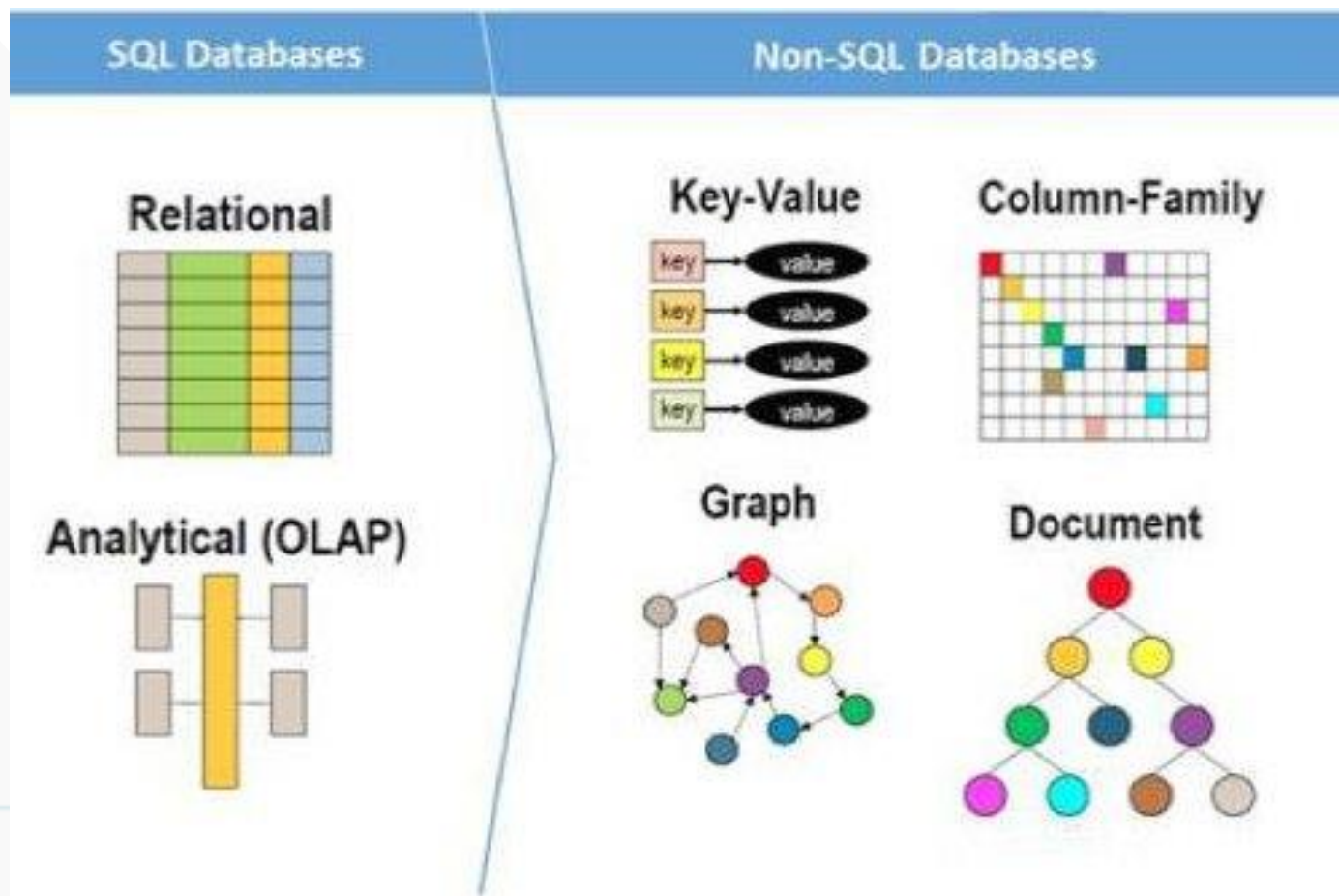




# Camadas e Serviços de Consumo de Dados

# Tipos de Bancos de Dados

# Banco de Dados Relacional



# Ranking Banco de Dados

<https://db-engines.com/en/ranking>

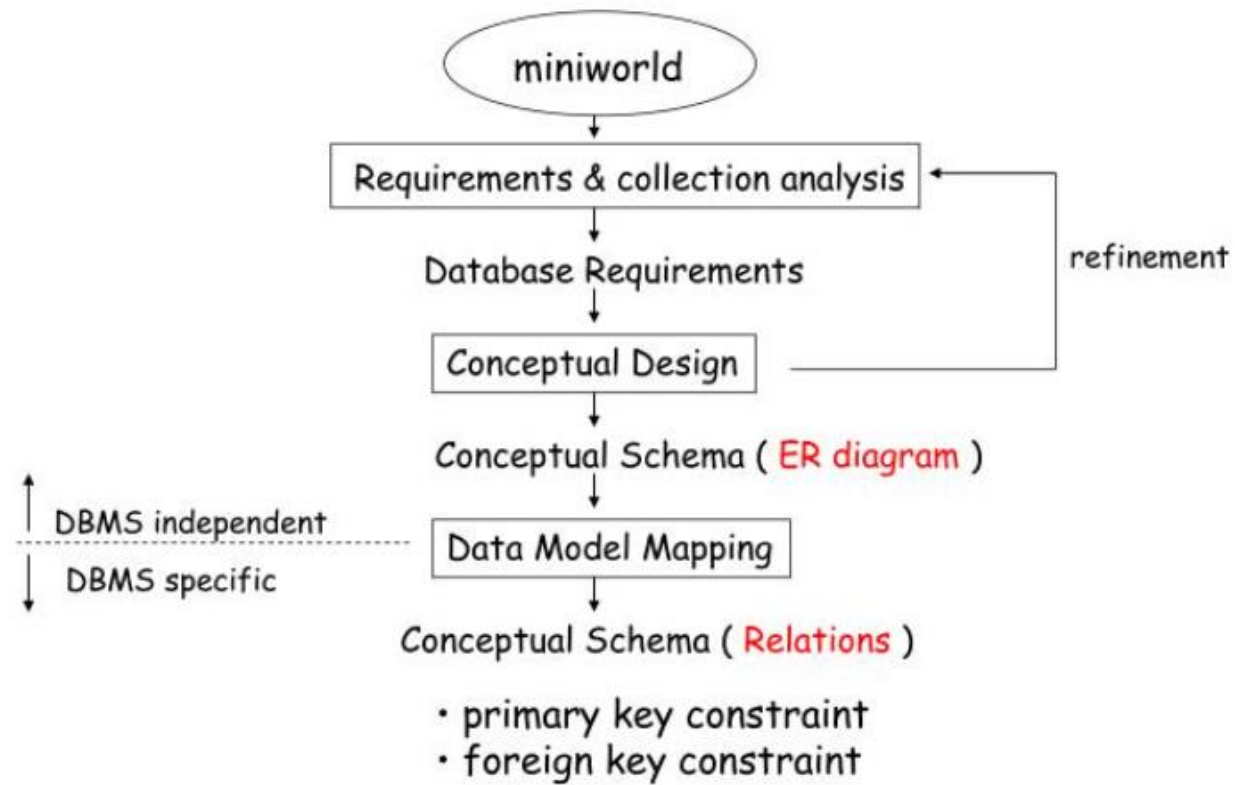
Rank			DBMS	Database Model	Score		
Aug 2020	Jul 2020	Aug 2019			Aug 2020	Jul 2020	Aug 2019
1.	1.	1.	Oracle +	Relational, Multi-model i	1355.16	+14.90	+15.68
2.	2.	2.	MySQL +	Relational, Multi-model i	1261.57	-6.93	+7.89
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	1075.87	+16.15	-17.30
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	536.77	+9.76	+55.43
5.	5.	5.	MongoDB +	Document, Multi-model i	443.56	+0.08	+38.99
6.	6.	6.	IBM Db2 +	Relational, Multi-model i	162.45	-0.72	-10.50
7.	↑ 8.	↑ 8.	Redis +	Key-value, Multi-model i	152.87	+2.83	+8.79
8.	↓ 7.	↓ 7.	Elasticsearch +	Search engine, Multi-model i	152.32	+0.73	+3.23
9.	9.	↑ 11.	SQLite +	Relational	126.82	-0.64	+4.10
10.	↑ 11.	↓ 9.	Microsoft Access	Relational	119.86	+3.32	-15.47
11.	↓ 10.	↓ 10.	Cassandra +	Wide column	119.84	-1.25	-5.37
12.	12.	↑ 13.	MariaDB +	Relational, Multi-model i	90.92	-0.21	+5.96
13.	13.	↓ 12.	Splunk	Search engine	89.91	+1.64	+4.03
14.	↑ 15.	↑ 15.	Teradata +	Relational, Multi-model i	76.78	+0.81	+0.14
15.	↓ 14.	↓ 14.	Hive	Relational	75.29	-1.14	-6.51

# Bancos de Dados Relacionais

# Banco de Dados Relacional

- Os dados são estruturados de acordo com o modelo relacional.
- SQL Server, Oracle, PostgreSQL, MySQL, DB2 e outros.
- Elementos básicos
  - Relações (tabelas) e registros (tuplas)
- Características fundamentais
  - ✓ Restrições de integridade
    - ✓ PK-primary key, FK-foreign key, UK-unique key, CK-check key, NN-not null
  - ✓ Normalização (formas normais)
  - ✓ Linguagem SQL (Structured Query Language)

# Modelagem



# Normalização

Normalização é uma ferramenta usada no projeto lógico que serve para reestruturar tabelas e atributos, reduzindo assim redundâncias e permitindo o correto crescimento do banco de dados.

O processo de normalização conta com 6 formas:

- 1º Forma Normal.
- 2º Forma Normal.
- 3º Forma Normal.
- FNBC (Forma normal de Boyce e Codd):
- 4º Forma Normal.
- 5º Forma Normal.

A partir da 3º Forma Normal diz-se que o banco de dados já se encontra normalizado. A FNBC, a 4FN e a 5FN são usadas para refinar ainda mais o banco.



# Normalização

Cod Cliente	Nome Cliente	Tel 1	Tel 2	Endereço	Cod Produto	Nome Produto	Preço	Quantidade
1	Marcio Duarte	2098837	3298889	Rua A	1122	YYY	50	2
1	Marcio Duarte	2098837	3298889	Rua A	3344	KKK	120	1
2	Vitor da Silva	5412324	5544123	Rua B	9987	PPP	30	7
3	André Magalhães	6574565	6521787	Rua C	3344	KKK	120	5
2	Vitor da Silva	5412324	5544123	Rua B	1122	YYY	50	1

## Problemas de Inserção:

Só é possível inserir um cliente se o mesmo adquirir um produto. Só é possível inserir um produto se algum cliente adquiri-lo.

## Problemas de alteração:

Para atualizar o telefone do cliente ou o preço do produto, todos os outros registros deverão ser atualizados, ou seja, dado redundante.

## Problemas de exclusão:

Se os produtos adquiridos por algum cliente forem excluídos, os dados cadastrais referente a esse cliente se perderão.

# Normalização

1FN → 2FN → 3FN

- ✓ **1ª Forma Normal (1FN):** toda relação deve ter uma chave primária e deve-se garantir que todo atributo seja atômico. Atributos compostos devem ser separados.

Tabela (cod\_cliente, nome\_cliente, tel1, tel2, endereco, cod\_produto, nome\_produto, preco, quantidade) ✗

Cliente (cod\_cliente, nome\_cliente, tel1, tel2, rua, bairro, cidade, estado) ✓

Produto (cod\_cliente, cod\_produto, nome\_produto, preco, quantidade) ✓

# Normalização

1FN → 2FN → 3FN

- ✓ 2ª Forma Normal (2FN): toda relação deve estar na 1FN e deve eliminar dependências funcionais parciais, ou seja, todo atributo não chave deve ser totalmente dependente da chave primária.

Cliente (cod\_cliente, nome\_cliente, tel1, tel2, rua, bairro, cidade, estado) ✓ (não possui chave primária composta)

Produto (cod\_cliente, cod\_produto, nome\_produto, preco, quantidade) ✗

cod\_produto → nome\_produto, preco (dependência parcial)

cod\_cliente, cod\_produto → quantidade (dependência total)

Resp: Produto (cod\_produto, nome\_produto, preco) ✓

Resp: Compra (cod\_cliente, cod\_produto, quantidade) ✓

# Normalização

1FN → 2FN → 3FN

- ✓ **3ª Forma Normal (3FN):** toda relação deve estar na 2FN e devem-se eliminar dependências funcionais transitivas, ou seja, todo atributo não chave deve ser mutuamente independente.

Carro (placa, modelo, km\_rodados, cod\_fabricante, nome\_fabricante) ✗

Placa, modelo → km\_rodados

Placa, modelo → cod\_fabricante

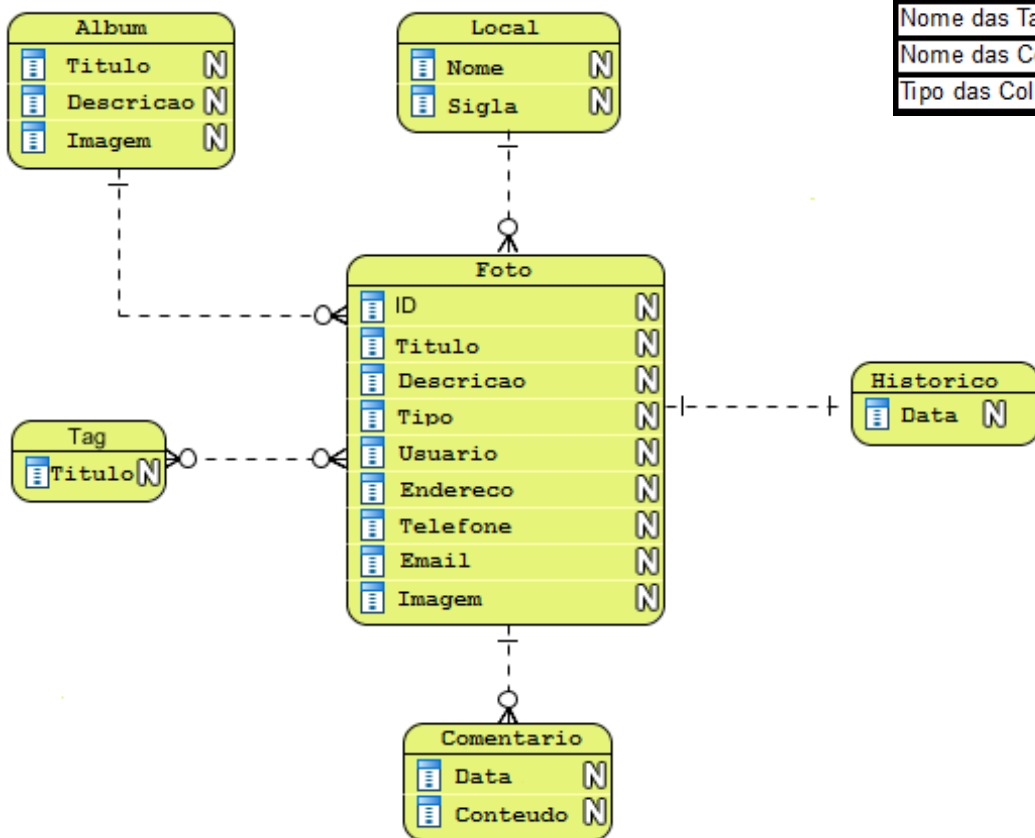
Placa, modelo → nome\_fabricante

Cod\_fabricante → nome\_fabricante

Carro (placa, modelo, kmRodados, cod\_fabricante) ✓

Fabricante (cod\_fabricante, nome\_fabricante) ✓

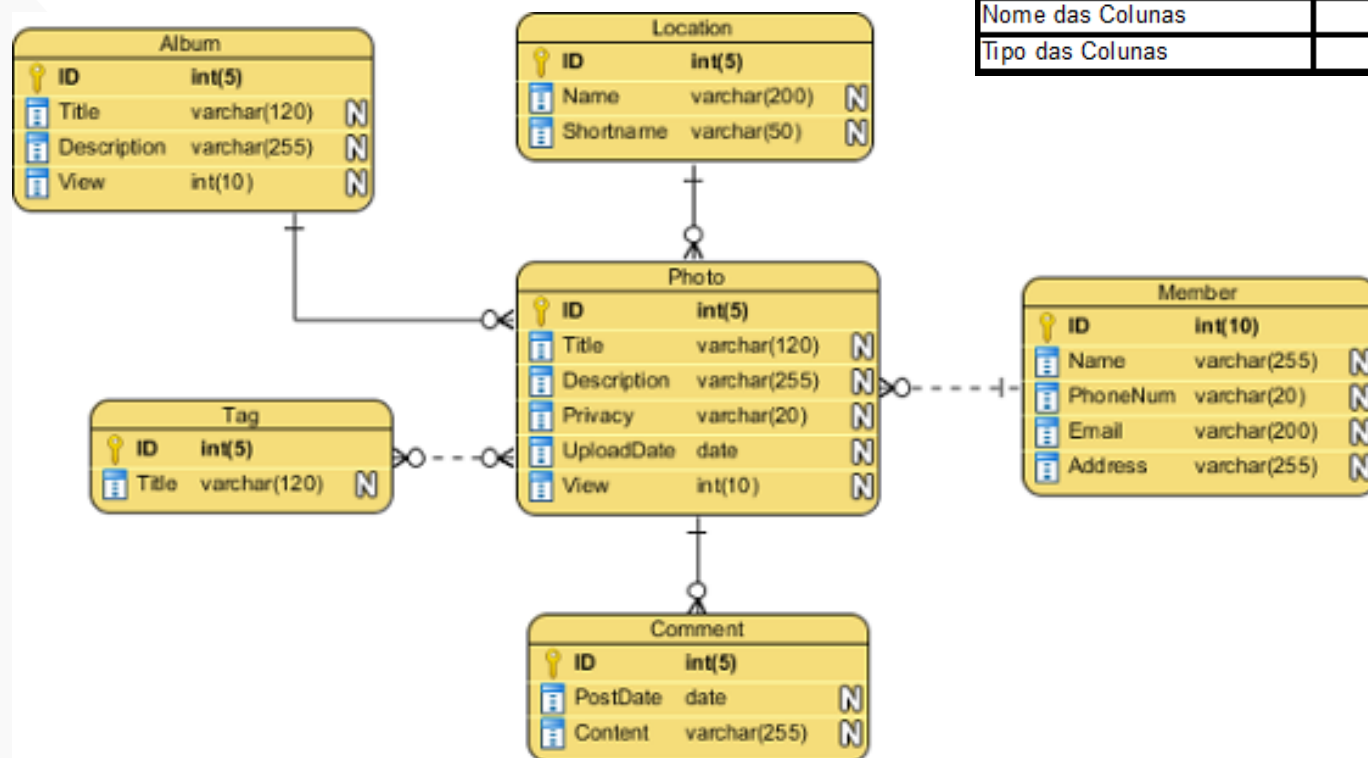
# Modelo Conceitual



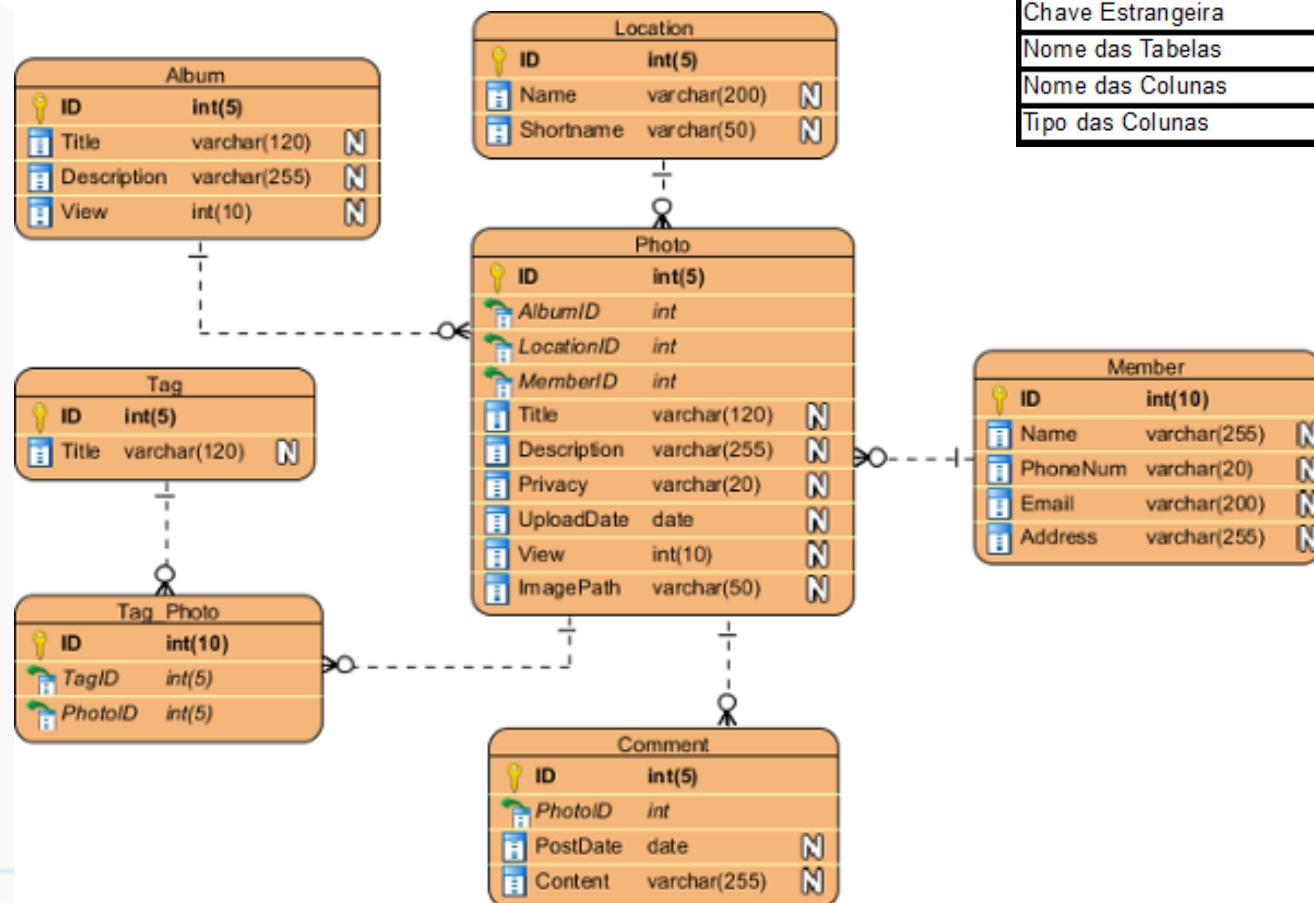
Característica	Conceitual	Lógico	Físico
Nome de Entidade	✓	✓	
Relacionamentos de Entidade	✓	✓	
Atributos	✓	✓	
Chave Primária		✓	✓
Chave Estrangeira		✓	✓
Nome das Tabelas			✓
Nome das Colunas			✓
Tipo das Colunas			✓

# Modelo Lógico

Característica	Conceitual	Lógico	Físico
Nome de Entidade	✓	✓	
Relacionamentos de Entidade	✓	✓	
Atributos	✓	✓	
Chave Primária		✓	✓
Chave Estrangeira		✓	✓
Nome das Tabelas			✓
Nome das Colunas			✓
Tipo das Colunas			✓



# Modelo Físico



Característica	Conceitual	Lógico	Físico
Nome de Entidade	✓	✓	
Relacionamentos de Entidade	✓	✓	
Atributos	✓	✓	
Chave Primária		✓	✓
Chave Estrangeira		✓	✓
Nome das Tabelas			✓
Nome das Colunas			✓
Tipo das Colunas			✓

# Bancos de Dados NoSQL



# NoSQL

NoSQL é um termo genérico que define **bancos de dados não-relacionais**.

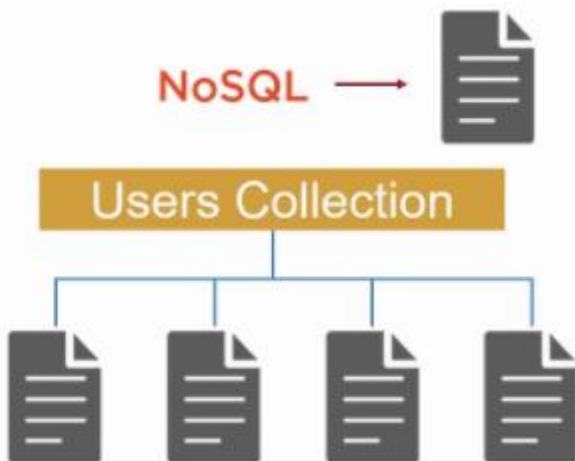
A tecnologia NoSQL foi iniciada por companhias líderes da Internet como Google, Facebook, Amazon e LinkedIn, para superar as limitações de banco de dados relacional para aplicações web modernas.

SQL

Table: Users

	id	name	createdAt	updatedAt
	Filter	Filter	Filter	Filter
1	1	Ashton	2018-08-12 1...	2018-08-12 1...
2	2	Jakeem	2018-08-12 1...	2018-08-12 1...
3	3	Uma	2018-08-12 1...	2018-08-12 1...

NoSQL →



# NoSQL

Não quer dizer que seus modelos não possuem relacionamentos e sim que ***não são orientados a tabelas.***

***Not Only SQL (não apenas SQL).***

Bancos de dados NoSQL são cada vez mais usados em ***Big Data e aplicações web de tempo real.***



# NoSQL - Características

- Escalabilidade horizontal.
- Ausência de esquema ou esquema flexível.
- Suporte a replicação.
- API simples.
- Nem sempre prima pela consistência.



# Alguns Bancos NoSQL

**Aerospike:** Banco de dados NoSQL que oferece *uma vantagem de velocidade de memória, atraindo empresas de anúncios de alta escala e aquelas que precisam de tempos de resposta em milissegundo*. Aerospike está apostando em novas categorias, incluindo jogos, e-commerce e segurança, onde a baixa latência é tudo.

**Apache Cassandra:** Os pontos fortes são a modelagem de dados NoSQL e *escalabilidade linear flexível em hardware* por conta do uso de cluster.

**Amazon DynamoDB:** foi desenvolvido pela Amazon para incrementar o seu e-commerce, possibilitando ter seus serviços altamente escaláveis. Inspirou o Cassandra, Riak, e outros projetos NoSQL.

# Alguns Bancos NoSQL

**MongoDB:** É o banco de dados mais popular NoSQL, com mais de sete milhões de downloads e centenas de milhares de implantações. ***Sua popularidade se deve à facilidade de desenvolvimento e manejo flexível dos dados.*** Muito utilizado em aplicações de redes sociais web e móvel.

**HBase:** É o banco de dados que roda em cima do ***HDFS (Hadoop Distributed File System – sistema de arquivos distribuído)***, por isso dá aos usuários a capacidade única de trabalhar diretamente com os dados armazenados no Hadoop. As características incluem grande escalabilidade.

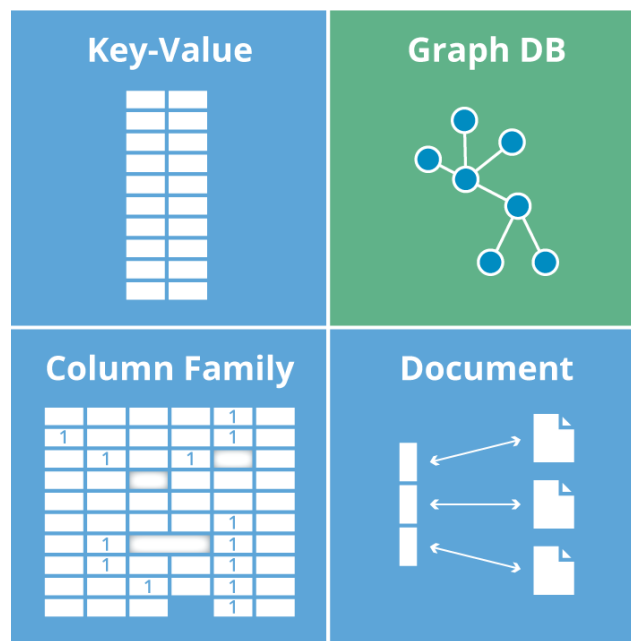
**Redis:** É o banco de dados NoSQL do tipo chave-valor mais conhecido. No mercado podemos encontrar diversas outras soluções que também são mecanismos de armazenamento baseado em chave-valor.

# Modelos de Bancos NoSQL

# Modelos de NoSQL

*Temos quatro categorias do NoSQL:*

- Chave-valor (key-Value)
- Orientado a Grafos
- Orientado a Coluna (Column Family)
- Orientado a Documentos



# Chave-Valor (Key-Value)

- Modelo mais simples.
- Permite a visualização do banco como uma grande tabela.
- Todo o banco é composto por um conjunto de chaves que estão associadas a um único valor.

Chave (Campo)	Valor (Instancia)
<b>Nome</b>	Hélio Rodrigues
<b>Idade</b>	45
<b>Sexo</b>	Masculino
<b>Fone</b>	99 99999999





# Orientado a Grafos

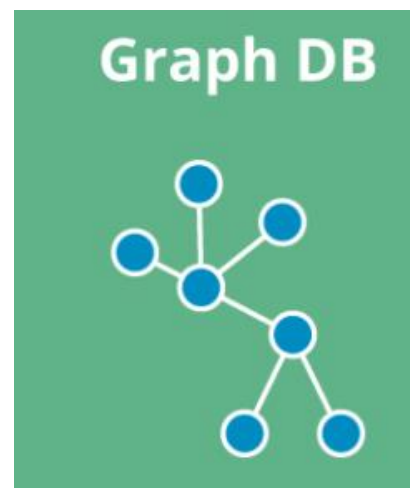
Este modelo possui três componentes básicos:

- Nós (vértices dos grafos).
- Os relacionamentos (arestas).
- As propriedades (conhecidos também como atributos).

É visto como um multigrafo rotulado e direcionado, onde cada par de nós ***pode ser conectado por mais de uma aresta.***

*A utilização deste modelo é muito útil quando é necessário fazer consultas demasiadamente complexas.*

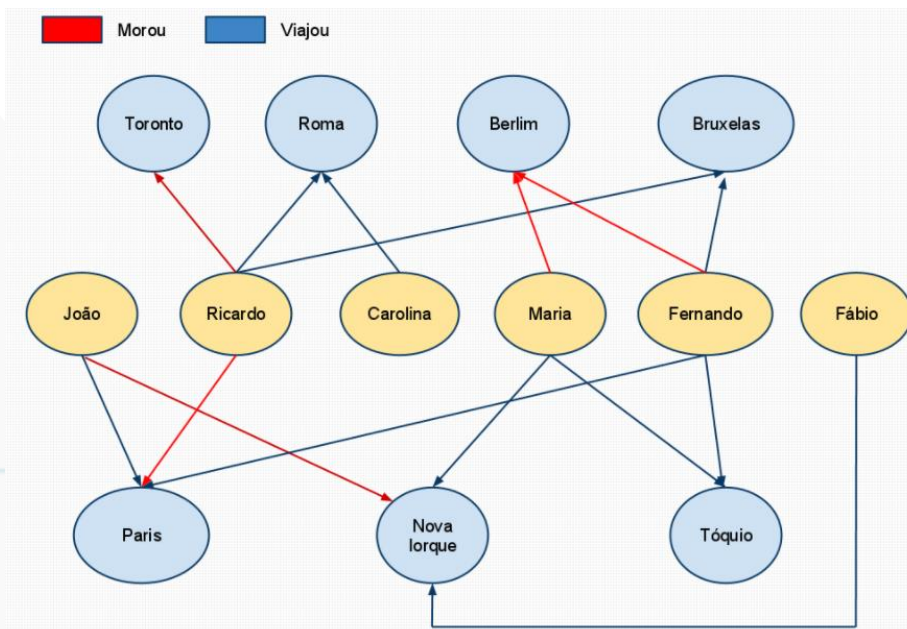
O modelo orientado a grafos possui uma alta performance, permitindo um bom desempenho nas aplicações.



# Orientado a Grafos

Imagine uma aplicação que mantém informações relativas à viagem. Uma consulta pertinente seria: “Quais cidades foram visitadas anteriormente por pessoas que foram para Nova Iorque?”

Temos diversas pessoas: João, Ricardo, Carolina, Maria, Fernando e Fábio que representam nós do grafo e estão conectadas a cidades que visitaram ou residiram.



# Orientado a Colunas

Este tipo de banco de dados *foi criado para armazenar e processar uma grande quantidade de dados distribuídos em diversas máquinas.*

Aqui existem *as chaves, mas neste caso, elas apontam para atributos ou colunas múltiplas.*

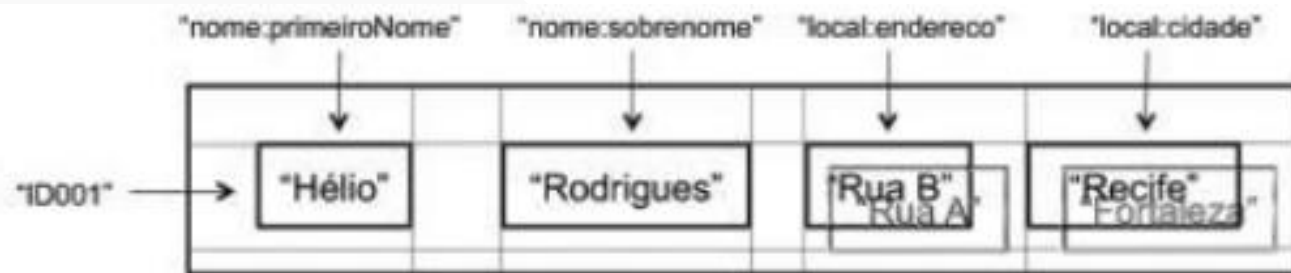
Os dados são indexados por uma tripla (coluna, linha e timestamp), *a coluna e linha são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado.*

Como o próprio nome sugere, as colunas *são organizadas por famílias de colunas.* Demonstra maior complexidade que o de chave-valor.

Column Family					
				1	
1				1	
	1		1		
				1	
	1			1	
	1			1	
		1		1	
				1	

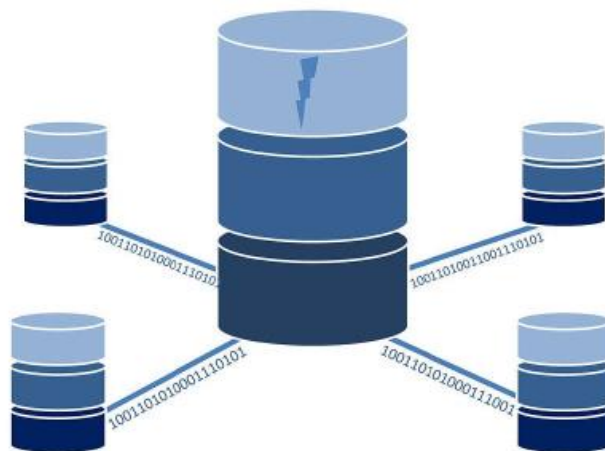
# Orientado a Colunas

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value



# Orientado a Documentos

- Armazenam chave/valor.
- O valor é um documento estruturado e indexado, com metadados.
- Valor (Documento), pode ser consultado.
- JSON: JavaScript Object Notation.
  - Feito para troca de dados.
  - Mais compacto e legível que XML.



# JSON

*É um acrônimo de JavaScript Object Notation*, e tem um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor.

```
1 {  
2   "id":1,  
3   "nome":"Alexandre Gama",  
4   "endereco":"R. Qualquer"  
5 }
```

# MongoDB

- ✓ Open source.
- ✓ Multiplataforma.
- ✓ Escalável.



## Relacional

**Banco de Dados**

**Tabela**

**Linha**

**Coluna**

## MongoDB

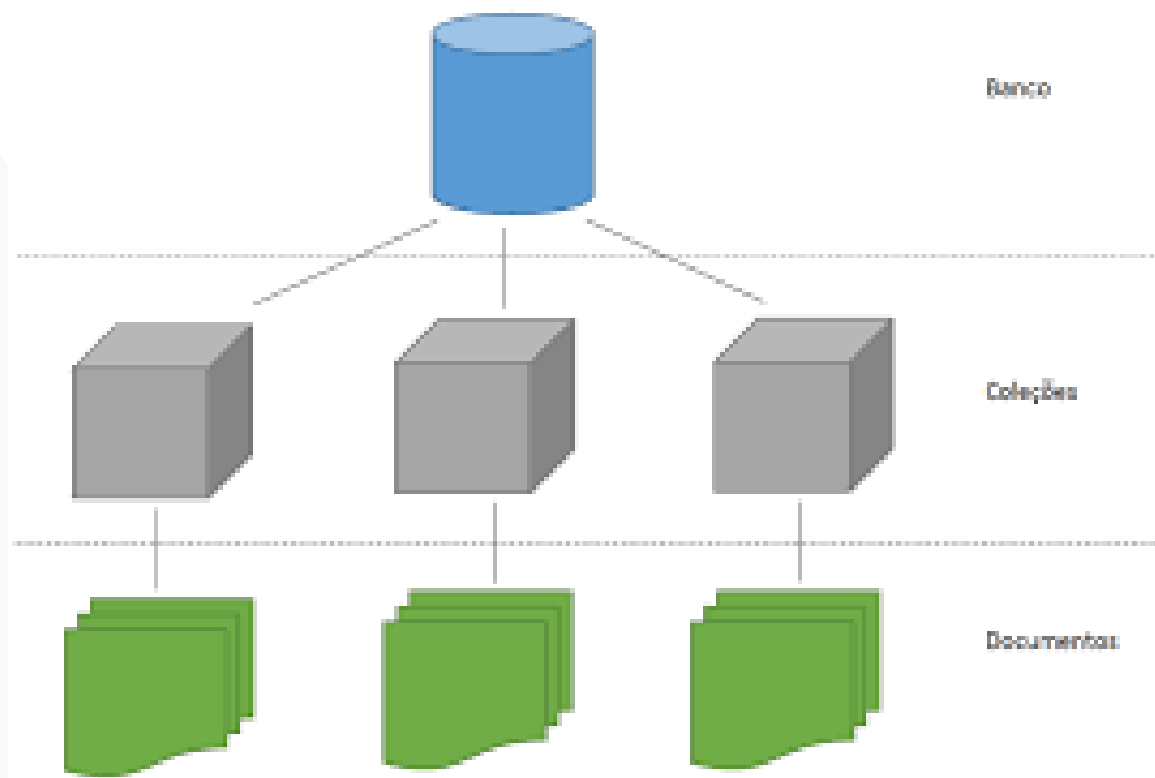
**Banco de Dados**

**Coleção**

**Documento**

**Campo**

# MongoDB - Definições





# Bancos de Dados NoSQL - Schema

# Ausência de Esquema

***Ausência de esquema (Schema-free) ou esquema flexível*** é uma outra característica em bancos de dados NoSQL.

Basicamente ***é a ausência parcial ou total de esquema que define a estrutura de dados.***

É justamente essa ausência de esquema que ***facilita uma alta escalabilidade*** e alta disponibilidade, ***mas em contrapartida não há a garantia de integridade dos dados, fato este que não ocorre no Modelo Relacional.***

# Ausência de Esquema

ID	Name	<u>IsActive</u>	Dob
1	John Smith	True	8/30/1964
2	Sarah Jones	False	2/18/2002
3	Adam Stark	True	7/13/1987

Document 1

```
{
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
}
```

Document 2

```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3

```
{
  "id": "3",
  "fullName":
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

# Bancos de Dados NewSQL

# NewSQL

*Pode ser definido como uma classe de SGBDs relacionais modernos que buscam fornecer o mesmo desempenho escalonável do NoSQL para cargas de trabalho OLTP e, simultaneamente, **garantir a conformidade ACID para transações como no RDBMS.***

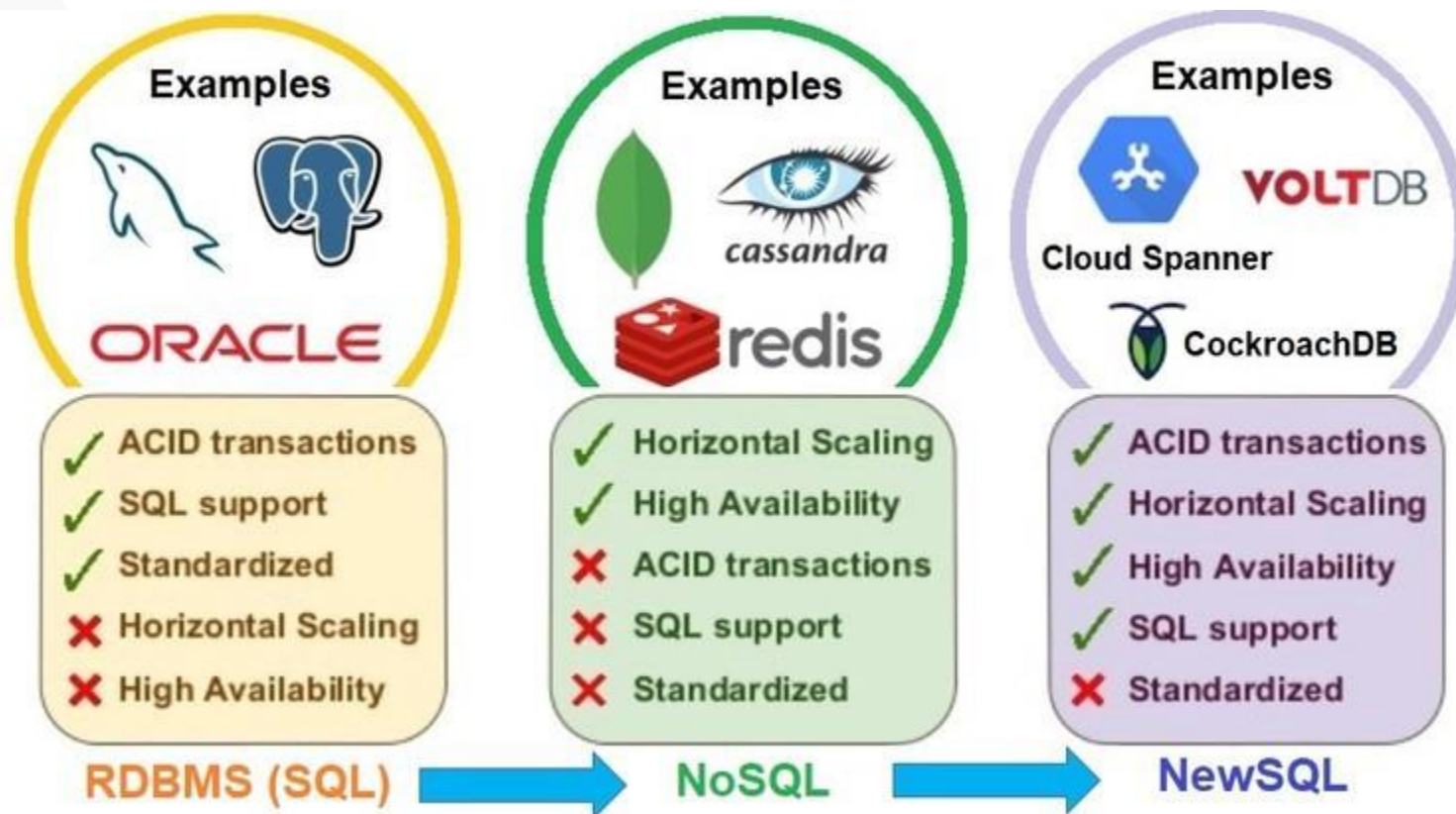
Basicamente esses sistemas desejam alcançar a escalabilidade do NoSQL sem ter que descartar o modelo relacional com SQL e suporte a transações do DBMS legado.



# SQL, NoSQL ou NewSQL

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

# SQL, NoSQL ou NewSQL



# Alguns Bancos NewSQL

**MemSQL:** Como o próprio nome sugere, *é operado em memória*, e é um sistema de banco de dados de alta escala por sua combinação de desempenho e compatibilidade com o SQL transacional e ACID na memória, adicionando uma interface relacional em uma camada de dados in-memory.

**VoltDB:** Projetado por vários pesquisadores de sistema de banco de dados bem conhecidos, esse banco *oferece a velocidade e a alta escalabilidade dos bancos de dados NoSQL, mas com garantias ACID*, e sua latência em milissegundo e *integração com Hadoop*.

**SQLFire:** Servidor de *banco de dados NewSQL da VMware*, desenvolvido para escalar em plataformas nas nuvens e tomar as vantagens de infraestrutura virtualizadas.

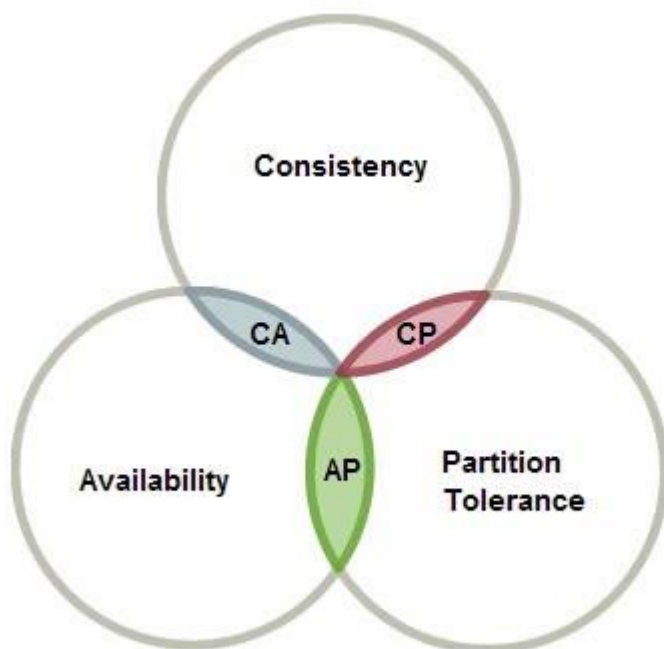
**MariaDB:** foi *desenvolvido pelo criador do MySQL* e é totalmente compatível com o MySQL. Também pode interagir com os bancos de dados NoSQL, como Cassandra e LevelDB.



# Teorema CAP

# Teorema CAP

De acordo com o teorema **CAP**, *um sistema distribuído de bancos de dados somente pode operar com dois desses comportamentos ao mesmo tempo*, mas *jamais com os três simultaneamente*.

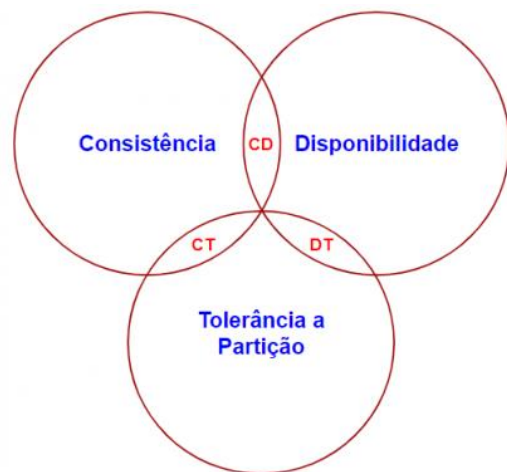


# Consistência Eventual

É um conceito interessante derivado do teorema CAP.

***O sistema prioriza as escritas de dados (armazenamento),*** sendo o sincronismo entre os nós do servidor realizado em um momento posterior – o que causa um pequeno intervalo de tempo no qual o sistema como um todo é inconsistente.

Para isso, ***são implementadas as propriedades Disponibilidade e Tolerância a Partição.***



# Consistência Eventual

Exemplos de sistemas de bancos de dados que implementam a consistência eventual são o ***MongoDB, Cassandra e RavenDB (bancos NoSQL)***, entre outros.

Em Bancos Relacionais, é muito comum implementar as propriedades **Consistência** e **Disponibilidade**. Como exemplos, citamos os SGBDRs ***Oracle, MySQL, PostgreSQL, SQL Server*** e outros.

Ao criar um banco de dados distribuído é importante ***ter em mente o teorema CAP.***

***Você terá de decidir se o banco será consistente ou disponível, pois bancos de dados distribuídos são sempre tolerantes a partição.***

# Escalabilidade

# Escalabilidade

- À medida em que o volume de dados cresce, aumenta-se a necessidade de escalabilidade e melhoria do desempenho.
- Podemos escalar **verticalmente** (adicionar CPU, memória e disco) ou podemos escalar **horizontalmente** (adicionar mais nós).



# Sistemas de Arquivos Distribuídos

# Sistemas de Arquivos

- Foram originalmente desenvolvidos como um **recurso do S.O** que fornece uma interface de programação conveniente para armazenamento em disco.
- **São responsáveis pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos.**
- Projetados para **armazenar e gerenciar um grande número de arquivos**, com recursos para criação atribuição de nomes e exclusão de arquivos.



# Sistemas de Arquivos

- Diretório é um arquivo de tipo especial;
- Fornece um mapeamento dos nomes textuais para identificadores internos;
- Podem incluir nomes de outros diretórios.

Tamanho do Arquivo
Horário de Criação
Horário de Acesso (Leitura)
Horário de Modificação (Escrita)
Horário de Alteração de Atributo
Contagem de Referência
Proprietário
Tipo de Arquivo
Lista de Controle de Acesso

# Sistemas de Arquivos Distribuídos (DFS ou SAD)

Um sistema de arquivos distribuídos *permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais*, possibilitando que os usuários acessem arquivos a partir de qualquer computador em uma rede.” (COULOURIS, et. al, p. 284).

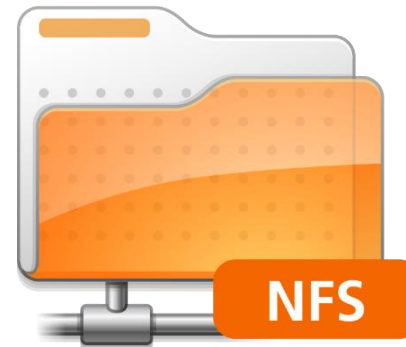
- **Objetivo:** permitir que os programas armazenem e acessem arquivos remotos exatamente como se fossem locais.
- **Permitem que vários processos** compartilhem dados por longos períodos, de modo seguro e confiável.
- **O desempenho e segurança** no acesso aos arquivos armazenados em um servidor devem ser compatíveis aos arquivos armazenados em discos locais.

# Requisitos de um Sistemas de Arquivos Distribuídos

- Transparência.
- Atualização concorrente de arquivos.
- Replicação de arquivos.
- Heterogeneidade.
- Tolerância a falha.
- Consistência.
- Segurança.
- Eficiência.

# Arquitetura do SAD

- Modelo abstrato de arquitetura que serve para o **Network File System (NFS)** e **Andrew File System (AFS)**.
- Divisão de responsabilidades entre três módulos.
  - Cliente.
  - Serviço de arquivos planos.
  - Serviço de diretórios.
- Design aberto
  - Diferentes módulos cliente podem ser utilizados para implementar diferentes interfaces.
  - Simulação de operações de arquivos de diferentes S.O.
  - Otimização de performance para diferentes configurações de hardware de clientes e servidores.



# Sistemas de Arquivos Distribuídos (DFS ou SAD)

Funções de um Sistema de Arquivos Distribuído:

- ***Armazenar e compartilhar programas e dados***
  - Funções idênticas às de um sistema centralizado (local).
- ***Ênfase na disponibilidade, confiabilidade e segurança.***
- ***Desempenho***
  - Questão importante porque acessos remotos podem ser significativamente mais lentos que os locais.
  - Não se pretende em geral que o SAD seja mais rápido que um SA local, mas sim que a degradação seja aceitável.

# Sistemas de Arquivos Distribuídos (DFS ou SAD)

Sistemas de arquivo distribuídos *devem ser vistos pelos clientes como um sistema de arquivo local.*

A *transparência* é muito importante para seu bom funcionamento.

É necessário *um bom controle de concorrência* no acesso.

*Cache é importante.*

# Apache Hadoop

# Hadoop

## Arquitetura HDFS



HDFS cluster possui 2 tipos de nodes:

Namenode (master node)  
Datanode (worker node)





# Apache Hadoop

Hadoop é uma *plataforma de software de código aberto para o armazenamento e processamento distribuído de grandes conjuntos de dados, utilizando clusters de computadores com hardware commodity.*

Os serviços do Hadoop fornecem armazenamento , processamento, acesso, governança, segurança e operações de Dados.

# Benefícios do Apache Hadoop

*Algumas das razões para se usar Hadoop é a sua “capacidade de armazenar, gerenciar e analisar grandes quantidades de dados estruturados e não estruturados de forma rápida, confiável, flexível e de baixo custo.*

- **Escalabilidade e desempenho** – distribuídos tratamento de dados local para cada nó em um cluster Hadoop permite armazenar, gerenciar, processar e analisar dados em escala petabyte.
- **Confiabilidade** – clusters de computação de grande porte são propensos a falhas de nós individuais no cluster. Hadoop é fundamentalmente resistente – quando um nó falha de processamento é redirecionado para os nós restantes no cluster e os dados são automaticamente re-replicado em preparação para falhas de nó futuras.

## Benefícios do Apache Hadoop

- **Flexibilidade** – ao contrário de sistemas de gerenciamento de banco de dados relacionais tradicionais, você não tem que esquemas estruturados criados antes de armazenar dados. ***Você pode armazenar dados em qualquer formato, incluindo formatos semi-estruturados ou não estruturados***, e em seguida, analisar e aplicar esquema para os dados quando ler.
- **Baixo custo** – ao contrário de software proprietário, ***o Hadoop é open source*** e é executado em hardware commodity de baixo custo.

# HDFS

O HDFS (Hadoop Distributed File System) é um sistema de arquivos distribuído, ***projetado para armazenar arquivos muito grandes***, com padrão de acesso aos dados streaming , utilizando clusters de servidores facilmente encontrados no mercado e de baixo ou médio custo.

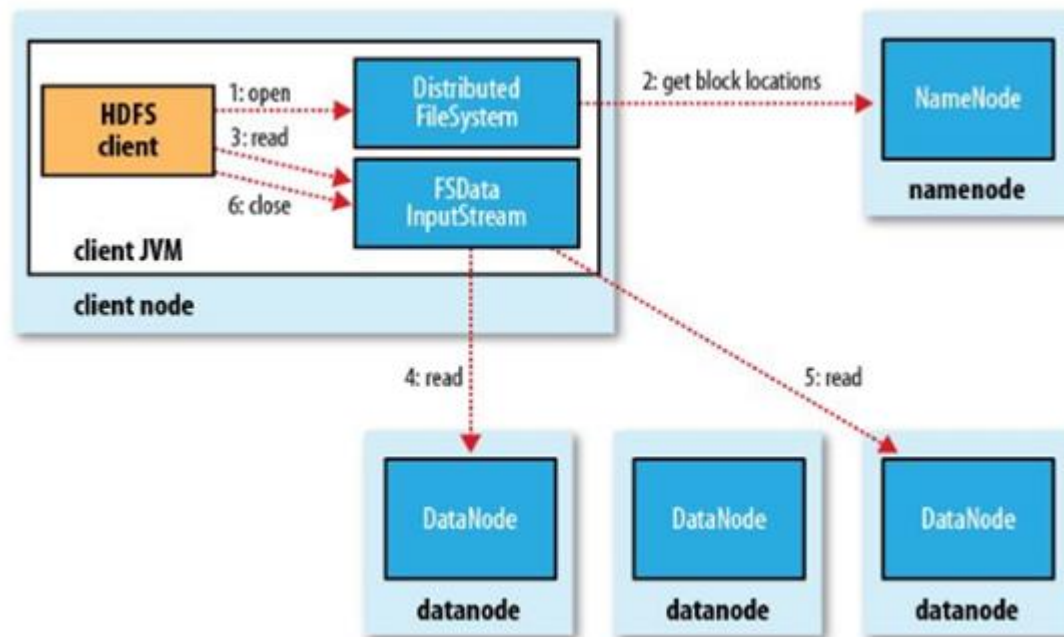
***Não deve ser utilizado*** para aplicações que precisem de acesso rápido a um determinado registro e sim para aplicações nas quais é necessário ler uma quantidade muito grande de dados.

Outra questão que deve ser observada é que ***não deve ser utilizado para ler muitos arquivos pequenos***, tendo em vista o overhead de memória envolvido.

# Recursos do HDFS

- O HDFS tem 2 tipos de Nós : **Master** (ou Namenode) e **Worker** (ou Datanode).
  - O **Master** armazena informações da distribuição de arquivos e metadados.
  - Já o **Worker** armazena os dados propriamente ditos. Logo o Master precisa sempre estar disponível. Para garantir a disponibilidade podemos ter um backup ( similar ao Cold Failover ) ou termos um Master Secundário em um outro servidor. Nesta segunda opção, em caso de falha do primário, o secundário pode assumir o controle muito rapidamente.
- Tal como um sistema Unix, é possível utilizar o HDFS via linha de comando.

# HDFS



# Apache Hadoop

- Hadoop é uma plataforma de software em Java de computação distribuída voltada para clusters e processamento de grandes volumes de dados, com atenção a tolerância a falhas.
- Foi inspirada no MapReduce (um modelo de programação paralela para processamento largamente distribuído de grandes volumes de dados proposto primeiramente pela empresa Google) e no GoogleFS (Google File System é um sistema de arquivos distribuído proprietário desenvolvido pelo Google para fornecer acesso eficiente e confiável aos dados usando grandes clusters de hardware comum).
- É disponibilizado pela Amazon e IBM em suas plataformas.



# Módulos do Framework do Apache Hadoop

- **Hadoop Common** - Contém as bibliotecas e arquivos comuns e necessários para todos os módulos Hadoop.
- **Hadoop Distributed File System (HDFS)** - Sistema de arquivos distribuído que armazena dados em máquinas dentro do cluster, sob demanda, permitindo uma largura de banda muito grande em todo o cluster.
- **Hadoop Yarn** - Trata-se de uma plataforma de gerenciamento de recursos responsável pelo gerenciamento dos recursos computacionais em cluster, assim como pelo agendamento dos recursos.
- **Hadoop MapReduce** - Modelo de programação para processamento em larga escala.



# Computação em Nuvem

# Computação em Nuvem

Nós temos três principais fornecedores de computação em nuvem, **AWS**, **Microsoft Azure** e **Google Cloud**, que possuem pontos fortes e fracos que os tornam ideais para diferentes cenários.



# Arquiteturas Monolíticas

***Com as arquiteturas monolíticas, todos os processos são altamente acoplados e executam como um único serviço.***

Isso significa que se um processo do aplicativo apresentar um pico de demanda, toda a arquitetura deverá ser escalada. A complexidade da adição ou do aprimoramento de recursos de aplicativos monolíticos aumenta com o crescimento da base de código. Essa complexidade limita a experimentação e dificulta a implementação de novas ideias.

***As arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo.***

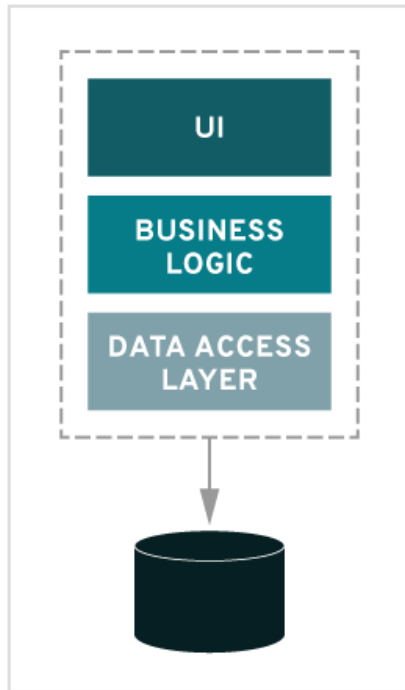
# Arquitetura de Microserviços

*Com uma arquitetura de microserviços, um aplicativo é criado como componentes independentes que executam cada processo do aplicativo como um serviço.*

Esses serviços se comunicam por meio de uma interface bem definida usando APIs leves. Os serviços são criados para recursos empresariais e cada serviço realiza uma única função. Como são executados de forma independente, cada serviço pode ser atualizado, implantado e escalado para atender a demanda de funções específicas de um aplicativo.

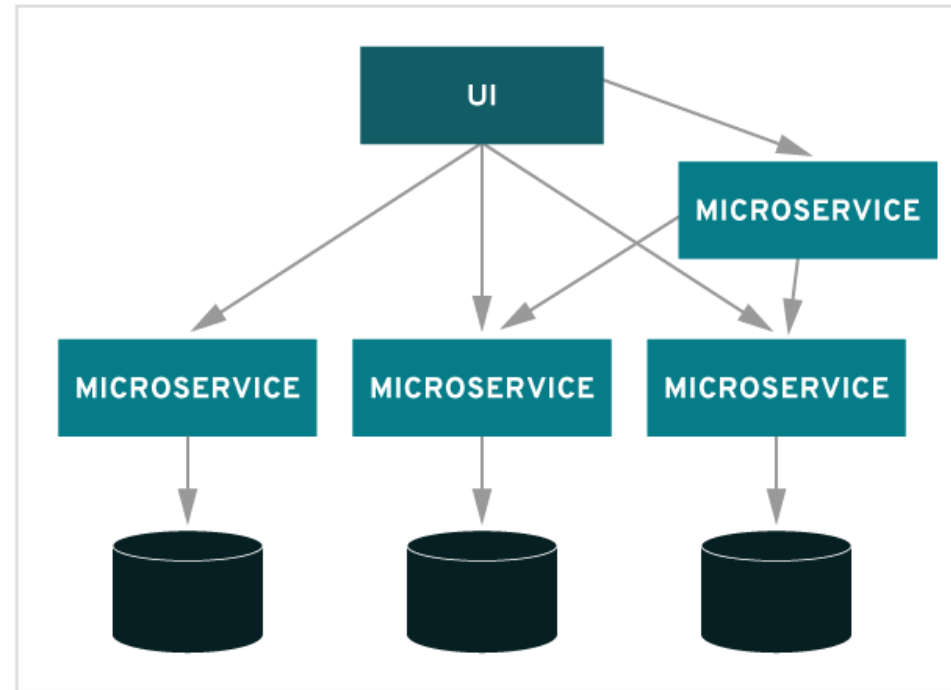
# Arquitetura

## MONOLITHIC



VS.

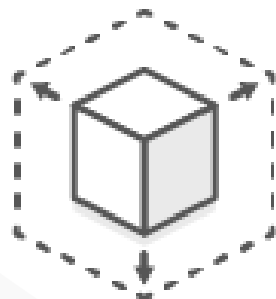
## MICROSERVICES



# Arquitetura de Microserviços

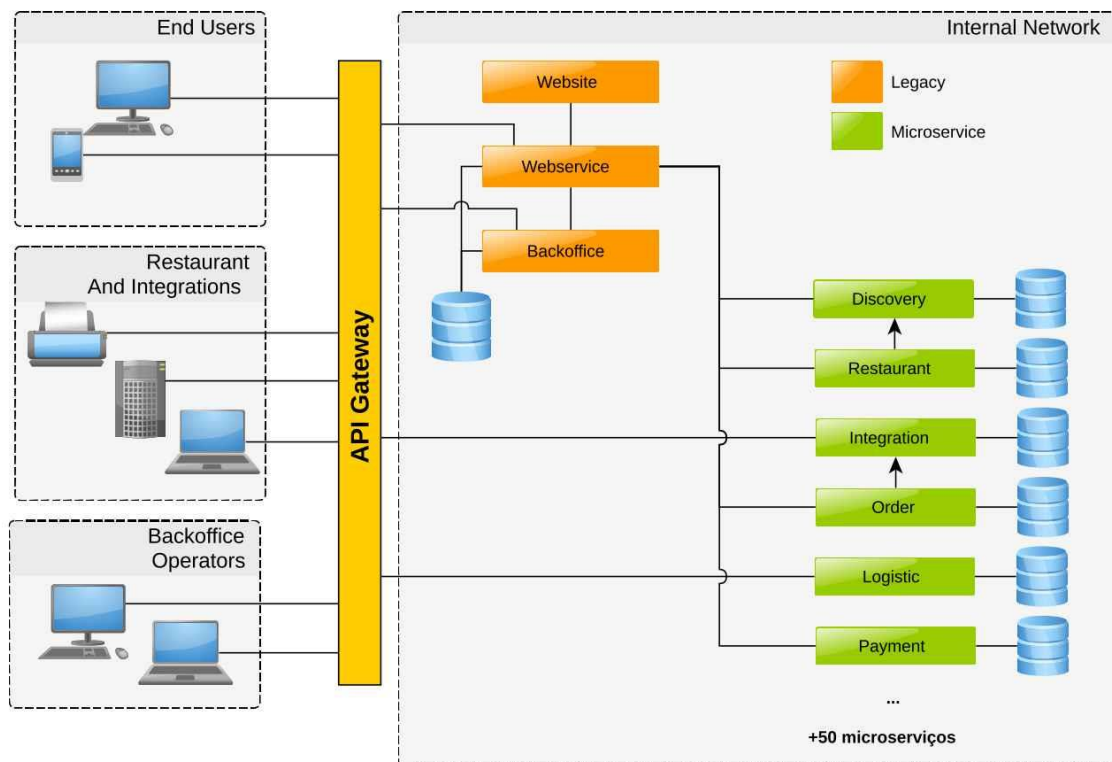
## Escalabilidade flexível

Os microserviços permitem que cada serviço seja escalado de forma independente para atender à demanda do recurso de aplicativo oferecido por esse serviço. Isso permite que as equipes dimensionem corretamente as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade quando um serviço experimenta um pico de demanda.



# Arquitetura

## Arquitetura de micro-serviços



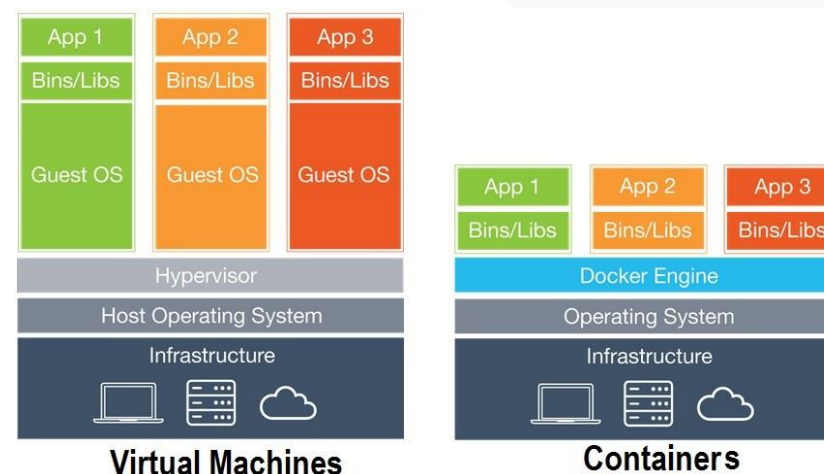
# Containers



# Container

*Em vez de usar um sistema operacional para cada estrutura, como na virtualização, os Containers são blocos de espaços divididos pelo Docker em um servidor, o que possibilita a implementação de estruturas de Microserviços que compartilham o mesmo sistema operacional. Porém, de forma limitada (conforme a demanda por capacidade).*

O fato de os Containers não terem seus próprios sistemas operacionais, permite que eles consumam menos recursos e, com isso, sejam mais leves.



# Ferramentas de Orquestração de Containers

*As ferramentas de orquestração de containers são aplicações em nuvem que permitem fazer o gerenciamento de múltiplos contêineres.*

Seus principais objetivos são:

- Cuidar do ciclo de vida dos containers de forma autônoma, subindo e distribuindo, conforme nossas especificações ou demandas;
- Gerenciar volumes e rede, que podem ser local ou no cloud provider de sua preferência.

# Orquestradores de Containers

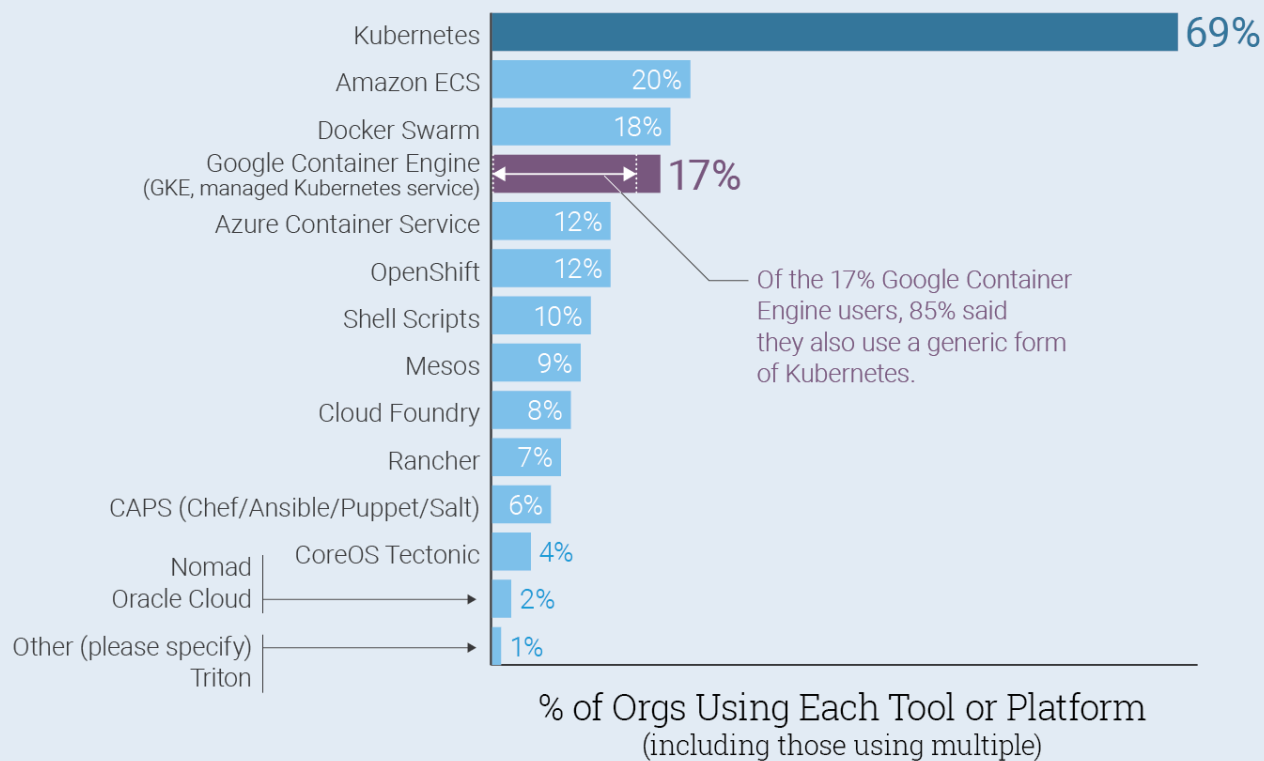
O Kubernetes, ECS e o Docker são as principais plataformas de gerenciamento de contêineres.

***Dessa forma, essa é uma ferramenta para viabilizar a utilização de Containers e Microserviços em servidores com mais facilidade, pois permite empacotar os aplicativos para que possam ser movimentados facilmente.***

O Docker permite, por exemplo, que uma biblioteca possa ser instalada em diferentes Containers sem que haja qualquer interdependência entre eles. Essa característica tem o objetivo de facilitar o gerenciamento de códigos e aplicativos.

# Orquestradores de Containers

## Kubernetes Manages Containers at 69% of Organizations Surveyed



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.  
Q. Your organization manages containers with... (check all that apply)? n=763.

# Arquiteturas com Alta Disponibilidade (HA)

# Alta Disponibilidade

- Alta disponibilidade: Refere-se a um conjunto de tecnologias que minimiza as interrupções de TI proporcionando a continuidade dos negócios de serviços de TI por meio de componentes redundantes e tolerantes a falhas ou protegidos contra failover no mesmo data center.

# Sistemas Distribuídos

Os sistemas distribuídos são recursos computacionais compartilhados em uma rede permitindo um aumento no desempenho, tolerância a falhas e escalabilidade do sistema.

A distinção fundamental é que em um sistema distribuído, uma coleção de computadores independentes mostra-se aos usuários como sendo um sistema único.

# Sistemas de Arquivos Distribuídos

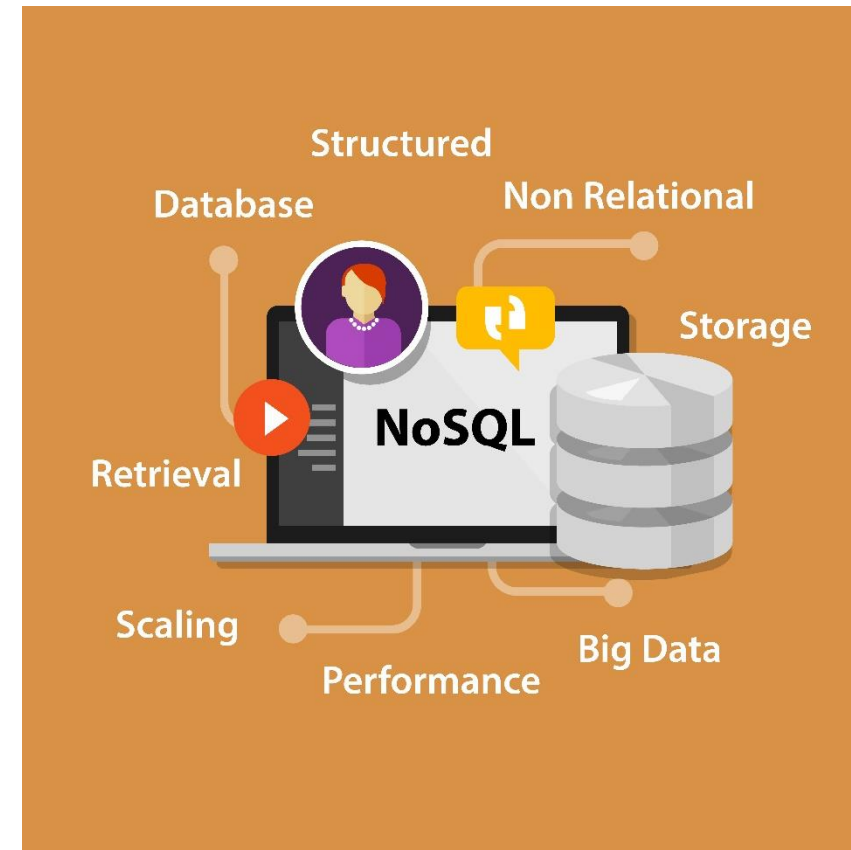
Sistema de arquivo distribuído (SAD) permite os programas armazenar e acessar arquivos remotos exatamente como se fosse um acesso local, permitindo o acesso a partir de qualquer computador em uma rede.

A **replicação** é o segredo da eficácia dos sistemas distribuídos, pois ela é a chave para prover melhor desempenho, alta disponibilidade e tolerância a falhas.



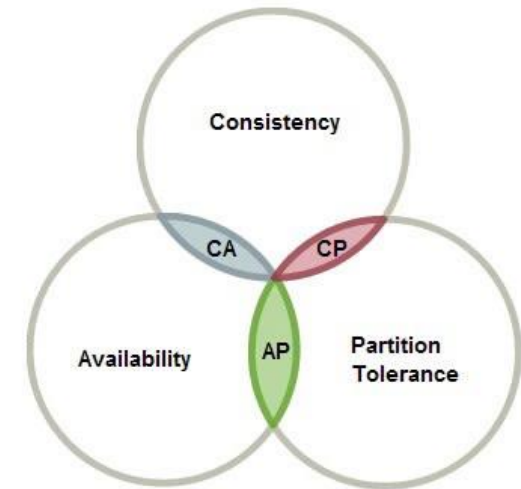
# Banco de Dados NoSQL

- Escalabilidade horizontal.
- Ausência de esquema ou esquema flexível.
- Suporte à replicação.
- API simples.
- Nem sempre prima pela consistência.



# Teorema CAP

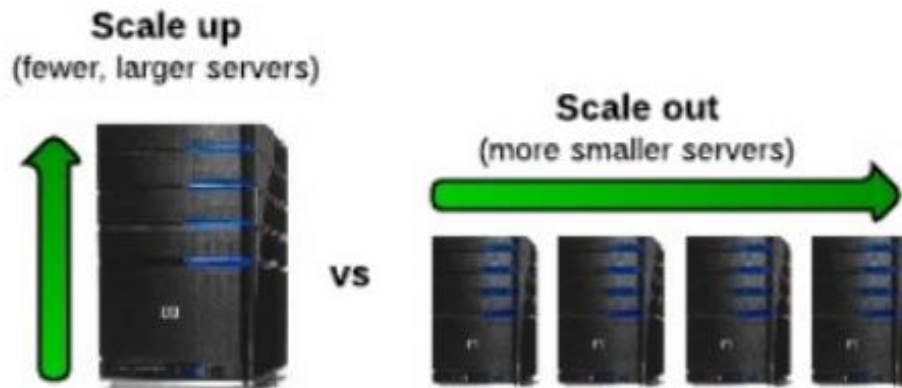
- De acordo com o teorema **CAP**, *um sistema distribuído de bancos de dados somente pode operar com dois desses comportamentos ao mesmo tempo*, mas *jamais com os três simultaneamente*.



# Arquiteturas com Alta Disponibilidade (HA) – Exemplo MongoDB

# Escalabilidade

- À medida em que o volume de dados cresce, aumenta-se a necessidade de escalabilidade e melhoria do desempenho.
- Podemos escalar **verticalmente** (adicionar CPU, memória e disco) ou podemos escalar **horizontalmente** (adicionar mais nós).



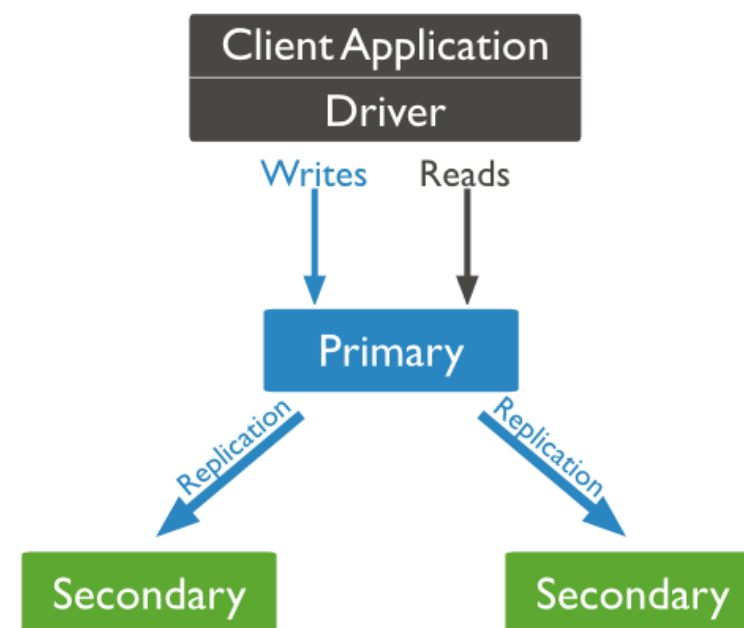
# Sistemas de Arquivos Distribuídos

Na busca de sistemas mais confiáveis, alguns meios foram desenvolvidos para oferecer mais confiança aos sistemas, entre eles está a **tolerância a falhas**. Dessa forma, é essencial para o sistema de arquivos distribuídos que **continuem a funcionar diante de falhas que aconteçam em servidores**.

A **replicação** é o método utilizado para **aumentar a disponibilidade de um serviço de arquivos**, onde os arquivos são armazenados em dois ou mais servidores e caso um deles não esteja disponível, outro servidor poderá fornecer os serviços solicitados.

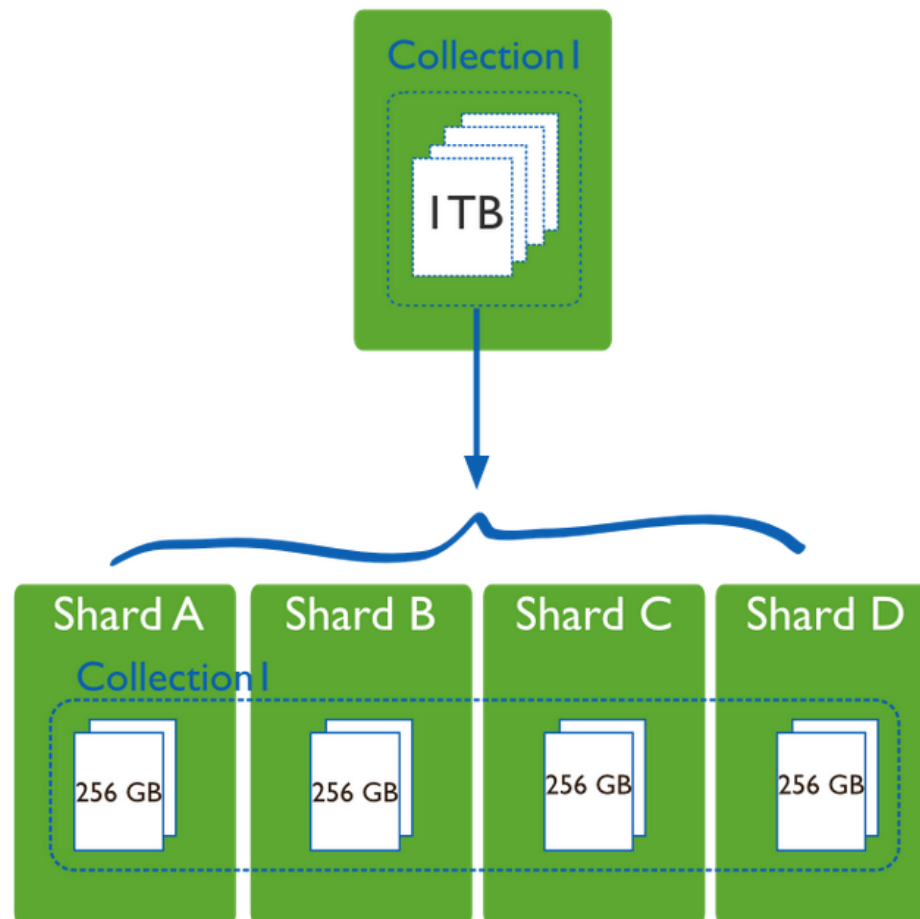
# Arquitetura MongoDB

- Um conjunto de réplicas no MongoDB é um grupo de instâncias mongod que mantêm o mesmo conjunto de dados. Ele contém vários nós de suporte de dados e, opcionalmente, um nó de árbitro.
- O nó primário recebe todas as operações de leitura e gravação e registra tudo nos seus conjuntos de dados e logs.
- Os secundários replicam o log do primário e aplicam as operações a seus conjuntos de dados de forma assíncrona.
- Tendo os conjuntos de dados secundários que refletem o conjunto de dados do primário, o conjunto de réplicas pode continuar a funcionar, apesar da falha de um ou mais membros.
- Se o nó primário não estiver disponível, um secundário elegível fará uma eleição para se eleger como o novo nó primário.

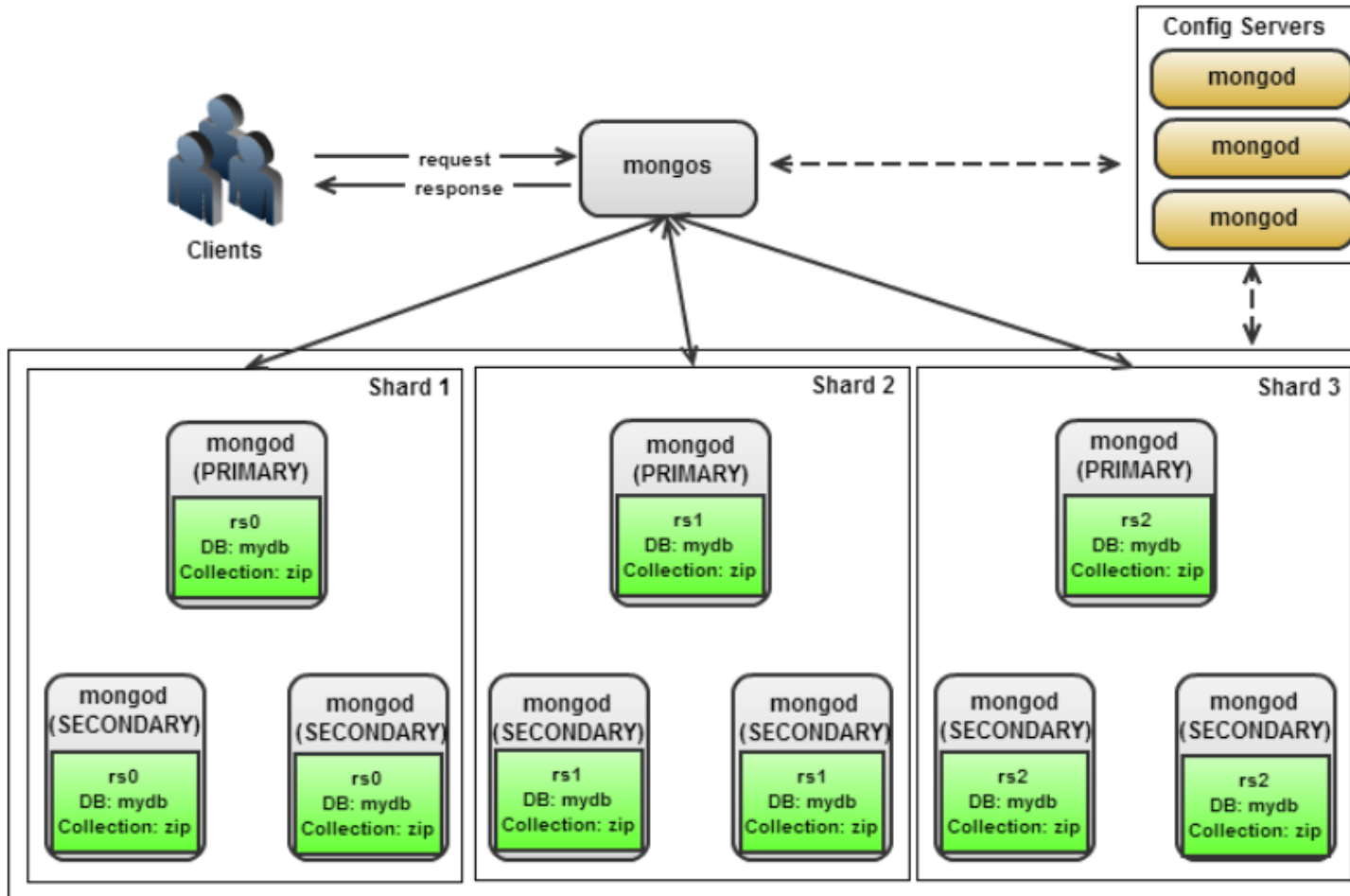


# Sharding

- Um cluster (agrupamento) no mongodb é um agrupamento fragmentado:
  - Escale leituras e gravações ao longo de vários nós.
  - Cada nó não lida com todos os dados, portanto, você pode separar os dados ao longo de todos os nós do fragmento.



# Arquitetura usando Sharding e ReplicaSet.





# Blocos Funcionais em uma Arquitetura de Dados

# Tipos de Sistemas Gerenciadores de Dados

- **Data Warehouses:** um Data Warehouse agrega dados de diferentes fontes de dados relacionais em uma empresa em um repositório único, central e consistente.
- **Data Marts:** um Data Mart é uma versão focada de um Data Warehouse que contém um subconjunto menor de dados importantes e necessários para uma única equipe ou um grupo seleto de usuários dentro de uma organização, como o departamento de RH.
- **Data Lakes:** enquanto os Data Warehouses armazenam dados processados, um Data Lake armazena dados brutos, normalmente petabytes deles. Um Data Lake pode armazenar dados estruturados e não estruturados, o que o torna exclusivo de outros repositórios de dados.

# Tipos de Arquitetura de Dados

- **Data Fabrics:** é uma arquitetura que se concentra na automação da integração de dados, engenharia de dados e governança em uma cadeia de valor de dados entre provedores de dados e consumidores de dados. Uma malha de dados é baseada na noção de “metadados ativos” que usa gráfico de conhecimento, semântica, mineração de dados e tecnologia de aprendizado de máquina (ML) para descobrir padrões em vários tipos de metadados (por exemplo, logs do sistema, social, etc.). Em seguida, aplica esse insight para automatizar e orquestrar a cadeia de valor dos dados. Por exemplo, ele pode permitir que um consumidor de dados encontre um produto de dados e, em seguida, tenha esse produto de dados provisionado para ele automaticamente.

# Tipos de Arquitetura de Dados

- **Data Meshes:** uma malha de dados é uma arquitetura de dados descentralizada que organiza os dados por domínio de negócios. Usando uma malha de dados, a organização precisa parar de pensar nos dados como um subproduto de um processo e começar a pensar neles como um produto por si só. Os produtores de dados atuam como proprietários de produtos de dados. Como especialistas no assunto, os produtores de dados podem usar sua compreensão dos principais consumidores dos dados para projetar APIs para eles. Essas APIs também podem ser acessadas de outras partes da organização, fornecendo acesso mais amplo aos dados gerenciados.

# Benefícios de Arquitetura de Dados

- **Reduzindo a redundância:** pode haver campos de dados sobrepostos em diferentes fontes, resultando em risco de inconsistência, imprecisões de dados e oportunidades perdidas de integração de dados. Uma boa arquitetura de dados pode padronizar como os dados são armazenados e potencialmente reduzir a duplicação, permitindo análises holísticas e de melhor qualidade.
- **Melhorando a qualidade dos dados:** Arquiteturas de dados bem projetadas podem resolver alguns dos desafios de Data Lakes mal gerenciados, também conhecidos como “pântanos de dados”. Um pântano de dados (DATA SWAMP) carece de qualidade de dados apropriada e práticas de governança de dados para fornecer aprendizados perspicazes.

# Benefícios de Arquitetura de Dados

- **Habilitando a integração:** os dados geralmente ficam isolados, como resultado de limitações técnicas no armazenamento de dados e barreiras organizacionais dentro da empresa. As arquiteturas de dados de hoje devem ter como objetivo facilitar a integração de dados entre domínios, para que diferentes geografias e funções de negócios tenham acesso aos dados uns dos outros.
- **Gerenciamento do ciclo de vida dos dados:** uma arquitetura de dados moderna pode abordar como os dados são gerenciados ao longo do tempo.

# Características de uma Arquitetura Moderna de Dados

- **Cloud-native e cloud-enabled**, para que a arquitetura de dados possa se beneficiar do dimensionamento elástico e da alta disponibilidade da nuvem.
- **Pipelines de dados robustos, escaláveis e portáteis**, que combinam fluxos de trabalho inteligentes, análises cognitivas e integração em tempo real em uma única estrutura.
- **Integração de dados perfeita**, usando interfaces de API padrão para conectar-se a aplicativos legados.
- **Habilitação de dados em tempo real**, incluindo validação, classificação, gerenciamento e governança.
- **Desacoplado e extensível**, portanto, não há dependências entre os serviços e os padrões abertos permitem a interoperabilidade.
- **Otimizado** para equilibrar custo e simplicidade.

# Camadas em uma Arquitetura de Dados



# Arquitetura de Dados

**Arquitetura de dados é...**

**“...a prática de examinar a estratégia empresarial e identificar os principais pontos de integração de dados que precisam ser ativados para executar essa estratégia, e estabelece um roteiro para criar os principais recursos para entregar essas integrações, para que as empresas possam aproveitar os dados como um ativo estratégico.”**

**“...a ciência de avaliar onde estão os dados da sua empresa e a arte de projetar a melhor maneira futura de armazenar dados em toda a empresa.”**

**“...a disciplina para ajudar as empresas a gerenciar seus dados da maneira mais eficaz e maneira segura, compatível e lucrativa.”**

# Arquitetura de Dados

**Arquitetura de dados é...**

**“...tudo que uma empresa faz para garantir que as informações sejam precisas e disponíveis para facilitar propósitos comerciais válidos, conforme definido por regulamentos/legais requisitos, proposições de valor das partes interessadas e clientes (internos/externo).”**

**“...um conjunto integrado de artefatos de especificação que definem requisitos de dados estratégicos, orienta a integração de ativos de dados e alinha investimentos de dados com a estratégia de negócio.**

**“...o projeto que analisa todo o cenário de dados e governa o ciclo de vida dos dados. Descreverá a linguagem a ser usada pelas partes interessadas. Inclui as pessoas, processos, ferramentas, tecnologias e artefatos necessários”.**

# Arquitetura de Dados

**Arquitetura de dados é...**

**“...regras, políticas e padrões que regem:**

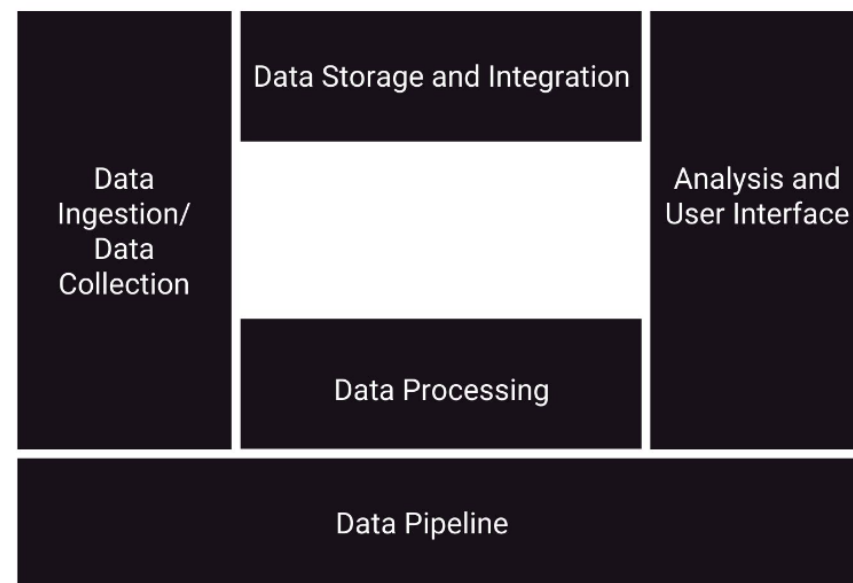
- 1. Dados e documentação de requisitos para sistemas;**
- 2. Projeto de banco de dados e modelagem;**
- 3. Documentação, gestão e disseminação/acesso de metadados;**
- 4. Coleta, uso e integração de dados em toda a empresa;**
- 5. Qualidade dos dados, definição e aplicação da Governança de Dados;**
- 6. Conformidade e retenção de arquivamento de dados - LGPD;**
- 7. Gerenciamento de Dados Mestres.”**

# Camadas em uma Arquitetura de Dados

As diferentes camadas da arquitetura da plataforma de dados incluem:

- Camada de ingestão de dados.
- Camada de armazenamento de dados.
- Camada de processamento e análise de dados.
- Camada de interface do usuário.
- Camada de pipeline de dados.

Layers of the Data Platform Architecture



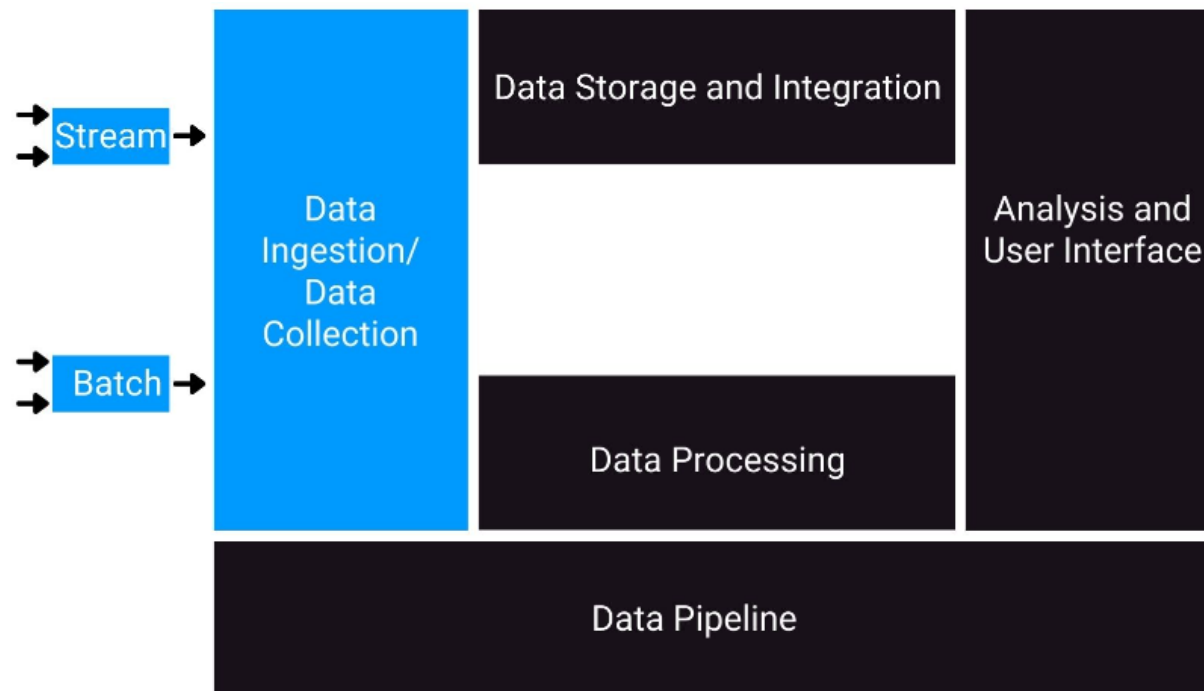
Fonte: <https://www.analyticsvidhya.com>

# Camada de Ingestão / Coleta de Dados

# Ingestão / Coleta de Dados

Esta é a primeira camada da arquitetura da plataforma de dados.

A camada de coleta de dados, como o nome sugere, é responsável por conectar-se aos sistemas de origem e trazer dados para a plataforma de dados de maneira periódica.



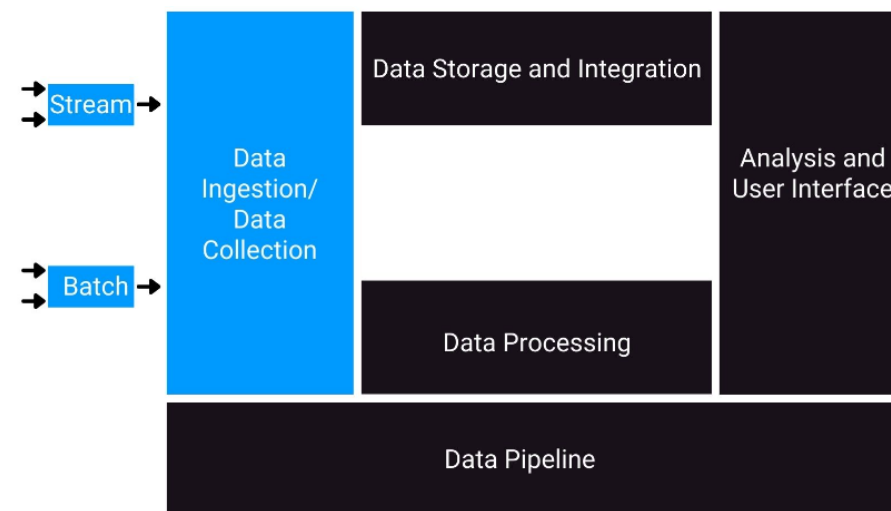
Fonte: <https://www.analyticsvidhya.com>

# Ingestão / Coleta de Dados

Esta camada executa as seguintes tarefas:

- É responsável pela conexão com as fontes de dados.
- É responsável pela transferência de dados das fontes de dados para a plataforma de dados no modo streaming, no modo batch ou em ambos.
- É responsável por manter as informações sobre os dados coletados no repositório de metadados.

Por exemplo, quantos dados são coletados na plataforma de dados e outras informações descritivas?



Fonte: <https://www.analyticsvidhya.com>

# Ingestão / Coleta de Dados

Existem várias ferramentas disponíveis no mercado.

Como ferramentas de podemos citar:

- Google Cloud Data Flow
- IBM Streams
- Amazon Kinesis
- Apache Kafka

São algumas das ferramentas mais usadas para ingestão de dados que suportam modos batch e streaming.



Cloud  
DataFlow



Fonte: <https://www.analyticsvidhya.com>



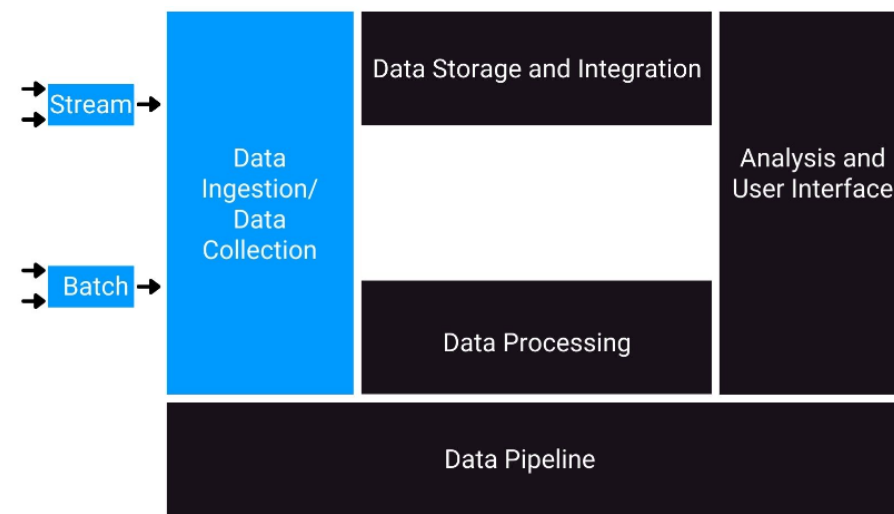
# Batch e Stream

# Batch e Stream

**Qual é a diferença entre batch e stream?**

**O batch é um lote de pontos de dados que foram agrupados em um intervalo de tempo específico. Outro termo frequentemente usado para isso é uma janela de dados.**

**Já o processamento de dados em stream lida com dados contínuos e é essencial para transformar de grandes a rápidos.**

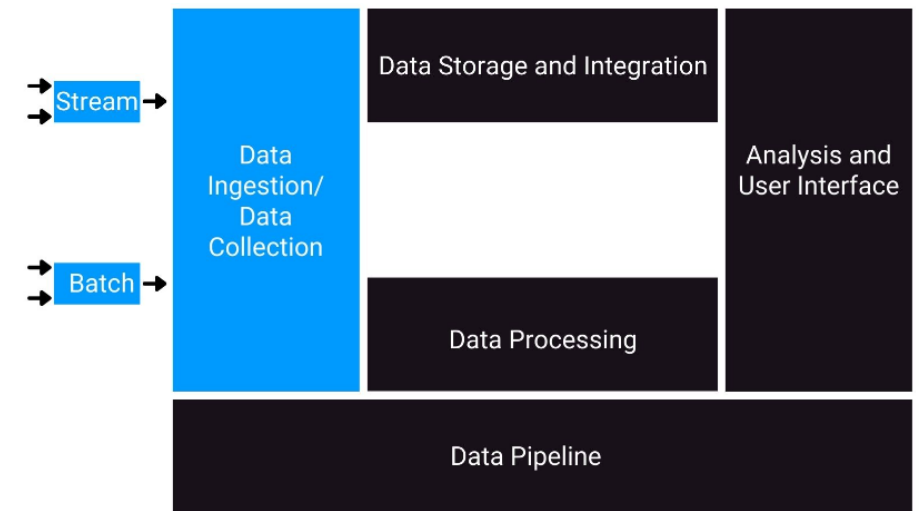


Fonte: <https://www.analyticsvidhya.com>

# Batch e Stream

Embora o modelo de processamento em batch exija um conjunto de dados coletados ao longo do tempo, o processamento de stream requer que os dados sejam inseridos em uma ferramenta de análise, geralmente, em lotes menores e em tempo real.

Nesse caso, é comum acontecer que as equipes de TI fiquem paradas esperando que todos os dados sejam carregados antes de iniciar a fase de análise.



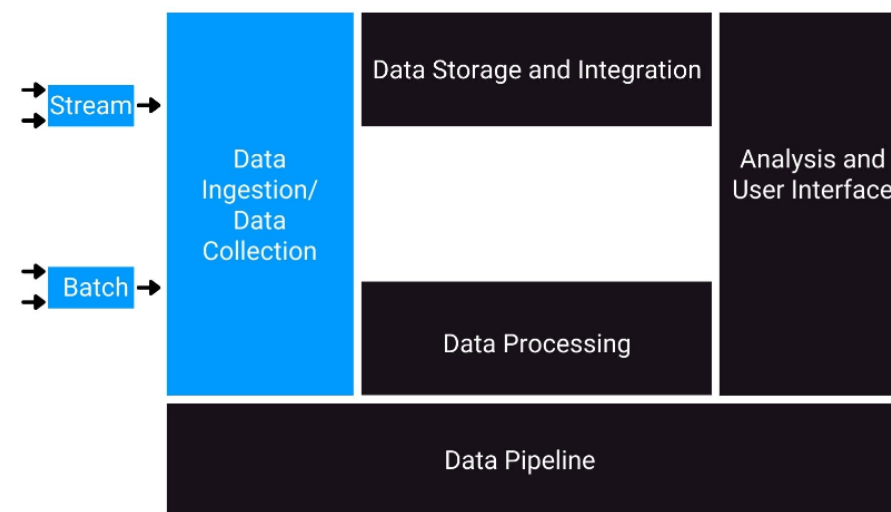
Fonte: <https://www.analyticsvidhya.com>

# Batch e Stream

O stream também podem estar envolvido no processamento de grandes quantidades de dados.

Já o batch funciona melhor quando você não precisa de análises em tempo real.

Como o processamento stream é responsável pelo processamento de dados em movimento e pelo rápido fornecimento de análise, ele gera resultados quase instantâneos, por exemplo usando plataformas de Business Intelligence.



Fonte: <https://www.analyticsvidhya.com>

# Quando escolher entre Batch e Stream

**Por que não podemos apenas o batch como nos sistemas tradicionais ?**

**Você certamente pode, mas se tiver enormes volumes de dados, não é uma questão de quando você precisar extraí-los, mas quando precisará usá-los.**

**As empresas veem os dados em tempo real como um divisor de águas, mas ainda pode ser um desafio chegar lá sem as ferramentas adequadas, principalmente, porque elas precisam trabalhar com volumes, variedades e tipos de dados cada vez maiores de diversos sistemas, como mídias sociais.**

**Normalmente, as organizações querem ter processos de dados mais ágeis para poderem passar da imaginação para a inovação mais rapidamente e responder às ameaças da concorrência.**

# Quando escolher entre Batch e Stream

**Por exemplo, os dados enviados dos sensores de uma turbina eólica estão sempre ligados. Assim, o fluxo de dados é ininterrupto e flui o tempo todo. Uma abordagem em batch aqui seria obsoleta, pois não há início ou parada do fluxo. Esse é um caso de uso perfeito em que o processamento de stream é o caminho mais adequado.**

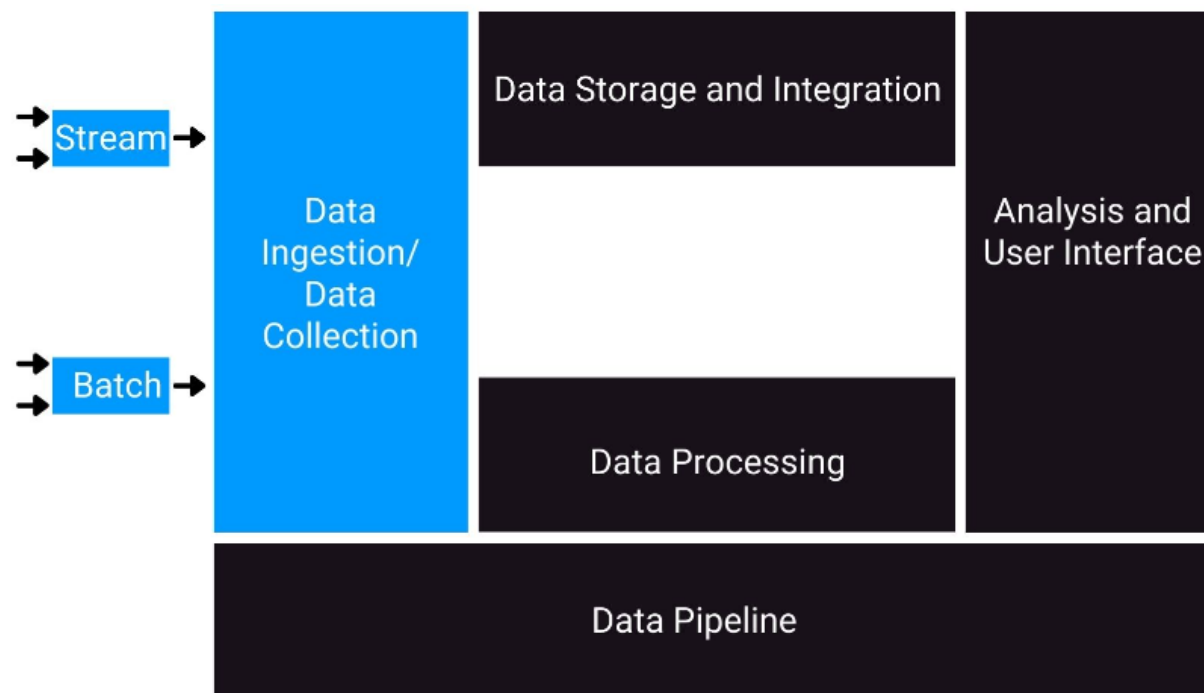
**O processamento de dados em batch e stream são dois modelos diferentes — não é uma questão de escolher um sobre o outro, é sobre ser assertivo e determinar qual é o melhor para cada caso de uso.**

# Camada de Armazenamento e Integração de Dados

# Ingestão / Coleta de Dados

Depois que os dados são coletados (data ingestion), eles precisam ser armazenados e integrados à plataforma de dados.

Para armazenar e integrar os dados, nos dirigimos para a segunda camada da plataforma de dados que é a camada de armazenamento de dados ou camada de integração de dados.



Fonte: <https://www.analyticsvidhya.com>

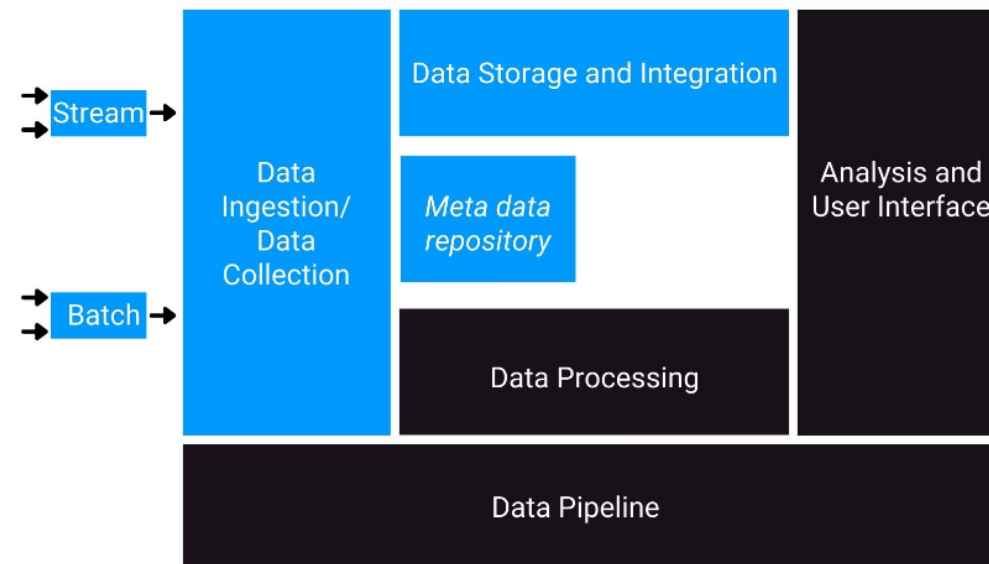


# Armazenamento e Integração de Dados

A camada de coleta de dados, como o nome sugere, é responsável por armazenar dados para processamento e uso a longo prazo.

Além disso, essa camada também é responsável por disponibilizar os dados para processamento nos modos streaming e batch.

Como essa camada é responsável por disponibilizar os dados para processamento, ela precisa ser confiável, escalável, de alto desempenho e econômica.



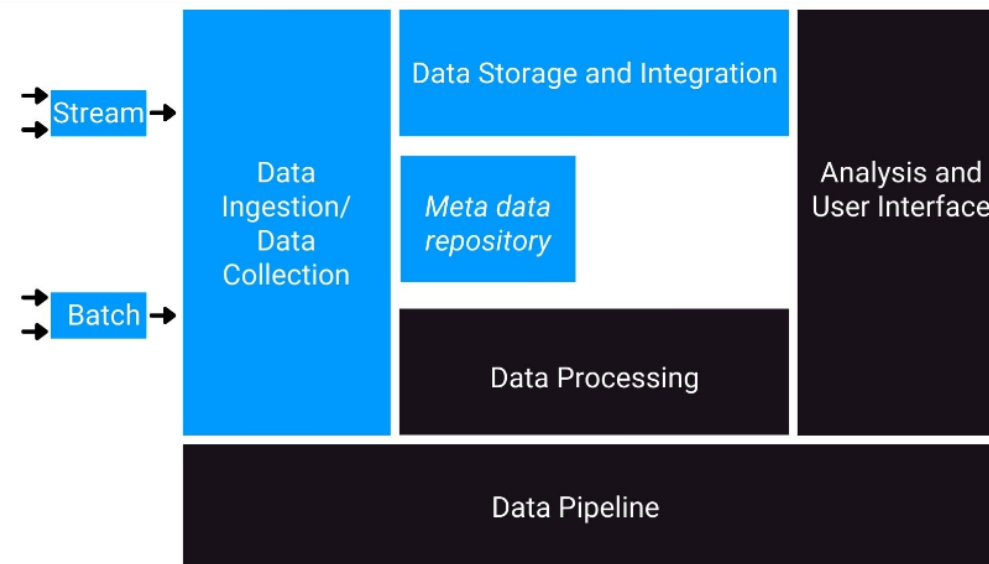
Fonte: <https://www.analyticsvidhya.com>

# Armazenamento e Integração de Dados

IBM DB2, Microsoft SQL Server, MySQL, Oracle Database e PostgreSQL são alguns dos bancos de dados relacionais populares.

Nos últimos anos, os bancos de dados relacionais baseados em nuvem vem ganhando popularidade - IBM DB2, Google Cloud SQL e SQL Azure.

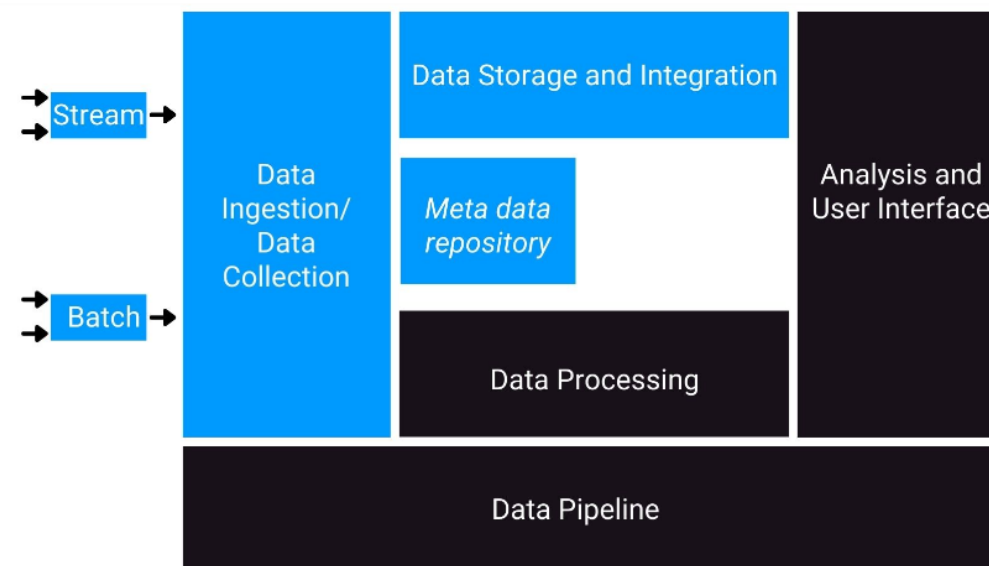
Também temos os sistemas de banco de dados NoSQL ou não relacional na nuvem, temos IBM Cloudant, Redis, MongoDB, Cassandra e Neo4J.



Fonte: <https://www.analyticsvidhya.com>

# Armazenamento e Integração de Dados

As ferramentas para integração incluem o IBM Cloud Pak for Integration e o Open Studio da IBM. Uma vez que os dados tenham sido ingeridos, armazenados e integrados, eles precisam ser processados. Com isso, avançamos para a Camada de Processamento de Dados.



Fonte: <https://www.analyticsvidhya.com>

# Armazenamento e Integração de Dados

Por exemplo, o software de integração do Cloud Pak for Integration fornece um conjunto abrangente de ferramentas de integração, conectando aplicações e dados em qualquer ambiente de nuvem ou no local. É uma plataforma cloud impulsionada por IA que permite a automação das operações de rede para que os provedores de serviços de comunicação (CSPs) possam fornecer serviços mais rapidamente.

A plataforma oferece interações de eventos em tempo real, transfere dados em qualquer nuvem, implementa e escala com arquitetura nativa em nuvem e disponibiliza serviços básicos compartilhados, tudo com segurança e criptografia de classificação corporativa de ponta a ponta.

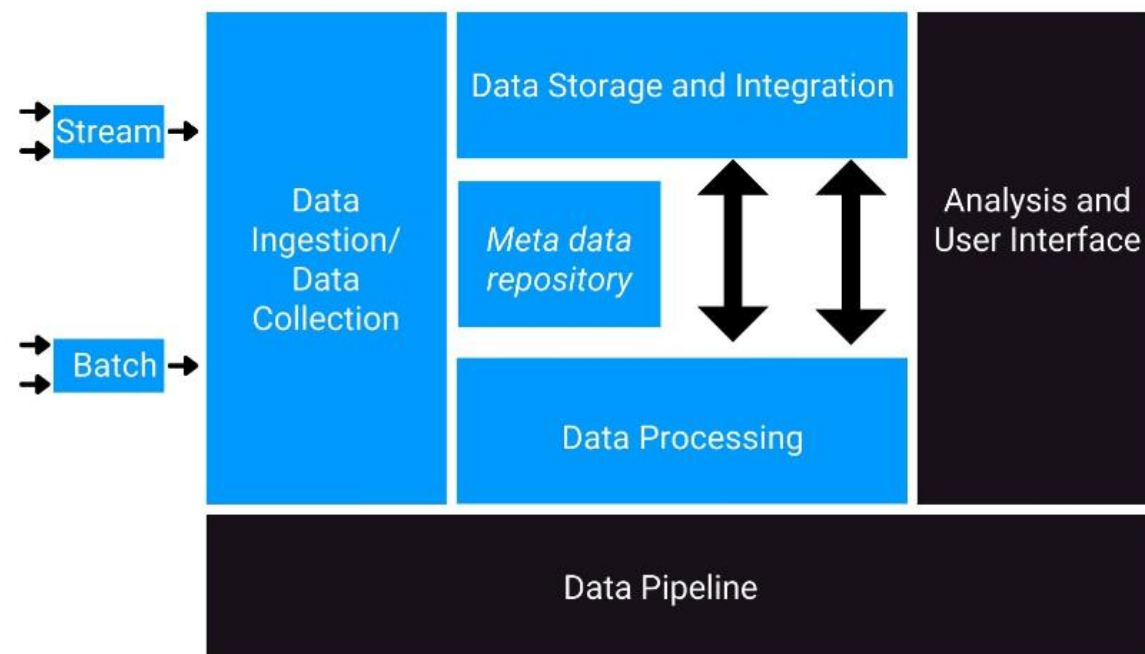
O Cloud Pak for Integration usa serviços e recursos da AWS, inclusive Virtual Private Clouds (VPCs), zonas de disponibilidade, grupos de segurança, o Amazon Elastic Block Store (Amazon EBS), o Amazon Elastic Compute Cloud (Amazon EC2) e o Elastic Load Balancing para criar uma plataforma de nuvem confiável e escalável.

# Camada de Processamento de Dados

# Processamento de Dados

Depois que os dados são coletados (data ingestion), eles precisam ser armazenados e integrados à plataforma de dados.

Para armazenar e integrar os dados, nos dirigimos para a segunda camada da plataforma de dados que é a camada de armazenamento de dados ou camada de integração de dados.



Fonte: <https://www.analyticsvidhya.com>

# Processamento de Dados

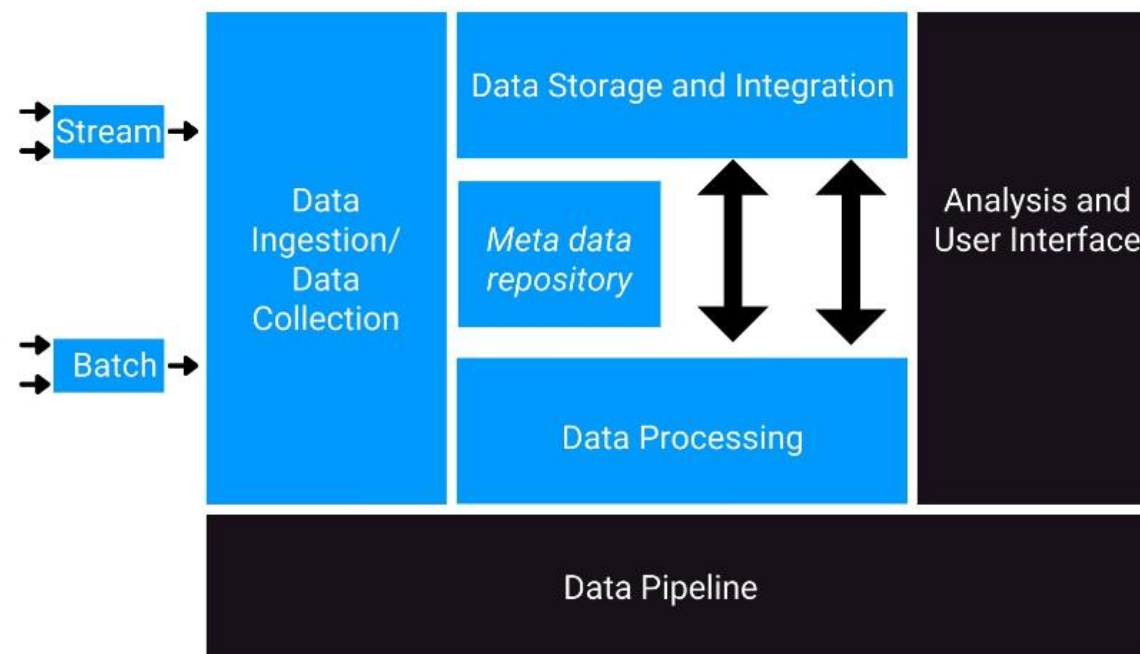
Esta camada é responsável por uma tarefa de processamento.

O processamento inclui validações de dados, transformações e aplicação de lógica de negócios aos dados.

A camada de processamento deve ser capaz de executar algumas tarefas que incluem:

Leia dados em batch (lote) ou modos de streaming do armazenamento e aplique transformações.

Suporte a ferramentas de consulta e linguagens de programação populares.



Fonte: <https://www.analyticsvidhya.com>

# Processamento de Dados

A camada de processamento deve ser capaz de executar algumas tarefas que incluem:

- Ler dados em batch (lote) ou modos de streaming do armazenamento e aplique transformações.
- Suporte a ferramentas de consulta e linguagens de programação populares.
- Escale para atender às demandas de processamento de um conjunto de dados crescente.
- Forneça uma maneira para analistas e cientistas de dados trabalharem com dados na plataforma de dados.



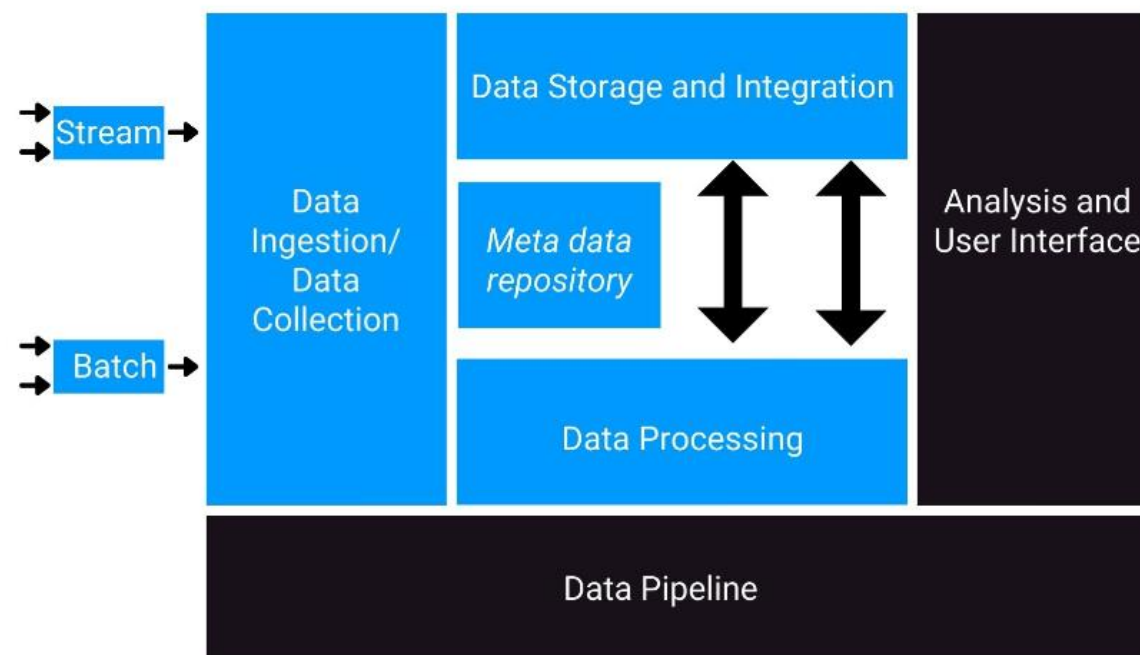
# Processamento de Dados

As tarefas de transformação que geralmente ocorrem nesta camada incluem:

- **Estruturação:** são as ações que alteram a estrutura dos dados. Essa mudança pode ser de natureza simples ou complexa. O simples também pode ser como alterar a disposição dos campos dentro do registro ou conjunto de dados ou complexo como combinar estruturas complexas de campos usando junções e uniões.
- **Normalização:** esta parte se concentra na redução da redundância e inconsistência. Ele também se concentra na limpeza do banco de dados de dados não utilizados.
- **Desnormalização:** a desnormalização é a tarefa de combinar dados de várias tabelas em uma única tabela para que ela possa ser consultada com mais eficiência para fins de relatório e análise.
- **Limpeza de dados:** corrige irregularidades nos dados para fornecer dados confiáveis para aplicativos e usos posteriores.

# Processamento de Dados

Existem inúmeras ferramentas disponíveis no mercado para realizar essas operações nos dados, incluindo planilhas, OpenRefine, Google DataPrep, Watson Studio Refinery e Trifacta Wrangler. Python e R também oferecem várias bibliotecas e pacotes criados explicitamente para processar dados.

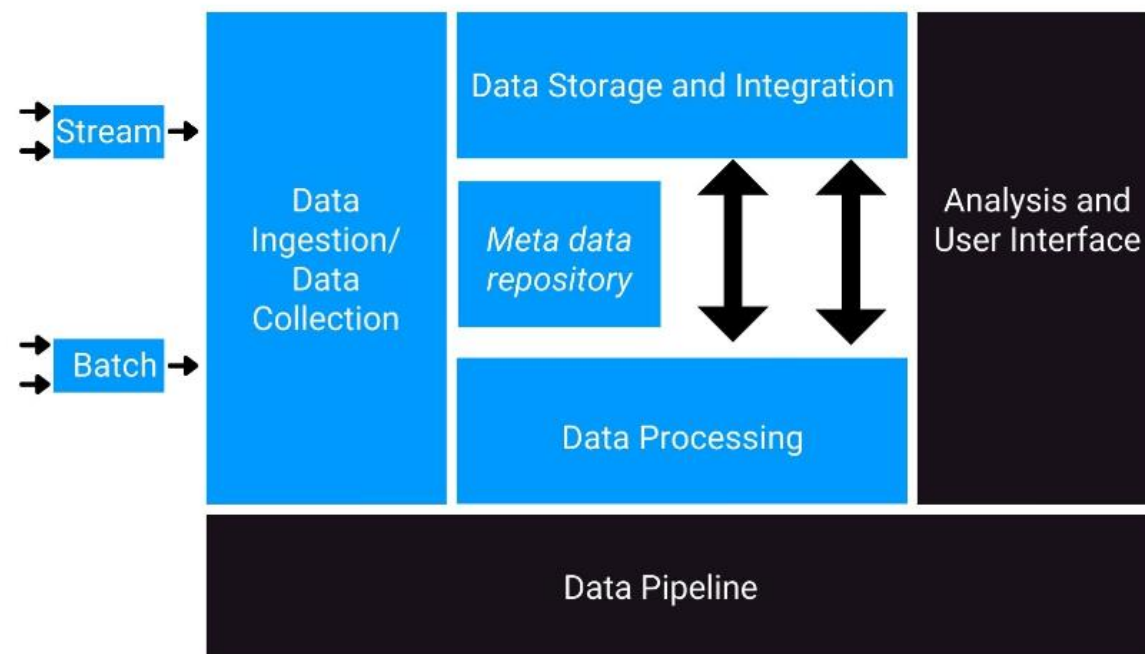


Fonte: <https://www.analyticsvidhya.com>

# Processamento de Dados

É muito importante saber que o armazenamento e o processamento nem sempre são realizados em camadas separadas.

Por exemplo, em bancos de dados relacionais, o armazenamento e o processamento ocorrem na mesma camada, enquanto nos sistemas de big data, os dados são primeiro armazenados no sistema Hadoop File Distributed e depois processados no mecanismo de processamento de dados, como o Spark.



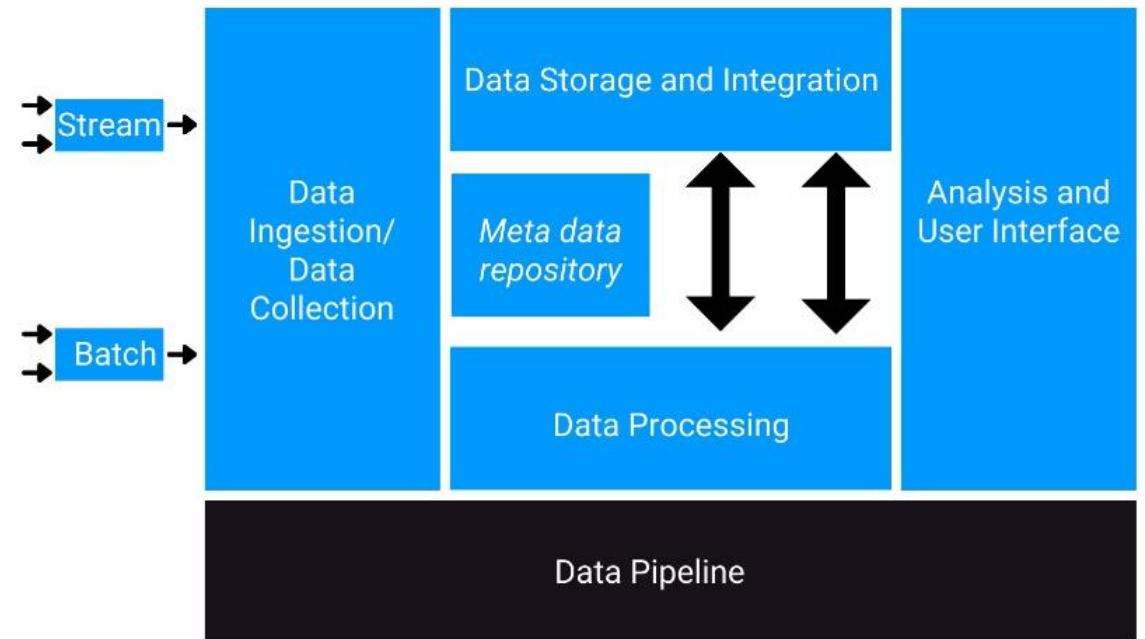
Fonte: <https://www.analyticsvidhya.com>

# Camada de Análise e Interface do Usuário

# Análise e Interface do Usuário

Essa camada é responsável por fornecer os dados do processo aos usuários finais, incluindo analistas de inteligência de negócios e partes interessadas nos negócios, que consomem esses dados com a ajuda de painéis e relatórios interativos.

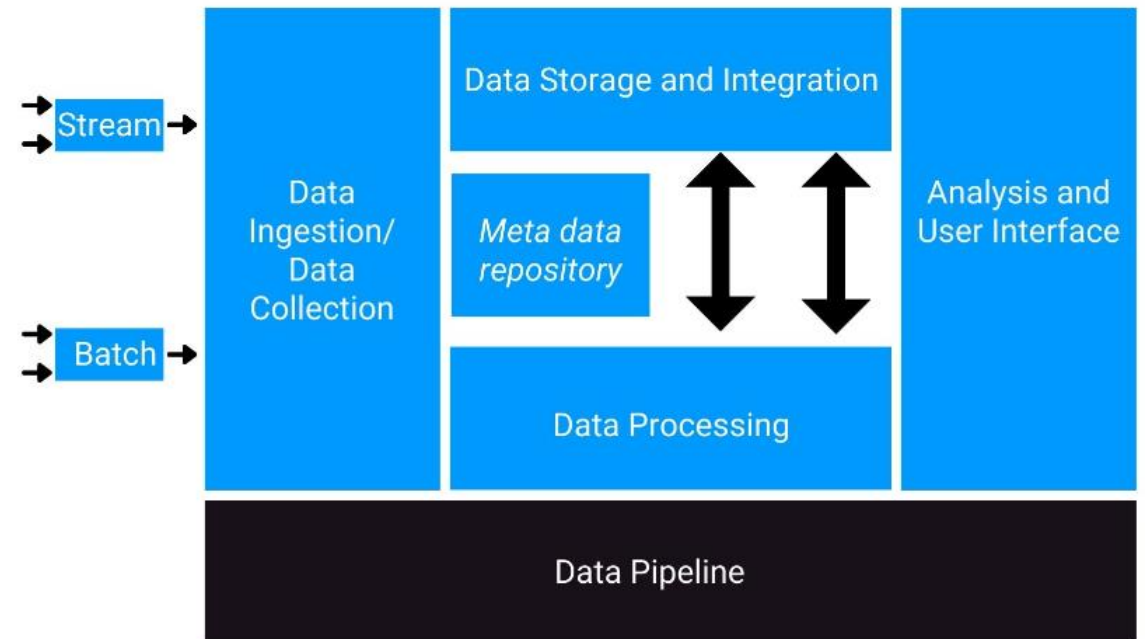
Além disso, cientistas de dados e analistas de dados se enquadram nessa categoria de usuário final que processe posteriormente esses dados para o caso de uso específico.



Fonte: <https://www.analyticsvidhya.com>

# Análise e Interface do Usuário

Essa camada precisa oferecer suporte a ferramentas de consulta, como ferramentas SQL e ferramentas No-SQL e linguagens de programação como Python, R e Java e, além disso, essas camadas precisam oferecer suporte a APIs que podem ser usadas para executar relatórios sobre dados para processamento online e offline.



Fonte: <https://www.analyticsvidhya.com>



<https://www.tableau.com>

A empresa Tableau Software, foi fundada em 2003, por Chris Stolte, Pat Hanrahan e Christian Chabot que tinham como intenção fornecer as visualizações dos dados do departamento de computação da Universidade de Stanford. Pat Hanrahan foi um dos membros fundadores da empresa de animação digital Pixar, então eles tinham a visão que a computação gráfica poderia ser de ajuda enorme na compreensão dos dados nas organizações (Tableau, 2017).

Como acontece com o Power BI, o grupo queria que o software fosse de fácil adaptação, onde qualquer pessoa familiarizada com o Excel poderia criar análises consistentes dos resultados de uma organização. Ficando a cargo da equipe de TI apenas as questões que envolvem a estruturação dos dados, como o DW e se necessário a criação de cubos OLAP.

# Power BI

<https://powerbi.microsoft.com/pt-br>



Juntamente com o Tableau são os líderes no mercado de Business Intelligence. Começou a ser desenvolvido em 2010 com o codinome “Project Crescent”, porém só teve seu lançamento em 2013 e com outro nome Microsoft Power BI para o Office 365. No começo, era como uma parte do software de gerenciamento de planilhas da empresa, Microsoft Excel, com as funções PowerView, PowerQuery e Power Pivot. Desde lá o Power BI se tornou um software completo para uma implementação de BI, não só contando com apenas a visualização das informações, mas também com a criação do DW e processo de ETL e do cubo OLAP. A figura 11, mostra a logo adotada pela Microsoft para o Power BI.



# Camada de Pipeline de Dados

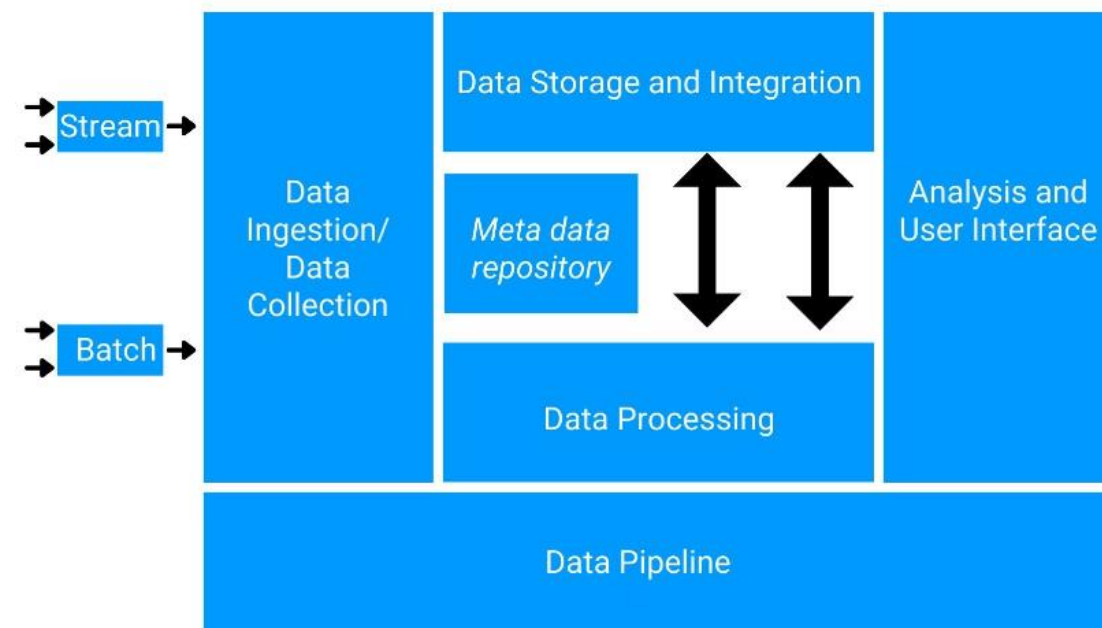
# Pipeline de Dados

Esta é a última camada desta arquitetura

Esta camada é responsável por implementar e manter um fluxo contínuo de dados através deste pipeline de dados.

É a camada que tem a capacidade de extrair, transformar e carregar ferramentas.

Há uma série de soluções de pipeline de dados disponíveis, sendo as mais populares Apache Airflow e DataFlow.



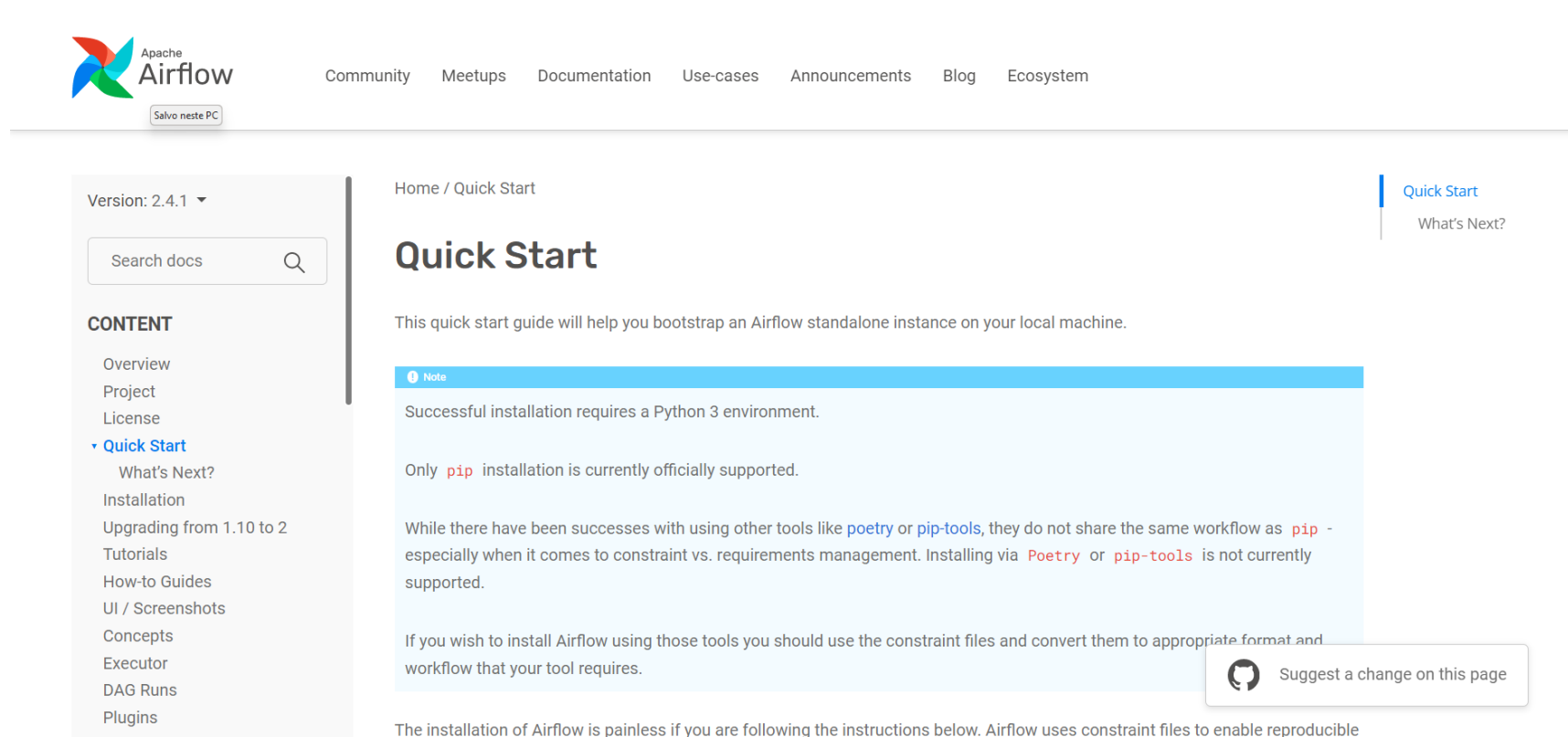
Fonte: <https://www.analyticsvidhya.com>

# Apache Airflow

- <https://airflow.apache.org/>
- O Airflow é uma plataforma criada pela comunidade para criar, agendar e monitorar de forma agendada os fluxos de trabalho.
- O Airflow tem uma arquitetura modular e usa uma fila de mensagens para orquestrar um número arbitrário de trabalhadores. O fluxo de ar está pronto para escalar até o infinito.
- Os pipelines de fluxo de dados são definidos em Python, permitindo a geração dinâmica de pipeline. Isso permite escrever código que instancia pipelines dinamicamente.

# Apache Airflow

- <https://airflow.apache.org/docs/apache-airflow/stable/start.html>



The screenshot shows the Apache Airflow documentation website. At the top, there's a navigation bar with links for Community, Meetups, Documentation, Use-cases, Announcements, Blog, and Ecosystem. The main content area is titled 'Quick Start' and includes a search bar, a table of contents, and a note about Python 3 requirements. A sidebar on the left lists various documentation sections, with 'Quick Start' highlighted. A 'Suggest a change' button is visible at the bottom right of the note section.

Version: 2.4.1 ▾

Search docs 🔍

**CONTENT**

- Overview
- Project
- License
- ▾ Quick Start
  - What's Next?
- Installation
- Upgrading from 1.10 to 2
- Tutorials
- How-to Guides
- UI / Screenshots
- Concepts
- Executor
- DAG Runs
- Plugins
- Security

Home / Quick Start

## Quick Start

This quick start guide will help you bootstrap an Airflow standalone instance on your local machine.


**Note**

Successful installation requires a Python 3 environment.

Only `pip` installation is currently officially supported.

While there have been successes with using other tools like `poetry` or `pip-tools`, they do not share the same workflow as `pip` - especially when it comes to constraint vs. requirements management. Installing via `Poetry` or `pip-tools` is not currently supported.

If you wish to install Airflow using those tools you should use the constraint files and convert them to appropriate format and workflow that your tool requires.

 Suggest a change on this page

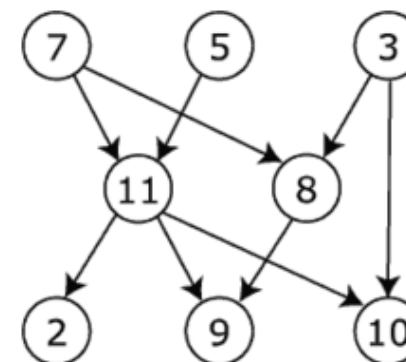
The installation of Airflow is painless if you are following the instructions below. Airflow uses constraint files to enable reproducible

# Apache Airflow


- <https://airflow.apache.org/>
- DAG - Grafos de tarefas/ usos.
- OPERATOR - O operador refere-se à etapa de transformação ainda dividida em
  - Sensor - Este tipo de operador realiza uma função de polling com frequência/timeout.
  - Executor - Este tipo de operador realiza operações de trigger, por exemplo, HiveOperator, Pig Operator.
  - TASK - Task é a principal entidade do DAG. O principal aqui é a instância de tarefa considerada para executar uma tarefa em um ponto do tempo.
  - HOOK - É considerado como a Interface para o Sistema externo como um gancho de JDBC e HTTP.

# Apache Airflow - DAG

- Em ciência da computação e matemática, um grafo acíclico direcionado (DAG – directed acyclic graphs) é um grafo direcionado que não possui ciclos direcionados.
- Na teoria dos grafos, um grafo refere-se a um conjunto de vértices conectados por linhas chamadas arestas.
- Em um grafo direcionado, cada aresta está associada a uma direção de um vértice inicial a um vértice final.
- Se viajarmos ao longo da direção das arestas e descobrirmos que não se formam laços fechados ao longo de qualquer caminho, diz-se que não há ciclos direcionados.
- E o gráfico formado é um DAG.



# Apache Airflow

 Airflow

DAGs


Data Profiling ▾

Browse ▾

Admin ▾

Docs ▾









































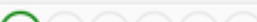

15:30 MSK



## DAGs

Show  entries

Search:

		DAG	Schedule	Owner	Recent Statuses 	Links
		oda_load_gc_event_microbatches	*H10 * * * *			
		oda_load_gc_event_to_hdp	@daily			
		oda_load_gwt	@daily			
		oda_load_hawk_new	1 day, 0:00:00			
		oda_load_hc_dicts	1 day, 0:00:00			
		oda_load_hc_pg_juc5	1 day, 0:00:00			
		oda_load_its	@hourly			
		oda_load_jh	1 day, 0:00:00			
		oda_load_jw	1 day, 0:00:00			
		oda_load_large_sf_ps4	1 day, 0:00:00			

Showing 31 to 40 of 99 entries

Previous

1

2

3

4

5

10

Next

SCREENSHOTER@mail.ru

# Benefícios do Apache Airflow

- **Dinâmico** - O pipeline construído pelo **Airflow dynamic**, construído na forma de código que dá uma vantagem para ser dinâmico.
- **Extensível** - é fácil iniciar os operadores, executores devido aos quais a biblioteca foi impulsionada para que possa se adequar ao nível de abstração para suportar um ambiente definido.
- **Elegante** – um pipeline desenvolvido com a ajuda do Airflow é angular e inequívoco porque o mecanismo de modelo Jinja usado para parametrizar os scripts são incorporados ao núcleo do Airflow.
- **Escalável** - a arquitetura do Airflow é composta por unidades padronizadas que também utilizam a técnica de mensagens para enfileirar o número de trabalhadores e além disso é escalável ao infinito.



# Apache Airflow

- O Apache Airflow é compatível com gerenciamento de dependências, extensível, **escalável e de código aberto**.
- As principais funções do Apache Airflow são **agendar fluxo de trabalho, monitorar e criar**.
- Essas funções são alcançadas **com Grafos Acíclicos Dirigidos (DAG) das tarefas**.

# Barramentos de Mensageria de Dados

# Mensageria de Alta Performance

O mundo tem gerado um volume cada vez maior de informações.

Um cenário comum é que estes dados, uma vez gerados, precisam ser transportados para diversas aplicações.

Por exemplo, quando fazemos uma compra com cartão de crédito a operadora precisa avisar várias aplicações diferentes para contabilizar a compra, os benefícios e até a análise contra fraudes.

Neste caso, é desejável que cada aplicação seja responsável pela implementação das regras do negócio. E apenas isso.

Uma aplicação não deveria se preocupar com a entrega ou captura dos dados. Este é um problema complexo envolvendo **data streaming**.

# Mensageria de Alta Performance

## **Data streaming**

Um fluxo de dados constante e sem controle é chamada de data streaming.

Sua principal característica é que os dados não tem limites definidos, assim, não é possível dizer quando começa ou acaba o fluxo (stream).

Os dados são processados à medida em que chegam no destino.

Alguns autores chamam de processamento em tempo real, ou on-line.

Uma abordagem diferente é o processamento em bloco, batch, ou off-line, na qual blocos de dados são processados em janelas de tempo.

# Mensageria de Alta Performance

## Sistema de Mensagem

O sistema de mensagens permite desacoplar a geração dos dados do seu processamento de forma assíncrona, assim, uma aplicação pode continuar funcionando sem esperar a outra terminar seu processamento.

# Mensageria de Alta Performance com Apache Kafka

## O que é o Apache Kafka?

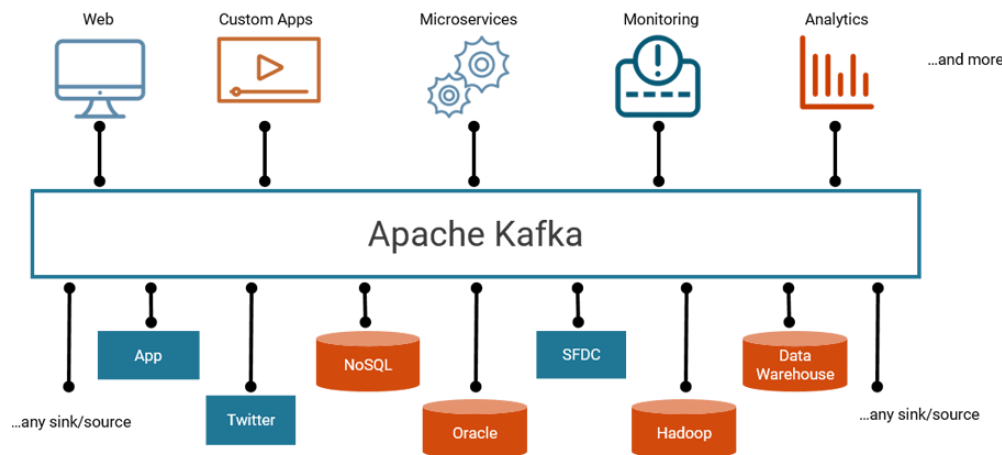
O Apache Kafka é uma plataforma open-source de mensageria usado para criar aplicações de streaming, ou seja, aquelas com fluxo de dados contínuo.

Kafka é baseado em logs, algumas vez chamado de write-ahead logs, commit logs ou até mesmo transaction logs. Um log é uma forma básica de armazenamento, porque cada nova informação é adicionada no final do arquivo. Este é o princípio de funcionamento do Kafka.

O Kafka é adequado para soluções com grande volume de dados (big data) porque uma das suas características é a alta taxa de transferência.

# Data Engineering- Apache Kafka

- Apache Kafka é uma plataforma open-source de processamento de streams desenvolvida pela Apache Software Foundation, escrita em Scala e Java. O projeto tem como objetivo fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real.



Fonte: <https://kafka.apache.org/>

# Mensageria de Alta Performance com Apache Kafka

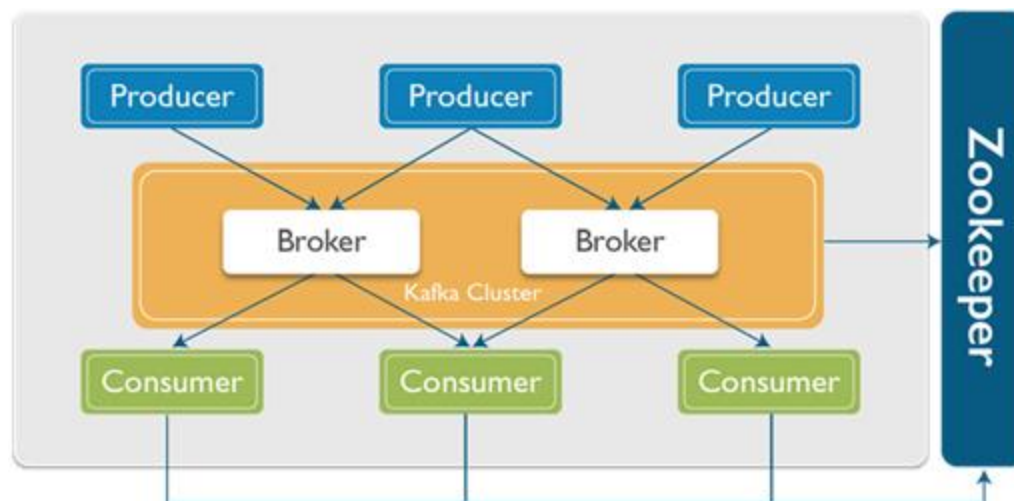
O Apache Kafka tem 3 funcionalidades principais:

- **Sistema de mensagens:** Usado para desacoplar sistemas com escopos diferentes. Sistema de mensagem do tipo *publish/subscribe*;
- **Sistema de armazenamento:** Usado para guardar os eventos (logs) de forma consistente, permitindo reconstruir o estado do sistema a qualquer momento a partir destes logs. As mensagens ficam armazenadas por um período de tempo pré-definido.
- **Processamento de \_stream:** Usado para transformar dados em tempo real, como mapeamentos, agregações, junções, etc. É possível transformar a mensagem imediatamente após o seu recebimento.



# Apache Kafka

Basicamente, o Kafka é um intermediário que coleta os dados da fonte e entrega para uma aplicação que consumirá esses dados, como visto na imagem:

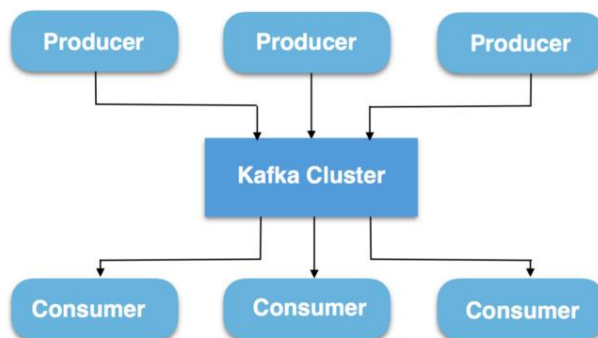


Fonte: <https://kustavo.github.io/guia-rapido/aplicativos/kafka/>

# Apache Kafka - Arquitetura Pub/Sub

O Kafka segue a arquitetura de mensageria no estilo Publisher/Consumer, em que um componente **publica** uma mensagem em uma fila chamado de **tópico** a um intermediário e um outro componente **consome** essa mensagem a partir do intermediário, no **tópico** que essa informação está.

Essas mensagens não são direcionadas a determinados consumidores, então para que eles possam acessar os dados eles se **inscrevem** nessa fila do intermediário para aguardarem novas informações.

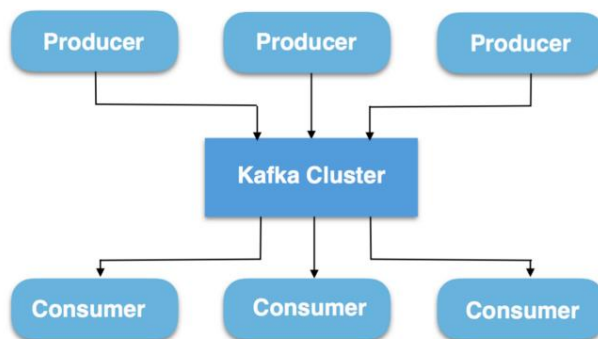


Fonte: <https://medium.com/data-hackers/mensageria-de-alta-performance-com-apache-kafka-1-c801d60a9299>

# Apache Kafka - Arquitetura Pub/Sub

Um cluster Kafka é composto por vários brokers. Um broker é um servidor Kafka que recebe mensagens dos producers e as grava no disco. Cada broker gerencia uma lista de tópicos e cada tópico é dividido em diversas partições.

A arquitetura do Apache Kafka é composta por producers, consumers e o próprio cluster. O producer é qualquer aplicação que publica mensagens no cluster. O consumer é qualquer aplicação que recebe as mensagens do Kafka. O cluster Kafka é um conjunto de nós (brokers) que funcionam como uma instância única do serviço de mensagem.



Fonte: <https://medium.com/data-hackers/mensageria-de-alta-performance-com-apache-kafka-1-c801d60a9299>

# Problema Real - Mensageria de Dados

# Mensageria de Alta Performance

Caso de uso com o Apache Kafka

Resolver problemas de integração entre a plataforma de marketplace e o e-commerce com seguintes problemas:

Lentidão na integração das ofertas dos lojistas marketplace com e-commerce que refletiam em:

- Um preço ou disponibilidade de uma oferta demorava até horas para refletir no site.
- O processo síncrono, e muitas vezes o alto volume das integrações gera indisponibilidade no site.
- O time precisava desligar as integrações, ligando somente à noite, para não haver risco de gerar indisponibilidade no site.
- A maioria das atualizações eram feitas por meio de processos *batch*.

# Mensageria de Alta Performance

Esses problemas faziam com que os produtos dos lojistas marketplace que estivessem sem estoque fossem exibidos no site como disponível, ou pior, produtos com preços desatualizados.

O Apache Kafka permitiu tornar essas integrações assíncronas e em tempo real, ao invés de integrar com processos batch.

Foi possível gerar eventos cada vez que uma oferta era criada, atualizada ou deletada, possibilitando atualizações distribuídas, entregando mais throughput.

# Mensageria de Alta Performance

**Pode agir como um buffer para que os sistemas não travem:**

- Muitos sistemas ainda fazem a transformação e o processamento de dados em lote no período da noite, o Apache Kafka resolve esse processo lento em várias etapas agindo como um intermediário recebendo os dados de um sistema de origem, e em seguida disponibilizando para o sistema de destino em tempo real.

# Mensageria de Alta Performance

## Redução da necessidade de múltiplas integrações:

- Imagine um cenário contendo 4 sistemas de origem e 6 sistemas de destino, são necessárias a criação de 24 integrações. Esse é um processo complicado, sem mencionar também que é uma maneira lenta e propensa a erros no fornecimento de dados, com vários sistemas fazendo **polling** de um mesmo dado, consumindo recursos desnecessários dos nossos sistemas de gerenciamento de bancos de dados.
- Com o Apache Kafka, ao invés de construir múltiplas integrações, é necessário apenas uma integração para cada sistema produtor e consumidor, diminuindo a fricção entre times e tornando mais flexível a substituição de sistemas legados.



# Mensageria de Alta Performance

## Baixa latência e alto *throughput*:

- O Apache Kafka torna mais fácil a criação de arquiteturas de sistemas em tempo real (*near real time*), reduz o tempo de espera pela disponibilidade de um determinado dado pela área usuária, e possibilita a tomada de decisões próximo ao tempo real, um exemplo seria o marketing de precisão.

# Mensageria de Alta Performance

## Todos podem acessar os dados:

- Como os dados estão centralizados no Apache Kafka, qualquer outro sistema pode acessar esses dados sem impactar nos sistemas de origens. A maioria das integrações geralmente busca os dados no sistema de origem, podendo às vezes causar lentidão ou indisponibilidade na origem.



**PUC Minas**  
**Virtual**