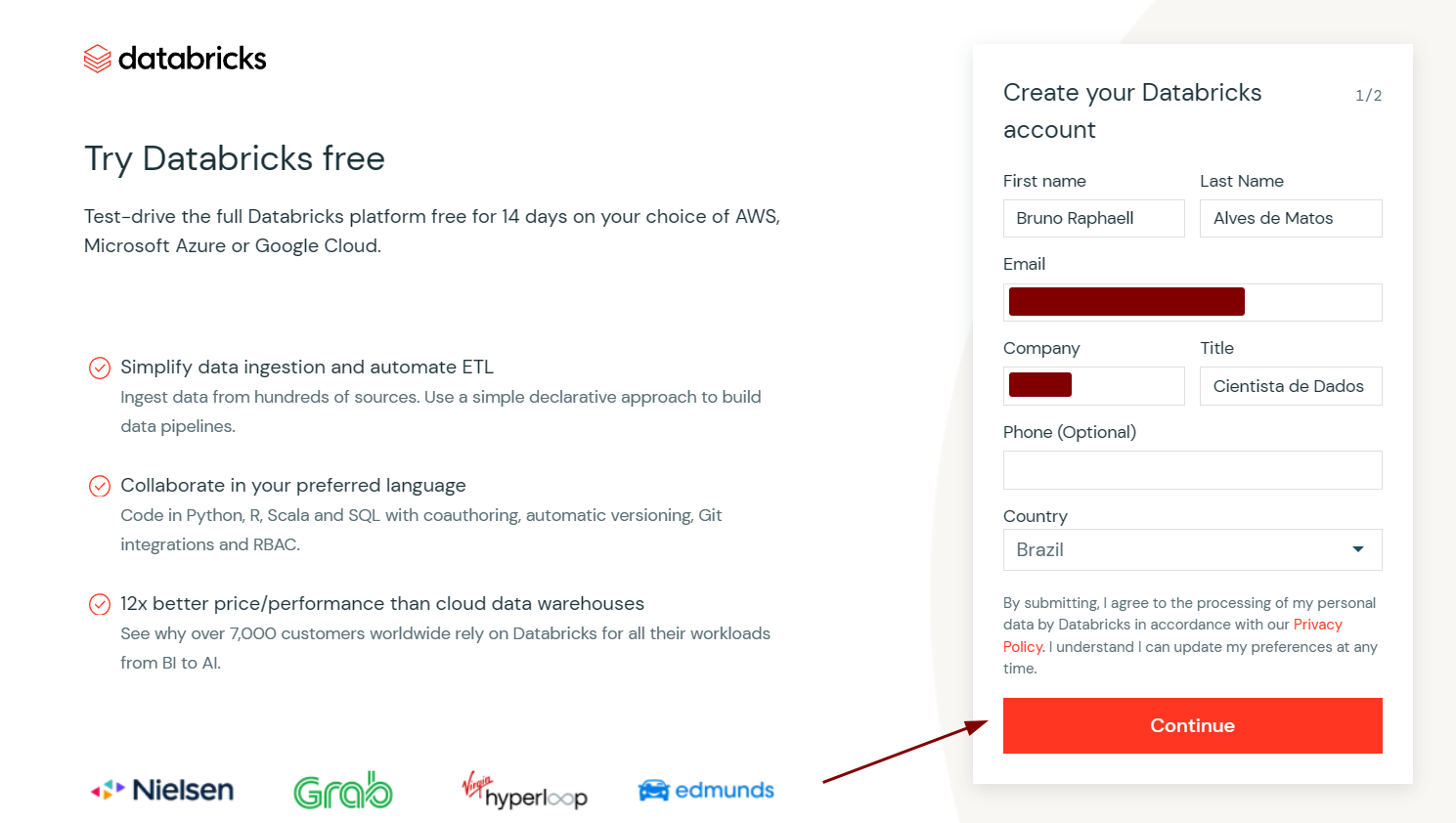
**criando conta no Databricks Community**

O curso utiliza uma conta na plataforma do Databricks Community. Caso você já possua uma conta, entre na [página de login](https://community.cloud.databricks.com/login.html), coloque seu e-mail e senha e aproveite os recursos gratuitos disponíveis.

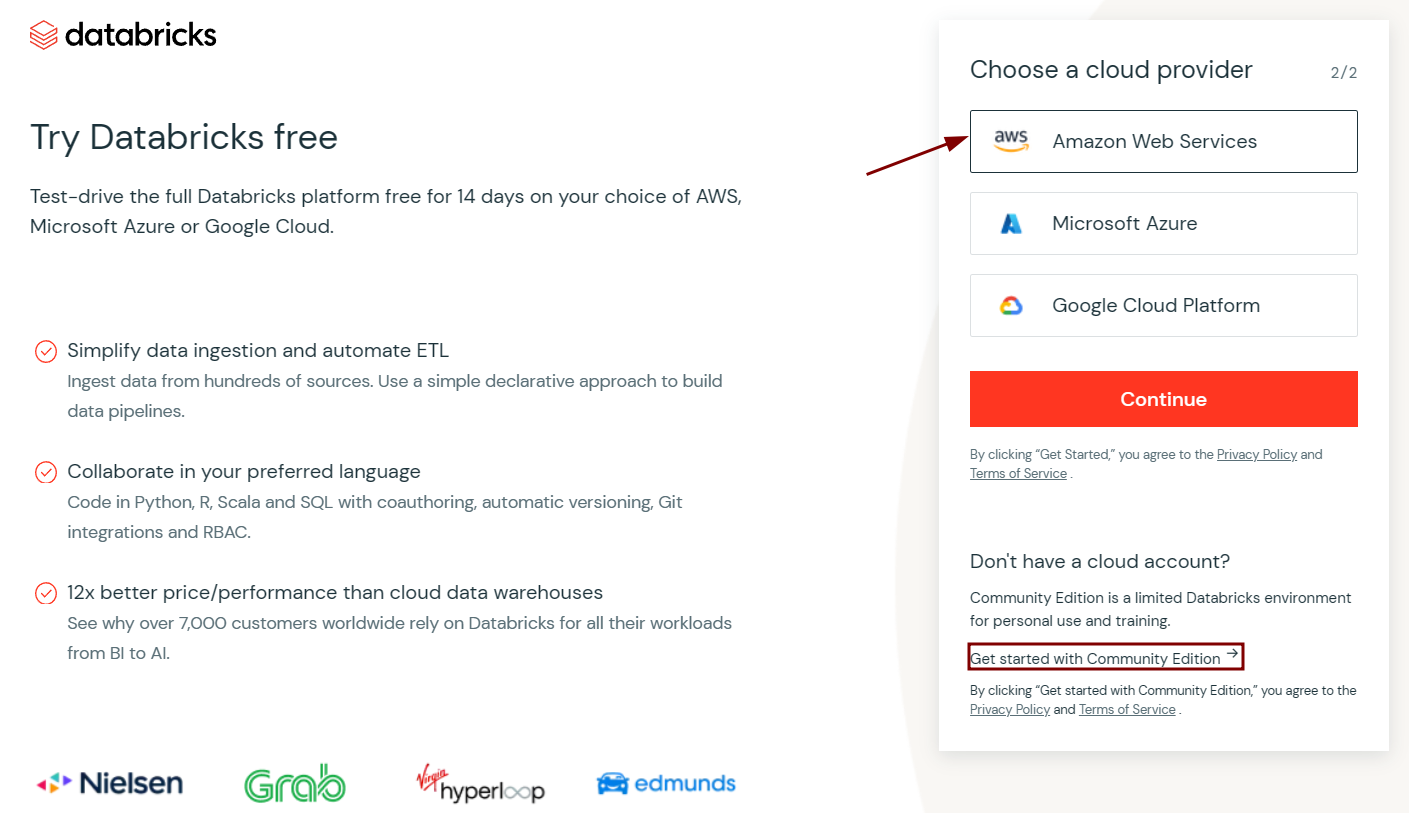
Caso ainda não possua essa conta, vamos aprender como criá-la.

1) Vá até a página [Try Databricks](https://www.databricks.com/try-databricks" \l "account" \t "_blank).

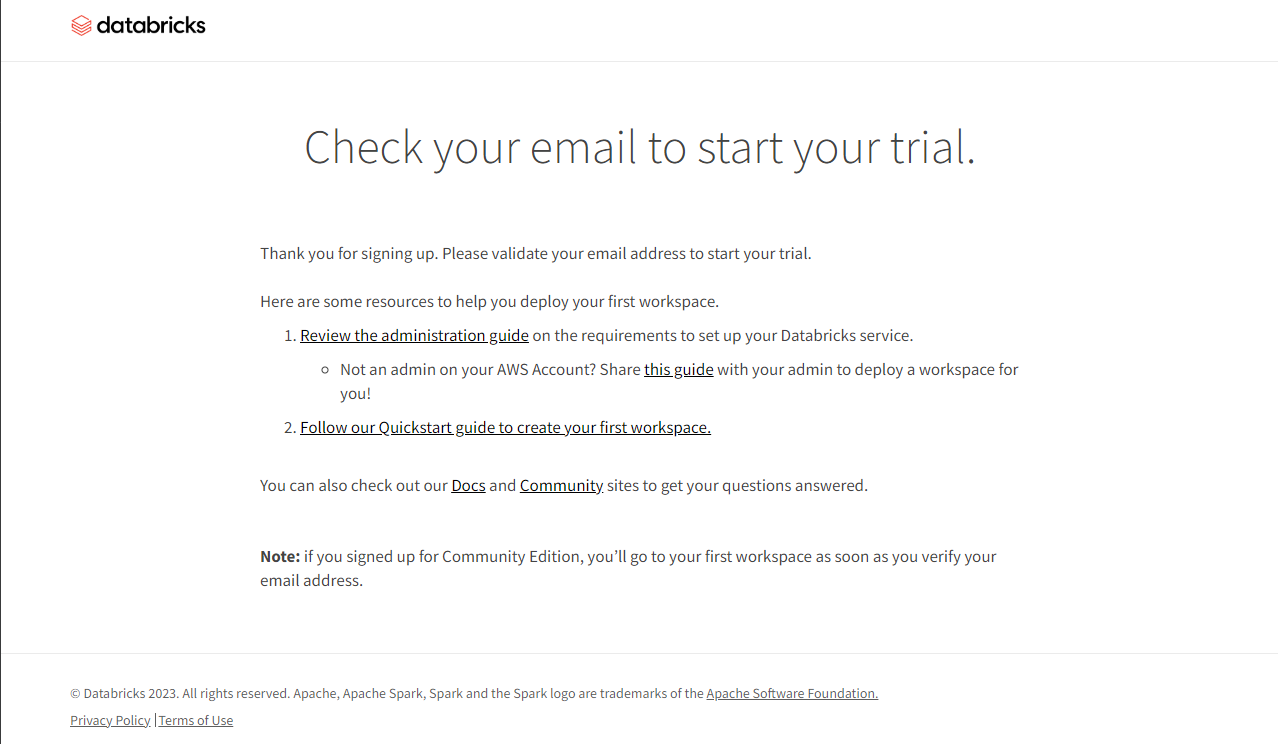
2) Preencha seu nome, empresa, e-mail e cargo. Depois clique em “Continue”, conforme imagem abaixo:



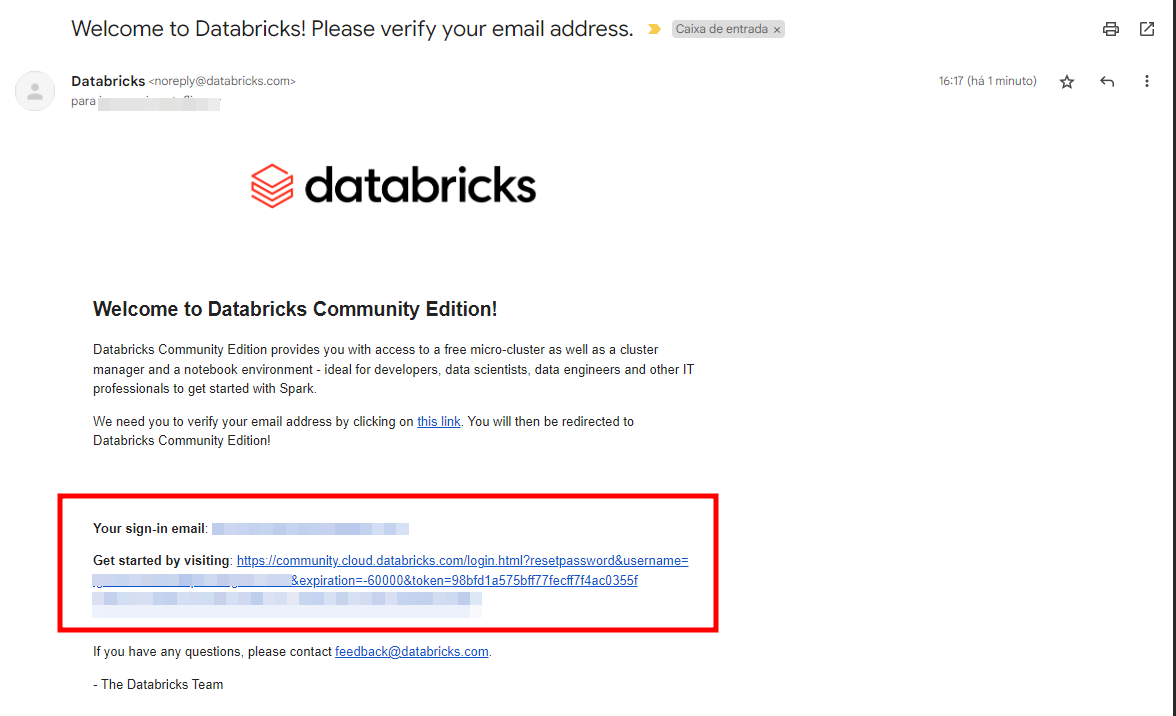
3) Na próxima tela é preciso escolher o provedor de Cloud. Vamos selecionar a **Amazon Web Services** e clicar sobre a opção **Get started with Community Edition**.



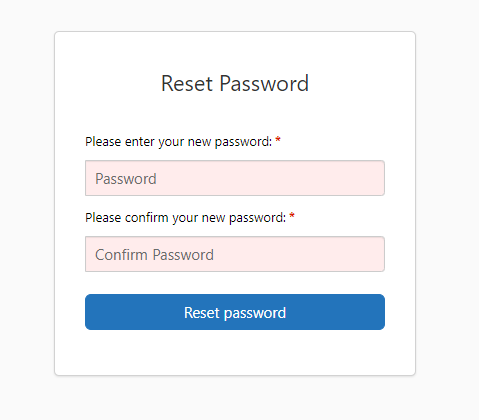
4) Após clicar sobre essa opção, você verá uma página anunciando que um e-mail foi enviado para o endereço fornecido.



5) Procure o e-mail enviado pelo Databricks e clique no link para verificar o endereço de e-mail. Ao fazer isso, será solicitado a criação de uma senha para acessar a plataforma.



6) Ao clicar em **Reset Password** você será direcionado(a) para a tela inicial do Databricks Community Edition.



**02Para saber mais: Databricks - Como surgiu?**

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/databricks-conhecendo-ferramenta/task/122263/next)

Para poder falar da tecnologia **Databricks**, é necessário compreender alguns aspectos históricos que deixem mais claro os porquês de seu desenvolvimento e quais necessidades ele se propõe a solucionar.

Antes de 2005, os sistemas de banco de dados funcionavam em computadores únicos, com algum sistema de armazenamento associado. À medida que os computadores ficaram cada vez mais rápidos, esses sistemas de armazenamento aumentaram também sua capacidade de processamento. Mas, basicamente, a arquitetura fundamental permaneceu a mesma.

Um conjunto limitado de dispositivos de armazenamento detinha todos os dados. Para respostas mais rápidas, era preciso investir em um computador melhor. Já para armazenar mais dados, o investimento se dava em um sistema de armazenamento mais caro.

**Hadoop**

Em 2005, principalmente com as redes sociais, muitos dados eram gerados em um pequeno espaço de tempo e processar todo este volume era um desafio. Então, em 2006, o **Hadoop** gerou um grande avanço no cenário de dados.

Com esta ferramenta, foi possível solucionar problemas em um contexto Big Data, escalonando de forma horizontal, fazendo com que **n** computadores passassem a trabalhar juntos em uma mesma tarefa. Desta maneira foi possível distribuir todos os processos de computação em todos os núcleos destes computadores.

Os componentes-chave do Hadoop são:

1. Um sistema de arquivos distribuído (**HDFS**) que trata todos os discos como um sistema de armazenamento; e
2. Uma estrutura de programação (**MapReduce**) que permite escrever um programa para executar em muitas CPUs.

Assim, o Hadoop foi um grande impulso nos sistemas de banco de dados. Com um menor custo, era possível até montar os próprios conjuntos de máquinas. Entretanto, o Hadoop teve suas desvantagens: O MapReduce era uma nova forma de pensar sobre programação, mas não era algo simples de se fazer.

A tarefa da pessoa engenheira de dados passou a ser, em parte, o desenvolvimento de programas complexos, com várias etapas de mapeamento e redução interdependentes. O MapReduce gerava numerosos resultados intermediários antes de chegar à resposta final, e esses resultados precisavam ser armazenados em disco. Isso era muito mais demorado do que acessar diretamente a partir da memória.

**Hive**

Em 2009 foi introduzido o **Hive** que buscava resolver os desafios enfrentados pelos profissionais de engenheira de dados devido ao uso crescente do Hadoop. Ele visava atender ao perfil de profissionais que não necessariamente eram programadores(as), mas precisavam lidar com grandes quantidades de dados.

O Hive trouxe uma maneira mais fácil e acessível de gerenciar esses dados, tornando possível para um maior número de pessoas trabalhar com o Hadoop. Além disso, ele é uma camada de abstração que permite trabalhar com dados armazenados no Hadoop de forma semelhante ao trabalho com bancos de dados relacionais.

Ele permite o uso de comandos SQL para consultar e gerenciar esses dados, tornando mais fácil para usuários(as) com esse conhecimento trabalharem com grandes conjuntos de dados no Hadoop. Hive também permite organizar os dados em tabelas e especificar relações entre elas, assim como em um banco de dados relacional, facilitando a hierarquização dos trabalhos e melhorando a performance.

**Spark**

A solução para as cargas de trabalho no Hadoop serem escritas em disco mostrava que ele era mais adequado para processamento de dados em lote do que para interação em tempo real. Com o crescimento do Big Data e a necessidade de processar grandes conjuntos de dados em tempo real, começou a ser necessário encontrar novas abordagens para o processamento de dados.

O projeto **Spark** foi iniciado em 2011 com o objetivo de fornecer uma solução de processamento de dados e computação escalável horizontalmente que realizasse o maior número possível de operações em memória.

O Spark alcançou esse objetivo e melhorou significativamente o desempenho em comparação com o Hadoop. Porém, a configuração e instalação em um cluster de vários servidores tornou-se complexa devido aos muitos parâmetros de ajuste necessários para alcançar o melhor desempenho possível.

**Databricks**

A solução para a complexidade necessária para trabalhar com Spark veio através do **Databricks**. Seu início se deu em 2015 com a proposta de ser uma maneira mais fácil de trabalhar com o Spark.

O uso de uma interface amigável disponibilizada na plataforma do Databricks possibilitou:

* criar e configurar clusters de servidores;
* fazer conexões com diversos sistemas de arquivos;
* ter compatibilidade com linguagens como Python, Scala, SQL e Java;
* ter um desligamento automatizado;
* tornar muitas outras funcionalidades mais simples.

Toda essa solução foi feita para funcionar como serviço em nuvem e tem disponibilidade na Microsoft Azure, Amazon AWS e Google Cloud. Para quem deseja utilizar a forma gratuita, existe a **versão Community** da ferramenta que apresenta algumas limitações.

O Databricks é uma plataforma fácil de usar que oferece uma variedade de recursos para se trabalhar com dados. Ele permite a execução de consultas em ambientes de armazenamento de dados, como data lakes e data warehouses, além de permitir a execução de scripts para realizar tarefas específicas.

Databricks também oferece vários tipos de visualizações para explorar resultados de consultas, permitindo construir e compartilhar dashboards para que os(as) usuários(as) possam explorar e compreender melhor os dados.

O Databricks é uma plataforma de soluções para a indústria de dados. Dentre seus principais destaques temos:

1. **Consultas em suas bases com SQL** que possibilitam a construção de notebooks com a linguagem mais utilizada no mundo para banco de dados: o SQL. Assim, é possível realizar consultas e analisar dados mais recentes com uma velocidade incomparável.
2. **Uso nativo do Apache Spark** como principal mecanismo de análise unificado para Big Data e aprendizado de máquina que existe no mundo, sendo utilizado pelas grandes corporações. Ele explora o uso de memória e outras otimizações nas suas execuções. Anteriormente as empresas utilizavam o Hadoop.
3. **Notebooks colaborativos** em R, Python, Scala e SQL. Por ser uma plataforma que possui multilinguagem em seu cerne, é possível construir Notebooks híbridos nessas linguagens, podendo explorar rapidamente dados de acordo com a sua escolha e conhecimento.
4. **Visualização de dados nativos** de forma a integrar análise de dados em resultados por scripts e gráficos poderosos. Para isso, o Databricks organiza suas visualizações em painéis ricos para compartilhar insights com os seus gestores em um ambiente integrado.
5. **Integração com as ferramentas de BI** possibilitando a conexão diretamente com o Databricks na nuvem, permitindo a leitura dos dados contidos no data lake e contando com a otimização do Spark a todo momento.

Para compreender melhor o que é a plataforma Databricks e ter uma visão global das funcionalidades, acesse o site do [Databricks](https://www.databricks.com/" \t "_blank). Na [documentação oficial](https://docs.databricks.com/), você poderá encontrar as utilidades específicas e aprender como testar as funcionalidades dentro da plataforma.

No [Databricks Community](https://community.databricks.com/s/" \t "_blank), você encontrará uma rede de membros que utilizam as tecnologias Databricks, com grupos da comunidade e fóruns de discussão para tirar dúvidas e trocar experiências.

O **DBFS** (Databricks File System ou sistema de arquivos do Databricks) é um sistema de arquivos distribuído montado no workspace do Databricks e disponível nos clusters. O DBFS é uma abstração do armazenamento escalável de objetos e oferece os seguintes benefícios:

* Permite montar objetos de armazenamento para que você possa acessar os dados de maneira contínua sem a exigência de credenciais.
* Permite que você interaja com o armazenamento de objetos usando a semântica do diretório e do arquivo ao invés de URLs de armazenamento.
* Mantém arquivos no armazenamento de objetos para que não haja perda de dados depois de encerrar um cluster.

Todos(as) os(as) usuários(as) têm acesso de leitura e gravação aos objetos no armazenamento de objetos montado em DBFS, com exceção da raiz do DBFS.

**Raiz do DBFS**

O local de armazenamento padrão no DBFS é conhecido como a **raiz DBFS**, na qual vários dados são armazenados nos seguintes locais:

* **/FileStore**: arquivos de dados importados, gráficos gerados e bibliotecas carregadas.
* **/databricks-datasets**: conjuntos de dados públicos de exemplo.
* **/databricks-results**: arquivos gerados ao baixar os resultados completos de uma consulta.
* **/databricks/init**: scripts de inicialização globais e com nome de cluster (preteridos).
* **/user/hive/warehouse**: dados e metadados de tabelas não externas do Hive.

A raiz DBFS também contém dados incluindo metadados, credenciais e determinados tipos de logs que não estão visíveis e não podem ser acessados diretamente.

Referências:

* [What is the Databricks File System (DBFS)?](https://docs.databricks.com/data/databricks-file-system.html)
* [FileStore](https://docs.databricks.com/data/filestore.html)
* [Sample datasets](https://docs.databricks.com/data/databricks-datasets.html)

O **Apache Hive** é um sistema de data warehouse de código aberto usado, originalmente, para consultar e analisar grandes conjuntos de dados armazenados no Hadoop. Esse sistema foi criado pelo time de Infraestrutura de Dados do Facebook e adotado por diversos gigantes, como Yahoo, eBay e Facebook.



O Hive facilita a leitura, gravação e gerenciamento de grandes conjuntos de dados que residem em armazenamento distribuído usando SQL, projetado para trabalhar com grandes conjuntos de dados. No Apache Hive, podemos criar tabelas para armazenar dados estruturados para que mais tarde possamos processá-los.

Embora novas melhorias do projeto Hive nos permitam obter os resultados de consultas em tempo inferior a 1 segundo, ele não é projetado para processamento de transações online. Hive é melhor utilizado para tarefas tradicionais de data warehousing.

Basicamente, podemos dizer que para se trabalhar com dados em *batch* (lotes de dados em um intervalo de tempo específico) podemos usar o Hive. Mas quando estamos trabalhando com *stream* (fluxo de dados em tempo real) ou dados não estruturados temos que utilizar um dos módulos do Spark.

O Hive apresenta as seguintes características:

* Interpretação de instruções SQL para jobs MapReduce.
* Leitura de dados de arquivos estruturados e semi-estruturados no HDFS, baseando-se em metadados para simular tabelas de um banco de dados relacional.
* Não possui e nem é um SGBD (Sistema Gerenciador de Banco de Dados).
* Foi desenhado para melhor performance analisando grandes quantidades de dados que se encontram em clusters.
* É um sistema de Data Warehouse para o Apache Hadoop.

**Hive no Databricks**

Assim como o HDFS do Hadoop, o Hive também tornou-se a base para serviços de plataforma em nuvem de grandes fornecedores, como o Amazon Web Services (AWS). Por exemplo, no Databricks, o Hive é usado como base para gerenciar e processar grandes conjuntos de dados. Na AWS, o Glue Catalog é baseado no Hive e é usado para gerenciar dados em data lakes. Isso demonstra a importância e a ampla utilização do Hive.

Não há bancos de dados propriamente ditos no Hive, mas sim um conjunto de diretórios, arquivos e metadados registrados. No Hive dentro do Databricks também existe o conceito de tabelas externas, nas quais os dados não "pertencem" ao Hive, ou seja, se a tabela for deletada os arquivos se manterão lá e apenas metadados serão perdido

D

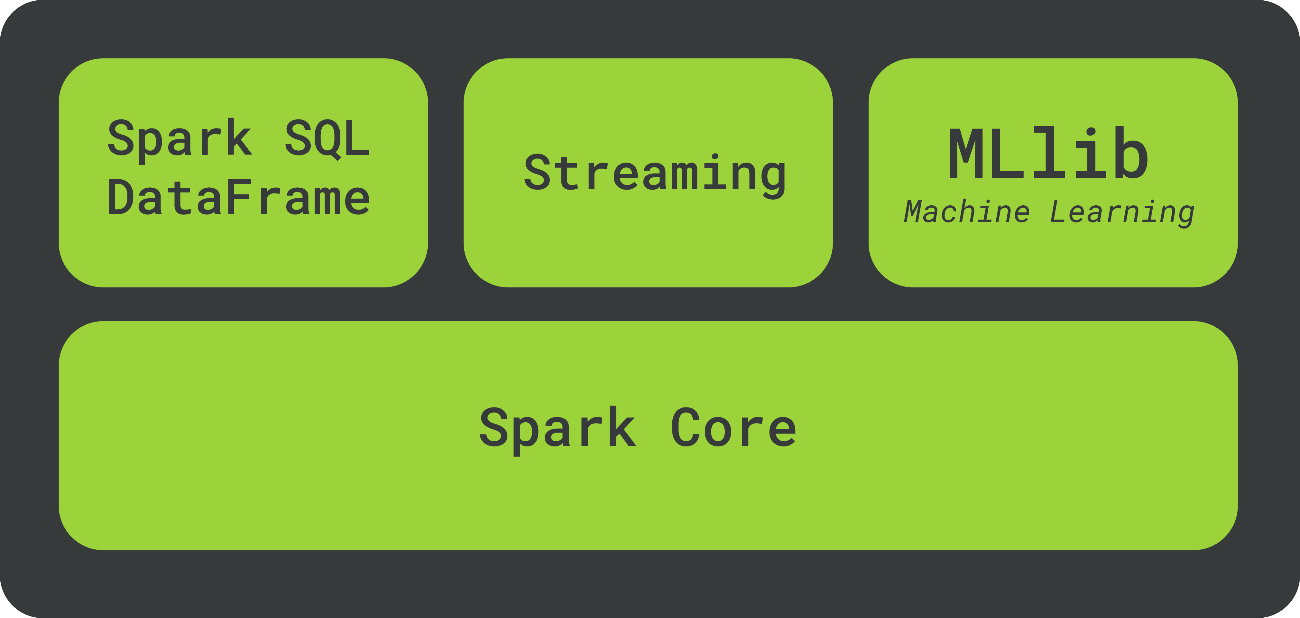
**Sobre o PySpark**

O Apache Spark é uma das melhores soluções quando o assunto é trabalhar com processamento de grandes quantidades de dados. Uma das interfaces do Apache Spark é o **PySpark** que permite criar aplicações Spark utilizando a linguagem Python.



Fonte: [PySpark](https://www.databricks.com/glossary/pyspark" \t "_blank)

Originalmente feito em Scala, o PySpark suporta a maioria dos recursos do Spark, incluindo Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) e Spark Core. Isso permite que as pessoas desenvolvedoras utilizem a linguagem Python para trabalhar com o Spark e seus recursos, tornando o processo de desenvolvimento mais fácil e acessível.



Fonte: Adaptado do [PySpark](https://spark.apache.org/docs/latest/api/python/" \t "_blank).

**SparkSQL**

É uma biblioteca do PySpark que possibilita o uso de SQL para fazer análises de dados estruturados. O SparkSQL é compatível com diversas fontes de dados, como CSV, [Parquet](https://www.alura.com.br/artigos/arquivos-parquet), JSON, Hive, entre outros.

**MLlib**

Biblioteca capaz de lidar com aprendizado de máquina no Spark, processando de forma paralela para armazenar e trabalhar com dados. Podemos utilizar essa biblioteca para trabalhar com modelos de classificação, regressão, clustering, etc.

**Streaming**

Permite a criação de aplicativos interativos e analíticos tanto para streaming de dados quanto para dados históricos. Ele oferece a capacidade de processar dados em tempo real, enquanto permite o armazenamento e análise de dados históricos.

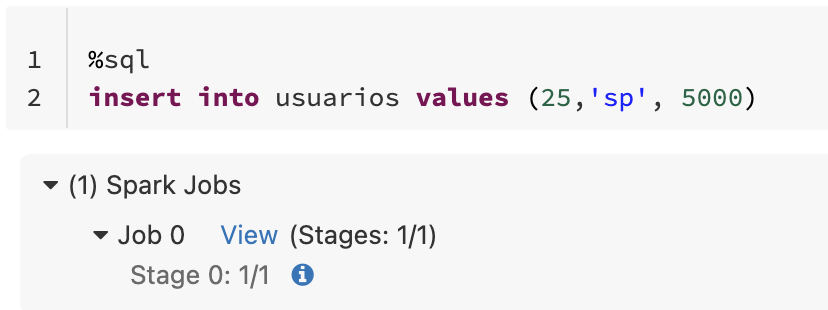
Ao usar o recurso de streaming no Spark, as pessoas desenvolvedoras podem herdar a facilidade de uso e as características de tolerância a falhas do Spark, tornando o processo de desenvolvimento mais simples e confiável.

**Spark Core**

É o mecanismo de execução geral subjacente para a plataforma Spark, sobre o qual todas as outras funcionalidades, como Spark SQL, DataFrame, Streaming e MLlib, são construídas. O Spark Core permite a distribuição de dados e cálculos em vários nós de um cluster, tornando possível o processamento paralelo de grandes conjuntos de dados. Ele também fornece uma arquitetura de gerenciamento de tarefas e recursos, que gerencia a distribuição de tarefas e aloca recursos, garantindo tolerância a falhas e escalabilidade.

**Spark Jobs**

Como podemos observar, mesmo utilizando o Hive vemos que na saída, logo abaixo de cada célula do notebook, aparece o label **Spark Jobs**. Ele pode ser expandido como mostra a figura abaixo.



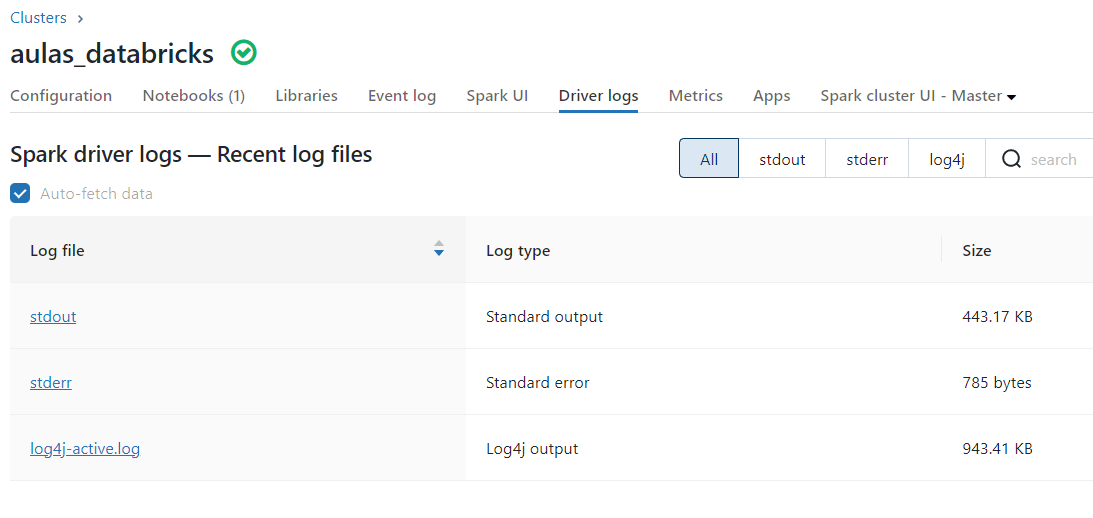
O Spark tem algumas [definições listadas](https://spark.apache.org/docs/1.1.0/cluster-overview.html). Dentre elas temos que um **Job** é:

“Uma computação paralela que consiste em várias tarefas que são geradas em resposta a uma ação do Spark (por exemplo, save, collect); você verá esse termo sendo usado nos logs do driver”.

Ou seja, sempre que uma ação que utiliza o Spark por baixo dos panos for executada, teremos como verificar o seu histórico. Ao clicar em Spark Jobs, é possível verificar que a ação ilustrada na figura acima foi um job de um estágio apenas e que ele foi completado. Por isso, tem-se entre parênteses: (1/1).

Um job em computação paralela consiste de múltiplas tarefas que são produzidas em resposta a uma ação do Spark, como: salvar, transformar, visualizar. É possível também observar todos os jobs executados pelos logs do driver. Para acessá-los, basta ir em **Compute**, no menu lateral esquerdo, clicar sobre o nome do seu cluster (aulas\_databricks) e depois em **Driver logs**.

Dentro da página dos logs é possível ver o histórico de tudo o que foi ou está sendo executado em detalhes. Abaixo temos um exemplo dos logs:



Mas pode vir a pergunta: se estou executando o Hive, por que aparece Spark Jobs? Isso acontece porque o Hive se beneficia do Spark, principalmente com relação a paralelização. De forma geral, quando estamos falando de Big Data, o Hive pode armazenar os dados no HDFS. Mas quando estamos utilizando o Hive com Spark, ao invés do Hive armazenar dados no Hadoop, ele os armazena no Spark.

A razão pela qual as pessoas usam o Spark em vez do Hadoop é que ele é um **banco de dados totalmente em memória**. Portanto, os trabalhos do Hive serão executados muito mais rápido por lá. Isso leva as pessoas programadoras a usar um banco de dados comum, caso a empresa para a qual trabalham execute predominantemente o Spark. Também é possível escrever programas no Spark e usá-los para se conectar aos dados do Hive, ou seja, ir na direção oposta.

**conhecendo o Apache Spark**

O **Apache Spark** é uma ferramenta unificada cujo objetivo primordial consiste em processar grandes conjuntos de dados (Big Data) de forma paralela e distribuída. Foi originalmente desenvolvido na Universidade de Berkley na Califórnia e posteriormente doado para a Fundação Apache que o mantém até os dias de hoje.

O Spark possui armazenamento em memória para cálculos computacionais intermediários, tornando-o mais rápido que o MapReduce. Além de possuir módulos internos para streaming, SQL, machine learning, processamento de grafos e ser capaz de interagir com linguagens como Scala, Python, Java e R.

O Spark foi criado baseando-se em quatro características chaves:

* **Velocidade**: consegue executar tudo com rapidez.
* **Facilidade de uso**: menores dificuldades e complicações ao ser utilizado.
* **Modularidade**: possui unidades funcionais individuais.
* **Extensibilidade**: tem a possibilidade de adicionar novas funcionalidades.

O Apache Spark tem sua arquitetura baseada no conjunto de dados distribuídos resiliente (ou RDD - Resilient Distributed Dataset). É um conjunto múltiplo de itens contendo dados apenas de leitura distribuídos sobre um conjunto de máquinas tolerantes a falhas. Assim, os RDDs funcionam como um conjunto de ferramentas para programas distribuídos que oferecem uma forma restrita de memória compartilhada distribuída.

Dentro do Apache Spark, o fluxo de trabalho é gerenciado como um gráfico acíclico dirigido (DAG). Os nós representam RDDs, enquanto as terminações representam as operações realizadas sobre eles.

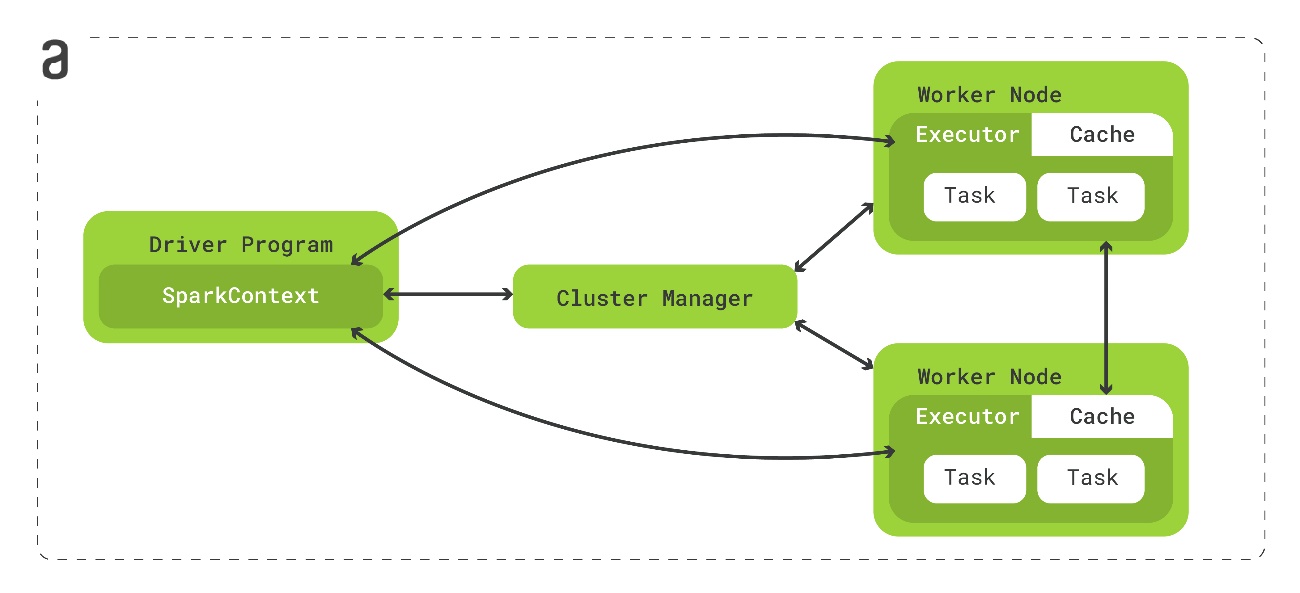
O Apache Spark requer um gerenciador de cluster e um sistema de armazenamento distribuído. Para o gerenciamento de cluster, o Spark suporta standalone, Hadoop YARN, Apache Mesos ou Kubernetes.

Standalone é um cluster Spark nativo no qual é possível lançar um cluster manualmente ou usar os scripts de lançamento fornecidos pelo pacote de instalação. Também é possível executar esses daemons (programas executados nos processos em segundo plano) em uma única máquina para testes.

Para armazenamento distribuído, o Spark pode interagir com uma grande variedade de sistemas, incluindo: Alluxio, Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, ou uma solução personalizada pode ser implementada.

O Spark também suporta um modo local pseudo-distribuído, normalmente usado apenas para fins de desenvolvimento ou teste. Nesse modo, o armazenamento distribuído não é necessário e o sistema de arquivo local pode ser usado em seu lugar. Em tal cenário, o Spark é executado em uma única máquina com um executor por núcleo de CPU.

O Driver é um processo que executa o programa principal de sua aplicação Spark e cria o SparkContext que coordena a execução dos trabalhos. Os executores (executers) são processos em execução nos nós de trabalho (worker nodes) do cluster que são responsáveis por executar as tarefas que o Driver lhes atribuiu. O gerente do cluster (como o Mesos ou YARN) é responsável pela alocação de recursos físicos para as aplicações Spark.



Fonte: Adaptado de [Apache Spark](https://spark.apache.org/docs/latest/cluster-overview.html#components)

## Pontos de Entradas

Toda aplicação Spark precisa de um ponto de entrada que permita a integração com fontes de dados, além de realizar determinadas operações como leitura e gravação de dados. Na versão Spark 1.0 foram introduzidos três pontos de entrada:

* **SparkContext**

É usado pelo Driver Process para estabelecer uma comunicação com o cluster e com os gerenciadores de recursos, a fim de coordenar e executar jobs. O SparkContext também permite o acesso às demais funcionalidades disponíveis, a saber: SQLContext e HiveContext.

* **SQLContext**

Ponto de entrada para SparkSQL que é um módulo para o processamento de dados estruturados. Uma vez que o SQLContext é inicializado, o(a) usuário(a) tem a possibilidade de realizar várias operações utilizando a linguagem SQL sobre Datasets e Dataframes.

* **HiveContext**

Tem todas as funcionalidades do SQLContext. Além de recursos adicionais tais como a capacidade de escrever consultas usando o analisador HiveQL e a de ler dados de tabelas Hive.

No Spark 2.0 foi introduzido um novo ponto de entrada que combina essencialmente todas as funcionalidades disponíveis nas três entradas anteriores.

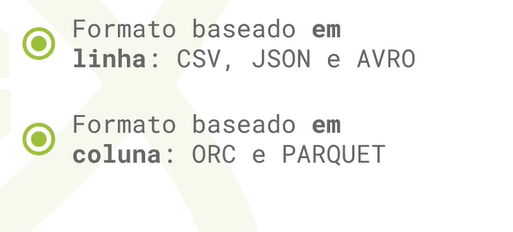
* **SparkSession**

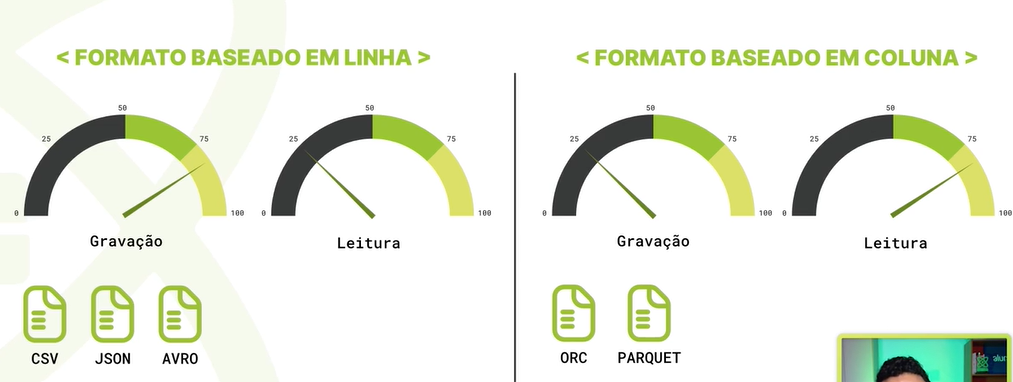
Este ponto de entrada se tornou uma opção do Spark para trabalhar com RDD, DataFrame e Dataset. Ele é capaz de substituir o SQLContext e o HiveContext. O **spark**, que é o objeto do SparkSession, é a variável padrão disponível no spark-shell e pode ser criado usando o padrão do construtor SparkSession. No Databricks já temos uma SparkSession iniciada e que pode ser acessada pela variável **spark**, como veremos nos próximos vídeos.

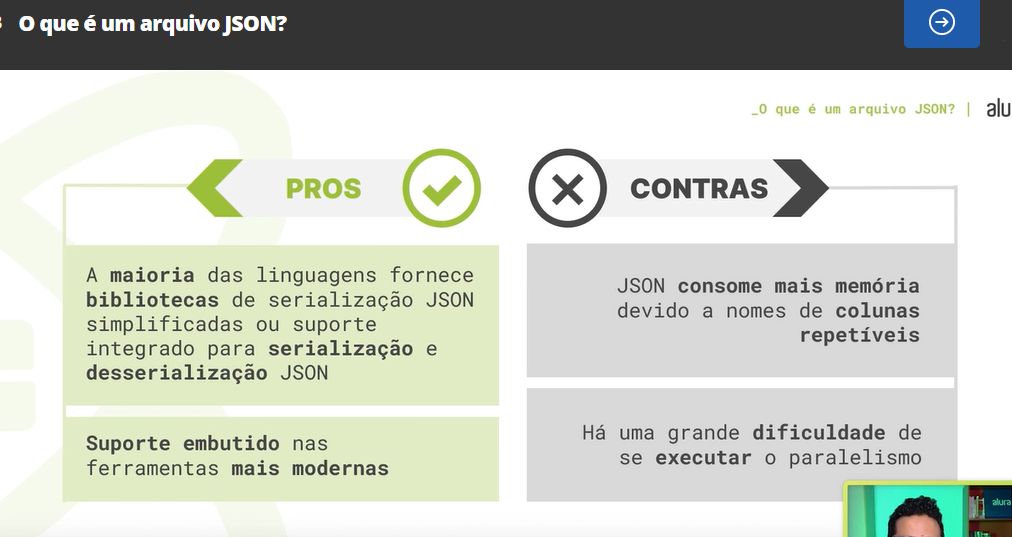
O **DBFS** é onde os dados trabalhados no Databricks serão armazenados.

O **dbutils** facilita a execução de tarefas, como por exemplo a visualização de diretórios, criação de pastas, entre outros.

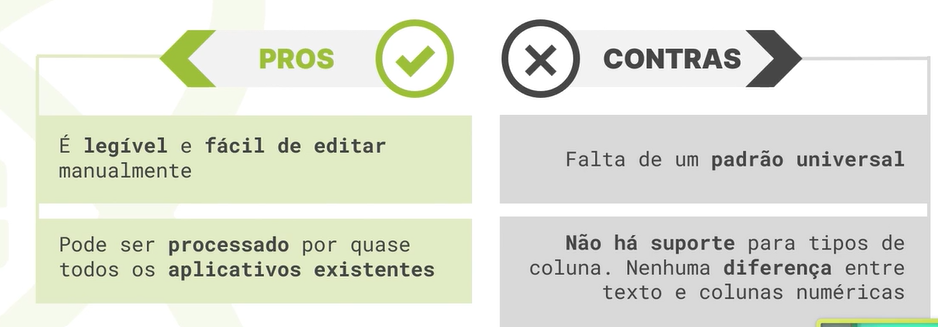
<https://docs.databricks.com/dbfs/index.html>



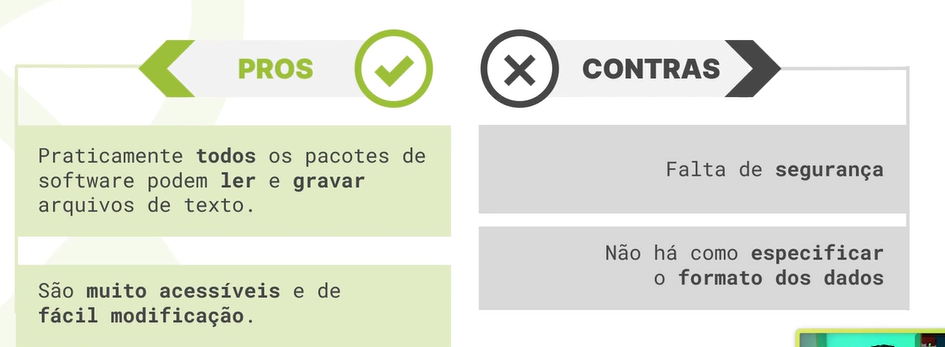




CSV



txt



# escolhendo o codec

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/databricks-formatos-tipos-arquivos/task/124109/next)

As compressões de arquivos são uma ótima forma de economizar espaço em disco e melhorar o desempenho na transferência de dados. No PySpark, existem vários formatos de compressão que podem ser utilizados em arquivos CSV, cada um com suas próprias vantagens e desvantagens.

| **Formato** | **Características** |
| --- | --- |
| **bzip2** | - Baseado no algoritmo de compressão Burrows-Wheeler.  - Altamente eficiente em termos de compressão.  - Útil em casos em que o espaço em disco é limitado e a velocidade de compressão  não é um fator crítico.  - Mais lento em comparação com outros formatos de compressão, como gzip ou snappy. |
|  |  |
| **gzip** | - Baseado no algoritmo de compressão DEFLATE.  - Amplamente suportado em vários sistemas operacionais e ferramentas de  processamento de dados.  - Fácil de usar.  - Oferece uma boa relação de compressão-tempo.  - Útil em casos em que a facilidade de uso é prioridade e a velocidade de compressão  não é um fator crítico. |
|  |  |
| **lz4** | - Opção bastante conhecida.  - Baseado no algoritmo LZ4.  - Projetado para ser rápido e eficiente.  - Boa relação de compressão-tempo.  - Amplamente utilizado em sistemas distribuídos, como Hadoop.  - Útil em casos em que a velocidade de compressão é prioridade e a eficiência  de compressão não é um fator crítico. |
|  |  |
| **snappy** | - Desenvolvido pela Google.  - Projetado para ser rápido e eficiente.  - Boa relação de compressão-tempo.  - Amplamente utilizado em sistemas distribuídos, como Hadoop e Apache Cassandra.  - Útil em casos em que a velocidade de compressão e eficiência de compressão  são ambas importantes. |
|  |  |
| **deflate** | - Algoritmo de compressão de dados.  - Amplamente utilizado em arquivos ZIP e gzip.  - É mais eficiente em termos de compressão do que o gzip, mas não em termos de  velocidade.  - Útil em casos em que a eficiência de compressão é prioridade e a velocidade  de compressão não é um fator crítico. |

Para comparar as compressões, você pode medir o desempenho de cada uma delas usando métricas como:

* taxa de compressão: a razão entre o tamanho do arquivo comprimido e o tamanho do arquivo original;
* velocidade de compressão: tempo necessário para comprimir o arquivo; e
* velocidade de descompressão: tempo necessário para descomprimir o arquivo.

Em resumo, a escolha do formato de compressão para arquivos CSV no PySpark dependerá de suas necessidades específicas. Cada formato (bzip2, gzip, lz4, snappy e deflate) tem suas próprias vantagens e desvantagens em termos de eficiência de compressão e velocidade de compressão. É importante testar e medir o desempenho para escolher o melhor para seu caso de uso.

O formato de arquivo TXT pode apresentar mais algumas vantagens, desvantagens e casos de uso para o contexto de Big Data com o PySpark. Vejamos cada uma delas:

### Vantagens:

* Compatibilidade com diferentes sistemas operacionais e aplicativos, o que facilita a importação e processamento de dados.
* Simplicidade e facilidade para entender, o que pode ser útil para a realização de tarefas de limpeza e pré-processamento de dados.
* Flexibilidade para o formato ser lido por muitas ferramentas e bibliotecas, incluindo o PySpark, tornando-o fácil de ser importado e processado.
* Facilidade de uso para ler e escrever, tornando-o uma escolha popular para armazenamento de dados brutos ou de log.

### Desvantagens

* Não possuir uma estrutura definida, o que pode dificultar o processamento de grandes volumes de dados.
* Falta de recursos avançados, como criptografia, que podem ser úteis para garantir a segurança dos dados.
* Limitação para trabalhar com colunas específicas, pois não possui uma estrutura bem definida, dificultando a seleção de colunas específicas para trabalhar.
* Escalabilidade limitada porque o formato não é otimizado para lidar com grandes volumes de dados e pode se tornar ineficiente em escalas maiores.
* Baixa performance por não ser projetado para alta performance e pode ser lento ao processar grandes volumes de dados.

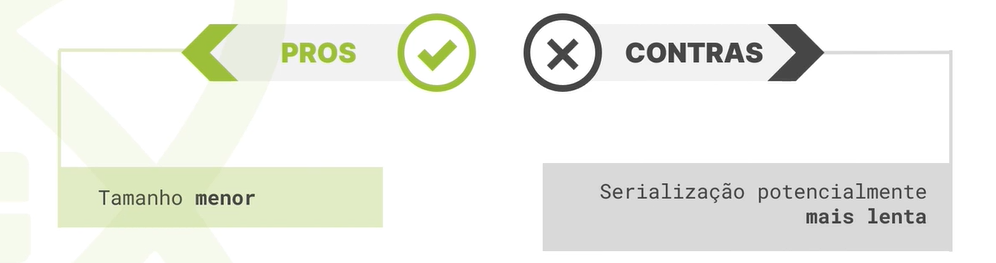
### Casos de uso:

* Armazenamento de logs de sistema que podem ser facilmente importados e processados ​​usando o PySpark.
* Os dados brutos podem ser armazenados em formato de arquivo TXT antes de serem limpos e transformados para outro formato mais apropriado para análise.
* Armazenamento de dados coletados de redes sociais que podem ser facilmente importados e processados ​​usando o PySpark para análise de dados de sentimento e tendências.

AVRO

AVRO é um formato de dados open source, ou código aberto, em que é possível agrupar os dados serializados junto com seu schema em um mesmo arquivo, o que facilita a gestão e organização dos dados. Ele utiliza o JSON para definir a tipagem dos dados e serializá-los em formato binário compacto.

Outro detalhe é que este formato é baseado em linhas, e não em colunas. Portanto, se sua intenção é gravar o arquivo de maneira otimizada, AVRO é uma excelente opção. Além do mais, seu uso principal é no \* Apache Hadoop, que utiliza arquitetura HDFS, ou \*Hadoop Distributed File System ("Sistema de Arquivos Distribuídos Hadoop", em tradução livre), assim como o Apache Spark. Como sua maneira de salvar os dados é em formato binário compacto, ele é ideal para trabalhar com grandes volumes de dados.



Lembre-se que serialização é o processo de transformar os dados ou objetos em um formato possível de ser transferido e armazenado.

# avroSchema

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/databricks-formatos-tipos-arquivos/task/124115/next)

O **avroSchema** é um formato de esquema de dados utilizado para descrever o formato de arquivos Avro. Ele é usado em conjunto com o Avro Serialization para serializar e desserializar dados em formatos Avro.

No PySpark, a opção avroSchema é usada para especificar o esquema dos dados a serem lidos de um arquivo Avro. Ele é especificado na leitura de dados usando a função spark.read.format("avro").load(<arquivo>) e passando o esquema Avro como parâmetro na opção avroSchema.

Um exemplo de utilização da opção avroSchema no PySpark é ler um arquivo Avro de dados de clientes. Nele o esquema inclui informações como nome, endereço, data de nascimento, etc., como podemos observar no código a seguir:

# Esquema Avro para dados de clientes

avro\_schema = """

{

"type": "record",

"name": "Cliente",

"fields": [

{

"name": "nome",

"type": "string"

},

{

"name": "endereco",

"type": "string"

},

{

"name": "data\_nascimento",

"type": "string"

},

{

"name": "idade",

"type": "int"

}

]

}

"""

# Leitura dos dados do arquivo Avro usando o esquema Avro

df = spark.read.format("avro")\

.option("avroSchema", avro\_schema)\

.load("<caminho\_arquivo\_avro>")

# Exibição dos dados lidos do arquivo Avro

display(df)COPIAR CÓDIGO

No Avro Schema temos os seguintes campos:

* **type**: especifica o tipo de registro do esquema. No exemplo acima, o tipo é "record", indicando que o esquema descreve um registro de dados.
* **name**: especifica o nome do registro. No nosso exemplo, o nome é "Cliente", indicando que o registro descreve dados de clientes.
* **fields**: é uma lista de campos que compõem o registro. Cada campo é descrito como um objeto JSON com duas propriedades: "name" e "type". No exemplo acima, o registro "Cliente" tem 4 campos: "nome", "endereco", "data\_nascimento" e "idade".

Estes campos do Avro Schema também facilitam a leitura e processamento dos dados por parte de ferramentas como o PySpark.

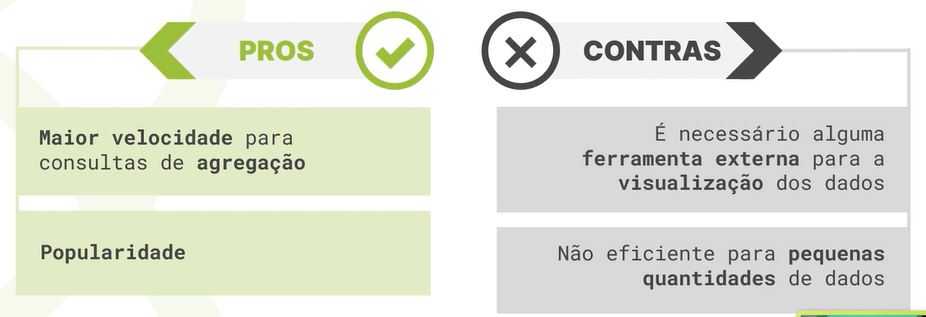
A evolução do avroSchema permite adicionar, remover ou modificar campos. Isso é possível porque o Avro usa o conceito de **evolução de esquemas** que pode ser realizada para:

* Adicionar novos campos a um registro. Neste caso, você pode simplesmente adicioná-los ao final da lista de campos no esquema Avro. Ferramentas como o PySpark são capazes de lidar com registros que possuem campos adicionais, preenchendo-os com valores padrão ou ignorando-os conforme necessário.
* Remover ou modificar um campo existente seguindo algumas regras de evolução de esquemas. De acordo com essas regras, você pode remover ou modificar o tipo de um campo, mas não pode mudar o nome ou a ordem dos campos. Além disso, não é possível mudar o tipo de um campo para um que seja incompatível. Por exemplo: mudar um campo "int" para "string".

Para garantir a compatibilidade dos dados ao longo do tempo, recomenda-se seguir essas regras ao fazer modificações em um esquema Avro. Além disso, é importante sempre testar o comportamento de ferramentas, como o PySpark, com novos esquemas para garantir a correta leitura e o processamento dos dados.

Recomenda-se ler a [documentação do Avro Schema](https://avro.apache.org/docs/current/spec.html) e a do [PySpark](https://spark.apache.org/docs/latest/sql-data-sources-avro.html" \t "_blank) para leitura de arquivos Avro.

O PARQUET é um formato de armazenamento colunar de código aberto pertencente ao ecossistema do Apache Hadoop. Ele possui esquemas eficientes de compactação e codificação dos dados, com desempenho aprimorado para lidar com dados complexos e em grande quantidade





Como vantagem, podemos citar a maior velocidade para consultas de agregação e maior popularidade, o que faz com que surjam novas ferramentas para lidar com este formato. Já a desvantagem fica por conta do fato de que é necessário uma ferramenta externa para a visualização dos dados, uma vez que o PARQUET é um formato binário. Além do mais, é ineficaz para pequenas quantidades de dados.

Na tabela a seguir, temos uma comparação, desenvolvida pelo Databricks, entre os formatos CSV e PARQUET:

| **conjunto de dados** | **tamanho no Amazon S3** | **tempo de execução da consulta** | **dados digitalizados** | **custo** |
| --- | --- | --- | --- | --- |
| dados armazenados como arquivos CSV | 1 TB | 236 segundos | 1,15 TB | $5.75 |
| dados armazenados no formato Apache Parquet | 130 GB | 6,78 segundos | 2,51 GB | $0.01 |
| poupança | 87% menos ao usar PARQUET | 34x mais rápido | 99% menos dados digitalizados | 99,7% de economia |

Note que os dados armazenados em PARQUET economizam espaço, são mais rápidos e mais baratos.

# Parquet ou CSV

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/databricks-formatos-tipos-arquivos/task/124118/next)

**Parquet** e **CSV** são formatos de arquivo utilizados para armazenar grandes quantidades de dados. Embora ambos tenham essa característica em comum, o Parquet é mais apropriado para o contexto de Big Data devido às suas vantagens em termos de desempenho e eficiência de armazenamento.

O Parquet é um formato de **arquivo colunar**, o que significa que ele armazena dados em colunas ao invés de linhas, ao contrário do CSV. Isso permite que as consultas sejam realizadas com mais eficiência, pois as informações necessárias para a consulta são todas armazenadas juntas, em vez de estarem dispersas em todo o arquivo.

O formato de arquivo Parquet oferece outras vantagens como:

* Segurança, pois suporta criptografia de coluna, o que significa que os dados sensíveis podem ser protegidos de forma eficiente sem afetar o desempenho das consultas.
* Performance excepcional em consultas paralelas, ou seja, as consultas podem ser executadas mais rapidamente em grandes quantidades de dados.
* Escalabilidade para armazenar grandes quantidades de dados, garantindo uma boa performance mesmo em contextos de Big Data.

Outra vantagem do Parquet é que ele é altamente otimizado para reduzir o tamanho dos dados, o que resulta em menor espaço de armazenamento e melhor desempenho em consultas. Além disso, o Parquet é compatível com a maioria das ferramentas de Big Data, incluindo Apache Spark, Apache Hive e Apache Impala. Logo, você pode aproveitar a escalabilidade e a performance dessas ferramentas ao trabalhar com dados em formato Parquet.

O CSV é um formato simples e amplamente utilizado. Em comparação ao Parquet, ele não é tão eficiente quando se trata de armazenamento e desempenho em consultas. Também não é tão escalável e pode ter problemas de desempenho ao lidar com grandes quantidades de dados.

O Databricks possui um arquivo que mostra mais alguns benefícios e um comparativo entre esses dois formatos e está disponível em [“Parquet”](https://www.databricks.com/glossary/what-is-parquet).

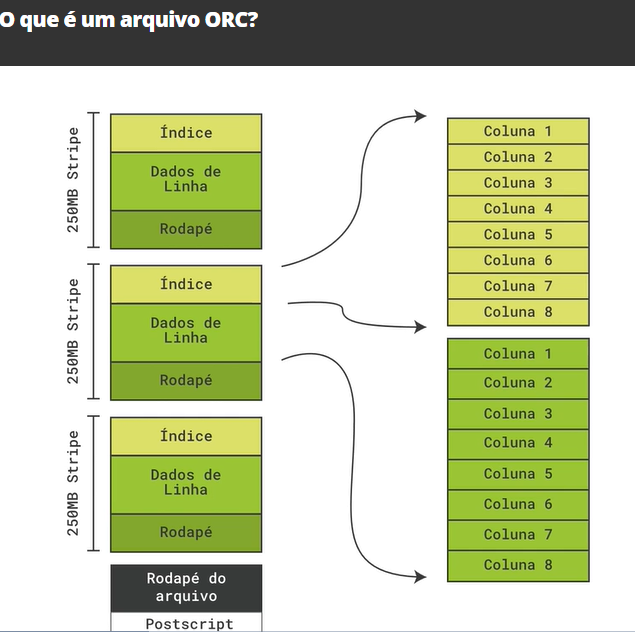
Em resumo, o Parquet é uma escolha melhor do que o CSV para o contexto de Big Data devido à sua eficiência de armazenamento, desempenho em consultas e compatibilidade com ferramentas. Ao escolher o Parquet, você pode aproveitar a escalabilidade e a performance dos seus dados, tornando-os mais fáceis de trabalhar e acessar.

ORC

 Ele fornece formas altamente eficientes para armazenar dados no formato colunar. Foi lançado originalmente em 2013 com o objetivo de acelerar o Hive e reduzir o armazenamento no Apache Hadoop.

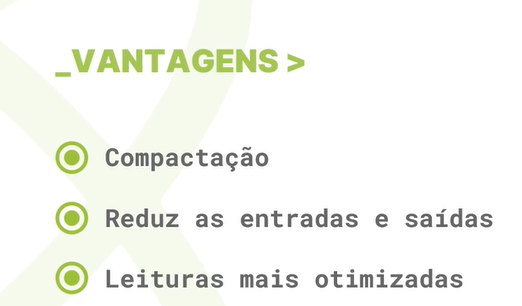
O Apache Hive é um sistema de Data Warehouse para o Apache Hadoop. O Hive permite fazer o resumo dos dados, consultas e análises.

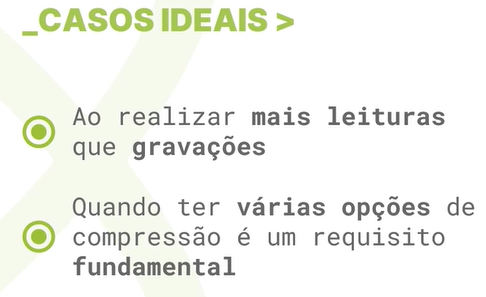
O formato ORC armazena coleções de linhas em um único arquivo de forma colunar. Isso significa que os dados de uma mesma coluna são armazenados juntos e esse tipo de armazenamento permite que seja realizado o processamento paralelo em um cluster. Além do mais, este tipo de arquivo utiliza os mesmo princípios de retornar apenas o necessário no sistema de arquivos PARQUET, que vimos anteriormente. Sendo assim, podemos afirmar que o ORC é uma evolução do PARQUET, que trabalha de forma mais rápida e consome menos armazenamento.

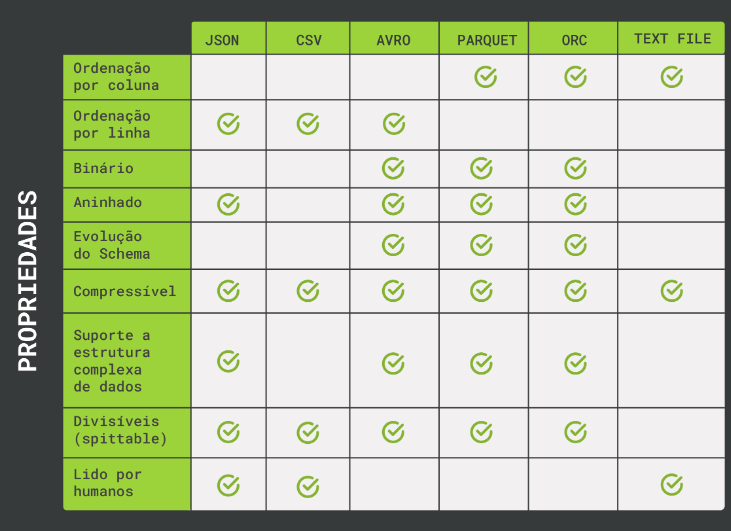


Lembre-se que os metadados são informações que acrescem aos dados e possuem como objetivo informar sobre eles para facilitar sua organização.

O tamanho padrão do stripe é de 250 MB, e é este tamanho que permite que as leituras sejam feitas de forma eficiente e otimizada. A seção do postscript fornece as informações necessárias para interpretar o restante do arquivo, incluindo o comprimento das seções de rodapé e metadados do arquivo, a versão do arquivo e o tipo de compactação geral utilizada







O Databricks é uma plataforma desenvolvida principalmente para a utilização do Spark. As características que tornam o Spark tão relevante são:

* operar em memória e em cluster;
* particionar dados em diferentes clusters;
* permite realizar paralelismo.

Partição é uma maneira de dividir um grande conjunto de dados em pedaços menores com base em uma ou mais chaves de partição. Ou seja, definimos uma coluna e os dados serão divididos com base nos valores dessa coluna.

As transformações em dados particionados são executadas mais rapidamente, porque ocorrem paralelamente em cada partição.