

PySpark na Prática

Engenharia de Dados
do Zero à Produção



Elison Melgueiro

AMOSTRA GRATUITA



PySpark na Prática

Engenharia de Dados do Zero à Produção

Esta amostra foi preparada para que você conheça a proposta editorial, a linguagem didática e a profundidade técnica do livro antes da compra.

Ela inclui o sumário completo e o Capítulo 1 integral - O que é Apache Spark?

32 capítulos

jornada progressiva

478 páginas

edição digital completa

Do zero à produção

conceitos + prática

Autor: Elison P Melgueiro

Uso exclusivo para avaliação. Venda e redistribuição desta amostra não são autorizadas.

PySpark na Prática

Engenharia de Dados do Zero à Produção

Elison P Melgueiro

1ª edição • 2026

© 2026 **Elison P Melgueiro**

Todos os direitos reservados.

PySpark na Prática — Engenharia de Dados do Zero à Produção

O texto, a organização editorial, os exercícios, as explicações didáticas, os diagramas e demais elementos autorais desta obra são protegidos por direitos autorais.

Os códigos-fonte e materiais técnicos disponibilizados no repositório oficial de apoio possuem licenciamento próprio, conforme os arquivos de licença do repositório.

Apache Spark, PySpark, Delta Lake, Databricks, GitHub, Python e demais marcas citadas pertencem aos seus respectivos titulares.

Primeira edição: 2026.

Sumário

- Capítulo 1 — O que é Apache Spark?.....1
- Capítulo 2 — Como o Apache Spark Funciona.....23
- Capítulo 3 — Preparando o Laboratório Linux para PySpark.....57
- Capítulo 4 — Conhecendo os DataFrames com PySpark.....89
- Capítulo 5 — Lendo Dados com PySpark.....125
- Capítulo 6 — Schemas e Tipos de Dados no PySpark.....184
- Capítulo 7 — Selecionando e Filtrando Dados com PySpark.....221
- Capítulo 8 — Criando e Transformando Colunas.....270
- Capítulo 9 — Qualidade de Dados com PySpark.....280
- Capítulo 10 — Agregações e Indicadores.....289
- Capítulo 11 — Joins com PySpark.....297
- Capítulo 12 — Window Functions.....305
- Capítulo 13 — ETL e ELT com PySpark.....312
- Capítulo 14 — Data Lake, Lakehouse e Arquitetura em Camadas.....320
- Capítulo 15 — Parquet em Profundidade.....328
- Capítulo 16 — Entendendo a Execução do Spark.....335
- Capítulo 17 — Particionamento e Paralelismo.....342
- Capítulo 18 — Otimização de Joins e Data Skew.....349
- Capítulo 19 — Cache, Persist, Catalyst e AQE.....357
- Capítulo 20 — Spark SQL.....364
- Capítulo 21 — Introdução ao Delta Lake.....371
- Capítulo 22 — Operações com Delta Lake.....378
- Capítulo 23 — Databricks na Prática: do Workspace ao Pipeline Profissional.....385
- Capítulo 24 — Jobs e Workflows.....406
- Capítulo 25 — Governança de Dados e Unity Catalog.....413
- Capítulo 26 — Processamento Incremental.....420
- Capítulo 27 — Structured Streaming.....427
- Capítulo 28 — Tratamento de Erros e Observabilidade.....434
- Capítulo 29 — Testando Pipelines PySpark.....441
- Capítulo 30 — PySpark em Produção.....448
- Capítulo 31 — Projeto Final — Construindo um Lakehouse Completo.....455
- Capítulo 32 — Publicando e Apresentando o Projeto.....462
- Referências Bibliográficas e Documentação Técnica.....470

Capítulo 1 — O que é Apache Spark?

Objetivo do capítulo: compreender por que tecnologias de processamento distribuído se tornaram necessárias, o que é Apache Spark, onde PySpark entra nesse ecossistema e quando essas tecnologias fazem sentido em projetos de Engenharia de Dados.

1.1 Antes de aprender Spark, precisamos entender o problema

Imagine que uma universidade precise analisar as informações acadêmicas de **5 mil estudantes**.

Uma aplicação Python tradicional provavelmente conseguiria processar esses dados sem grandes dificuldades.

Agora imagine que essa mesma instituição faça parte de uma grande rede educacional contendo:

- milhões de estudantes;
- registros de matrículas;
- notas;
- acessos aos ambientes virtuais;
- pagamentos;
- frequência;
- atividades acadêmicas;
- dados históricos acumulados durante vários anos.

O problema deixou de ser apenas **armazenar informações**.

Precisamos também conseguir **processá-las em tempo adequado**.

Esse mesmo desafio aparece diariamente em empresas de comércio eletrônico, bancos, indústrias, operadoras de telecomunicações, plataformas de streaming, redes sociais e inúmeras outras organizações.

Uma grande empresa de comércio eletrônico, por exemplo, pode precisar processar milhões de registros para responder perguntas aparentemente simples:

- Quanto vendemos hoje?
- Qual região apresentou maior faturamento?
- Quais produtos estão vendendo mais?
- Quantos clientes realizaram uma compra?
- Existem transações duplicadas?
- Quais pedidos ainda não foram entregues?
- Qual foi o comportamento das vendas nos últimos meses?

Quando o volume de dados cresce significativamente, executar todo o processamento em uma única máquina pode se tornar lento, caro ou até inviável.

É nesse cenário que começamos nossa jornada pelo **Apache Spark**.

1.2 O crescimento dos dados

Durante muitos anos, sistemas computacionais foram construídos considerando que grande parte do processamento poderia acontecer em um único computador ou servidor.

Esse modelo funciona muito bem em inúmeras situações.

Imagine um computador contendo:

```
graph TD; A[Processador] --- B[Memória RAM]; B --- C[Armazenamento];
```

Processador
|
Memória RAM
|
Armazenamento

Uma aplicação recebe os dados, realiza determinado processamento e apresenta o resultado.

Entretanto, à medida que o volume de informações aumenta, essa máquina passa a precisar de mais:

- memória;
- processamento;

- armazenamento;
- capacidade de leitura e escrita.

Uma solução aparentemente simples seria comprar uma máquina cada vez mais poderosa.

Essa estratégia é conhecida como **escalabilidade vertical**.

1.3 Escalabilidade vertical

Escalar verticalmente significa aumentar os recursos disponíveis em uma única máquina.

Por exemplo:

Servidor inicial

CPU: 4 núcleos
RAM: 16 GB
Disco: 500 GB

Depois de algum tempo:

Servidor maior

CPU: 32 núcleos
RAM: 128 GB
Disco: 4 TB

O servidor tornou-se muito mais poderoso.

Essa estratégia pode funcionar perfeitamente para muitos sistemas.

Porém, existe um limite.

Quanto mais potente a máquina, normalmente maior será seu custo. Além disso, eventualmente podemos chegar a um ponto em que continuar aumentando seus recursos deixa de ser economicamente ou tecnicamente interessante.

Entenda assim

É como tentar transportar cada vez mais mercadorias utilizando apenas um caminhão.

Primeiro utilizamos um caminhão pequeno.

Depois compramos um maior.

Depois outro ainda maior.

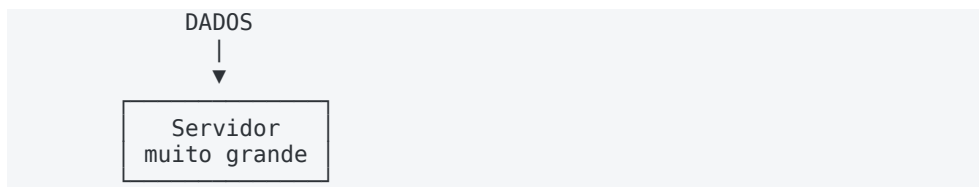
Em algum momento, talvez seja mais eficiente utilizar **vários caminhos trabalhando simultaneamente**.

Essa segunda abordagem nos leva à escalabilidade horizontal.

1.4 Escalabilidade horizontal

Escalar horizontalmente significa adicionar **mais máquinas** para dividir o trabalho.

Em vez de:



podemos ter:



Agora o processamento pode ser distribuído entre diferentes computadores.

Chamamos esse conjunto de computadores de **cluster**.

Conceito importante — Cluster

Um cluster é um conjunto de computadores que trabalham de forma coordenada para executar determinadas tarefas.

Em ambientes de processamento de dados, diferentes máquinas do cluster podem processar diferentes partes de um conjunto de dados.

1.5 Uma analogia: corrigindo um milhão de provas

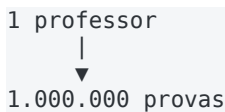
Vamos utilizar uma analogia que nos acompanhará durante os próximos capítulos.

Imagine uma universidade que realizou uma grande prova nacional.

Existem:

1.000.000 de provas para corrigir.

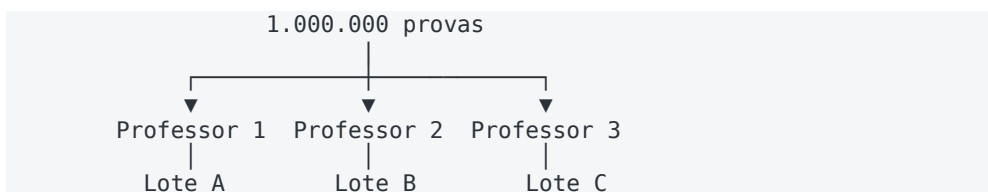
Se apenas um professor for responsável pela correção, o processo poderá levar muito tempo.



Podemos contratar um professor extremamente rápido.

Mesmo assim, existe um limite para a quantidade de provas que uma única pessoa consegue corrigir.

Outra estratégia seria dividir as provas entre vários professores.



Cada professor trabalha em uma parte do problema.

Ao final, os resultados são reunidos.

Essa é uma maneira simplificada de compreender o **processamento distribuído**.

1.6 O que é processamento distribuído?

Processamento distribuído é uma abordagem na qual um trabalho computacional pode ser dividido entre diferentes recursos computacionais que cooperam para produzir um resultado.

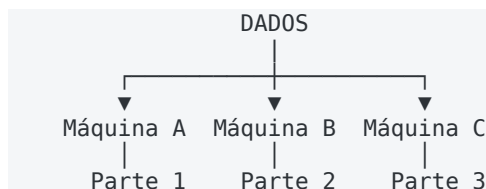
Em vez de uma única máquina realizar todo o processamento:

Máquina

```
graph TD; M[Máquina] --- D1[dado 1]; M --- D2[dado 2]; M --- D3[dado 3]; M --- D4[dado 4]; M --- D5[dado 5];
```

dado 1
dado 2
dado 3
dado 4
dado 5

podemos distribuir partes do trabalho:



Esse conceito é fundamental para compreender o Apache Spark.

⚠ Atenção

Processamento distribuído não significa simplesmente "ter vários computadores".

Esses computadores precisam trabalhar de maneira coordenada.

É necessário decidir, entre outras coisas:

- como dividir os dados;
- onde executar cada tarefa;
- como acompanhar a execução;
- como lidar com falhas;
- como reunir os resultados.

Gerenciar tudo isso manualmente seria extremamente complexo.

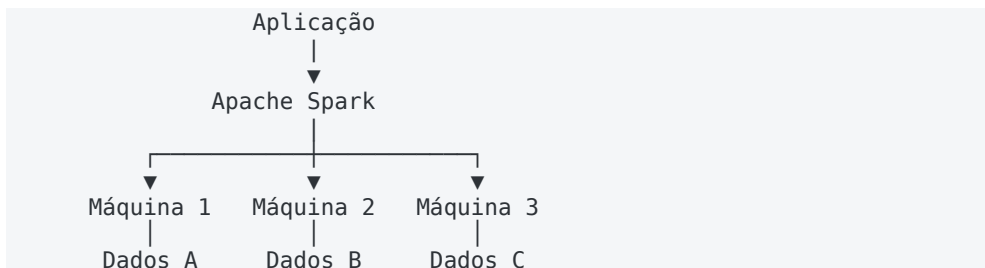
Frameworks como Apache Spark ajudam justamente a abstrair grande parte dessa complexidade.

1.7 Então, o que é Apache Spark?

Apache Spark é um mecanismo de processamento distribuído de dados projetado para executar operações sobre grandes conjuntos de dados de maneira eficiente e escalável.

Ele permite que um processamento seja distribuído entre recursos computacionais disponíveis em um cluster.

Em uma representação bastante simplificada:



O desenvolvedor descreve **o que deseja fazer com os dados**, enquanto o Spark cuida de diversos detalhes relacionados à execução distribuída.

Por exemplo, podemos querer:

- ler arquivos;
- selecionar registros;
- limpar dados;
- realizar cálculos;
- relacionar conjuntos de dados;
- agregar informações;
- gravar resultados.

Durante este livro aprenderemos como realizar essas operações utilizando **PySpark**.

1.8 Apache Spark não é um banco de dados

Esse é um ponto importante para quem está começando.

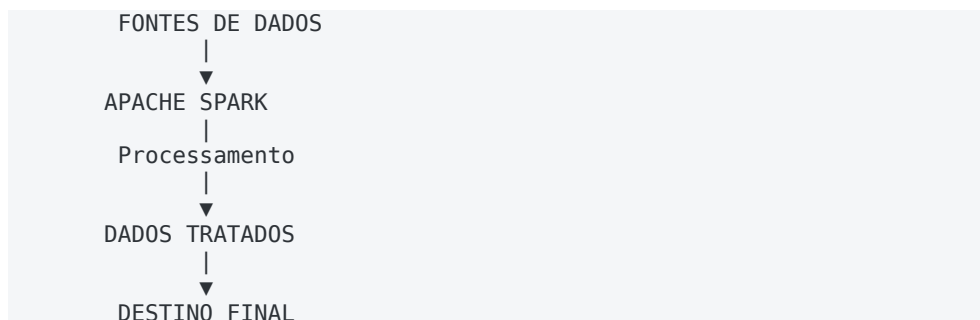
O Spark **não deve ser entendido simplesmente como um banco de dados.**

Seu papel principal é o processamento.

Os dados utilizados pelo Spark podem estar armazenados em diferentes tecnologias e formatos, como:

- CSV;
- JSON;
- Parquet;
- tabelas;
- Data Lakes;
- sistemas de armazenamento distribuído;
- serviços de armazenamento em nuvem.

Podemos pensar inicialmente da seguinte maneira:



Mais adiante veremos que arquiteturas modernas podem envolver tecnologias como **Delta Lake** e plataformas como **Databricks**.

Por enquanto, basta compreender que o Spark é principalmente responsável por **processar dados**.

1.9 O que é PySpark?

Agora chegamos ao protagonista deste livro.

Apache Spark possui APIs que permitem trabalhar com diferentes linguagens.

Entre elas está Python.

PySpark é a interface do Apache Spark para Python.

Isso permite utilizar uma sintaxe baseada em Python para construir aplicações de processamento de dados executadas pelo Spark.

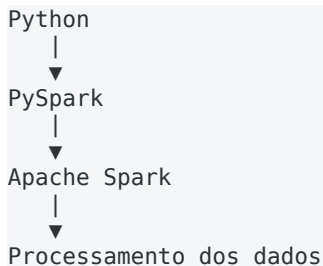
Um exemplo simples:

```
df = spark.read.csv(  
    "vendas.csv",  
    header=True  
)  
  
df.show()
```

Não se preocupe em entender cada detalhe agora.

Ao longo dos próximos capítulos aprenderemos cada elemento dessa instrução.

Por enquanto, observe apenas a ideia:



Conceito importante

Python e PySpark não são a mesma coisa.

Python é uma linguagem de programação.

PySpark é uma interface que permite utilizar Python para trabalhar com Apache Spark.

1.10 Python tradicional versus PySpark

Imagine que temos um arquivo contendo vendas.

Em Python poderíamos utilizar diferentes ferramentas para trabalhar com esses dados.

Em determinadas situações, todo o conjunto de dados cabe tranquilamente na memória de uma única máquina.

Nesse cenário, utilizar processamento distribuído talvez seja desnecessário.

Agora imagine que precisamos processar centenas de milhões ou bilhões de registros e que o ambiente foi projetado para processamento distribuído.

PySpark passa a fazer muito mais sentido.

Uma comparação conceitual:

CenárioAbordagem possível	
Pequeno conjunto de dados	Python/Pandas pode ser suficiente
Dados confortavelmente processáveis em uma máquina	Processamento local pode ser suficiente
Grandes volumes	Spark pode ser adequado
Processamento distribuído	Spark é uma forte opção
Pipelines de Engenharia de Dados em escala	PySpark é amplamente aplicável

Atenção

Isso não significa que "PySpark é sempre melhor que Pandas".

Ferramentas diferentes resolvem problemas diferentes.

Um bom Engenheiro de Dados não escolhe uma tecnologia porque ela parece mais poderosa.

Ele escolhe a tecnologia adequada ao problema.

1.11 Onde PySpark aparece na Engenharia de Dados?

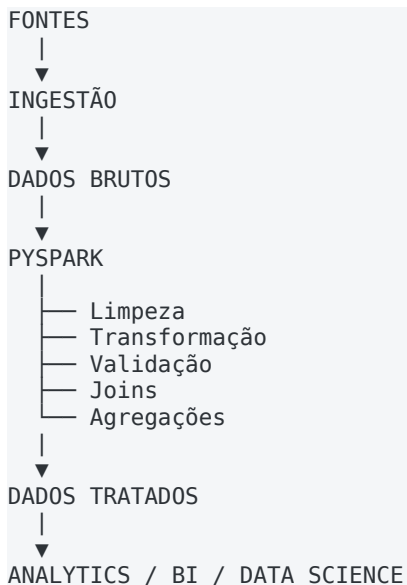
Imagine uma empresa que recebe diariamente dados provenientes de:

```
ERP
CRM
E-commerce
Aplicativo
Arquivos
APIs
Sistemas internos
```

Esses dados podem precisar ser:

1. coletados;
2. armazenados;
3. limpos;
4. padronizados;
5. relacionados;
6. enriquecidos;
7. agregados;
8. disponibilizados para consumo.

Podemos representar um fluxo simplificado:



É nesse tipo de cenário que PySpark se torna extremamente relevante para Engenharia de Dados.

1.12 Um exemplo empresarial

Imagine uma rede de lojas chamada **DataShop**.

Durante um dia, suas unidades geram registros contendo:

```
pedido_id  
cliente_id  
produto_id  
quantidade  
valor  
data_compra  
cidade  
estado
```

Ao final do dia, a empresa deseja descobrir:

- faturamento total;
- faturamento por estado;
- produtos mais vendidos;
- ticket médio;
- quantidade de pedidos;
- clientes que mais compraram.

Inicialmente, o volume pode ser pequeno.

Mas a empresa cresce.

Depois de alguns anos, temos:

```
2019 → milhões de registros  
2020 → milhões de registros  
2021 → dezenas de milhões  
2022 → dezenas de milhões  
2023 → centenas de milhões  
2024 → centenas de milhões  
2025 → ...
```

Agora precisamos construir pipelines capazes de trabalhar com esse crescimento.

Durante este livro, problemas semelhantes serão utilizados para aprendermos PySpark.

1.13 Nosso primeiro contato com código PySpark

Observe este exemplo:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("PrimeiroProjeto") \
    .getOrCreate()

dados = [
    ("Amazonas", 1500.00),
    ("São Paulo", 3200.00),
    ("Pará", 1800.00)
]

df = spark.createDataFrame(
    dados,
    ["estado", "faturamento"]
)

df.show()
```

O resultado seria semelhante a:

```
+-----+-----+
| estado|faturamento|
+-----+-----+
| Amazonas|      1500.0|
| São Paulo|     3200.0|
| Pará|      1800.0|
+-----+-----+
```

Não precisamos memorizar esse código.

Nosso objetivo neste momento é apenas observar alguns elementos que aparecerão frequentemente:

```
SparkSession
  |
  ▼
DataFrame
  |
  ▼
Transformações
  |
  ▼
Resultados
```

Aprenderemos tudo isso gradualmente.

1.14 Quando utilizar Spark?

Spark pode ser uma boa escolha quando existem necessidades como:

- processamento de grandes volumes de dados;
- transformação de dados em escala;
- construção de pipelines distribuídos;
- processamento batch;
- processamento de streams;
- integração com arquiteturas de Data Lake;
- workloads analíticos;
- Engenharia de Dados em ambientes distribuídos.

Entretanto, tamanho não é o único critério.

Também devemos considerar:

- arquitetura existente;
- infraestrutura disponível;
- custo;
- complexidade;
- frequência do processamento;
- conhecimento da equipe;
- requisitos de desempenho.

1.15 Quando Spark pode ser desnecessário?

Imagine um arquivo contendo apenas:

2.000 registros

Você precisa calcular o total de uma coluna uma única vez.

Criar toda uma infraestrutura distribuída para isso provavelmente seria exagerado.

Uma solução Python simples, SQL ou até mesmo uma planilha poderia resolver o problema dependendo do contexto.



Entenda assim

Não utilizamos um caminhão de mineração para transportar uma mochila.

Da mesma maneira, não precisamos de processamento distribuído para todo problema relacionado a dados.

Complexidade também possui custo.

Essa ideia será importante durante todo o livro.

1.16 Spark e Big Data

Apache Spark aparece frequentemente associado ao termo **Big Data**.

Mas Big Data não significa apenas "muitos dados".

O conceito tradicionalmente envolve características como:

Volume — grande quantidade de dados.

Velocidade — dados sendo produzidos e processados rapidamente.

Variedade — diferentes formatos e fontes.

Outras características também podem ser acrescentadas, como:

Veracidade — confiabilidade e qualidade.

Valor — capacidade de transformar dados em informação útil.

Spark tornou-se uma das tecnologias importantes desse ecossistema porque consegue trabalhar com processamento distribuído e diferentes tipos de workloads.

1.17 O papel do Engenheiro de Dados

Conhecer comandos PySpark é importante.

Mas isso não é suficiente.

Um Engenheiro de Dados precisa compreender perguntas como:

De onde os dados vieram?

Como estão armazenados?

Qual é o volume?

Com que frequência chegam?
Existem duplicidades?
Qual é o schema?
Como garantir qualidade?
Como processá-los eficientemente?
Onde o resultado será armazenado?
Quem utilizará esses dados?

É por isso que este livro não será apenas uma sequência de comandos.

Nosso objetivo será aprender a **resolver problemas de Engenharia de Dados utilizando PySpark**.

1.18 Conceitos importantes do capítulo

Antes de continuar, certifique-se de compreender estes conceitos:

Escalabilidade vertical

Aumentar os recursos de uma máquina.

Mais CPU
Mais RAM
Mais armazenamento

Escalabilidade horizontal

Adicionar máquinas para distribuir a carga.

Máquina 1
Máquina 2
Máquina 3
...

Cluster

Conjunto de computadores trabalhando de maneira coordenada.

Processamento distribuído

Divisão de um processamento entre diferentes recursos computacionais.

Apache Spark

Mecanismo de processamento de dados distribuído.

PySpark

Interface do Apache Spark para Python.

1.19 Erros conceituais comuns

Erro 1 — "Spark é Python"

Não.

Spark é um mecanismo de processamento de dados.

PySpark permite trabalhar com Spark utilizando Python.

Erro 2 — "Spark é um banco de dados"

Não exatamente.

Spark é principalmente um mecanismo de processamento.

Erro 3 — "Se tenho dados, preciso de Spark"

Não.

A tecnologia deve ser proporcional ao problema.

Erro 4 — "Quanto mais máquinas, mais rápido será"

Nem sempre.

Distribuição inadequada, movimentação excessiva de dados e configurações incorretas podem prejudicar o desempenho.

Entenderemos esses problemas nos capítulos de performance.

1.20 Na prática — Pensando como Engenheiro de Dados

Uma empresa possui:

```
clientes.csv  
produtos.csv  
pedidos.csv  
itens_pedido.csv
```

Os arquivos são gerados diariamente.

A diretoria deseja conhecer:

- faturamento diário;
- faturamento por estado;
- produto mais vendido;
- quantidade de clientes ativos;
- ticket médio.

Antes de escrever código, um Engenheiro de Dados deveria fazer perguntas.

Por exemplo:

Qual é o volume dos arquivos?

Quantos arquivos são recebidos diariamente?

O formato sempre será CSV?

O schema pode mudar?

Existem registros duplicados?

Existem valores nulos?

Precisamos manter histórico?

Quanto tempo temos para processar os dados?

Essa maneira de pensar será tão importante quanto aprender:

```
select()  
filter()  
groupBy()  
join()
```

1.21 Desafio do Engenheiro

Considere três situações.

Situação A

Uma pequena empresa possui um arquivo Excel com 3 mil vendas.

Ela precisa calcular o faturamento mensal.

Situação B

Uma plataforma possui 500 milhões de registros distribuídos em arquivos Parquet e precisa realizar transformações diariamente.

Situação C

Uma aplicação recebe continuamente grandes quantidades de eventos que precisam ser processados.

Pense:

Em quais situações Spark poderia fazer sentido?

Em qual delas provavelmente seria desnecessário?

Por quê?

Não procure apenas a resposta.

Tente justificar sua decisão como se estivesse apresentando uma solução técnica para sua equipe.

1.22 Resumo para revisão

Neste capítulo aprendemos que o crescimento do volume de dados criou a necessidade de novas estratégias de processamento.

Vimos duas formas importantes de escalabilidade:

VERTICAL

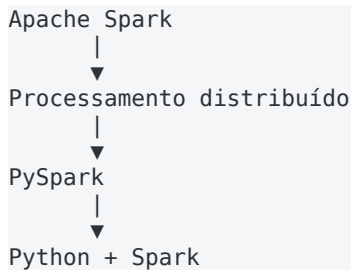
└─ aumentar uma máquina

HORIZONTAL

└─ adicionar máquinas

Conhecemos também o conceito de **processamento distribuído**, no qual diferentes recursos computacionais podem cooperar na execução de um trabalho.

A partir daí chegamos ao:



Mais importante do que decorar definições é compreender o problema que essas tecnologias procuram resolver.

1.23 Perguntas para fixação

Antes de seguir para o próximo capítulo, tente responder sem consultar o texto:

1. O que significa processamento distribuído?
2. Qual é a diferença entre escalabilidade vertical e horizontal?
3. O que é um cluster?
4. O que é Apache Spark?
5. O que é PySpark?
6. Python e PySpark são a mesma coisa?
7. Spark é um banco de dados?
8. Todo projeto de dados precisa utilizar Spark?
9. Cite duas situações em que Spark poderia ser utilizado.
10. Por que um Engenheiro de Dados precisa compreender o problema antes de escolher a tecnologia?

1.24 O que você deve saber antes de avançar

Você não precisa saber programar Spark ainda.

Ao terminar este capítulo, basta conseguir explicar com suas próprias palavras:

"Quando uma única máquina deixa de ser suficiente ou quando a arquitetura exige processamento distribuído, podemos dividir o trabalho entre diferentes recursos. Apache Spark é uma tecnologia criada para realizar esse tipo de processamento de dados em escala. PySpark permite trabalhar com Spark utilizando Python."

Se essa ideia estiver clara, alcançamos nosso objetivo.

Próximo capítulo

Capítulo 2 — Como o Apache Spark Funciona

Agora que sabemos **por que Spark existe**, chegou o momento de olhar por dentro.

No próximo capítulo conheceremos:

- Driver;
- Executors;
- Cluster Manager;
- Jobs;
- Stages;
- Tasks;
- Partitions;
- Transformations;
- Actions;
- Lazy Evaluation;
- DAG.

E voltaremos à nossa universidade com **um milhão de provas**.

Desta vez, não queremos apenas saber que vários professores podem trabalhar juntos.

Queremos descobrir:

Quem coordena os professores?

Quem decide quais provas cada professor recebe?

Como o trabalho é dividido?

O que acontece quando uma etapa depende de outra?

É exatamente essa lógica que nos ajudará a compreender a arquitetura do Apache Spark.

Continue a jornada.

A edição completa de PySpark na Prática conduz você dos fundamentos do Apache Spark até pipelines, Delta Lake, Databricks, processamento incremental, Structured Streaming, testes, observabilidade, produção e um projeto final Lakehouse.

- DataFrames, transformações, joins e Window Functions
- Arquitetura Medallion: Bronze, Silver e Gold
- Delta Lake, Databricks, Jobs e Unity Catalog
- Qualidade, testes, observabilidade e performance
- Projeto final: Lakehouse completo

ADQUIRA A EDIÇÃO COMPLETA NA PÁGINA DE VENDA