
# Aula 4 - Spark

Lucio Monteiro

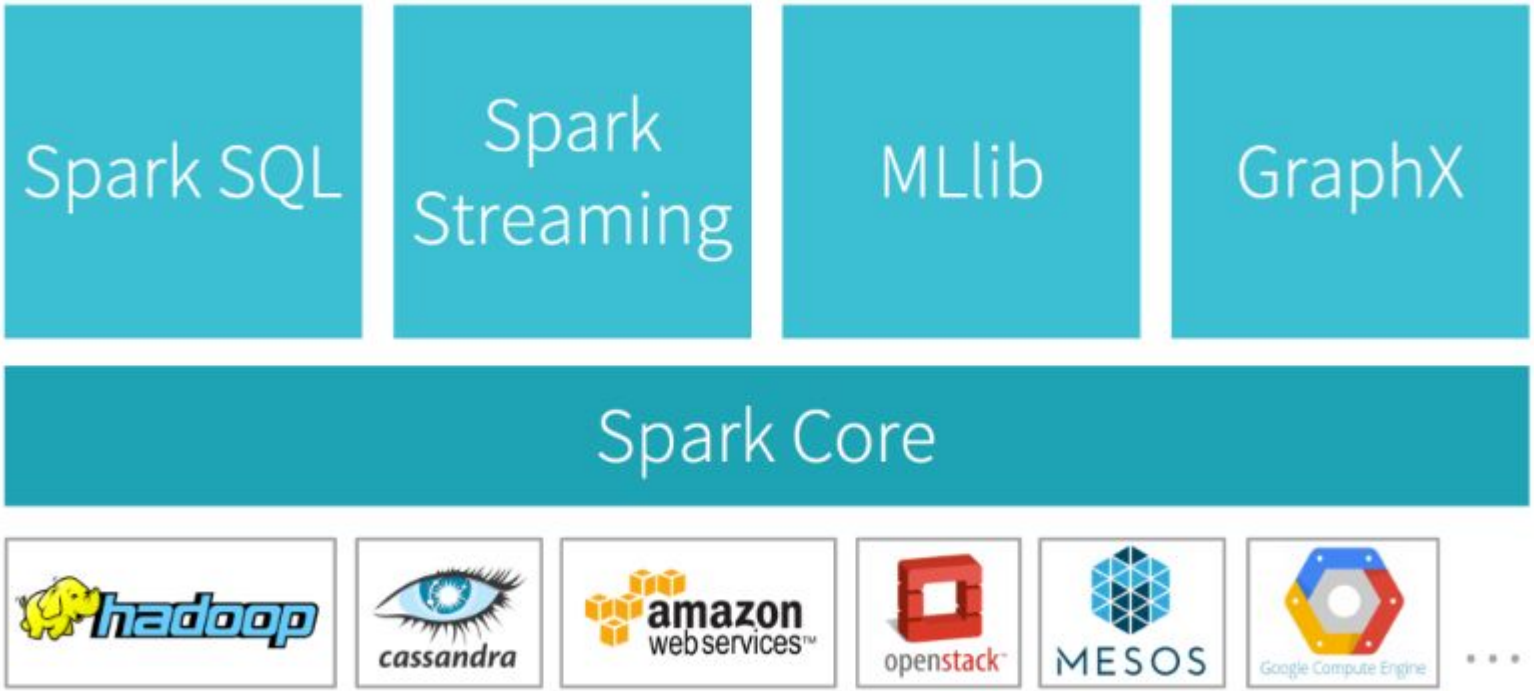
## O que é?

"O Spark é um framework para processamento de Big Data construído com foco em velocidade, facilidade de uso e análises sofisticadas. Oferece APIs de alto nível em Java, Scala e Python, bem como um conjunto de bibliotecas que o tornam capaz de trabalhar de forma integrada, em uma mesma aplicação, com SQL, streaming e análises complexas, para lidar com uma grande variedade de situações de processamento de dados."

## Características

- Plataforma de computação em Cluster rápida, tolerante a falhas e de propósito geral.
- Até 100x mais rápido que Mapreduce em memória.
- Até 10x mais rápido que Mapreduce em disco.
- Compatível com Hadoop.
- Open Source.
- Desenvolvido em Scala.
- Aplicações em Java, Scala, Python e R.
- Bibliotecas para SQL, Streaming, Machine Learning e Grafos.

## Componentes




Fonte: Databricks

## Spark Core

**Spark Core** é o mecanismo de execução geral subjacente para a plataforma *Spark* em que todas as outras funcionalidades são construídas. Ele fornece computação em memória e conjuntos de dados de referência em sistemas de armazenamento externo.

## Spark SQL

**Spark SQL** é o módulo do Apache Spark para trabalhar com dados estruturados. As interfaces oferecidas pelo Spark SQL fornecem ao Spark mais informações sobre a estrutura dos dados e do cálculo que está sendo executado.

## Spark Streaming

Este componente permite que o *Spark* processe dados de streaming em tempo real. Os dados podem ser ingeridos de muitas fontes, como *Apache Kafka*, *Flume* e *HDFS*. Em seguida, os dados podem ser processados usando algoritmos complexos e enviados para sistemas de arquivos, bancos de dados e painéis ativos.

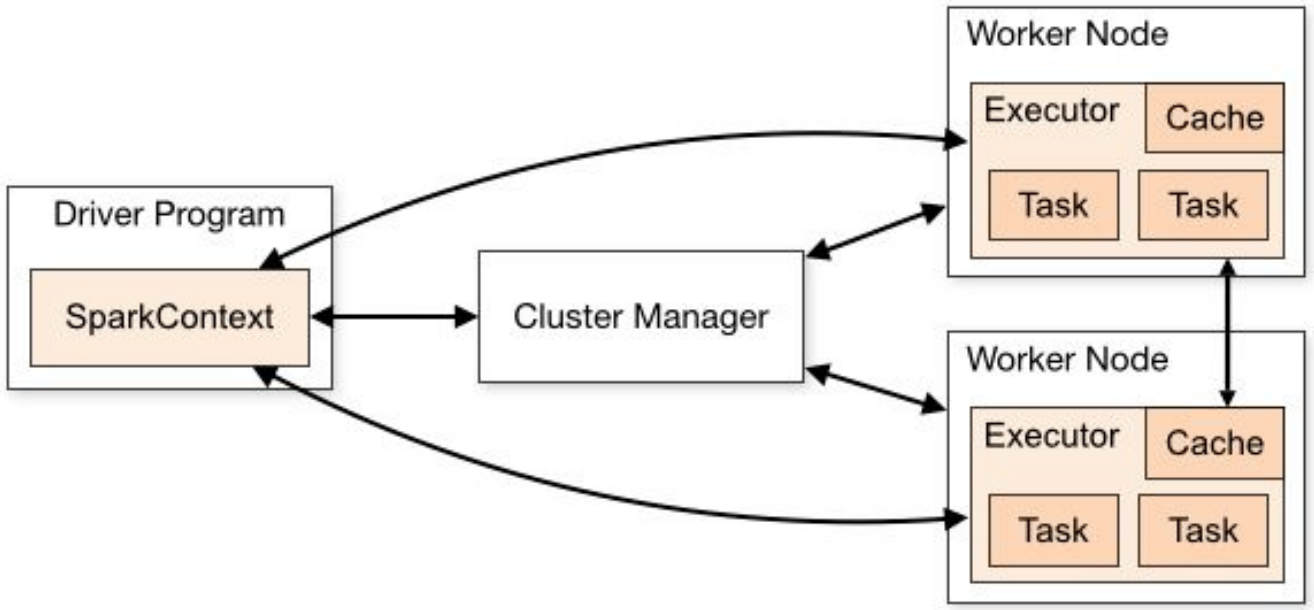
## MLib

O *Apache Spark* é equipado com uma rica biblioteca conhecida como **MLlib**. Essa biblioteca contém uma ampla gama de algoritmos de aprendizado de máquina (Machine Learning - ML) para *classificação*, *regressão*, *clustering* e *filtragem colaborativa*. Ele também inclui outras ferramentas para construir, avaliar e ajustar canais de *ML*. Todas essas funcionalidades ajudam o *Spark* a escalar horizontalmente em um cluster.

## GraphX

O *Spark* também possui uma biblioteca para manipular bancos de dados gráficos e realizar cálculos chamados **GraphX**. O *GraphX* unifica o processo *ETL* (Extract, Transform, and Load), análise exploratória e computação gráfica interativa em um único sistema.

# Arquitetura




Fonte: Apache - cluster overview

## Driver Program

É o ponto central e o ponto de entrada do *Spark Shell* (Scala, Python e R). O programa do driver executa a função main() do aplicativo e é o local onde o *Contexto Spark* é criado. O *Driver Spark* contém vários componentes - DAGScheduler, TaskScheduler, BackendScheduler e BlockManager, responsáveis pela tradução do código do usuário do spark em trabalhos de *Spark* reais executados no cluster.

## Características

- Principal programa da sua aplicação Spark.
- Servidor onde está sendo executado é chamado de nó Driver.
- Processo é chamado de processo Driver.
- Driver se comunica com o Cluster Manager para distribuir tarefas aos Executors.

## Cluster Manager

É o componente principal para gerenciamento do cluster *Spark*.

## Características

- Spark tem a capacidade de trabalhar com uma infinidade de gerentes de cluster, incluindo YARN, Mesos e um gerenciador de cluster autônomo.
- Um gerenciador de cluster autônomo consiste em dois daemons de longa duração, um nó mestre e um em cada um dos nós de trabalho.

## Executer

Executors são responsáveis por executar tarefas e manter os dados na memória ou armazenamento em disco.

## Características

- Executors só são iniciados quando uma execução de trabalho começa em um *Worker*.
- Cada aplicação possui seus próprios processos executors.

# Cluster

## Spark Cluster

**Spark Cluster** é um gerenciador de cluster simples incluído no *Spark* que facilita a configuração de um cluster.

- Para executar em um cluster, o *SparkContext* pode se conectar a vários tipos de gerenciadores de cluster (o gerenciador standalone do Spark, Mesos ou YARN), que aloca recursos em aplicativos.
- Uma vez conectado, o *Spark* adquire executores em nós no cluster, que são processos que executam cálculos e armazenam dados para sua aplicação.
- Em seguida, envia seu código de aplicativo para os executores.
- Finalmente, o *SparkContext* envia tarefas aos executores para executar.

## Yarn Cluster

## Spark Cluster

O **YARN** segue a arquitetura master e slave. O daemon master é chamado ResourceManager e o daemon slave é chamado *NodeManager*. Além dessa aplicação, o gerenciamento do ciclo de vida é feito por ApplicationMaster, que pode ser gerado em qualquer nó slave e permaneceria ativo durante a vida útil de uma aplicação. Quando o *Spark* é executado no *YARN*, ResourceManager desempenha a função do master do *Spark* e NodeManagers funciona como nós executores. Ao executar o *Spark* com *YARN*, cada executor do Spark é executado como um contêiner *YARN*.

## Características

- Possui Resource Manager (similar ao Master) para cada cluster e Node Manager (similar Slave) para cada nó no cluster.
- Aplicações no YARN são executadas em containers.
- Processo driver do *Spark* atua como Application Master.
- Node Manager monitora recursos usados por containers e reporta ao Resource Manager.

# RDD

**RDD**, ou conjunto de dados distribuídos resilientes, é uma coleção de registros de partição somente leitura. *RDD* é a estrutura de dados fundamental do *Spark*, que permite que um programador execute cálculos na memória em grandes grupos de maneira tolerante a falhas.

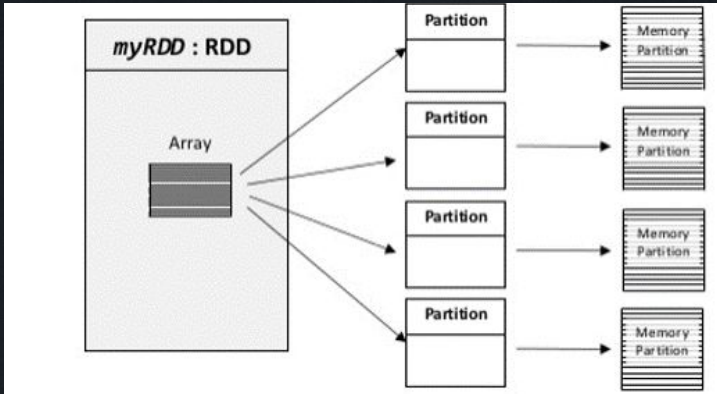
RDD foi a principal API voltada para o usuário no *Spark* desde o seu início. No núcleo, um *RDD* é uma coleção distribuída imutável de elementos de seus dados, particionada em nós no cluster que pode ser operada em paralelo com uma API de baixo nível que oferece transformações e ações.

Exemplos de transformações:

- map() - Aplica uma função a cada elemento no RDD e retorna um RDD do resultado.
- filter() - Retorna um RDD com os elementos que correspondem à condição de filtro.
- union() – Retorna um RDD contendo elementos de ambos os RDDs.

Exemplos de ações:

- collect() - Retornar todos os elementos do RDD.
- count() - Retorna o número de elementos do RDD.
- take(10) - Retorna 10 elementos do RDD.
- foreach(func) - Aplica a função fornecida a cada elemento do RDD.




Fonte: RDD

# Dataframe

Diferentemente de um RDD, em um Dataframe os dados são organizados em colunas nomeadas, como em uma tabela de um banco de dados relacional. Um DataFrame no Spark permite que desenvolvedores imponham uma estrutura em uma coleção distribuída de dados, permitindo abstração de nível superior.

# Características

- Um *DataFrame* é uma coleção distribuída de dados organizados em colunas nomeadas. É conceitualmente igual a uma tabela em um banco de dados relacional.
- Funciona apenas em dados estruturados e semiestruturados, organizando-os em colunas nomeadas.
- Os *DataFrames* permitem que o Spark gerencie o esquema.
- A API da fonte de dados permite o processamento de dados em diferentes formatos tais quais AVRO, CSV, JSON, tabelas Hive e MySQL. Além de ler e gravar em todos os formatos mencionados.
- Após a transformação no DataFrame, não é possível regenerar um objeto de domínio. Por exemplo, se você gerar testDF a partir de testRDD, não poderá recuperar o RDD original da classe de teste.

# PySpark

O *Apache Spark* é nativamente manipulado através da linguagem *Scala*, contudo, por conta da ampla relação da linguagem de programação Python com a área de dados, o **PySpark** surgiu como possibilidade para que o framework pudesse ser utilizado através dela também.

O *PySpark* pode ser utilizado via terminal, caso tenha sido instalado localmente, e pode ser manipulado em outras ferramentas e plataformas, como no *Databricks*.

## Operações básicas

Criando um *RDD* (Conjuntos de dados distribuídos resilientes), como o abaixo, podemos realizar operações tais quais:

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
```

Mostrar primeiro elemento:

```
numeros.first()
```

Mostrar os 5 maiores:

```
numeros.top(5)
```

Mostrar todos os elementos:

```
numeros.collect()
```

# PySpark

## Operações básicas

Contar os elementos:

```
numeros.count()
```

Media dos números:

```
numeros.mean()
```

Somar os elementos:

```
numeros.sum()
```

Mostrar maior elemento:

```
numeros.max()
```

Mostrar menor elemento:

```
numeros.min()
```

Calcular desvio padrão:

```
numeros.stdev()
```

# Pandas no Spark

Muitos usuários de *Python* estão acostumados com a biblioteca *Pandas* para manipulação de dados e *DataFrames*. Embora fosse possível utilizar o *Pandas* no *Apache Spark*, como ele não era compatível com computação distribuída, havia pouco sentido visto que as tarefas computacionais seriam executadas em apenas um nó, não explorando todo poder de processamento de um *Cluster Spark*.

Para atender esta demanda da comunidade de dados o *Koalas* foi criado, projeto *Open Source* do time criador do *Spark* e do *Databricks* que implementava *Pandas DataFrames* sobre o *Apache Spark*. Por conta da similaridade entre comandos e cláusulas, o *Koalas* foi amplamente utilizado pela comunidade de cientistas de dados que precisavam do alto poder de processamento de um *Cluster Spark*, mas sem deixar de lado suas expertises com *Pandas*, ainda que ambas as bibliotecas não fossem idênticas.

Em outubro de 2021, o pandas API passou a estar disponível com o lançamento do Apache Spark 3.2, podendo ser importado da seguinte forma, em uma célula de um notebook Databricks:

```
import pyspark.pandas as ps
```

Sendo assim, em versões mais recentes do *Apache Spark* é possível utilizar o *Pandas*, sem deixar de usufruir do poder de processamento de um *Cluster Spark*. Em versões mais antigas, o *Koalas* segue sendo uma ótima opção para profissionais de dados acostumados ao *Pandas*.

# Indicações e Bibliografia

PySpark Overview

Apache Spark

Apache Spark - definição Google Cloud

Apache Spark - definição Microsoft

Spark no Databricks

O que é RDD

Dataframes Python no Databricks

Pandas

Koalas: pandas API no Apache Spark

Pandas no Spark

Livro - Learning Spark: Lightning-Fast Data Analytics

Livro - Spark: The Definitive Guide: Big Data Processing Made Simple

Livro - Advanced Analytics with Spark: Patterns for Learning from Data at Scale

Livro - High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark

Obrig.ada