Spark

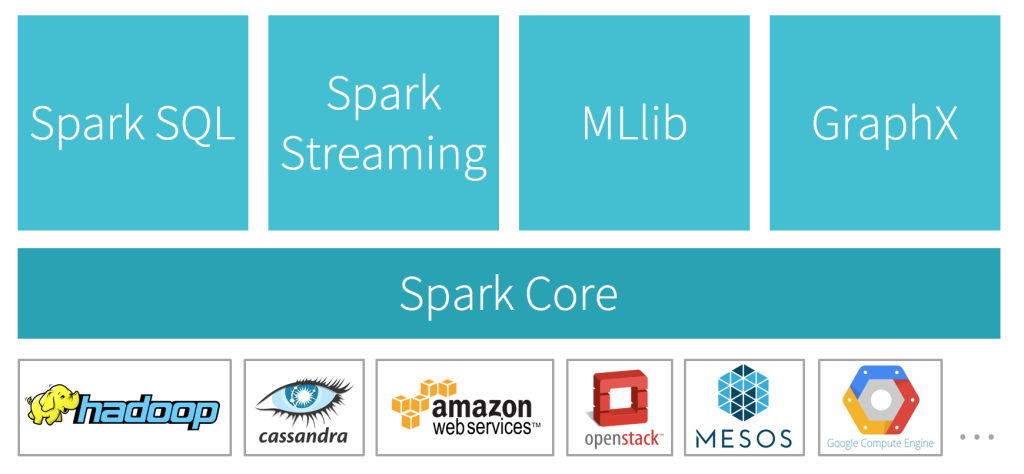
## O que é?

“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](https://databricks.com/blog/2016/02/02/an-illustrated-guide-to-advertising-analytics.html)

### 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.

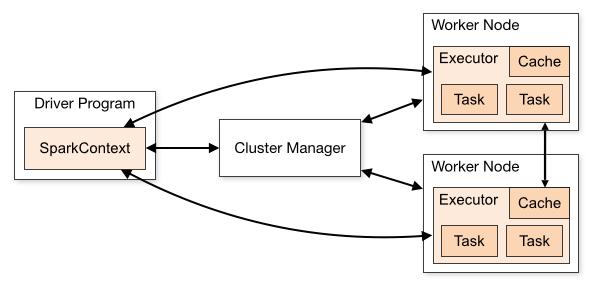
### MLlib

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](https://spark.apache.org/docs/latest/cluster-overview.html)

### 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.

### SparkContext

SparkContext é um ponto de entrada para o Spark, desde a versão 1.x usado para criar programaticamente RDDs do Spark, acumuladores e variáveis de transmissão no cluster. Seu objeto sc é o padrão disponível no spark-shell e pode ser criado usando a SparkContextclasse.

#### Características

* É o ponto de entrada da sessão Spark.
* Pode ser usado para criar RDDs, acumuladores e variáveis de transmissão no cluster.
* Em modo local (spark-shell ou PySpark) um objeto SparkContext é criado automaticamente e a variável sc refere-se ao objeto SparkContext.

### 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

O YARN segue a arquitetura master e slave. O daemon master é chamado ResourceManagere 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.

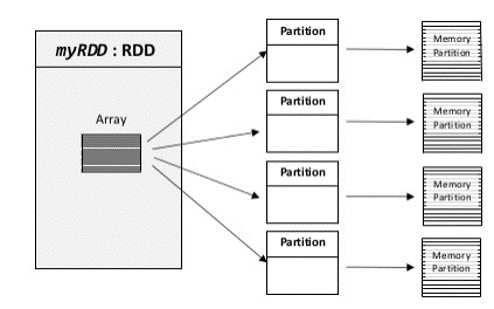
#### Características

* 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.

 Fonte: [RDD](https://image.slidesharecdn.com/youtubespk-141216130447-conversion-gate02/85/apache-spark-rdd-101-3-320.jpg?cb=1418735466)

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.

## 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.

[History of Spark APIs](https://s3-sa-east-1.amazonaws.com/lcpi/4bbb0e79-9571-4a87-9f18-05bac885c9b7.png) Fonte: [Databricks](https://databricks.com/glossary/what-is-rdd)

## Spark-submit

É usado para iniciar, parar e monitorar uma aplicação localmente ou distribuída em um cluster Spark.

Em um terminal, caso esteja trabalhando com Spark On-premise, é possível utilizar os comandos abaixo:

Executar pacotes Python e Java:

$spark-submit --option value application jar | python file [application\_arguments]

$spark-submit --help

--executor-cores: Número de núcleos de processador para alocar em cada executor. --executor-memory: O tamanho máximo de memória para alocar a cada executor. --num-executors: O número total de containers YARN a serem alocados. --name: Nome do aplicativo. --jars: Jars adicionais a aplicação como bibliotecas externas.

Local com 2 cores, tendo um arquivo HelloWorld.py como exemplo:

spark-submit --master local[2] --name myApp helloWorld.py

YARN Cluster:

spark-submit --master yarn --deploy-mode cluster --executor-memory 1G

--num-executors 2 --name myApp helloWorld.py

YARN Client:

spark-submit --master yarn-client --executor-memory 1G --num-executors 2 --name myApp helloWorld.py

Ao executar um programa Spark com Yarn, podemos acompanhar a execução pela interface web do Resource Manager no endereço localhost:8088

## 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()

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()

### On Premise - terminal

#### Analisar dados do Hive (DW)

Caso esteja trabalhando em ambiente on-premise, para que o Apache Spark seja capaz de consultar dados do Hive é preciso copiar um arquivo de configuração, que se instalados em ambiente Linux (mais comum para o Apache Hadoop), nos diretórios padrões, pode ser feito através dos comandos:

ls /usr/lib/hive/conf/hive-site.xml

cat /usr/lib/hive/conf/hive-site.xml

sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/

Agora em um terminal, digite o comando abaixo para iniciar o PySpark:

pyspark

Criar contexto HiveContext

from pyspark.sql import HiveContext

contexto = HiveContext(sc)

Conectar no banco de dados na tabela:

banco = contexto.table("nome\_bd.nome\_tabela")

banco.show()

É importante registrar a tabela no Spark para que seja possível executar queries SQL no Dataframe:

banco.registerTempTable("nome\_tabela")

contexto.sql("select \* from nome\_tabela").show()

contexto.sql("select sum(nome\_coluna) from nome\_tabela").show()

#### Criar um Dataframe

Conectar no banco de dados na tabela:

banco = contexto.table("nome\_bd.nome\_tabela")

banco.registerTempTable("nome\_tabela")

A variável df será o Dataframe:

df = contexto.sql("select \* from nome\_tabela")

df.show()

df.show(100)

df.printSchema()

Exemplos de comandos sobre o Dataframe:

df.select('nome\_coluna', 'nome\_coluna2').show()

df.select('nome\_coluna', 'nome\_coluna2').distinct().show(30)

df.select('nome\_coluna', 'nome\_coluna2').show()

df.filter(df.nome\_coluna=='abc').show()

df.filter(df.nome\_coluna=='abc').count()

### Databricks

#### Dataframes a partir de arquivos

É possível acessar arquivos e carregá-los em Dataframes usando PySpark, como no exemplo abaixo:

df = spark.read.csv('<caminho do arquivo csv>')

Além de arquivos do tipo csv, também é possível ler em outros formatos, como parquet e json, apenas adaptando o comando.

Durante a leitura, também é possível indicar se o arquivo possui headers e que automáticamente inferir os tipos de dados, com os complementos ao comando já visto:

df = spark.read.csv('<caminho do arquivo csv>', header=True,inferSchema=True)

Saiba que o processo pode não resultar nos tipos de dados corretos, valhe checar e eventualmente realizar transformações:

df.printSchema()

#### Dataframes a partir de tabelas

Tendo uma tabela criada no Databricks como origem, independente do seu tipo de dados, podemos criar um DF a partir dela com o comando:

df = spark.table('nome\_tabela')

#### Análises com PySpark

Mostraremos exemplos de comandos para servirem como guia. Não deixe de consultar as indicações e bibliografia ao final deste material caso precise de algum aqui não listado.

Select:

df.select('nome\_coluna').show()

Filter ou Where (aqui é possível usar a cláusula Where caso tenha mais familiaridade com SQL, ambos funcionam da mesma forma)

df.select('nome\_coluna').where('nome\_coluna == 1').show()

df.select('nome\_coluna').filter(('nome\_coluna == 1') | ('nome\_coluna == 2')).show()

##### Agregações

Min

df.select('nome\_coluna').groupby().min().show()

Max

df.select('nome\_coluna').groupby().max().show()

Mean

df.select('nome\_coluna').groupby().mean().show()

Cláusula AGG e desvio padrão (também pode ser usado com média etc.)

df.agg({'nome\_coluna': 'stddev'}).show()

##### Joins

Left Join:

df\_join = df\_1.join(nome\_coluna, on=df\_1.col1==df\_2.col2, how='left')

Right Join:

df\_join = df\_1.join(nome\_coluna, on=df\_1.col1==df\_2.col2, how='right')

Inner Join:

df\_join = df\_1.join(nome\_coluna, on=df\_1.col1==df\_2.col2, how='inner')

### Pandas no Apache 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

[Apache Spark](https://spark.apache.org/)

[Apache Spark - definição Google Cloud](https://cloud.google.com/learn/what-is-apache-spark?hl=pt-br)

[Apache Spark - definição Microsoft](https://docs.microsoft.com/pt-br/azure/hdinsight/spark/apache-spark-overview)

[Spark no Databricks](https://databricks.com/spark/getting-started-with-apache-spark)

[O que é RDD](https://databricks.com/glossary/what-is-rdd)

[Dataframes Python no Databricks](https://docs.microsoft.com/pt-br/azure/databricks/spark/latest/dataframes-datasets/introduction-to-dataframes-python)

[Pandas](https://pandas.pydata.org/)

[Koalas: pandas API no Apache Spark](https://koalas.readthedocs.io/en/latest/)

[Pandas no Spark](https://databricks.com/blog/2021/10/04/pandas-api-on-upcoming-apache-spark-3-2.html)

Livro - [Learning Spark: Lightning-Fast Data Analytics](https://www.oreilly.com/library/view/learning-spark-2nd/9781492050032/)

Livro - [Spark: The Definitive Guide: Big Data Processing Made Simple](https://www.oreilly.com/library/view/spark-the-definitive/9781491912201/)

Livro - [Advanced Analytics with Spark: Patterns for Learning from Data at Scale](https://www.oreilly.com/library/view/advanced-analytics-with/9781491912751/)

Livro - [High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark](https://www.oreilly.com/library/view/high-performance-spark/9781491943199/)