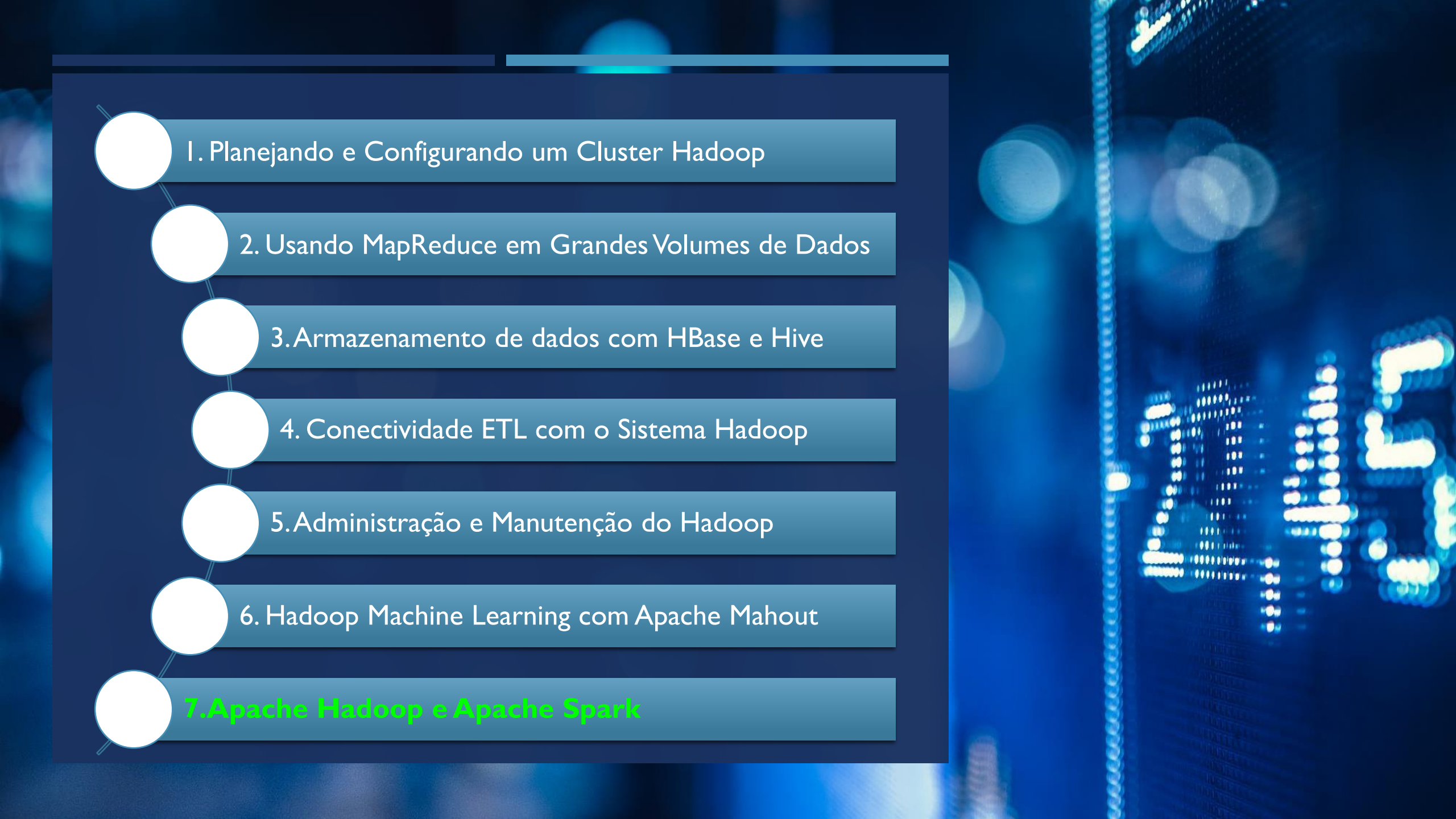
ENG DE DADOS COM HADOOP E SPARK 7


1. Planejando e Configurando um Cluster Hadoop
2. Usando MapReduce em Grandes Volumes de Dados
3. Armazenamento de dados com HBase e Hive
4. Conectividade ETL com o Sistema Hadoop
5. Administração e Manutenção do Hadoop
6. Hadoop Machine Learning com Apache Mahout
7. Apache Hadoop e Apache Spark

7.1 APACHE HADOOP E APACHE SPARK

Framework para desenvolvimento de aplicações distribuídas

Framework: conjunto de aplicações que estão interconectadas, onde podemos usar módulos dessa aplicação. (App dividida em módulos!)

Framework

Principais módulos do Hadoop:

- HDFS (Armazenamento distribuído)
- MapReduce (Processamento distribuído)
- Yarn (Gestão dos Jobs)

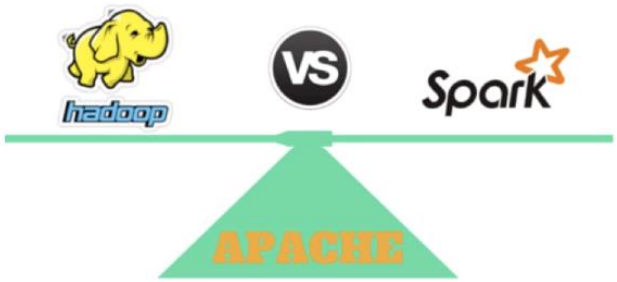
Framework para processamento de Big Data

Framework

Principais módulos do Spark:

- Spark SQL
- Spark Streaming
- Mllib (Machine Learning)
- GraphX (graph)

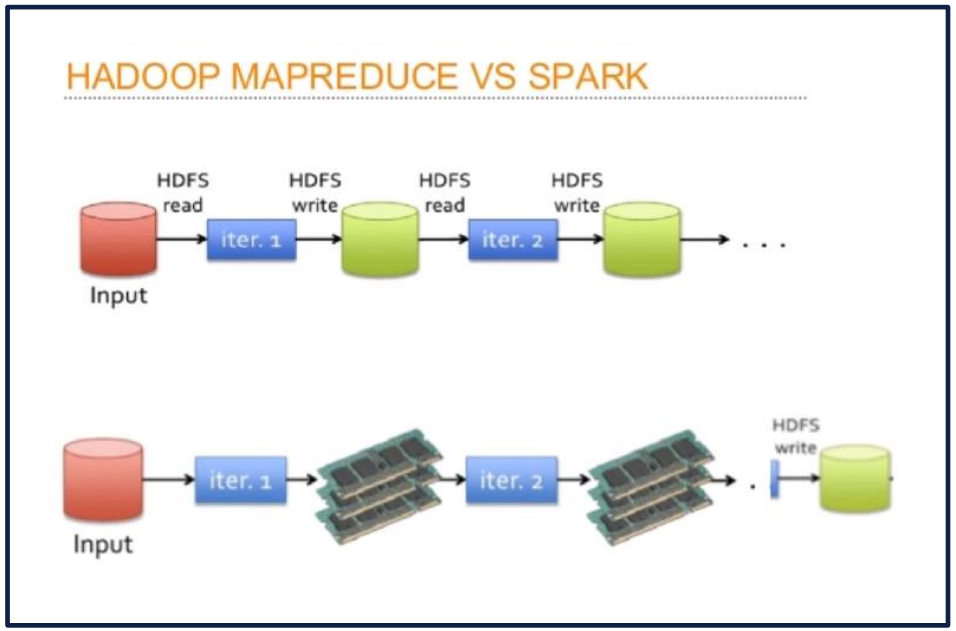
APACHE HADOOP E APACHE SPARK



- Custo
- Velocidade
- Versatilidade
- Competências
- Fornecedores

Hadoop e Spark são tecnologias complementares!

Porém, é possível usar um sem o outro!



APACHE HADOOP E APACHE SPARK

Com o Spark podemos realizar:

- Operações de ETL
- Análise Preditiva e Machine Learning
- Operações de Acesso a Dados com SQL
- Text Mining
- Processamento de Eventos em Tempo Real
- Aplicações Gráficas
- Reconhecimento de Padrões
- Sistemas de Recomendação

Hadoop e Spark fazem coisas diferentes

Você pode usar um sem o outro

O Spark é mais rápido

Mas você pode não precisar da velocidade do Spark

Mecanismos diferentes de recuperação a falhas

O Spark é normalmente utilizado com o HDFS, mas outros sistemas de arquivos ou sistemas de armazenamento podem ser usados, tais como:

- Sistema de arquivos local ou de rede (NFS)
- Amazon S3
- RDBMS
- NoSQL (Apache Cassandra, Hbase)
- Sistemas de Mensagens (Kafka)

Embora seja escrito em Scala, o Spark suporta:

Scala Java python R SQL

7.2 COMO O SPARK FUNCIONA SOBRE O HDFS?

Para ***ler*** arquivos do HDFS com Spark usamos:

```
textfile = sc.textFile("hdfs://mycluster/data/file.txt")
```

Leitura

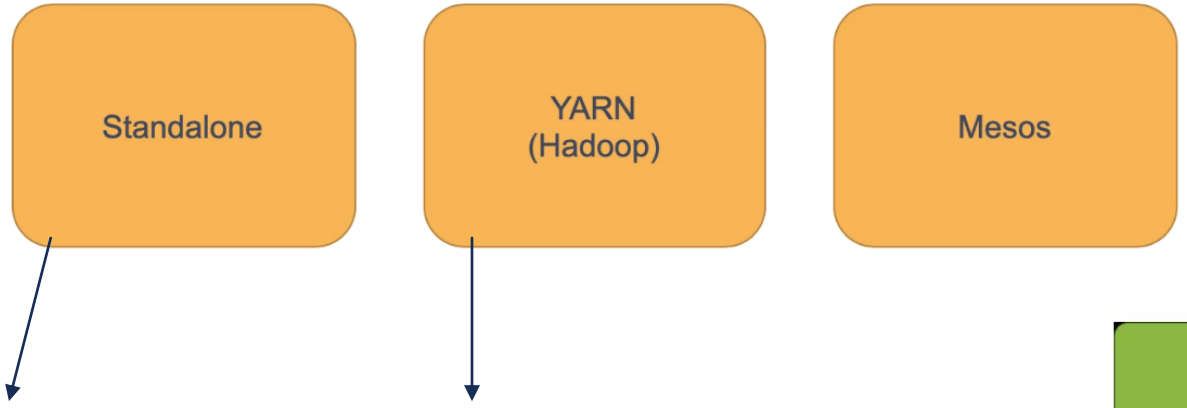
Para ***gravar*** arquivos no HDFS com Spark usamos:

```
myRDD.saveAsTextFile("hdfs://mycluster/data/output.txt")
```

Gravação

7.3 MODOS DE INSTALAÇÃO DO SPARK

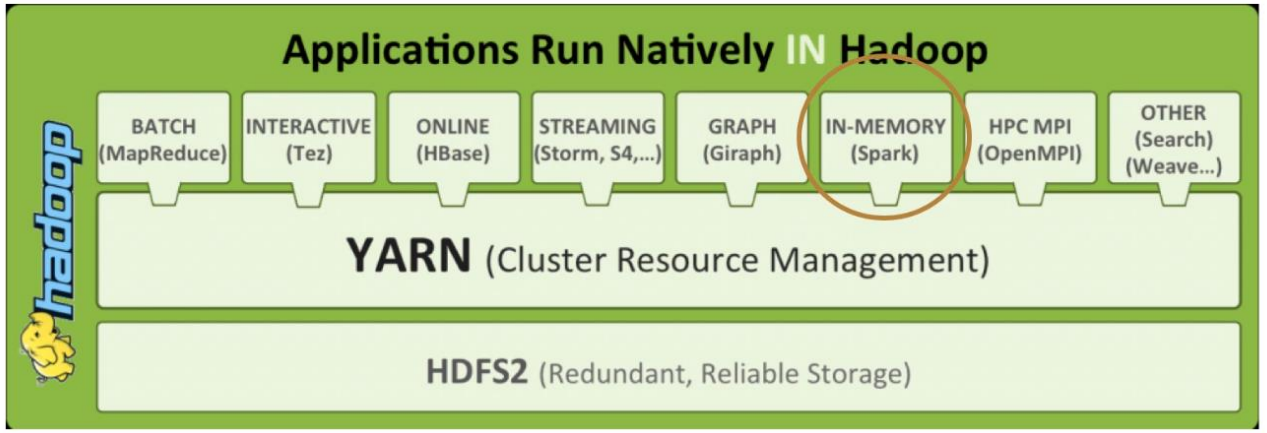
Instalação do Spark



Sem agendador de tarefas

Usar o Yarn como agendador de tarefas para o Apache Spark

YARN e Spark



7.4 ANATOMIA DE UMA APLICAÇÃO SPARK

Módulos adicionais

Cassandra Spark Connector

Spark R

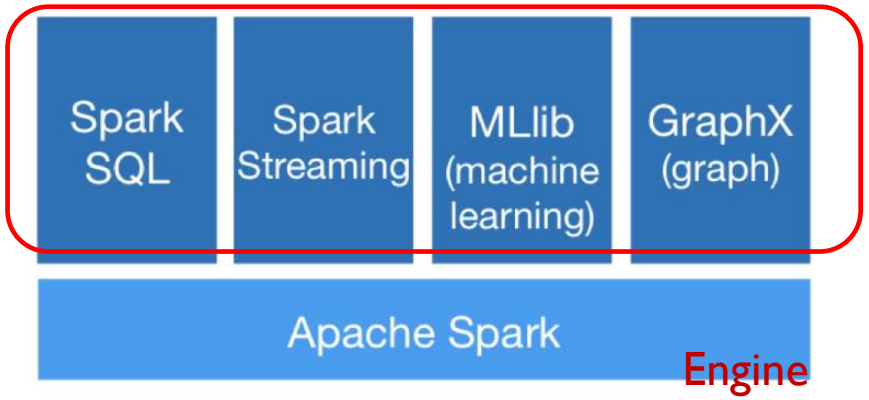
Spark
Lightning-Fast Cluster Computing

Se eu posso construir meu processo de análise com Python ou R, por exemplo, por que usaria o Spark?

1- Porque você precisa processar um grande volume de dados.

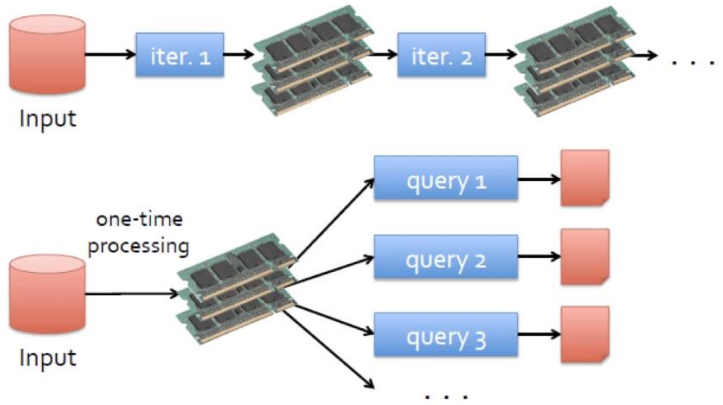
2- Porque você quer usar uma das APIs prontas do Spark, como SQL ou Streaming, por exemplo!

APIs → módulos que compõem o framework Spark



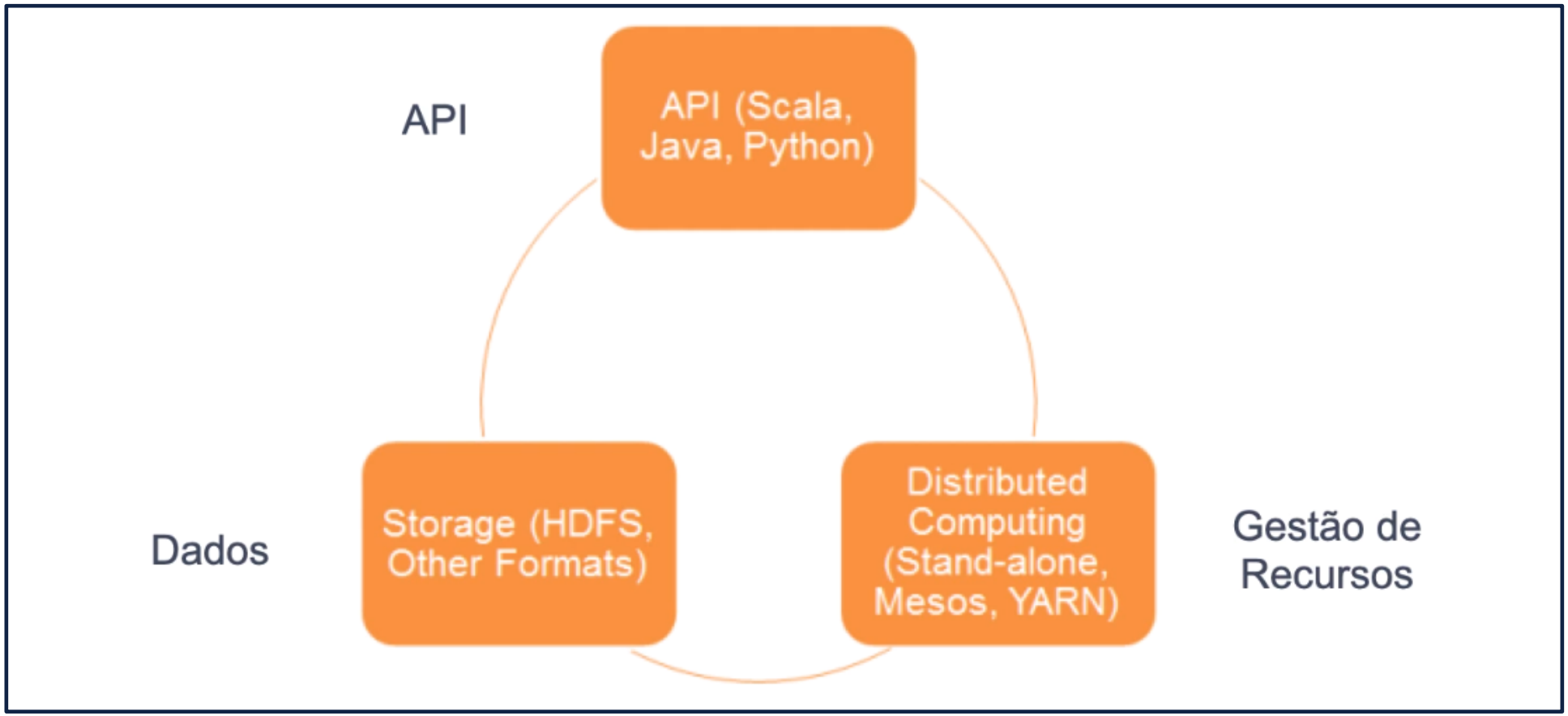
Iterações:

- Os dados passam várias vezes pelo Algoritmo.

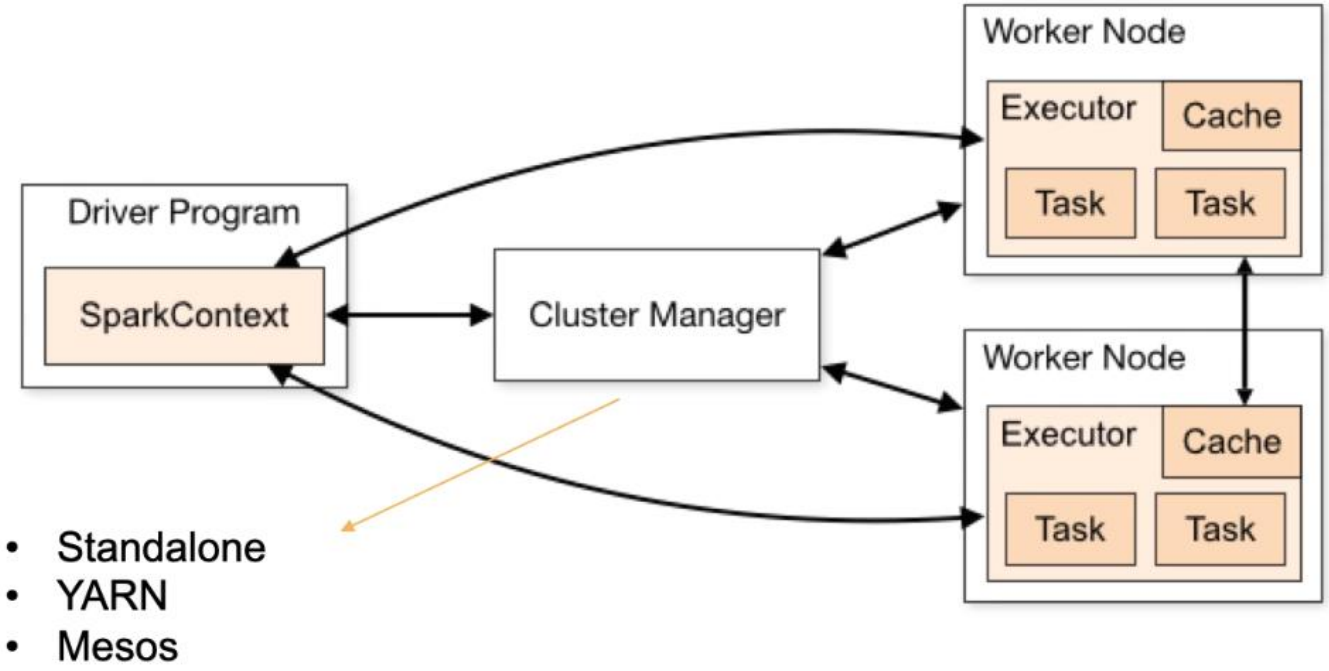


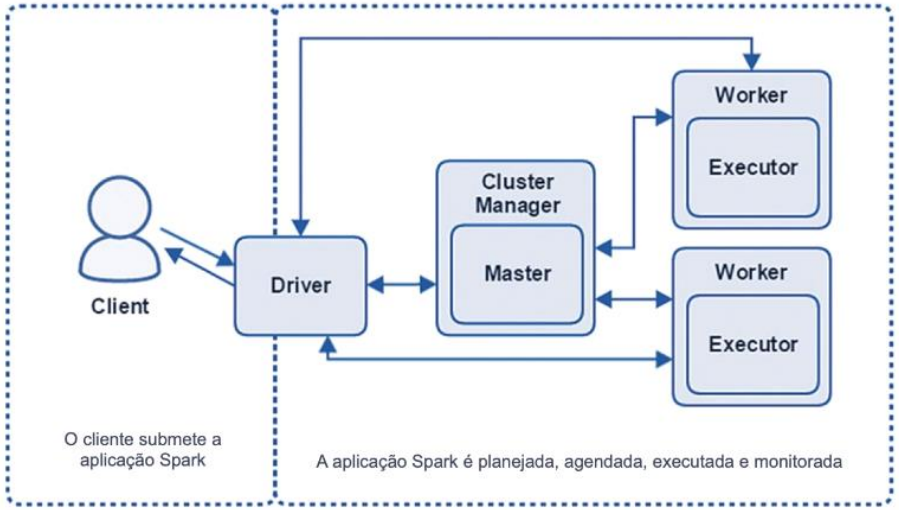
7.5 ARQUITETURA DA APLICAÇÃO

Arquitetura do Spark



7.6 PROCESSAMENTO DE UMA APLICAÇÃO SPARK





O cliente submete a aplicação Spark

A aplicação Spark é planejada, agendada, executada e monitorada

7.7 OUTRAS CARACTERÍSTICAS DO SPARK

Outras Características do Spark:

- Suporta mais do que apenas as funções de Map e Reduce
- Otimiza o uso de operadores de grafos arbitrários
- Avaliação sob demanda de consultas de Big Data contribui com otimização do fluxo global do processamento de dados
- Fornece APIs concisas e consistentes em Scala, Java e Python
- Oferece shell interativo para Scala, Python e R. O shell ainda não está disponível em Java

7.8 RDD'S E DATAFRAMES

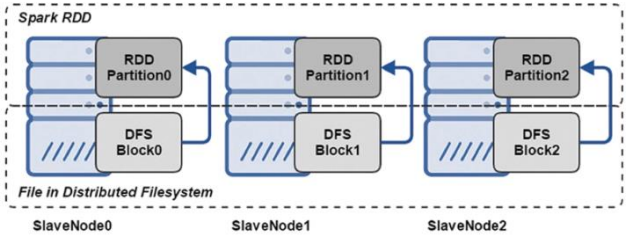
RDDs

Resilient Distributed Datasets

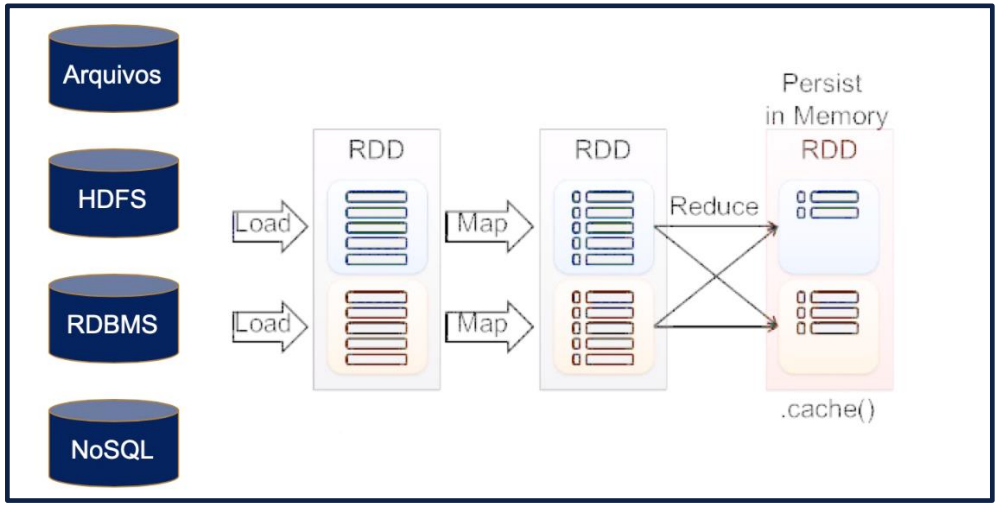
(Dados não estruturados)

RDD é Conceito Central do Framework Spark!

RDD's são imutáveis!



RDD é uma coleção de objetos distribuída e imutável. Cada conjunto de dados no RDD é dividido em partições lógicas, que podem ser computadas em diferentes nodes do cluster.



No Hadoop MapReduce (que não possui o conceito de RDD), cada resultado intermediário é gravado em disco. Ou seja, imagine um algoritmo de ML que precisa realizar diversas iterações nos dados. O disco será usado com muito frequência, deixando o processo mais lento.

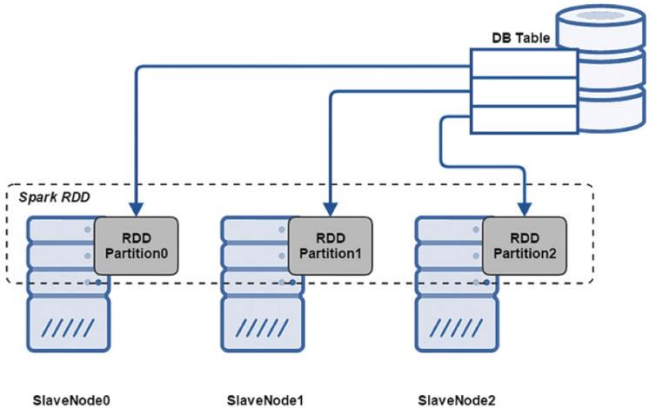
Com o conceito de RDD, o Spark armazena os resultados intermediários em memória, permitindo que operações iterativas que precisam acessar os dados diversas vezes, possam recorrer a memória do computador e não ao disco. Os dados serão gravados em disco apenas ao fim do processo ou se durante o processo, não houver memória disponível. Lembre que estamos falando aqui de cluster de computadores, com Terabytes de memória RAM quando se combina a memória de cada node do cluster.

Existem 2 formas de criar um RDD:

Paralelizando uma coleção existente
(função sc.parallelize)

Referenciando um dataset externo
(HDFS, RDBMS, NoSQL, S3)

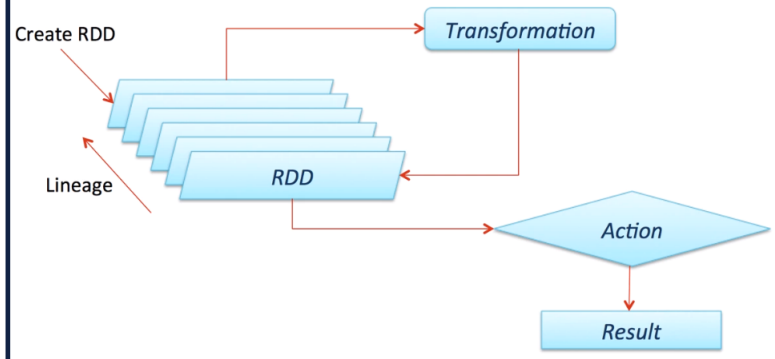
RDD'S E DATAFRAMES



Os RDD's podem ser particionados e persistidos em memória ou disco!

cache() x persist()

Transformações e Ações

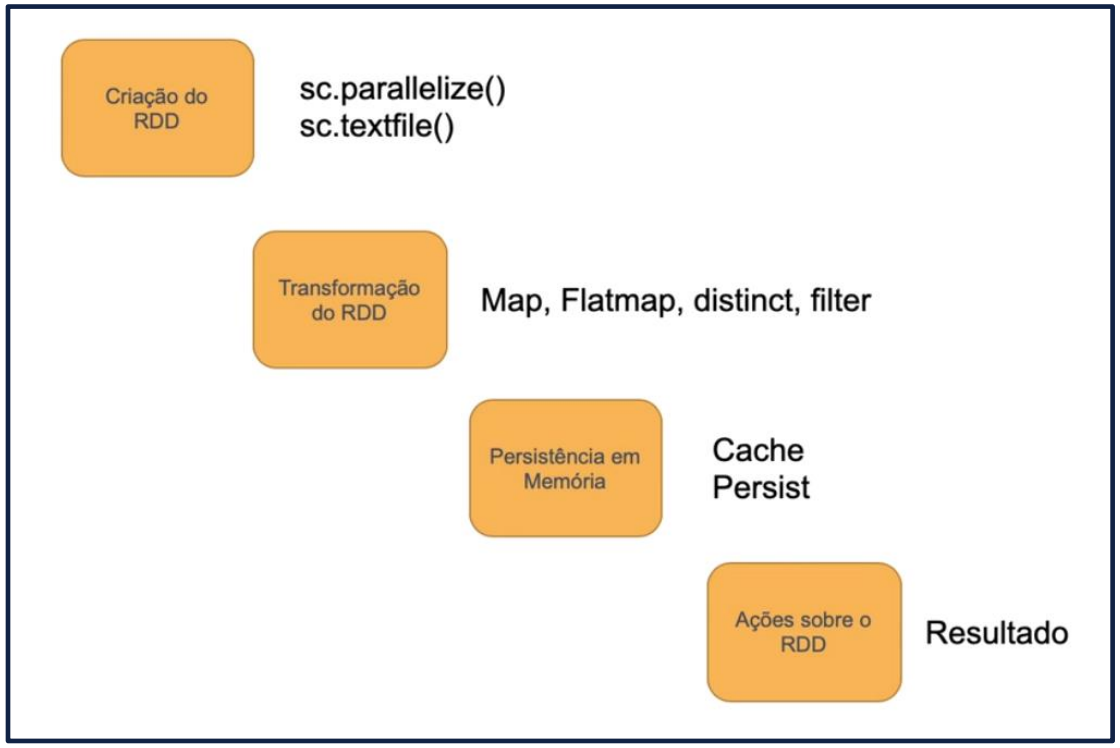


Lazy Evaluation

O RDD suporta dois tipos de operações:

Transformações	Ações
map()	reduce()
filter()	collect()
flatMap()	first()
reduceByKey()	take()
aggregateByKey	countByKey()

RDD'S E DATAFRAMES



Estruturas de dados **estruturados** no Spark:

Dataframe: linguagens Python e R

Dataset: linguagens Scala e Java

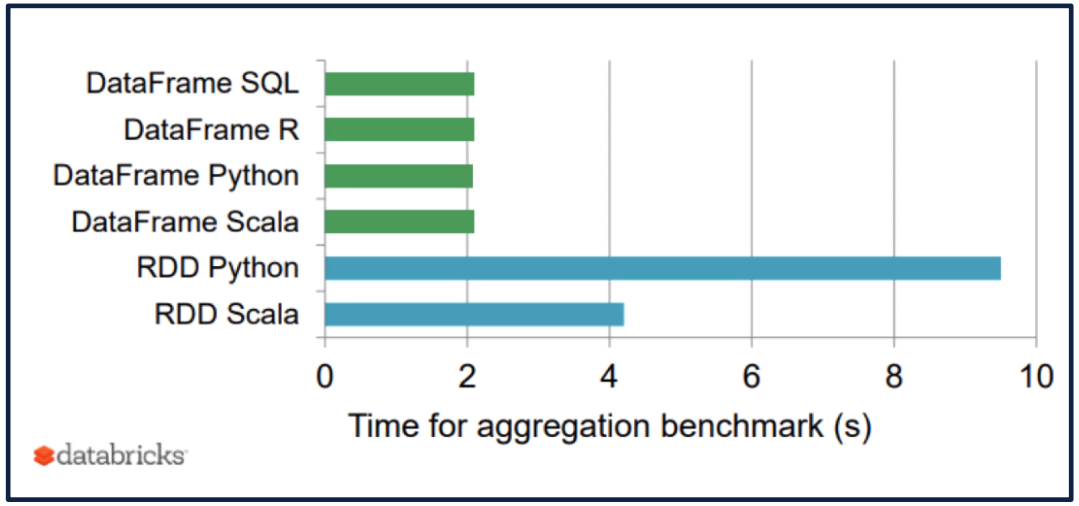
Quando usamos RDDs?

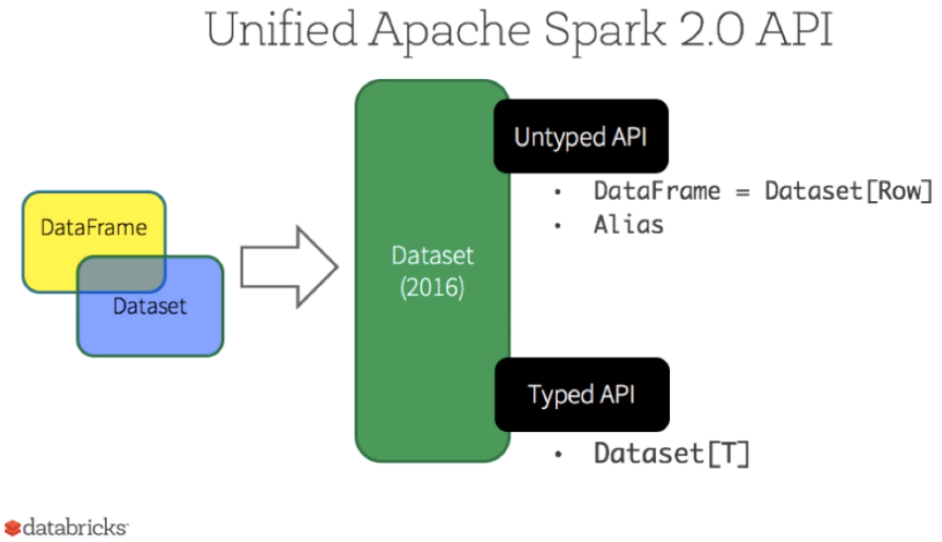
- Você deseja transformações e ações de baixo nível e controle no seu conjunto de dados. Seus dados não são estruturados, como fluxos de mídia ou de texto;
- Você deseja manipular seus dados com construções de programação funcional;
- Você não se preocupa em impor um esquema, como formato colunar, ao processar ou acessar atributos de dados por nome ou coluna;
- Você pode renunciar a alguns benefícios de otimização e desempenho disponíveis com Dataframes e Datasets para dados estruturados e semiestruturados.

RDD'S E DATAFRAMES

Dataframes

(Dados estruturados ou semi-estruturados)





Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

RDD'S E DATAFRAMES

- Como um RDD, um Dataframe é uma coleção distribuída imutável de dados.
- Ao contrário de um RDD, os dados são organizados em colunas nomeadas, como uma tabela em um banco de dados relacional.
- Projetado para facilitar ainda mais o processamento de grandes conjuntos de dados, o Dataframe permite que os desenvolvedores imponham uma estrutura em uma coleção distribuída de dados, permitindo abstração de nível superior.

Quando usamos Dataframes?

- Se você deseja semântica rica, abstrações de alto nível e APIs específicas do domínio, use Dataframe ou Dataset.
- Se o seu processamento exigir expressões de alto nível, filtros, mapas, agregação, médias, soma, consultas SQL, acesso colunar e uso de funções lambda em dados semiestruturados, use Dataframe ou Dataset.
- Se você deseja unificação e simplificação de APIs nas bibliotecas Spark, use Dataframe ou Dataset.
- Se você é um usuário R, use Dataframes.
- Se você é um usuário Python, use Dataframes e recorra aos RDDs se precisar de mais controle.

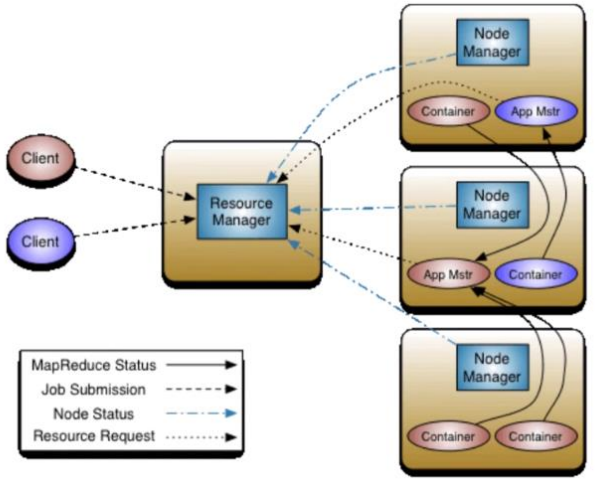
Em resumo, a escolha de quando usar RDD ou Dataframe e/ou Dataset parece óbvia. Enquanto o primeiro oferece funcionalidade e controle de baixo nível, o último permite visualização e estrutura personalizadas, oferece operações específicas de alto nível e domínio, economiza espaço e é executado em velocidades superiores.

7.9 ARQUITETURA DO APACHE YARN

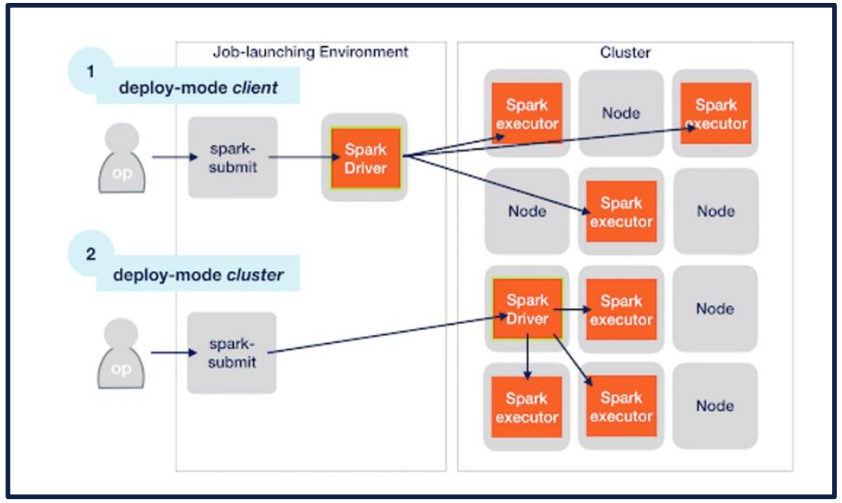
Modules

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.
- Hadoop Ozone: An object store for Hadoop.



Deploy do Spark em um Cluster Hadoop com Yarn



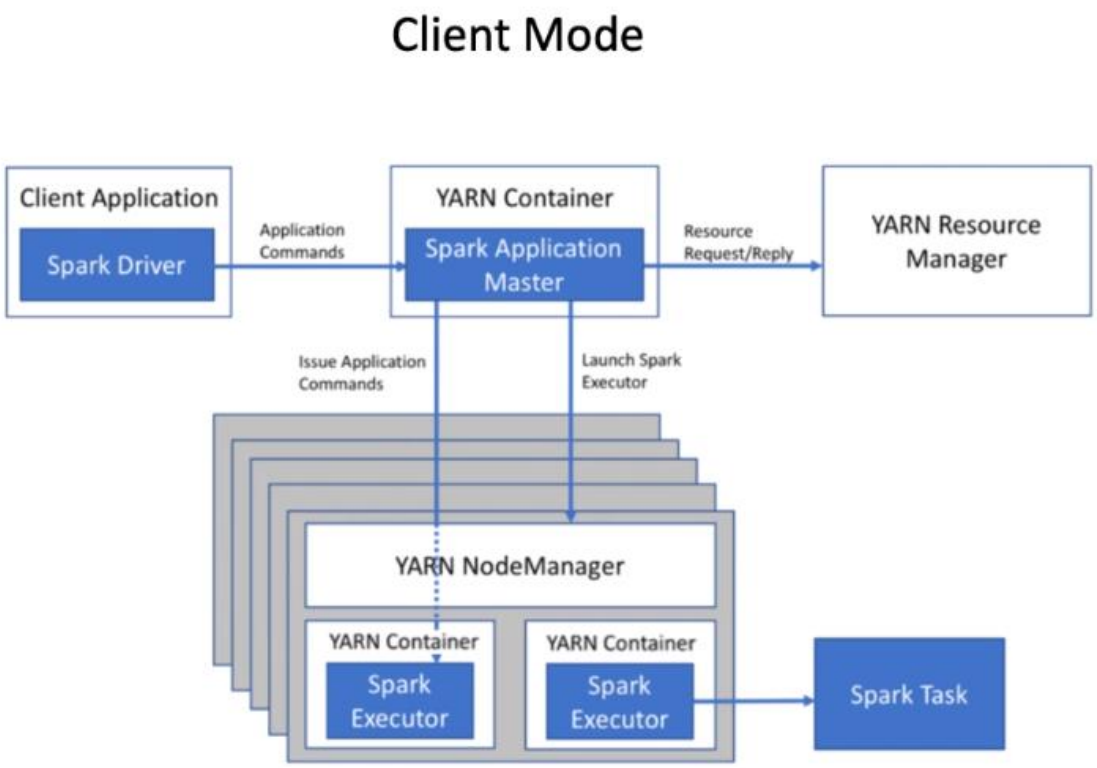
Parâmetros

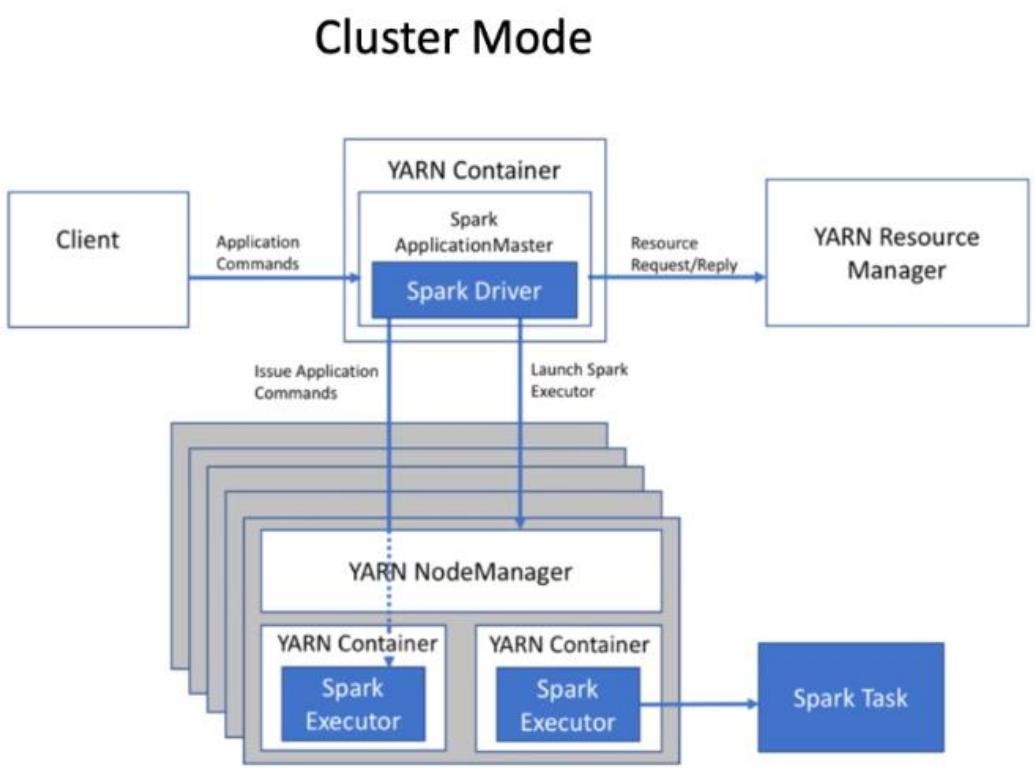
hadoop.apache.org/docs/r3.2.1/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

yarn.nodemanager.aux-services 1/2

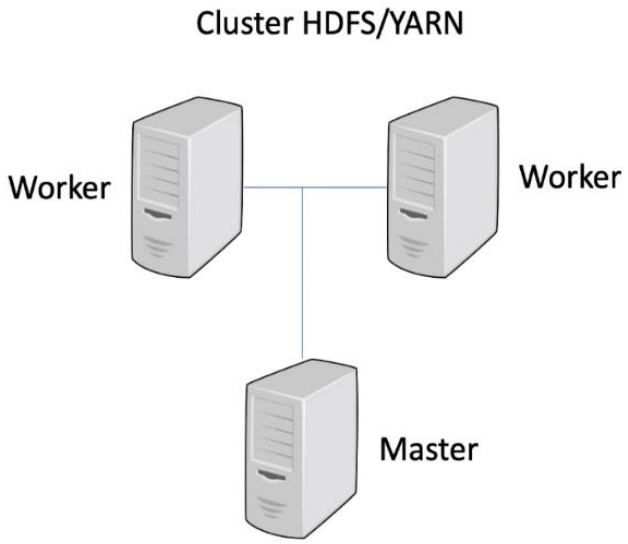
yarn.nodemanager.runtime.linux.sandbox-mode.local-dirs.permissions	read
yarn.nodemanager.runtime.linux.sandbox-mode.policy	
yarn.nodemanager.runtime.linux.sandbox-mode.whitelist-group	
yarn.nodemanager.windows-container.memory-limit.enabled	false
yarn.nodemanager.windows-container.cpu-limit.enabled	false
yarn.nodemanager.linux-container-executor.cgroups.delete-timeout-ms	1000
yarn.nodemanager.log-aggregation.compression-type	none
yarn.nodemanager.principal	
yarn.nodemanager.aux-services	

ARQUITETURA DO APACHE YARN





ARQUITETURA DO APACHE YARN




```
# Hadoop
export HADOOP_HOME=/opt/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

Agora quando o Spark for executado em modo de deploy cliente ele irá ler as variáveis de ambiente!

Spark Local
(Mesma Máquina
do Cluster Hadoop)

Spark
Remoto

THANKS