escola britânica de artes criativas & tecnologia

Profissão: Analista de dados

escola britânica de artes criativas & tecnologia

BIG DATA I - PROCESSAMENTO

GUIA DA AULA 2

escola britânica de artes criativas & tecnologia

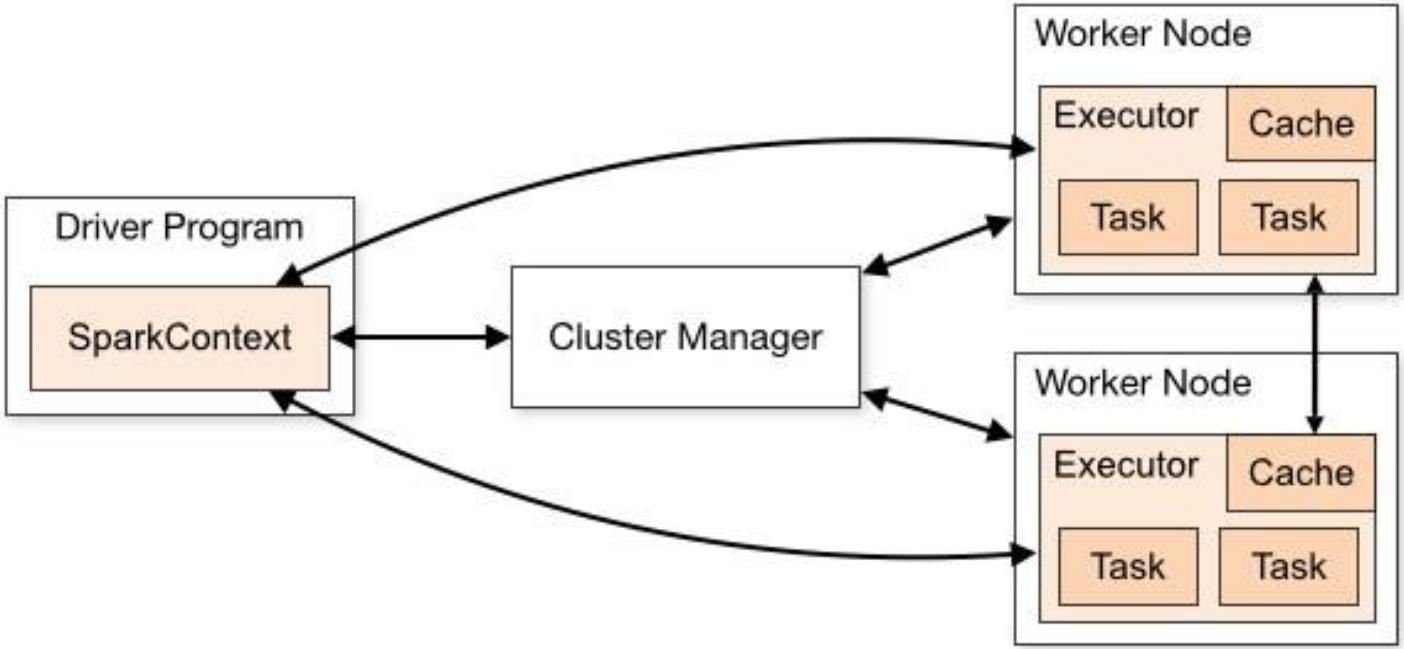
Entenda o Apache Spark

- Introdução
- Instalação
- Configuração
- Conexão

Acompanhe aqui os temas que serão tratados na videoaula

Introdução

Spark é um motor de processamento de dados para engenharia, análise e ciência de dados otimizado para *clusters* de computadores. Permite que operações comuns no processamento de dados como seleção de colunas, filtros e *joins* sejam feitas utilizando o paradigma de computação distribuída e paralela através de APIs de alto nível disponíveis em diversas linguagens.


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task

O Spark divide as operações em transformações (`filter`, `select`, `withColumn` etc.) e ações (`read.csv`, `write.csv` etc.). Operações de transformação são encadeadas até que uma operação de ação seja executada, fazendo com que todas as operações sejam executadas de uma vez. Essa característica é conhecida como *lazy evaluation*.

Nota 1: Em geral, a instalação (item 2.1) e configuração (item 2.2) de um *cluster* Spark é feito por especialistas, como uma pessoa engenheira de dados. É comum uma pessoa analista/cientista de dados começar a interagir com um *cluster* Spark a partir do item 2.3.

Nota 2: O AWS EMR (*elastic map reduce*) fornece *clusters* com gerenciador Apache Hadoop e com o Apache Spark instalado. Preço computado sobre a hora dos nós, máquinas virtuais do AWS EC2.

Instalação

Spark é uma aplicação desenvolvida na linguagem de programação Scala, que funciona em uma máquina virtual Java (JVM). Por isso, é necessário fazer o *download* da aplicação e instalar o Java em todas as máquinas (nós) do *cluster*.

- Download do Spark, versão 3.0.0.

In [ ]:

```
%%capture

!wget -q https://archive.apache.org/ +
      dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.7.tgz

!tar xf spark-3.0.0-bin-hadoop2.7.tgz
!rm spark-3.0.0-bin-hadoop2.7.tgz
```

- *Download* e instalação do Java, versão 8.

In [ ]:

```
%%capture
!apt-get remove openjdk*
!apt-get update --fix-missing
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Mesmo sendo uma aplicação Scala, o Spark disponibiliza APIs de integração em diversas linguagens de programação. O pacote Python PySpark é a API para Python. Com ele, é possível interagir com o Spark como se fosse uma aplicação Python nativa. A API é similar ao pacote Pandas e sua documentação pode ser encontrada no *link* https://spark.apache.org/docs/latest/api/python/.

Nota: a versão do PySpark deve ser a mesma que a versão da aplicação Spark.

In [ ]:
```
!pip install -q pyspark==3.0.0
```

Configuração

Na etapa de configuração, é necessário configurar as máquinas (nós) do *cluster* para que tanto a aplicação do Spark quanto a instalação do Java possam ser encontrados pelo PySpark e, consequentemente, pelo Python. Para isso, basta preencher as variáveis de ambiente `JAVA_HOME` e `SPARK_HOME` com o seus respectivos caminhos de instalação.

In [ ]:

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] =
"/content/spark-3.0.0-bin-hadoop2.7"
```

Por fim, para conectar o PySpark (e o Python) ao Spark e ao Java, pode-se utilizar o pacote Python FindSpark.

In [ ]:
```
!pip install -q findspark==1.4.2
```

O método `init()` injeta as variáveis de ambiente `JAVA_HOME` e `SPARK_HOME` no ambiente de execução Python, permitindo assim a correta conexão entre o pacote PySpark com a aplicação Spark.

In [ ]:
```
import findspark

findspark.init()
```

Conexão

Com o *cluster* devidamente configurado, vamos criar uma aplicação Spark. O objeto `SparkSession` do pacote PySpark (e seu atributo `builder`) auxiliam na criação da aplicação:

- `master:` endereço (local ou remoto) do *cluster*;
- `appName:` nome da aplicação;
- `getOrCreate:` método que de fato cria os recursos e instância a aplicação.

In [ ]:

```
from pyspark.sql import SparkSession

spark = SparkSession.\
    builder.\
    master("local[*]").
    \
    appName("pyspark-notebook").
    \ getOrCreate()
```

Com o objeto `SparkSession` devidamente instanciado, podemos começar a interagir com os dados utilizando os recursos do *cluster* através de uma estrutura de dados que já conhecemos: `DataFrames`

In [ ]:
```
!wget -q "https://raw.githubusercontent.com/"+
    "cluster-apps-on-docker/spark-standalone-cluster-on-docker/"
    + "master/build/workspace/data/uk-macroeconomic-data.csv"
-O "uk-macroeconomic-data.csv"
```

In [ ]:
```
data = spark.read.csv(
    path="uk-macroeconomic-data.csv"
    , sep=",",
    header=True
)
```

In [ ]:
```
data.show()
```

In [ ]:
```
data.printSchema()
```