

Prezado​ ​candidato.

Gostaríamos de fazer um teste que será usado para sabermos a sua proficiência nas habilidades para a vaga. O teste consiste em algumas perguntas e exercícios práticos sobre Spark e as respostas e códigos implementados devem ser armazenados​ ​no​ ​GitHub.​ ​O​ ​link​ ​do​ ​seu​ ​repositório​ ​deve​ ​ser​ ​compartilhado​ ​conosco​ ​ao​ ​final​ ​do​ ​teste.

Quando usar alguma referência ou biblioteca externa, informe no arquivo README do seu projeto. Se tiver alguma dúvida,​ ​use​ ​o​ ​bom​ ​senso​ ​e​ ​se​ ​precisar​ ​deixe​ ​isso​ ​registrado​ ​na​ ​documentação​ ​do​ ​projeto.

|  |
| --- |
| Qual​ ​o​ ​objetivo​ ​do​ ​comando​ **cache**​em​ ​Spark? |
| O objetivo da função “cache”, é como o próprio nome diz, criar um cache de dados, ou seja, guardar em memória (extraindo/consultando) um set de dados para serem acessados repetidas vezes. |



|  |
| --- |
| O​ ​mesmo​ ​código​ ​implementado​ ​em​ ​Spark​ ​é​ ​normalmente​ ​mais​ ​rápido​ ​que​ ​a​ ​implementação​ ​equivalente​ ​em MapReduce.​ ​Por​ ​quê? |
| Exato. O Spark é mais rápido que o MapReduce, já que que o MapReduce têm uma performance “pior” se comparado ao Spark, pois o MapReduce é executado em disco (o que exige máquinas mais “parrudas” e envolve riscos de I/O) enquanto o Spark é executado em memória, logo mais rápido. |

|  |
| --- |
| Qual​ ​é​ ​a​ ​função​ ​do​ **SparkContext**​? |
| A função da classe “SparkContext” é a própria “ConnectionString” se que ao invés de para um banco é para Cluster Spark. Através dessa classe é que pode ser feita toda a manipulação necessária com os dados. |

|  |
| --- |
| Explique​ ​com​ ​suas​ ​palavras​ ​o​ ​que​ ​é​ **Resilient**​​**Distributed**​​**Datasets**​ ​(RDD). |
| O RDD permite o reuso dos dados em diferentes aplicações. E é um processo tolerante a falhas, ou seja, está preparado para gravar os dados “parciais” em memória (permitindo aumentar o número de complexidade das consultas).  Ao permitir gravar as consultas em forma “parcial”, as mesmas tendem a ser mais rápidas. |

|  |
| --- |
| **GroupByKey**​é​ ​menos​ ​eficiente​ ​que​ **reduceByKey**​em​ ​grandes​ ​dataset.​ ​Por​ ​quê? |
| Porque o comando “groupByKey”, todos os dados são agrupados se baseando na chave e são embaralhados e passados para frente (pela rede), sendo que o “reduceByKey” além de agrupar pela chave também agrega as chaves primeiro para depois distribuir os dados de forma embaralhada. Ou seja, o “reduceByKey” é muito mais performático na distribuição de dados, pois ele “perde” menos tempo no embaralhamento dos dados (o que impacta diretamente no consumo de memória). |

Este​ ​documento​ ​é​ ​confidencial​ ​e​ ​não​ ​pode​ ​ser​ ​distribuído,​ ​copiado​ ​em​ ​parte​ ​ou​ ​na​ ​sua​ ​totalidade



Explique​ ​o​ ​que​ ​o​ ​código​ ​Scala​ ​abaixo​ ​faz.

val​ textFile​ =​ sc​.​textFile​(​"hdfs://..."​)

val​ counts​ =​ textFile​.​flatMap​(​line​ =>​ line​.​split​(​"​ ​"​))

​ .​map​(​word​ =>​ (​word​,​ 1​))

​ .​reduceByKey​(​\_​ +​ \_​) counts​.​saveAsTextFile​(​"hdfs://..."​)

* Criou-se uma variável para armazenar um dado arquivo;
* Sobre este arquivo foi criado uma transformação para manipulá-lo, se baseando na quebra por “ ” (espaço);
* Depois são feitas duas operações:
  + Map: para encontrar todas as palavras do arquivo;
  + Reduce: para encontrar a quantidade de ocorrências de cada palavra;
* Com o objetivo salvar o resultado obtido através da operação de MapReduce em outro arquivo, que conterá as palavras e quantidade de ocorrências de cada uma.

# HTTP ​ requests​ ​ to​ ​ the​ ​ NASA​ ​ Kennedy​​ ​Space ​ Center​ ​ WWW​ ​ ​server

**Fonte**​​**oficial**​​**do**​​**dateset**​:​ http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html **Dados**​:

* Jul​ ​01​ ​to​ ​Jul​ ​31,​ ​ASCII​ ​format, ​20.7​​ ​MB​ ​gzip​ ​compressed​,​ ​205.2​ ​MB.
* Aug​ ​04​ ​to​ ​Aug​ ​31,​ ​ASCII​ ​format, ​21.8​​ ​MB​ ​gzip​ ​compressed​,​ ​167.8​ ​MB.

**Sobre o dataset**​: Esses dois conjuntos de dados possuem todas as requisições HTTP para o servidor da NASA Kennedy

Space​ ​Center​ ​WWW​ ​na​ ​Flórida​ ​para​ ​um​ ​período​ ​específico.



Os​ ​logs​ ​estão​ ​em​ ​arquivos​ ​ASCII​ ​com​ ​uma​ ​linha​ ​por​ ​requisição​ ​com​ ​as​ ​seguintes​ ​colunas:

* **Host fazendo a requisição**​. Um hostname quando possível, caso contrário o endereço de internet se o nome não​ ​puder​ ​ser​ ​identificado.
* **Timestamp**​no​ ​formato​ ​"DIA/MÊS/ANO:HH:MM:SS​ ​TIMEZONE"
* **Requisição**​​**(entre**​​**aspas)**
* **Código**​​**do**​​**retorno**​​**HTTP**
* **Total**​​**de**​​**bytes**​​**retornados**

## Questões

​ ​Responda​ ​as​ ​seguintes​ ​questões​ ​devem​ ​ser​ ​desenvolvidas​ ​em​ ​Spark​ ​utilizando​ ​a​ ​sua​ ​linguagem​ ​de​ ​preferência.

1. Número​ ​de​ ​hosts​ ​únicos.
2. O​ ​total​ ​de​ ​erros​ ​404.
3. Os​ ​5​ ​URLs​ ​que​ ​mais​ ​causaram​ ​erro​ ​404.
4. Quantidade​ ​de​ ​erros​ ​404​ ​por​ ​dia.
5. O​ ​total​ ​de​ ​bytes​ ​retornados.

Este​ ​documento​ ​é​ ​confidencial​ ​e​ ​não​ ​pode​ ​ser​ ​distribuído,​ ​copiado​ ​em​ ​parte​ ​ou​ ​na​ ​sua​ ​totalidade