**PRACTICO DE APACHE SPARK + DOCKER**

Apache Spark es un marco de trabajo de código abierto diseñado para el procesamiento y análisis de datos a gran escala. Proporciona una plataforma unificada para la computación distribuida y el procesamiento en memoria, lo que permite manejar grandes volúmenes de datos de manera eficiente.

Spark se basa en el concepto de conjuntos de datos distribuidos llamados RDD (Resilient Distributed Datasets), que permiten a los desarrolladores realizar operaciones en paralelo en clústeres de computadoras. Estos RDD son tolerantes a fallos y se almacenan en la memoria principal, lo que acelera significativamente el rendimiento de las operaciones.

Una de las características destacadas de Spark es su capacidad para ejecutar tareas en memoria, lo que lo convierte en una opción ideal para aplicaciones que requieren procesamiento de datos en tiempo real o análisis iterativos. Además, Spark admite varios lenguajes de programación, incluyendo Scala, Java, Python y R, lo que facilita a los desarrolladores utilizar el lenguaje de su elección.

Spark ofrece un conjunto amplio de módulos y bibliotecas que amplían su funcionalidad, como Spark SQL para consultas SQL en datos estructurados, Spark Streaming para procesamiento en tiempo real de datos de transmisión, MLlib para aprendizaje automático distribuido y GraphX para el análisis de grafos.

Debido a su arquitectura flexible y escalable, Spark se ha convertido en una herramienta popular en el ámbito del análisis de big data y la ciencia de datos. Puede ser utilizado en una variedad de casos de uso, desde el procesamiento de datos en lotes hasta el análisis en tiempo real, y ha sido adoptado por muchas organizaciones para realizar análisis complejos y extraer información valiosa de grandes conjuntos de datos.

## Arquitectura de Apache Spark

Apache Spark tiene tres componentes principales: el controlador, los ejecutores y el administrador de clústeres. Las aplicaciones Spark se ejecutan como conjuntos independientes de procesos en un clúster, coordinados por el programa de controladores.

Diagrama

Descripción generada automáticamente

DATA FRAMES Y RDD  
A continuación, se presenta un cuadro comparativo entre los DataFrames y los RDD en Apache Spark:

|  |  |  |
| --- | --- | --- |
| Característica | DataFrames | RDDs |
| Abstracción | Estructura de datos tabulares con esquema definido. | Colección distribuida de objetos en bruto sin esquema definido. |
| Tipado | Fuertemente tipado. | No tipado, los datos se almacenan como objetos genéricos. |
| Operaciones | Ofrece una amplia gama de operaciones de alto nivel (transformaciones y acciones) como filtros, agregaciones, joins, etc. | Proporciona operaciones de bajo nivel como map, filter, reduce, etc. |
| Lenguajes de programación | Admite múltiples lenguajes de programación, incluyendo Scala, Java, Python y R. | Admite múltiples lenguajes de programación, incluyendo Scala, Java, Python y R. |
| Soporte para SQL | Proporciona una interfaz SQL para consultas estructuradas. | No proporciona una interfaz SQL, pero se pueden realizar operaciones similares utilizando transformaciones y acciones. |
| Tipo de dato principal | DataFrame, una colección de filas con esquema definido. | RDD, una colección de objetos distribuidos sin esquema definido. |
| Uso recomendado | Recomendado para el procesamiento de datos estructurados y consultas analíticas. | Recomendado para el procesamiento de datos sin estructura o semiestructurados y para tareas de bajo nivel y personalizadas. |

Es importante tener en cuenta que los DataFrames en Spark se construyen sobre los RDDs y proporcionan una abstracción más alta y optimizaciones específicas para consultas estructuradas y análisis de datos tabulares. Los RDDs, por otro lado, son una abstracción más baja y flexible que permite realizar operaciones de bajo nivel y personalizadas, lo que puede ser útil en escenarios más complejos o fuera de la estructura de datos tabulares tradicionales.

**Creación de Carpetas para el práctico**

Crear una carpeta en el disco C con el nombre Docker y dentro spark. En la carpeta Spark copiar el o los datasets.

# Docker

Instalar Docker: Primero, Asegúrese de tener Docker instalado en tu sistema. Puedes descargar e instalar Docker desde el sitio web oficial de Docker (https://www.docker.com/get-started).

[bitnami/spark - Docker Image | Docker Hub](https://hub.docker.com/r/bitnami/spark) Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**docker pull bitnami/spark**

Vamos a utilizar Docker compose para configurar el contenedor:

[containers/docker-compose.yml at main · bitnami/containers · GitHub](https://github.com/bitnami/containers/blob/main/bitnami/spark/docker-compose.yml)

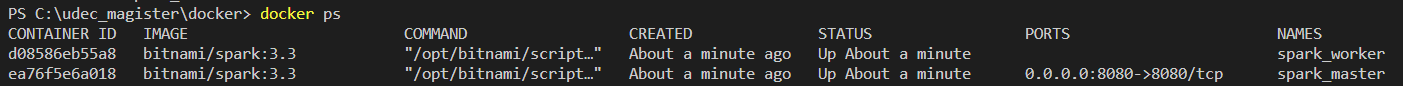
El cual contiene la siguientes parámetros, en donde agregamos el volumen. Este servirá para persistir los datos del contenedor aun cuando se apague.

|  |
| --- |
| version: '2'  services:    spark:      image: docker.io/bitnami/spark:3.3      container\_name: spark\_master      environment:        - SPARK\_MODE=master        - SPARK\_RPC\_AUTHENTICATION\_ENABLED=no        - SPARK\_RPC\_ENCRYPTION\_ENABLED=no        - SPARK\_LOCAL\_STORAGE\_ENCRYPTION\_ENABLED=no        - SPARK\_SSL\_ENABLED=no        - SPARK\_USER=spark      ports:        - '8080:8080'      volumes:        - '.:/opt/spark'    spark-worker:      image: docker.io/bitnami/spark:3.3      container\_name: spark\_worker      environment:        - SPARK\_MODE=worker        - SPARK\_MASTER\_URL=spark://spark:7077        - SPARK\_WORKER\_MEMORY=1G        - SPARK\_WORKER\_CORES=1        - SPARK\_RPC\_AUTHENTICATION\_ENABLED=no        - SPARK\_RPC\_ENCRYPTION\_ENABLED=no        - SPARK\_LOCAL\_STORAGE\_ENCRYPTION\_ENABLED=no        - SPARK\_SSL\_ENABLED=no        - SPARK\_USER=spark |

Escribimos un archivo con el nombre: docker-compose,yml con el contenido antes descrito, luego en la carpeta donde esta el archivo

**docker-compose up -d**

E jecutamos Docker ps y veremos los contenedores creados:



Si lo visualizamos en Docker desktop:

Interfaz de usuario gráfica

Descripción generada automáticamente

Podemos monitorear nuestro Master en localhost:8080

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ahora vamos a entrar en modo interactivo al contenedor master:

**ocker exec -it spark\_master bash**

Luego vamos revisar la carpeta /opt/spark si es que tiene los archivos que queremos cargar.

Vamos a escribir spark-shell (por defecto ocupa Scala)

Leemos los datasets (CSV) para tranformarlos a un Dataframe y poder manipularlos

**val df = spark.read.format("csv").option("header","true").load("/opt/spark/\*.csv")**

Luego podemos ver su esquema:

**df.printSchema**

Texto

Descripción generada automáticamente

Luego podemos seleccionar algunas columnas:

**df.select("event\_time","event\_type","price").show()**

o filtrar por una o varias columnas:

**df.select("event\_time","event\_type","price").filter("event\_type=='view'").show()**

también podemos obtener ,por ejemplo , las distintas marcas:

**df.select("brand").distinct().show()**

Ejemplo, buscar los productos que se han vendido en conjunto con otro producto (seleccionamos uno a azar o el primero)

Seleccionamos el primer product\_id del df cuyo evento\_type es cart.

**df.select("product\_id").filter("event\_type='cart'").first()**

Luego podemos definir un dataframe con el resultado de la consulta sobre el DF original, en donde seleccionar las sesiones distintas de los usuarios cuyo evento type es= cart y el producto es el que seleccionamos previamente

**val sesions=df.select("user\_session").filter("event\_type='cart' AND product\_id=xxx").distinct()**

Luego obtenermos los productos que han participado con esta condición:

**val products=df.select("product\_id").filter("event\_type='cart' AND product\_id<>5844305").filter(df("user\_session").isin(sesions("user\_session")) )**

Contamos los registros

**products.count()**

Sin embargo necesitamos los productos distintos y los almacenamos:

**val pd = products.distinct()**

Ahora exportamos los resultados:

**pd.write.mode("overwrite").csv("/opt/spark/resultados")**

Abrimos la carpeta:

Captura de pantalla de computadora

Descripción generada automáticamente

Crear una vista temporal para ocupar SQL

**df.createOrReplaceTempView("data")**

Luego podemos hacer consultas SQL sobre los datos

**spark.sql("select \* from data limit 5").show()**

[Spark Web UI - Understanding Spark Execution - Spark By {Examples} (sparkbyexamples.com)](https://sparkbyexamples.com/spark/spark-web-ui-understanding/)