

De 0 a 100: Apache Spark y Azure Databricks para .NET

Bienvenidos
Acerca de...



Jose María Flores Zazo, autor

*¡Hola! Gracias por entrar en
“De 0 a 100: Apache Spark y Azure Databricks para .NET”.
Espero poder aportarte los conocimientos mínimos y necesarios
con este workshop para que puedas ponerlo en práctica.*



¿Qué es Apache Spark?

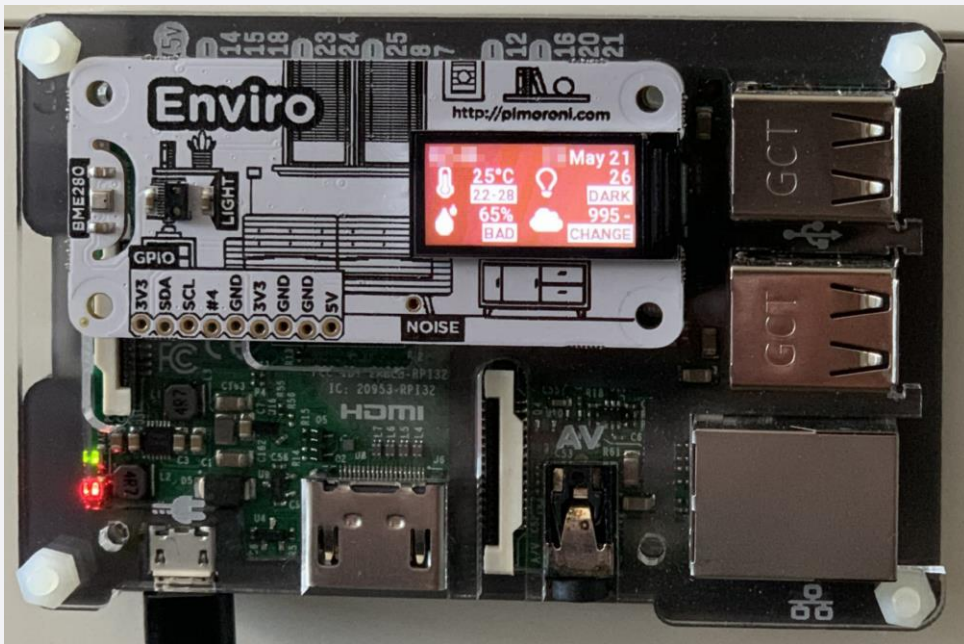
Introducción (1/12)

Apache Spark – Procesamiento distribuido para conjuntos de datos masivos

Es una plataforma de análisis de datos que ha hecho posible que el big data accesible a todos los desarrolladores a través del procesamiento a gran escala.

Apache Spark tiene la capacidad, por ejemplo, de leer un archivo CSV con millones de registros.

El fichero `monitoring.csv`, contiene la lectura real de mi entorno de trabajo mediante una [Raspberry Pi 3 Model B](#) y [Enviro de Pimoroni](#). El fichero se encuentra en el proyecto `ReadEnviroCSV` así como el código en Python que escribe en un Azure Storage Table.



PartitionKey	RowKey	Timestamp	CreateAt	Temperature	Humidity	Light	Pressure
office	000031fc-b016-11eb-b59f-b827eb4165bb	2021-05-08T15:56:50.133Z	1620489410009885	25.691572457023673	67.74881587489502	9.6637	996.9107
office	00003164-b3aa-11eb-b59f-b827eb4165bb	2021-05-13T05:13:49.122Z	16208828209007862	24.87502133255893	63.32720669198268	4.83185	995.1824281
office	00004230-aeaf-11eb-b59f-b827eb4165bb	2021-05-08T20:02:35.314Z	1620331355.2073176	25.552952978854025	68.94154808207062	0	997.3548
office	0000496e-af50-11eb-b59f-b827eb4165bb	2021-05-07T23:30:09.025Z	1620403066.894684	25.23610540962697	66.44119201159731	0	1001.95576
office	00007738-bd0a-11eb-b59f-b827eb4165bb	2021-05-08T11:59:23.564Z	1620561945.4291978	26.302187274830937	67.8807800638279	7.89335	991.14854
office	00006968-bd0a-11eb-b59f-b827eb4165bb	2021-05-08T10:04:53.594Z	1620554663.4873755	25.0721462216902783	69.6024304593617	39.59605	991.02035
office	000077a2-b289-11eb-b59f-b827eb4165bb	2021-05-11T18:45:04.557Z	1620758704.4550195	25.633698479983362	64.80291159767853	7.15555	998.9217431
office	00008278-ae32-11eb-b59f-b827eb4165bb	2021-05-08T06:12:13.886Z	1620281533.5888724	23.3764194643621	72.83640781588897	10.7704	998.1844
office	0000828e-b10-11eb-b59f-b827eb4165bb	2021-05-11T00:29:51.601Z	1620692991.455663	25.442304368514154	70.35289599051829	0	990.8785904
office	0000a494-b138-11eb-b59f-b827eb4165bb	2021-05-10T06:21:48.056Z	1620627707.9550629	26.33048389767536	70.82930437043541	15.0489	989.8420
office	0000afac-b148-11eb-b59f-b827eb4165bb	2021-05-07T04:27:18.125Z	1620462636.00734497	25.820860360274487	69.24627523566277	0	989.545975
office	0000af12-b2aa-11eb-b59f-b827eb4165bb	2021-05-13T05:43:27.108Z	1620884546.9986215	25.258226295027143	64.20705410076608	64.843	995.267436
office	0000a538-b315-11eb-b59f-b827eb4165bb	2021-05-12T11:27:14.099Z	1620818833.9833976	26.04872302962908	62.846200811312244	57.2634	993.6160
office	000108fa-b31a-11eb-b59f-b827eb4165bb	2021-05-12T12:03:01.586Z	1620820981.9838152	26.08048139744357	64.1863195830729	62.09525	993.2705407
office	00010990-b323-11eb-b59f-b827eb4165bb	2021-05-12T13:07:27.058Z	1620824846.95487	25.56698342400484	62.22703388655874	58.6097	992.3384
office	00010c4c-b27a-11eb-b59f-b827eb4165bb	2021-05-11T16:57:42.120Z	1620752262.00773	24.023773454638615	62.5055797363100214	40.77245	997.666739
office	00011c7c-b019-11eb-b59f-b827eb4165bb	2021-05-08T16:18:18.618Z	1620406698.5020652	25.77005957043999	67.17039613859916	6.3626	996.8154
office	0001215e-ba33-11eb-b59f-b827eb4165bb	2021-05-08T06:19:23.184Z	1620281963.0802138	23.23076386245181	72.87785118010979	4.8591	998.23177
office	000129ee-b313-11eb-b59f-b827eb4165bb	2021-05-12T11:12:55.124Z	1620817975.0077138	25.920561328175669	62.7500001640815	60.9334	993.686113
office	00013124-b322-11eb-b59f-b827eb4165bb	2021-05-12T13:00:17.562Z	1620824417.4572465	25.80384121471057	61.763281406106906	58.3701	992.4974
office	0001341e-b280-11eb-b59f-b827eb4165bb	2021-05-11T17:40:39.130Z	1620754838.986928	25.484038529798394	64.07261445083724	27.6448	997.9829601
office	0001415c-b3ac-11eb-b59f-b827eb4165bb	2021-05-13T05:28:08.111Z	1620883688.002486	25.110766336848222	63.85744226673918	53.77785	995.1889
office	00014480-b0b0-11eb-b59f-b827eb4165bb	2021-05-08T10:18:12.593Z	1620555552.5073137	25.424907906061627	69.1972223734316	71.04015	991.120381
office	00014718-b147-11eb-b59f-b827eb4165bb	2021-05-10T04:20:06.616Z	1620625406.5085692	25.920078803240499	68.77800115812093	0	989.37669
office	0001485a-aecc-11eb-b59f-b827eb4165bb	2021-05-07T03:34:38.201Z	1620347676.0899315	24.481786130999315	69.4702095108127	0	998.8104
office	00015386-ba20-11eb-b59f-b827eb4165bb	2021-05-08T18:00:54.072Z	1620487140.954551	25.8201437391233	66.50095350319104	4.83185	996.263400
office	00015932-aeec-11eb-b59f-b827eb4165bb	2021-05-08T22:40:04.253Z	1620340804.4539328	25.30764824589987	68.71399215961289	0	997.8081
office	00018564-b14c-11eb-b59f-b827eb4165bb	2021-05-10T04:55:54.110Z	1620622553.9882824	25.853903134912613	68.96123442136542	0	989.5820281
office	000188ee-b27b-11eb-b59f-b827eb4165bb	2021-05-11T17:04:51.606Z	1620752691.5021656	25.976999504085616	63.855418683631045	33.23775	997.6400
office	0001a236-b28b-11eb-b59f-b827eb4165bb	2021-05-11T18:59:23.803Z	1620795963.4555843	25.80260973255329	64.14287858402027	0.8296	999.18915
office	0001b02e-b0af-11eb-b59f-b827eb4165bb	2021-05-08T10:15:20.099Z	1620595123.0990222	25.7278242400783	68.04623787935062	53.8746	995.08925
office	0001c32a-b31a-11eb-b59f-b827eb4165bb	2021-05-12T13:24:30.207Z	1620822369.9788975	26.370423043715848	63.41476036339864	68.21825	993.8444
office	0001c798-ba20-11eb-b59f-b827eb4165bb	2021-05-08T18:12:50.557Z	1620487570.454823	25.738527184144466	66.9656571076807	1.6592	996.274744
office	0001d1a0-b1f2-11eb-b59f-b827eb4165bb	2021-05-11T00:44:10.603Z	1620693850.4559007	26.09033830501756	68.93743398172285	0	991.0177
office	0001ec0f-b0b1-11eb-b59f-b827eb4165bb	2021-05-08T10:26:22.091Z	1620555982.0078657	25.68081834261753	68.6155350236939	49.37005	991.5020871
office	0001f08e-af90-11eb-b59f-b827eb4165bb	2021-05-07T23:57:37.588Z	1620431857.4549258	25.4377163014339	65.0855068085877	0	1001.6330
office	0001f8c6-b13a-11eb-b59f-b827eb4165bb	2021-05-10T06:36:07.058Z	1620628566.9531633	26.372801018527607	70.57368315496925	12.17185	990.33628
office	00021c70-b3af-11eb-b59f-b827eb4165bb	2021-05-13T05:49:56.868Z	1620884076.5011818	25.106532631988332	64.89504738533434	24.54575	995.28564
office	0002237e-wd9b-11eb-b59f-b827eb4165bb	2021-05-05T13:18:29.202Z	1620217109.0890513	26.27175889420996	65.15654988218991	23.256	996.6020



¿Por qué Apache Spark?

Introducción (2/12)

Con el fichero generado `measures.csv`, vamos a realizar un **hands on lab**; esta vez voy a enfocar de forma diferente los workshops e iré mezclando teoría con práctica.

El fichero como podrás observar pesa 270MB aproximadamente, esto no es big data, pero si juntáramos los 1000 dispositivos Enviro que tuviéramos en todas las habitaciones de una cadena de hoteles, tendríamos la siguiente relación:

El fichero contiene aproximadamente los datos de una semana (7 días) que han generado 270MB, por tanto, estamos hablando que para 1000 dispositivos las cifras ascienden 270GB. Esto aun no se considera big data, pero podríamos extrapolar a un año de datos (54 semanas) y aquí si hablamos de big data, ya que se trata de 14,6TB aproximadamente. Una consulta para saber la media de temperatura del año para todos los dispositivos de una semana puede que en SQL Server no tarde más de 15 segundos, pero para los 14,6TB tardará varios minutos o puede que alguna hora en calcular una simple media de tiempo.

Una de las definiciones que uso para explicar muy rápidamente que:

Big Data es tal cantidad de datos que no puede ser procesada en tiempo real¹

1. Es mi definición, no una definición formal.



¿Por qué Apache Spark?

Introducción (3/12)

El anterior problema puede ser atacado de diversas formas, pero la más rápida es cocinar datos de una semana, llevarlo a una tabla donde acumulamos y al final tenemos 54 semanas x 1000 registros = Valores Medios, esto se obtiene en 0,5 segundos. O bien cambiamos la granularidad de los datos o bien dejamos el que dispositivo nos envíe información media cada hora, o ...

Pero vamos a dar un paso más allá y nos abstraemos de las cifras anteriores y las soluciones anteriores. Pensemos que estamos generando diariamente TB de información telemétrica que ninguna de las anteriores técnicas logra procesar en tiempo aceptables. Y que queremos analizar los datos en bruto por razones científicas o las que estipule el owner de los datos.

Una vez aclarado que es Big Data y por qué debemos procesar esos TB de información, vamos a ir armando nuestro entorno y aprendiendo a procesar esa ingente cantidad de datos con la potencia que nos ofrece Apache Spark.



¿Cómo usamos Apache Spark?

Introducción (4/12)

Apache Spark – <https://spark.apache.org>

Aunque soy partidario de usar Docker y generar una imagen para poder trabajar directamente con ese software, en esta ocasión vamos a realizar una instalación local. Apache Spark ya de por si es una herramienta compleja como para añadirle más variables usando Docker.

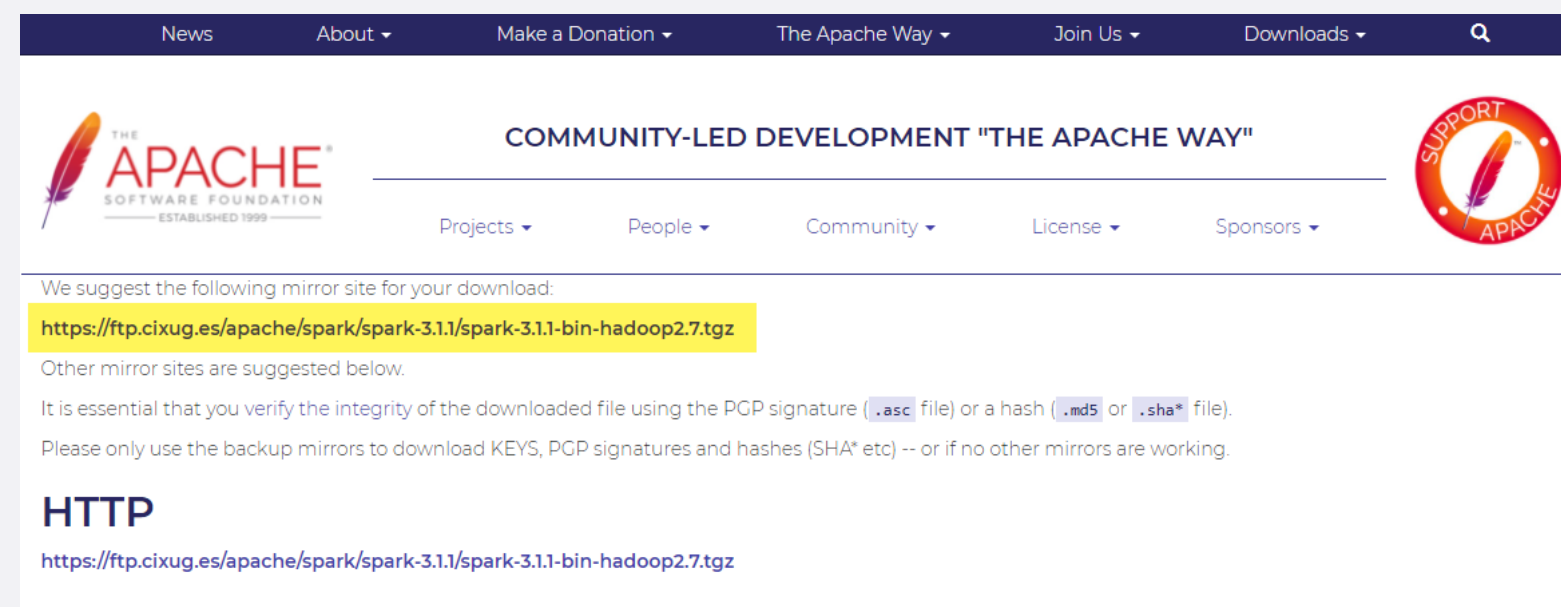
Por tanto, instalemos Apache Spark en Windows 10.

Mi recomendación es que instales un JDK superior al 8, mira que versión tienes instalada:

```
java --version
```

Y que descargues la versión más reciente de Apache Spark:

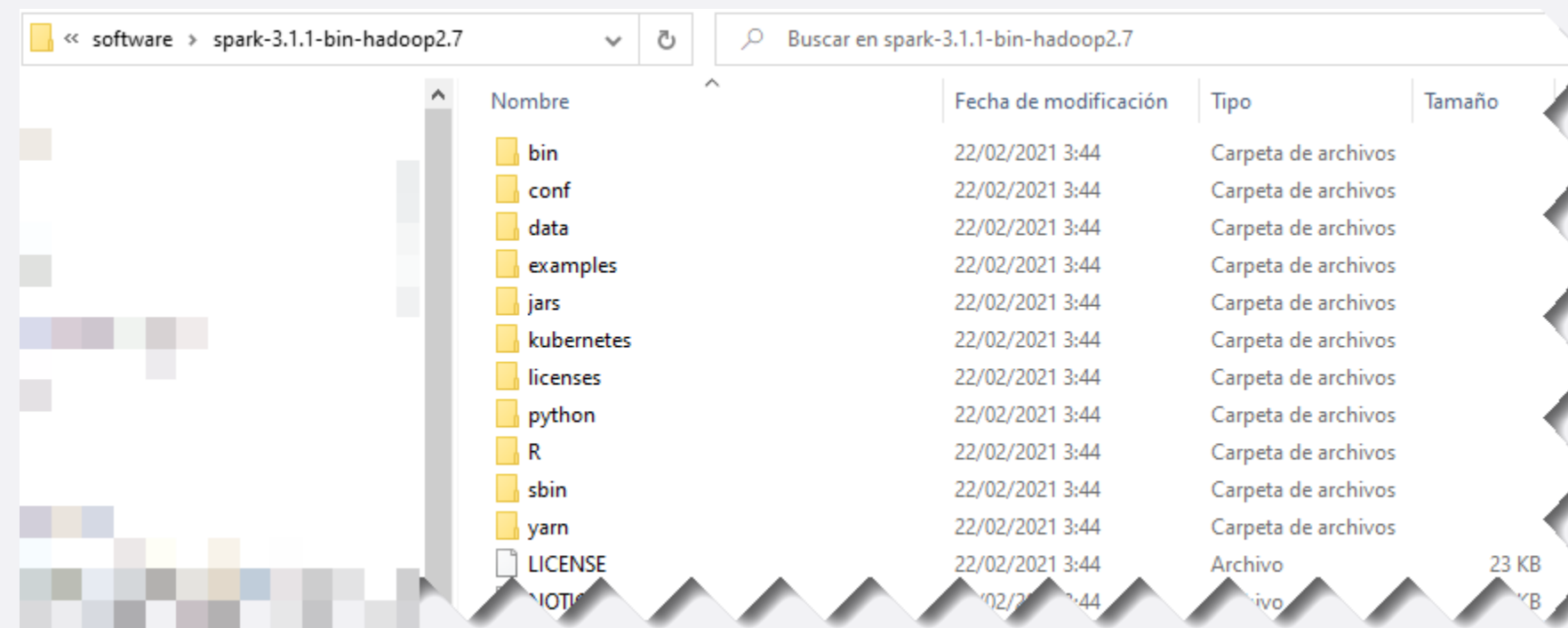
<https://spark.apache.org/downloads.html>

The screenshot shows the Apache Spark download page. At the top is a dark blue navigation bar with links: News, About, Make a Donation, The Apache Way, Join Us, and Downloads. Below this is the Apache Software Foundation logo on the left and a circular 'SUPPORT' logo on the right. The main heading is 'COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"'. Below the heading are links for Projects, People, Community, License, and Sponsors. The main content area states: 'We suggest the following mirror site for your download:' followed by a highlighted URL: 'https://ftp.cixug.es/apache/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz'. It then says 'Other mirror sites are suggested below.' and provides instructions on how to verify the integrity of the downloaded file using PGP signatures or hashes. At the bottom, under the heading 'HTTP', it provides another URL: 'https://ftp.cixug.es/apache/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz'.

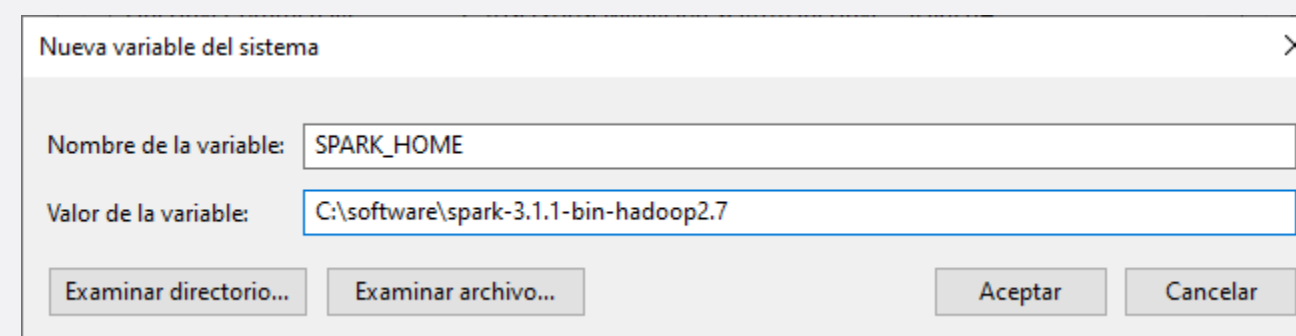
¿Cómo usamos Apache Spark?

Introducción (5/12)

Una vez descargado el fichero, descomprímelo en el lugar que mejor te venga:



Crea la variable de entorno de Spark.



¿Cómo usamos Apache Spark?

Introducción (6/12)

Ahora debemos instalar las [Winutils](#), que no es nada más los binarios compilados para Windows de Hadoop.

Si hemos bajado la versión `spark-3.1.1-bin-hadoop2.7`, deberás bajar los últimos binarios que se corresponda con la versión 2.7, en nuestro caso: 2.7.7

hadoop-2.7.4/bin	fixed exe and lib 265-312
hadoop-2.7.6/bin	fixed exe and lib 265-312
hadoop-2.7.7/bin	fixed exe and lib 265-312
hadoop-2.8.0/bin	fixed exe and lib 265-312

Si lo deseas también puedes usar los binarios que compiles en tu propia máquina.

El contenido `bin` de la versión de Hadoop, lo extraes en la carpeta `bin` de Spark.

Ya no necesitamos hacer nada más.



¿Cómo usamos Apache Spark?

Introducción (7/12)

Apache Hadoop – <https://hadoop.apache.org>

Apache Hadoop es un entorno de trabajo para programar aplicaciones distribuidas que manejan grandes volúmenes de datos (Big Data).

Permite a las aplicaciones trabajar con miles de nodos de red y petabytes de datos.

Esta herramienta se inspiró en la documentación de Google sobre MapReduce y Gogle File System (GFS).

Esta auspiciado bajo la organización Apache, por tanto, tiene una licencia libre.



Y hasta aquí este pequeño incido sobre Apache Hadoop.



¿Cómo usamos Apache Spark?

Introducción (8/12)

Test – ¿Nuestro entorno esta bien configurado?

Desde el CLI de Windows 10, vamos a ejecutar el siguiente comando, yo he usado la opción de administrador:

`spark-shell`

```
Administrador: Símbolo del sistema - spark-shell
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>spark-shell
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/C:/software/spark-3.1.1-bin-hadoop2.7/jars/spark-unsafe_2.12-3.1.1.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1620970260885).
Spark session available as 'spark'.
Welcome to

  ____      __
 / _  \    /  \
/_  ___/  _/___\
 \___  \/_____\
    \/

version 3.1.1

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 9.0.4)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 21/05/14 07:31:13 WARN ProofsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of P
rocessTree metrics is stopped
```

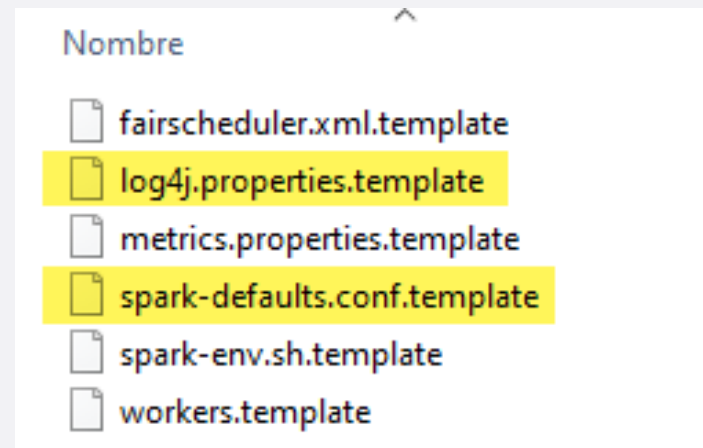
Lo normal es que te salgan una serie de warnings y es debido a que nos falta un paso para configurar, ya nos lo advierten en la ventana anterior.



¿Cómo usamos Apache Spark?

Introducción (9/12)

Establecemos la configuración por defecto, para ello entra en el directorio conf de la instalación de Spark. Y dupliquemos estos 2 ficheros (que actualmente son plantillas) para convertirlos en la configuración:



Por tanto, tendremos esos dos ficheros en la misma carpeta, pero sin la extensión template. Además, en log4j.properties debemos:

- Modificar el nivel de advertencia:

```
log4j.rootCategory=ERROR, console
```

- Añadimos estas dos líneas más:

```
log4j.logger.org.apache.spark.util.ShutdownHookManager=OFF  
log4j.logger.org.apache.spark.SparkEnv=ERROR
```



¿Cómo usamos Apache Spark?

Introducción (10/12)

El fichero spark-default.conf, debe configurarse un poco a las necesidades de nuestra máquina. Yo tengo 16GB, no puedo usar todo, por tanto, reservo solo 6 GB y también quiero que use **Delta Lake** Library para usar **Databricks**.

```
spark.executor.memory=6g
spark.jars.packages=io.delta:delta-core_2.12:0.7.0
```

Volvemos a lanzar el `spark-shell`:

```
C:\WINDOWS\system32>spark-shell
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/C:/software/spark-3.1.1-bin-hadoop2.7/jars/unsafe-0.2.0.jar) of method java.lang.Object.toString()
WARNING: Please consider reporting this to the vendor of the offending class.
```

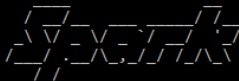
```
conf: [default]
found io.delta#delta-core_2.12;0.7.0 in central
found org.antlr#antlr4;4.7 in central
found org.antlr#antlr4-runtime;4.7 in central
found org.antlr#antlr-runtime;3.5.2 in central
found org.antlr#antlr4;4.7 in central
```

```
org.antlr#antlr4;4.7 from central in [default]
org.antlr#antlr4-runtime;4.7 from central in [default]
org.glassfish#javax.json;1.0.4 from central in [default]
```

	conf	number	modules search	dwnlded	evicted		artifacts number	dwnlded
	default	8	8	8	0		8	8

```
:: retrieving :: org.apache.spark#spark-submit-parent-b277de37-b16a-41a3-9989-33de87dbf403
confs: [default]
8 artifacts copied, 0 already retrieved (15071kB/87ms)
Setting default log level to "WARN"
```

```
Spark context available as 'sc' (master = local[*], app id = local-1620912730880).
Spark session available as 'spark'.
Welcome to
```

 version 3.1.1

```
Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 9.0.4)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```



¿Cómo usamos Apache Spark?

Introducción (11/12)

Test – ¿Nuestro entorno esta bien configurado?

Y ejecutamos, el que espero sea, el único comando de **Scala** de todo el workshop:

```
spark.sql("select * from range(10)").withColumn("VALUE", col("ID")).show
```

```
scala> spark.sql("select * from range(10)").withColumn("VALUE", col("ID")).show
+-----+
| id|VALUE|
+-----+
| 0|    0|
| 1|    1|
| 2|    2|
| 3|    3|
| 4|    4|
| 5|    5|
| 6|    6|
| 7|    7|
| 8|    8|
| 9|    9|
+-----+
```

Han salido anteriormente dos palabras nuevas, por tanto, realizo un inciso:

- **Databrick** y **Delta Lake**, no confundir con las compañías del mismo nombre fundadas por los creadores de Apache Spark o Delta Lake. A grosso modo, es la plataforma analítica basada en Apache Spark y que nos permite auto escalar y dimensionar Apache Spark.
- **Scala**, el lenguaje de programación multiparadigma diseñado para esperar patrones comunes de programación, integra características de los lenguajes funcionales y orientados a objetos. Yo hace tiempo que no lo uso, desde que puedo utilizar la integración con .NET.



¿Cómo configuramos Workers de .NET?

Introducción (12/12)

Worker – Otro concepto más a explicar

En resumen, es como se conecta .NET con el código JVM. Lo realiza mediante un socket TCP, por tanto, el código de .NET podrá llamar al código JVM.

Existe una excepción, las User-Defined-Functions, o UDFs. Estas se ejecutan en otro proceso separado y específico de Microsoft al que conecta con la JVM y envía mensajes tipo forward y backward (es decir hacia delante y hacia atrás). Para ello necesitamos un ejecutable .NET para Apache Spark y configurar una variable de entorno para apuntar al ese ejecutable.

Solo quería hacer mención del concepto y que lo tengas en mente por qué lo veremos más adelante.

Para configurar UDF debemos ir a: <https://github.com/dotnet/spark/releases>, elegir la versión que quieres usar, extraerlo y generar la variable de entorno DOTNET_WORKER_DIR al directorio Microsoft.Spark.Worker.

Pero tal y como he dicho antes, más adelante veremos un pequeño ejemplo y quizá sea lo único que veas, yo no suelo usar este Worker de .NET.



Programando con .NET para Apache Spark

Manos a la obra (1/5)

Primer programa – HelloWorldSpark

Vamos a introducirnos con el clásico, no podría ser de otra forma. Cuando tengamos este ejemplo funcionando, ya crearemos nuestro segundo programa y será relacionado con el dataset del que hablaba en la introducción... un poco de paciencia.

¿Qué vamos a necesitar?

- dotnet
- VS Code
- Conocimientos de C#.

Creamos el directorio: HelloWorldSpark

Ejecutamos: `cd HelloWorldSpark`

Inicializamos: `dotnet new console`

Añadimos el NuGet: `dotnet add package Microsoft.Spark`

Y entramos en VS Code: `code .`



Programando con .NET para Apache Spark

Manos a la obra (2/5)

Este es el programa que vamos a realizar:

The screenshot shows a C# program in Visual Studio. The code is as follows:

```
1 using Microsoft.Spark.Sql;
2 using System;
3
4 namespace HelloWorldSpark
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             var spark = SparkSession
11                 .Builder().AppName("HelloWorldSparkApp").GetOrCreate();
12             var dataframe = spark.Sql("select id, rand() as random_number from range(1000)");
13             dataframe // DataFrame
14                 .Write() // DataFrameWriter
15                 .Format("csv").Option("header", true).Option("sep", "|").Mode("overwrite").Save(args[1]);
16             foreach (var row in dataframe.Collect())
17             {
18                 if (row[0] as int? % 2 == 0)
19                 {
20                     Console.WriteLine($"row: {row[0]}");
21                 }
22             }
23         }
24     }
25 }
26
```

Four red callout boxes provide explanations for specific parts of the code:

- Usando SQL creamos 1000 registros** (Using SQL we create 1000 records) - points to line 12.
- Crear la sesión de Spark con C#** (Create the Spark session with C#) - points to line 10.
- Escribimos el DataFrame en disco** (We write the DataFrame to disk) - points to line 15.
- Mostramos por consola** (We show on console) - points to line 20.



Programando con .NET para Apache Spark

Manos a la obra ^(3/5)

Y ejecutamos desde la línea de comandos, para ello usaremos un comando nuevo de Spark:

```
spark-submit
```

Como resumen podéis usar la directiva `--help` para que veáis lo que podemos llegar a hacer. De momento solamente nos interesa:

- `spark-submit`.
- `--class`, el nombre de la clase de `DotNetRunner`.
- Ruta al fichero JAR, el fichero jar que podéis encontrar en `/debug/net5.0/microsoft-spark-[version].jar`
- Y los argumentos para nuestro programa de dotnet.

Asegúrate de la versión de JAR que debemos indicar usando: `spark-shell --version`, si tu version no está contenida en los ficheros JAR anteriores deberás hacer un downgrade de tu instalación de Spark.

Ejecutamos:

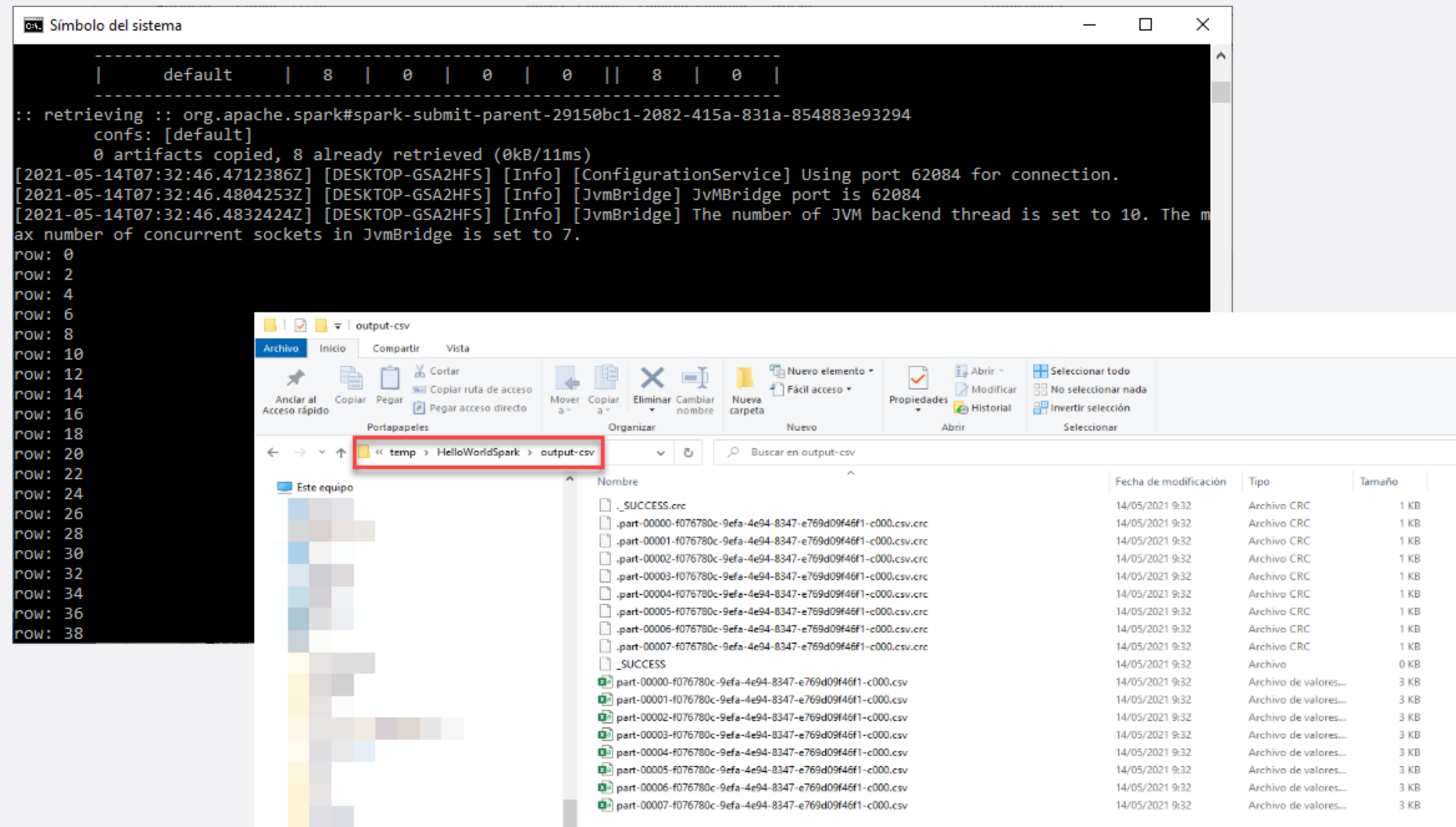
```
spark-submit --class org.apache.spark.deploy.dotnet.DotnetRunner  
"C:\temp\HelloWorldSpark\bin\Debug\net5.0\microsoft-spark-3-0_2.12-1.1.1.jar"  
dotnet run "C:\temp\HelloWorldSpark" "C:\temp\HelloWorldSpark\output-csv"
```



Programando con .NET para Apache Spark

Manos a la obra (4/5)

Y como podrás comprobar aquí tenemos el resultado:



The image shows a Windows command prompt window titled "Símbolo del sistema" and a File Explorer window titled "output-csv".

The command prompt window displays the following output:

```
-----  
|      default      | 8 | 0 | 0 | 0 | 0 | | 8 | 0 |  
-----  
:: retrieving :: org.apache.spark#spark-submit-parent-29150bc1-2082-415a-831a-854883e93294  
confs: [default]  
0 artifacts copied, 8 already retrieved (0kB/11ms)  
[2021-05-14T07:32:46.4712386Z] [DESKTOP-GSA2HFS] [Info] [ConfigurationService] Using port 62084 for connection.  
[2021-05-14T07:32:46.4804253Z] [DESKTOP-GSA2HFS] [Info] [JvmBridge] JvmBridge port is 62084  
[2021-05-14T07:32:46.4832424Z] [DESKTOP-GSA2HFS] [Info] [JvmBridge] The number of JVM backend thread is set to 10. The m  
ax number of concurrent sockets in JvmBridge is set to 7.  
row: 0  
row: 2  
row: 4  
row: 6  
row: 8  
row: 10  
row: 12  
row: 14  
row: 16  
row: 18  
row: 20  
row: 22  
row: 24  
row: 26  
row: 28  
row: 30  
row: 32  
row: 34  
row: 36  
row: 38
```

The File Explorer window shows the contents of the "output-csv" folder. The address bar shows the path: "temp > HelloWorldSpark > output-csv". The folder contains the following files:

Nombre	Fecha de modificación	Tipo	Tamaño
._SUCCESS.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00000-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00001-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00002-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00003-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00004-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00005-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00006-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
.part-00007-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv.crc	14/05/2021 9:32	Archivo CRC	1 KB
._SUCCESS	14/05/2021 9:32	Archivo	0 KB
part-00000-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00001-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00002-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00003-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00004-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00005-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00006-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB
part-00007-f076780c-9efa-4e94-8347-e769d09f46f1-c000.csv	14/05/2021 9:32	Archivo de valores...	3 KB



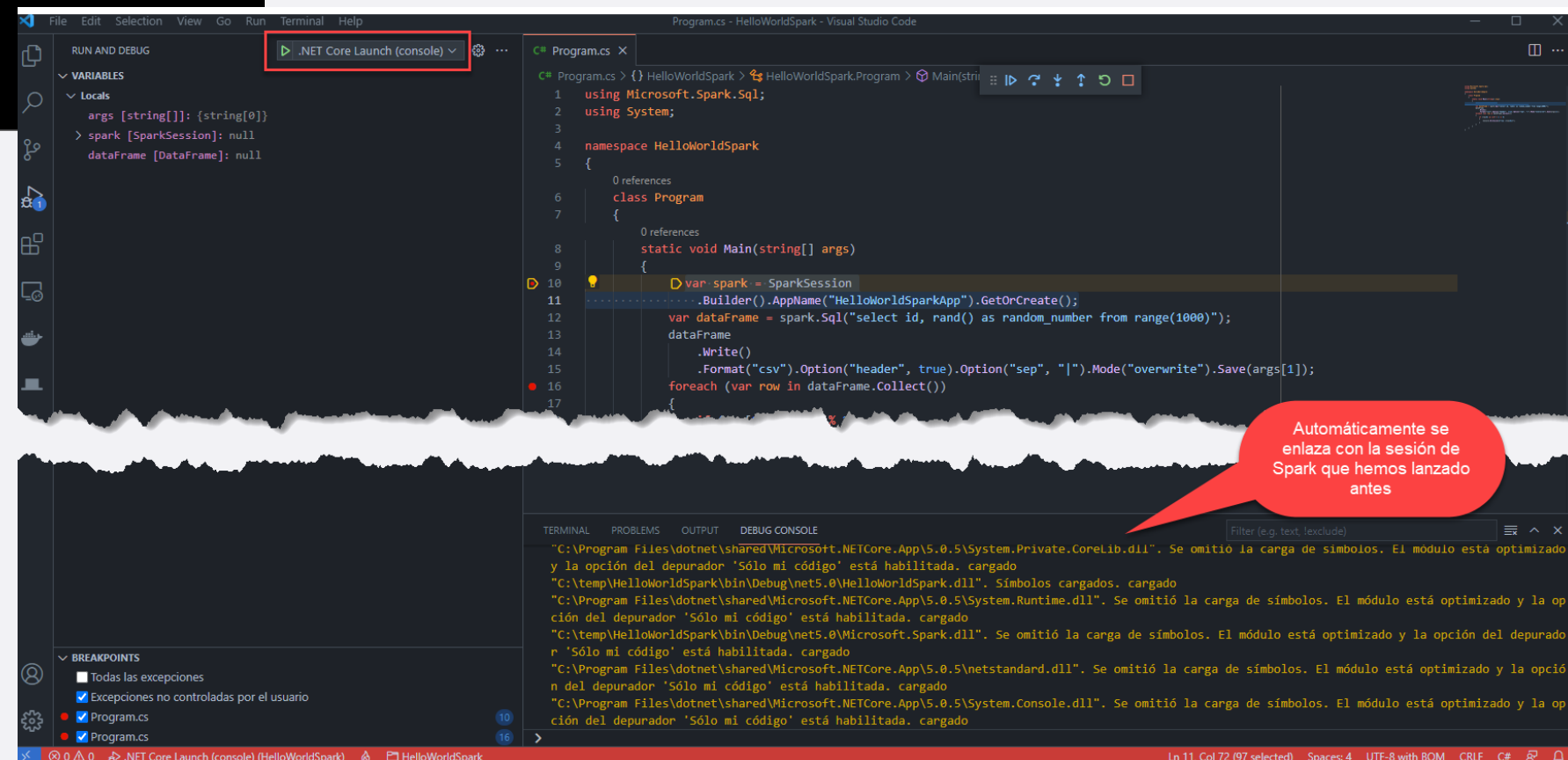
Programando con .NET para Apache Spark

Manos a la obra (5/5)

¿Cómo depuramos?

```
spark-submit --class org.apache.spark.deploy.dotnet.DotnetRunner  
"C:\temp\HelloWorldSpark\bin\Debug\net5.0\microsoft-spark-3-0_2.12-1.1.1.jar"  
debug
```

```
      conf | number| search|dwnlded|evicted|| number|dwnlded|  
-----  
      default |      8 |      0 |      0 |      0 ||      8 |      0 |  
-----  
:: retrieving :: org.apache.spark#spark-submit-parent-b30913fb-0185-4790-b049-6b1664fcf51b  
  confs: [default]  
  0 artifacts copied, 8 already retrieved (0kB/11ms)  
*****  
* .NET Backend running debug mode. Press enter to exit *  
*****
```



¡Atención!
Si te da error; cuidado con la variable de entorno.



Demo

Probando nuestro dataset de Enviro ^(1/2)

Segundo programa – ReadEnviroCSV

Vamos a dar una solución rápida al ejemplo que habíamos planteado de la media de temperatura del fichero que genero con la Raspberry PI + Enviro + la exportación a CSV de la tabla que tengo en Azure Storage Tables:

Primero lanzamos en modo depuración:

```
spark-submit --class org.apache.spark.deploy.dotnet.DotnetRunner  
"C:\temp\HelloWorldSpark\bin\Debug\net5.0\microsoft-spark-3-0_2.12-1.1.1.jar"  
debug
```

Ejecutamos el programa:

```
var spark = SparkSession  
    .Builder()  
    .AppName("Enviro_Spark")  
    .GetOrCreate();  
DataFrame dataframe = spark.Read().Csv(@"C:\temp\ReadEnviroCSV\monitoring.csv");  
dataframe.CreateOrReplaceTempView("Measures");  
DataFrame sqlDF = spark.Sql("SELECT AVG(Measures._c5) as AvgTemperature FROM Measures");  
sqlDF.Show();  
spark.Stop();
```



Demo

Probando nuestro dataset de Enviro (2/2)

Instantáneamente obtenemos el resultado, esto mismo en SQL Server hubiera tardado un par de segundos en mi máquina (tener esto presente, siempre hablo de mi máquina para tener una medida controlada):

```
1  using System;
2  using Microsoft.Spark.Sql;
3
4  namespace ReadEnviroCSV
5  {
6      0 references
7      class Program
8      {
9          0 references
10         static void Main(string[] args)
11         {
12             var spark = SparkSession
13                 .Builder()
14                 .AppName("Enviro_Spark")
15                 .GetOrCreate();
16             DataFrame dataframe = spark.Read().Csv(@"C:\temp\ReadEnviroCSV\monitoring.csv");
17             dataframe.CreateOrReplaceTempView("Measures");
18             DataFrame sqlDF = spark.Sql("SELECT AVG(Measures._c5) as AvgTemperature FROM Measures");
19             sqlDF.Show();
20             spark.Stop();
21         }
22     }
23 }
24
25
26
27
28
29
```



```
+-----+
| AvgTemperature |
+-----+
| 25.44890240172511 |
+-----+
```



Una breve guía

Convertir aplicaciones de PySpark/Scala a .NET ^(1/4)

Un poco de ayuda – Existen más proyectos en esos lenguajes que en C#

Debido a que la mayoría de los programas o bien están desarrollados en Scala o en PySpark, es necesario que conozcas un poco como se trabaja con ellos para que puedas adaptarlo a .NET.

Tambien encontrarás programas en Java y R, pero los anteriormente indicados son más abundantes.

Con Python:

La parte más importante son las importaciones en .NET es un `using`, en Python son `open`.

```
open Microsoft.Spark.Sql → using Microsoft.Spark.Sql
```

Una función en Python es:

```
def basic_df_example(Spark):  
    df = spark.read.json ...  
    ...
```

Mientras que en C# es:

```
Static void BasicDfExample (SparkSession Spark)  
{  
    var df = Spark.read().Json ...  
    ...  
}
```



Una breve guía

Convertir aplicaciones de PySpark/Scala a .NET ^(2/4)

Imprimir un esquema de DataFrame:

```
df.printSchema() → dataframe.PrintSchema()
```

Mostrar el contenido de una sola columna:

```
df.select("name").show() → dataframe.Select("name").show()
```

Mostrar el contenido de dos columnas y hacer una operación:

```
df.select(df['name'], df['age'] + 1).show() → dataframe.Select(dataFrame["name"], dataframe["age"] + 1).show()
```

Un filtro:

```
df.filter(df['age'] > 44).show() → dataframe.Filter(dataFrame["age"].Gt(44)).show()
```

Un agregado:

```
df.groupBy("age").count().show() → dataframe.GroupBy(dataFrame["age"]).Count().show()
```



Una breve guía

Convertir aplicaciones de PySpark/Scala a .NET ^(3/4)

Un DataFrame con un contexto de SQL:

```
df.createOrReplaceTempView("measures")
sqlDf = spark.sql("SELECT * FROM measures")

dataFrame.createOrReplaceTempView("measures");
Var sqlDataFrame = spark.Sql("SELECT * FROM measures");
```

Un DataFrame accesible para otra sesión de Spark via contexto SQL:

```
df.createGlobalTempView("measures")
sqlDf = spark.sql("SELECT * FROM measures").show()
spark.newSession().sql("SELECT * FROM global_temp.measures").show()

dataFrame.CreateGlobalTempView("measures");
spark.Sql("SELECT * FROM measures").show();
spark.NewSession().Sql("SELECT * FROM global_temp.measures").show();
```

Y con esta breve confrontación, podrás ir entendiendo como traducir desde Python a C#.



Una breve guía

Convertir aplicaciones de PySpark y Scala a .NET (4/4)

Con Scala, os he engañado un poco y si que veremos algo más de Scala, lo siento ;)

En Scala las columnas se referencian como:

```
dataFrame.Select($"ColumnName")
```

Scala dispone de Datasets, una característica que no existe en .NET y Python:

<https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html>

Cuidado con este punto ya que en Scala puede que el Dataset esté referenciando a las columnas por el nombre de la propiedad y no por el nombre de la columna y además implique uso de funciones Lambda:

```
case class Measure(  
  partitionkey: String,  
  temperature: Long  
)  
val measure = dataframe.as[Measure]  
measure.filter(m => m. temperature > 25)
```

Por lo demás es muy similar a Python, no tendrás mayor problema incluso en pasar de Python a Scala o viceversa.



UDFs/UDAFs

Las APIs (1/17)

UDFs/UDAFs – User-Defined Functions / User-Defined Aggregate Functions

Aunque yo no soy partidario de realizar este tipo de acciones, existen por un propósito: poder desarrollar cualquier cosa que no esté desarrollado en .NET pero que si exista en la JVM. Es decir, existe un desfase y ciertas acciones que no están soportadas por .NET pero que existen en Apache Spark y la única forma de poder trabajar con ellas o es usar scala o es crear tus propias UDFs/UDAFs.

Veamos un ejemplo:

Si quisiera crear mi propio agregador para sumar o contar sin usar el código nativo de Apache Spark, debo escribir yo mi propia UDAF. Donde debo operar en cada fila y devolverlo.

```
// Sample UDF
var spark = SparkSession
    .Builder().AppName("UDFSparkApp").GetOrCreate();
Func<Column, Column> udfIntToString = Udf<int, string>(
    str => $"Id = {str}");
var dataframe = spark.Sql("SELECT ID from range(1000)");
dataframe.Select(udfIntToString(dataframe["ID"])).Show();
```



UDFs/UDAFs

Las APIs (2/17)

Cuando veáis como funciona internamente esto, entenderéis por qué no soy partidario de estas acciones.

¿Cómo funciona esto?

Una UDF de .NET es similar a un controlador de .NET, la clase DotNetRunner inicia el código .NET y se abre un socket para enviar solicitudes via proxy (forward / backward). Si el anterior punto es complejo, esto usa también reflexión y para terminar debido a como se lanzan los procesos se pierde el estado compartido.

Para poder trabajar con ellos además debes crear una variable de entorno DONET_WORKER_PROCESS que apunte al contenido descargado de:

<https://github.com/dotnet/spark/releases>

Que otro aspecto entra en conflicto desde mi punto de vista: el rendimiento. Apache Spark, está diseñado para optimizar el rendimiento y pasar información entre procesos sobrecarga el rendimiento.

Por tanto y tal como he repetido antes, yo descarto trabajar así, pero es necesario que conozcáis que existe y que en algún momento dado no queda más remedio que atacar el problema por esta vía.

Los ejemplos los podrás encontrar en UDFandUDAFSpark.



UDFs/UDAFs

Las APIs (3/17)

Si no te queda más remedio que usar esta técnica, continúa leyendo esta parte teórica, si no, salta de esta sección... no suelo hacer esto en los workshops, quiero que se aprenda todo lo que aquí cuento, pero en esta ocasión, rompo la regla.

Existe un concepto el decapado o **pickling**, que podemos usar para llamar a una UDF. Se trata de definir una función usando tipos nativos y los llamamos directamente. Que ocurre con esta técnica, que con conjuntos de datos grandes es muy lento. Por tanto, ¿para que necesitamos Apache Spark si estoy volviendo lento el procesado?.

Otra técnica que nos ayuda a mejorar el rendimiento son los Apache Arrow, que no es nada más que una implementación para que los datos puedan ser compartidos entre procesos. Algo de mejora ganamos, pero volvemos a incidir en la esencia de Apache Spark.

Todo lo anterior añade más complejidad cuando trabajamos con UDAFs, ya no pasamos datos que no que estamos pasando operaciones de datasets agrupados al proceso, para que como resultado solamente tengamos un grupo, el que hemos definido. Aun más complejidad y más bajada de rendimiento.

Reafirmo: si podemos completar nuestros procesos sin tener que meter esa especie de proxy, siempre aprovecharemos la rapidez y la simplicidad de Apache Spark. Para mi el rendimiento es obligatorio cuando usamos Apache Spark, en otro caso, usaríamos otras herramientas para procesar datos.



DataFrame – Principalmente los desarrolladores de .NET usamos esta

Nativamente Apache Spark tiene 2 APIs diferentes: Resilient Distributed Dataset (RDD) y DataFrame.

Usaremos DataFrame ya que nos da todo lo que necesitamos y más que nada a que RDD no está disponible en .NET.

RDD es una abstracción de lo que podría ser archivos de datos masivos al dividir los archivos y procesarlos en distintos nodos de computo. Cuando apareció Apache Spark, era la única opción disponible.

DataFrame es una abstracción de nivel superior y se basa en columnas de datos distribuidas sobre RDD. El objeto columna incluye muchos métodos que podemos usar para escribir procesamiento de datos de forma eficiente.

Ahora entiendes por que Microsoft .Spark no dispone de RDD. Solamente trabajamos sobre un parte que, aunque no este deprecada, supone un trabajo por parte de los ingenieros que al final no tendrá nicho de mercado, no vamos a trabajar sobre algo que en teoría es obsoleto.

Antes de adéntranos en los DataFrames y profundizar en el, debes comprender la diferencia entre una **acción** y una **transformación**, dos conceptos básicos de los DataFrames.

Una transformación es algo que potencialmente se aplica sobre un DataFrame y una acción se aplica sobre todas las transformaciones de un DataFrame.



DataFrameReader – Lectura de ficheros y orígenes de datos

Una transformación es algo que potencialmente se aplica sobre un DataFrame y una acción se aplica sobre todas las transformaciones de un DataFrame.

Por ejemplo, leemos un csv:

```
var spark = SparkSession.Builder().GetOrCreate();
DataFrameReader reader = spark.Read().Format("csv").Option("header", true).Option("sep", ",");
var dataframe = reader.Load("file.csv");
dataframe.Show();
```

Tenemos dos formas de leer:

- Load: `spark.Read().CSV("file.csv")`
- Format: `Spark.Read().Format("csv").Load("file.csv")`

¿Cuándo usar una u otra? Pues dependerá del programa que estés realizando, el resultado en ambas es el mismo.

¿Qué formatos admite? De forma nativa Text, JSON, Parquet, ORC y JDBC.

¿Admite opciones? Por supuesto, un CSV puede estar separado por “,” o por otro carácter, en las opciones puedes configurar cosas como estas, leer la cabecera, etc.



Schema – Esquemas

Algunos formatos por parquet o avro (par leer XLSX) incluye un esquema de los metadatos, además de los datos. Otros como JSON o CSV no lo incluyen.

Apache Spark, puede inferir el esquema o no, con la opción `inferSchema`.

En el caso de los JSON siempre intentará inferir el esquema a menos que nosotros se lo especifiquemos de forma manual.

Las dos formas de pasar un esquema es mediante:

- Con DDL, como los de las bases de datos tradicionales.
- Con StructType, la definición del esquema.

En los siguientes ejemplos podemos ver la diferencia entre uno y otro:

```
var sparkSchemaDDL = SparkSession.Builder().GetOrCreate();
var dataframeSchemaDDL = sparkSchemaDDL.Read().Option("sep", ","). Option("header", "false")
    .Schema("PartitionKey string, RowKey string")
    .CSV("file.csv");
dataframeSchemaDDL.PrintSchema();
```

```
var sparkSchemaST = SparkSession.Builder().GetOrCreate();
var schema = new StructType(new List<StructField>() {
    new StructField("ParitionKey", new StringType()),
    new StructField("RowKey", new StringType())
});
```



CreateDataFrames – Creación de DataFrames desde código

No es nada habitual querer crear DataFrames al vuelo, pero existe la posibilidad. Lo normal es usar Apache Spark para procesar datos desde fuentes externas y no desde el programa internamente.

La única ocasión en las que yo los uso es para hacer demostraciones de ejemplo.

Podemos crearlos de varias formas con CreateDataFrame o SQL:

0 references

```
static void Main(string[] args)
{
    var spark = SparkSession.Builder().GetOrCreate();
    spark.CreateDataFrame(new [] {"1", "2", "3"}).Show();

    var schema = new StructType(new List<StructField>() {
        new StructField("PartitionKey", new StringType()),
        new StructField("RowKey", new StringType())
    });
    IEnumerable<GenericRow> rows = new List<GenericRow>() {
        new GenericRow(new object[] {"1", 2, 3})
    };
    spark.CreateDataFrame(rows, schema).Show();
}
```

```
var spark = SparkSession.Builder().GetOrCreate();
spark.Sql("SELECT 'Hello' as ValueOne, 'World!' as ValueTwo union SELECT 'Hi', 'All']").Show();
```



DataFramesWriter – Creamos los datos de salida

El similar a DataFrmeReader, nos sirve para escribir en un fichero o en un origen de datos.

Al igual que en cualquier lenguaje de programación, escribir en ficheros tiene uno modos estándar: `overwrite`, `ignore`, `append`, ... Explora las propiedades de este método para conocer las opciones disponibles.

```
var spark = SparkSession.Builder().GetOrCreate();  
var dataframe = spark.Range(100);  
dataframe.Write().Mode("overwrite").Csv("output.csv");
```



PartitionBy – Creamos particiones

Una de las características más potentes de Apache Spark es la velocidad y optimización que tiene para particionar ficheros o fuentes de datos.

Supongamos que tenemos el fichero de measures.csv generado por nuestras Raspberry Pi, con el ID del dispositivo que está generando la información y queremos generar de ese TB de fichero vario ficheros particionados por el ID. Cualquier sistema que conozcas de gestión de ficheros, no llega ni a la mitad de la velocidad que tiene Apache Spark para separa los datos de forma particionada o incluso para particionarlos en memoria y sacar la información agregada por dispositivo.

```
dataFrame.Write().PartitionBy("PartitionKey").Csv("output.csv");
```

Esto nos lleva a comentar que debes controlar el naming de los ficheros de carga el siguiente ejemplo, APISReadandFilter, cuando lo ejecutes (descomprime el fichero CSV), ejecútalo, verás que es casi instantáneo la partición realizada del fichero principal. Este luego se procesa por ejemplo en otro punto de tu ETL o ELT (explicado al final del workshop).

```
jmfloreszazo@ubuntu:~/mnt/c/temp/APISReadandFilter$ tree output.csv
output.csv
├── PartitionKey=1
│   ├── part-00000-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   ├── part-00001-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   ├── part-00002-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   └── part-00003-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
├── PartitionKey=2
│   ├── part-00003-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   ├── part-00004-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   ├── part-00005-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   ├── part-00006-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
│   └── part-00007-3985a6c5-d790-4625-8ea7-002356e7fc83.c000.csv
└── _SUCCESS
2 directories, 10 files
```



DataFrame

Las APIs (10/17)

Piensa que la relación del fichero son 100MB a 500K registros y la ejecución es casi instantánea, tardamos más en cargar Spark y hacer un run, que en ver el resultado final.

El anterior ejemplo:

```
var spark = SparkSession
    .Builder().AppName("APIsReadandFilterApp").GetOrCreate();
var reader = spark.Read().Format("csv").Option("header", true).Option("sep", ",");
var dataframe = reader.Load(path: "monitoringoffice.csv");
dataframe.Write().PartitionBy("PartitionKey").Csv(path: "output.csv");
```

Realiza la siguiente partición: ¿pensabas que saldría un fichero output.csv? Como has observado es una salida de un directorio y con diferente subcarpeta, el naming en esta ocasión no es muy acertado, pero se entiende de que a que punto van datos de la partición 1 y 2.

Como desarrollador de .NET podrás pensar que, muy bien, lo puedo escribir yo haciendo un Nuget y compartirlo en la comunidad o usar uno y existente, o exportarlo a SQL Server y luego lanzar consultas para exportar las particiones a CSV o cargarlo en memoria, que se yo, muchas soluciones. Pero ninguna tan optimizada como la anterior, si no, observa cuando hilos a creado o pensabas ¿que con las anteriores aproximaciones sin meter hilos llegarías a acercarte al rendimiento de Spark?

CALL STACK	RUNNING
Thread #19692	RUNNING
Thread #32224	RUNNING
Thread #21120	RUNNING
Thread #32244	RUNNING
Thread #25712	RUNNING
Thread #21724	RUNNING
Thread #34252	RUNNING
Thread #30532	RUNNING



Columnas y Funciones – Ultimo punto importante a revisar

Lo que hace que el DataFrame API sea tan sencillo de usar es el uso de Columnas, si lo comparas con una operación de map o reduce de RDD API. Ya que será quien soporte los procesos.

Column es un miembro estático de Microsoft.Spark.Sql.Functions.Column que soporta un alias llamado Col, puedes usarlo indistintamente. Más información aquí:

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.spark.sql.column?view=spark-dotnet>

```
using static Microsoft.Spark. Sql.Functions;
using Microsoft.Spark.Sql;
using System;

namespace 
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {

            dataframe.Select(Column("ID")).Show();
            dataframe.Select(Col("ID")).Show();

            dataframe.Select(Column("ID").Name("Not ID")).Show();
            dataframe.Select(Col("ID").Name("Not ID")).Show();

            dataframe.Filter(Column("ID").Gt(100)).Show();
            dataframe.Filter(Col("ID").Gt(100)).Show();
        }
    }
}
```



Historia – Un poco de historia nunca viene mal

La API de SQL nos permitirá escribir consultas ANSI SQL:2003, el estándar para SQL. Esto quiere decir que podemos almacenar nuestros datos en ficheros, es decir, que podremos almacenarlos con casi total seguridad en cualquier data lake.

Antes de Apache Spark, Facebook creo **Apache Hive**, una forma de ejecutar consultas SQL sobre Hadoop (incluod HDFS). Apache Hive es un componente de un *metastore*, es decir, un conjunto de datos sobre archivos que permite a los desarrolladores leerlos como si fueran tablas de una base de datos y con un motor propio de consultas SQL tipo map/reducer contra estos archivos.

Cuando Apache Spark apareció en escena tenía el API RDD que no era compatible con SQL, pero en la version 2.0 se incluyo este analizador y la conectividad a Apache Hive. Esto quiere decir que ahora podemos usar el motor de Spark de SQL para atacar a los metadatos de Hive.

Toda esta historia que os cuento es para que situéis cada pieza del ecosistema de Big Data y poco a poco podías ir viendo la relación que existe con Azure HDInsight.



Veamos un ejemplo donde usamos SQL y contextos:

```
var spark = SparkSession
    .Builder()
    .Config("spark.sql.legacy.allowCreatingManagedTableUsingNonemptyLocation", "true")
    .GetOrCreate();
var dataframe = spark.CreateDataFrame(new [] {1, 2, 3, 4, 5})
    .WithColumnRenamed("_1", "ID");
dataframe.Write()
    .Mode("overwrite")
    .SaveAsTable("my_table");
spark.Sql("select * from my_table").Show();
```

Como podréis observar la primera parte que tenemos es un DataFrame que escribe una secuencia numérica en un output, que es un fichero. Y la segunda parte es la lectura de datos de un contexto anterior de my_table.

Como podréis observar no es algo muy complejo. La salida que origina es una tabla en pantalla con ID como cabecera y la secuencia 5, 4, 3, 2, 1.



Unos puntos importantes que tenemos que tener en cuenta con SQL es:

- Que podemos crear tablas temporales con `CreateTempView`.
- Que podemos crear o modificar tablas temporales con `CreateOrReplaceTempView`.
- Que podemos crear vistas DataFrame disponibles para toda la sesión de Spark con `CreateGlobalTempView` y por supuesto podemos proceder con `CreateOrReplaceGlobalTempView`.
- Que mientras que `DataFrameWriter.SaveAsTable` crea ficheros en Hive, las vistas se crean el paso intermedio de los ficheros.
- Que en termino de workspaces de DataBricks la combinación de unas u otras formas de crear vistas afectan a los Jobs de diferentes usuarios debido a las sesiones de Spark. Como recomendación cuando uses vistas globales y temporales, es que pongas un prefijo al nombre de la tabla "global_temp_view", "temp_view", esta nomenclatura es la que solemos adoptar de factor los desarrolladores de Spark.

Otro punto importante y he mencionado antes de pasada son las sesiones, el llamado `SparkSession Catalog`. Un objeto que nos permite inspecciona y modificar los metadatos almacenados a nivel de metastore de Hive, por ejemplo:

```
var spark = SparkSession.Builder().GetOrCreate();
spark.Sql("CREATE DATABASE SampleData");
spark.Catalog.SetCurrentDatabase("SampleData");
spark.Catalog.CreateTable("id_list", "ID");
var tables = spark.Catalog.ListTables("SampleData");
foreach (var row in tables.Collect()) {
    //TODO
}
```



Y lógicamente con el catalogo podemos hacer diversas acciones:

- GeDataBase.
- GetFunctions.
- GetTable

Conocer estas acciones nos permitirá ir poco a poco a juntar piezas del puzzle que nos permitirán hacer programas de verdad.

En resumen, Apache Spark SQL es un parser de SQL con una serie de elementos muy potentes que nos permiten acercarnos a la solución de problemas reales.

Como debes profundizar más aquí te dejo tarea:

<https://spark.apache.org/docs/latest/api/sql/index.html>



ML – Machine Learning

Y aquí es cuando te dije, que puedes ver mi otro workshop de ML.NET para completar un poco el binomio Big Data y Machine Learning:

<http://jmfloreszazo.com/machine-learning-para-desarrolladores-de-net/>

La API ML no forma parte del Core de Spark (lo mismo que el punto siguiente), pero que es necesario que puedas comprender que esta pieza se volverá cada vez más compleja, completa y necesario tal y como evoluciona el mercado.

Existen 2 enfoques para atacar el trabajo con ML.

- Usando UDFs y ML.NET
- Usando Spark para lo que es, para la parte de ETL que corresponde y luego usar ML.NET sobre el resultado de los datos que hemos obtenido.

Si te interesa saber más aquí os dejo información:

<https://spark.apache.org/mllib/>



Grafos – Machine Learning

A colación también puedo deciros que podéis ver mi intervención sobre grafos:

<http://jmfloreszazo.com/grafos-la-tercera-via-para-nuestros-datos/>

Ya que Spark también nos permite trabajar con Grafos:

<https://spark.apache.org/graphx/>

No vamos a entrar en este punto como no lo paso con ML, ya que se escapa a este workshop, pero si os diré, que juntando tanto ML como Grafos en un solo motor como es Spark, podemos ir aplicando el ETLT (leer ultima sección de Bonus).

Como podéis observar existen piezas y elementos que se entremezclan, es importante conocerlos para completar el ciclo de ETLT.

Hasta aquí las APIs de Apache Spark, toca que por tu parte profundices en los puntos que mas lagunas tengas o donde no he podido dejarlo del todo claro.



Procesado de datos Batch

Casos de Uso ^(1/2)

Batch – Lotes

Un procesamiento de datos típico leer los datos del origen y los analizar para escribirlos en un formato que el consumidor pueda usar.

Los datos de origen en muchos casos, en todos me atrevería decir, son datos imperfectos: faltan campos, llevan nulos, el formateo no es correcto, unimos 2 fuentes donde las unidades en uno se representan en miles y otras en unidades, etc. Etc. Por tanto, lo primero que tenemos que hacer es normalizar los datos.

El segundo paso es una vez que las fuentes de datos han sido normalizadas, es guardar todo en un formato común, si por ejemplo unificamos las temperaturas de la oficina de USA y la España, una irá en grados Fahrenheit y la otra en Celsius, ambos con los datos normalizados, decidimos guardarlos en formato Celsius.

El tercer paso, es que nos piden calcular la media mensual, pues con los datos normalizados y estandarizados, es coger una consulta de agrupación para sacar las medias de esa temperatura por oficinas, por mes y con el valor de temperatura, que guardamos en el destino de datos para que una aplicación como puede ser PowerBI muestre un panel al usuario de destino. O bien lo granularizamos en 3 tablas, diaria, semanal y mensual para que ese software no sufra sobre carga en memoria a la hora de mostrar los datos.

Esta sería la forma de procesar lotes, que no es mas que leer ficheros, normalizarlos y generar un resultado unificado para el análisis de datos que nos pidan.



Procesado de datos Streaming

Casos de Uso (2/2)

Streaming – Transmision

En vez de trabajar con conjuntos de datos estáticos como antes, lo que hacemos es trabajar con micro lotes de datos utilizando un procesamiento de flujo escalable y torearte a fallos construido con Kafka.

La aplicación lo que hace es, examinar el mensaje para detectar por ejemplo una anomalía y otra acción que será recopilar y juntar los datos de cada 5 minutos en una base de datos agregada y luego podemos usar PowerBI.

Por ejemplo, puedes seguir los pasos del siguiente ejemplo que nos proporciona Microsoft:

<https://docs.microsoft.com/es-es/azure/event-hubs/apache-kafka-developer-guide>

No tiene sentido que desarrolle uno en particular cuando este ejemplo es muy completo.



Usando el Logging

Cuando estamos en un lio ^(1/3)

log4j.properties – Modifica el nivel de información del log de Spark

Puedes modificar este fichero para establecer un nivel de información que nos ayude a obtener algo de luz: OFF, FATAL, ERROR, WARN, ...

Otra opción es a nivel de SparkContext o SparkSession, podemos controlar este valor desde aquí.

En:

<https://logging.apache.org/log4j/2.x/index.html>

Puedes revisar el funcionamiento de la pieza que va adjunta en Apache Spark.

Supongo que como desarrollador de .NET conocerás:

<https://logging.apache.org/log4net/>

Si es así, algo de ventajas tendrás al poder revisar la información y conocerás la herramienta Chainsaw que puedes obtener desde las direcciones anteriores.



Spark UI

Cuando estamos en un lio (2/3)

UI – Interface de Usuario

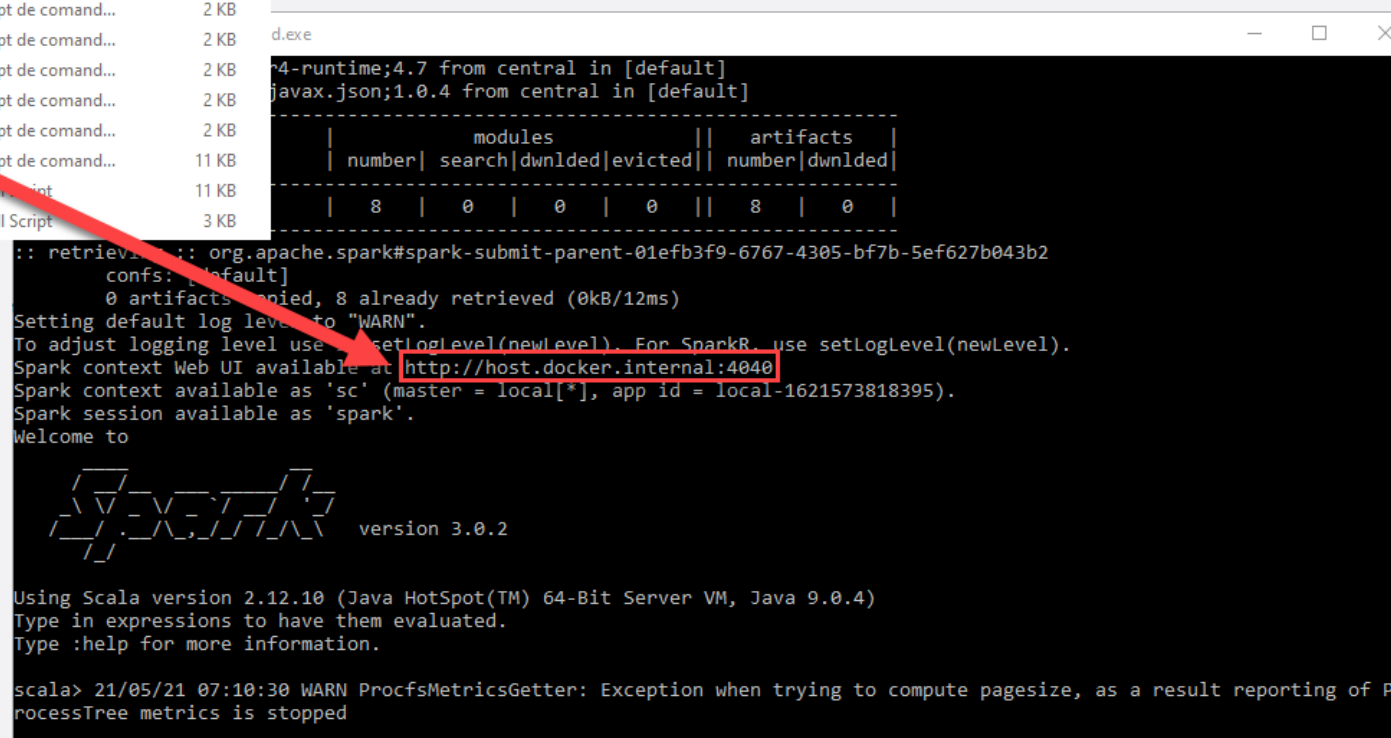
Cuando cargas para trabajar Apache Spark, puedes entrar en <http://localhost:4040> y entrará en la parte visual del servicio, en ella tienes la posibilidad de ver historiales, ...

Para ello establece el log a INFO y un par de directrices en `spark-defaults.conf`. Luego ejecutas `spark-shell.cmd` de tu carpeta de instalación:

```
18 # Set everything to be logged to the console
19 log4j.rootCategory=INFO, console
20 log4j.appender.console=org.apache.log4j.ConsoleAppender
21 log4j.appender.console.target=System.err
```

```
29 spark.executor.memory=6g
30 spark.jars.packages=io.delta:delta-core_2.12:0.7.0
31 spark.eventLog.enabled true
32 spark.eventLog.dir /tmp/spark-history-logs
```

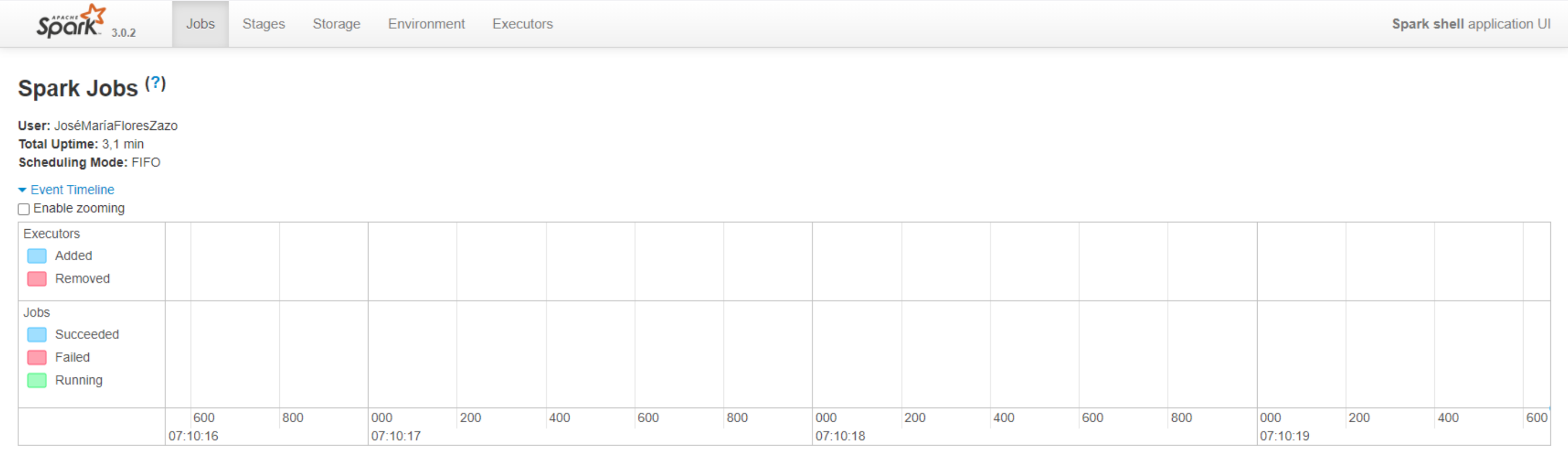
spark-class2.cmd	16/02/2021 6:21	Script de comand...	3 KB
sparkR.cmd	16/02/2021 6:21	Script de comand...	2 KB
sparkR2.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-shell.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-shell2.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-sql.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-sql2.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-submit.cmd	16/02/2021 6:21	Script de comand...	2 KB
spark-submit2.cmd	16/02/2021 6:21	Script de comand...	2 KB
yarn.cmd	14/05/2021 7:20	Script de comand...	11 KB
docker-image-tool.sh	16/02/2021 6:21	Shell Script	11 KB
load-spark-env.sh	16/02/2021 6:21	Shell Script	3 KB



Spark UI

Cuando estamos en un lio (3/3)

Y podrás ver una serie de pestañas que podrás ir explorando para ver que está ocurriendo con tu proyecto:



| ¿Qué es? Delta Lake ^(1/2)

Extensión – <https://delta.io/>

Aunque Mllib y GrpahX son extensiones, esta merece se tratada en un punto a parte. Ya que esta creada por la compaía Databricks (<https://databricks.com/>, creadora de Apache Spark) y como proyecto OS.

Tiene por objetivo hacer una escritura en data lakes empresariales de forma eficiente, sin importar el data lake que utilices: Azure Data Lake Storages, AWS S3 o Hadoop.

Posee propiedades ACID (<https://es.wikipedia.org/wiki/ACID>) de una base de datos relacional, como puede ser SQL Server o MySQL.

Viene a resolver varios problemas, por ejemplo:

- ¿Qué sucede si Apache Spark esta escribiendo en un directorio y a la mitad falla dejando el proceso incompleto?, ¿Qué deben hacer los reader?, ¿Saben que son incompletos? O incluso si se dan cuenta que están incompletos, ¿Cómo vuelvo a recuperar los datos sobrescritos?.
- Leer datos de un directorio sin son Big Data, es costoso y puede ralentizar otros trabajos.

En resumen, debes controlar muchos de los típicos problemas de ficheros. Pero para eso nacieron las las bases de datos relacionales y para eso han creado Delta Lake, para poder gestionar estos problemas con las típicas instrucciones de insertar, actualizar, eliminar, etc.



¿Qué es?

Delta Lake (2/2)

Pero va más allá de lo que entendemos que hace una BBDD relaciona, es capaz incluso de controlar concurrencias manteniendo datos intactos controlando versiones, lo que se conoce como: MVCC (**Multi-Version-Concurrency-Control**).

Para poder usarlo debemos cambiar el fichero de configuración y añadir:

```
spark.jars.packages io.delta:delta-core_2.12:0.7.0
```

Crear una sesión de Spark en .NET especificando que queremos usarlo:

```
var spark = SparkSession.Builder()  
    .Config("spark.sql.extensions", "io.delta.sql. DeltaSparkSessionExtension")  
    .GetOrCreate();
```

Y comenzar a convertir nuestros datos a DeltaTables:

```
DeltaTable.ConvertToDelta
```

Puedes ver un buen ejemplo en:

<https://docs.microsoft.com/es-es/azure/synapse-analytics/spark/apache-spark-delta-lake-overview?pivots=programming-language-csharp>



| ¿Qué es? Azure Data Lake

No es más que un repositorio de datos – De varias fuentes y sin procesar

Que nos permite almacenar y analizar archivos de petabytes y billones de entidades.

Que nos hace la vida más fácil para ejecutar programa en paralelo que procesan datos masivos de forma sencilla.

Y que cumple con todas las expectativas de seguridad, no obstante, estamos trabajando con datos y aquí jugamos en un partido donde entra en conflicto con muchas leyes gubernamentales. Desde mi punto de vista es preferible pagar por algo que ya lo cumple a tener que montarte todo en on-premise si no sabes que puede incurrir en un delito: puede que salga más cara la nube a priori, pero piensa que todo lo que existe detrás al final es un ahorro ya que, si no, lo debes invertir tu.

Azure Data Lake se compone de:

- Azure Data Lake Analytics, no dejar de ser un servicio de análisis usando U-SQL, R, Python o .NET, en modo SaaS y que permite el pago por uso.
- HDInsight: servicio de Apache Spark y Hadoop.
- Data Lake Store: un repositorio de datos en teoría sin límites.



¿Qué es Azure Databrick?

Moviéndonos a Azure (1/7)

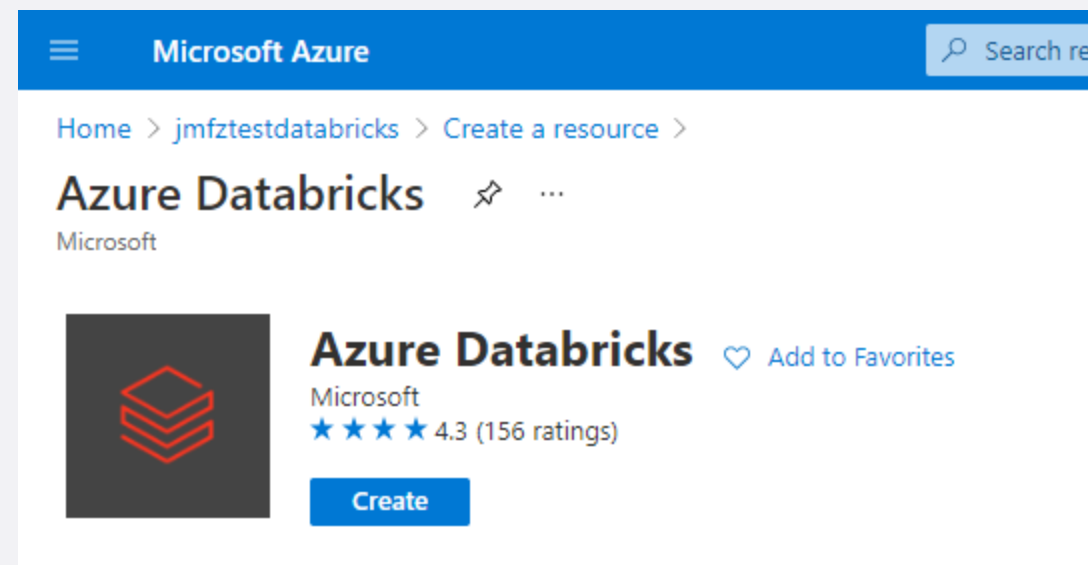
Databrick – <https://docs.microsoft.com/es-es/azure/databricks/>

Se trata de la plataforma de análisis de Microsoft en Azure. Existen dos entornos para desarrollar aplicaciones:

- Azure Databricks SQL Analytics
- Azure Databricks Workspace.

Nosotros vamos a usar los workspaces ya que es el nexo con todo lo anteriormente aprendido, ya que permite la lectura de datos desde distintas fuentes Kafka, IoT Hub, ... y convertirlos en información con Spark.

Nos ponemos manos a la obra para que nuestro HelloWorldSpark se mueva a la nube.



¿Qué es Azure Databricks?

Moviéndonos a Azure (2/7)

Le damos las características al recurso:

Create an Azure Databricks workspace ...

Basics

Networking

Advanced

Tags

Review + create

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Visual Studio Enterprise – MPN

Resource group * ⓘ

jmfztestdatabricks

Create new

Instance Details

Workspace name *

jmfztestdatabricks

Region *

West Europe

Pricing Tier * ⓘ

Trial (Premium - 14-Days Free DBUs)

Microsoft Azure

Search resources, services, and docs (G+/)

ifztestdatabricks_jmfztestdatabricks >

jmfztestdatabricks

Databricks Service

Ctrl+/) <<

Delete

Essentials

Status : Active

Resource group : jmfztestdatabricks

Location : West Europe

Subscription : Visual Studio Enterprise – MPN

Subscription ID :

Tags (change) : Click here to add tags

Managed Resource Group :

URL :

Pricing Tier : Trial (Premium - 14-Days Free DBUs)

JSON View

Launch Workspace

Upgrade to Premium

Documentation

Getting Started

Import Data from File

Import Data from Azure Storage

Notebook

Admin Guide

Link Azure ML workspace



¿Qué es Azure Databrick?

Moviéndonos a Azure ^(3/7)

Necesitamos instalar las siguientes herramientas:

- Python 3, si quieres ver si lo tienes instalado, ejecuta `python3 --version`, si no ([ve a este enlace](#))
- Instala los componentes: `pip3 install Databricks-cli`
- Comprobar que tenemos instalado el modulo: `databricks --version`

Ahora toca configurar la autenticación:

1. `databricks configure --token`
2. Busca tu host en Azure, en mi caso: <https://adb-8175482530995299.19.azuredatabricks.net>, lo necesitas para el paso anterior.
3. Entra en el portal (ver imágenes anteriores) y pulsa sobre “Launch Workspace”.
4. Una vez dentro, ir al usuario >> settings y generate token, que te piden en el paso 1.

Microsoft Azure | Databricks

Portal jose.flores@tokiota.com

Free trial ends in 14 days. [Upgrade to Premium in Azu](#)

1

2

3

User Settings

Access Tokens Git Integration Notebook Settings Model Registry Settings

Personal access tokens can be used for secure authentication to the [Databricks API](#) instead of passwords.

Generate New Token

Comment	Creation ↑	Expiration
localhost	2021-05-21 09:03:53 CEST	2021-08-19 09:03:53 CEST

Signed in as

User Settings

Admin Console

Manage Account

Log Out

Workspaces

✓ jmfztestdatabricks



¿Qué es Azure Databrick?

Moviéndonos a Azure (4/7)

Cargamos nuestro proyecto en VS Code y ejecutamos en un terminal:

```
dotnet publish -c Release -f net5.0 -r ubuntu.16.04-x64
```

```
PS C:\temp\HelloWorldSpark> dotnet publish -c Release -f net5.0 -r ubuntu.16.04-x64
Microsoft (R) Build Engine versión 16.9.0+57a23d249 para .NET
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Determinando los proyectos que se van a restaurar...
Todos los proyectos están actualizados para la restauración.
HelloWorldSpark -> C:\temp\HelloWorldSpark\bin\Release\net5.0\ubuntu.16.04-x64\HelloWorldSpark.dll
HelloWorldSpark -> C:\temp\HelloWorldSpark\bin\Release\net5.0\ubuntu.16.04-x64\publish\
PS C:\temp\HelloWorldSpark> █
```

Ahora debemos generar un ZIP de la carpeta:

```
C:\temp\HelloWorldSpark\bin\Release\net5.0\ubuntu.16.04-x64
```

Con el nombre: publish.zip



¿Qué es Azure Databrick?

Moviéndonos a Azure (5/7)

Debemos cargar los archivos necesarios:

```
databricks fs cp publish.zip dbfs:/spark-dotnet/publish.zip  
databricks fs cp [YOUR_PATH]microsoft-spark-3-0_2.12-1.1.1.jar dbfs:/spark-dotnet/microsoft-spark-3-0_2.12-1.1.1.jar
```

Desplegamos a través de spark-submit:

```
[\"--class\",  
\"org.apache.spark.deploy.dotnet.DotnetRunner\",  
\"/dbfs/spark-dotnet/microsoft-spark-3-0_2.12-1.1.1.jar\",  
\"/dbfs/spark-dotnet/publish.zip\",  
\"HelloWorldSparkApp\"]
```



¿Qué es Azure Databrick?

Moviéndonos a Azure (6/7)

Ejecutamos la aplicación:

[Jobs](#) / MyJob NEW

Free trial ends in 14 days

MyJob

[Run Now](#) ▼

[More](#) ...

[Runs](#) [Configuration](#)

ID: 5

Creator:

Schedule: None

ativ Run

A partir de aquí ya puedes ir a ver el resultado con las distintas opciones que dispone al entrar en la ejecución:

Completed Runs (past 60 days)

[Latest Successful Run \(Refreshes Automatically\)](#)

[Refresh](#)

Run	Start Time	Launched	Duration	Spark	Status
View Details		Manually	5s	Spark UI / Logs / Metrics	

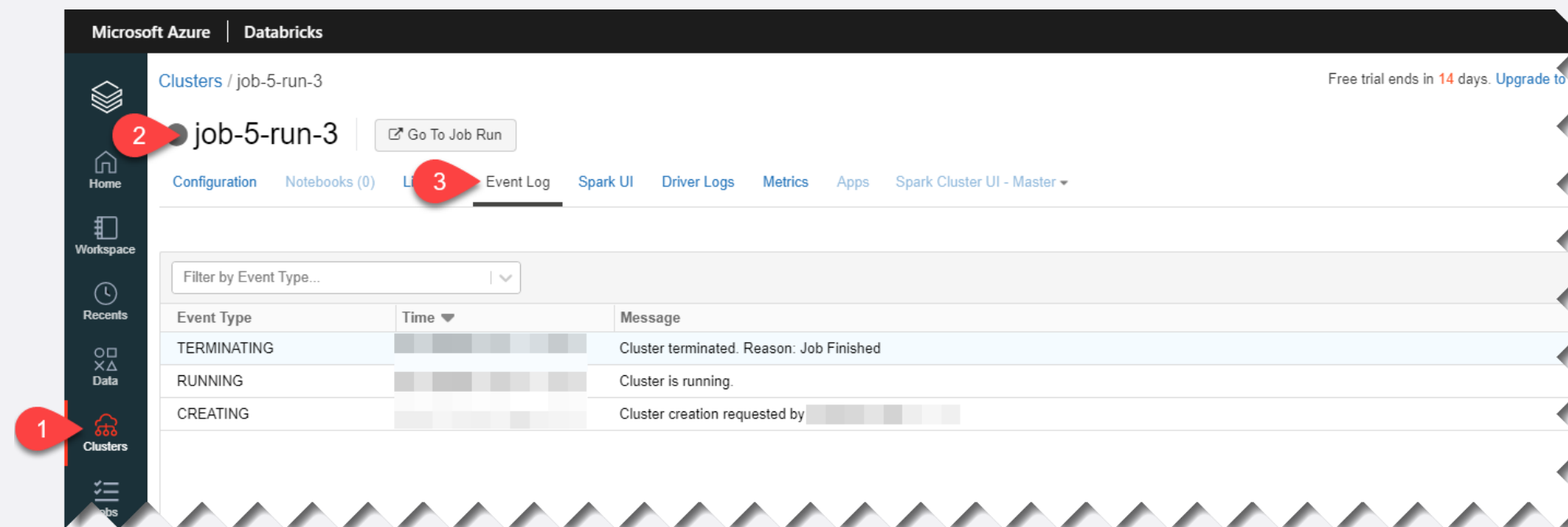
1 - 1 < > 20 / Page



¿Qué es Azure Databrick?

Moviéndonos a Azure (7/7)

Si tenemos problemas por versiones de Spark o ver por que no arranca o no saca la salida de datos un Job, podéis entrar en el cluster e investigar:



Muchas veces es problema de las versiones de Spark que usamos en .Net y la que despliega el cluster, en resumen, el JAR que subimos antes en el proceso y la version que generamos con el cluster. Una version 2 en el cluster y una version 3 en el código.

Aquí un buen ejemplo de Microsoft:

<https://docs.microsoft.com/es-es/dotnet/spark/tutorials/databricks-deployment>



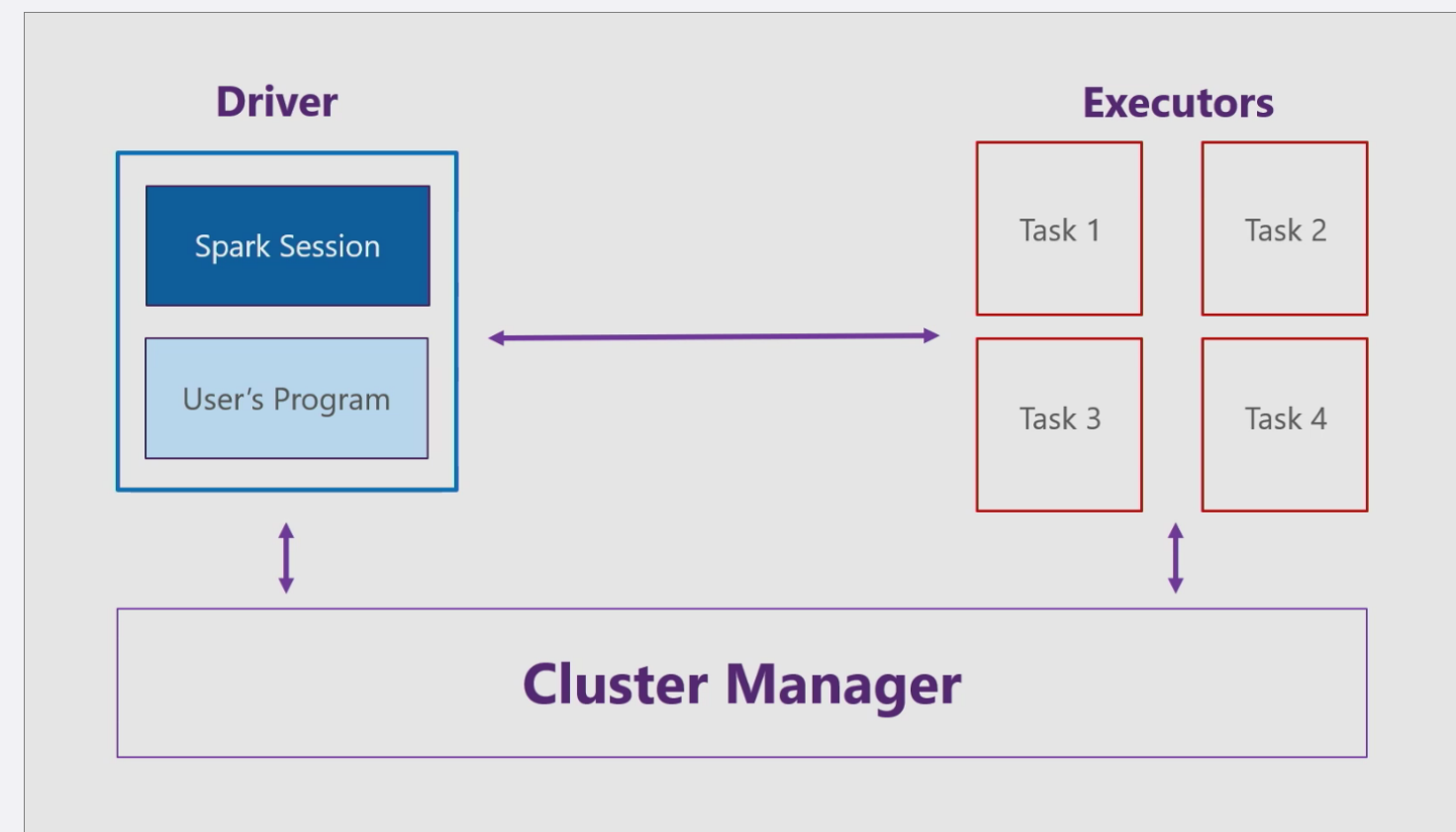
Arquitectura de Apache Spark

Bonus (1/2)

Master/Worker Architecture – Arquitectura Maestro/Trabajador

La traducción es un poco libre, pero creo que entendéis que el concepto principal es: un jefe y un obrero.

Posee tres componentes principales: el controlador, los ejecutores y los administradores del cluster. Como esto esta muy bien explicado en la web de Microsoft, solo os pongo el enlace y la imagen de la vista a mayor nivel:



<https://docs.microsoft.com/es-es/dotnet/spark/what-is-spark>



¿Qué ETL y ELT?

Bonus (2/2)

Extract, Transform, Load – Extraer, transformar y cargar

Concepto muy relacionado hoy en día con el Big Data (de lo que estamos tratando en este workshop). Ambos procesos pueden sonar similares, ambos se encargan de mover grandes volúmenes de datos, integrarlos e ingestarlos en un lugar común para que esté accesibles con un formato adecuado.

La diferencia se encuentra en el orden de los procesos, cada modelo, evidentemente, se comporta mejor para resolver un problema.

ETL, extrae datos de una o varias fuentes, por ejemplo, BBDD relacionales, lo transforman con agregaciones, normalizaciones, cambios de tipos, cruzando datos, ... y por último carga los datos preparados (en el argot estadístico: cocinados) para el almacenamiento final, por ejemplo, un Data Warehouse.

- Se comporta mejor con datos estructurados.
- La fuente y destino de datos suelen ser tecnológicas diferentes.
- Las transformaciones son intensivas a nivel de cómputo.
- Es fácil implementar procesos de calidad del dato.

ELT, en este caso transformamos los datos una vez cargados en la base de datos de destino sin realizar un procesamiento previo. Es decir, la carga de datos es el paso intermedio de este método. Las transformaciones generalmente se realizan sobre clusters de Hadoop y BBDD NoSQL.

- Casos de datos no estructurados.
- La fuente y destino del dato suele ser la misma tecnología, un MongoDB, por poner un ejemplo.
- Los volúmenes de datos son muy grandes, pero computables por el motor de la BBDD.
- Rápidos en ingesta grandes volúmenes de datos no estructurados.

No pienses que es un ETL vs ELT, en muchas ocasiones en un ETLT = ETL + ELT. Este modelo híbrido vuelve a cocinar los datos, en muchas ocasiones necesitas datos con más o menos granularidad, por poner un ejemplo.



¡Gracias!

Puedes encontrarme buscando por **jmfloreszazo** en



<https://jmfloreszazo.com>