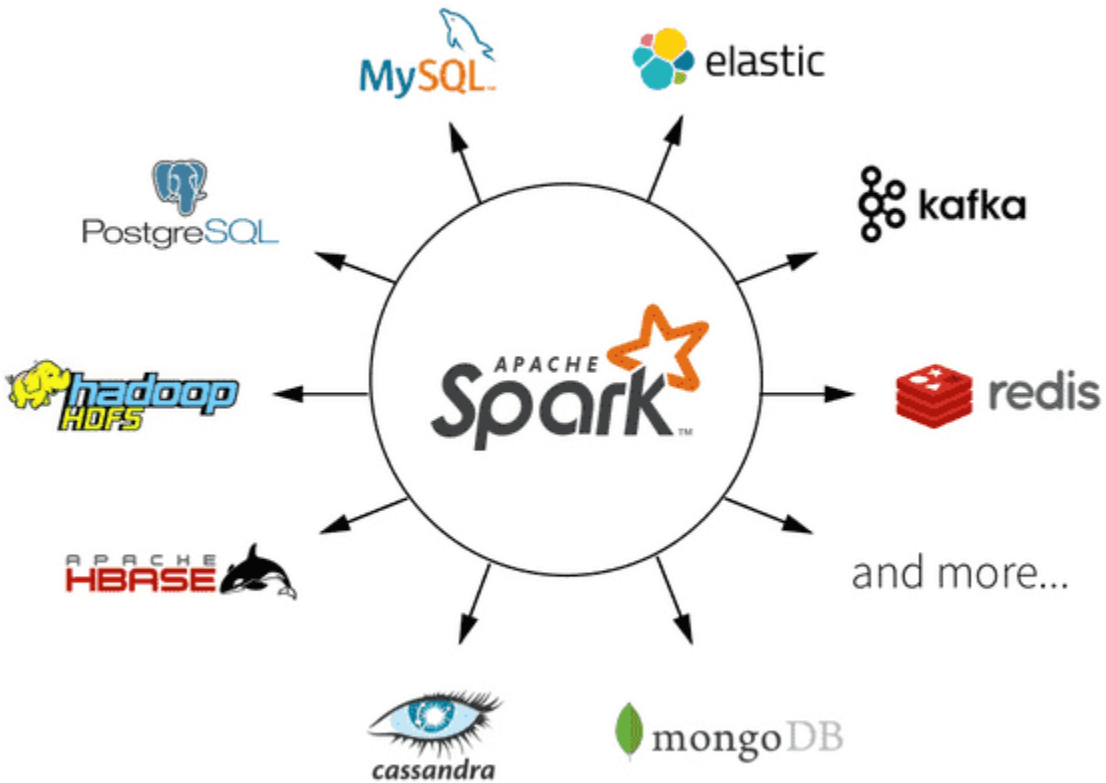
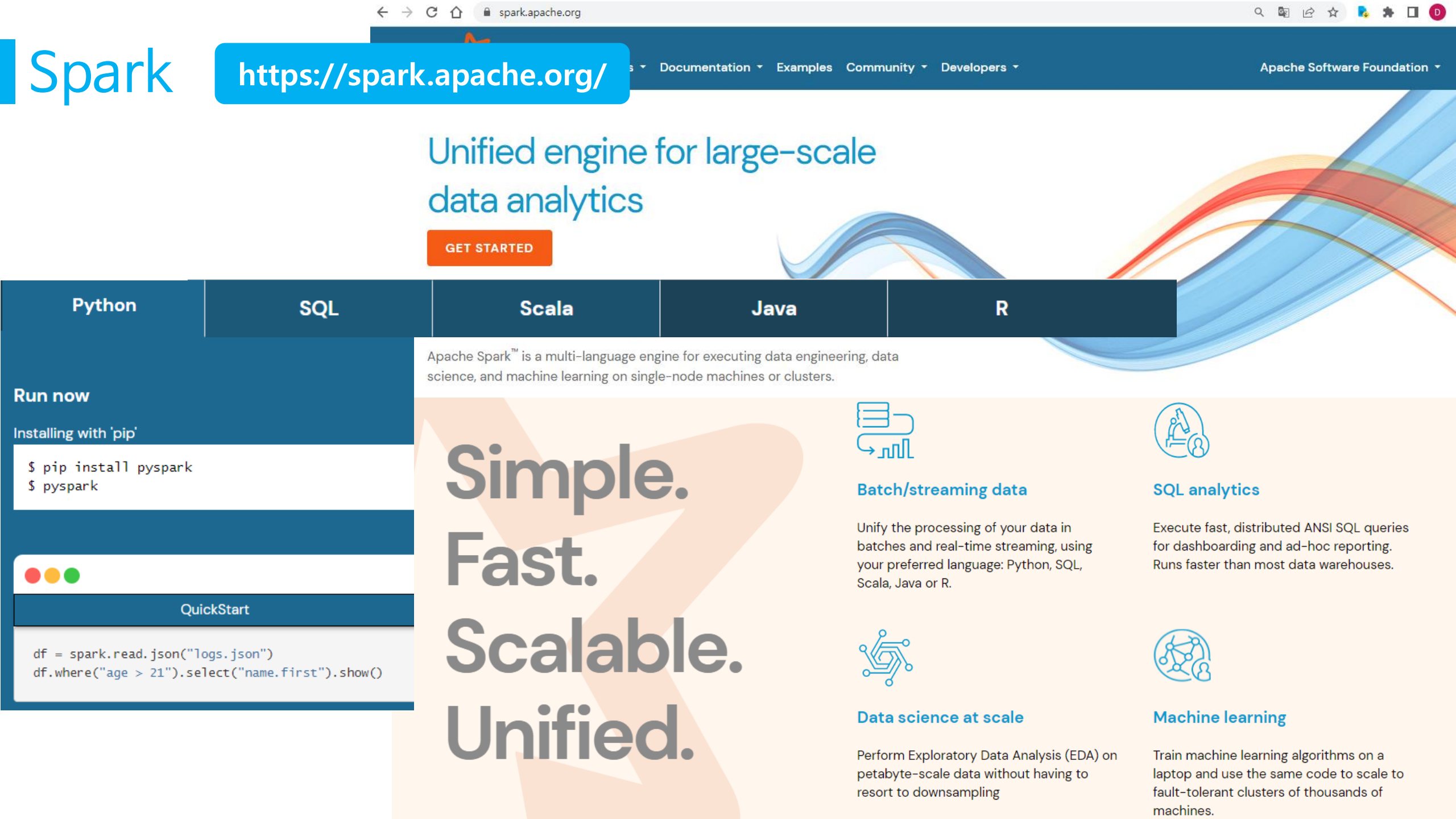
빅데이터 프로세싱 엔진 Spark



박경규

Spark

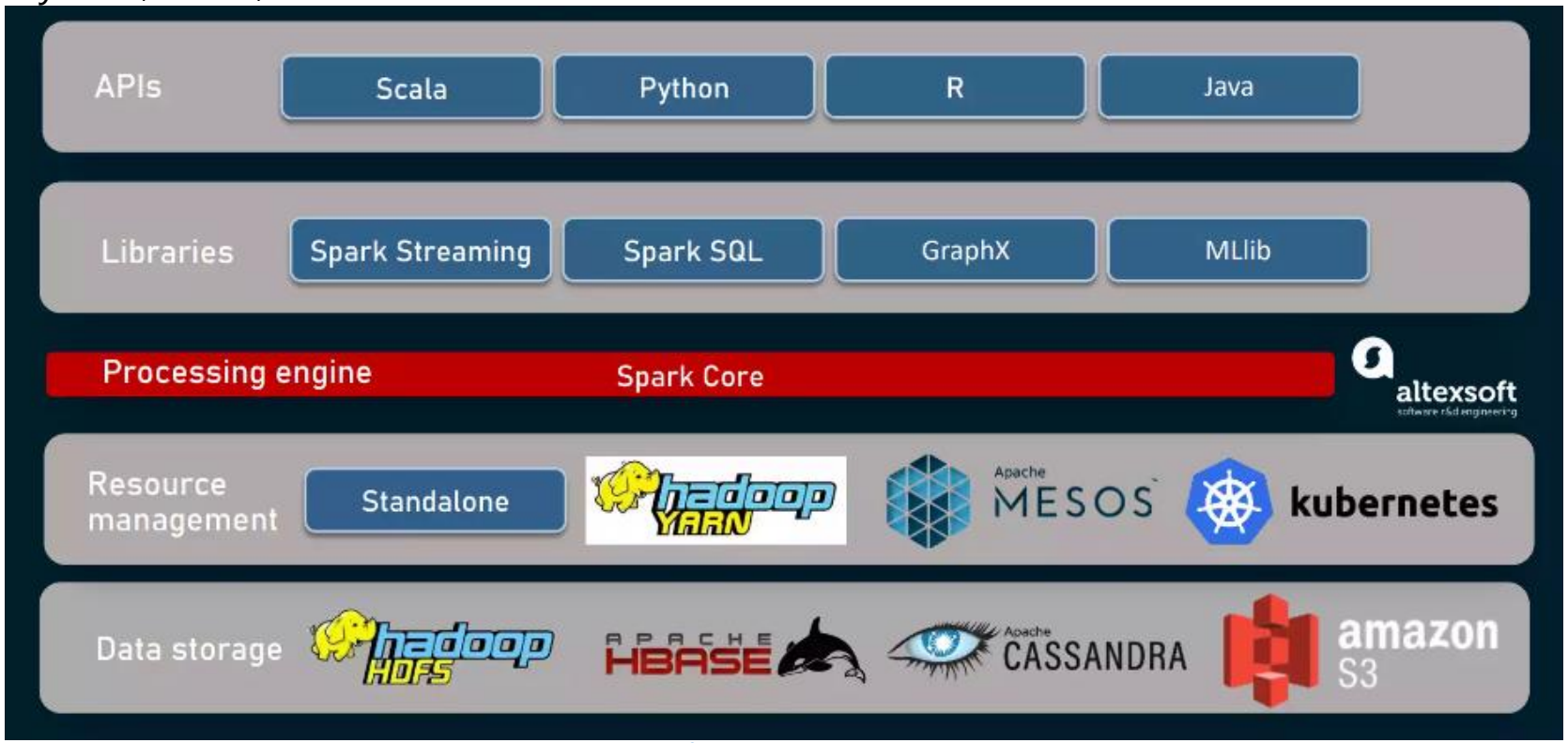
https://spark.apache.org/



Spark 특징

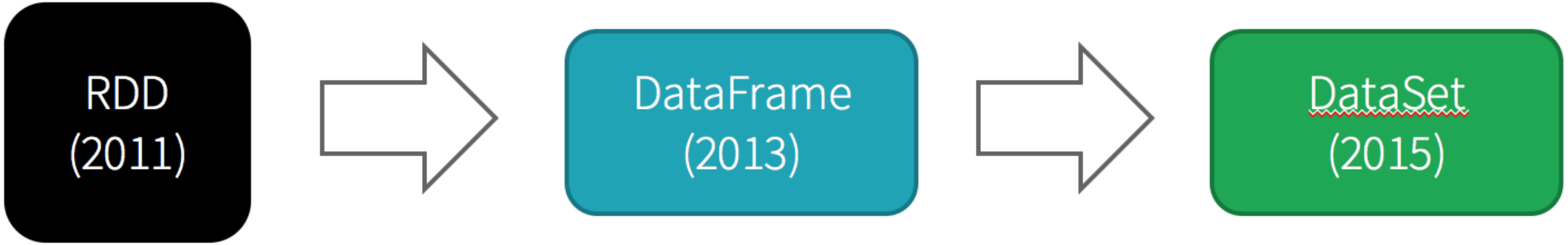
- 인메모리(In-Memory) 컴퓨팅, Disk 기반도 가능
- RDD(Resilient Distributed DataSet) 데이터모델, 빠른 데이터 프로세싱(In-Memory Cached RDD, Up to 100x faster)
- 실시간(Real-Time) Stream Processing
- 다양한 개발언어(Scala, Python, Java, R, SQL) 지원, 개발자 친화적인 수많은 API 제공, 2~10x Less Code
- 사용자의 데이터 처리 명령을 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 스케줄링
- Hadoop(HDFS, YARN, HBase 등)과 유연한 연계
- 대화형 질의를 위한 Interactive Shell : Python, Scala, R 인터프리터
- 하나의 애플리케이션에서 배치,
  SQL 쿼리, 스트리밍, 머신러닝과
  같은 다양한 작업을
  하나의 워크플로우로 결합 가능

참고 : https://www.itworld.co.kr/news/147556



https://www.altexsoft.com/blog/hadoop-vs-spark/

History of Spark APIs



Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

Internally rows, externally JVM objects

Almost the "Best of both worlds": **type safe + fast**

But slower than DF
Not as good for interactive analysis, especially Python

databricks

Hadoop vs Spark

- 대부분의 Hadoop 배포판에 Spark가 포함되어 있으며, Hadoop vs Spark 비교는 다소 부적절합니다.
- Spark는 인메모리 데이터 엔진을 통해 특정 상황에서 맵리듀스보다 100배 더 빠르게 작업을 수행하고, 개발자 친화적인 API 제공 이점 덕분에 빅데이터 처리 분야에서 Haddop MapReduce 패러다임을 추월하여 가장 유력한 프레임워크로 부상 하였습니다.

	Hadoop	Spark
What is it?	Open-source framework for distributed data storage and processing	Open-source framework for in-memory distributed data processing and app development
Initial release	2006	2014
Supported languages	Java	Scala, Java, Python, R
Processing methods	Batch processing, using hard discs to read/write data	Batch and micro-batch processing in RAM
Built-in capabilities	✓ File system (HDFS) ✓ Resource management (Yarn) ✓ Processing engine (MapReduce)	✓ Processing engine (Spark Core) ✓ Near real-time processing (Spark Streaming) ✓ Structured data processing (Spark SQL) ✓ Graph data management (GraphX) ✓ ML library (MLlib)

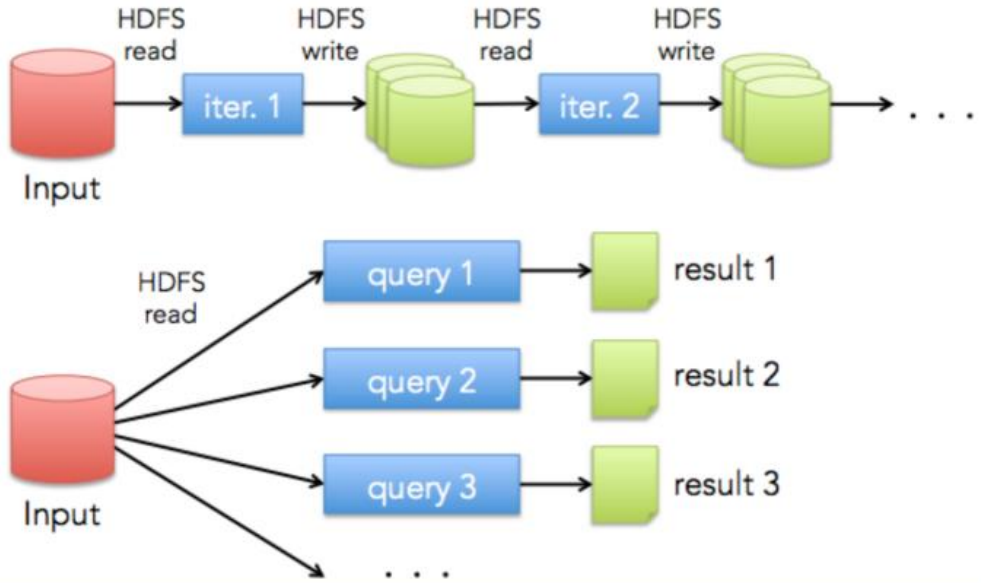
	Hadoop	Spark
Best fit for	Delay-tolerant processing tasks, involving huge datasets	Almost instant processing of live data and quick analytics app development
Real-life use cases	✓ Enterprise archived data processing ✓ Sentiment analysis ✓ Predictive maintenance ✓ Log files analysis	✓ Fraud detection ✓ Telematics analytics ✓ User behavior analysis ✓ Near real-time recommender systems ✓ Stock market trends prediction ✓ Risk management

https://www.altexsoft.com/blog/hadoop-vs-spark/

Hadoop vs Spark

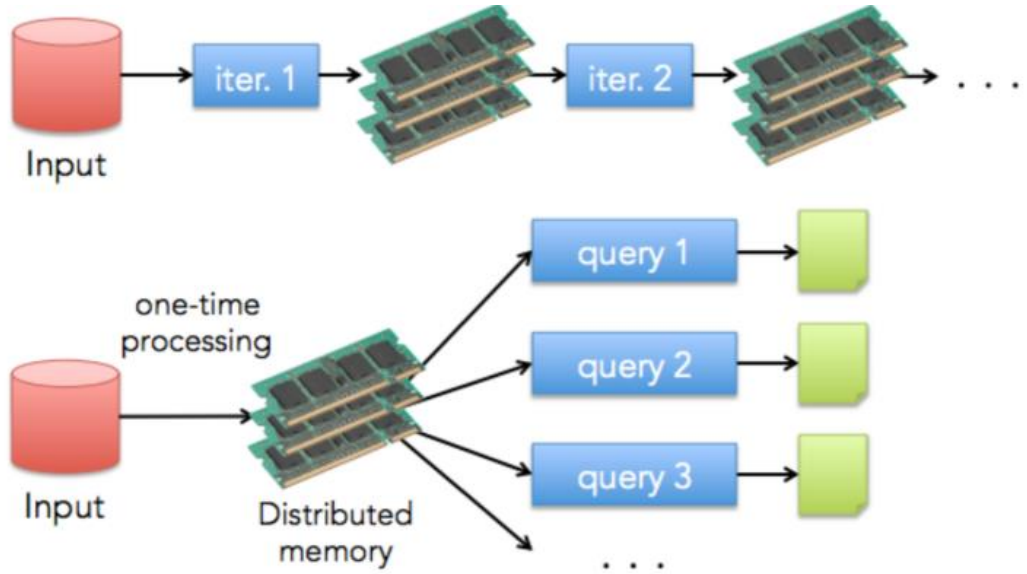
- Hadoop MapReduce는 각 map, reduce 이후에 대부분의 데이터를 디스크(HDFS)에 저장합니다.
- Spark은 각 변환 후 대부분의 데이터를 메모리에 보관합니다 .
- Spark은 인메모리 기반의 처리로 Hadoop MapReduce 대비 10~100배 빠른 속도를 제공합니다.
- Spark은 머신러닝, 그래프처리 등 빅데이터 분석을 위한 통합 컴포넌트를 제공합니다.

Hadoop



Slow due to data replication and disk I/O

Spark



10-100× faster than network and disk

Hadoop vs Spark

Spark wins Daytona Gray Sort 100TB Benchmark

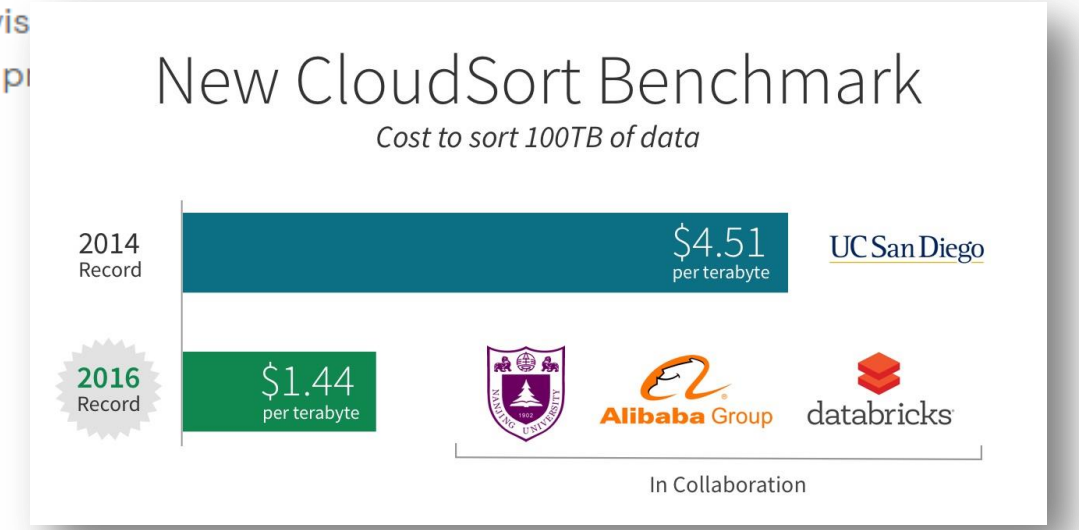
We are proud to announce that Spark won the 2014 Gray Sort Benchmark (Daytona 100TB category). A team from Databricks including Spark committers, Reynold Xin, Xiangrui Meng, and Matei Zaharia, entered the benchmark using Spark. Spark won a tie with the Themis team from UCSD, and jointly set a new world record in sorting.

They used Spark and sorted 100TB of data using 206 EC2 i2.8xlarge machines in 23 minutes. The previous world record was 72 minutes, set by a Hadoop MapReduce cluster of 2100 nodes. This means that Spark sorted the same data 3X faster using 10X fewer machines. All the sorting took place on disk (HDFS), without using Spark's in-memory cache.

Outperforming large Hadoop MapReduce clusters on sorting not only validates the vis
work done by the Spark community, but also demonstrates that Spark is fulfilling its pr
to serve as a faster and more scalable engine for data processing of all sizes.

For more information, see the Databricks blog article written by the Reynold Xin.

https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark.html



https://www.databricks.com/blog/2016/11/14/setting-new-world-record-apache-spark.html

Spark 설치

■ Apache Hadoop 설치: 빅데이터플랫폼 Hadoop교재 내용 참조

■ Apache Spark 다운로드 및 설치

```
cd ~/
wget https://downloads.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz --no-check-certificate
tar zxvf spark-3.3.2-bin-hadoop3.tgz
mv spark-3.3.2-bin-hadoop3 spark3
mkdir ~/spark3/spark-events
```

■ ~/.bashrc 파일 내용 추가

```
export SPARK_HOME=/home/abc/spark3
export PATH=$SPARK_HOME/bin:$SPARK_HOME/sbin:$PATH'
```

■ Spark 설정 파일 복사

```
cd ~/spark3/conf
cp spark-env.sh.template spark-env.sh
cp spark-defaults.conf.template spark-defaults.conf
```

Spark 설정

■ spark-env.sh 파일 내용 추가

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

export SPARK_MASTER_HOST=localhost

export HADOOP_HOME=/home/abc/hadoop3

export HADOOP_CONF_DIR=/home/abc/hadoop3/etc/hadoop

■ spark-defaults.conf 파일 내용 추가

spark.master yarn

spark.eventLog.enabled true

spark.eventLog.dir file:/home/abc/spark3/spark-events

spark.history.fs.logDirectory file:/home/abc/spark3/spark-events

■ workers 파일에 Worker Node 호스트네임 추가

worker1

worker2

worker3

Spark 실행

Spark 실행

cd ~/spark3/sbin

./start-all.sh

./start-history-server.sh

Spark 실행 확인

jps

Spark Master : http://localhost:8080/

Spark History Server : http://localhost:18080/

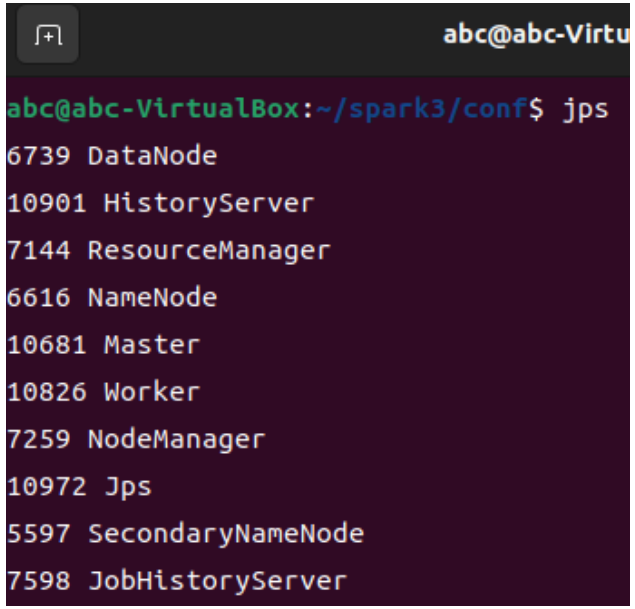
Spark Worker : http://worker1:8081/
http://worker2:8081/
http://worker3:8081/

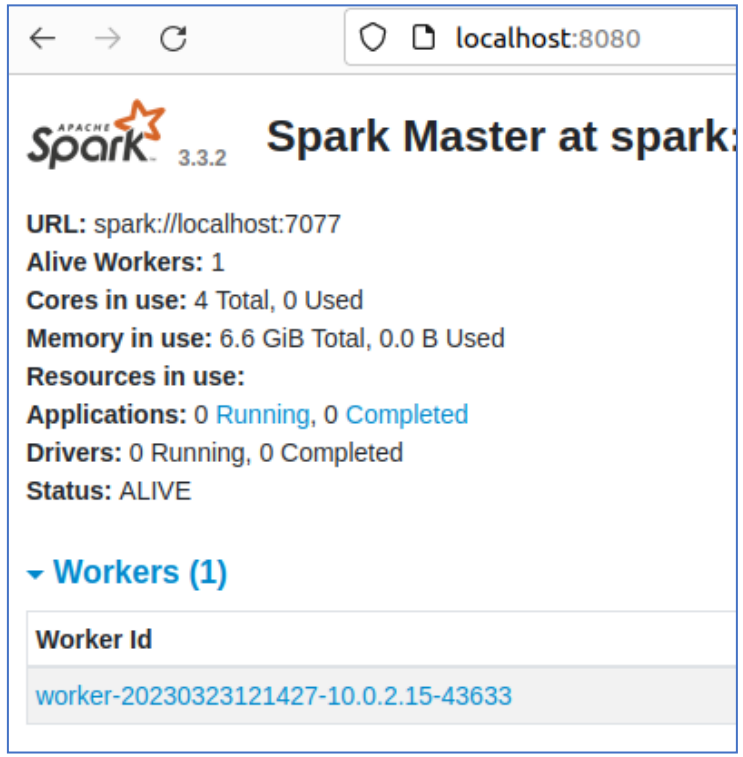
Spark 정지

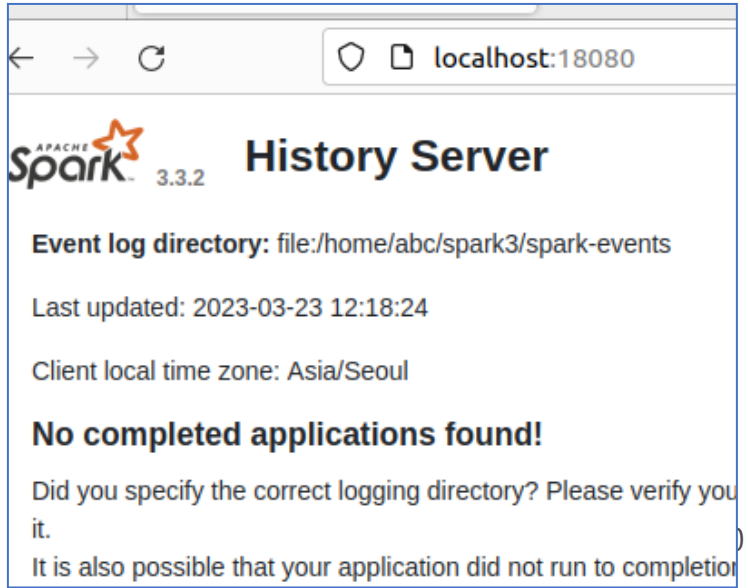
cd ~/spark3/sbin ./stop-workers.sh

./stop-all.sh ./stop-master.sh

./stop-history-server.sh





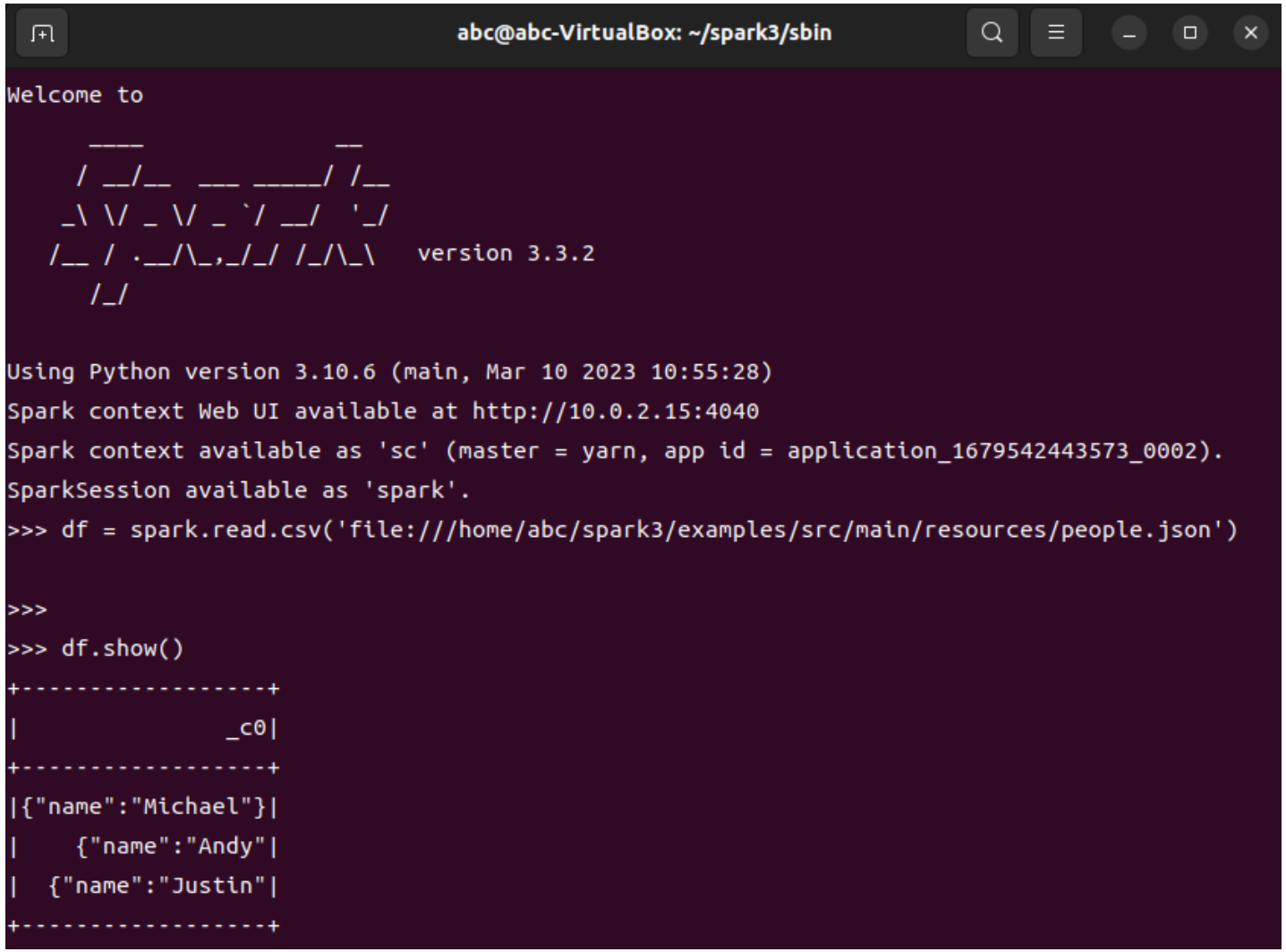


Spark Shell 사용

Spark Shell 사용

spark-shell

:q



PySpark Shell 사용

pyspark

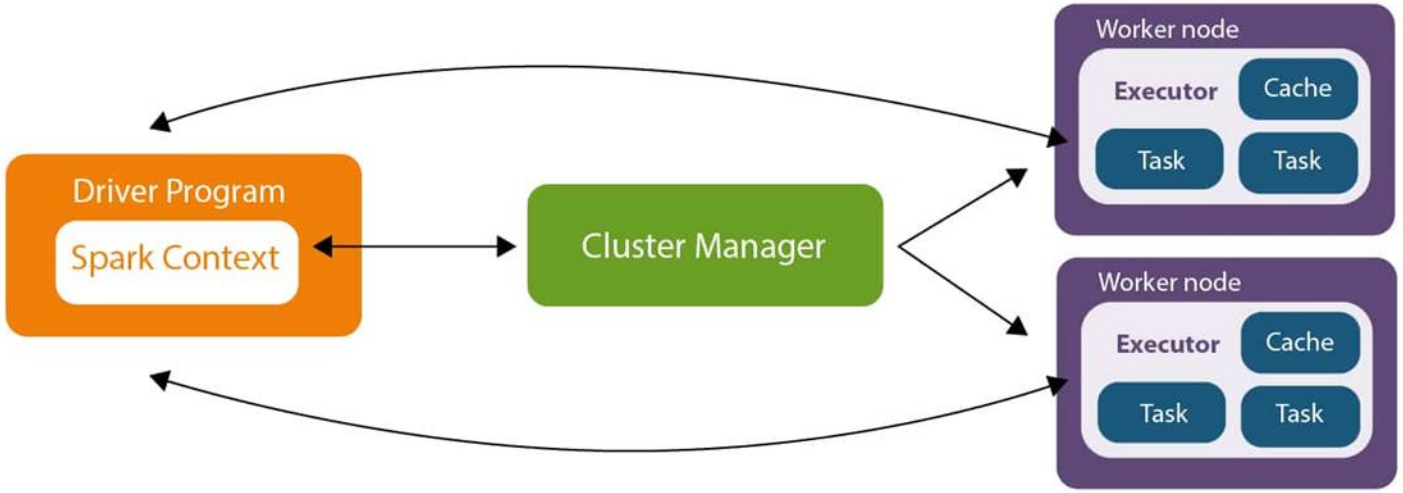
df = spark.read.csv('file:///home/abc/spark3/examples/src/main/resources/people.json')

df.show()

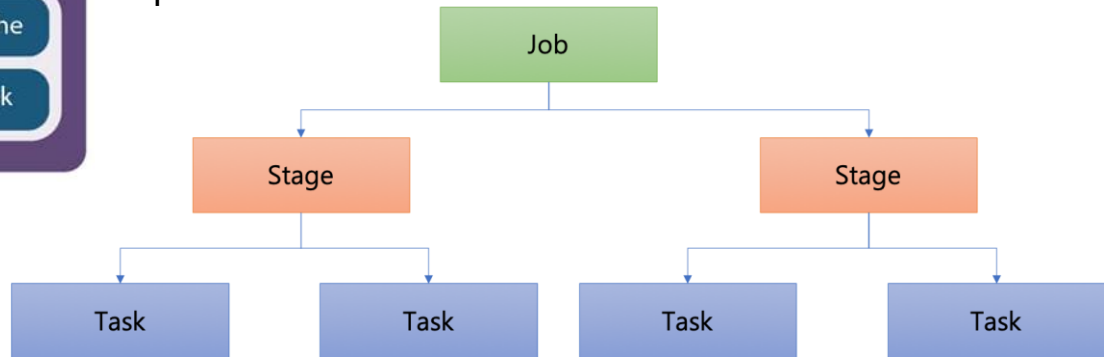
quit()

Spark Application 구조

- Spark Application : Spark에서 수행되는 사용자 프로그램으로 1개의 Driver Program과 N개의 Executor로 구성
- Driver Program : Spark Application의 main() 함수를 실행하고 SparkContext를 생성하는 프로세스
  Spark Driver 라고도 하며, 사용자 프로그램을 실제 수행 단위인 Task로 변환 해 Executor에 할당하고 Task들을 스케줄링
- SparkContext :Driver Program에서 Job을 Executor에 실행하기 위한 Endpoint. Cluster Manager와 연결
- Cluster Manager : 클러스터 환경에서 Application(Driver와 Executor) 사이의 자원을 관리해주는 역할을 담당
- Executor : Worker Node에서 실행되는 프로세스. Driver 의 요청을 받아서 Task를 실행



- Task : 하나의 Executor에서 수행되는 최소 작업 단위
- Stage : Task의 집합
- Job : Stage의 집합, Application에서 Spark에 요청하는 일련의 작업



Spark Cluster 종류

Local Mode

- 클러스터 없이 하나의 JVM 에 driver 1개와 executor 1개씩만 생성합니다.
- 단순 테스트 용도로 사용하며, Executor는 스레드를 여러개 생성할 수 있습니다.

Local Cluster Mode

- Master 와 Worker 프로세스가 있습니다. Worker 는 각각의 JVM 에서 실행됩니다.

Spark Standalone Cluster Mode

- Master 와 Worker 프로세스가 있습니다. Master 프로세스가 클러스터 매니저 역할을 합니다.

Spark Yarn Cluster Mode

- Yarn 이 클러스터 매니저 역할을 합니다. Hadoop Cluster 에 같이 설치합니다,

Spark Mesos Cluster Mode

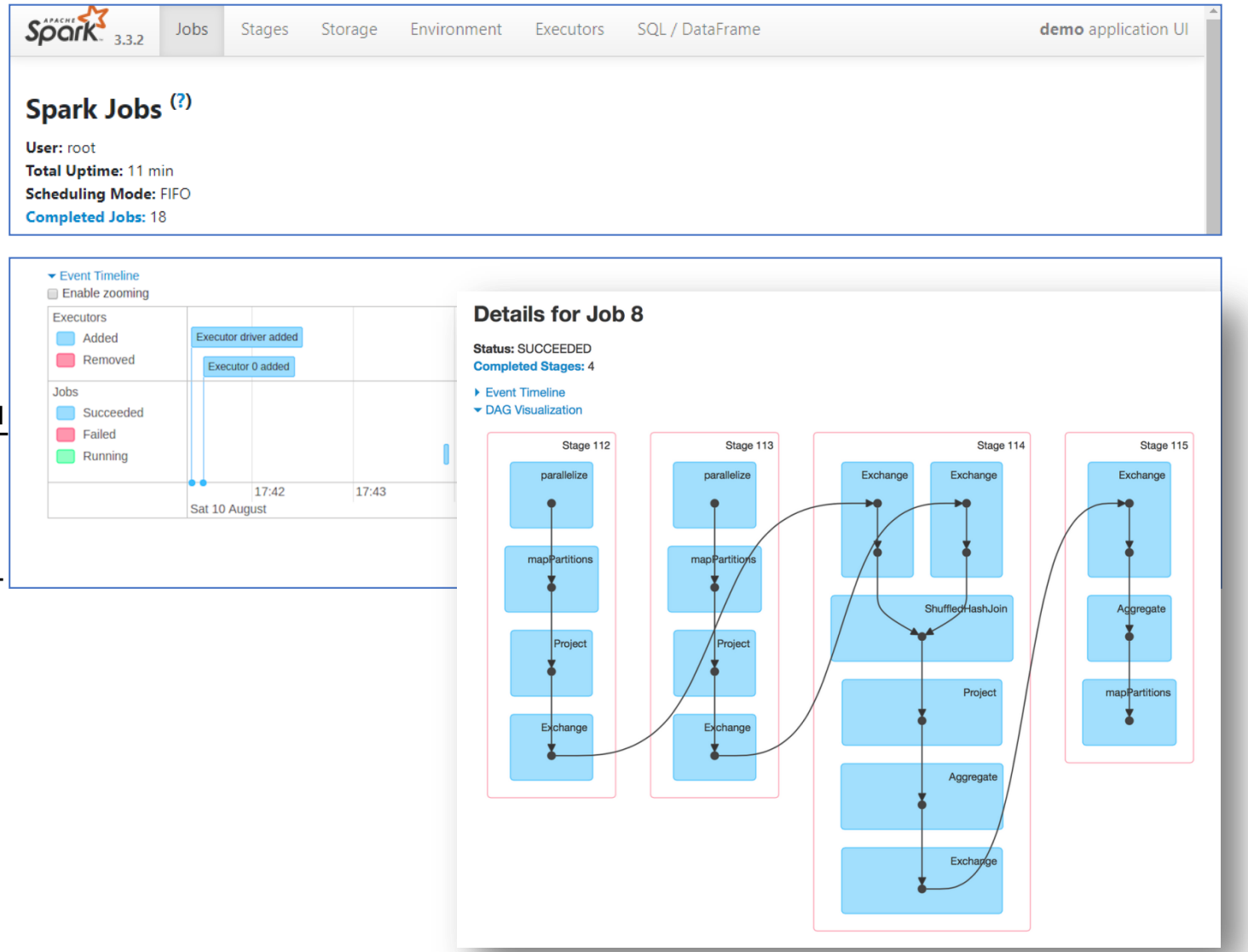
- Deprecated 되었습니다

Kubernetes

- Driver, Executor pod 에 대한 스케줄링을 쿠버네티스가 핸들링하는 형태입니다.
- 쿠버네티스에서 지원하는 Docker 컨테이너 이미지로 사용 가능합니다.

Spark Web UI

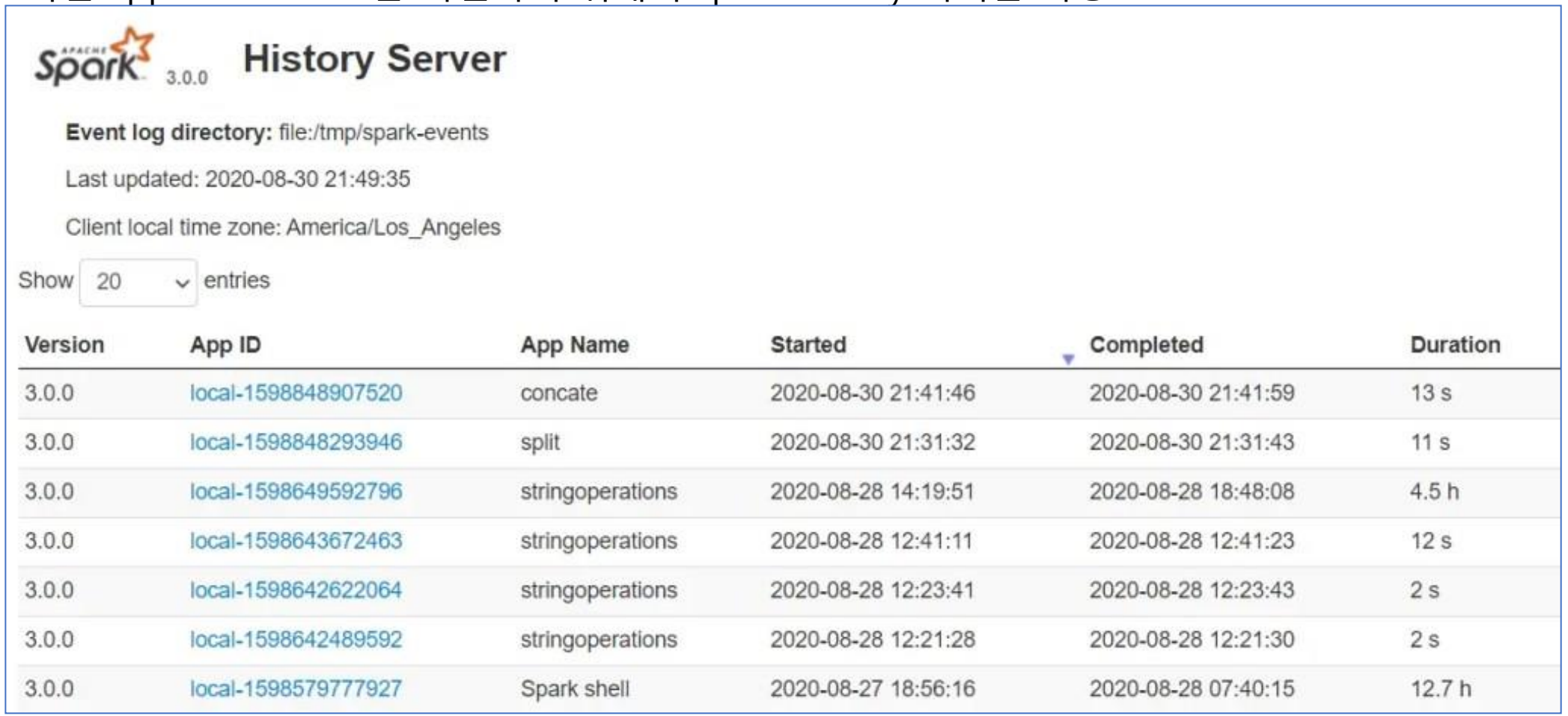
- http://<Drive Node>:4040(default)
- Jobs : Spark 애플리케이션의 모든 job에 대한 요약 정보
- Stages : jobs의 모든 stages의 현재 상태 요약 정보
- Storage : persisted RDD와 DataFrame 정보
- Environment : 다양한 환경 변수값
- Executors : Executor 정보. 메모리와 디스크 사용량, task, shuffle 정보 등
- SQL/DataFrame : 애플리케이션이 Spark SQL 쿼리 실행 시 정보 제공



https://spark.apache.org/docs/latest/web-ui.html

History Server Web UI

- http://<Drive Node>:7077(default)
- 이전 application 로그를 확인하기 위해서 spark history 서버를 사용



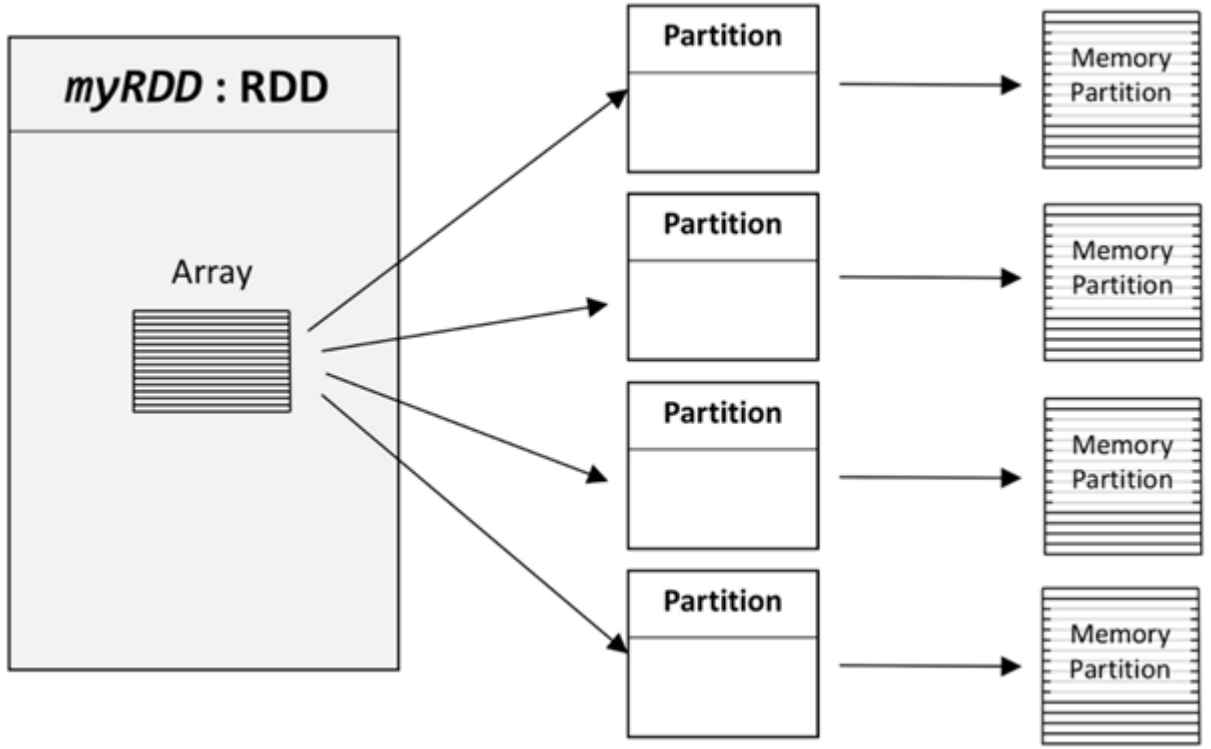
Version	App ID	App Name	Started	Completed	Duration
3.0.0	local-1598848907520	concate	2020-08-30 21:41:46	2020-08-30 21:41:59	13 s
3.0.0	local-1598848293946	split	2020-08-30 21:31:32	2020-08-30 21:31:43	11 s
3.0.0	local-1598649592796	stringoperations	2020-08-28 14:19:51	2020-08-28 18:48:08	4.5 h
3.0.0	local-1598643672463	stringoperations	2020-08-28 12:41:11	2020-08-28 12:41:23	12 s
3.0.0	local-1598642622064	stringoperations	2020-08-28 12:23:41	2020-08-28 12:23:43	2 s
3.0.0	local-1598642489592	stringoperations	2020-08-28 12:21:28	2020-08-28 12:21:30	2 s
3.0.0	local-1598579777927	Spark shell	2020-08-27 18:56:16	2020-08-28 07:40:15	12.7 h

RDD (Resilient Distributed Dataset)

- Spark에서 사용되는 가장 기본적인 데이터 객체입니다.
- Spark에서 데이터는 클러스터 메모리에 분산되어 Partition 단위로 분산 저장됩니다.
- Lineage(RDD를 만드는 일련의 단계)를 기록하여 노드의 장애/실패 발생 시 데이터를 재구성할 수 있습니다.
- RDD는 외부 데이터를 읽어서 처리하거나, 자체적으로 컬렉션 데이터를 생성하여 처리할 수 있습니다.

■ RDD 주요 특성(feature)

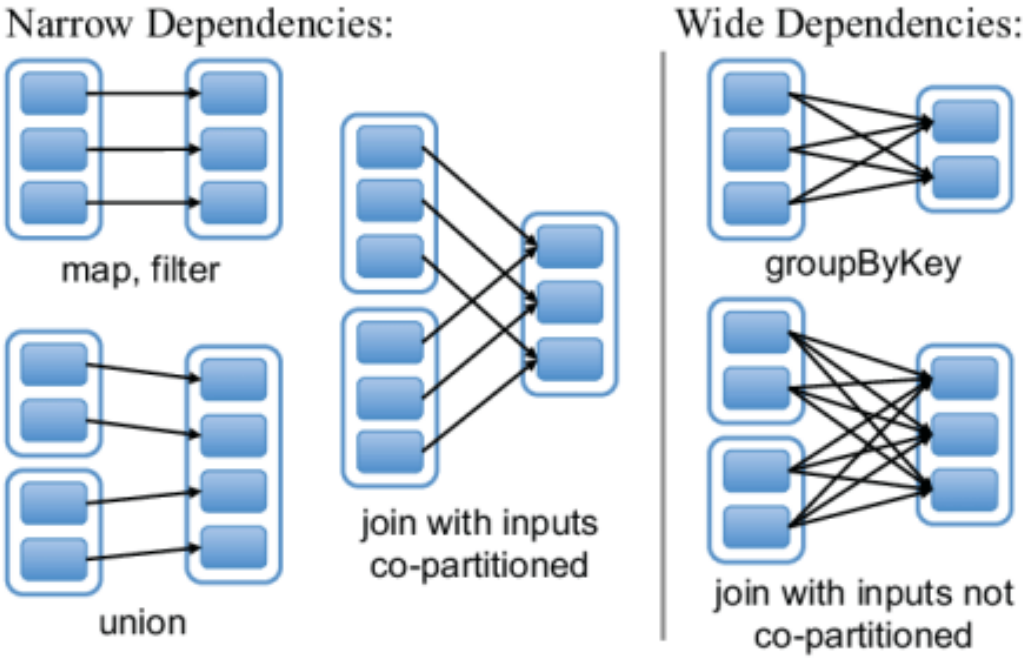
- Distributed Collection of Data
- Fault-tolerant
- Parallel operation – partitioned
- Ability to use many data sources



RDD Operation

Transformations

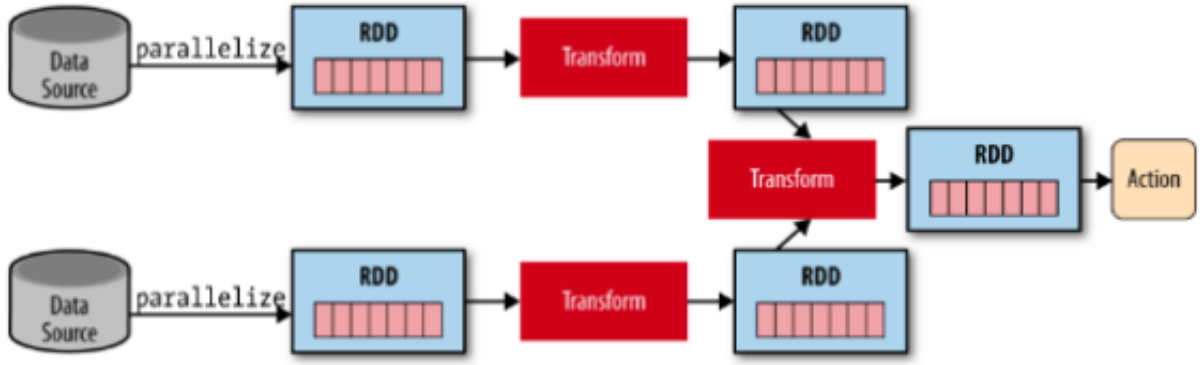
- 데이터를 처리(변경)하는 명령으로 새로운 RDD를 생성함. map, filter, flatMap, join 등
- RDD 생성 → RDD 변환 → RDD 연산
- Action을 실행 때까지 지연 연산(Lazy Evaluation)



참고 : https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

Actions

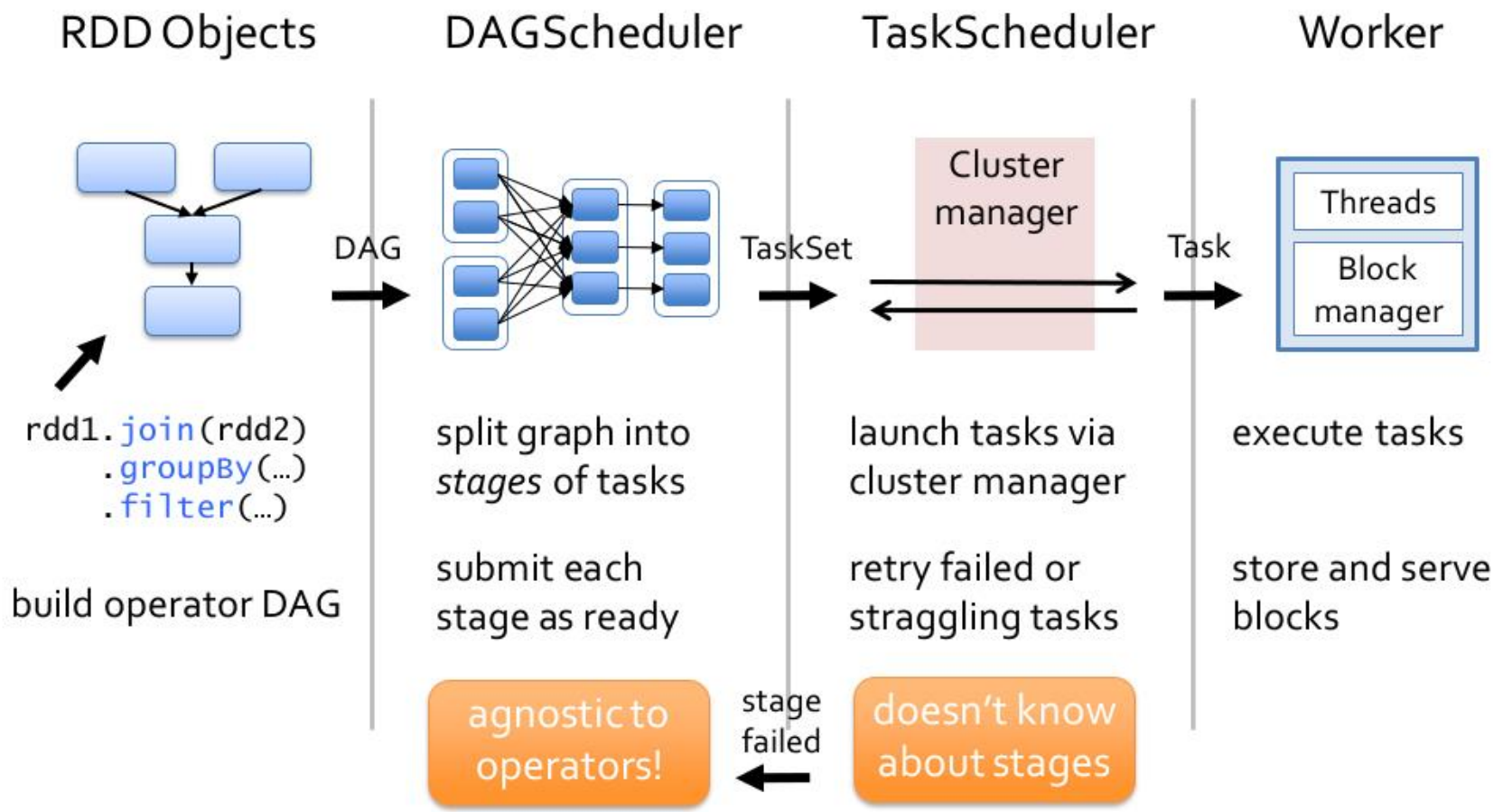
- Transformation의 결과연산을 리턴/저장하는 명령
- count, collect, reduce, save 등

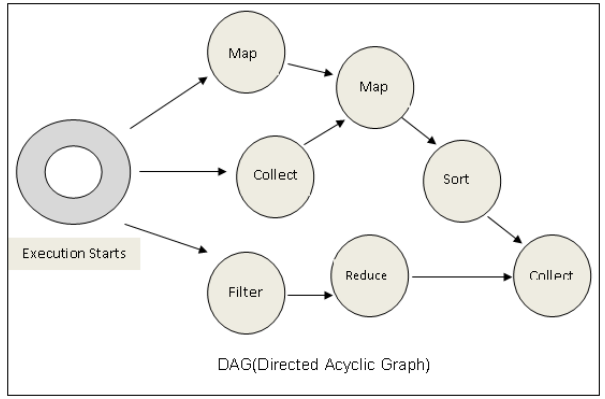


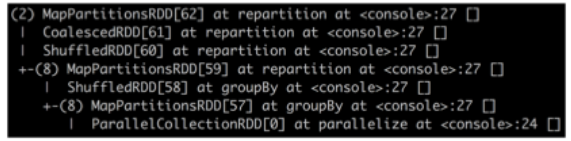
참고 : https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

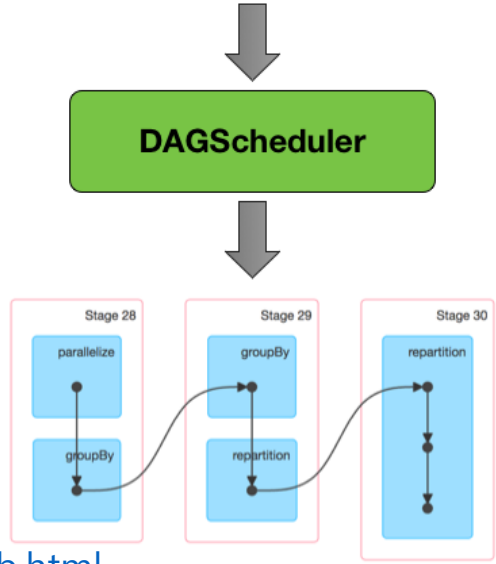
Spark 동작 방식

- RDD operation을 통해 DAG가 정의 되고 Task Scheduler를 이용하여 데이터를 처리합니다.









http://simpledbs.blogspot.com/2017/03/understanding-beauty-of-sparks-job.html

RDD 생성

- nums = parallelize([1, 2, 3, 4])
- sc.textFile("file:///home/abc/spark3/README.md") or s3n:// . hdfs://
- hiveCtx = HiveContext(sc) rows = hiveCtx.sql("SELECT name, age FROM users")
- Can also create from:
  - JDBC
  - Cassandra
  - Hbase
  - Elasticsearch
  - JSON, CSV, sequence files, object files, various compressed formats

RDD Transformations

- map
- flatmap
- filter
- distinct
- sample
- union, intersection, subtract, cartesian

map example

```
rdd = sc.parallelize([1, 2, 3, 4])
quaredRDD = rdd.map(lambda x: x*3)

This yeilds 1, 4, 9, 16
```

참고 : https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

RDD actions

- collect
- count
- countByValue
- take
- top
- reduce
- … and more …

참고 : https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

Spark SQL

- SQL 또는 DataFrame API를 사용하여 Spark 프로그램 내에서 구조화된 데이터를 쿼리할 수 있습니다.
- Hive, Avro, Parquet, ORC, JSON, JDBC등 다양한 데이터 소스에 액세스할 수 있습니다.

Integrated

Seamlessly mix SQL queries with Spark programs.

Spark SQL lets you query structured data inside Spark programs, using either SQL or a familiar DataFrame API. Usable in Java, Scala, Python and R.

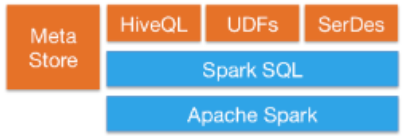
```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda
p: p.name)
```

Apply functions to results of SQL queries.

Hive integration

Run SQL or HiveQL queries on existing warehouses.

Spark SQL supports the HiveQL syntax as well as Hive SerDes and UDFs, allowing you to access existing Hive



Spark SQL can use existing Hive metastores, SerDes, and UDFs.

Uniform data access

Connect to any data source the same way.

DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. You can even join data across these sources.

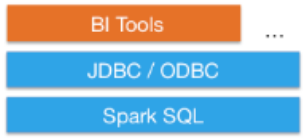
```
spark.read.json("s3n://..."
)
  .registerTempTable("json"
)
results = spark.sql(
  """SELECT *
     FROM people
     JOIN json ..""")
```

Query and join different data sources.

Standard connectivity

Connect through JDBC or ODBC.

A server mode provides industry standard JDBC and ODBC connectivity for business intelligence tools.

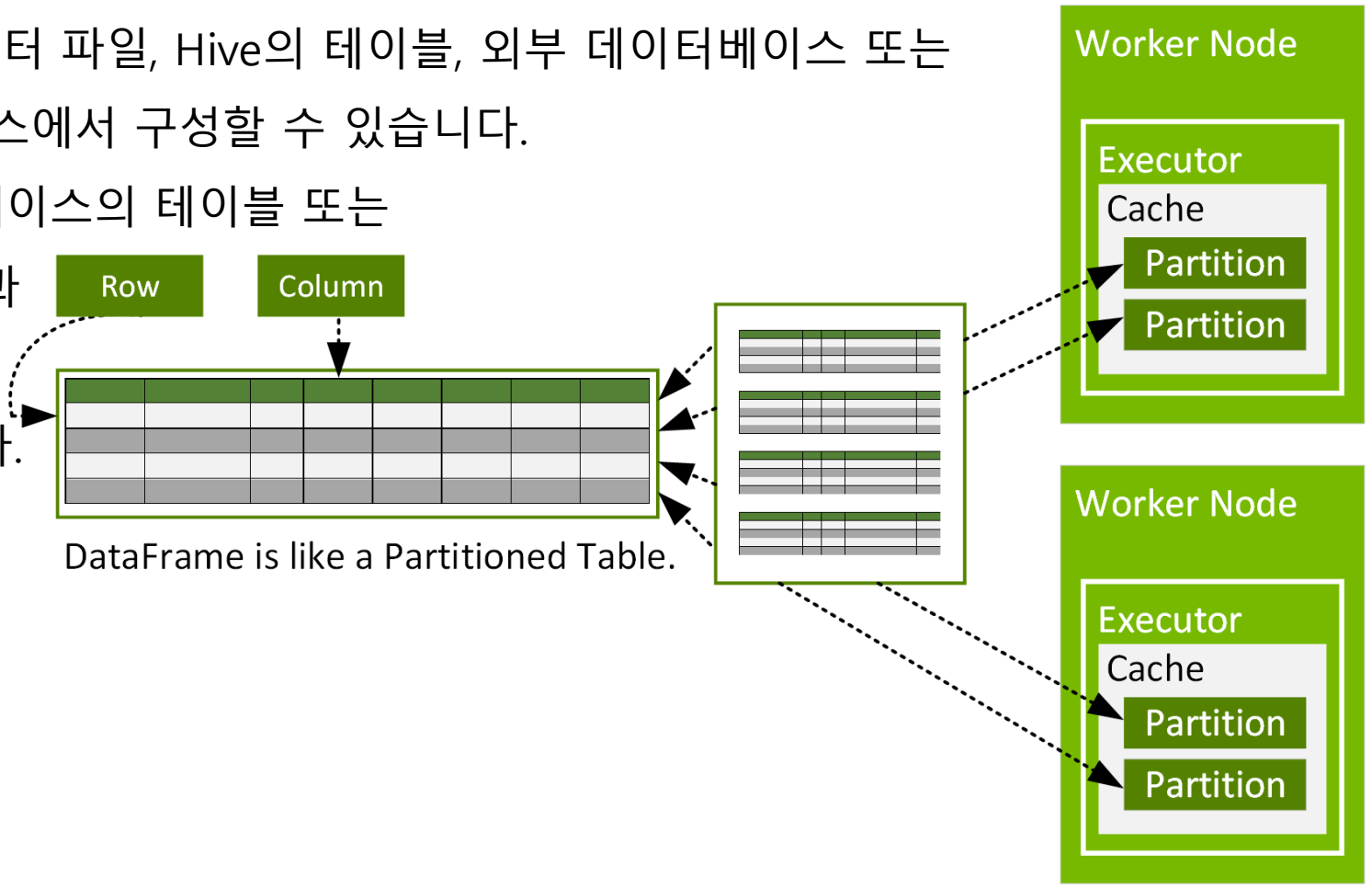


Use your existing BI tools to query big data.

https://spark.apache.org/sql/

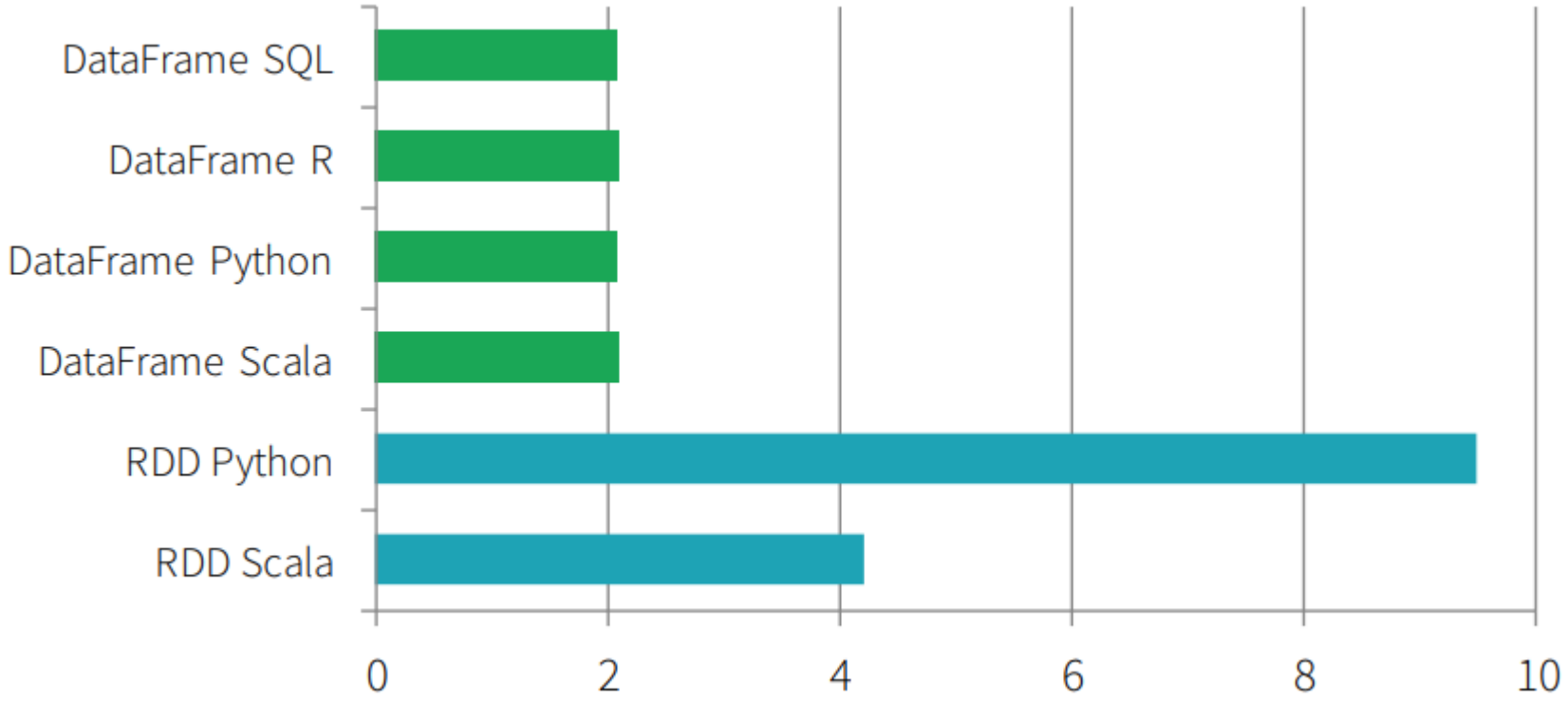
Spark DataFrame

- Spark DataFrame은 클러스터의 여러 노드에 분할되어 병렬로 작동할 수 있는 org.apache.spark.sql.Row 개체의 분산된 데이터셋입니다.
- DataFrame은 파티션으로 구성되며, 각 파티션은 DataNode의 Cache에 있는 행 범위입니다.
- DataFrame은 구조화된 데이터 파일, Hive의 테이블, 외부 데이터베이스 또는 기존 RDD와 같은 다양한 소스에서 구성할 수 있습니다.
- 개념적으로 관계형 데이터베이스의 테이블 또는 R/Python의 데이터 프레임과 동일하지만 내부적으로 더 풍부한 최적화가 있습니다.



Spark DataFrame

- DataFrame은 RDD보다 훨씬 빠를 수 있습니다.
- 그리고, 언어에 관계없이 동일하게 수행합니다.



Time to aggregate 10 million integer pairs (in seconds)

https://piazza.com/class_profile/get_resource/jde8xt6ku6rsq/jde8xuaztu0tr

Spark DataFrame

Write Less Code: Compute an Average

hadoop

```
private IntWritable one = new IntWritable(1);
private IntWritable output =new IntWritable();
protected void map(LongWritable key,
                   Text value,
                   Context context) {
    String[] fields = value.split("\t");
    output.set(Integer.parseInt(fields[1]));
    context.write(one, output);
}

------------------------------------------------------------------------
--

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable();

protected void reduce(IntWritable key,
                      Iterable<IntWritable>
values,
                      Context context) {
    int sum = 0;
    int count = 0;
    for (IntWritable value: values) {
        sum += value.get();
        count++;
    }
    average.set(sum / (double) count);
    context.write(key, average);
}
```

Spark

Using RDDs

```
var data = sc.textFile(...).split("\t")
data.map { x => (x(0), (x(1), 1)) }
    .reduceByKey { case (x, y) =>
      (x._1 + y._1, x._2 + y._2) }
    .map { x => (x._1, x._2(0) / x._2(1)) }
    .collect()
```

Using DataFrames

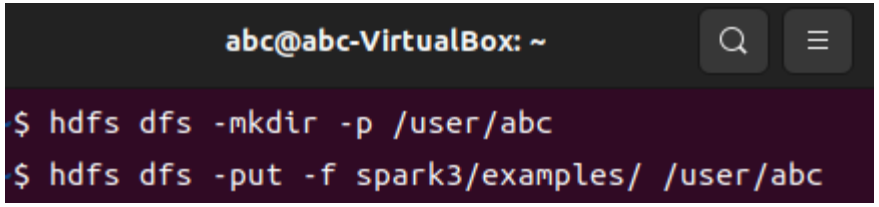
```
sqlContext.table("people")
          .groupBy("name")
          .agg("name", avg("age"))
          .collect()
```

databricks

https://piazza.com/class_profile/get_resource/jde8xt6ku6rsq/jde8xuaztu0tr

Spark DataFrame

```
# spark is an existing SparkSession
df = spark.read.json("examples/src/main/resources/people.json")
# Displays the content of the DataFrame to stdout
df.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)
```



hdfs dfs -mkdir -p /user/abc

hdfs dfs -put -f spark3/examples/ /user/abc

https://spark.apache.org/docs/latest/sql-getting-started.html#creating-dataframes

Spark DataFrame

```
# Select only the "name" column
df.select("name").show()
# +-------+
# |   name|
# +-------+
# |Michael|
# |   Andy|
# | Justin|
# +-------+

# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-------+---------+
# |   name|(age + 1)|
# +-------+---------+
# |Michael|     null|
# |   Andy|       31|
# | Justin|       20|
# +-------+---------+
```

```
# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+----+
# |age|name|
# +---+----+
# | 30|Andy|
# +---+----+

# Count people by age
df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |null|    1|
# |  30|    1|
# +----+-----+
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

https://spark.apache.org/docs/latest/sql-getting-started.html#creating-dataframes

Colab에서 PySpark 사용하는 방법

spark_in_colab.ipynb

■ 방법 #1

```
!pip install pyspark py4j
```

■ 방법 #2

```
# !apt-get install openjdk-8-jdk-headless -qq
!wget -q !wget -q https://downloads.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz
!tar -xf spark-3.3.2-bin-hadoop3.tgz
!pip install -q findspark

import os
import findspark

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.2-bin-hadoop3"

findspark.init()
findspark.find()
```

참조 : https://medium.com/@TheITspace/running-pyspark-on-google-colab-2552435972b3

PySpark RDD 실습

spark_rdd.ipynb

■ textFile() 메소드를 사용해 RDD 만들기

```
[7]  # licenses RDD에 전체 디렉토리의 내용 로드
     license_files = sc.textFile("file:///content/spark-3.3.2-bin-hadoop3/licenses/")
```

```
[8]  # 생성된 객체 검사
     license_files
```

```
     file:///content/spark-3.3.2-bin-hadoop3/licenses/ MapPartitionsRDD[6] at textFile at NativeMethodAccessorImpl.java:0
```

```
[9]  license_files.take(1)
```

```
     ['<HTML>']
```

```
[10] license_files.getNumPartitions()
```

```
     58
```

```
[11] # 모든 파일에서 총 라인 수
     license_files.count()
```

```
     2998
```

PySpark RDD 실습

spark_rdd.ipynb

■ 데이터 소스에서 RDD 생성

```
[13] people = spark.read.json("/content/spark-3.3.2-bin-hadoop3/examples/src/main/resources/people.json")
```

```
[14] people
```

```
DataFrame[age: bigint, name: string]
```

```
[15] people.dtypes
```

```
[('age', 'bigint'), ('name', 'string')]
```

```
[16] people.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
[17] from pyspark.sql import SQLContext

     # as with all DataFrames you can create use them to run SQL queries as follows
     sqlContext = SQLContext(sc)
     sqlContext.registerDataFrameAsTable(people, "people")
     df2 = spark.sql("SELECT name, age FROM people WHERE age > 20")
     df2.show()
```

```
/content/spark-3.3.2-bin-hadoop3/python/pyspark/sql/context.py:112: FutureWarnin
  warnings.warn(
+----+---+
|name|age|
+----+---+
|Andy| 30|
+----+---+
```

PySpark RDD 실습

spark_rdd.ipynb

■ 프로그래밍 방식으로 RDD 생성

```
[18] parallel_rdd = sc.parallelize([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
[19] parallel_rdd
```

```
ParallelCollectionRDD[21] at readRDDFromFile at PythonRDD.scala:274
```

```
[20] parallel_rdd.count()
```

```
9
```

```
[21] # 0에서 시작해서 1000개의 정수로, 2개의 파티션에서 1씩 증가하는 RDD 생성
     range_rdd = sc.range(0, 1000, 1, 2)
     range_rdd
```

```
PythonRDD[24] at RDD at PythonRDD.scala:53
```

```
[22] range_rdd.getNumPartitions()
```

```
2
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ 데이터 파일 : people.json

{"name":"Michael"}

{"name":"Andy", "age":30}

{"name":"Justin", "age":19}

■ Creating a DataFrame

```
[2]  from pyspark.sql import SparkSession
```

```
[3]  # May take a little while on a local computer
     spark = SparkSession.builder.appName("Basics").getOrCreate()
```

```
[4]  df = spark.read.json('people.json')
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ Showing the data

```
[5]  # Note how data is missing!
     df.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
[6]  df.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

```
[7]  df.columns
```

```
['age', 'name']
```

```
[8]  df.describe()
```

```
DataFrame[summary: string, age: string, name: string]
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ Infer schema

```
[9]  from pyspark.sql.types import StructField,StringType,IntegerType,StructType
```

```
[10] data_schema = [StructField("age", IntegerType(), True),StructField("name", StringType(), True)]
```

```
[11] final_struc = StructType(fields=data_schema)
```

```
[12] df = spark.read.json('people.json', schema=final_struc)
```

```
[13] df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- name: string (nullable = true)
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ Grabbing the data

```
[14]  df['age']
```

```
Column<'age'>
```

```
[15]  type(df['age'])
```

```
pyspark.sql.column.Column
```

```
[16]  df.select('age')
```

```
DataFrame[age: int]
```

```
[17]  type(df.select('age'))
```

```
pyspark.sql.dataframe.DataFrame
```

```
[18]  df.select('age').show()
```

```
+----+
| age|
+----+
|null|
|  30|
|  19|
+----+
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ Multiple Columns

```
[20]  df.select(['age','name'])
```

```
DataFrame[age: int, name: string]
```

```
[21]  df.select(['age','name']).show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

■ Creating new columns

```
[22]  # Adding a new column with a simple copy
      df.withColumn('newage',df['age']).show()
```

```
+----+-------+------+
| age|   name|newage|
+----+-------+------+
|null|Michael|  null|
|  30|   Andy|    30|
|  19| Justin|    19|
+----+-------+------+
```

PySpark DataFrame 실습

spark_dataframe.ipynb

■ More complicated operations to create new columns

```
[25]  df.withColumn('doubleage',df['age']*2).show()
```

```
+----+-------+---------+
| age|   name|doubleage|
+----+-------+---------+
|null|Michael|     null|
|  30|   Andy|       60|
|  19| Justin|       38|
+----+-------+---------+
```

```
[26]  df.withColumn('add_one_age',df['age']+1).show()
```

```
+----+-------+-----------+
| age|   name|add_one_age|
+----+-------+-----------+
|null|Michael|       null|
|  30|   Andy|         31|
|  19| Justin|         20|
+----+-------+-----------+
```

https://sparkbyexamples.com/pyspark-tutorial/

PySpark ML 실습

spark_linear_regression.ipynb

spark_logistic_regression.ipynb

spark_tree_model.ipynb

PySpark Tutorial

https://sparkbyexamples.com/pyspark-tutorial/

- What is PySpark
  - Introduction
  - Who uses PySpark
  - Features
  - Advantages
- PySpark Architecture
- Cluster Manager Types
- Modules and Packages
- PySpark Installation on windows
- Spyder IDE & Jupyter Notebook
- PySpark RDD
  - RDD creation
  - RDD operations
- PySpark DataFrame
  - Is PySpark faster than pandas?
  - DataFrame creation
  - DataFrame Operations
  - DataFrame external data sources
  - Supported file formats
- PySpark SQL
- PySpark Streaming
  - Streaming from TCP Socket
  - Streaming from Kafka
- PySpark GraphFrames
  - GraphX vs GraphFrames

Thank you

kgpark88@gmail.com