Spark for Data Engineers with Databrick

Tomaž Kaštrun

DataPlatformGeeks

www.DataPlatformGeeks.com | www.SQLServerGeeks.com

@TheDataGeeks | @SQLServerGeeks

About

- BI Developer | data analyst | data scientist
- SQL Server, SAS, R, Python, C#, SAP, SPSS
- 20years experience MSSQL, DEV, BI, DM
- Frequent community speaker
- Avid coffee drinker & bicycle junkie

http://tomaztsql.wordpress.com
tomaz.kastrun@gmail.com
@tomaz_tsql
/in/tomaztsql
http://github.com/tomaztk
https://mvp.microsoft.com/PublicProfile/5002196
https://medium.com/@tomazkastrun

Demo: tomaztk/Spark-for-data-engineers: Apache Spark for data engineers (github.com)

Agenda

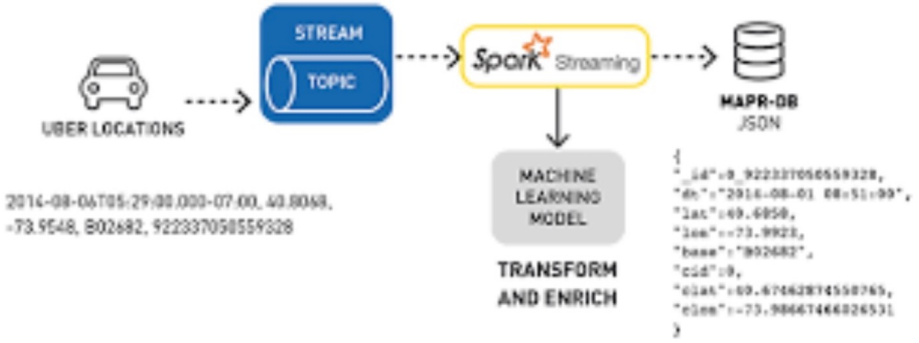
- What is Spark and why Databricks
- Spark in everyday life
- Modes of execution in Spark
- RDD
- ETL, DataFrames, Datasets and Spark API
- Python and Scala

On Apache Spark

- Fast, expressive, general-purpose in-memory cluster computing framework compatible with Apache Hadoop and built around speed, ease of use and streaming analytics
  Faster and easier than Hadoop MapReduce*
  Large community and 3rd party libraries
- Provides high-level APIs (Java, Scala, Python, R) Supports variety of workloads
  - interactive queries, streaming, machine learning and graph processing

Spark Use-Cases

- Logs processing (Uber)
- Event detection and real-time analysis
- Interactive analysis
- Latency reduction
- Advanced ad-targeting (Yahoo!)
- Recommendation systems (Netflix, Pinterest)
- Fraud detection
- Sentiment analysis (Twitter)



Spark Use-Cases

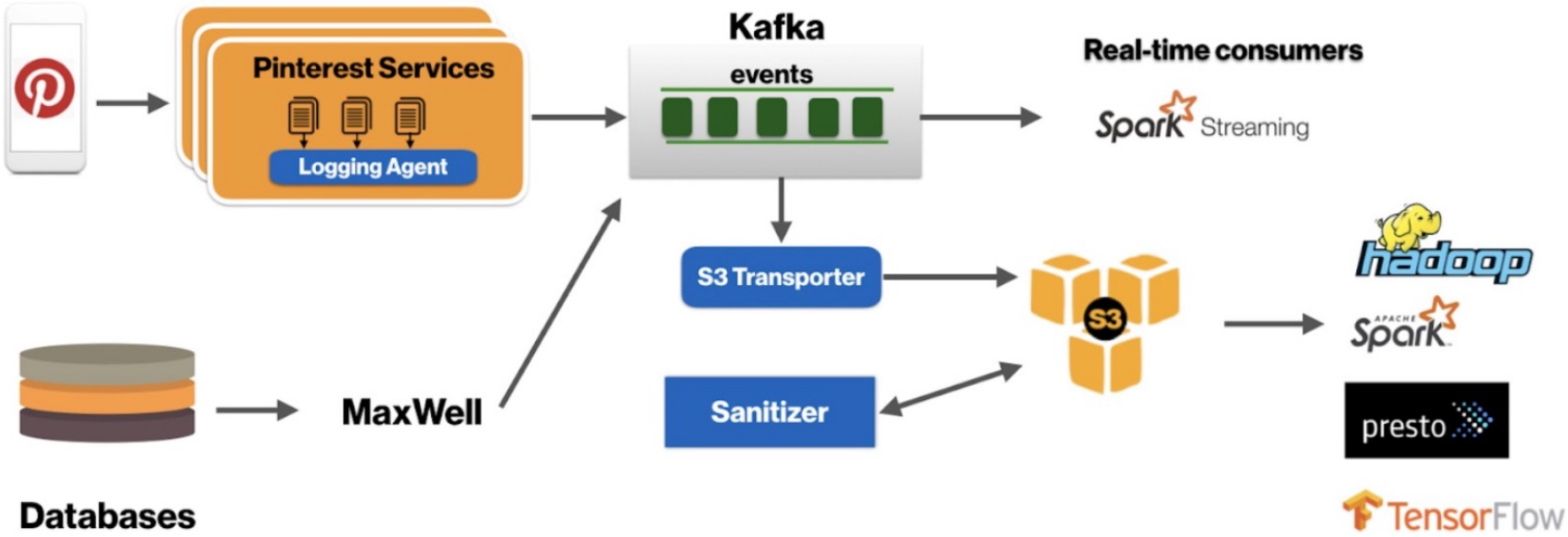
- Logs processing (Uber)
- Event detection and real-time analysis
- Advanced ad-targeting (Yahoo!)

Spark Use-Cases

- Logs processing (Uber)
- Event detection and real-time analysis
- Interactive analysis
- Latency reduction
- Advanced ad-targeting (Yahoo!)
- Recommendation systems (Netflix, Pinterest)
- Fraud detection
- Sentiment analysis (Twitter)

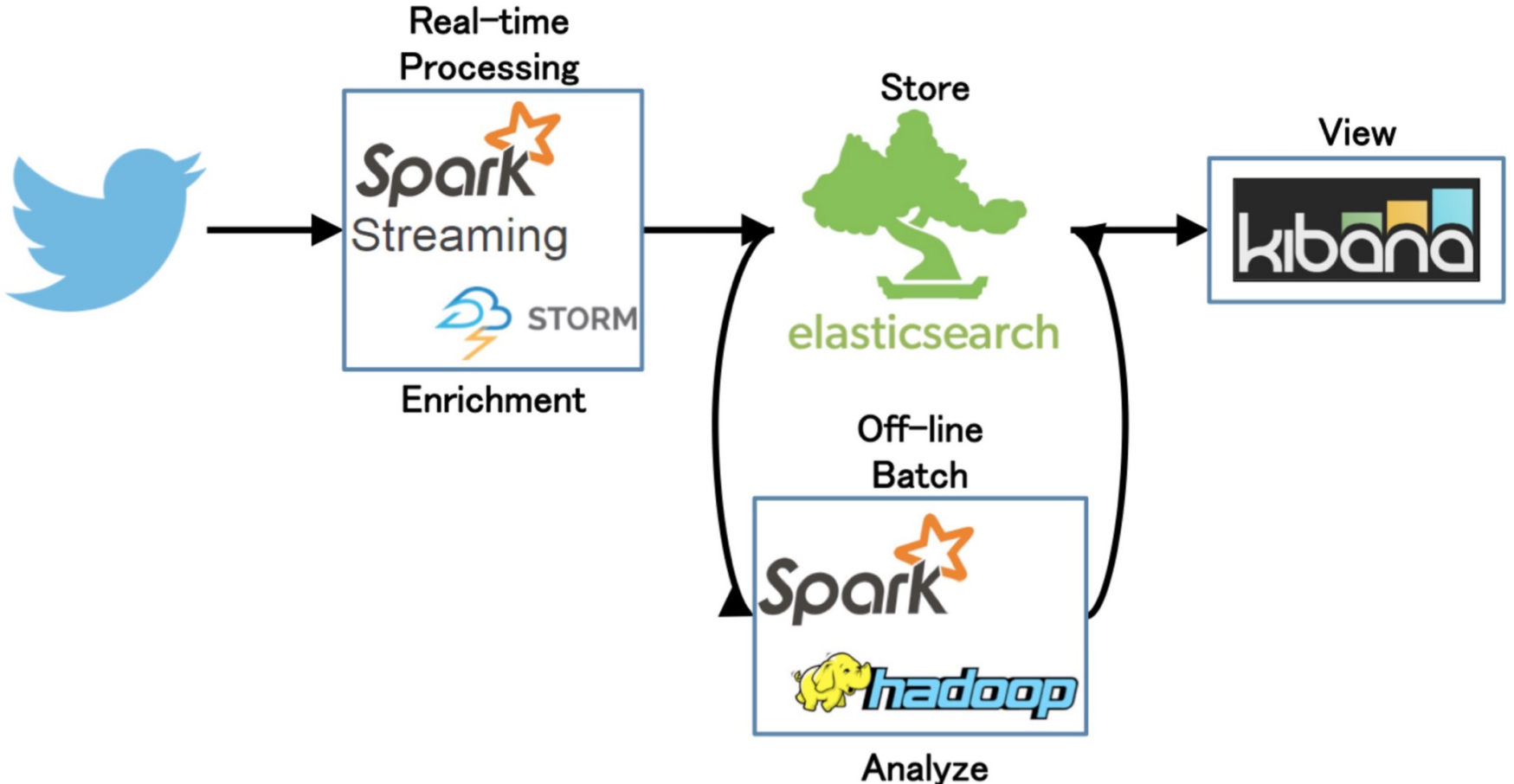
Spark Use-Cases

- Recommendation systems (Netflix, Pinterest)

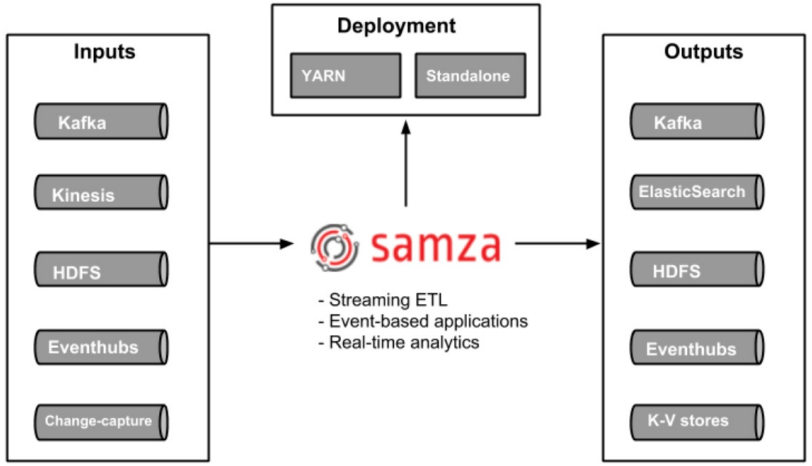


Spark Use-Cases

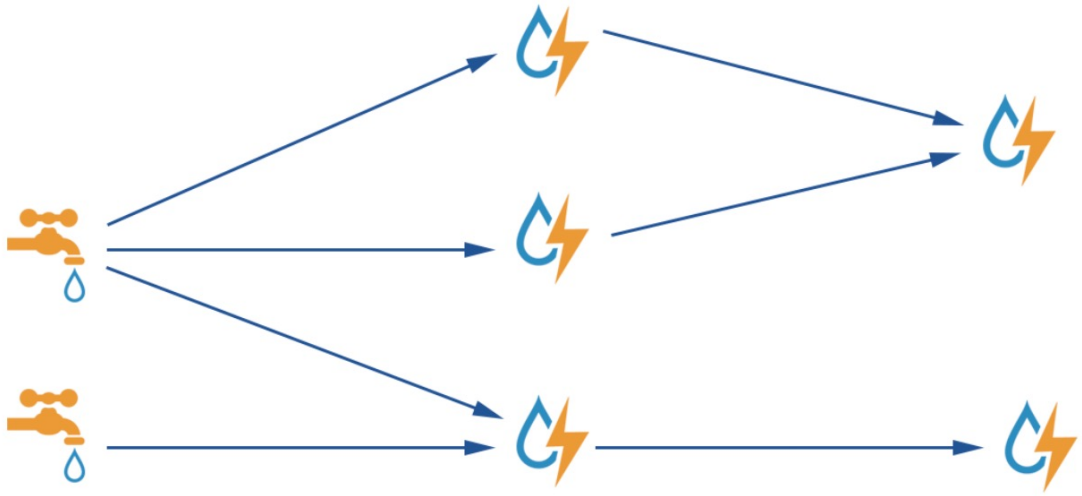
- Sentiment analysis (Twitter)



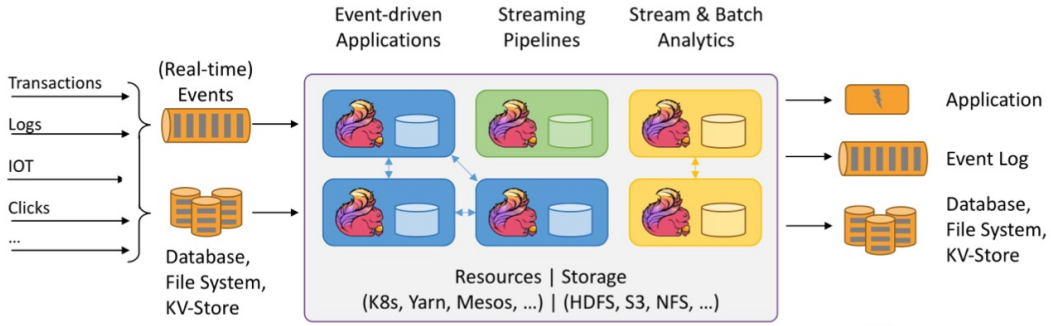
Apache Spark “real-time”



- Apache Samza (library/framework)
- Apache Storm (real-time stream processing)
- Apache Flink (native streaming support for all workloads)



APACHE STORM™



Flink

Apache Spark 3.x

- Latest release – v3.3.0. (Jun 16, 2022)
- Improvements over Spark 2 (v2.4.1)
  - Python 2 deprecated
  - Adaptive execution of Spark SQL (merging intermediate results among workers)
  - Dynamic partition pruning
  - Support for deep learning (GPU support for Nvidia, AMD, Intel)
  - Better Kubernetes integration
  - Graph features (Morpheus as extension of Cypher, neo4j support)
  - ACID transactions (for Delta Lake storage)
  - Apache Arrow data format integration (columnar format for analytics)

Hadoop MapReduce vs. Apache Spark

Big data frameworks

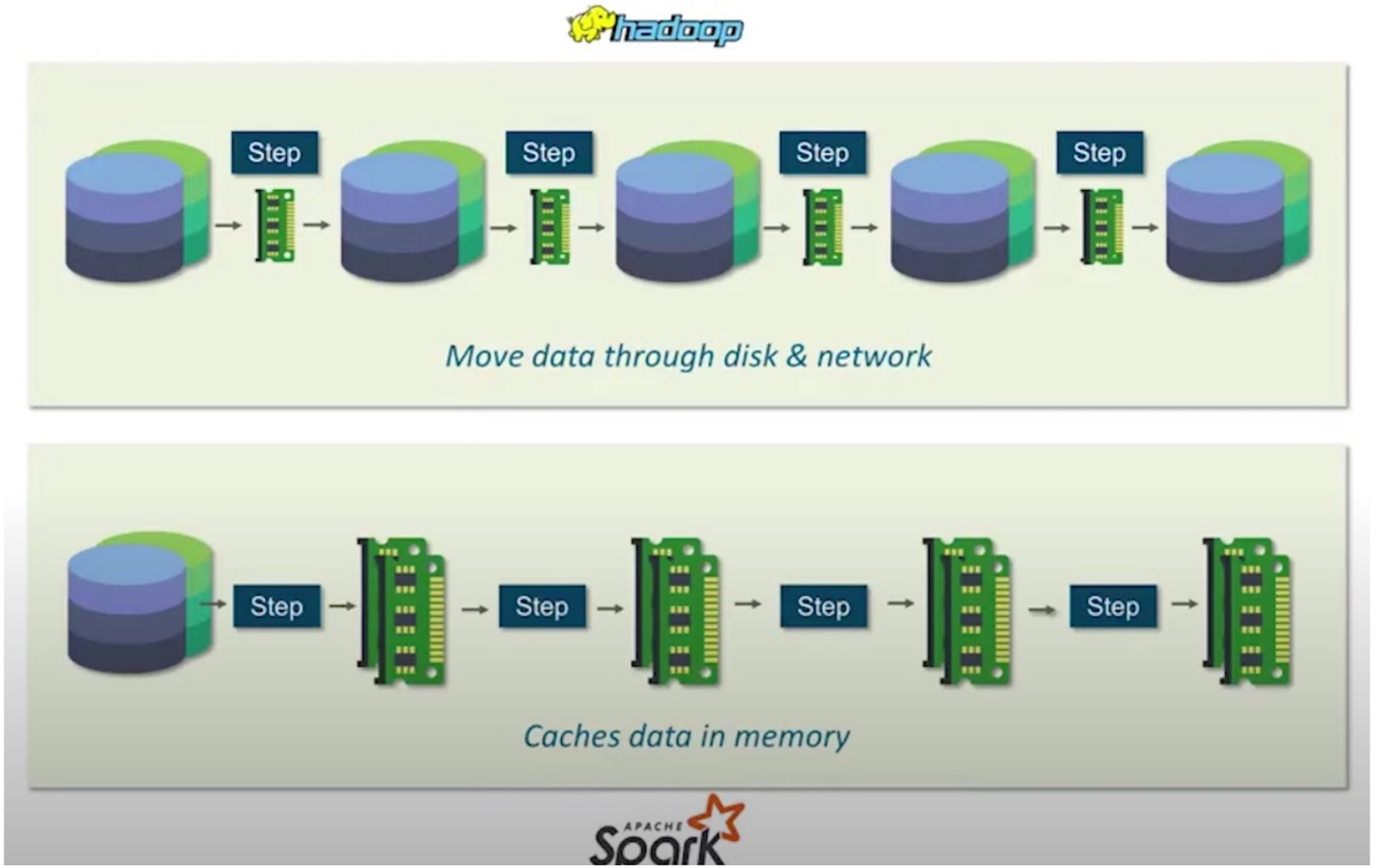
- Performance
- Ease of use
- Costs
- Data processing
- Fault tolerance
- Security

Hadoop

Archival data analysis

Spark

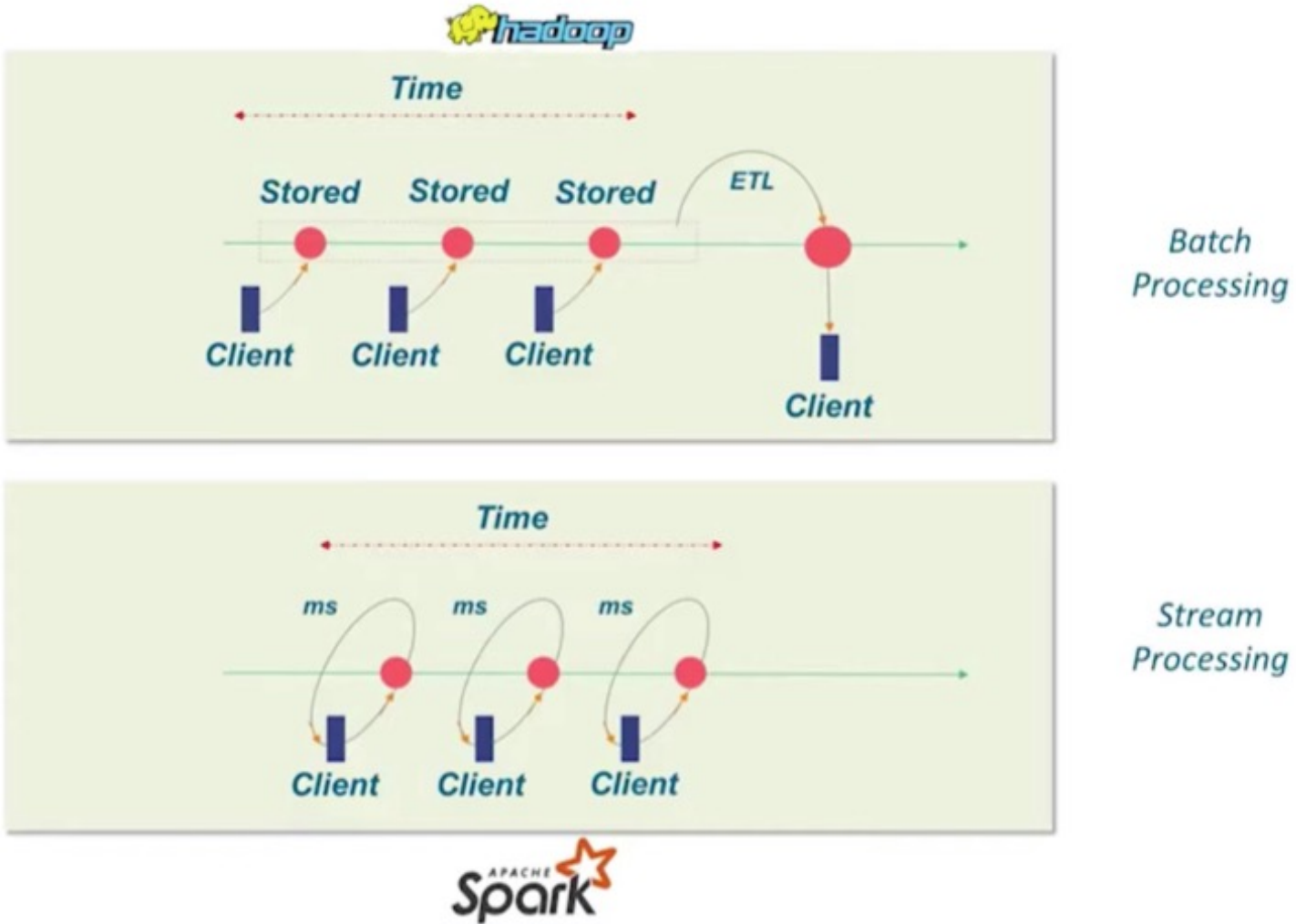
Real-time data analysis



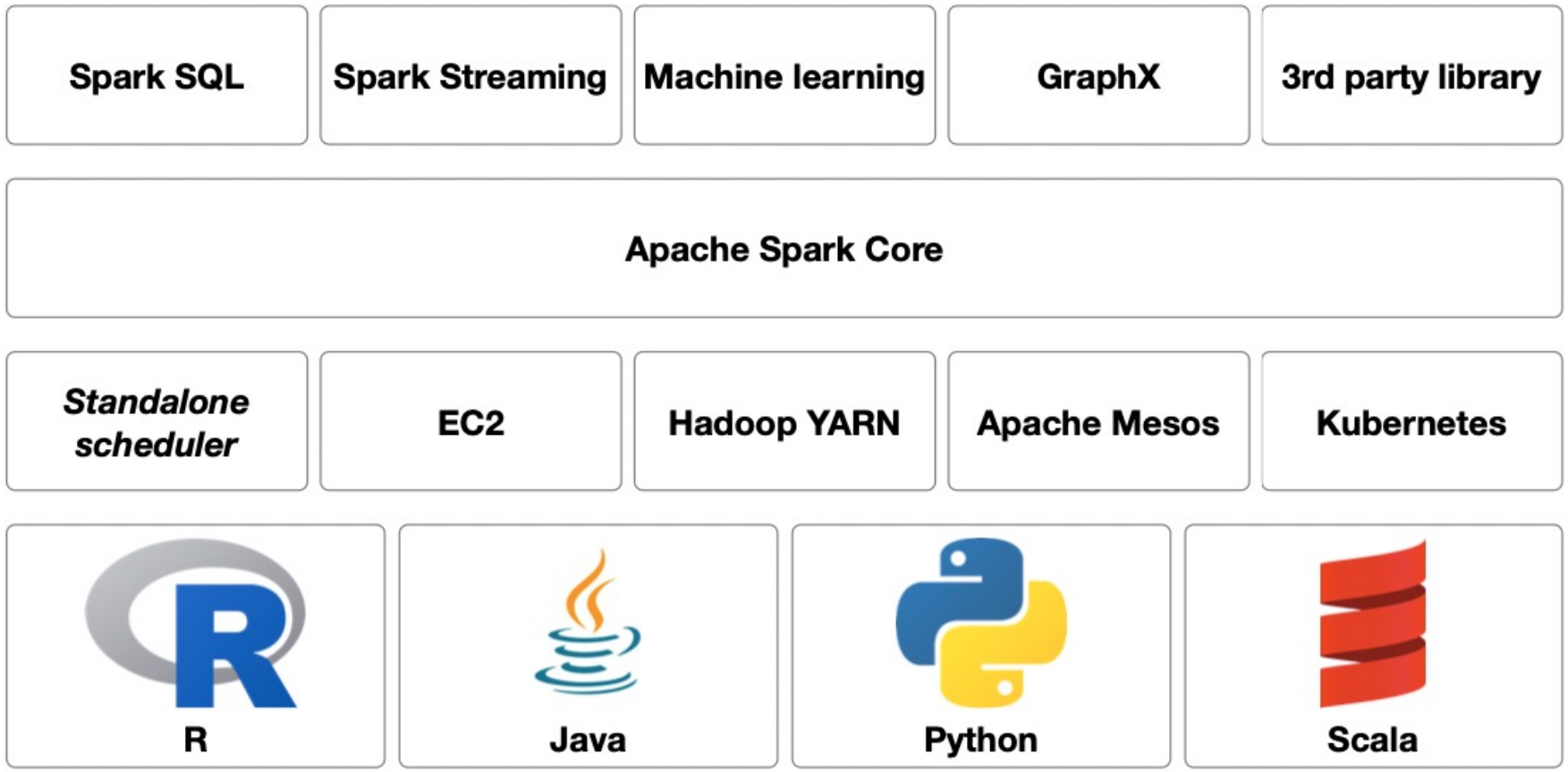
Hadoop MapReduce vs. Apache Spark

Hadoop MapReduce vs. Apache Spark

Hadoop MapReduce vs. Apache Spark



Apache Eco-System

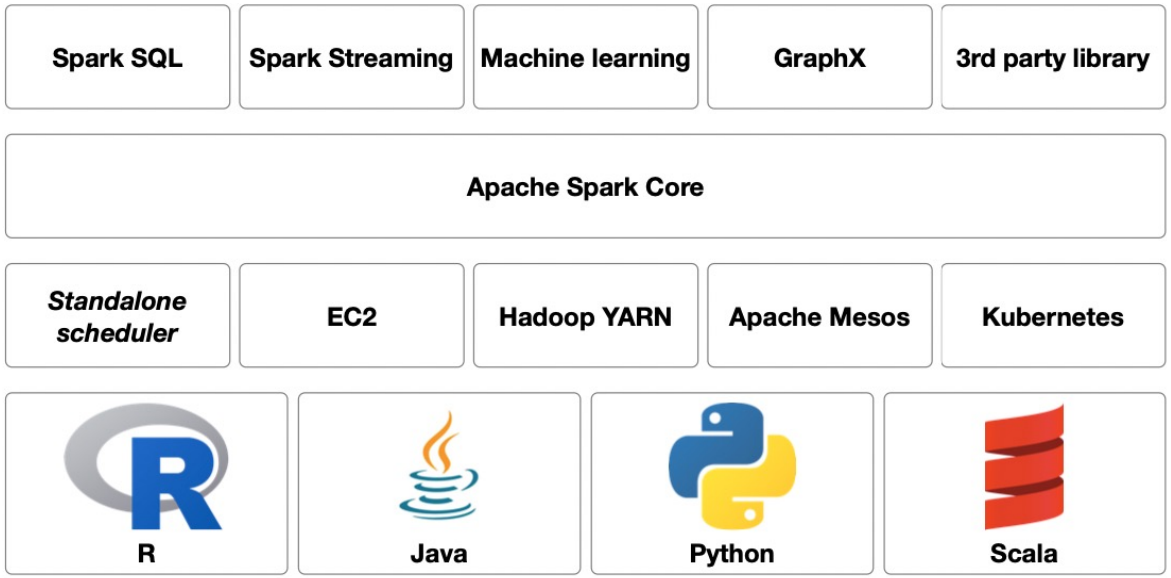


Spark Core functionalities

Core functionalities
- task scheduling
- memory management
- fault recovery
- storage systems interaction

etc.

Basic data structure definitions/abstractions
- Resilient Distributed Data sets (RDDs)
  main Spark data structure
- Directed Acyclic Graph (DAG)



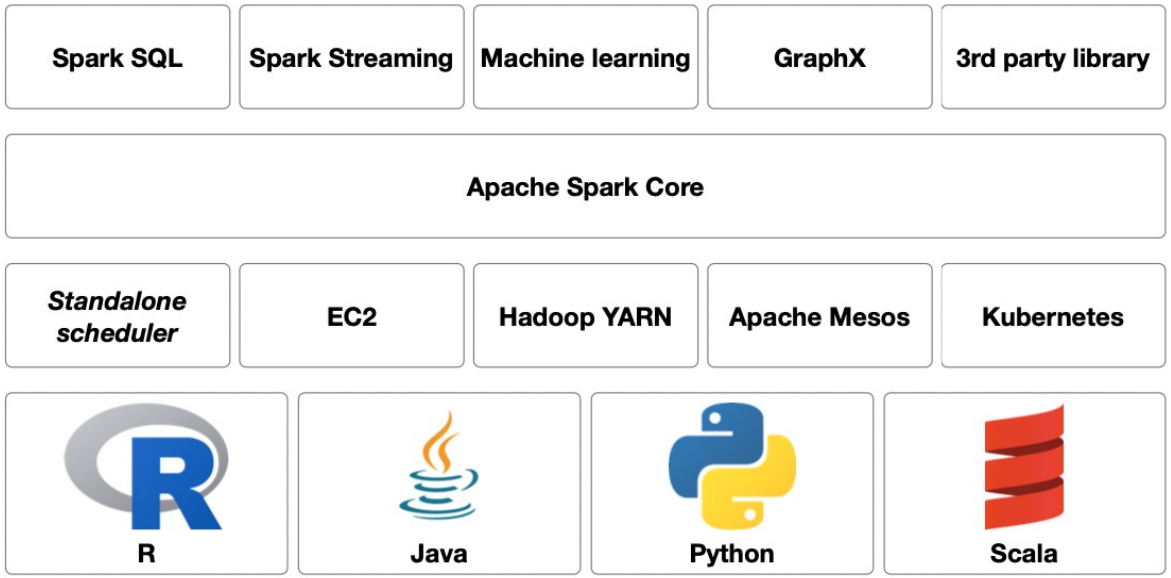
Ecosystem: Spark SQL

Structured data manipulation
- Data Frames definition

Table-like data representation
- RDDs extension
- Schema definition

SQL queries execution

Native support for schema-based data
- Hive, Paquet, JSON, CSV



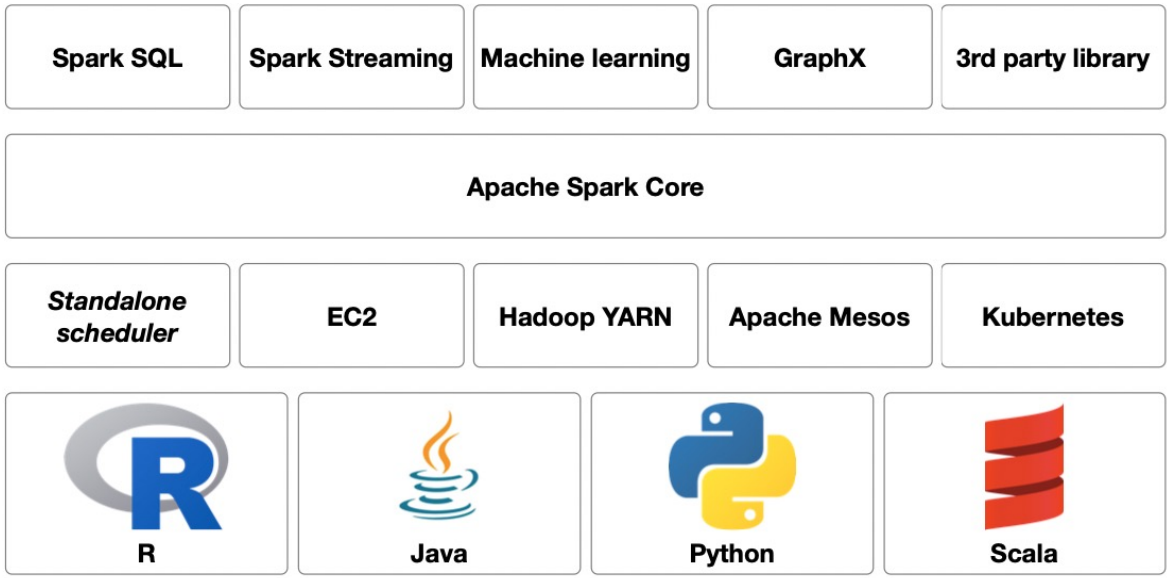
Ecosystem: Spark Streaming

Data analysis of streaming data
- e.g. tweets, log messages, SCADA

Features of stream processing
- High-throughput
- Fault-tolerant
- End-to-end
- Exactly once

High-level abstraction of a discretized stream
- Stream represented as a sequence of RDD

With Spark 2.3.x and above continuous processing
- End-to-end low latency (< 1ms)

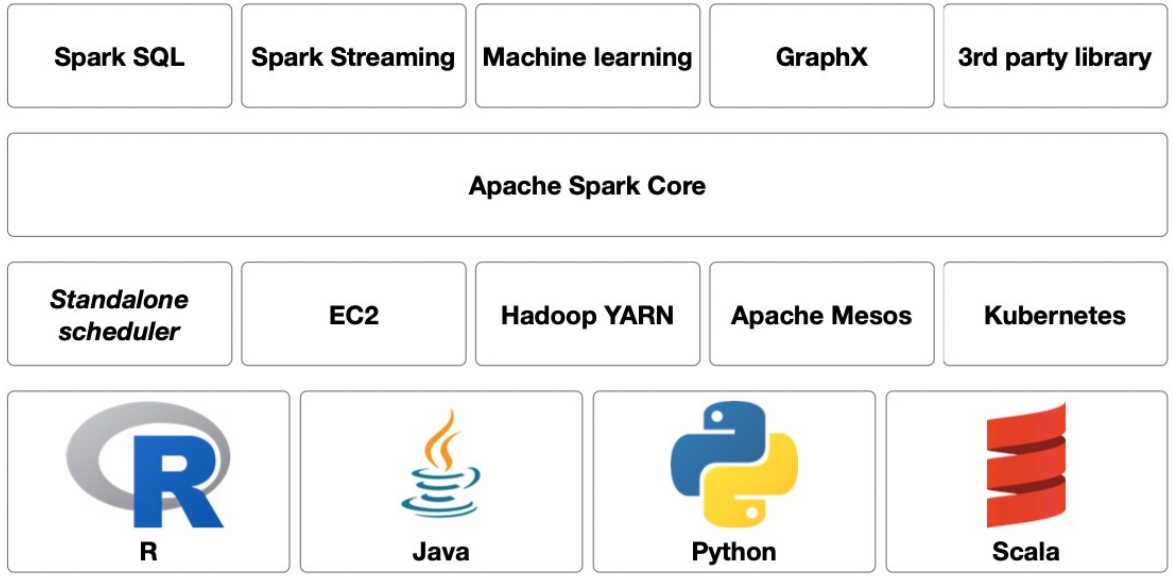


Ecosystem: Spark MLlib for Machine Learning

Common ML functionalities
- ML Algorithms
  - common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization
  - feature extraction, transformation, dimensionality reduction, and selection
- Pipelines
  - tools for constructing, evaluating, and tuning ML Pipelines
- Persistence
  - saving and load algorithms, models, and Pipelines
- Utilities
  - linear algebra, statistics, data handling, etc.

Two APIs
- RDD-based API (*spark.mllib package*)
- Spark 2.0+, DataFrame-based API (*spark.ml package*)

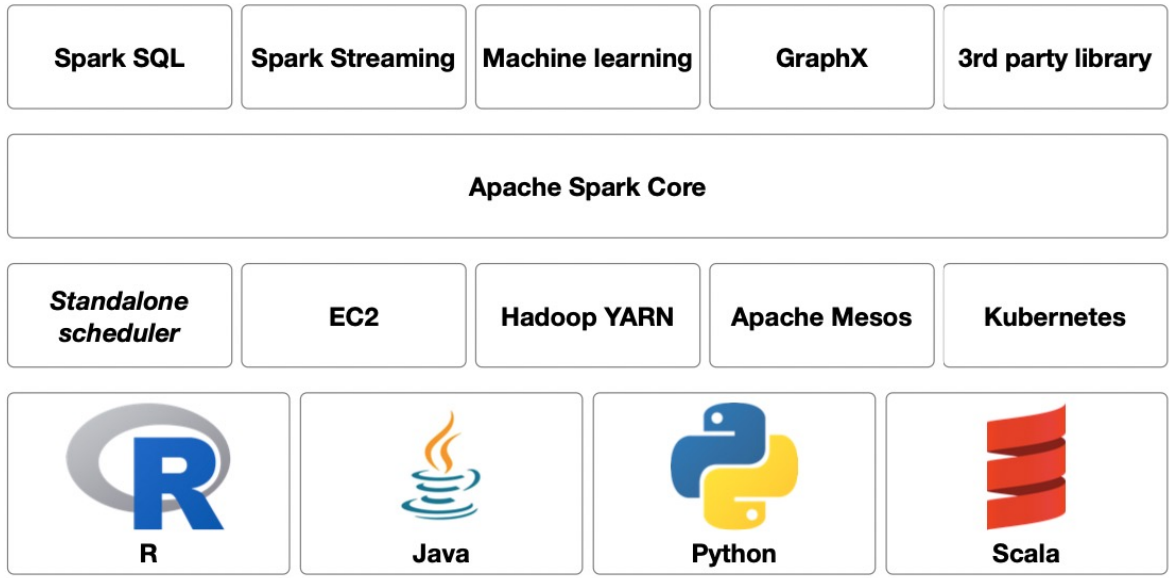
Methods scale out across the cluster by default

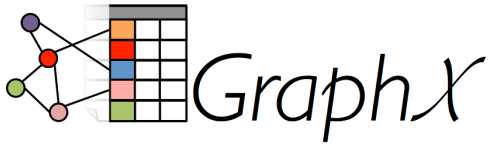


Ecosystem: GraphX

Support for graphs and graph-parallel computation
- Extension of RDDs (Graph)
- Direct multigraph with properties on vertices and edges

Graph computation operators
- subgraph, joinVertices, and aggregateMessages, etc.
- Pregel API support





Spark Modes

Spark operates in 4 different modes:

- **Standalone Mode:** Here all processes run within the same JVM process.
- **Standalone Cluster Mode:** In this mode, it uses the Job-Scheduling framework in-built in Spark.
- **Apache Mesos:** In this mode, the work nodes run on various machines, but the driver runs only in the master node.
- **Hadoop YARN:** In this mode, the drivers run inside the application's master node and is handled by YARN on the Cluster.

Spark Context

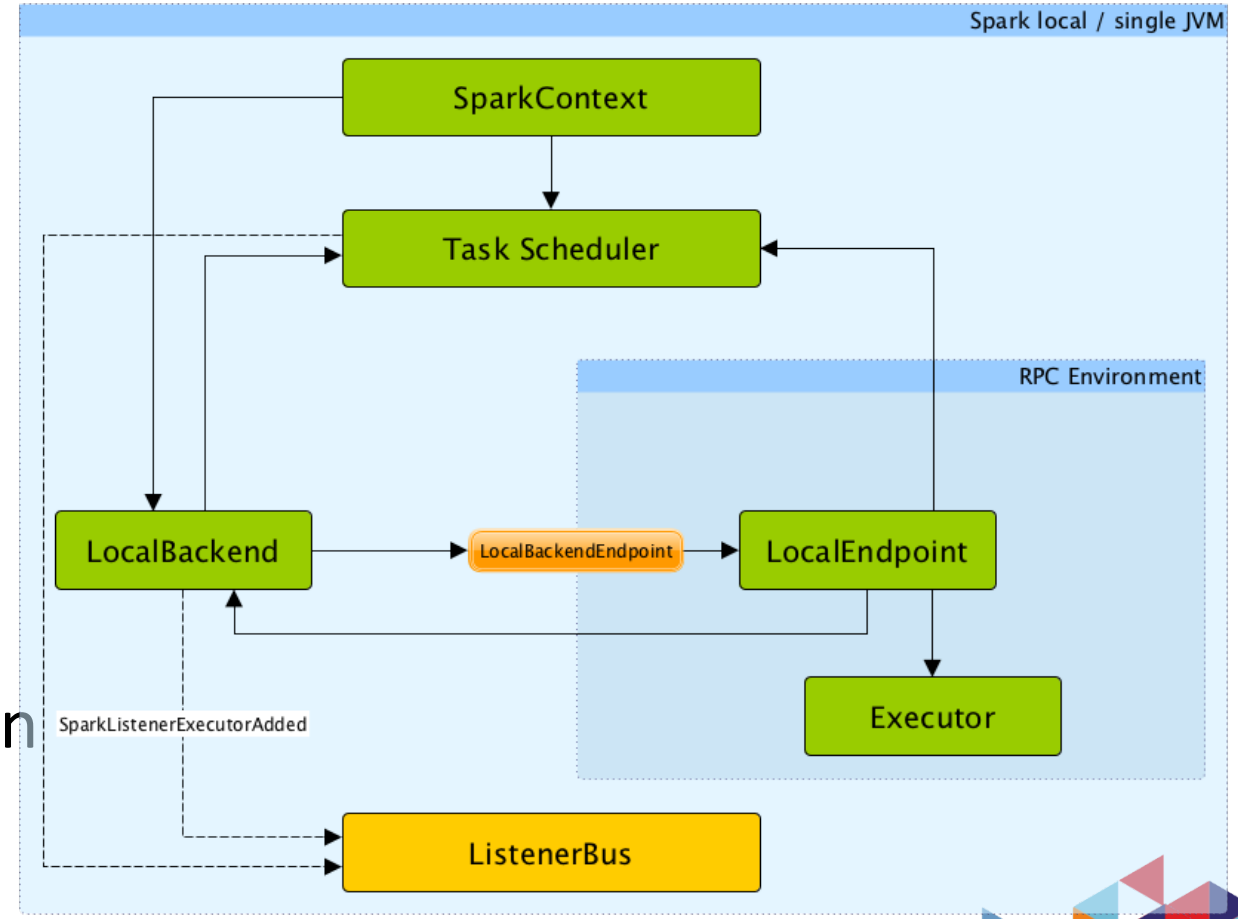
- driver Spark application which communicates the user commands to the Spark Workers
- `SparkContext` is an object, which coordinates with the cluster manager about the resources required for execution and the actual tasks that need to be executed
- Submitting spark applications: Submitting Applications - Spark 3.2.1 Documentation (apache.org)

Execution modes

- Local mode
  - „Pseudo-cluster“ ad-hoc setup using script
- Cluster mode
  - Running via cluster manager
- Interactive mode
  - Direct manipulation in a shell (*pyspark*, *spark-shell*)

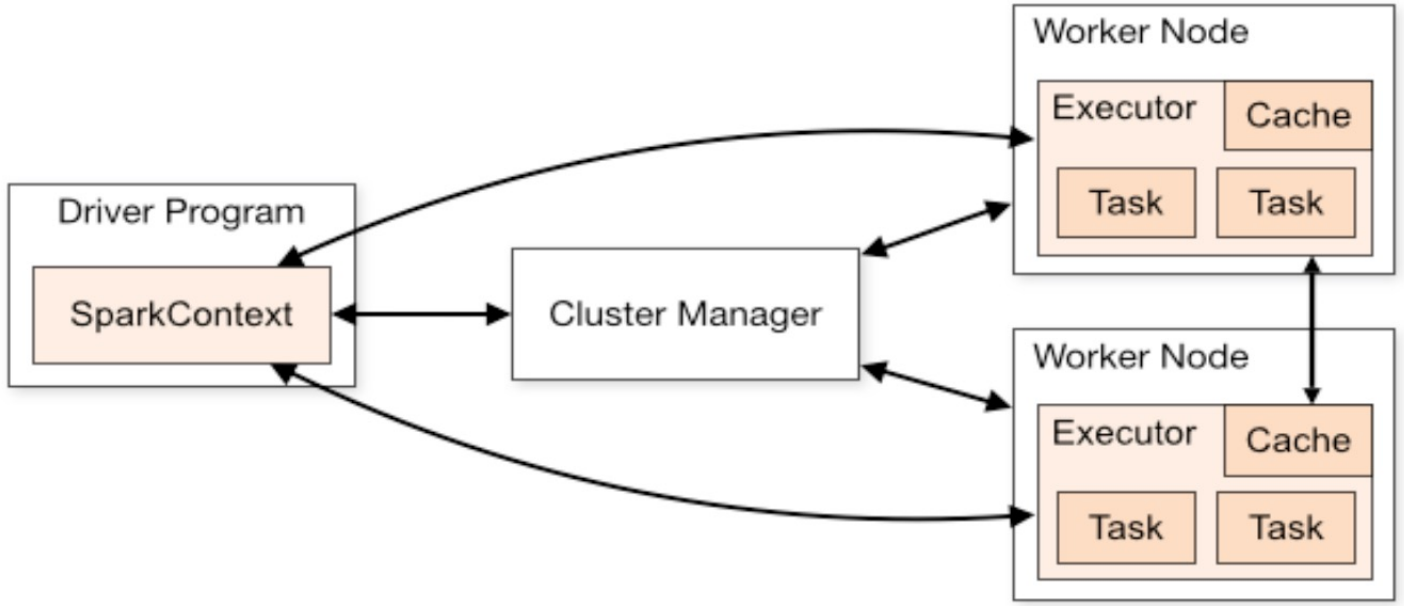
Spark execution modes Local mode

- Non-distributed single-JVM deployment mode
- Spark library spawns (in a JVM)
  - driver
  - scheduler
  - master
  - executor
- Parallelism is the number of threads defined by a parameter N in a spark master URL
  - *local[N]*

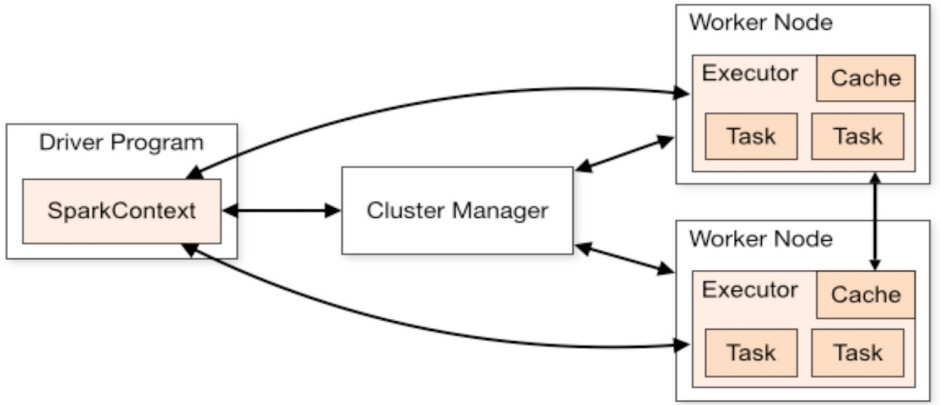


Spark execution modes Cluster mode

- Deployment on a private cluster
  - Apache Mesos
  - Hadoop YARN
  - Kubernetes
  - Standalone mode, ...

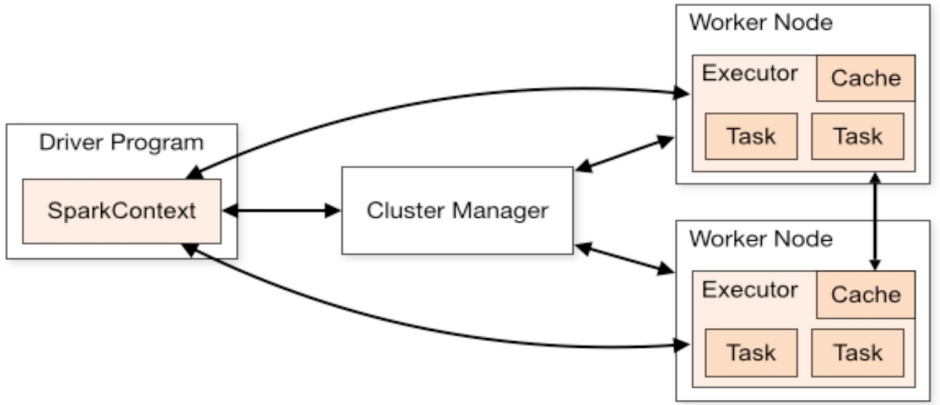


Spark execution modes Cluster mode



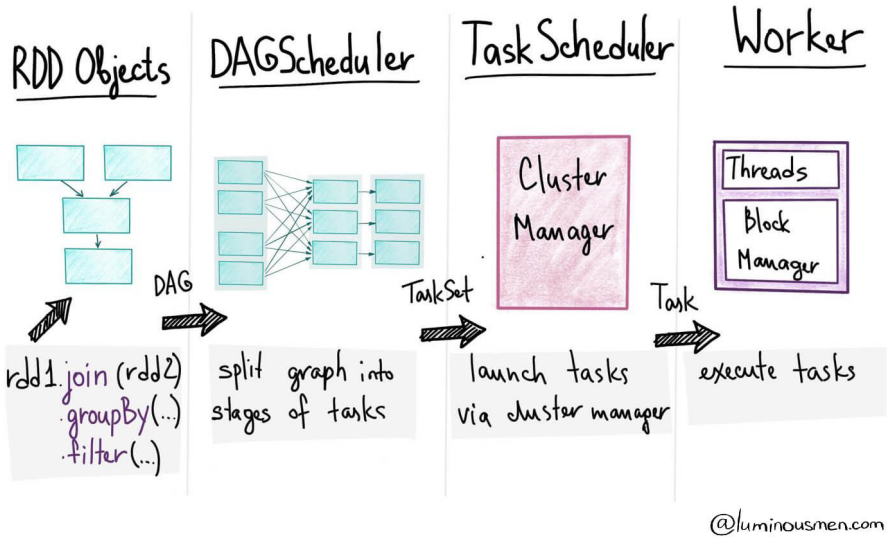
- Components
  - Worker
    - Node in a cluster, managed by an executor
    - Executor manages computation, storage and caching
  - Cluster manager
    - Allocates resources via SparkContext with Driver program
  - Driver program
    - A program holding SparkContext and main code to execute in Spark
    - Sends application code to executors to execute
    - Listens to incoming connections from executors

Spark execution modes Cluster mode



- Deploy modes (*standalone clusters*)
  - Client mode (default)
    - Driver runs in the same process as client that submits the app
  - Cluster mode
    - Driver launched from a worker process
    - Client process exits immediately after application submission

Spark execution process



1. Data preparation/import
   - RDDs creation – i.e. parallel dataset with partitions
2. Transformations/actions definition*
   - Creation of tasks (units of work) sent to one executor
   - Job is a set of tasks executed by an action*
3. Creation of a directed acyclic graph (DAG)
   - Contains a graph of RDD operations
   - Defition of stages – set of tasks to be executed in parallel (i.e. at a partition level)
4. Execution of a program (application)

Shell vs.Cluster mode

- Shell Is interactive
- Cluster is for connecting into private network with several machines
- In cluster mode, it’s called submitting applications

Submitting applications

Pseudo code

./bin/spark-submit ****
--class *<main-class>* ****
--master *<master-url>* ****
--deploy-mode *<deploy-mode>* ****
--conf *<key>=<value>* ****
... # other options <application-jar> ****
[application-arguments]

Applications: Locally, in Standalone cluster

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000
```

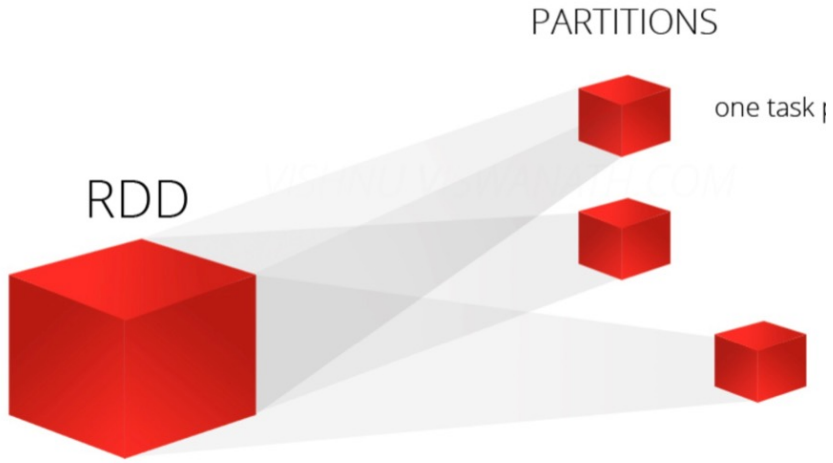
Applications: Mesos and Kubernetes

```
# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000
```

```
# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

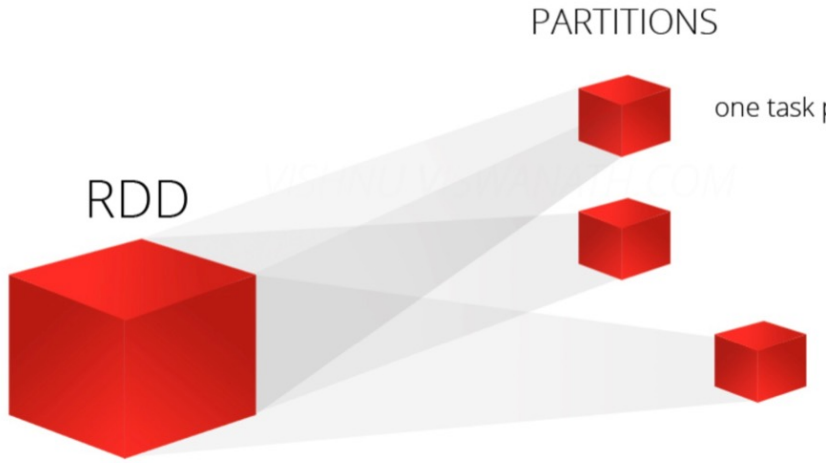
Spark Programming concepts - RDD

- RDD – Resilient distributed datases
- Basic data representation in Spark
- A distributed collection of items – partitions
  - Enables parallel operations
- RDDs are immutable (”read-only“)
- Fault-tolerant
  - Achieving 100% data preseration, every function creates new partition
- Caching and different storage levels possible
- Supports a set of Spark transformation and actions



Spark Programming concepts - RDD

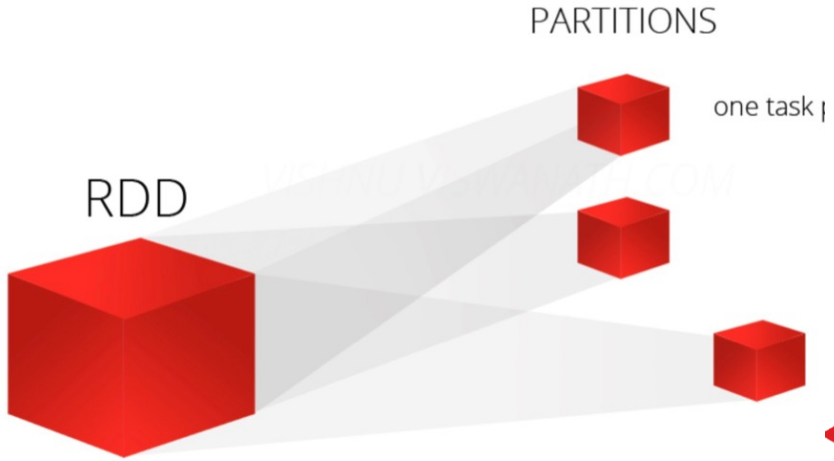
- Computations are expressed using
  - Creation of new RDDs
  - Transforming existing RDDs
  - Operations on RDDs to compute results (actions!)



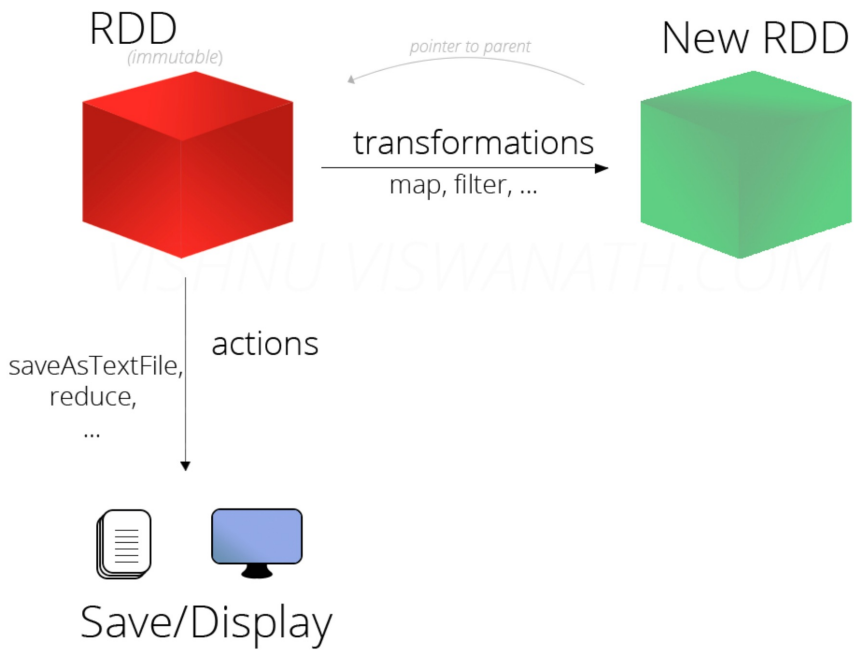
- Distributes the data within RDDs across nodes (executors) in the clust and parallelizes the calculations

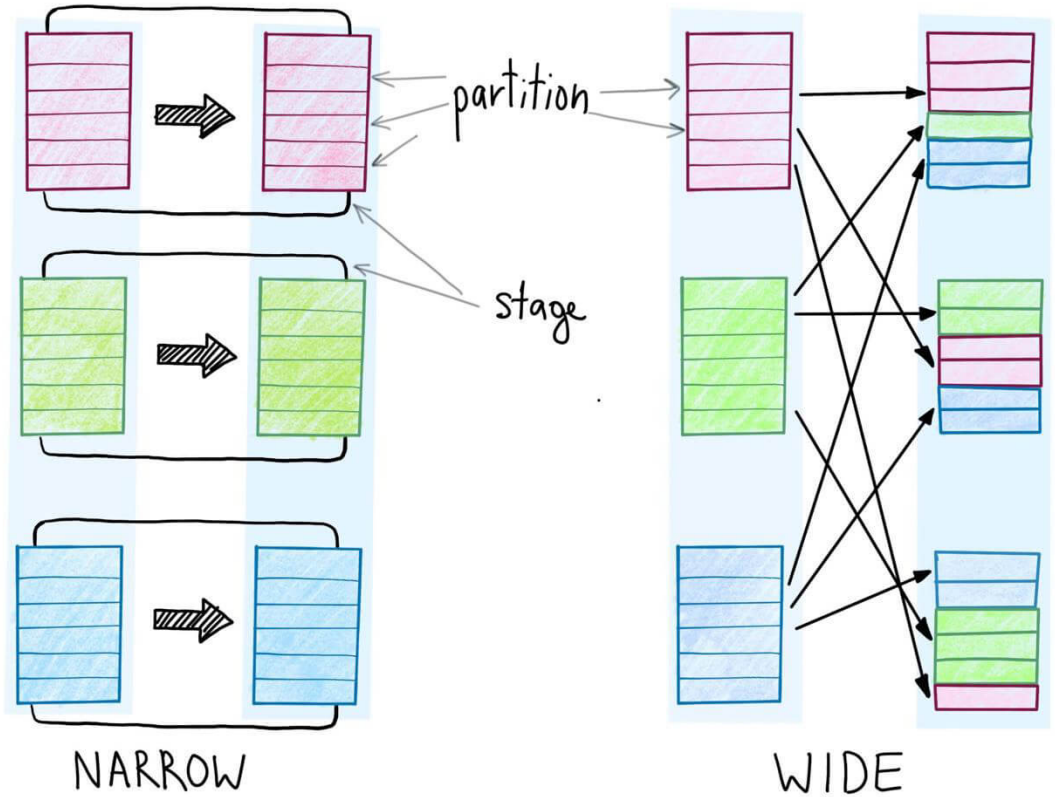
RDD operations

- RDDs enable following operations
  - **transformations**
    - lazy operations that return a new RDD from input RDDs
    - narrow or wide types
    - examples: map, filter, join, groupByKey...
  - **actions**
    - return a result or write to storage,
      execute transformations
    - examples: count, collect, save



RDD: Transformation and actions





RDD Transformation vs. Actions

 = easy  = medium

Essential Core & Intermediate Spark Operations

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

ACTIONS

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

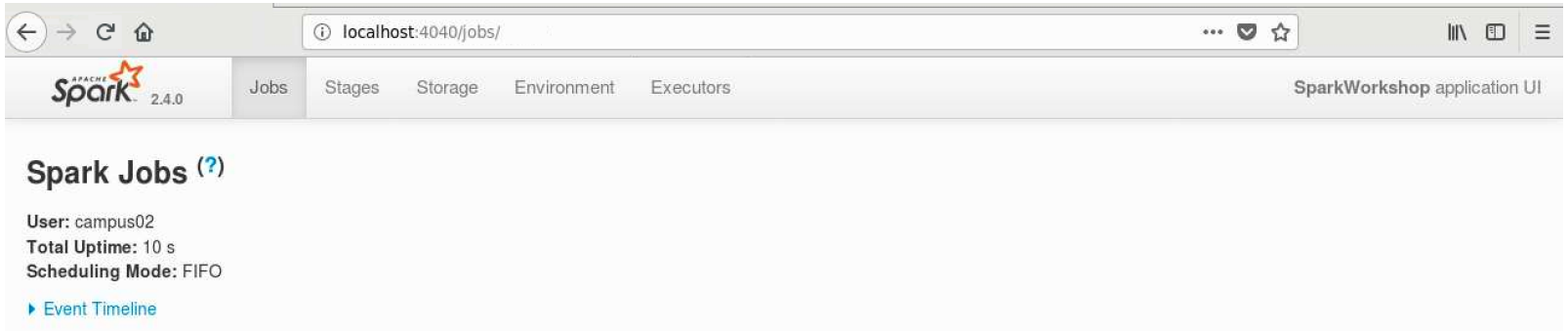
Source: URL

RDD in Spark and explore Web UI

We will use pySpark library interactively

```
import pyspark
sc = pyspark.SparkContext(appName=,‘SparkWorkshopDSP2022', master='local[1]')
```

```
2020-09-15 16:28:01 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java clas
ses where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
2020-09-15 16:28:02 WARN  Utils:66 - Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
>>>
```



RDD in Spark and explore Web UI

Creation of RDDs

- From a collection

```
rdd1 = sc.parallelize([('John', 23), ('Mark', 11), ('Jenna', 44), ('Sandra', 61)])
```

- From a file

```
rdd2 = sc.textFile('data/IMDB Dataset.csv‘)
```

Basic transformations map(), filter(), flatMap()

```
older = rdd1.filter(lambda x: x[1] > 18)
anonymized = older.map(lambda x: (x[0][0], x[1]))

birthdays = rdd1.map(lambda x: list(range(1, x[1]+1)))
birthdays = rdd1.flatMap(lambda x: list(range(1, x[1]+1)))

birthdays.map(lambda x: (x, 1)).groupByKey() \
     .mapValues(lambda vals: len(list(vals))).sortByKey()
```

RDD in Spark and explore Web UI

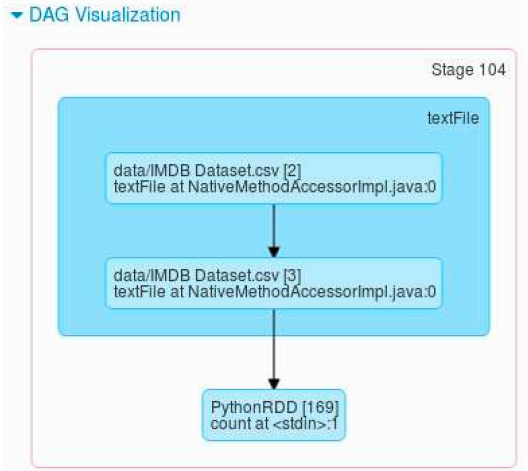
- Further actions, transformations

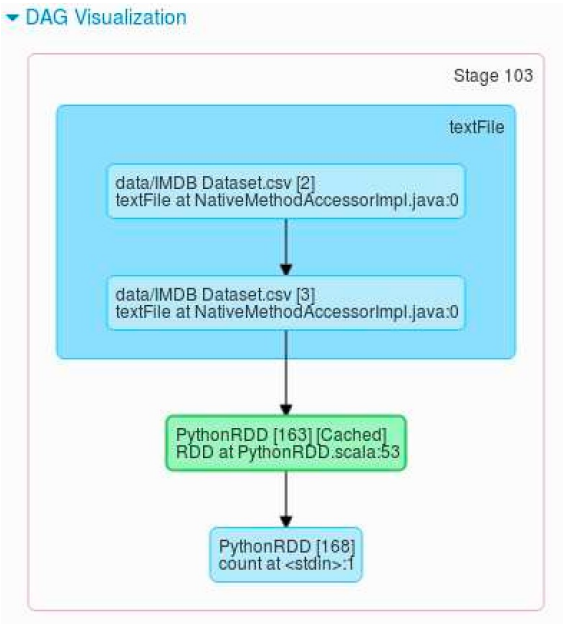
```
rdd2.take(2)

def organize(line):
    data = line.split('",')
    data = data if len(data) == 2 else line.split(',')
    return (data[1], data[0][1:51] + ' ...')
movies = rdd2.filter(lambda x: x !=
'review,sentiment').map(organize)

movies.count() // 50.000
movies = movies.filter(lambda x: x[0] in ['positive', 'negative'])
movies.count() // 45.936
movieCounts = movies.groupByKey().map(lambda x: (x[0], len(x[1])))
```

RDD in Spark and explore Web UI





Tasks (2)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	138	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:53	0,2 s	4 ms	32.1 MB / 25311	
1	139	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:53	0,2 s	4 ms	31.1 MB / 24690	

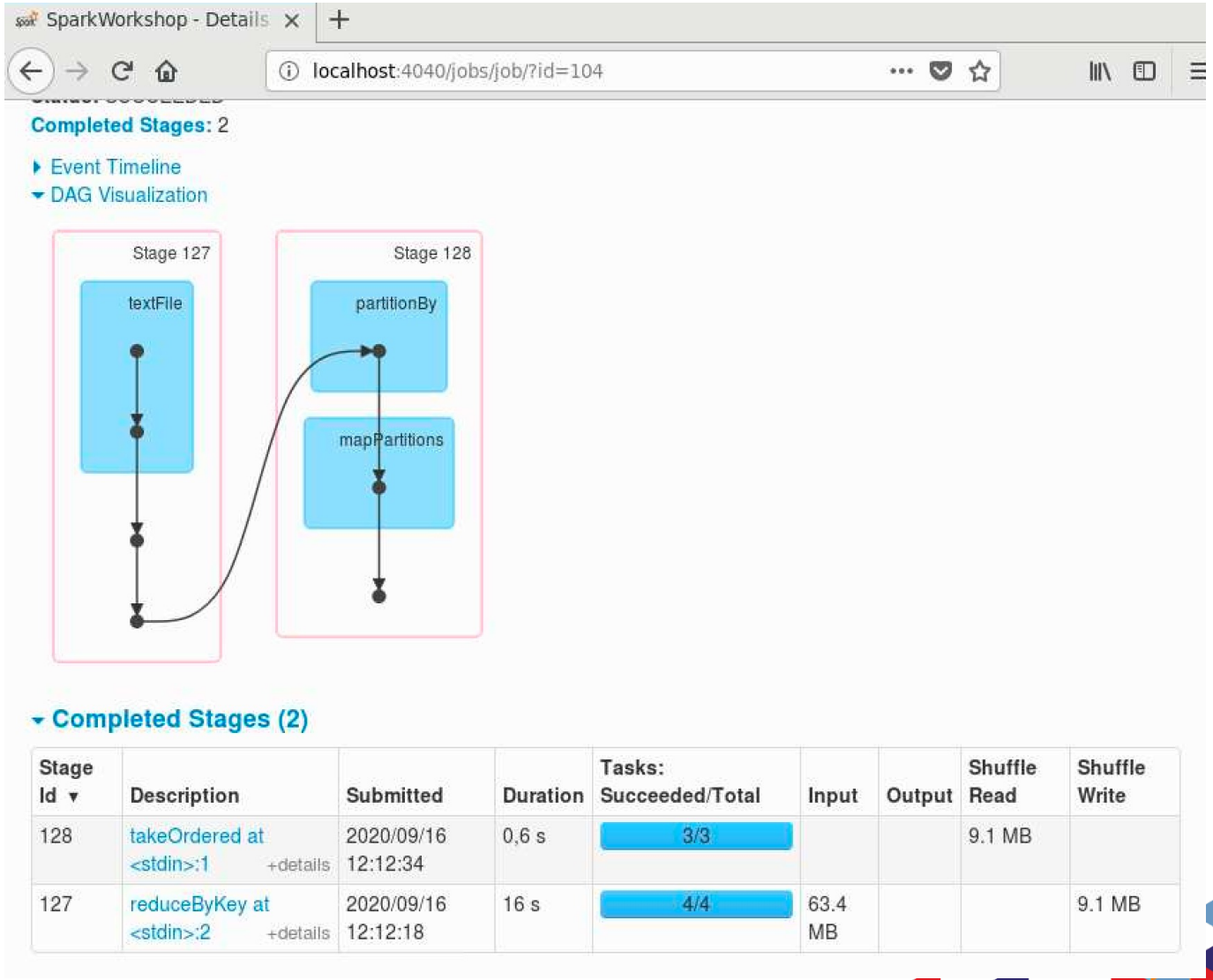
Tasks (2)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	136	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:46	18 ms		469.2 KB / 16	
1	137	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:46	12 ms		453.7 KB / 16	

RDD in Spark and explore Web UI

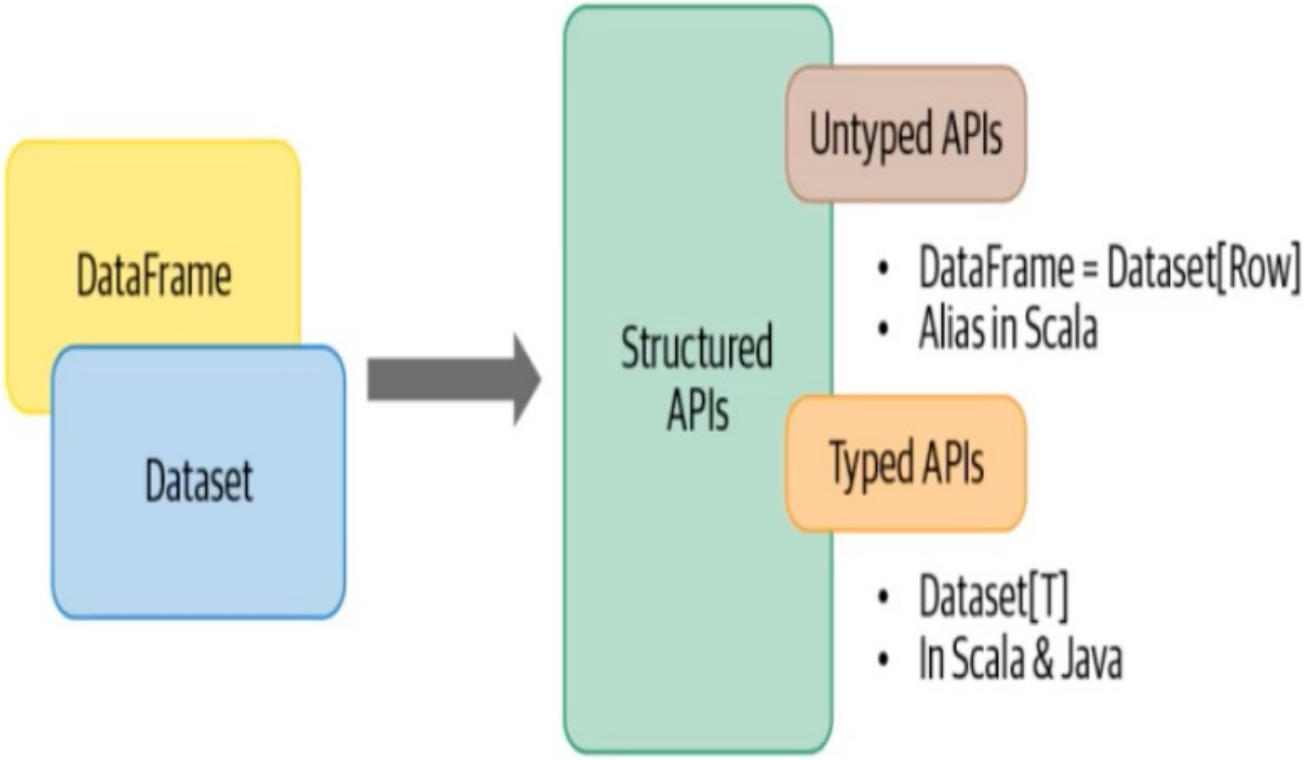
DAG exploration

(admin console result)



Structured API

Stuctured API – to address faster adoption and ease of use in comparison with RDD API



RDD API

```
from pyspark.sql import SparkSession
spark = (SparkSession.builder.appName("SampleRDDAPI").getOrCreate())
```

Command took 0.04 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:06:59 on CL1_DB_Standard

Cmd 87

```
dataRDD = spark.sparkContext.parallelize([("Person",20),("Dog",2),("Person",34),("Person",63), ("Dog", 6)])
```

Command took 0.03 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:04 on CL1_DB_Standard

Cmd 88

```
# Use map and reduceByKey Transformation with lamba function
# to aggreate and compute an average
agesRDD = (dataRDD
           .map(lambda x: (x[0], (x[1], 1)))
            .reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1]))
            .map(lambda x: (x[0], x[1][0]/x[1][1]))
           )
```

Command took 0.05 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:06 on CL1_DB_Standard

Cmd 89

```
for data in agesRDD.collect():
  print(data)
```

▸ (1) Spark Jobs

```
('Person', 39.0)
('Dog', 4.0)
```

RDD API vs SQL

```
1  from pyspark.sql import SparkSession
2  spark = (SparkSession.builder.appName("SampleRDDAPI").getOrCreate())
```

Command took 0.04 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:06:59 on CL1_DB_Standard

Cmd 87

```
1  dataRDD = spark.sparkContext.parallelize([("Person",20),("Dog",2),("Person",34),("Person",63), ("Dog", 6)])
```

Command took 0.03 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:04 on CL1_DB_Standard

Cmd 88

```
1  # Use map and reduceByKey Transformation with lamba function
2  # to aggreate and compute an average
3  agesRDD = (dataRDD
4            .map(lambda x: (x[0], (x[1], 1)))
5             .reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1]))
6             .map(lambda x: (x[0], x[1][0]/x[1][1]))
7            )
```

Command took 0.05 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:06 on CL1_DB_Standard

Cmd 89

```
1  for data in agesRDD.collect():
2    print(data)
```

▸ (1) Spark Jobs

```
('Person', 39.0)
('Dog', 4.0)
```

```
1
2   SELECT
3      AVG(Age) AS Average_age
4      ,typeBinary
5   FROM
6      dbo.RDDAPITable
7   GROUP BY typeBinary
8
9
```

From RDD API to DataFrame and Dataset API

The dataset API is a collection of strongly typed JVM (Java Virtual Machine) data object in Scala and classes in Java

Typed:

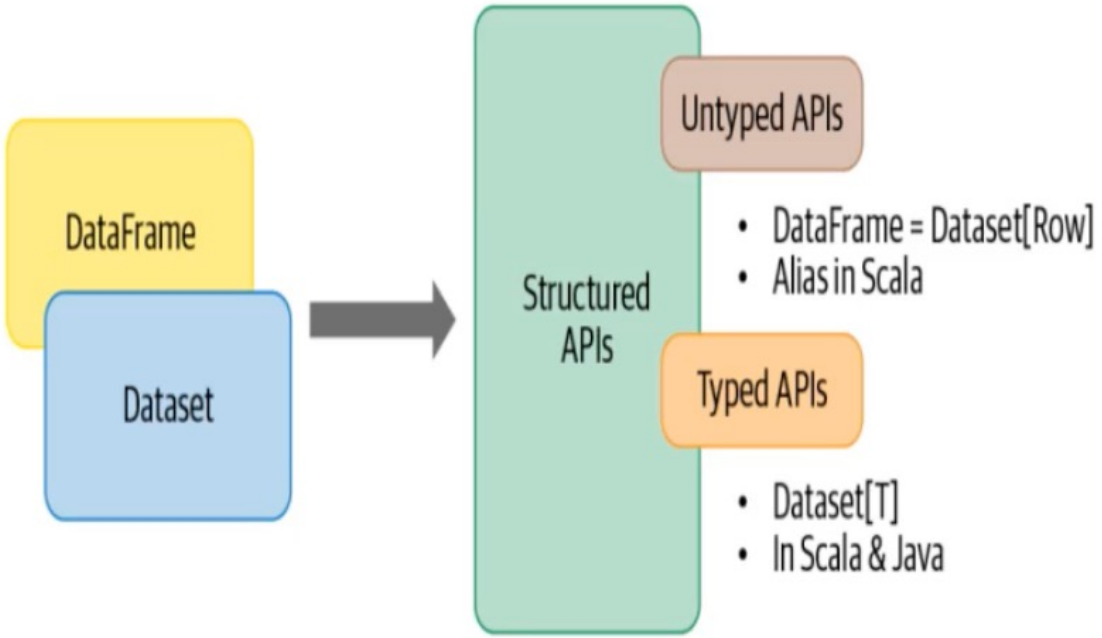
Float avg_age = 23.4

avg_age = “SQLBits” // will return error

Untyped:

avg_age = 23.4

avg_age = “SQLBits” // will NOT(!) return error



The DataFrame is an Untyped API that has support for Python and Scala

Data Engineers prefere JAVA and Scala due to typed nature, because it provides better data integratity.

RDD API

```
from pyspark.sql import SparkSession
spark = (SparkSession.builder.appName("SampleRDDAPI").getOrCreate())
```

Command took 0.04 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:06:59 on CL1_DB_Standard

Cmd 87

```
dataRDD = spark.sparkContext.parallelize([("Person",20),("Dog",2),("Person",34),("Person",63), ("Dog", 6)])
```

Command took 0.03 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:04 on CL1_DB_Standard

Cmd 88

```
# Use map and reduceByKey Transformation with lamba function
# to aggreate and compute an average
agesRDD = (dataRDD
           .map(lambda x: (x[0], (x[1], 1)))
            .reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1]))
            .map(lambda x: (x[0], x[1][0]/x[1][1]))
           )
```

Command took 0.05 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:17:06 on CL1_DB_Standard

Cmd 89

```
for data in agesRDD.collect():
  print(data)
```

▸ (1) Spark Jobs

```
('Person', 39.0)
('Dog', 4.0)
```

DataFrame API

Cmd 90

```
#Create a DataFrame
data_df = spark.createDataFrame([("Person",20),("Dog",2),("Person",34),("Person",63), ("Dog", 6)],["type", "age"])
```

▸ data_df: pyspark.sql.dataframe.DataFrame = [type: string, age: long]

Command took 0.06 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:32:00 on CL1_DB_Standard

Cmd 91

```
avg_df = data_df.groupby("type").avg("age")
avg_df.show()
```

▾ avg_df: pyspark.sql.dataframe.DataFrame

```
type: string
avg(age): double
```

```
+------+--------+
|  type|avg(age)|
+------+--------+
|Person|    39.0|
|   Dog|     4.0|
+------+--------+
```

Command took 0.88 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:35:54 on CL1_DB_Standard

Cmd 92

DataFrame API

- The DataFrame accounts for readable „English“ like code which resembles SQL queries
- The DataFrame allows you to have a structured presenttion of your data into rows and columns like database tables
- The DataFrame allows you to assign schema to address potential the data integrity issues

Cmd 90

```
1  #Create a DataFrame
2  data_df = spark.createDataFrame([("Person",20),("Dog",2),("Person",34),("Person",63), ("Dog", 6)],["type
```

data_df: pyspark.sql.dataframe.DataFrame = [type: string, age: long]

Command took 0.06 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:32:00 on CL1_DB_Standard

```
1  avg_df = data_df.groupby("type").avg("age")
2  avg_df.show()
```

(2) Spark Jobs

avg_df: pyspark.sql.dataframe.DataFrame

```
type: string
avg(age): double
```

```
+------+--------+
|  type|avg(age)|
+------+--------+
|Person|    39.0|
|   Dog|     4.0|
+------+--------+
```

Command took 0.88 seconds -- by tomaz.kastrun@gmail.com at 06/03/2022, 10:35:54 on CL1_DB_Standard

Cmd 92

Demo time ☺

- Working with Spark API
- Spark DataFrame API
- Structured API (SQL)
- Structured API (Python, R)
- Exploring Scala and Python

Recap

- Spark is powerful general-purpose in-memory cluster computing framework
- It supports Python, Scala, R, SQL and Java
- Databricks is build on top of Spark and gives you all data engineering possibilities
- Easy to embeed into existing environment, esp. Azure
- Dataframes are API based for easy, fast data operations

Demo: tomaztk/Spark-for-data-engineers: Apache Spark for data engineers (github.com)

http://tomaztsql.wordpress.com
tomaz.kastrun@gmail.com
@tomaz_tsql
/in/tomaztsql
http://github.com/tomaztk
https://mvp.microsoft.com/PublicProfile/5002196
https://medium.com/@tomazkastrun

Thank You For Your Time!

@TheDataGeeks
@SQLServerGeeks

www.YouTube.com/SQLServerGeeks

www.DataPlatformGeeks.com
www.SQLServerGeeks.com

Follow Us

linkedin.com/showcase/dataplatformgeeks
facebook.com/DataPlatformGeeks
facebook.com/SQLServerGeeks
https://t.me/dataplatformgeeks

Join The Community

linkedin.com/groups/6753546/
facebook.com/groups/theSQLGeeks


http://tomaztsql.wordpress.com


tomaz.kastrun@gmail.com


@tomaz_tsql


/in/tomaztsql


http://github.com/tomaztk


https://mvp.microsoft.com/PublicProfile/5002196


https://medium.com/@tomazkastrun