# Apache Spark Through Email

Markus Dale, medale@asymmetrik.com

July 2024

# Intro, Slides And Code

- Slides: https://github.com/medale/spark-mail/blob/master/presentation/ApacheSparkThroughEmail.pdf
- Spark Code Examples: https://github.com/medale/spark-mail/
  - README.md describes how to get and parse Enron email dataset

# Goals

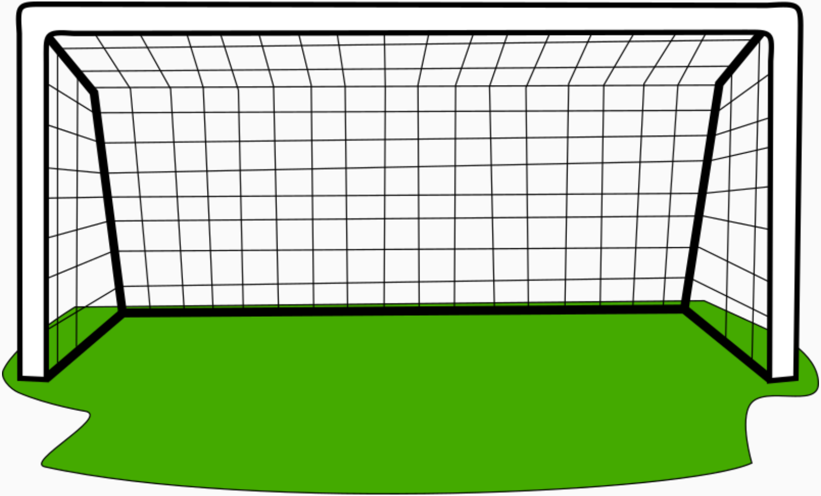


**Figure 1:** Intro to Apache Spark

# Data Science for Small Dataset/Exploratory Data Analysis (EDA)

Figure 2: Laptop

# Data Science for Larger Dataset (Vertical Scaling)

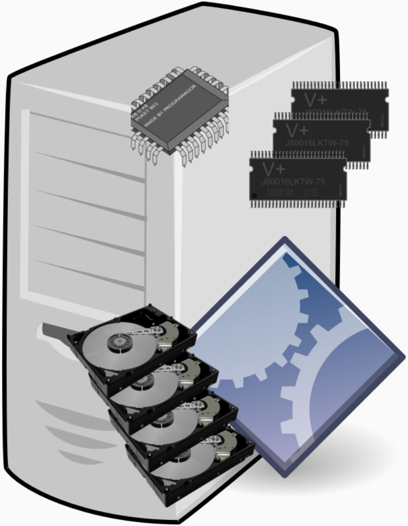
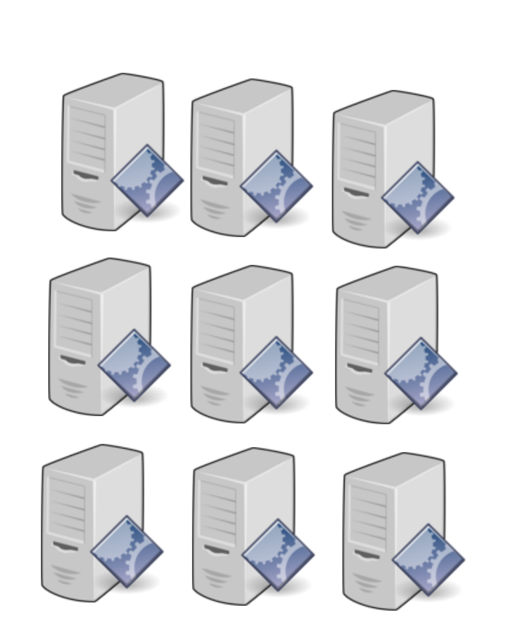


Figure 3: Beefed-up Server

# Data Science for Large Datasets (Horizontal Scaling)



# Early Big Data Framework - Apache Hadoop

Figure 5: HDFS, MapReduce

# Hadoop Ecosystem

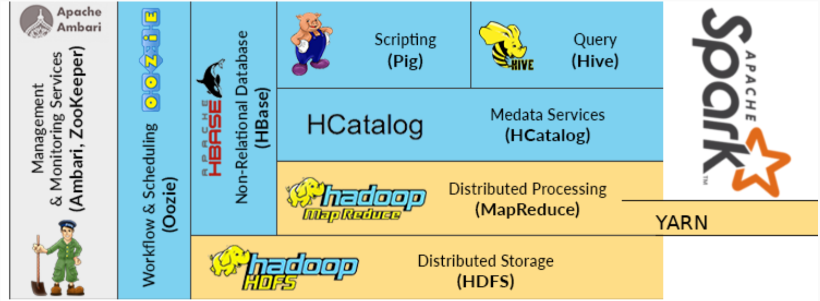
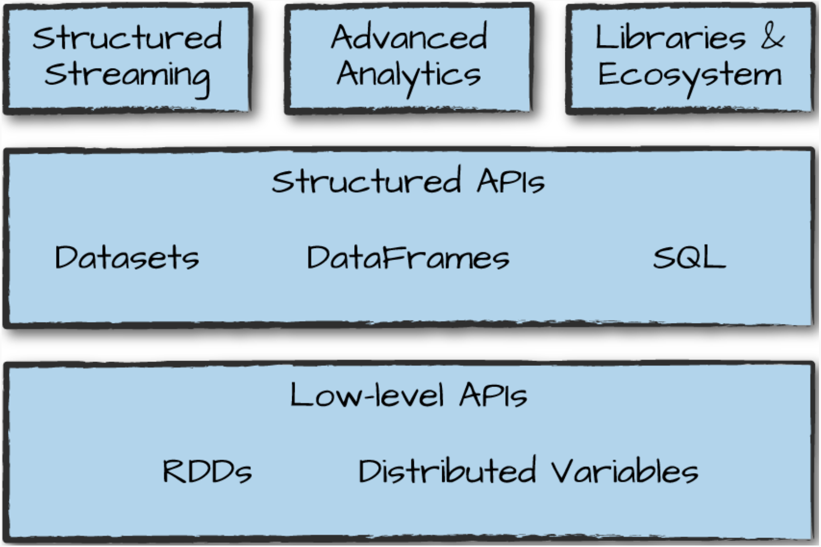

Apache Ambari
Management & Monitoring Services (Ambari, ZooKeeper)
Workflow & Scheduling (Oozie)
OOZIE
APACHE HBASE
Non-Relational Database (HBase)
Scripting (Pig)
HIVE
Query (Hive)
HCatalog
Medata Services (HCatalog)
hadoop MapReduce
Distributed Processing (MapReduce)
APACHE Spark
YARN
hadoop HDFS
Distributed Storage (HDFS)

Figure 6: Some Frameworks Around Hadoop

# Running Spark

- Local
    - Download from https://spark.apache.org, untar, add to PATH
    - SDKMAN - `curl -s "https://get.sdkman.io" | bash`
        - `sdk install spark`
    - `spark-shell` or `pyspark`
    - Edit `$SPARK_HOME/conf/spark-defaults.conf` (from template)
        - `spark.driver.memory                    8g`
- Standalone cluster, Hadoop YARN
    - Need shared file system or common datastore (e.g. AWS S3)
- Cloud-based managed:
    - AWS EMR
    - GCP Dataproc
    - Databricks on Azure, GCP or AWS

# Apache Spark Components



Structured Streaming

Advanced Analytics

Libraries & Ecosystem

Structured APIs

Datasets

DataFrames

SQL

Low-level APIs

RDDs

Distributed Variables

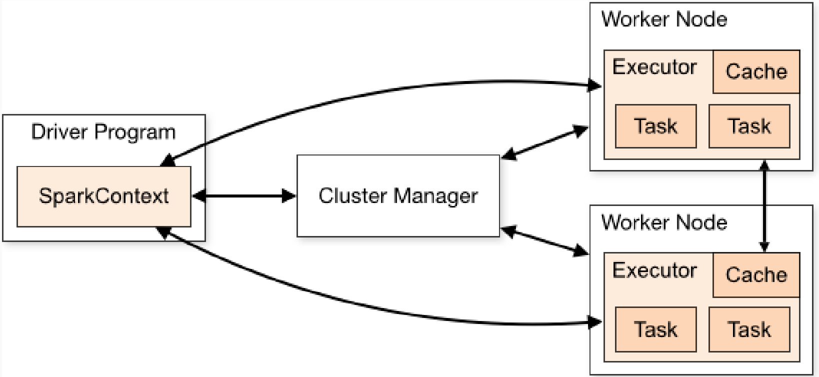
Source: Spark: The Definitive Guide

# Hello, Spark Email World!

```
spark-shell --master "local[4]" --driver-memory 8G
```

- Jupyter notebook with Apache Toree Notebook ../notebooks/html/ApacheSparkThroughEmail1.html

# Cluster Manager, Driver, Executors, (Jobs -> Tasks)



Source: Apache Spark website

# SparkSession: Entry to cluster

- `spark`: spark.sql.SparkSession

```
//SparkSession provided by notebook or shell as spark
val homeDir = sys.props("user.home")
val records = spark.read.
  parquet(s"$homeDir/datasets/enron/enron-small.parquet")

//In regular code for spark-submit
//com.uebercomputing.spark.dataset.TopNEmailMessageSenders
val spark = SparkSession.builder().
  appName("TopNEmailMessageSenders").
  master("local[2]").getOrCreate()
```

# DataFrameReader/Writer: Input/Output for structured data

- `spark.read/write`: spark.sql.DataFrameReader/Writer
  - jdbc
  - json
  - parquet
  - text...
  - Also: https://spark-packages.org - Redshift, MongoDB...

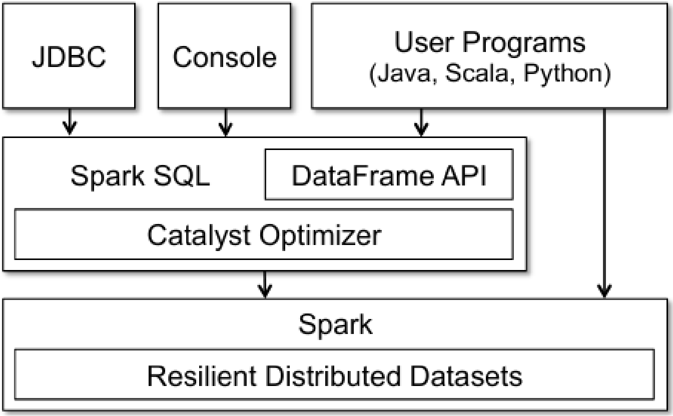
# Convert Dataset Format

```
import org.apache.spark.sql.functions._
val homeDir = sys.props("user.home")
val records = spark.read.parquet(s"$homeDir/datasets/enron/enron-small.parquet")

// write block-size file(s)
records.write.json(s"$homeDir/datasets/enron/json-parts")
records.repartition(1).write.json(s"$homeDir/datasets/enron/json-single")

// Dataset has mailfields/RE: and mailfields/re: fields
spark.conf.set('spark.sql.caseSensitive', true)
val jsonIn = spark.read.json(s"$homeDir/datasets/enron/json-parts")
```

# Transformations vs. Actions

- Transformation: returns a new RDD (nothing gets executed)
  - `read`, `cache`, `select`, `where`...
- Actions: trigger execution, catalyst query optimizer, Tungsten code generation
  - `count`
  - Bring rows back to driver: `take`, `collect` (watch OOM!)
  - `write`

# Unified Language Interface via Catalyst



# Phases of Catalyst Query Planning

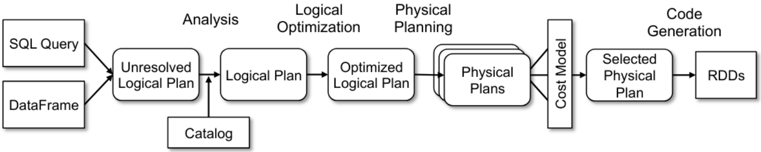
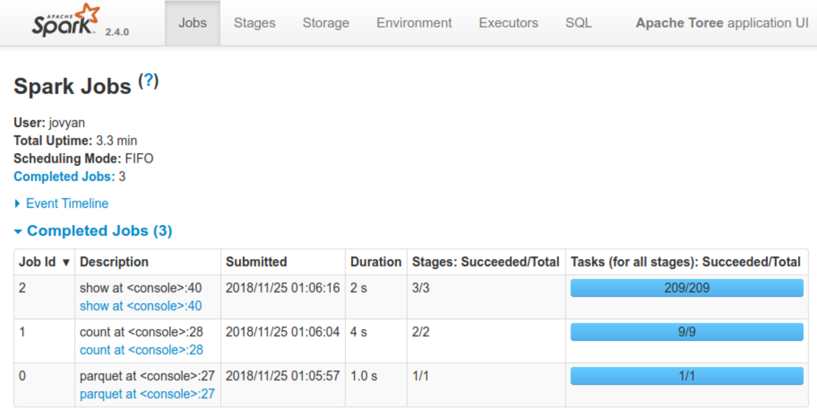


**Figure 3: Phases of query planning in Spark SQL. Rounded rectangles represent Catalyst trees.**

# Scaling Behind the Scenes



## Spark Jobs (?)

**User:** jovyan
**Total Uptime:** 3.3 min
**Scheduling Mode:** FIFO
**Completed Jobs:** 3

▸ Event Timeline

▾ **Completed Jobs (3)**

| Job Id ▾ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 2 | show at <console>:40<br>show at <console>:40 | 2018/11/25 01:06:16 | 2 s | 3/3 | 209/209 |
| 1 | count at <console>:28<br>count at <console>:28 | 2018/11/25 01:06:04 | 4 s | 2/2 | 9/9 |
| 0 | parquet at <console>:27<br>parquet at <console>:27 | 2018/11/25 01:05:57 | 1.0 s | 1/1 | 1/1 |

Figure 7: Jobs and Tasks

# Stages: Pipeline work per stage - shuffle

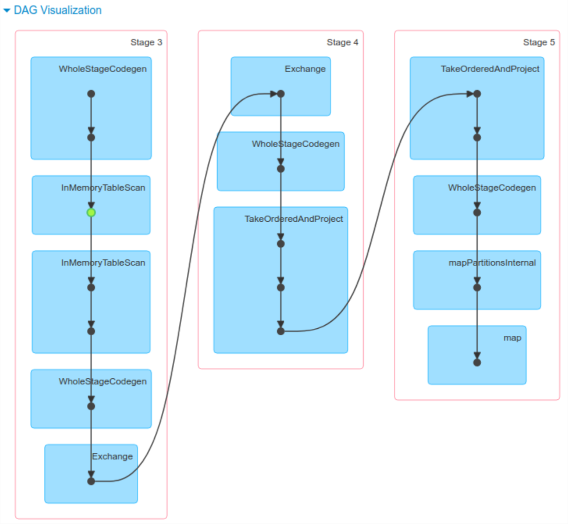
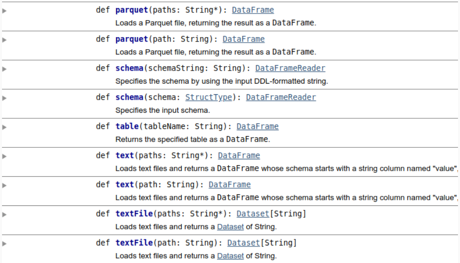


Figure 8: Stages

# Where clause, Column methods, Built-in functions

- See Notebook ../notebooks/html/ApacheSparkThroughEmail2.html

# Spark APIs - DataFrameReader, Dataset, Column, functions



`def parquet(paths: String*): DataFrame`
Loads a Parquet file, returning the result as a DataFrame.

`def parquet(path: String): DataFrame`
Loads a Parquet file, returning the result as a DataFrame.

`def schema(schemaString: String): DataFrameReader`
Specifies the schema by using the input DDL-formatted string.

`def schema(schema: StructType): DataFrameReader`
Specifies the input schema.

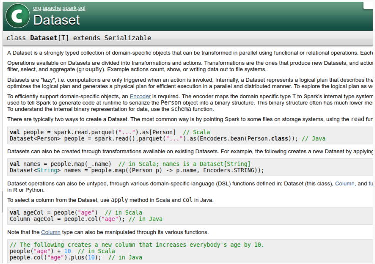
`def table(tableName: String): DataFrame`
Returns the specified table as a DataFrame.

`def text(paths: String*): DataFrame`
Loads text files and returns a DataFrame whose schema starts with a string column named "value",

`def text(path: String): DataFrame`
Loads text files and returns a DataFrame whose schema starts with a string column named "value",

`def textFile(paths: String*): Dataset[String]`
Loads text files and returns a Dataset of String.

`def textFile(path: String): Dataset[String]`
Loads text files and returns a Dataset of String.



org.apache.spark.sql

## Dataset

`class Dataset[T] extends Serializable`

A Dataset is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each

Operations available on Datasets are divided into transformations and actions. Transformations are the ones that produce new Datasets, and actio
filter, select, and aggregate (groupBy). Example actions count, show, or writing data out to file systems.

Datasets are "lazy", i.e. computations are only triggered when an action is invoked. Internally, a Dataset represents a logical plan that describes the
optimizes the logical plan and generates a physical plan for efficient execution in a parallel and distributed manner. To explore the logical plan as w

To efficiently support domain-specific objects, an Encoder is required. The encoder maps the domain specific type T to Spark's internal type system
used to tell Spark to generate code at runtime to serialize the Person object into a binary structure. This binary structure often has much lower me
To understand the internal binary representation for data, use the schema function.

There are typically two ways to create a Dataset. The most common way is by pointing Spark to some files on storage systems, using the read fur

```
val people = spark.read.parquet("...").as[Person]  // Scala
Dataset<Person> people = spark.read().parquet("...").as(Encoders.bean(Person.class)); // Java
```

Datasets can also be created through transformations available on existing Datasets. For example, the following creates a new Dataset by applying

```
val names = people.map(_.name)  // in Scala; names is a Dataset[String]
Dataset<String> names = people.map((Person p) -> p.name, Encoders.STRING));
```

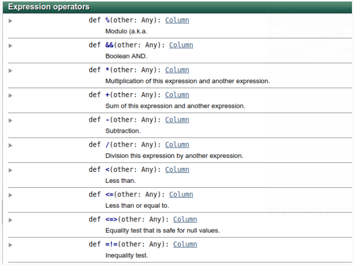
Dataset operations can also be untyped, through various domain-specific-language (DSL) functions defined in: Dataset (this class), Column, and fu
in R or Python.

To select a column from the Dataset, use apply method in Scala and col in Java.

```
val ageCol = people("age")  // in Scala
Column ageCol = people.col("age"); // in Java
```

Note that the Column type can also be manipulated through its various functions.

```
// The following creates a new column that increases everybody's age by 10.
people("age") + 10  // in Scala
people.col("age").plus(10);  // in Java
```



### Expression operators

`def %(other: Any): Column`
Modulo (a.k.a.

`def &&(other: Any): Column`
Boolean AND.

`def *(other: Any): Column`
Multiplication of this expression and another expression.

`def +(other: Any): Column`
Sum of this expression and another expression.

`def -(other: Any): Column`
Subtraction.

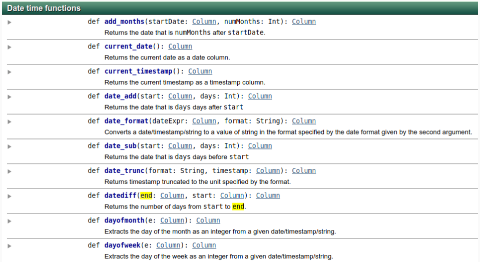
`def /(other: Any): Column`
Division this expression by another expression.

`def <(other: Any): Column`
Less than.

`def <=(other: Any): Column`
Less than or equal to.

`def <=>(other: Any): Column`
Equality test that is safe for null values.

`def =!=(other: Any): Column`
Inequality test.



### Date time functions

`def add_months(startDate: Column, numMonths: Int): Column`
Returns the date that is numMonths after startDate.

`def current_date(): Column`
Returns the current date as a date column.

`def current_timestamp(): Column`
Returns the current timestamp as a timestamp column.

`def date_add(start: Column, days: Int): Column`
Returns the date that is days days after start

`def date_format(dateExpr: Column, format: String): Column`
Converts a date/timestamp/string to a value of string in the format specified by the date format given by the second argument.

`def date_sub(start: Column, days: Int): Column`
Returns the date that is days days before start

`def date_trunc(format: String, timestamp: Column): Column`
Returns timestamp truncated to the unit specified by the format.

`def datediff(end: Column, start: Column): Column`
Returns the number of days from start to end.

`def dayofmonth(e: Column): Column`
Extracts the day of the month as an integer from a given date/timestamp/string.

`def dayofweek(e: Column): Column`
Extracts the day of the week as an integer from a given date/timestamp/string.

# Parallelism and Partitioning

- Goldilocks - not too many, not too few
- Initial parallelism - number of input “blocks”
- Splittable file formats (e.g. parquet, avro, bzip2)
    - Not zip, gzip!
- Shuffle - Adaptive Query Execution (dynamic partitioning)

# Explode, Shuffle Partitions, UDF, Parquet partition

- See Notebook ../notebooks/html/ApacheSparkThroughEmail3.html

# Pandas on Spark

- See https://spark.apache.org/pandas-on-spark/

# Apache Parquet/Apache Arrow

- Avro - record-oriented data format
- Parquet - column-oriented data format by page
- Arrow - share memory for Python
- https://spark.apache.org/docs/latest/api/python/user_guide/sql/arrow_pa

# Resources

- https://spark.apache.org/
- https://spark-packages.org/ - Community 3rd party packages (e.g. data sources)
- https://sparkbyexamples.com/
- RDD - https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf
- Spark SQL - https://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf
- Adaptive query execution - https://www.databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html

# And now for something completely different: Colon Cancer

- Screening saves lives!
  - Colonoscopy - talk to your doc
- Colorectal Cancer Alliance

# Questions?

- markus.dale@bluehalo.com
- Infrequent blog/past presentations http://uebercomputing.com/
- Spark Mail repo https://github.com/medale/spark-mail/