

Speaker Notes: Data Engineering with Apache Spark

Markus Dale, medale@asymmetrik.com

May 2019

- Open Spark API:

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

- Bio:
 - mostly Java, big data with Hadoop
 - big data with Spark, Databricks, Scala
 - Now Asymmetrik - Scala, Spark, Elasticsearch, Akka...
 - Data Engineer
- Slides: <https://github.com/medale/prez-spark-dataengineering/blob/master/presentation/SparkDataEngineering.pdf>
- Scala Spark Code Examples:
<https://github.com/medale/prez-spark-dataengineering>
- Also <https://github.com/medale/spark-mail>

- <https://www.gharchive.org/>

- <https://www.gharchive.org/>
- Old API/Events API (we won't deal with old API)
- Events API - PullRequestEvent

- dataquest.io: “transform data into a useful format for analysis”

- Shell for exploration at scale
- Dataset batch API - many supported input sources/formats
 - builds on Hadoop and other 3rd party libraries
- Streaming API
- ML library
- Graph library

- Take subset of data
- Figure out structure, approaches

- Server-grade machine - more cores
- More memory, more data

- Cluster manager manages resources
- Spark manages Spark application (driver, executors)
 - Sunny day
 - Error handling (machine dies, slows, network...)

Cluster Manager - Manage cores, memory, special capabilities

- Spark local mode (not a cluster manager)
- Spark Standalone
- Kubernetes, Mesos
- Spark on Hadoop YARN
- In cloud: Spark on AWS EMR, Google, Azure, Databricks
- Schedule resources

Anatomy of a Spark Application

- One cluster manager - multiple Spark applications
- Per Spark application
 - 1 driver
 - n executors (cache memory, task slots)

```
import org.apache.spark.sql.SparkSession

object HelloSparkWorld {

  def process(spark: SparkSession): (Long,Long) = {
    val records = spark.read.json( path = "file:///datasets/github/data")
    records.cache()
    val totalEventCount = records.count()

    val prs = records.where(records("type") === "PullRequestEvent")
    val pullRequestEventCount = prs.count()

    records.unpersist()
    (totalEventCount, pullRequestEventCount)
  }

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().
      appName( name = "HelloSparkWorld").
```

Hello, Spark World!

- spark session
- spark.read (DataFrameReader) - json (.gz, 1 json per line)
- lazy transformation - read to get schema
- count - action - execute a job
- Datasets, DataFrame and RDD are immutable
- contain lineage (how did we get to this dataset?)
- where - transformation

- builder static method - Builder
- appName
- config
- master
- getOrCreate()

- spark.apache.org - Documentation - API Docs
- Object (“static” methods) vs. class

- read - DataFrameReader input
- createDataFrame/createDataset
- emptyDataFrame/emptyDataset
- spark.implicits._ (\$ and Scala object encoder)
- udf
- stop

- csv
- json
- parquet
- text - DataFrame - column "value"
- textFile - Dataset[String]
- schema - specify read schema

- sql package object - `type DataFrame = Dataset[Row]`
- DataFrame has a schema

- `printSchema`
- `.schema ()`

- subset to local drive
- for production - need distributed storage system
 - S3
 - Hadoop HDFS

- What's in the gz?
- How many lines?

- visual exploration - seems like one JSON per line

Pretty Print One Record?

- `split`
- `python json.tool`

- see JSON, look at fields for one record

Starting Spark Standalone Cluster Manager

- start master to explicit host (default port 7077)
- one or more workers to spark://...:

- `-master` - what cluster manager to ask for resources
- `-deploy-mode` (default client or cluster)
- driver: coordinates this Spark application
- executors - cores - how many tasks in parallel
- jars (built via assembly)

- web ui (for this Spark application)
 - 4040, 4041 etc.
- Special vars: spark: SparkSession, sc: SparkContext
- Exit: `:quit`

Spark Standalone Cluster Manager - 1 running application

- 4 cores (total)
- 2 executors with 2 cores each
- 2GB/executor
- [Link to Spark shell - Spark application UI](#)

- SparkContext - old
- spark.implicits._ - \$ function, encoders for Scala primitives and case classes
- spark.sql package - DataFrame (Dataset[Row])
- functions: math, string, date for columns

- urls - file, hdfs, s3a
- schema - superset of all JSONs
- just execute job (list files, read to find schema)

- job0 - read 3 unsplittable files, determine JSON schema
 - 1 stage - everything in parallel
 - 2 executors with two task slots each
 - 3 tasks - read unsplittable files
- job1 - count
 - 2 stages - count local, shuffle, add up total
 - 4 tasks - 3 local counts, 1 shuffle add total

- 3 stages - last stage 1 task

- input, output

- See two stages - shuffle

One Job = n lazy transformations, 1 action

- lazy transformations
- Dataset api - select (projection)
- distinct
- show - action, count (2 job)
- cache/unpersist

- cache (persist level)
- unpersist
- in memory, spill to local disk

- no URL prefix needed for defaultFS
- file, hdfs, s3a

Input partitions - splittable file?

- Splittable: bzip2, parquet, avro
- Non-splittable: gzip (1 task per file)
- small file problem

- Catalyst query optimizer for built-in functions
- Project Tungsten - memory management
 - Row storage (Apache Arrow)
 - Encoders for Dataset objects (`spark.implicits._`)

- where clause
- groupBy - RelationalGroupedDataset
 - count
 - avg, sum, agg (agg functions mean, std dev...)
- where(String), where(Column)
- getting column - apply on Dataset, \$, col function

- <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-transformations.html>
- narrow, wide

- 200 (“spark.sql.shuffle.partitions”)

- “spark.sql.shuffle.partitions”

- Less time - 0.1 vs. 0.8

- select
- where
- distinct
- limit
- orderBy
- join

- collect
- count
- take(n)
- head
- write - DataFrameWriter

- DataSourceRegister, implicits

- text - one column “value”

- _ hidden
- also hidden .crc files
- part-00000-UUID...

- Some dataset - use string, some columns
- +, -, *, %
- ==, !=, >, <, ...
- asc, desc
- startsWith, contains, endsWith, like, rlike
- isNull, isNaN, isIn

- array functions, explode
- date/time functions
- math, string
- use built-in to allow Catalyst optimization
- also user-defined-functions, user-defined-agg-functions

- `created_at` is a string but in ISO 8601

- immutable, 4 transformations
- use time functions (require column)
- `spark.implicits._` \$

- year=2019 etc.

- Schema is stored with Parquet file header
- Predicate pushdown - partitioning (no read)
- Columns - skip unused

- prune to only read hour 21

- reads all (some task read more than one file)

Memory pressure - partitions, executors, shuffle partitions


```
sc.hadoopConfiguration.set("fs.s3a.secret.key",  
sc.hadoopConfiguration.set("fs.s3a.access.key",
```