

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 does on JSON look like pretty printed?

Starting Spark Standalone Cluster Manager

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

- Driver: deploy-mode client or cluster, memory
- Executors
 - how many total?
 - how many parallel tasks per executor (cores)
 - memory

Code executing on driver vs. executor

-deploy-mode (default client)

- each stage - all transformations that can be done without shuffle (narrow transformations)
- wide transformation - shuffle (data in a partition was created from many other partitions)

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

Shuffle partitions

- inferred
- read/write schema

- data partitioning

Memory pressure - partitions, executors, shuffle partitions


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

```
https://www.gharchive.org/ wget
```

```
http://data.gharchive.org/2019-04-28-0.json.gz wget
```

```
http://data.gharchive.org/2019-04-28-1.json.gz wget
```

```
http://data.gharchive.org/2019-04-28-13.json.gz
```

store under data directory run spark-shell from parent of data directory (gz
of .json file with one json per line)

```
val records = spark.read.json("data")
```

```
//slow - needs to figure out JSON schema
```

```
records.cache
```

```
records.count
```

```
//235728
```