Putting the Spark into Functional Fashion Tech Analytics

Gareth Rogers **Data Engineer**

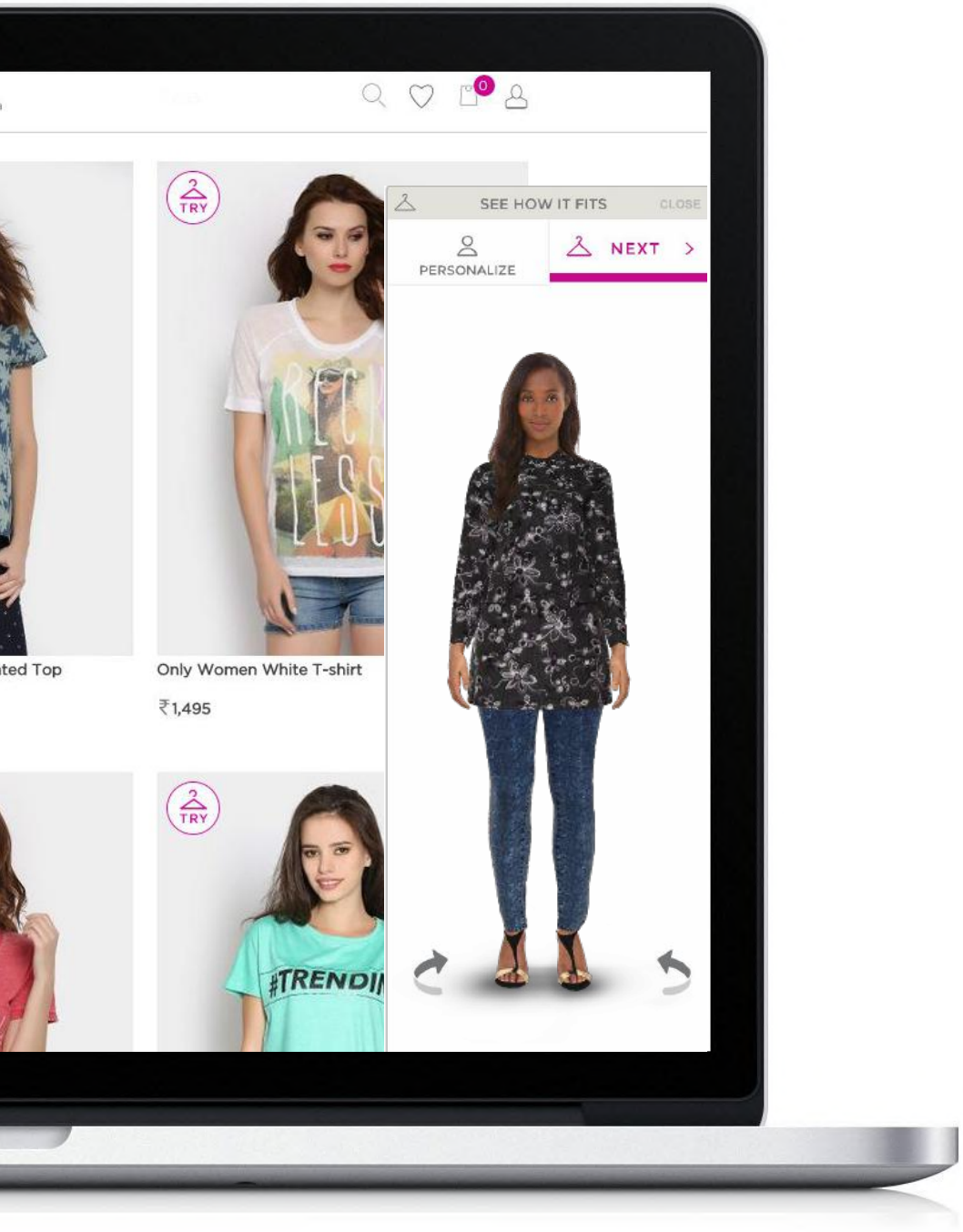
November 2018

Introduction

- Who are Metail and what we do
- Our data pipeline
- What is Apache Spark and our experiences

Metail

Making clothing fit for all



The Metail Experience

- Create your MeModel with just a few clicks
- See how the clothes look on you
- Primarily clickstream analysis
- Understanding the user journey

http://trymetail.com

Composed Photography

With Metail

Shoot model once

Choose poses

Shoot clothes at source

Compositing

Style & restyle

- Understanding process flow
- Discovering bottlenecks
- Optimising the workflow
- Understanding costs

Functional Pipeline

- Our pipeline is heavily influenced by functional programmings paradigms
- Immutable data structures
- Declarative
- Pure functions -- effects only dependent on input state
- Minimise side effects

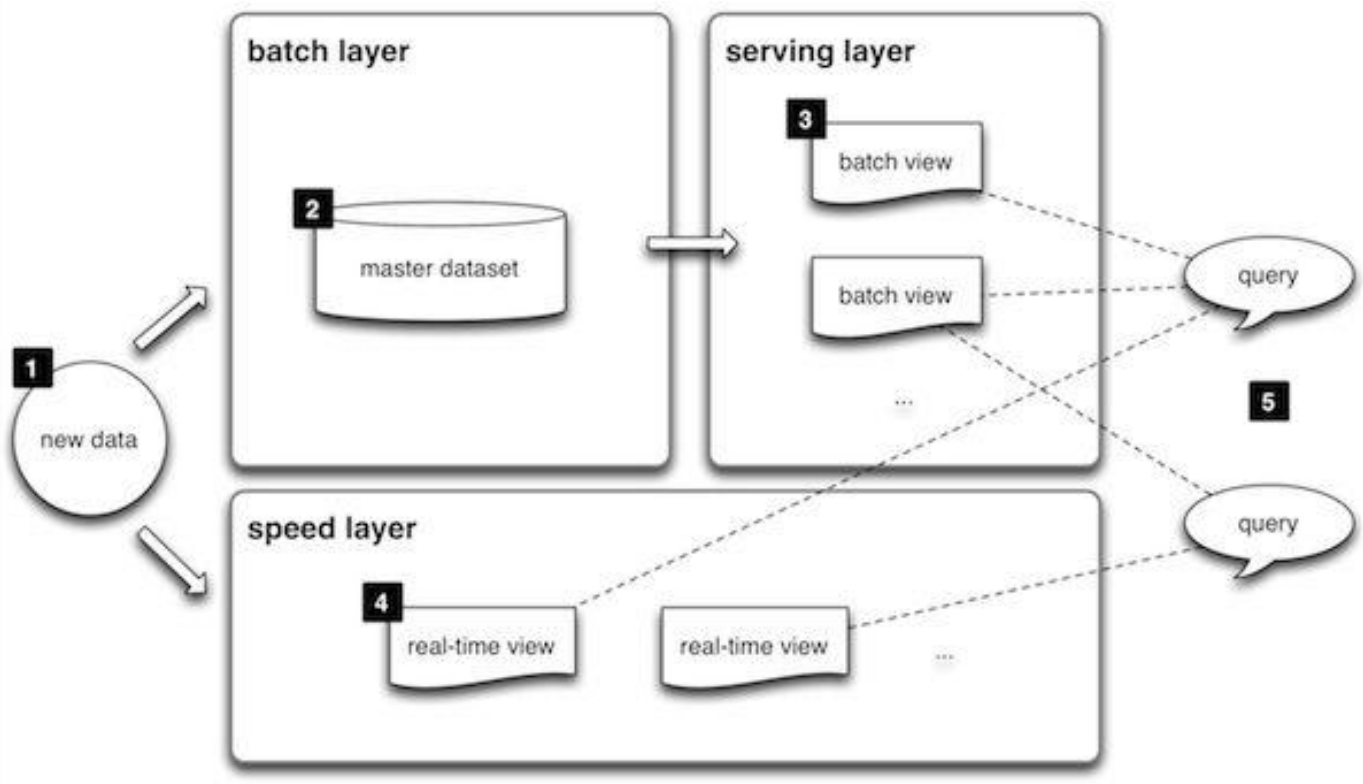
Metail's Data Pipeline

- Batch pipeline modelled on a lambda architecture

Metail's Data Pipeline - The lambda architecture

- Batch pipeline modelled on a [lambda architecture]

- Immutable datasets
- Batch layer append only
- Rebuild views rather than edit
- Serving layer for visualization
- Speed layer samples input data
  - [Kappa] architecture



Metail's Data Pipeline - Driving the Pipeline

- Managed by applications written in Clojure

Clojure

- Clojure is predominately a functional programming language
- Aims to be approachable
- Allow interactive development
- A lisp programming language
- Everything is dynamically typed
- Runs on the JVM or compiled to JavaScript

Metail's Data Pipeline - Driving the Pipeline

- Running on the JVM
  - access to all the Java ecosystem and learnings
  - Java interop well supported
  - not Java though

Metail's Data Pipeline - Driving the Pipeline

```
(defschema NonEmptyStr (s/constrained s/Str (complement empty?) 'not-empty?))

(defschema HadoopJarStep {:jar NonEmptyStr :args [s/Str]})

(defschema HadoopStep
  {:name              NonEmptyStr
   :action-on-failure ActionOnFailure
   :hadoop-jar-step   HadoopJarStep})
```

```
(s/defn shred-enriched-events :- common/HadoopStep
  [{:keys [snowplow] :as config} run-id]
  {:name              (with-epoch (str "snowplow-shred-enriched-events-" run-id))
   :action-on-failure ActionOnFailure/CONTINUE
   :hadoop-jar-step   (spark-submit
                       (get-in snowplow  [:shred :jar])
                       (get-in snowplow  [:shred :class])
                       "--iglu-config"   (base64-json (:iglu-config snowplow))
                       "--input-folder"  (hdfs-path :shred-input config run-id)
                       "--output-folder" (hdfs-path :shred-output config run-id)
                       "--bad-folder"    (s3-path :shred-bad-rows config run-id)
                       "--duplicate-storage-config" (base64-json (:duplicate-storage snowplow)))})
```

Snowplow Analytics (shameful plug, we use their platform and I like the founders)

Metail's Data Pipeline - Driving the Pipeline

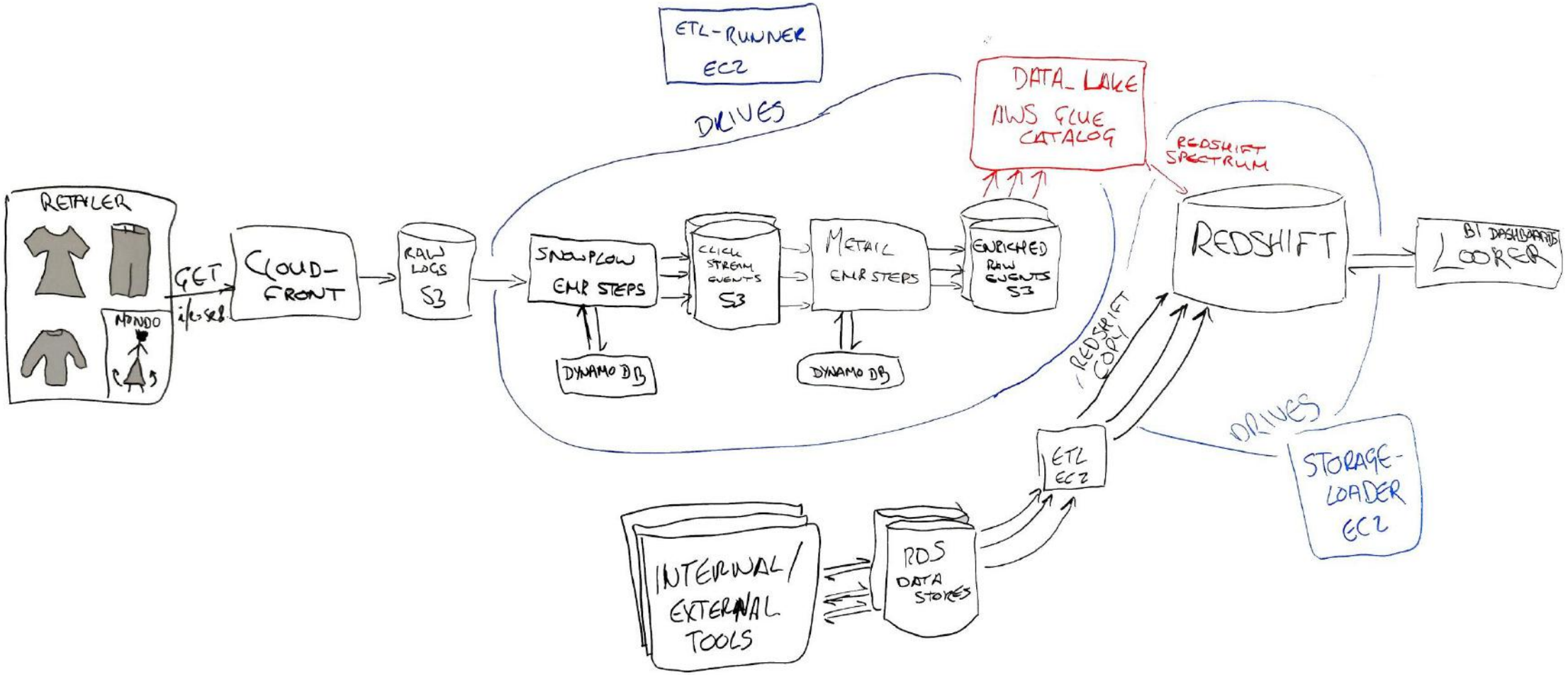
```
{:step-fn             shred-enriched-events
 :next                ::copy-shredded-data-to-s3
 :allow-new-jobflow? false
 :continue-on-error? false
 :retry?              false}
```

```
(let [step                (first steps)
      allow-new-jobflow? (:allow-new-jobflow? step)
      continue-on-error? (:continue-on-error? step)
      want-retry?         (:retry? step)
      step                (dissoc step :allow-new-jobflow? :continue-on-error? :retry?)
      step-start-time     (t/now)
      ok?                 (if (:hadoop-jar-step step)
                            (emr/supervise config step want-retry? allow-new-jobflow?)
                            (run-local-fn step))
      step-duration       (time-since step-start-time)]
  (if ok?
    (do
      (log/infof "Step %s completed successfully in %s" (:name step) step-duration)
      (recur (rest steps)))
    (let [message (format "Step %s failed after %s" (:name step) step-duration)]
      (log/error message)
      (slack/notify-error config message)
```

Metail's Data Pipeline - Transforming the Data

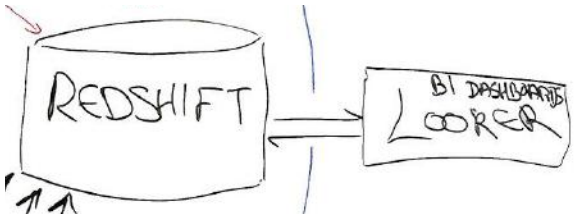
- Transformed by Clojure, Spark in Clojure or SQL
- Clojure used for datasets with well defined size
  - These easily run on a single JVM
  - Dataset size always within an order of magnitude
- Spark
  - Dataset sizes can vary over a few orders of magnitude
- SQL is typically used in the serving layer and for dashboarding and BI tools

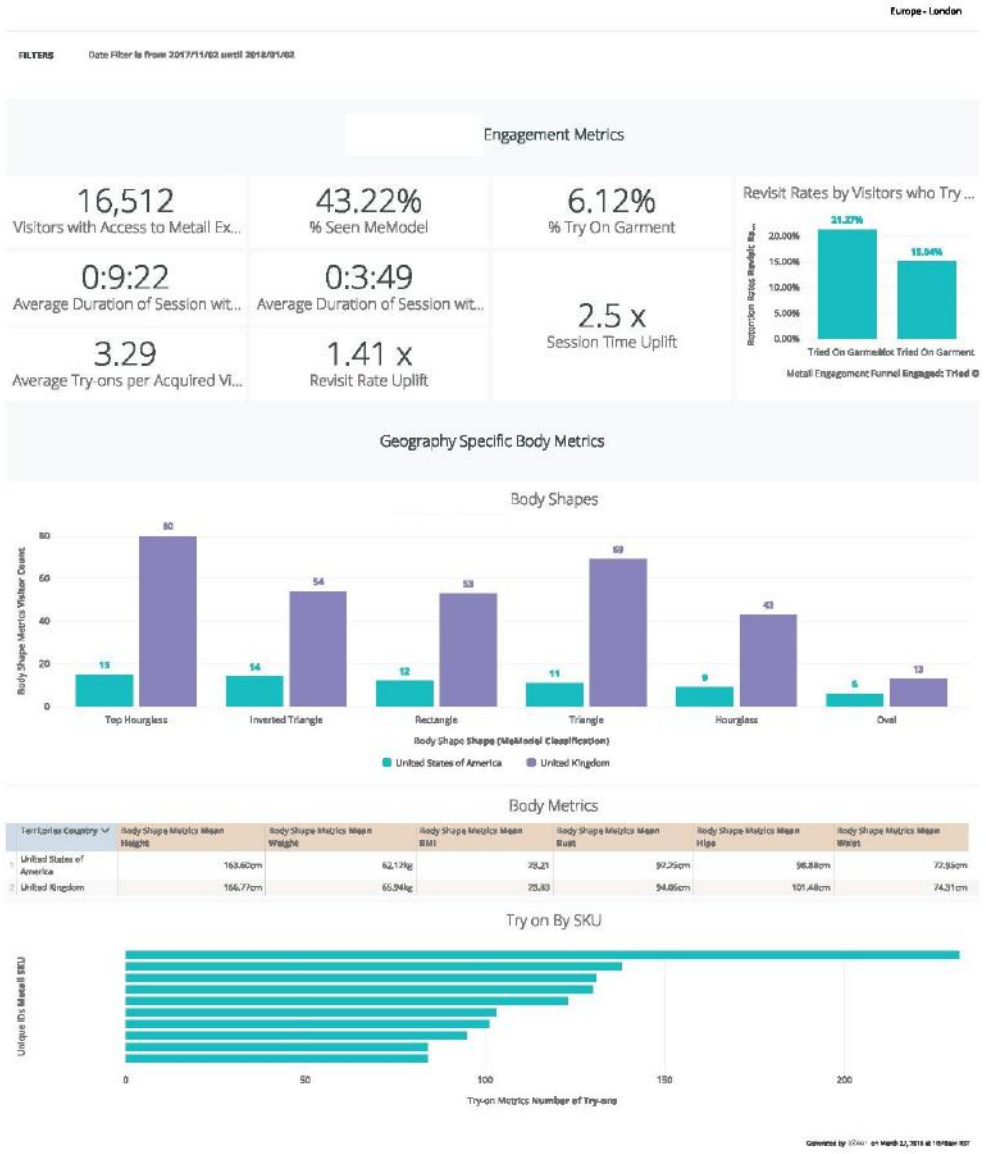
Metail's Data Pipeline



Metail's Data Pipeline

To analytics dashboards





https://looker.com/

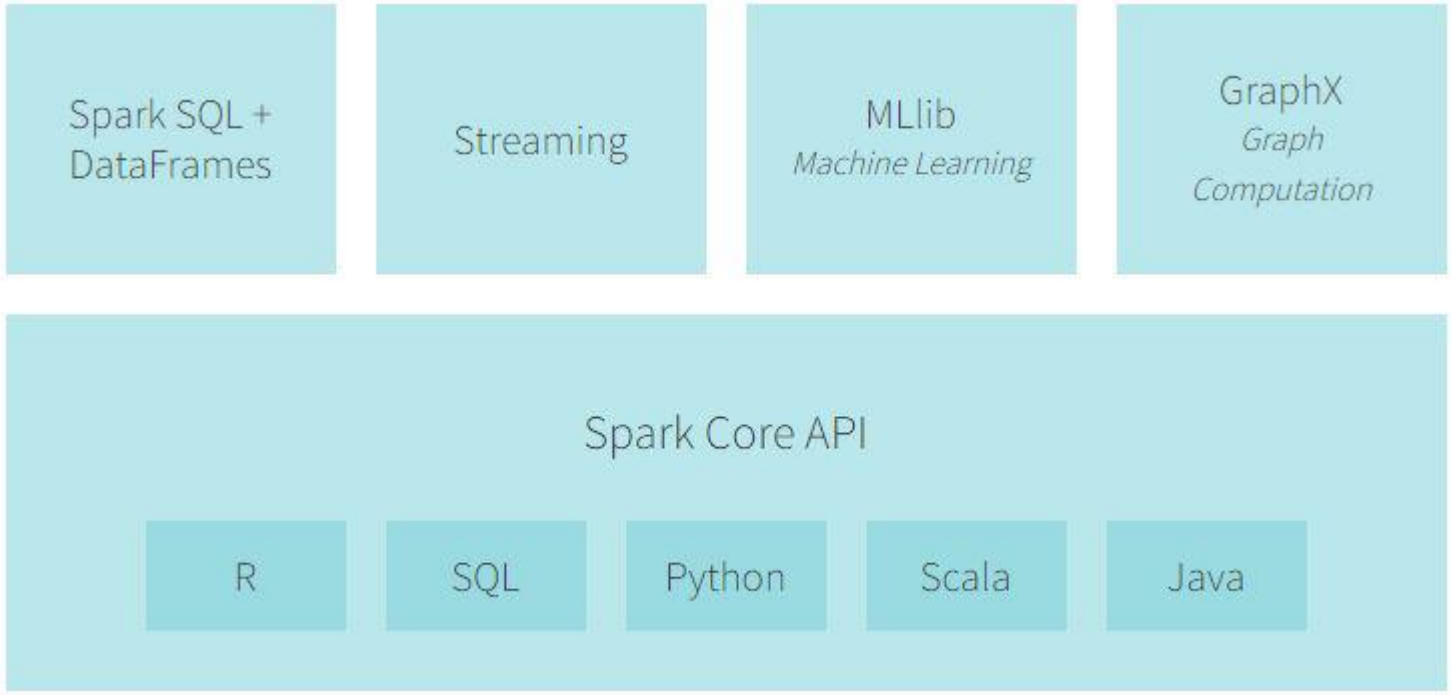
Apache Spark

- Spark is a general purpose distributed data processing engine
  - Can scale from processing a single line to terabytes of data

- Functional paradigm
  - Declarative
  - Functions are first class
  - Immutable datasets

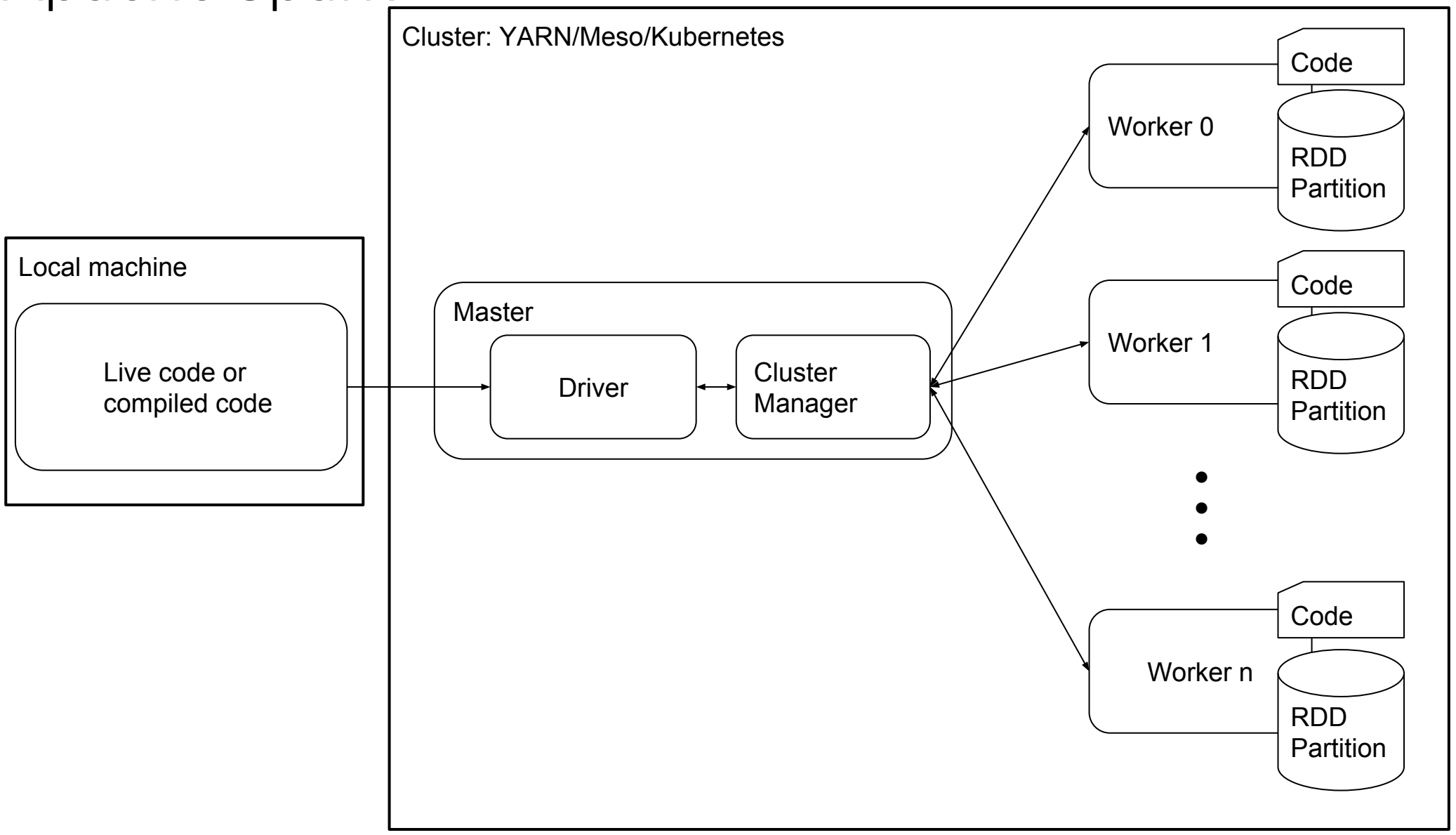
- Written in Scala a JVM based language
  - Just like Clojure
  - Has a Java API
  - Clojure has great Java interop

Apache Spark

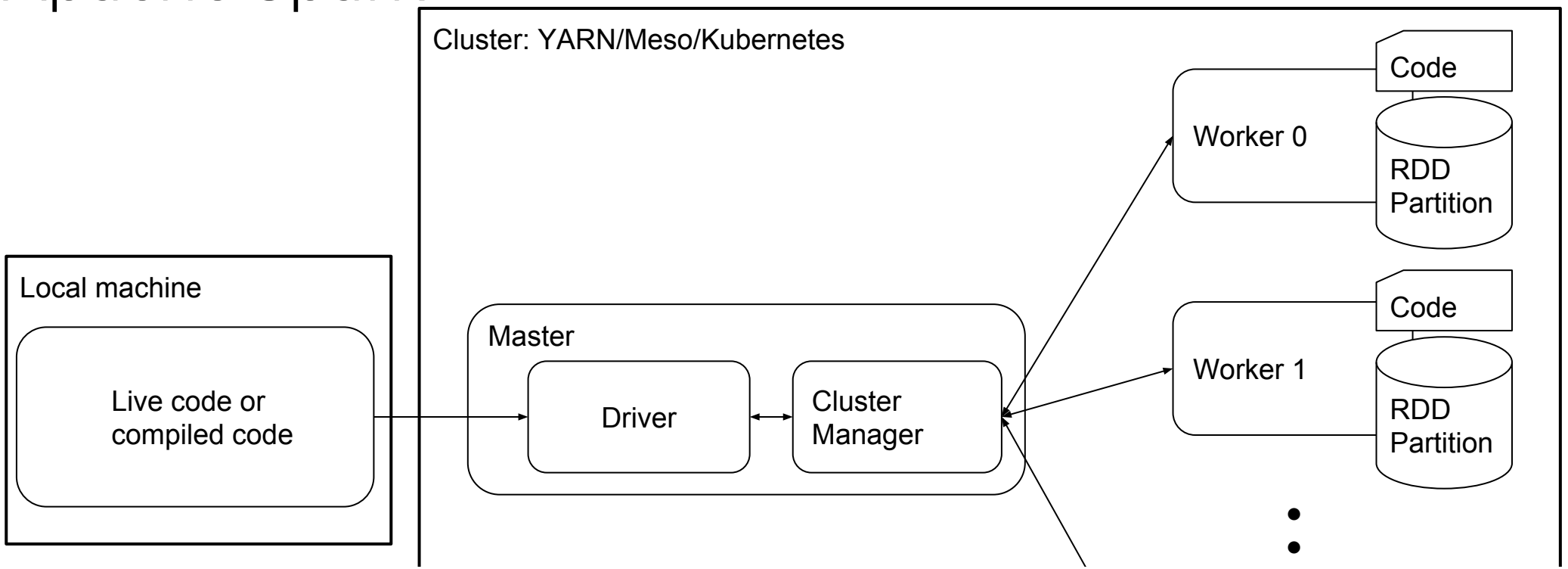


https://databricks.com/spark/about

Apache Spark



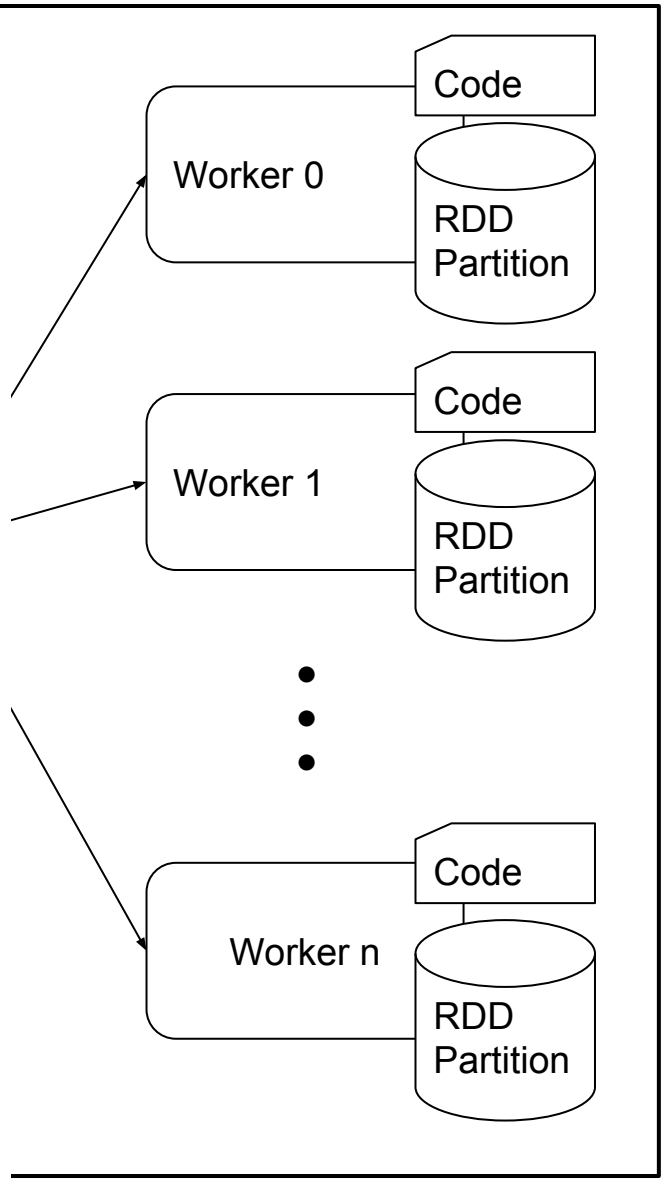
Apache Spark



- Consists of a driver and multiple workers
- Declare a Spark session and build a job graph
- Driver coordinates with a cluster to execute the graph on the workers

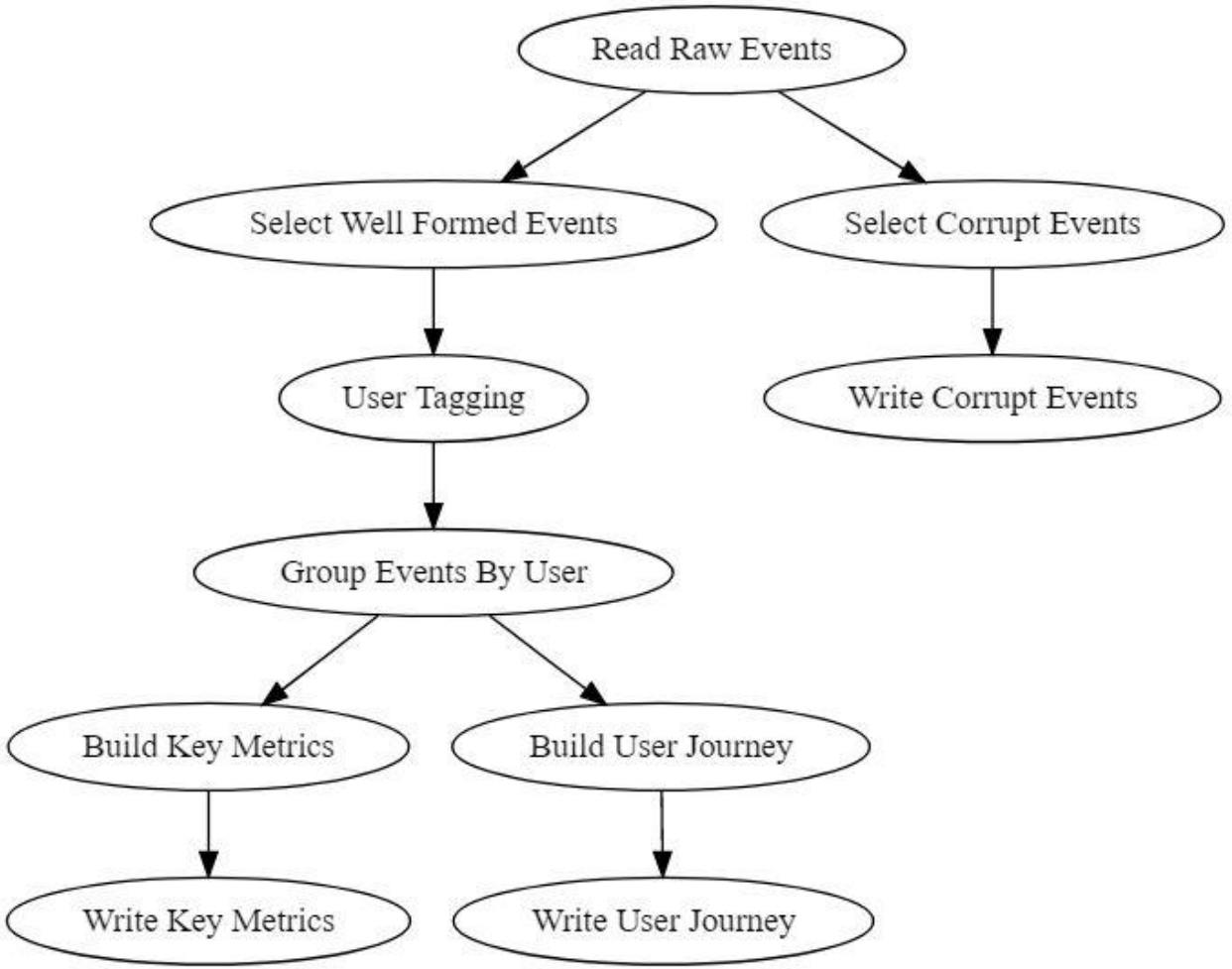
Apache Spark

- Operates on in-memory datasets
  - where possible, it will spill to disk
- Based on Resilient Distributed Datasets (RDDs)
- Each RDD represents on dataset
- Split into multiple partitions distributed through the cluster



Apache Spark

- Operates on a Directed Acyclic Graph (DAG)
  - Link together data processing steps
  - May be dependent on multiple parent steps
  - Transforms cannot return to an older partition



Metail and Spark

- Using the Sparkling Clojure library
  - This wraps the Spark Java API
  - Handles Clojure data structure and function serialisation
- For example counting user-sessions
  - This is simple but doesn't scale, returns everything to driver memory

```
(let [events  (->> (spark/parallelize (session/java-spark-context spark-session)
                                      ["{\"user-id\": \"abcd\", \"session\": 0}"
                                       "{\"user-id\": \"efgh\", \"session\": 0}"
                                       "{\"user-id\": \"abcd\", \"session\": 1}"
                                       "{\"user-id\": \"ijkl\", \"session\": 0}"
                                       "{\"user-id\": 7, \"session\": 2}"])
                   (spark/map (fn [s] (json/parse-string s true))))
      corrupt (spark/filter (fn [{:keys [user-id]}] (not (string? user-id))) events)
      user-sessions (->> events
                         (spark/filter (fn [{:keys [user-id]}] (string? user-id)))
                         (spark/map-to-pair (fn [{:keys [user-id] :as m}]
                                              (spark/tuple user-id m)))
                         (spark/count-by-key))]
  (println "count-by-key:" user-sessions))
```

```
18/11/28 13:14:38 INFO BlockManager: In
40661, None)
count-by-key: {abcd 2, efgh 1, ijkl 1}
lhcb-opendata>
```

Metail and Spark

- Two variants that return PairRRDs
- *combineByKey* is more general than *aggregateByKey*
- Note I did the Java interop as Sparkling doesn't cover this API
  - Easy when you can use their serialization library

```
      user-sessions-0 (spark/combine-by-key (fn [_] 1)
                                            (fn [v c] (inc v))
                                            (fn [c0 c1] (+ c0 c1))
                                            users)
      user-sessions-1 (.aggregateByKey users
                                       0
                                       (func/function2 (fn [v u] (inc v)))
                                       (func/function2 (fn [u0 u1] (+ u0 u1))))]
(println "combine-by-key:" (spark/collect user-sessions-0))
(println "aggregate:" (spark/collect user-sessions-1)))
```

```
combine-by-key: #object[scala.collection.convert.Wrappers$SeqWrapper 0x3be2e988 [(abcd,2), (efgh,
1), (ijkl,1)]]
aggregate: #object[scala.collection.convert.Wrappers$SeqWrapper 0x329c0d25 [(abcd,2), (efgh,1), (
ijkl,1)]]
```

Metail and Spark

- Mostly using the core API
- RDDs holding rows of Clojure maps
- Would like to migrate to the Dataset API

```
(let [schema     (types/struct-type [{:name      "user-id"
                                      :type      (types/string-type)
                                      :nullable? false}
                                     {:name      "session"
                                      :type      (types/integer-type)
                                      :nullable? false}])
      fake-file (spark/parallelize (session/java-spark-context spark-session)
                                   ["{\"user-id\": \"abcd\", \"session\": 0}"
                                    "{\"user-id\": \"efgh\", \"session\": 0}"
                                    "{\"user-id\": \"abcd\", \"session\": 1}"
                                    "{\"user-id\": \"ijkl\", \"session\": 0}"
                                    "{\"user-id\": 7, \"session\": 2}"])
      events     (.. (.read spark-session)
                     (schema schema)
                     (options {"mode" "DROPMALFORMED"})
                     (json fake-file))
      user-sessions (-> events
                        (ds/group-by "user-id")
                        (.count))]
  (ds/show user-sessions 5 0 false))
```

```
+-------+-----+
|user-id|count|
+-------+-----+
|abcd   |2    |
|efgh   |1    |
|ijkl   |1    |
|7      |1    |
+-------+-----+
```

Metail and Spark

- Runs on AWS Elastic MapReduce
  - Tune your cluster

```
(session/spark-session {:app-name "lhcb-opendata-repl"
                        :master   "local[1]"
                        :config   {"spark.driver.cores"            "1"
                                   "spark.driver.memory"           "1G"
                                   "spark.driver.memoryOverhead"   "100M"
                                   "spark.executor.cores"          "1"
                                   "spark.executor.memory"         "1G"
                                   "spark.executor.memoryOverhead" "100M"
                                   "spark.sql.shuffle.partitions"  "1"
                                   "spark.default.parallelism"     "1"}}))
```

- This is an example for my limited VM, a cluster would use bigger values!

Metail and Spark - Pros and Cons

- Very scalable
  - but the distributed environment adds overhead
- Sparkling does a lot of the hard work
  - Clojure not a supported language
- Good documentation
  - Sometimes it's hard to figure out which way to do something
  - Lots of deprecated methods
- Declarative language + Clojure interop makes stacktraces hard to interpret
- Dataset API is heavily optimised
  - Would remove a lot of the Clojure interop

Summary

- Metail is making clothing fit for all
- We’re incorporating metrics derived from our collected data
- We have several pipelines collecting, transforming and visualising our data
- When dealing with datasets functional programming offers many advantages
- Give them a go!
  - https://github.com/gareth625/lhcb-opendata

Resources

- Learning Clojure: https://www.braveclojure.com/
- A random web based Clojure REPL: https://repl.it/repls/
- Basis of Metail Experience pipeline https://snowplowanalytics.com
- Dashboarding and SQL warehouse management: https://looker.com
- Tuning your Spark cluster
  http://c2fo.io/c2fo/spark/aws/emr/2016/07/06/apache-spark-config-cheatsheet/

Questions?