twilio

#ODSC

OPEN DATA SCIENCE CONFERENCE

Boston | April 30 - May 4th, 2019

@newfront | @odsc

#ODSC

BOSTON
APR 30 – MAY 3

Real-ish Time Predictive Analytics with Spark Structured Streaming

Scott Haines
Principal Software Engineer
Twilio

Introductions

Scott Haines, Principal Software Engineer,
Voice Insights, **Twilio**

http://bit.ly/odsceastrealtime
http://bit.ly/odsc-east-2019-realish-predict

@newfront | @odsc | @twilio

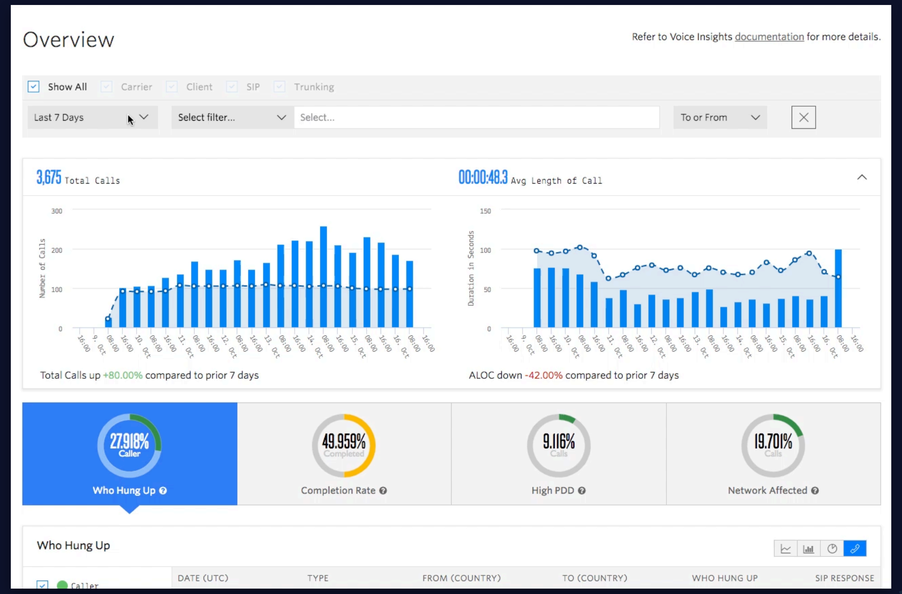
What I Do
Voice Insights

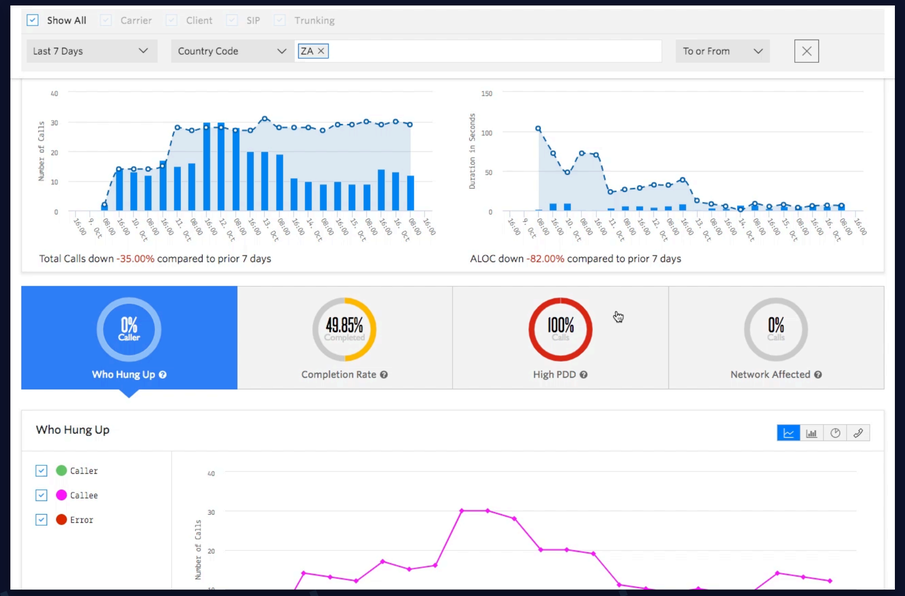
Voice Analytics Platform

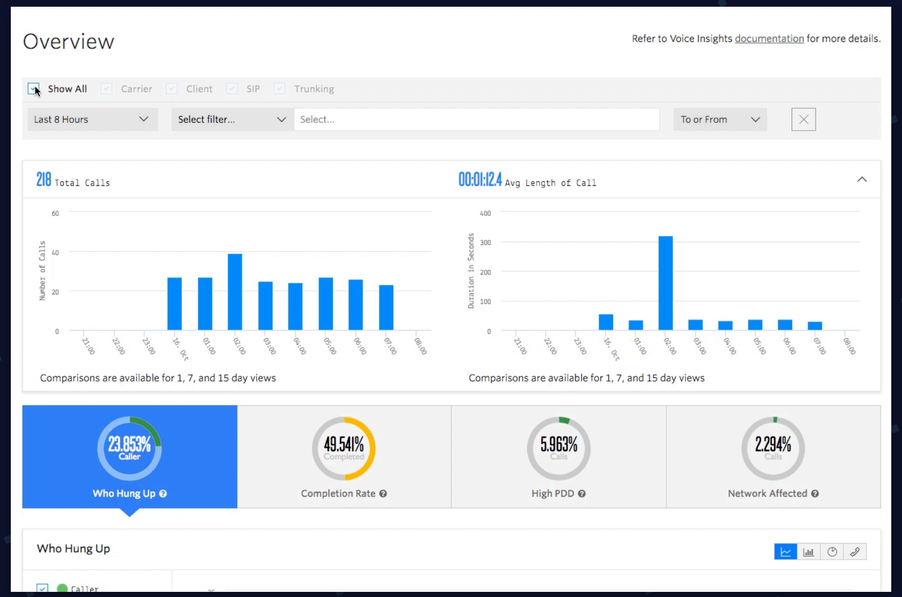
Millions of complex time series aggregations daily across billions of raw events

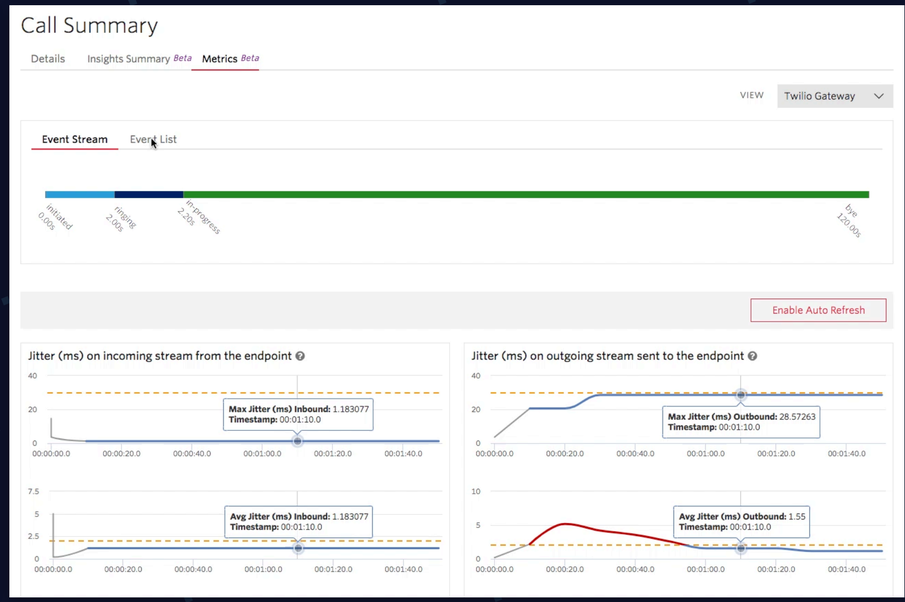
Supports internal and external customers.

*Near Real-Time









WORKSHOP

Data Engineering

Exploratory Data Analysis

Clustering and Tagging

Naive Bayes and Probability

Streaming Predictions

http://bit.ly/odsc-east-2019-realish-predict

DATA ENGINEERING

http://bit.ly/odsc-east-2019-realish-predict

Data Engineering

- Define Schemas
- ETL (extract transform load)
- Clean and Sanitize Data
- Own your Data

Structured Data

- JSON has structure. But it isn't Structured Data.

>95%

Of people love JSON*.

```
{
  "type": "employee",
  "nickname": "shaines",
  "attributes": [
    {
      "height": 6,
      "unit": "ft"
    }
  ]
}
```

Structured Data

- JSON has poor runtime guarantees due to its flexible nature. **Optimize** for compile time guarantees.

```
{
  "type": "employee",
  "nickname": null,
  "attributes":"bae345"
}
```

100%

Of people saddened by bad data*

Structured Data

- Protocol Buffers or Avro (arrow, etc) give you **types** and **compile time guarantees**
- **Versioning your API** now just means sticking to a version of your schema.
- Rely on **release for versioning**.
- Perfect for Pipeline interop

```
message Employee {
  uint64 created  = 1;
  string uuid     = 2;
  string nickname = 3 [default="you"];
  repeated Attribute attributes = 4;
}
```

100%

Of people like when things work between releases!

Structured Data

- Parquet is awesome format for data at rest
- Columnar Data with Column Stats in partition header
- Plays natively with Apache Spark, Presto, Impala and Hive
- Schema changes across partitions can be tough...

```
spark.read
  .parquet("hdfs://.../year=2019")
```

100%

Also like not having to define Encoders

ETL

- **Extract** from one source
- **Transform** data or join with other sources
- **Load** into sink (Kafka, HDFS, etc)

```
def readScores(fileName: String): DataFrame = {
  spark.read.schema(rowSchema).json(s"hdfs://localhost:9000/voiceinsights/es/callscores/$fileName")
}

def flattenCallScore(df: DataFrame): DataFrame = {
  val withMetrics = flattenStruct(df, "metrics")
    .drop("metrics").drop("score_data").drop("dimension_hash").drop("model_version").drop("model_type")
    .withColumn("call_sid_enc", base64encode(col("call_sid")))
    .withColumn("account_sid_enc", base64encode(col("account_sid")))
    .drop("call_sid").drop("account_sid")
    .withColumnRenamed("call_sid_enc", "call_sid")
    .withColumnRenamed("account_sid_enc", "account_sid")

  val withDimensions = flattenStruct(withMetrics, "dimensions").drop("dimensions")
    .withColumn("dimensions_carrier_sid", base64encode(col("dimensions_provider_sid")))
    .drop("dimensions_provider_sid").drop("dimensions_account_sid").drop("dimensions_signaling_ip").drop(

  expandArray(withDimensions, col("insight_tags")).drop("insight_tags")
    .withColumn("ts", functions.from_utc_timestamp(col("iso_timestamp"), "UTC")).drop("iso_timestamp")
    .withColumn("year", functions.year(col("ts")))
    .withColumn("month", functions.month(col("ts")))
    .withColumn("day", functions.dayofmonth(col("ts")))
    .withColumn("hour", functions.hour(col("ts")))
    .withColumn("dayofyear", functions.dayofyear(col("ts")))
}

def writeScores(df: DataFrame, path: String): Unit = {
  df
  .repartitionByRange(4, col("day"))
  .sortWithinPartitions(col("ts"))
  .write
  .partitionBy("year","month","day")
  .mode(SaveMode.Append)
  .format("parquet")
  .option("path", path)
  .save
}
```

Own your Data

- What does your data need to do for you?
- How is storing the data going to help your company or organization?
- Does your data meet GDPR guidelines?

Data Engineering

What we've learned

1. **Own** your data. You or your team holds a very valuable asset in your hands and you are the stewards of that Data.

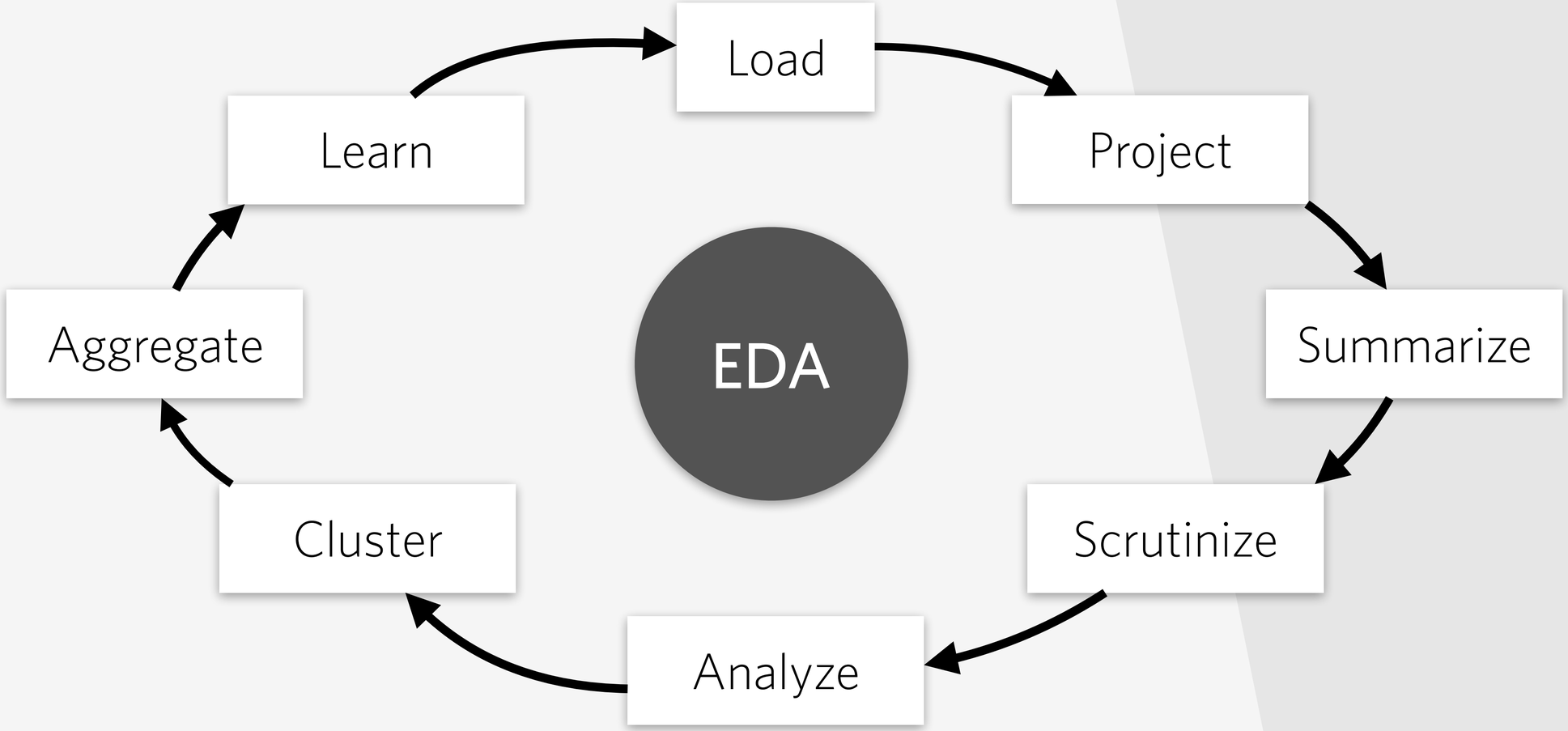
2. **Pipelines** should be built with versioned structured data. **Compile Time guarantees** help keep pipelines healthy.

3. Store only what you will need and **compact your parquet** to reduce FileSystem.io

EXPLORATORY DATA ANALYSIS

http://bit.ly/odsc-east-2019-realish-predict

Exploratory Data Analysis



EDA

Zeppelin Notebook Job Search anonymous

/ODSC-East-2019/GentleIntroduction Head default

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
//import com.twilio.open.odsc.realish._
```

Took 2 sec. Last updated by anonymous at April 30 2019, 7:35:24 PM.

```
%spark
val scoresData = spark.read.option("basePath", "hdfs://localhost:9000/user/ds/odsc/east
    /realish/eda/callscores/").parquet("hdfs://localhost:9000/user/ds/odsc/east/realish/eda
```

FINISHED

```
scoresData: org.apache.spark.sql.DataFrame = [timestamp: bigint, score: float ... 33 more fi
elds]
```

Took 0 sec. Last updated by anonymous at April 30 2019, 7:35:24 PM. (outdated)

```
%spark
scoresData.inputFiles
```

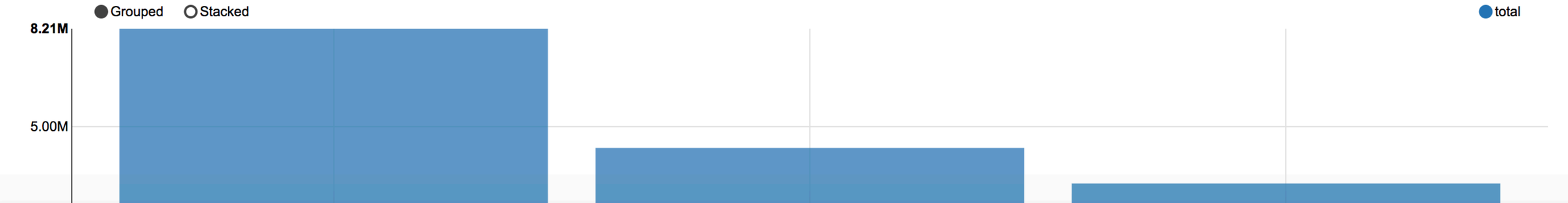
FINISHED

Took 1 sec. Last updated by anonymous at April 30 2019, 7:35:25 PM. (outdated)

```
%sql
select count(*) as total, day from scores where year=2019 and month=1 and day in (25, 26, 27) GROUP BY day
```

FINISHED

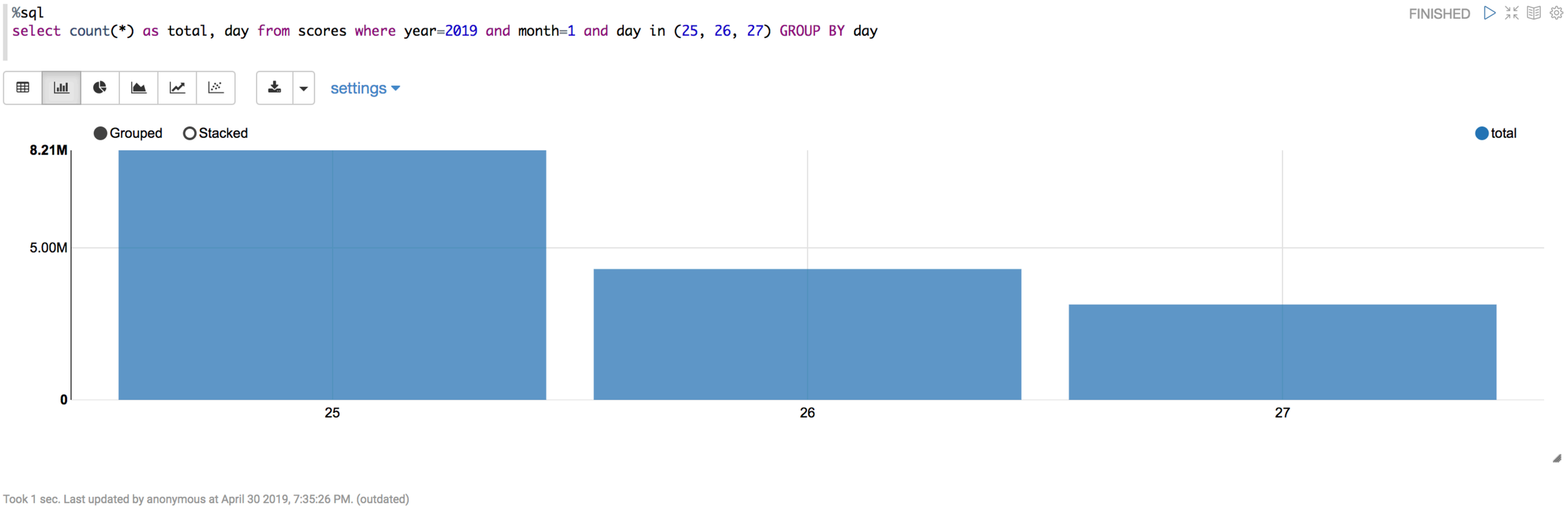
settings



EDA

```
%sql
select count(*) as total, day from scores where year=2019 and month=1 and day in (25, 26, 27) GROUP BY day
```

FINISHED



Took 1 sec. Last updated by anonymous at April 30 2019, 7:35:26 PM. (outdated)

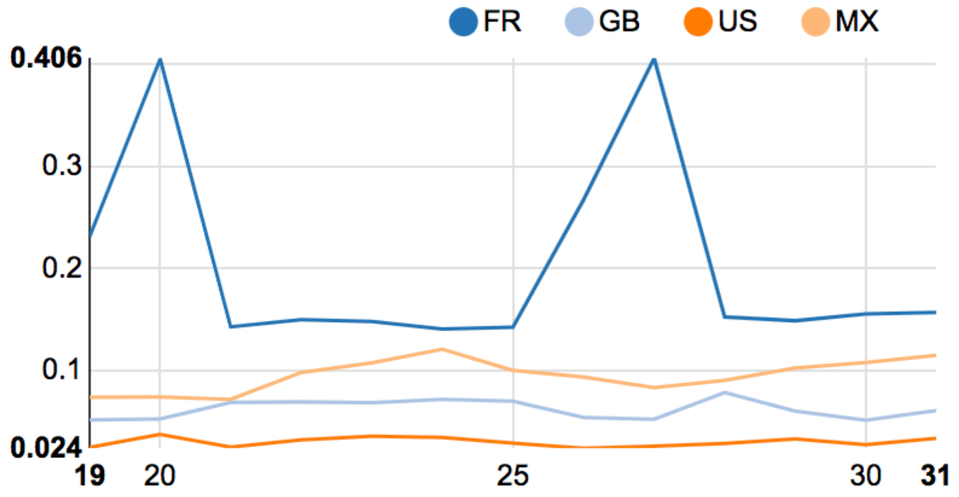
https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/odsc_east_intro_part1.json

EDA

```
%sql
select dayofyear, dimensions_country_code, avg(pktloss_sum
    /total) as avg_pktloss_percentage from
    carrier_by_country_scores where
    dimensions_country_code in ("US", "FR", "GB", "MX")
```

FINISHED

settings

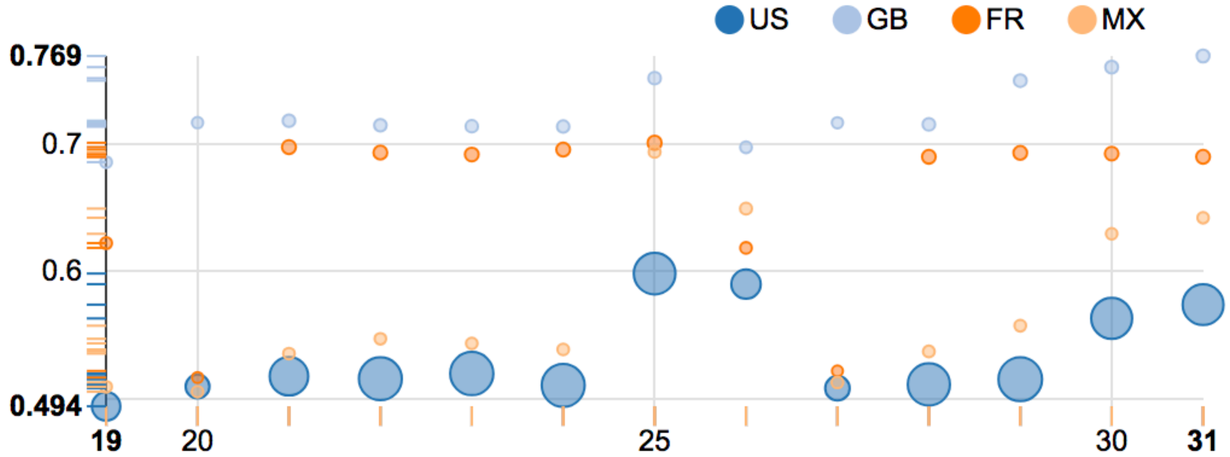


Took 9 sec. Last updated by anonymous at April 30 2019, 7:35:59 PM. (outdated)

```
%sql
select count(*) as total, min(score) as min_score, avg(score) as
    mean_score, max(score) as max_score, dimensions_country_code,
    dayofyear from scores where year=2019 and month=1 and
    dimensions_country_code IN ("US","FR","GB", "MX") GROUP BY
    dimensions_country_code,dayofyear HAVING total > 1000 ORDER BY
```

FINISHED

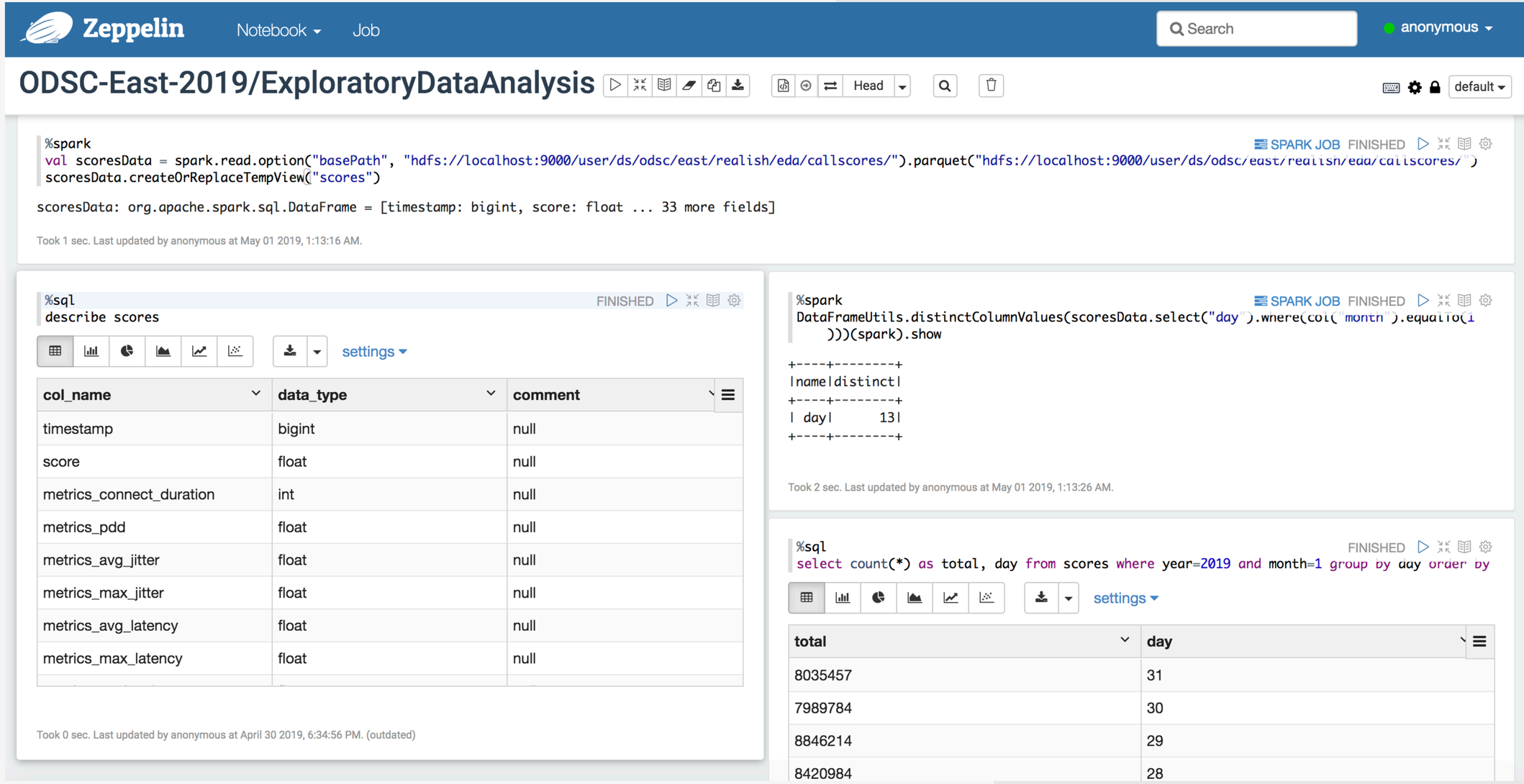
settings



Took 8 sec. Last updated by anonymous at April 30 2019, 7:36:07 PM.

https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/odsc_east_intro_part1.json

EDA



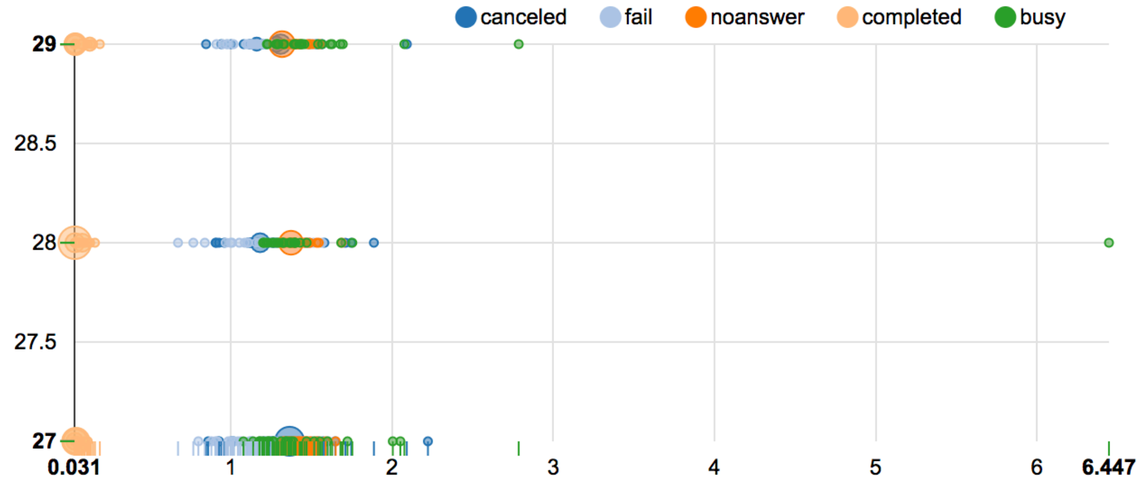
https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/odsc_east_eda_part2.json

EDA

```
%sql
select dimensions_call_state, day, hour, avg(metrics_pdd) as mean_pdd, percentile_approx
    (metrics_pdd, 0.25) as p25, percentile_approx(metrics_pdd, 0.5) as p50,
    percentile_approx(metrics_pdd, 0.75) as p75, max(metrics_pdd) as max_pdd from
```

FINISHED

settings

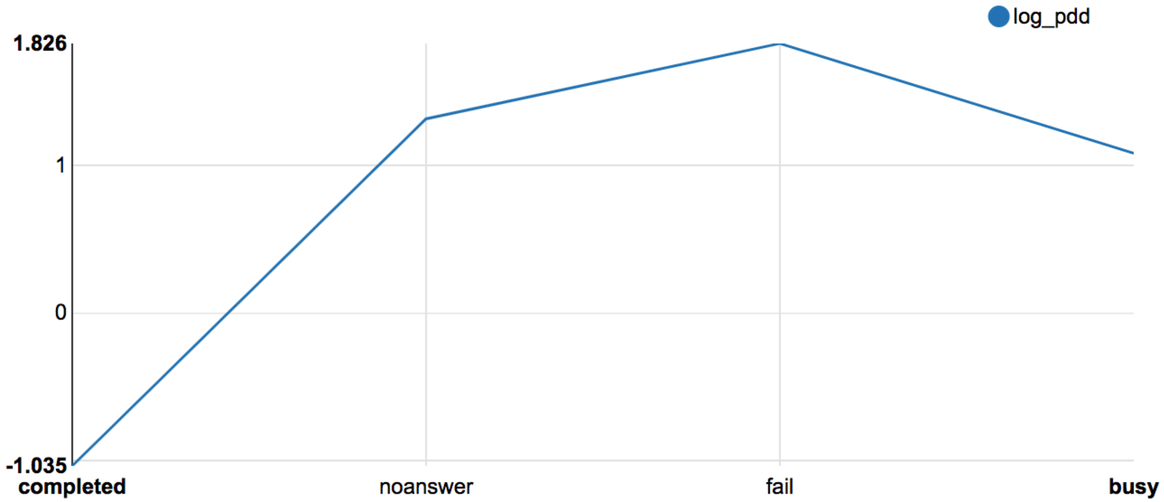


Took 25 sec. Last updated by anonymous at April 30 2019, 6:16:30 PM. (outdated)

```
%sql
select log(metrics_pdd) as log_pdd, dimensions_call_state from pdd_call_states
```

FINISHED

settings



Took 0 sec. Last updated by anonymous at April 30 2019, 7:27:23 PM. (outdated)

https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/odsc_east_eda_part2.json

Exploratory Data Analysis

What we've learned

1. Uncover the **Shape of your Data**

2. **Find interesting relationships** between values in your data

3. Understand what is Normal.

CLUSTERING AND TAGGING

http://bit.ly/odsc-east-2019-realish-predict

Clustering and Tagging

- Look at the Universal Set vs SubSet Trends
- Find Patterns in the Noise
- Understand where natural thresholds reside

Clustering and Tagging

```
%spark                                                    FINISHED
val packetLossByCountry = scoresData
  .select("day","account_sid","dimensions_country_code"
      ,"metrics_connect_duration"
      ,"metrics_packet_loss_percentage","score")
  .where(col("day").equalTo(28)
    .and(col("account_sid").isNotNull
      .and(col("dimensions_country_code").notEqual("")
        .and(col("metrics_connect_duration").notEqual(0)
          .and(col("metrics_packet_loss_percentage"
```

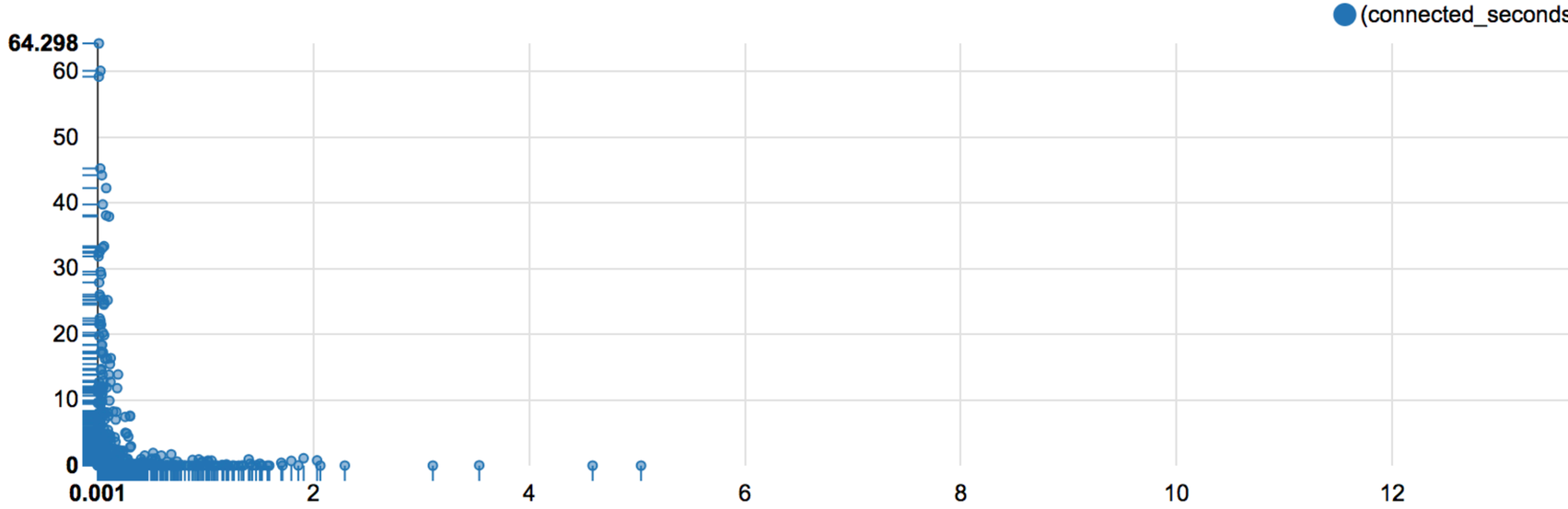
Took 0 sec. Last updated by anonymous at May 01 2019, 1:27:24 AM. (outdated)

~2SEC

tolerance of bad packet loss.

```
%sql                                                      SPARK JOB FINISHED
select (metrics_connect_duration/1000) as connected_seconds, metrics_packet_loss_percentage as loss_percentage, scor
from loss_by_country
where dimensions_country_code in ("US","MX","JP","BR")
```

settings



Took 0 sec. Last updated by anonymous at May 01 2019, 1:27:08 AM. (outdated)

https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/projections_and_naive_bayes.json

Clustering and Tagging

What we've learned

1. Simple **GroupBy and Filtering** yields Insight into trends in your data.

2. **Applying tags** at specific thresholds in your data will allow you to find signal in the noise.

STREAMING PREDICTIONS

http://bit.ly/odsc-east-2019-realish-predict

Streaming Predictions

```
val streamingQuery = metricData.toDF()
  .withWatermark("timestamp", "2 hours")
  .groupBy(col("metric"), col("countryCode"), window($"timestamp", "5 minutes
  .agg(
    min("value") as "min",
    avg("value") as "mean",
    max("value") as "max",
    count("*") as "total"
  )
  .writeStream
  .format("memory")
  .queryName("datastream")
  .outputMode(OutputMode.Append())
  .trigger(processingTimeTrigger)
  .start()

metricData.addData(backingData)

streamingQuery.processAllAvailable()

spark.sql("select * from datastream").show(20, false)

val checkChange = spark.sql("select * from datastream")
  .groupBy("metric","countryCode")
  .agg(
    sum("total") as "total",
    avg("mean") as "mean"
  )

checkChange.show(20, false)

// now can do interesting things with minor back tracking...
```

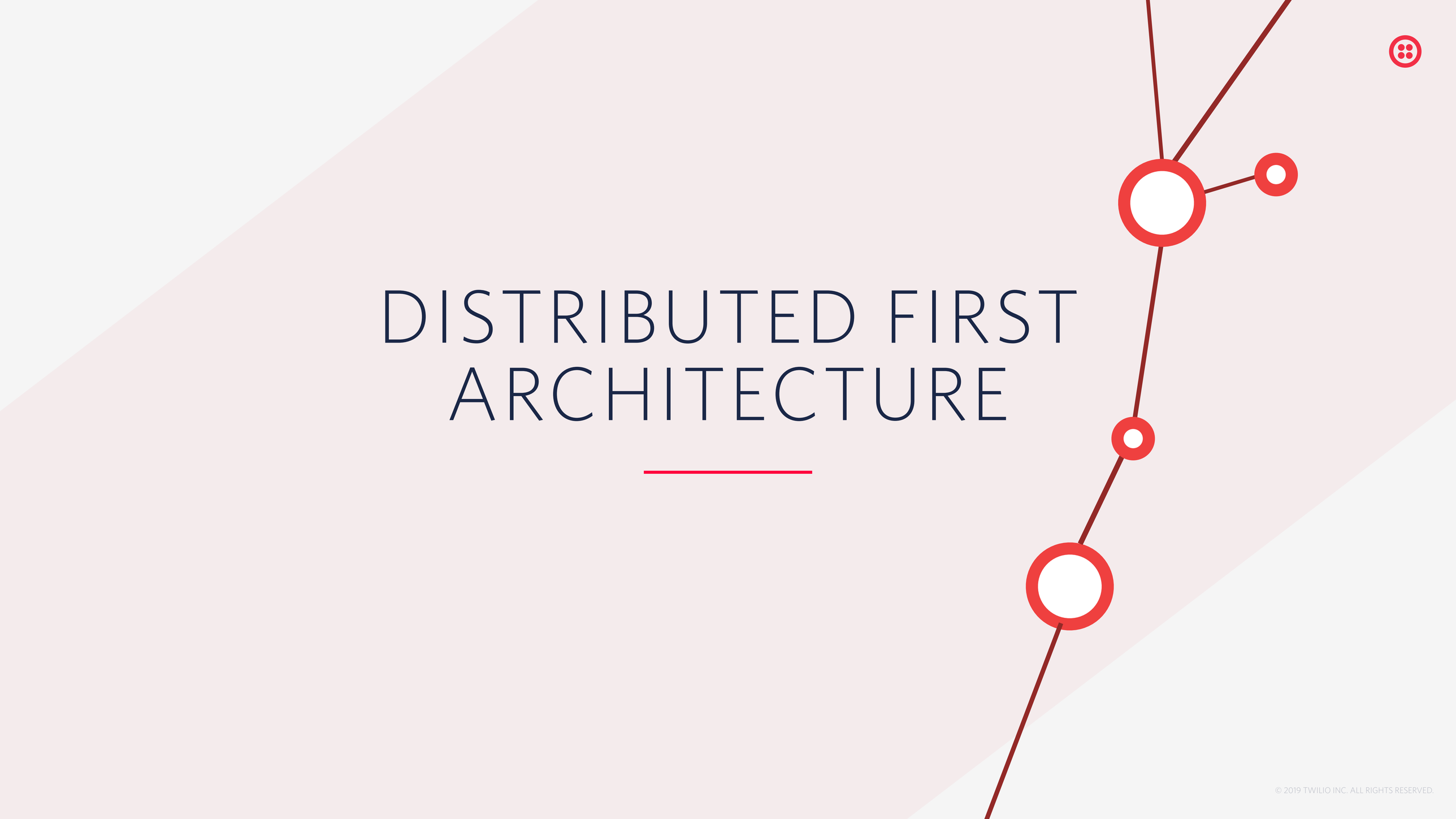
Streaming Predictions

What we've learned

1. Given a simple streaming application. You can now go off and build a masterpiece!

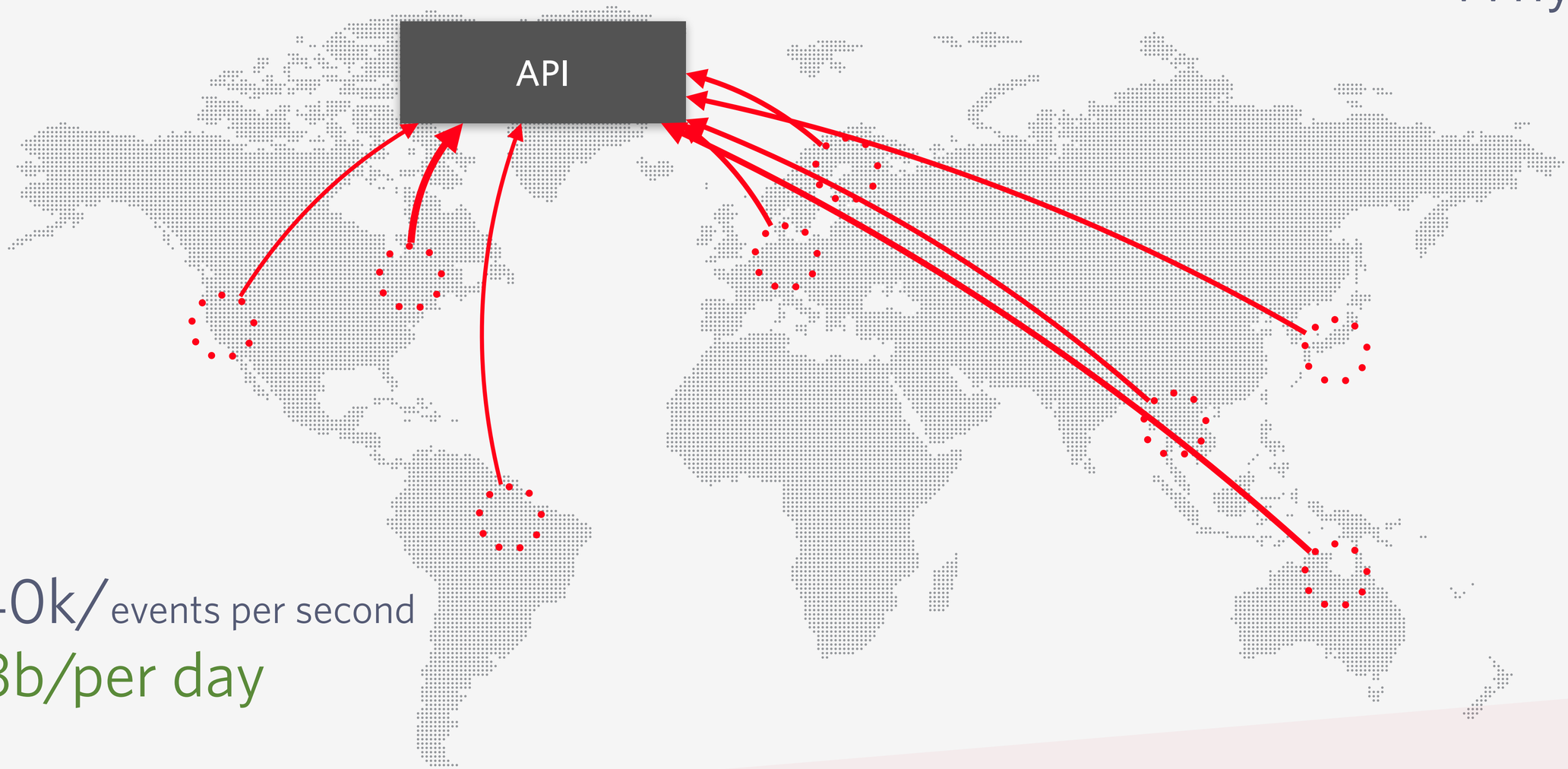
2. Remember. Keep your record sizes small or you'll have to cough up a ton of CPU and RAM to handle those poor choices. We did!

DISTRIBUTED FIRST ARCHITECTURE



DISTRIBUTED FIRST ARCHITECTURE

Why?

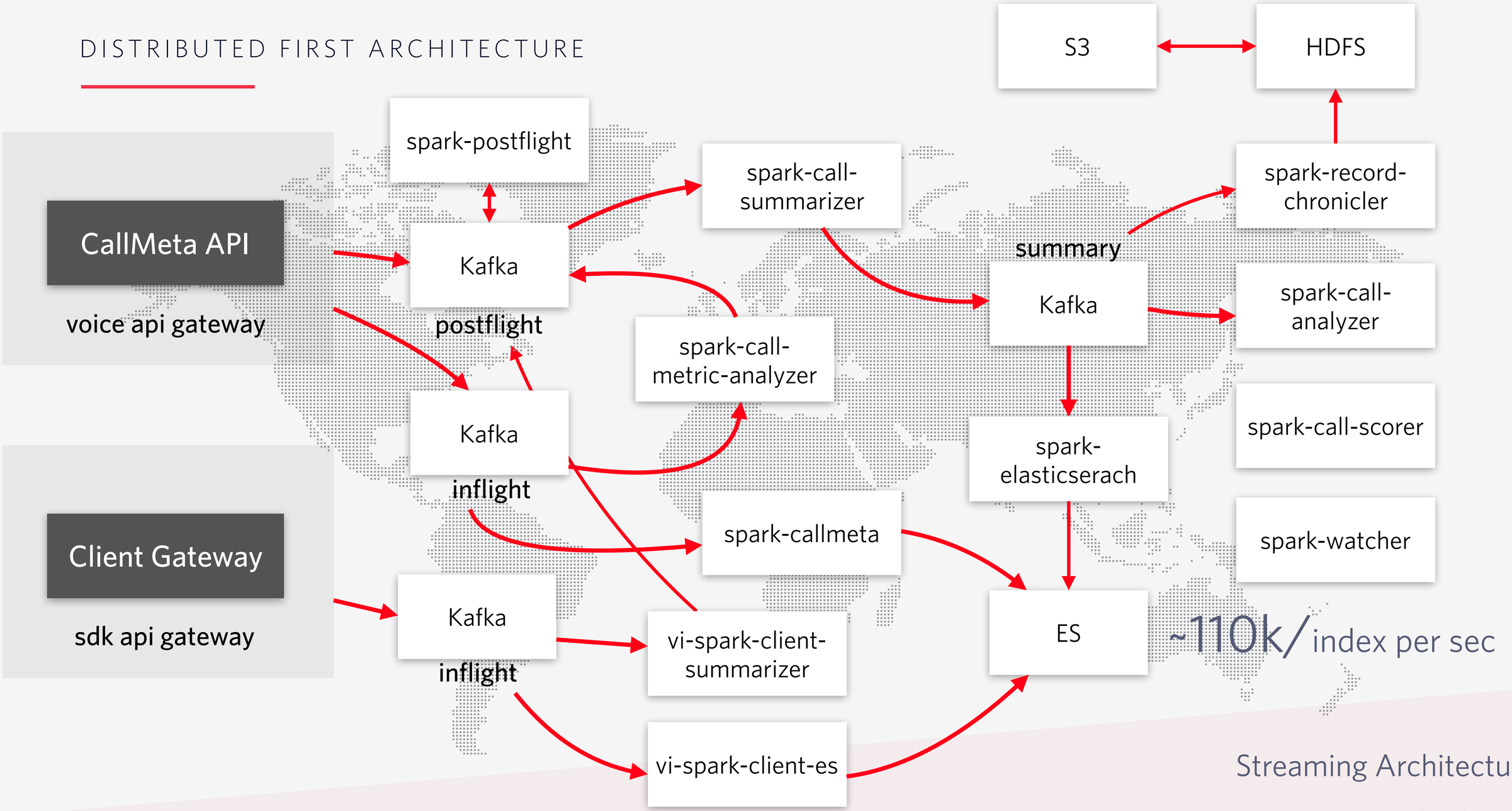


~40k/ events per second

>3b/per day

How

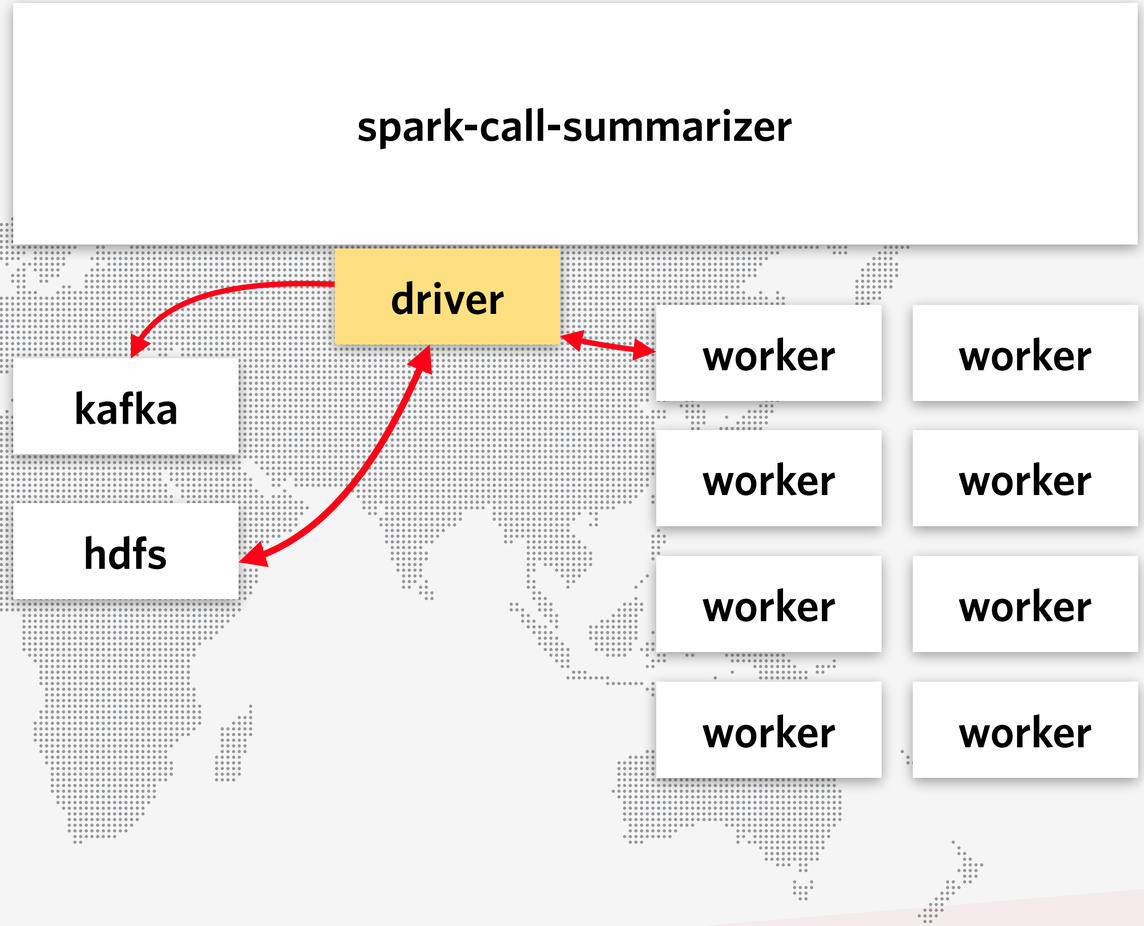
DISTRIBUTED FIRST ARCHITECTURE



Streaming Architecture

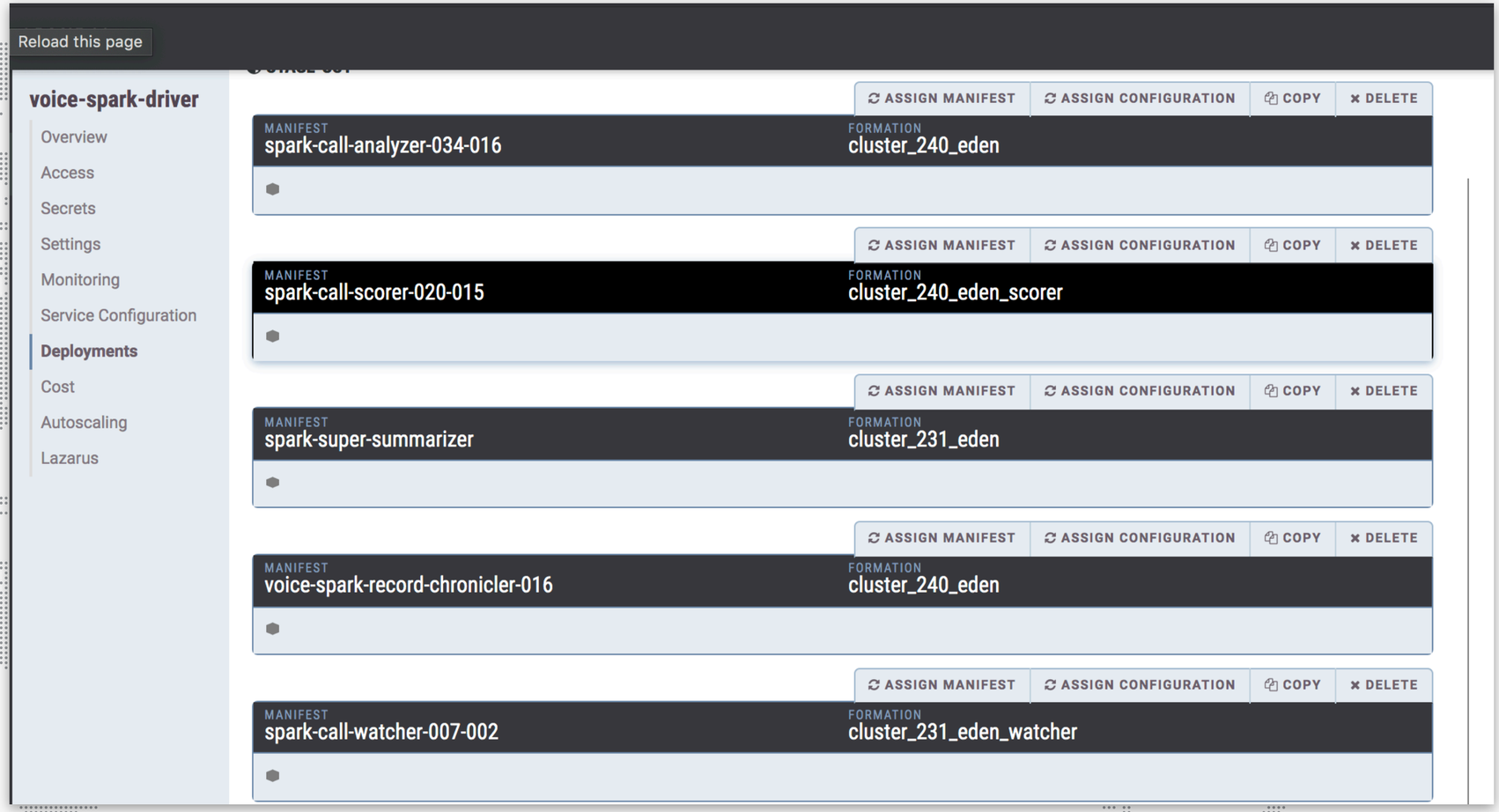
SPARK APP ARCHITECTURE

- Each App **does one thing well**.
- Apps can be **deployed via Admiral**.
- Apps can be scaled up or down independently.
- Apps can be run in recovery or real-time mode
- Apps **checkpoint state to HDFS** and state survive between releases and upgrades



SPARK APP ARCHITECTURE

- Apps can be deployed to **one or many logical spark clusters**



twitter: @newfront
blog: blog.twilio.com

THANK YOU