



Introductions

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

<http://bit.ly/odsceastrealtime>

@newfront | @twilio





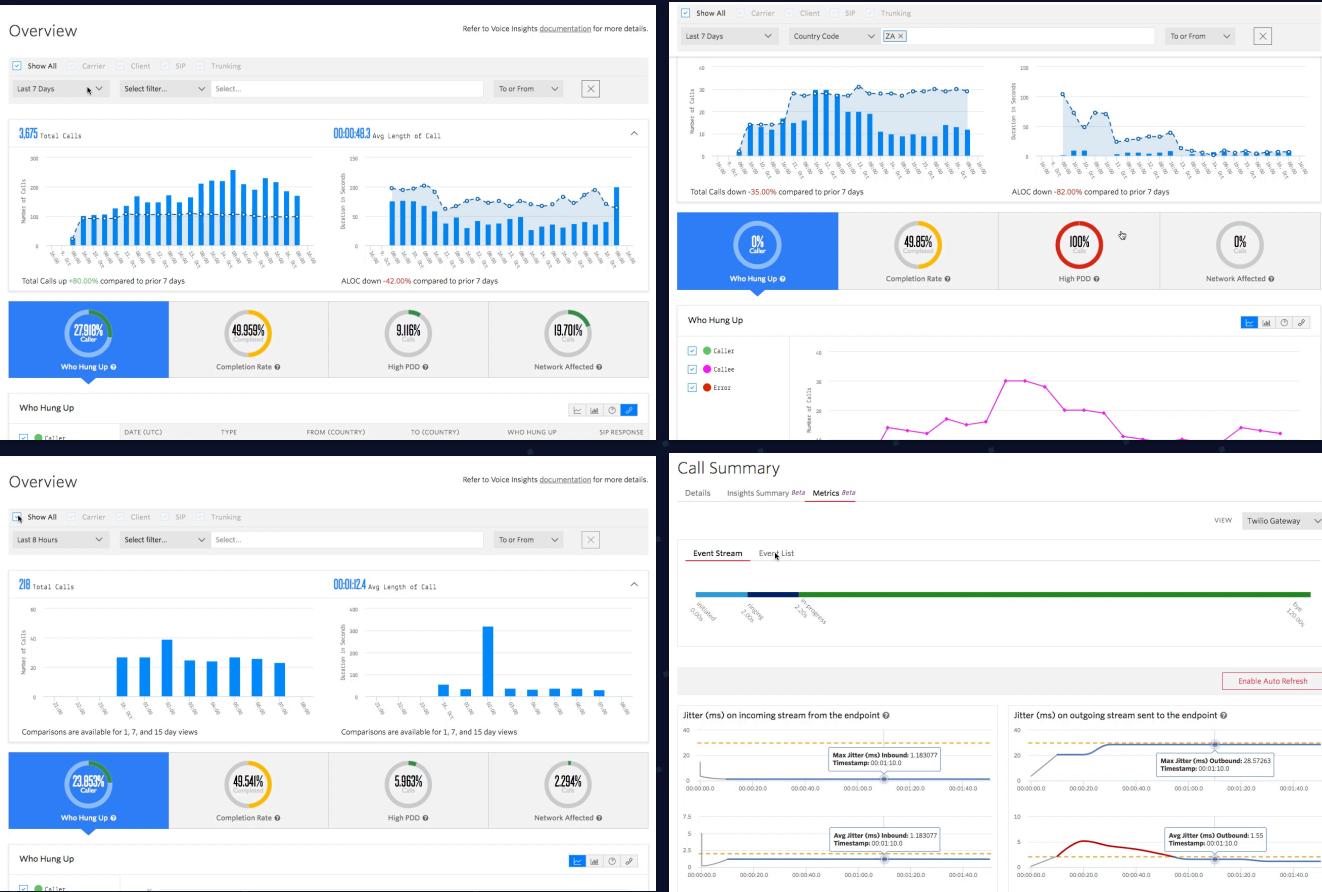
What I Do Voice Insights

Voice Analytics Platform

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

Supports internal and external customers.

*Near Real-Time





WORKSHOP

<http://bit.ly/odsceastrealtime>

Spark Primer

Exploratory Analysis With
Spark SQL

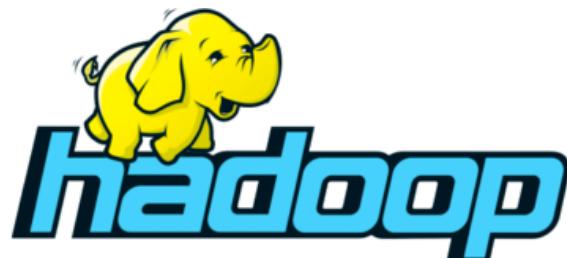
Finding Structure with
Unsupervised Learning

Workshop



Before we begin

Technologies



version: 2.7.7



version: 2.4.0



Apache Zeppelin

version: 0.8.1



Spark Basics : Generating DataFrames

```
import spark.implicits._

case class CallNetworkEvent(timestamp: Long, event: String, metric: Float, dimensions: Map[String, String])

val callEventDF = Seq(CallNetworkEvent(100L, "high_jitter", 25.0f, Map("region" -> "us"))).toDF
```

FINISHED ▶ ✎ 📄⚙️

```
callEventDF.printSchema()
```

```
root
|-- timestamp: long (nullable = false)
|-- event: string (nullable = true)
|-- metric: float (nullable = false)
|-- dimensions: map (nullable = true)
|   |-- key: string
|   |-- value: string (valueContainsNull = true)
```

FINISHED ▶ ✎ 📄⚙️

Took 0 sec. Last updated by anonymous at February 17 2019, 3:44:52 PM.



Spark Basics : Loading JSON via HDFS

```
/* Load up the Wine Reviews Dataset (via kaggle) */  
val dataLocation = "/user/ds/odsc/"  
val wineReviewsDf = spark.read.option("inferSchema", "true").json(s"$dataLocation/winereviews.json")
```

SPARK JOB FINISHED ▶ ✎ 🗑️ 🔍

```
wineReviewsDf.printSchema
```

```
root  
|-- country: string (nullable = true)  
|-- description: string (nullable = true)  
|-- designation: string (nullable = true)  
|-- points: string (nullable = true)  
|-- price: long (nullable = true)  
|-- province: string (nullable = true)  
|-- region_1: string (nullable = true)  
|-- region_2: string (nullable = true)  
|-- taster_name: string (nullable = true)  
|-- taster_twitter_handle: string (nullable = true)  
|-- title: string (nullable = true)  
|-- variety: string (nullable = true)  
|-- winery: string (nullable = true)
```

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/user/ds/odsc Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	shaines	supergroup	75.61 MB	2/17/2019, 4:00:17 PM	1	128 MB	winereviews.json

Hadoop, 2018.

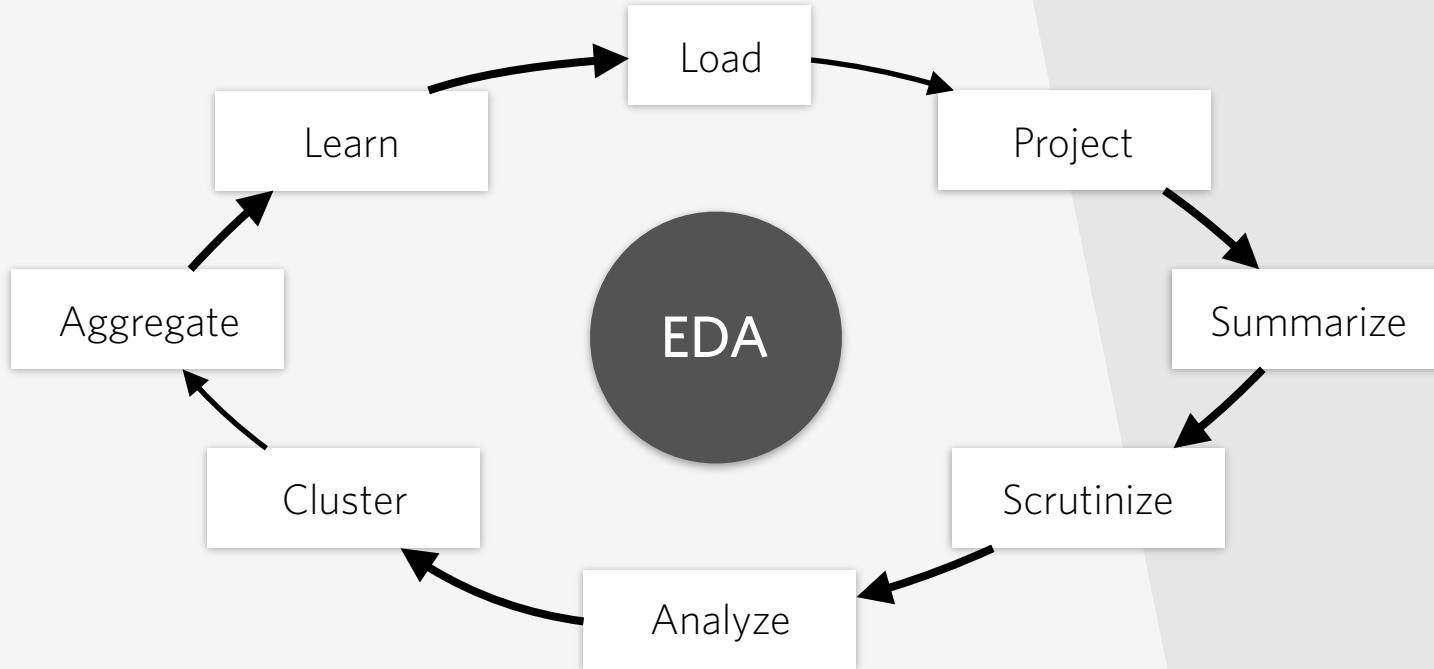


Warm Up

Next Up

1. Spark Basics (Quick Overview)
2. Exploratory Analytics With Spark SQL
3. Finding Structure with Unsupervised Learning
4. Teaser: Workshop Goals

Exploratory Data Analysis





Exploratory Analytics With Spark SQL

Mental Model

1. Understand the Fields and Types of the Data
2. Investigate the properties of the data set (missing data, distinct, ranges)
3. Generate Column Statistics (min, p25, p50, p75, etc)
4. Imputation Strategies for missing data (explore wine prices and fill)



Exploratory Analytics : Column Statistics

1. Create a Temporary SQL Table
2. Enables us to query with simple
SQL vs Scala API
3. Enables Zeppelin UI to easily
print graphs etc as well.

```
/* Load up the Wine Reviews Dataset (via kaggle) */  
val dataLocation = "/user/ds/odsc/"  
val wineReviewsDf = spark.read.option("inferSchema", true).json(s"$dataLocation")  
wineReviewsDf.cache()
```

Took 9 sec. Last updated by anonymous at May 24 2019, 5:56:30 PM.

```
wineReviewsDf.createOrReplaceTempView("reviews")
```

Took 1 sec. Last updated by anonymous at February 18 2019, 10:27:50 AM.



Exploratory Analytics : Projecting / Viewing Data

```
%sql  
select country, designation, points, price, variety, winery from reviews  
where designation IS NOT NULL  
order by price desc
```

SPARK JOB FINISHED ▶ ✎ 📄 ⚙



settings ▾

country	designation	points	price	variety	wine
US	Roger Rose Vineyard	91	2013	Chardonnay	Blair
Austria	Ried Loibenberg Smaragd	94	1100	Grüner Veltliner	Emmerich
Portugal	90-year Old Tawny	97	1000	Port	W. & J. G
Portugal	Colheita White	94	980	Port	Kopke
Italy	Riserva	94	900	Sangiovese Grosso	Biondi S
Australia	Grange	99	850	Shiraz	Penfolds
Australia	Grange	98	850	Shiraz	Penfolds
Australia	Hill of Grace	96	820	Shiraz	Henschke
France	Moroaues	87	800	Rosé	Domaine



Exploratory Analytics : Finding Missing Data

```
def missingValues(df: DataFrame) = {  
    df.schema.map { r => (r.name, df.where(col(r.name).isNull).count) }.toDF("name", "missing")  
}
```

FINISHED ▶ ↻ 🔍 ⚙

```
missingValues(wineReviewsDf).show
```

SPARK JOBS FINISHED ▶ ↻ 🔍 ⚙

```
+-----+-----+  
|           name | missing |  
+-----+-----+  
|      country |     63 |  
| description |      0 |  
| designation |  37465 |  
|      points |      0 |  
|      price |  8996 |  
| province |     63 |  
| region_1 |  21247 |  
| region_2 |  79460 |  
| taster_name |  26244 |  
|taster_twitter_handle |  31213 |  
|      title |      0 |  
|   variety |      1 |  
|   winery |      0 |  
+-----+-----+
```



Exploratory Analytics : Finding Distinct Values

```
def distinctValues(df: DataFrame) = {
    df.schema.map { r =>
        (r.name, df.select(col(r.name)).where(col(r.name).isNotNull).distinct.count)
    }.toDF("name", "distinct")
}
```

FINISHED ▶ ↻ ↺ 📄⚙️

```
distinctValues(wineReviewsDf).show(50, false)
```

SPARK JOBS FINISHED ▶ ↻ ↺ 📄⚙️

name	distinct
country	43
description	119955
designation	37979
points	21
price	390
province	425
region_1	1229
region_2	17
taster_name	19
taster_twitter_handle	15
title	118840
variety	707
winery	16757



Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data

```
def columnStats(df: DataFrame) = {
    df.schema.map { r =>
        val colStats = df.select(col(r.name)).where(col(r.name).isNotNull).describe()
        /*colStats.printSchema*/
        colStats.show()
    }
}

columnStats: (df: org.apache.spark.sql.DataFrame)Seq[Unit]
```

Took 1 sec. Last updated by anonymous at February 18 2019, 10:26:22 AM. (outdated)

```
columnStats(wineReviewsDf)
```



Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data
2. View The summary stats.

```
+-----+  
|summary| country|  
+-----+  
| count | 129908|  
| mean  | null  |  
| stddev| null  |  
| min   | Argentinal  
| max   | Uruguay|  
+-----+  
  
+-----+-----+  
|summary|      description|  
+-----+-----+  
| count | 129971|  
| mean  | null  |  
| stddev| null  |  
| min  | "Chremisa," the a...|  
| max  | "Wow" is the firs...|  
+-----+
```

```
columnStats(wineReviewsDf)
```

```
+-----+  
|summary|      designation|  
+-----+  
| count | 92506|  
| mean  | 1494.4644378698224|  
| stddev| 7115.55431803001|  
| min  |#19 Phantom Limb ...|  
| max  | "P"|  
+-----+  
  
+-----+-----+  
|summary|      points|  
+-----+-----+  
| count | 129971|  
| mean  | 88.44713820775404|  
| stddev| 3.0397302029160067|  
| min  | 100|  
| max  | 99|  
+-----+
```



Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data
2. View The summary stats.
3. Find Issues with the Data

```
+-----+  
|summary| country|  
+-----+  
| count | 129908|  
| mean  | null  |  
| stddev| null  |  
| min   | Argentinal  
| max   | Uruguay|  
+-----+  
  
+-----+-----+  
|summary|      description|  
+-----+-----+  
| count | 129971|  
| mean  | null  |  
| stddev| null  |  
| min  | "Chremisa," the a...|  
| max  | "Wow" is the firs...|  
+-----+
```

```
columnStats(wineReviewsDf)
```

```
+-----+  
|summary|      designation|  
+-----+  
| count | 92506|  
| mean  | 1494.4644378698224|  
| stddev| 7115.55431803001|  
| min  |#19 Phantom Limb ...|  
| max  | "P"|  
+-----+  
  
+-----+-----+  
|summary|      points|  
+-----+-----+  
| count | 129971|  
| mean  | 88.44713820775404|  
| stddev| 3.0397302029160067|  
| min  | 100|  
| max  | 99|  
+-----+
```



Exploratory Analytics : Build Your Own ColumnStats

```
def calculateColumnAggregates(df: DataFrame) = {
    df.schema.map { r =>
        val name = r.name
        val colData = df.select(col(name)).where(col(name).isNotNull)
        colData.agg(
            lit(name) as "colname",
            count(name) as "count",
            min(name) as "min",
            avg(name) as "avg",
            stddev_pop(name) as "sd_pop",
            stddev_samp(name) as "sd_sample",
            max(name) as "max"
        ).show()
    }
}
```



Exploratory Analytics : Fixing Data

```
val fixedReviews = wineReviewsDf  
  .withColumn("num_points", 'points.cast("Double"))  
  .drop("points")  
  .withColumnRenamed("num_points", "points")
```

FINISHED ▶ ↻ 🔍 ⚙️

```
calculateColumnAggregates(fixedReviews)
```

SPARK JOBS FINISHED ▶ ↻ 🔍 ⚙️

```
+-----+-----+-----+-----+-----+  
| colname | count | min | avg | sd_pop | sd_sample | max |  
+-----+-----+-----+-----+-----+  
| variety | 129970 | Abouriou | null | null | Žilavka |  
+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+  
| colname | count | min | avg | sd_pop | sd_sample | max |  
+-----+-----+-----+-----+-----+  
| winery | 129971 | 1+1=3 | Infinity | NaN | NaN | Štokal |  
+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+  
| colname | count | min | avg | sd_pop | sd_sample | max |  
+-----+-----+-----+-----+-----+  
| points | 129971 | 80.0 | 88.44713820775404 | 13.0397185090148677 | 13.0397302029160067 | 100.0 |  
+-----+-----+-----+-----+-----+
```



Exploratory Analytics : Percentile Data

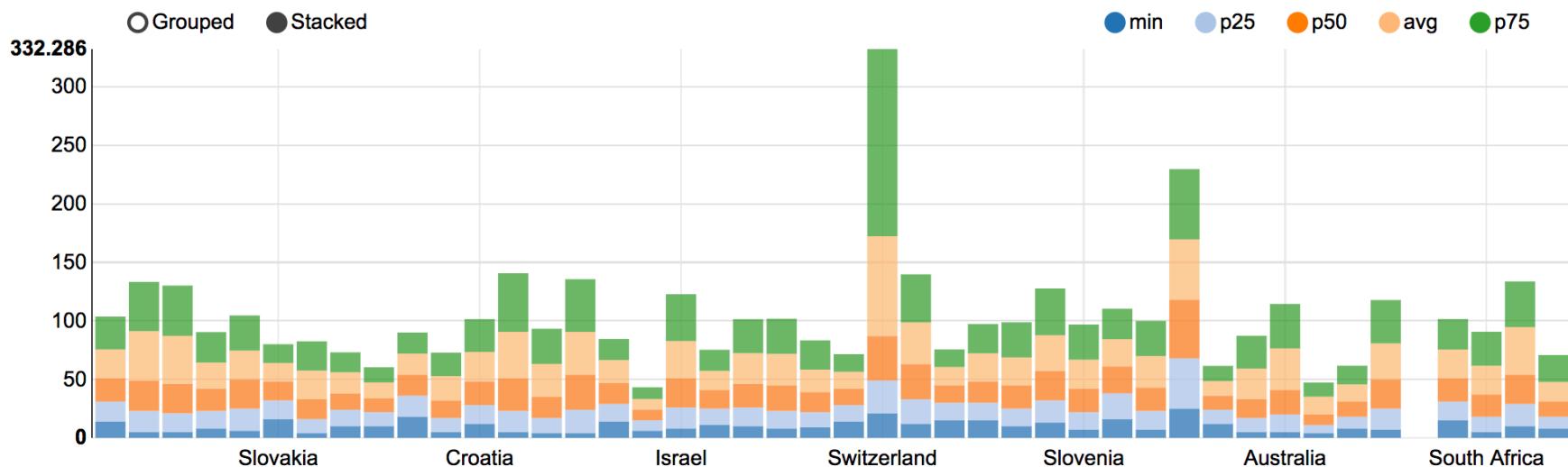
%sql

```
select country, count(*) as total, min(price) as min, percentile_approx(price, 0.25) as p25, percentile_approx(price, 0.5) as p50, percentile_approx(price, 0.75) as p75, percentile_approx(price, 0.90) as p90, percentile_approx(price, 0.99) as p99, max(price) as max, avg(price) as avg, stddev_pop(price) as stddev
from freviews group by country
```

SPARK JOBS FINISHED ▶ ↻ ↺ ↻ ⚙



settings ▾





Exploratory Analytics : Price Field Imputation

FINISHED ▶ ↻ ⌂

```
/* Price Imputation */

val fullPriceData = spark.sql("select country, count(*) as total, min(price) as min, percentile_approx(price, 0.25) as p25, percentile_approx(price, 0.5) as p50, percentile_approx(price, 0.75) as p75, percentile_approx(price, 0.90) as p90, percentile_approx(price, 0.99) as p99, max(price) as max, avg(price) as avg, stddev_pop(price) as sd_pop, stddev_samp(price) as sd_sample from winereviews group by country")

val missingPriceRows = fixedReviews
  .where(col("price").isNull.and(col("country").isNotNull)).groupBy("country")
  .agg(count("*") as "missing")

val joinedWithOtherPrices = missingPriceRows
  .join(fullPriceData, "country")
  .withColumn("miss_perc", ((col("missing")/col("total"))*100).cast("Float"))
  .withColumn("iqr", (col("p75")-col("p25")).cast("Float"))
  .withColumn("min_max_dist", (col("max")-col("min")).cast("Integer"))
  .withColumn("median_mean_dist", (col("p50")-col("avg")).cast("Float")).where(col("total") > 1)

joinedWithOtherPrices.createOrReplaceTempView("priceimputation")
```

joinedWithOtherPrices.printSchema

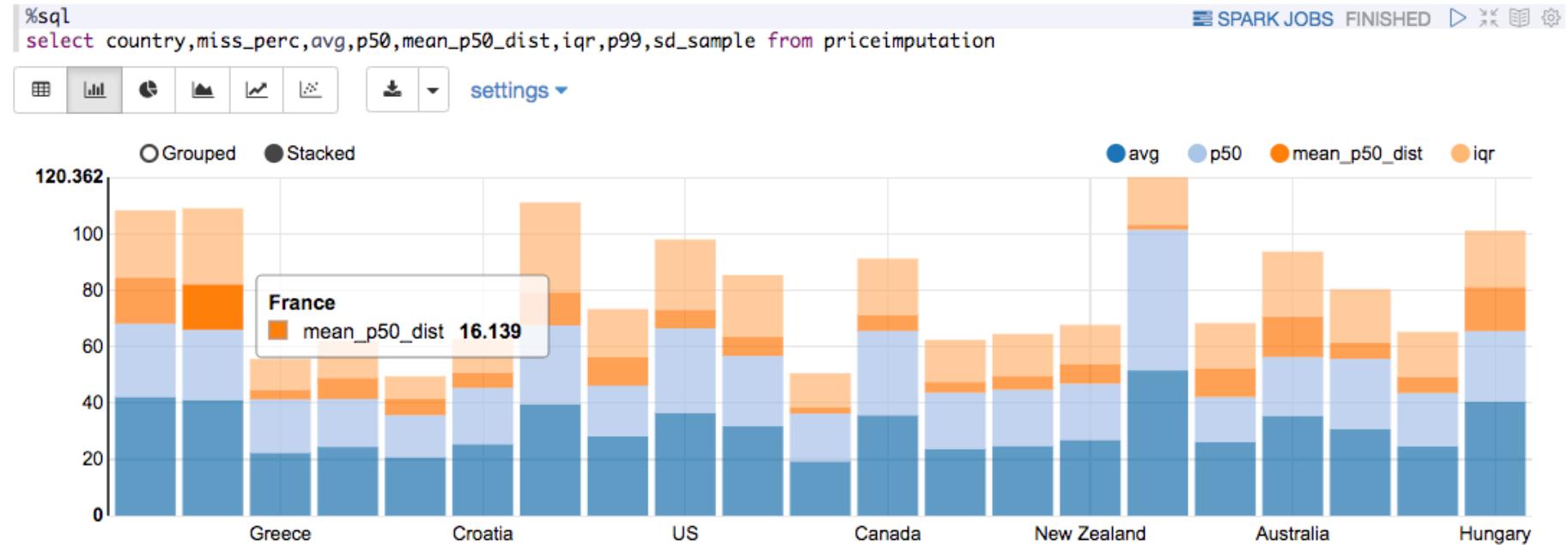
root

```
|-- country: string (nullable = true)
|-- missing: long (nullable = false)
|-- total: long (nullable = false)
|-- min: double (nullable = true)
|-- p25: double (nullable = true)
|-- p50: double (nullable = true)
|-- p75: double (nullable = true)
|-- p90: double (nullable = true)
|-- p99: double (nullable = true)
|-- max: double (nullable = true)
|-- avg: double (nullable = true)
|-- sd_pop: double (nullable = true)
|-- sd_sample: double (nullable = true)
|-- miss_perc: float (nullable = true)
|-- iqr: float (nullable = true)
|-- min_max_dist: integer (nullable = true)
|-- median_mean_dist: float (nullable = true)
```

1. Find missing prices by country and join with price statistics data
2. Generate new temp table to visualize data.



Exploratory Analytics : Price Field Imputation





Exploratory Analytics : Price Field Imputation

```
/* Use imputation data to fill in unknown prices */
val missingPriceDf = fixedReviews.where(col("price").isNull.and(col("country").isNotNull))
val normalDf = fixedReviews.where(col("price").isNotNull)

val imputedPrice = missingPriceDf
  .join(joinedWithOtherPrices, "country")
  .withColumn("price", when(col("avg")>col("p50"), col("p50")).otherwise(col("avg")))
  .drop("total", "min", "p25", "p50", "p75", "p90", "p99", "max", "avg", "sd_pop", "sd_sample",
    "miss_perc", "iqR", "min_max_dist", "median_mean_dist", "missing")
/* both schemas must match for union */
val filledPriceDf = normalDf.union(imputedPrice)
filledPriceDf.cache()
filledPriceDf.createOrReplaceTempView("winereviews")
```

```
filledPriceDf
  .where(col("price").isNull)
  .agg(count("*") as "total").show
```

```
+----+
|total|
+----+
|    0|
+----+
```

1. Simplified: Mean or Median Fill based on skew
2. Can be smarter - Lookup [MICE](#) Strategy



Warm Up

Next Up

1. Spark Basics (Quick Overview)
2. Exploratory Analytics With Spark SQL
3. Finding Structure with Unsupervised Learning
4. Teaser: Workshop Goals

Clustering and Statistics

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

Clustering and Tagging

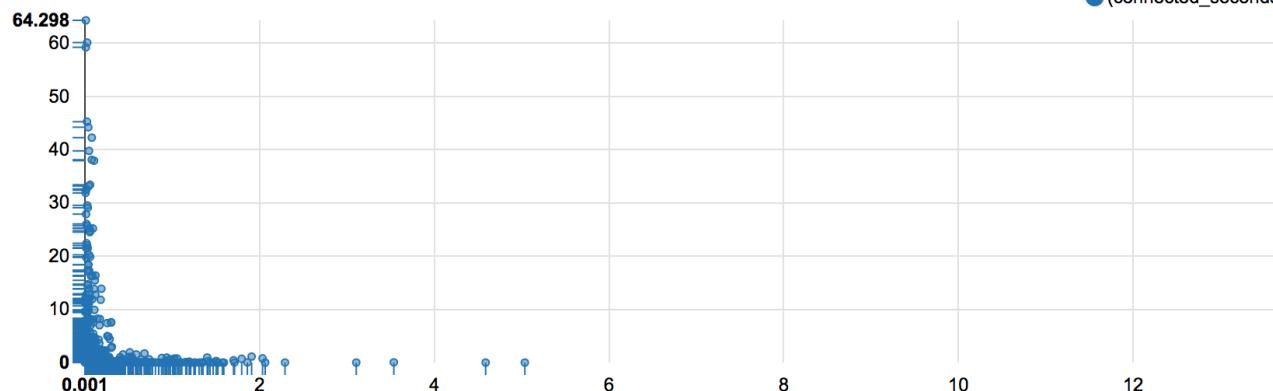
```
%spark  
val packetLossByCountry = scoresData  
  .select("day", "account_sid", "dimensions_country_code"  
    , "metrics_connect_duration"  
    , "metrics_packet_loss_percentage", "score")  
  .where(col("day").equalTo(28)  
    .and(col("account_sid").isNotNull  
    .and(col("dimensions_country_code").notEqual("")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



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

SPARK JOB FINISHED ▶



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



Exploratory Analytics : Unsupervised Learning

```
/* How to use KMeans Model */  
  
/* which countries can be filled in by varietal */  
/* note: indexers can't handle null values */  
val normalizedWine = filledPriceDf.where(col("country").isNotNull.and(col("variety").isNotNull))  
normalizedWine.cache()  
  
val indexerCountry = new StringIndexer().setInputCol("country").setOutputCol("country_index")  
val encoderCountry = new OneHotEncoder().setInputCol("country_index").setOutputCol("country_encoded")  
val indexerVariety = new StringIndexer().setInputCol("variety").setOutputCol("variety_index")  
val encoderVariety = new OneHotEncoder().setInputCol("variety_index").setOutputCol("variety_encoded")  
val wineVectorAssembler = new VectorAssembler().setInputCols(Array("country_encoded", "variety_encoded", "price", "points")).setOutputCol  
  ("features")  
val transformerPipeline = new Pipeline().setStages(Array(indexerCountry, encoderCountry, indexerVariety, encoderVariety, wineVectorAssembler))  
  
val fittedPipeline = transformerPipeline.fit(normalizedWine)  
val Array(trainingData, testData) = normalizedWine.randomSplit(Array(0.7, 0.3))  
val transformedTraining = fittedPipeline.transform(trainingData)  
transformedTraining.cache()  
val kmeans = new KMeans().setK(6).setSeed(1L)  
val kmModel = kmeans.fit(transformedTraining)  
kmModel.computeCost(transformedTraining)  
val transformedTest = fittedPipeline.transform(testData)  
transformedTest.cache()  
kmModel.computeCost(transformedTest)  
kmModel.transform(transformedTest)  
  
kmModel.transform(transformedTest).createOrReplaceTempView("kmeans")
```

SPARK JOBS FINISHED ▶ ✎ ☰ ⚙

FINISHED ▶ ✎ ☰ ⚙



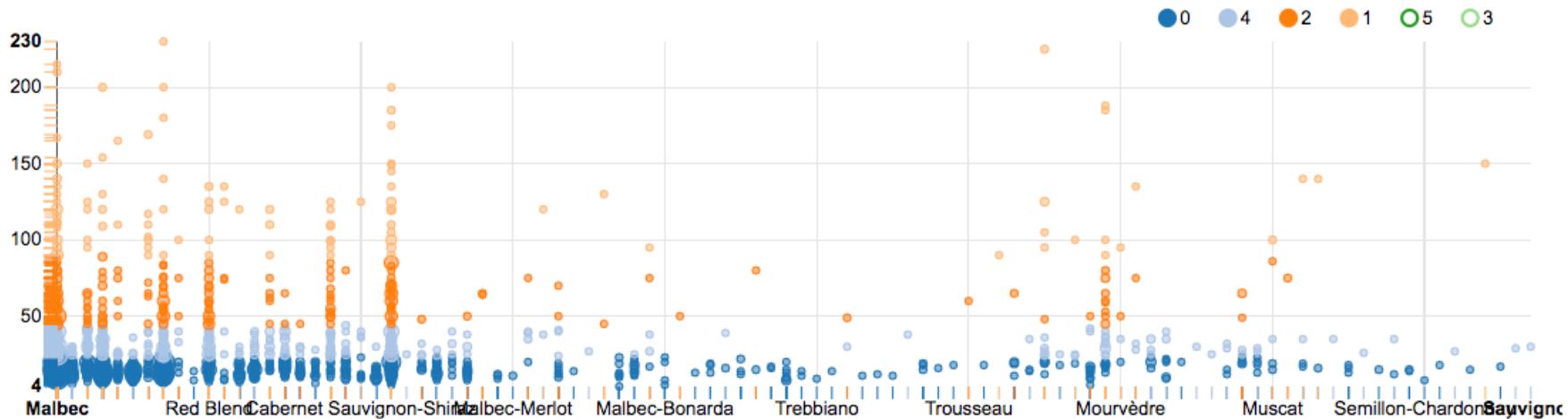
Exploratory Analytics : Unsupervised Learning

```
%sql  
select * from kmeanstrain
```

SPARK JOB FINISHED ▶ ✎ 📁 ⚙



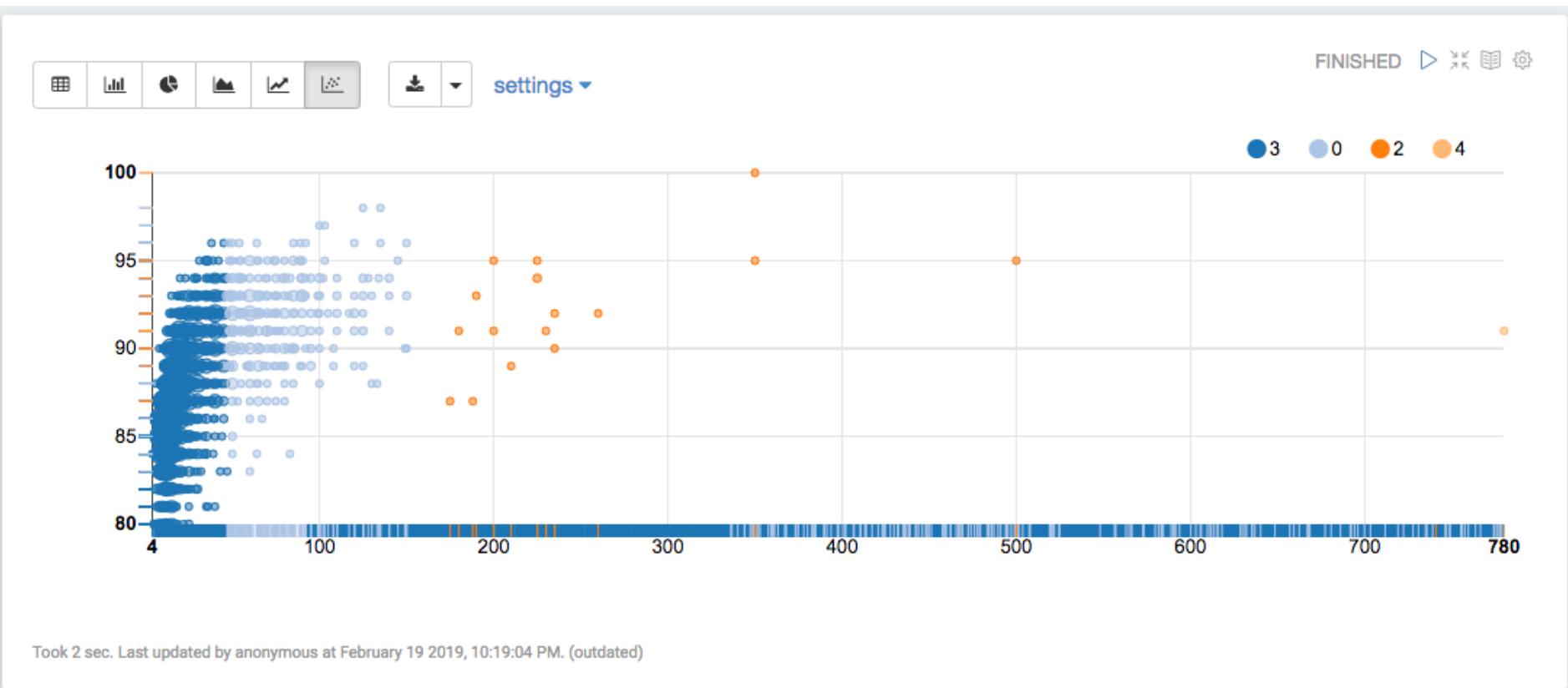
settings ▾



Took 1 sec. Last updated by anonymous at February 19 2019, 3:06:45 PM. (outdated)



Exploratory Analytics : Unsupervised Learning





Exploratory Analytics : Unsupervised Learning

Apriori Algorithm

1. Well know for the diapers + beer decision.
2. Shoppers who bought X also bought Y
3. Yelp Review Highlights
4. Or Auto Wine Reviews

```
INPUT:  $S$ , support where  $S = \text{dataset}$ , min_support = real  
OUTPUT: Set of Frequent Itemsets  
Require:  $S \neq \emptyset$ ,  $0 \leq \text{min\_support} \leq 1$   
1: procedure GETFREQUENTITEMSETS  
2:   freqSets[ ]  $\leftarrow$  null  
3:   for all Itemsets  $i$  in  $S$  do  
4:     if support  $\geq \text{min\_support}$  then  
5:       freqSets[ ]  $\leftarrow i$   
6:     end if  
7:   end for  
8: end procedure
```

AKA - Market Basket Analysis



Exploratory Analytics : Unsupervised Learning

```
def tastingNotes(df: DataFrame, minSupport: Double = 0.05, minConfidence: Double = 0.6, freq: Int = 400): String = {  
    // can mess around easily with different selection criteria by changing the minSupport and minConfidence coefficents  
    val fpg = new FPGrowth()  
        .setItemsCol("items")  
        .setMinSupport(minSupport)  
        .setMinConfidence(minConfidence)  
  
    val remover = new StopWordsRemover()  
        .setInputCol("items")  
        .setOutputCol("filteredItems")  
  
    // Cleaning up the Wine Descriptions  
    val descriptions = df  
        .select(col("description")).where(col("description").isNotNull)  
        .map { case Row(s:String) =>  
            s.replace(",","").replace(".","");
            .replace("wine", "").split(" ")
            .toSet.toSeq
        }  
        .toDF("items")  
  
    // remove StopWords  
    val filteredDescriptions = remover.transform(descriptions)  
    val stopWordsFiltered = filteredDescriptions.select("filteredItems").toDF("items")  
    val model = fpg.fit(stopWordsFiltered)  
    val freqItems = model.freqItemsets.sort(desc("freq"))  
    val notes = freqItems.select("items").where(col("freq")>freq)  
    val topWords = notes.flatMap { case Row(notes: Seq[String]) => notes }.groupBy("value").count().sort(desc("count"))  
    val tastingNotes = topWords.select("value").collect().map { case Row(s: String) => s }.toSeq.mkString(",")  
    tastingNotes  
}
```

FINISH



Exploratory Analytics : Unsupervised Learning

```
val redBlends = spark.sql("select * from winereviews where `variety` == 'Red Blend'")  
redBlends.cache  
redBlends.explain(true)  
val redBlendTastingNotes = tastingNotes(redBlends)
```

palate,tannins,aromas,cherry,black,blend,Cabernet,Sauvignon,flavors,Merlot,spice,finish,alongside,plum,fruit,offers,berry,red,pepper,Syrah,Drink,Sangiovese,ripe,opens,blackberry,acidity,notes,nose,dried,delivers,licorice,oak,Franç,Petit,dark,Verdot,tobacco,juicy,leather,firm,chocolate,vanilla,clove,raspberry,hint,white,note,Aromas,lead,currant,spicy,rich,tannic,touch,along,dry,earth,whiff,fresh,Malbec,Made,espresso,soft,smooth,herb,fruits,sweet,texture,wild,cassis,herbal,Petite,coffee,Sirah,well,savory,herbs,shows,bright,mocha



Warm Up

Next Up

1. Spark Basics (Quick Overview)
2. Exploratory Analytics With Spark SQL
3. Finding Structure with Unsupervised Learning
4. Workshop

Workshop

Part 1: Exploratory Data Analytics on Wine Reviews

<https://github.com/newfront/odsc-east2019-warmup>

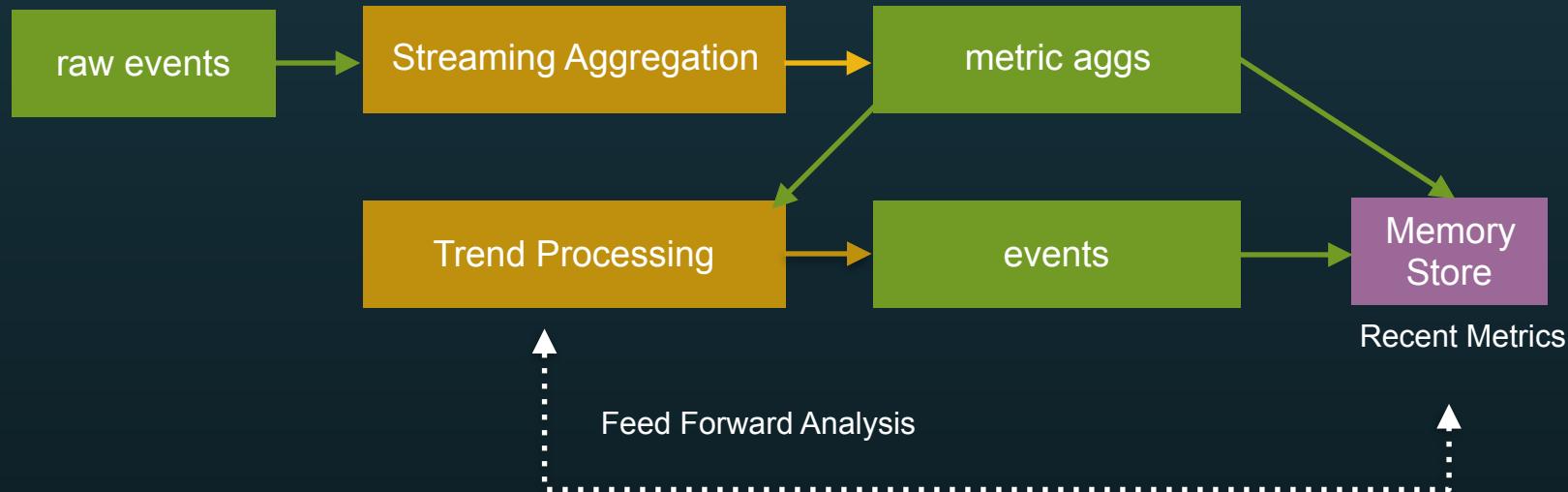
Part 2: Walk through and Play with Spark Structured Streaming

<https://github.com/newfront/odsc-west-streaming-trends/tree/master/part3/streaming-trend-discovery>

Workshop Highlights



Learn Best Practices for Structured Streaming Applications





Workshop Highlights

Use Statistics to auto-generate event tags

```
if (metric.value > p99(historic.metric)
```



```
if (metric.dimension.country == 'US' && p99[US{historic.metric}])
```



```
tag: high_{metric}_{dimensions}
```

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...
```



Workshop Highlights

Write an End to End Structured Streaming Application

```
object TrendDiscoveryApp {  
    val logger = LoggerFactory.getLogger(classOf[TrendDiscoveryApp])  
  
    def main(args: Array[String]): Unit = {  
        logger.info("application arguments: $args")  
        assert(args.length > 0, "No application config supplied to app args")  
        val config = AppConfig(args(0))  
  
        val sparkConf = new SparkConf()  
            .setAppName(config.appName)  
            .setJars(SparkContext.jarOfClass(classOf[TrendDiscoveryApp]).toList)  
            .setAll(config.sparkCoreConfig)  
  
        logger.info(s"sparkConfig: ${sparkConf.toDebugString}")  
  
        val sparkSession = SparkSession.builder()  
            .config(sparkConf)  
            .enableHiveSupport() // *need to build spark with hive jars  
            .getOrCreate()  
  
        // Add Listener Interface to gain more information about how Spark is running  
        sparkSession.sparkContext.addSparkListener(SparkApplicationListener(sparkSession))  
  
        logger.info(s"warehouse path: ${sparkSession.sharedState.warehousePath}")  
  
        // create instance of the monitored application  
        new TrendDiscoveryApp(config, sparkSession)  
            .monitoredRun()  
    }  
}
```

```
class TrendDiscoveryApp(override val config: AppConfig, override val spark: SparkSession)  
    extends RestartableStreamingApp[AppConfig] {  
    override val logger: Logger = TrendDiscoveryApp.logger  
  
    lazy val kafkaStreamingSource: DataStreamReader = {  
        val kafkaConfig = config.callEventsTopic  
        val reader = spark.readStream.format("kafka")  
        kafkaConfig.conf.foreach { entry =>  
            reader.option(entry._1, entry._2)  
        }  
        reader.option(kafkaConfig.subscriptionType, kafkaConfig.topic)  
        reader  
    }  
  
    override def run(): Unit = {  
        // creates a streaming query that will aggregate events from kafka  
        // and produce statistical histogram aggregations to be consumed downstream  
        val eventProcessor = EventAggregation(config)  
        eventProcessor.process(readKafkaStream())(spark)  
            .writeStream  
            .queryName("prediction.stream")  
            .outputMode(eventProcessor.outputMode)  
            .trigger(Trigger.ProcessingTime(Duration(config.triggerInterval)))  
            .format("kafka")  
            .option("topic", "odsc.predictions")  
            .options(Map(  
                "checkpointLocation" -> config.checkpointPath  
            ))  
            .start()  
    }  
  
    def readKafkaStream(): DataFrame = {  
        val stream = kafkaStreamingSource.load()  
        stream.printSchema()  
        stream  
    }  
}
```



Workshop Highlights

Write Unit Tests like a Boss

```
test("Should aggregate call events") {
    implicit val sqlContext: SQLContext = session.sqlContext
    import session.implicits._
    val appConfig = appConfigForTest()
    val scenario = TestHelper.loadScenario[CallEvent](s"$pathToTestScenarios/pdd_events.json")
    val scenarioIter = scenario.toIterator
    scenario.nonEmpty shouldBe true

    val trendDiscoveryApp = new TrendDiscoveryApp(appConfigForTest(), session)
    val kafkaData = MemoryStream[MockKafkaDataFrame]
    val processingTimeTrigger = Trigger.ProcessingTime(2.seconds)
    val eventAggregation = EventAggregation(appConfig)

    val processingStream = eventAggregation.process(kafkaData.toDF())(session)
        .writeStream
        .format("memory")
        .queryName("calleventags")
        .outputMode(eventAggregation.outputMode)
        .trigger(processingTimeTrigger)
        .start()

    // 22 events
    kafkaData.addData(scenarioIter.take(11).map(TestHelper.asMockKafkaData))
    processingStream.processAllAvailable()
    kafkaData.addData(scenarioIter.take(10).map(TestHelper.asMockKafkaData))
    processingStream.processAllAvailable()
    kafkaData.addData(scenarioIter.take(1).map(TestHelper.asMockKafkaDataFrame))
    processingStream.processAllAvailable()

    val df = session.sql("select * from calleventags")
    val res = session
        .sql("select avg(stats.p99) from calleventags")
        .collect().map { r => r.getAs[Double](0) }.head

    DiscoveryUtils.round(res) shouldEqual 7.56

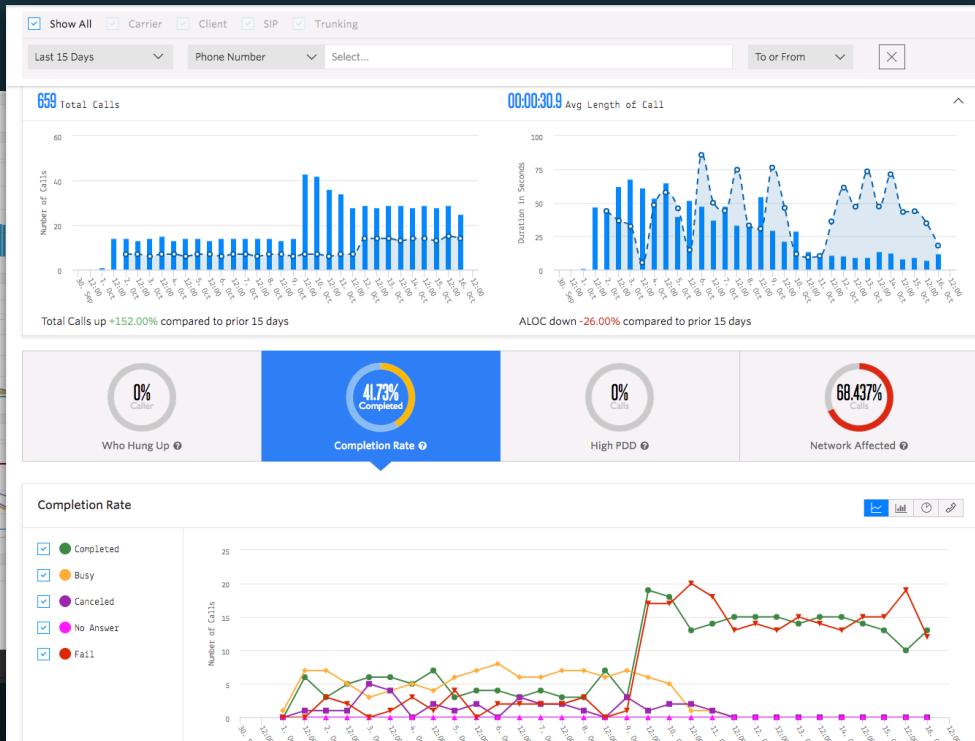
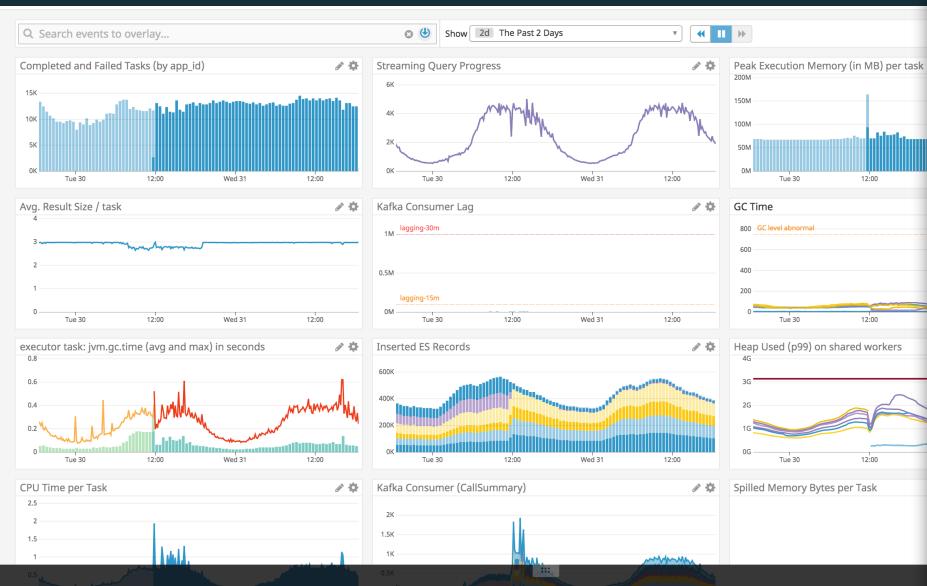
    processingStream.stop()
}
```

```
val kafkaData = MemoryStream[MockKafkaDataFrame]
val processingTimeTrigger = Trigger.ProcessingTime(2.seconds)
val eventAggregation = EventAggregation(appConfig)

val processingStream = eventAggregation.process(kafkaData.toDF()(session))
    .writeStream
    .format("memory")
    .queryName("calleventags")
    .outputMode(eventAggregation.outputMode)
    .trigger(processingTimeTrigger)
    .start()
```

Spark Application Monitoring

Streaming is hard...



@newfront | <http://bit.ly/odsceastrealtime>

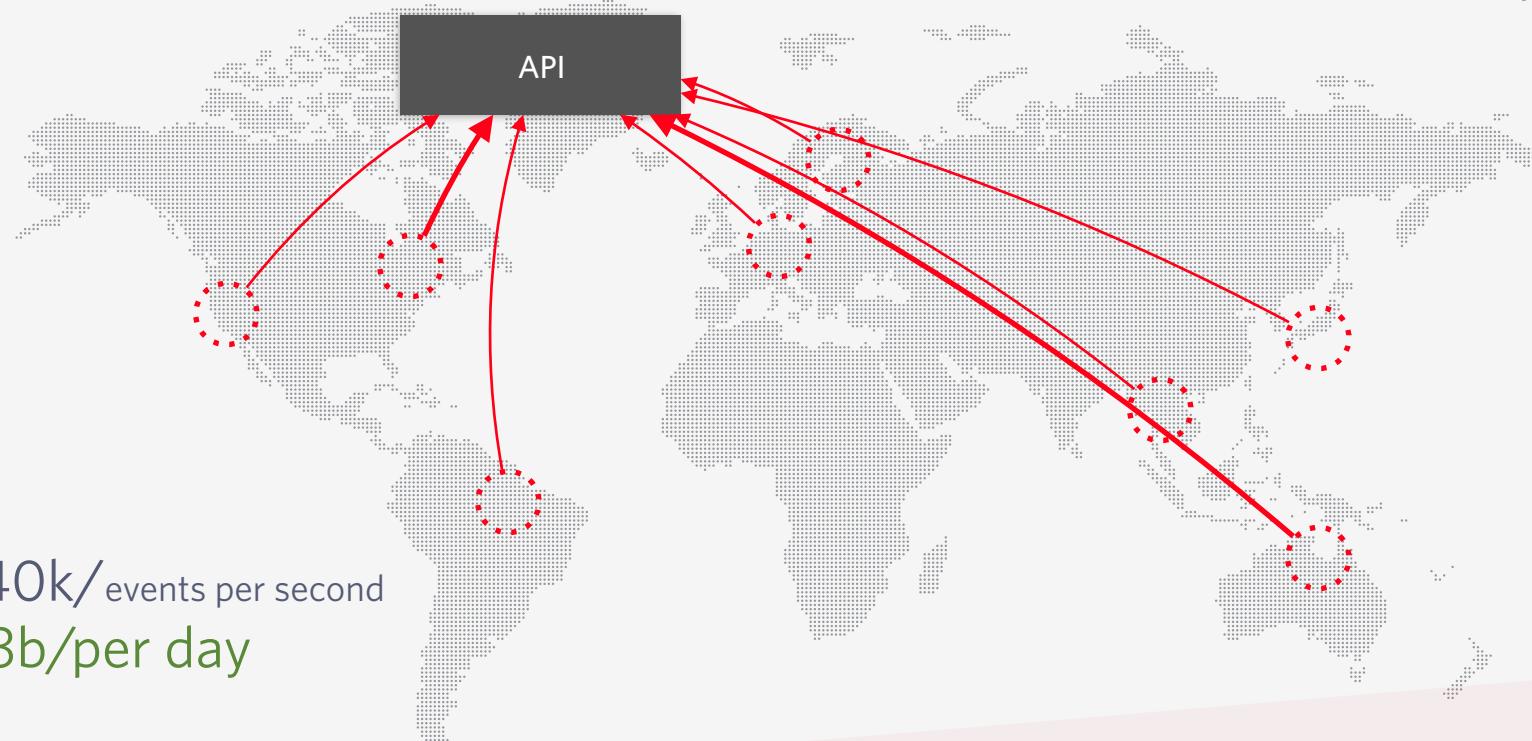
DISTRIBUTED FIRST ARCHITECTURE





DISTRIBUTED FIRST ARCHITECTURE

Why?

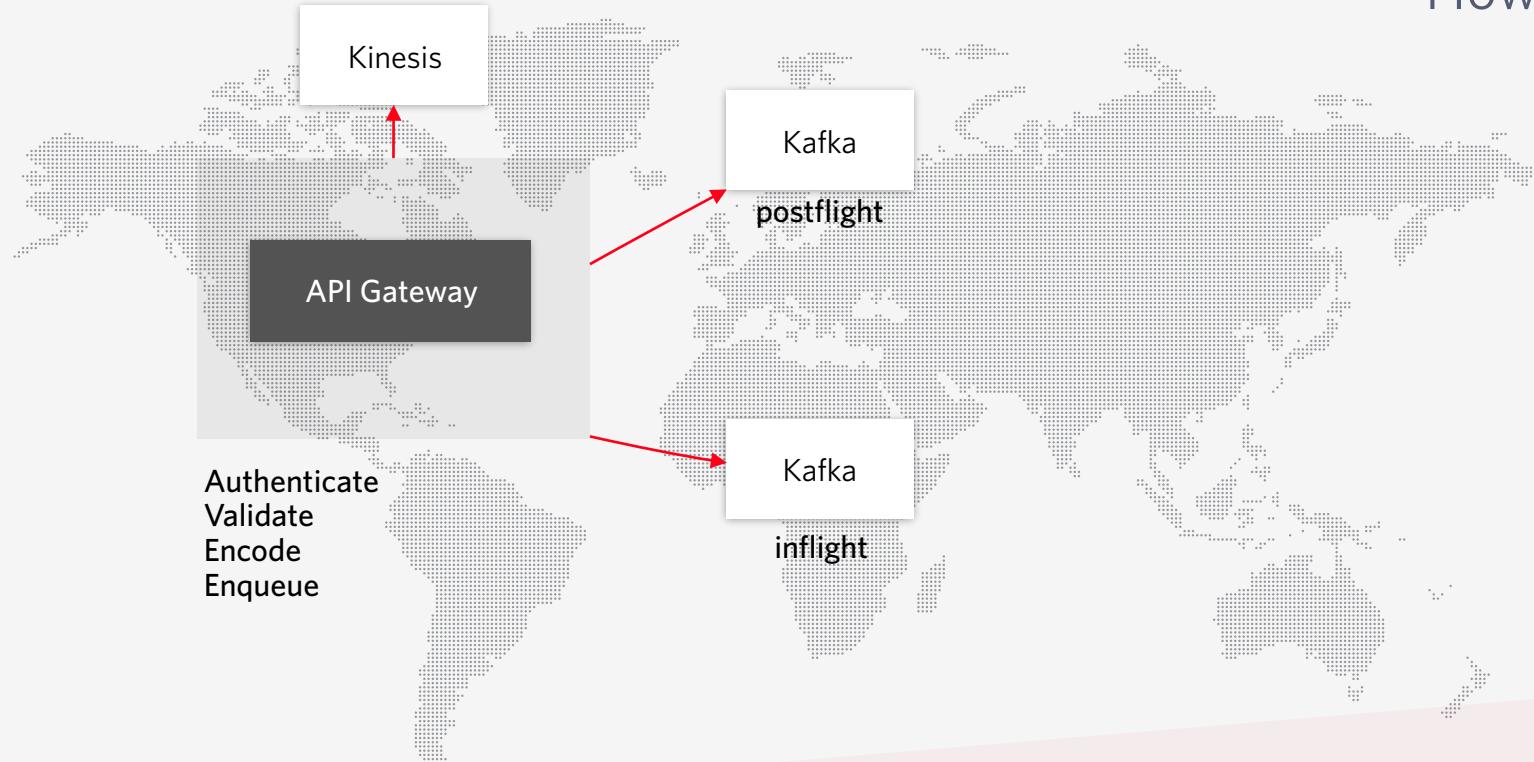


$\sim 40k$ /events per second
 $>3b$ /per day



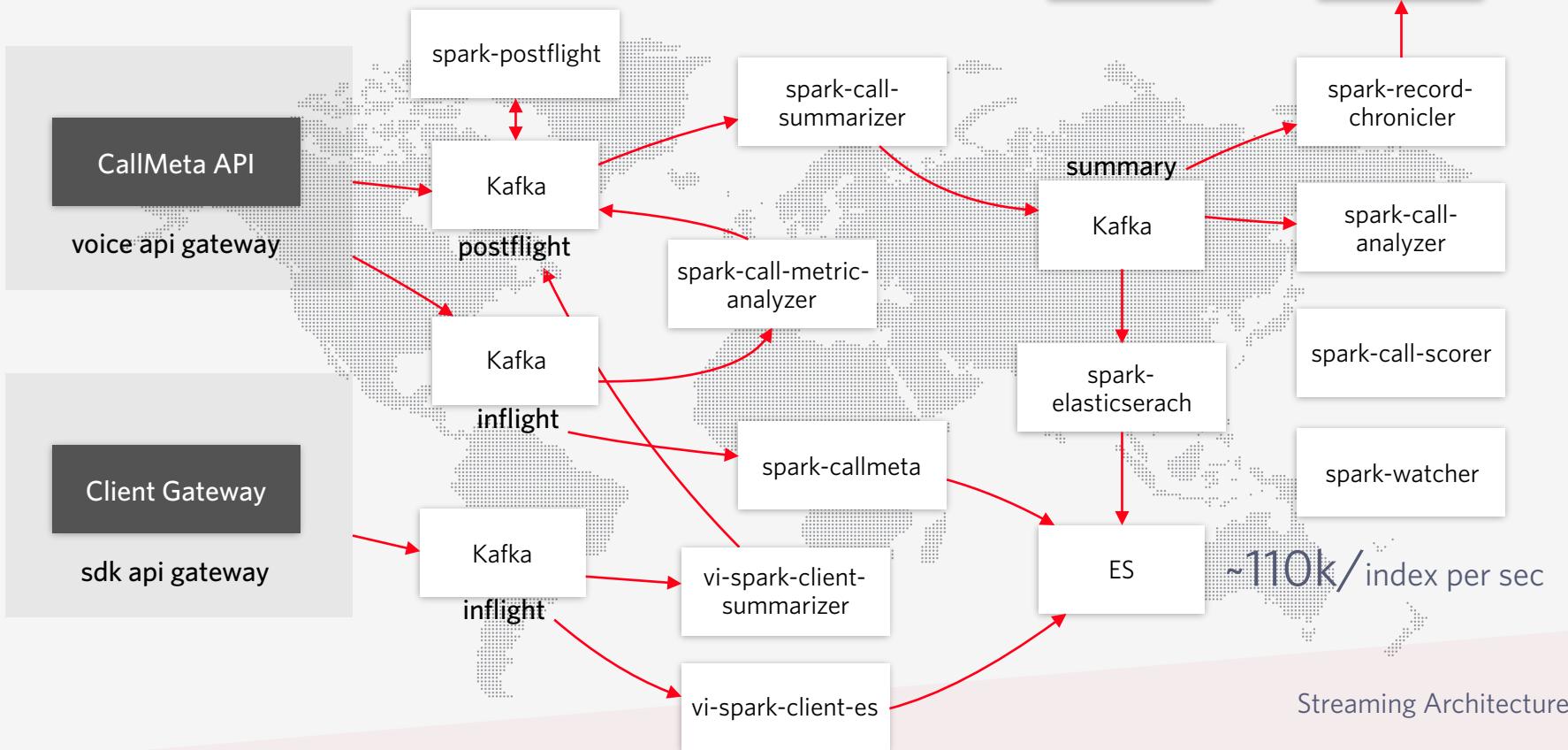
DISTRIBUTED FIRST ARCHITECTURE

How



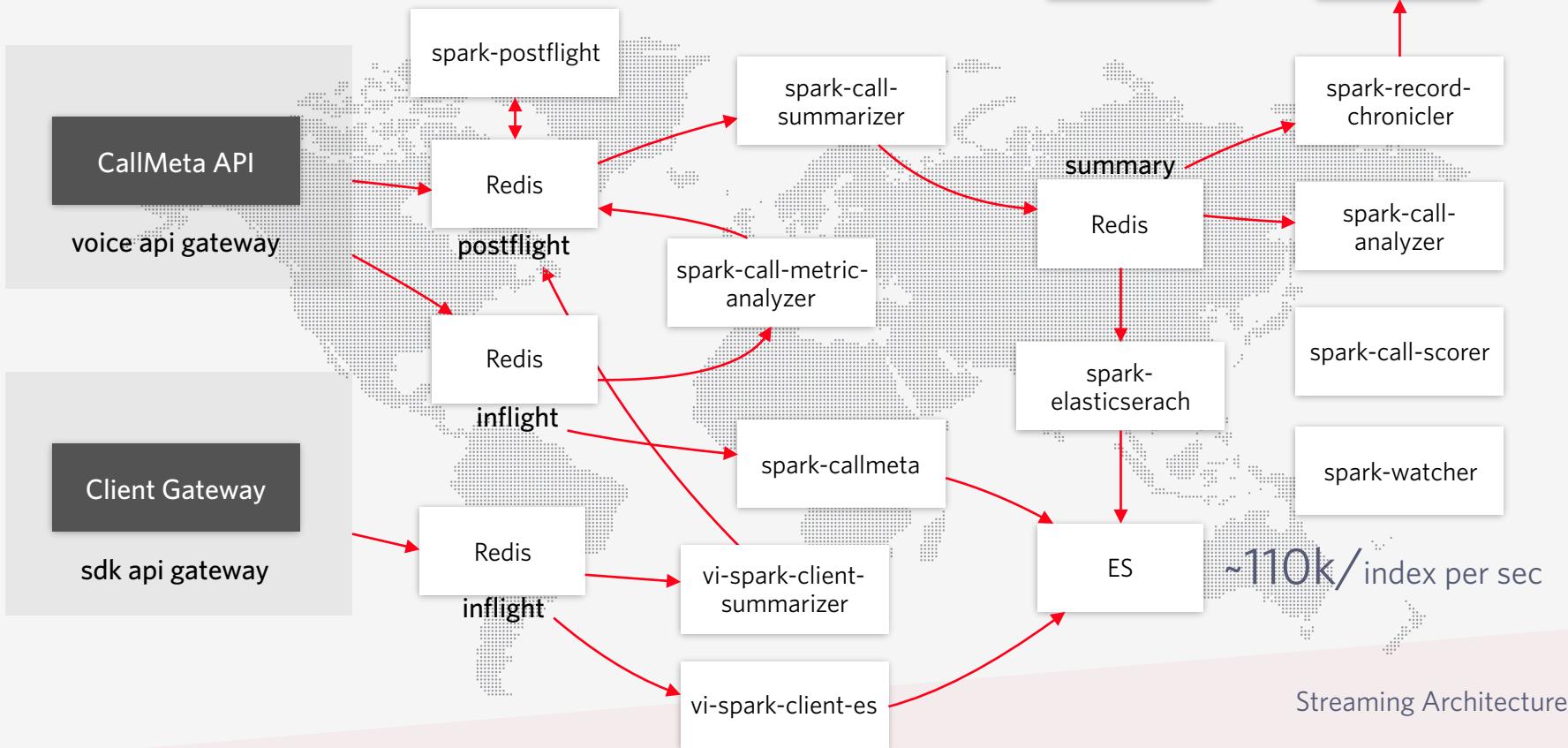


DISTRIBUTED FIRST ARCHITECTURE: KAFKA PIPES





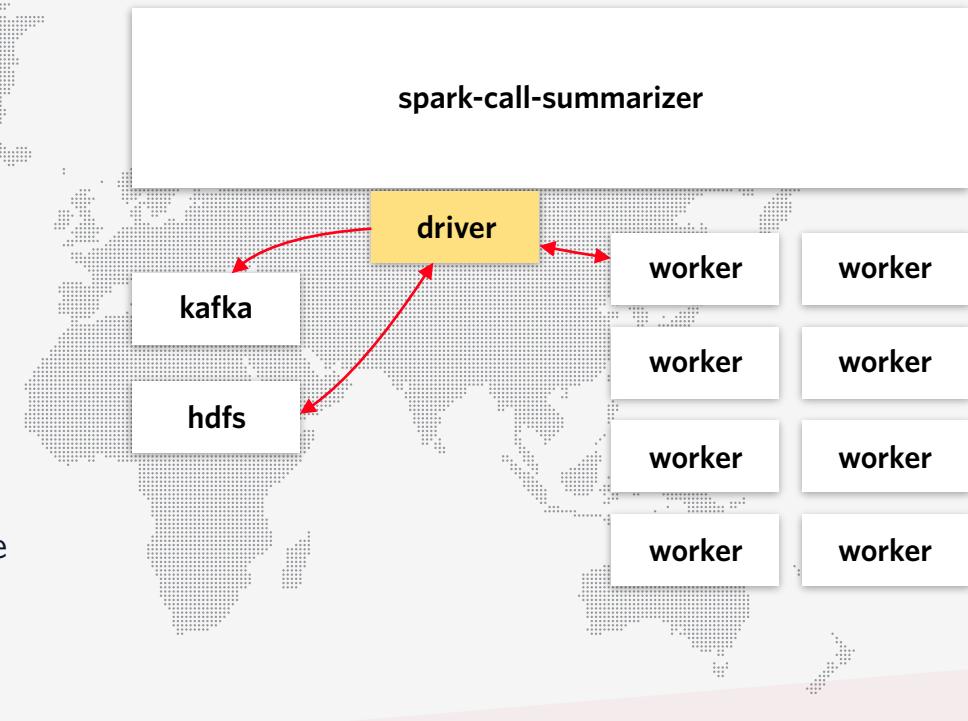
DISTRIBUTED FIRST ARCHITECTURE: REDIS STREAMS





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



Streaming Architecture



SPARK APP ARCHITECTURE

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

The screenshot shows a web-based application interface for managing multiple logical spark clusters. On the left, a sidebar menu for the app "voice-spark-driver" includes options like Overview, Access, Secrets, Settings, Monitoring, Service Configuration, **Deployments**, Cost, Autoscaling, and Lazarus. The Deployments section is currently selected.

The main content area displays five separate entries, each representing a logical spark cluster:

- MANIFEST: spark-call-analyzer-034-016** assigned to **FORMATION: cluster_240_eden**. Action buttons: ASSIGN MANIFEST, ASSIGN CONFIGURATION, COPY, DELETE.
- MANIFEST: spark-call-scorer-020-015** assigned to **FORMATION: cluster_240_eden_scorer**. Action buttons: ASSIGN MANIFEST, ASSIGN CONFIGURATION, COPY, DELETE.
- MANIFEST: spark-super-summarizer** assigned to **FORMATION: cluster_231_eden**. Action buttons: ASSIGN MANIFEST, ASSIGN CONFIGURATION, COPY, DELETE.
- MANIFEST: voice-spark-record-chronicler-016** assigned to **FORMATION: cluster_240_eden**. Action buttons: ASSIGN MANIFEST, ASSIGN CONFIGURATION, COPY, DELETE.
- MANIFEST: spark-call-watcher-007-002** assigned to **FORMATION: cluster_231_eden_watcher**. Action buttons: ASSIGN MANIFEST, ASSIGN CONFIGURATION, COPY, DELETE.

Streaming Architecture

Workshop

Part 1: Exploratory Data Analytics on Wine Reviews

<https://github.com/newfront/odsc-east2019-warmup>

Part 2: Walk through and Play with Spark Structured Streaming

<https://github.com/newfront/odsc-west-streaming-trends/tree/master/part3/streaming-trend-discovery>

@newfront | @twilio



Thanks to machine-learning algorithms,
the robot apocalypse was short-lived.

twitter: [@newfront](#)

blog: blog.twilio.com

THANK YOU

