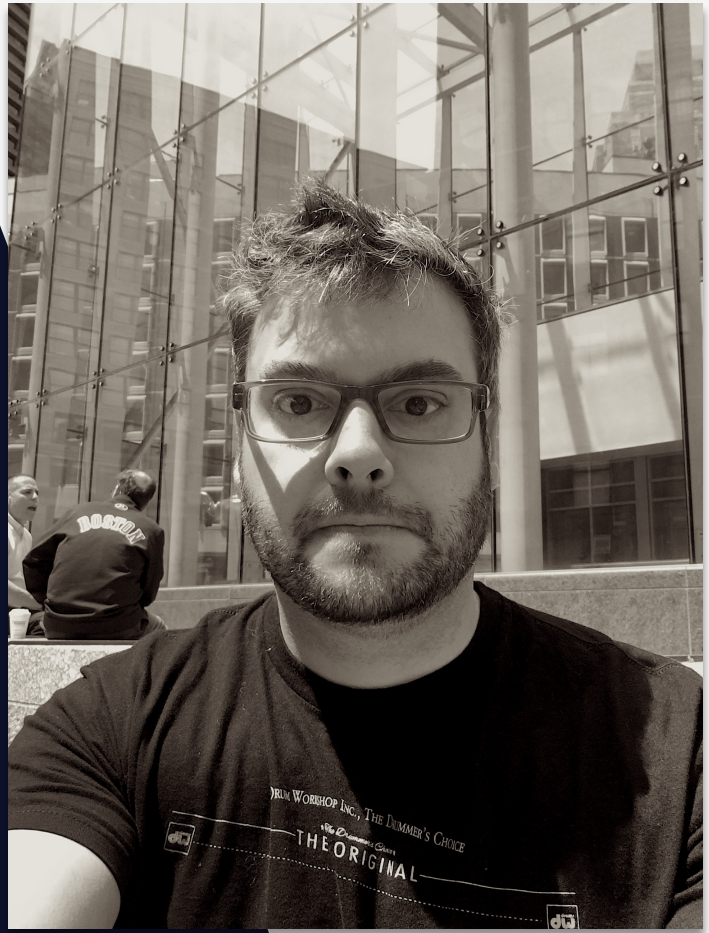
Beyond SQL

Building hybrid analytics & insights systems on top of Apache Spark

Scott Haines, Principal Software Engineer,
Voice Insights, Twilio

@newfront
@twilio

@newfrontcreative



TWILIO SIGNAL

10% OFF: ENGINEERING10

SIGNAL: CUSTOMER AND DEVELOPER CONFERENCE

Explore the intersection of technology, innovation, and communications.

AUGUST 6 & 7 MOSCONE WEST, SF CA

2 days of Twilio learning, talks, and networking.

@newfront @twilio

Laying out the concepts

SOME BACKGROUND

SQL / NoSQL / NewSQL

Modern Data Architecture

Zero to Hero: SparkSQL

Beyond SQL

@newfront @twilio

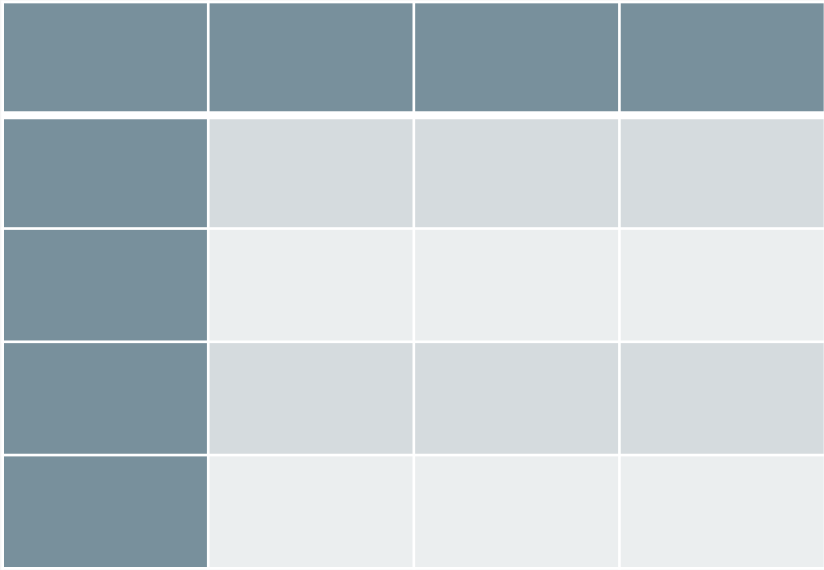
Definitions

- **ACID** (Atomicity, Consistency, Isolation, Durability)
- **Atomicity**: Transactions are guaranteed. Eg. All or Nothing
- **Consistency**: Data operations are consistent, rule based fashion
- **Isolation**: Separation of concerns affects visibility of data
- **Durability**: Data state safety. Fault Tolerance

@newfront @twilio

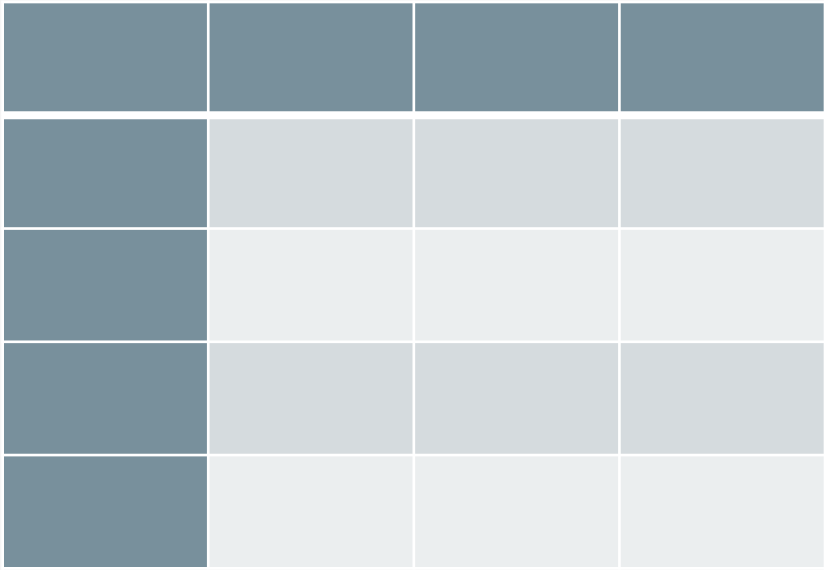
OG SQL

- **Flavors**: MySQL, SQL Server, Oracle, etc
- **Architecture**: Master, Slave
- **Scalability**: Share Everything
- **OLTP**: Online Transaction Processing



OG SQL

- **Structured**: Introduced structured table data via schemas
- **Relational**: Star Pattern: (foreign keys (fk), has and belongs to many, has relationships
- **Projectable**: `select fieldA, fieldB from table`
- **Filterable**: `select * from table where conditionA and conditionB and conditionN`



@newfront @twilio

OLAP SQL

online analytical processing

- **Flavors**: Apache **Druid**, Apache **Pinot***, Apache **Kylin**, and more
- **Aggregation**: Pre-Aggregation (read optimized), Just-in-time Aggregation (disk optimized, pay in memory)
- **Why**: Allows analytics and insight magic backed by SQL or SQL-like stores

Dashboard magic delivered

@newfront @twilio

No SQL

non-relational

- **Flavors**: Redis, Cassandra*, RocksDB, MongoDB
- **Architectures**: High Availability (HA), Mixed Master (MaMsSS, MaMaSS), Ring/Quorum, Share-Nothing, Eventually Consistent
- **Scalability**: Horizontal via Cluster Sharding

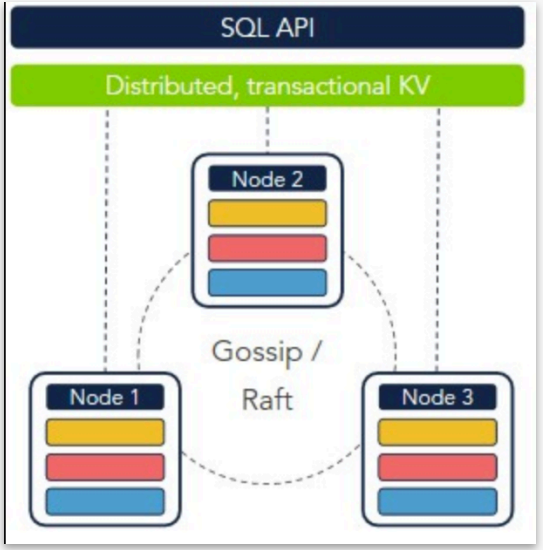
Blob Store	Data Structure Store	Document Store
JSON, binary, or anything	JSON, binary, or anything	JSON
*cassandra, rocks	*redis	*mongodb

@newfront @twilio

NewSQL

cloud optimized

- **Flavors**: MemSQL, CockroachDB, Google Spanner
- **Architectures**: HA, ACID*, Distributed, Share-Nothing, Cloud Optimized, All Nodes can serve requests or relay requests.
- **Scalability**: Horizontal, Globally Distributed, Low Latency
- **Supports**: Distributed SQL Queries. Low Latency



Nodes use message passing to resolve distributed queries with low latency

@newfront @twilio

Musings regarding Data Stores

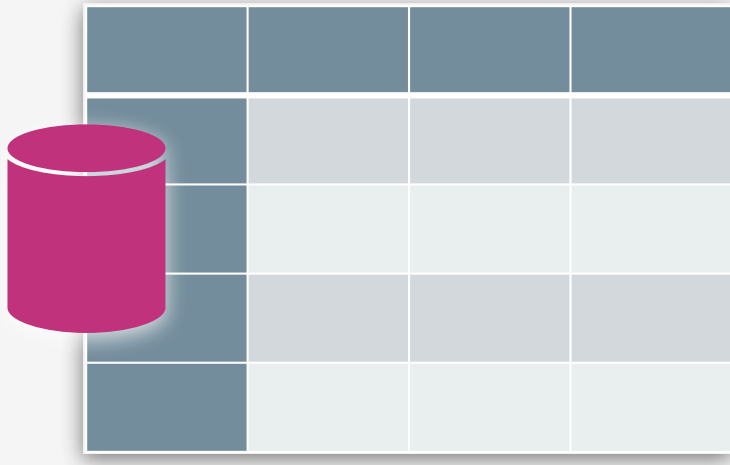
- **Data Warehouses**: OG Shared Data Store
- **Data Lakes**: Bottomless Pits for Good or for Bad

The Data Warehouse

Pattern One

- **How:** Shared Monolithic SQL Database
- **Why**: Organizations need shared access to be able to solve common Business Analytics needs
- **Pros**: Single Shared Database. Uses Grant Permissions to restrict access to Tables
- **Cons**: Single Shared Database. Scalability…

Kimball Group coined 'Warehouse'



One Silo to Rule Them all.
One Silo to bring them together…
Something something Bind them…

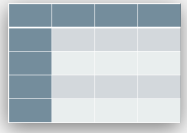
@newfront @twilio

The Data Lake

Pattern Two

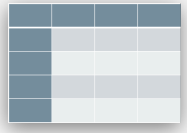
- **How:** Reliable File Store (s3, hdfs) and connector to hive, looker, etc
- **Why**: Organizations need to be able to store tons of data and analyze in big data fashion.
- **Pros**: Scales out almost indefinitely.
- **Cons**: Schema / San-Schema / Maybe Schema

Scales Out. Distributed Storage



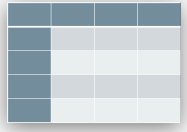
```
{
  'type': ['json', 'csv']
}
```

Semi-Structured



```
{
  'type': ['avro', 'protobuf']
}
```

Structured Binary



```
{
  'type': ['parquet', 'orc']
}
```

Columnar

@newfront @twilio

Musings about Structured Data

"Don't drown in your Data Lake"

- **JSON <= runtime headache**
- **Data Engineering Dos**: Data Contracts and Defined and Versioned Structs.

Structured Data

- JSON has Structure.
- But JSON isn't **strictly** Structured Data.

>95%

Of people love JSON*.

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

Structured Data

- JSON has **poor runtime guarantees** due to its flexible nature. **Optimize** for compile time guarantees.
- **Debugging** corrupt data in a large distributed system ruins hopes and dreams.

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

100%

Of people saddened by bad data*

Structured Data
Versioned & Compiled

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

```
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!

@newfront @twilio

Structured Data
columnar

- **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")
```

Spark Encoders Joke ->

100%

Also like not having to define Encoders

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?

5 min break & QA then

LAYERING CONCEPTS

SQL / NoSQL / NewSQL

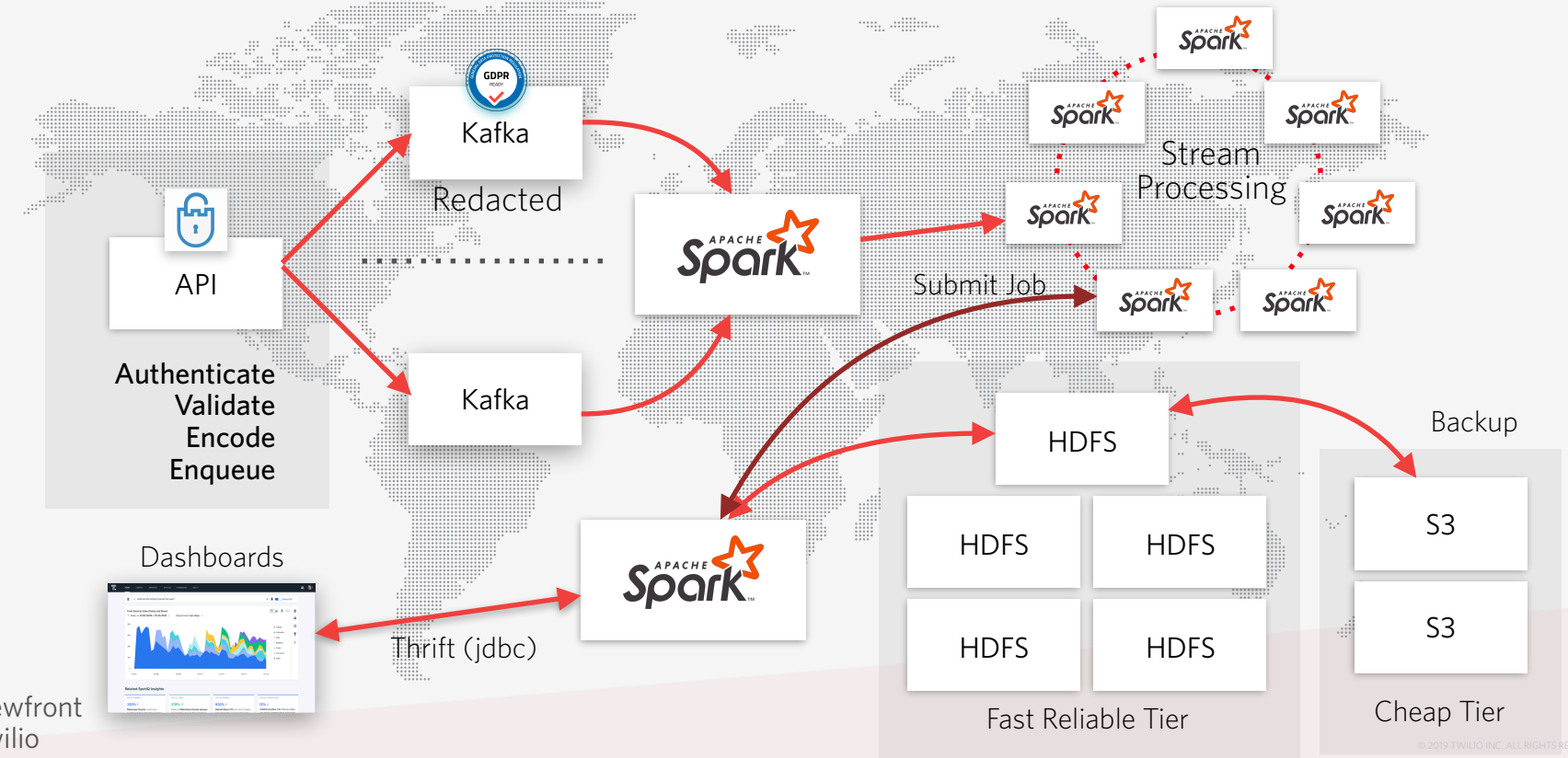
Modern Data Architecture

Zero to Hero: SparkSQL

Beyond SQL

@newfront
@twilio

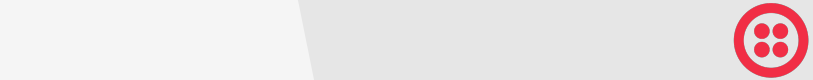
MODERN* DATA ARCHITECTURE
STREAMING MICROSERVICE ARCH



Zero to Hero: SparkSQL

"Let's look at Supporting OG-style SQL"

- **Load** all kinds of data
- **Transform** loaded data
- **Project** transformed data
- **Filtering** and more...



Zero to Hero
SparkSQL

- **The Spark Session**: Literally the GodFather of your application
- Controls and Delegates work
- Jobs, Stages and Tasks report back in and if they don't then we kill em off

```
object ChroniclerApp extends App {

  val log: Logger = LoggerFactory.getLogger(clas
  log.info(s"$args")

  val configPath = if (args.length > 0) args(0)
  val config = AppConfig.parse(configPath)

  val sparkConf = new SparkConf()
    .setAppName(config.sparkAppConfig.appName)
    .setJars(SparkContext.jarOfClass(classOf[Chr
    .setAll(config.sparkAppConfig.core)

  log.info(s"sparkConfig: ${sparkConf.toDebugStr

  val session = SparkSession.builder
    .config(sparkConf)
    .enableHiveSupport()
    .getOrCreate()

  session.sparkContext.addSparkListener(SparkApp

  Chronicler(config, session).monitoredRun()

}
```

Zero to Hero
SparkSQL

- **Loading** data for batch or streaming can be simple.

```
spark
  .read
  .option("inferSchema", "true")
  .json(path)
```

DataFrame Loaded

Zero to Hero
SparkSQL

- **Creating Projections** on your data is a breeze (once loaded)
- Creating **temporary views can simplify** using SparkSQL

```
spark
.read
.option("inferSchema", "true")
.json(path)
.createOrReplaceTempView("table")

val result = spark.sql(
  "select fieldA, fieldB from table
  where condition>90"
)
```

Zero to Hero
SparkSQL

- **Loading** data for **streaming** is just a we bit harder.
- **Must Provide** a Schema (StructType)
- Many **options to throttle** ingest on streaming data

```
def jsonSchema: StructType = {
  val updateToString = Set("start_time", "end_time")

  val baseSchema = summaryEncoder.schema
    .add(StructField("@timestamp", StringType, nullable = true
  val fields: Array[StructField] = baseSchema.map { field =>
    if (updateToString.contains(field.name)) {
      StructField(field.name, StringType, nullable = true)
    } else field
  }.toArray

  baseSchema.copy(fields)

}

lazy val fileRecoveryStream: DataStreamReader = fileStreamRead
def fileStreamReader(): DataStreamReader = spark.readStream
  .schema(jsonSchema)
  .option("maxFilesPerTrigger", recoveryMaxFilesPerTrigger)

override def run(): Unit = {
  assert(recoveryInputDir.nonEmpty, "spark.chronicler.recovery
  assert(recoveryOutputDir.nonEmpty, "spark.chronicler.recover
  assert(recoveryCheckpointPath.nonEmpty, "spark.chronicler.re

  val streamingQuery = RecoveryProcessor(config)
    .process(fileRecoveryStream.json(recoveryInputDir))(spark)
    .writeStream
```

Zero to Hero
SparkSQL

- **Loading** common data sources comes natively.
- **Data Sources**: Kafka, Flume, FileSystem, TCP
- **Formats**: CSV, JSON, AVRO, PARQUET
- **Bring your Own Format**: Extend ExpressionEncoder

```
val reader = spark.readStream.format("kafka")
```

+

```
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()
}
```

Zero to Hero
SparkSQL

- **ETL** aka Extract Transform Load
- **Create simplified projections** from complex nested data (parquet supports nesting, so does JSON)
- **Sanitize Data** - aka clean for GDPR & Data Governance

```
val withMetrics = flattenStruct(df, "metrics")
  .drop("metrics").drop("score_data").drop("dimension_hash").dro
  .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").dr
  .withColumn("dimensions_carrier_sid", base64encode(col("dimens
  .drop("dimensions_provider_sid").drop("dimensions_account_sid"

expandArray(withDimensions, col("insight_tags")).drop("insight_t
  .withColumn("ts", functions.from_utc_timestamp(col("iso_timest
  .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")))
```

Utility Methods -> http://bit.ly/spark-utils

Zero to Hero
SparkSQL

- Parquet + File System Push Down
- Lazy fetches data to resolve a given query.

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

scoresData: org.apache.spark.sql.DataFrame = [timestamp: bigint, score: flo
```

Took 7 sec. Last updated by anonymous at June 26 2019, 3:54:04 PM.

```
%spark
scoresData.inputFiles

res3: Array[String] = Array(hdfs://localhost:9000/user/ds/odsc/east/realish
rt-00000-0dbb1b79-64c4-456e-8650-65752b53844f.c000.snappy.parquet, hdfs://l
a/callscores/year=2019/month=1/day=25/part-00000-e895e0fb-e067-45d4-854b-b4
lhost:9000/user/ds/odsc/east/realish/eda/callscores/year=2019/month=1/day=2
eac29e.c000.snappy.parquet, hdfs://localhost:9000/user/ds/odsc/east/realish
rt-00000-30fea43a-140c-4f3c-be88-cd8a386add22.c000.snappy.parquet, hdfs://l
a/callscores/year=2019/month=1/day=30/part-00000-4fde69c3-39ee-4fe3-a032-98
```

Took 0 sec. Last updated by anonymous at June 26 2019, 4:00:28 PM.

```
scoresData.createOrReplaceTempView("scores")
```

5 min break & QA then

OLAP ANALYTICS

SQL / NoSQL / NewSQL

Modern Data Architecture

Zero to Hero: SparkSQL

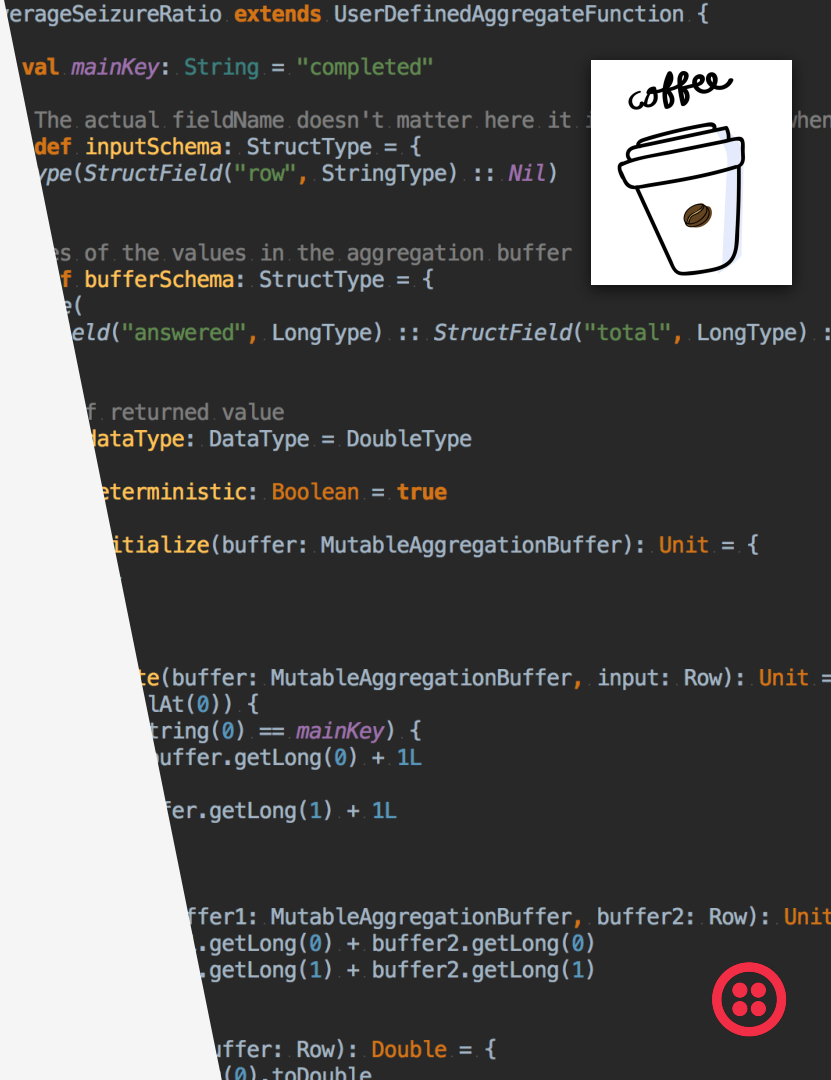
Beyond SQL

@newfront
@twilio

Zero to Hero: Analytics

"Supporting OLAP with SparkSQL"

- Aggregation Functions
- Windowing and Watermarking
- Extending Aggregations (MutableAggregationBuffers)



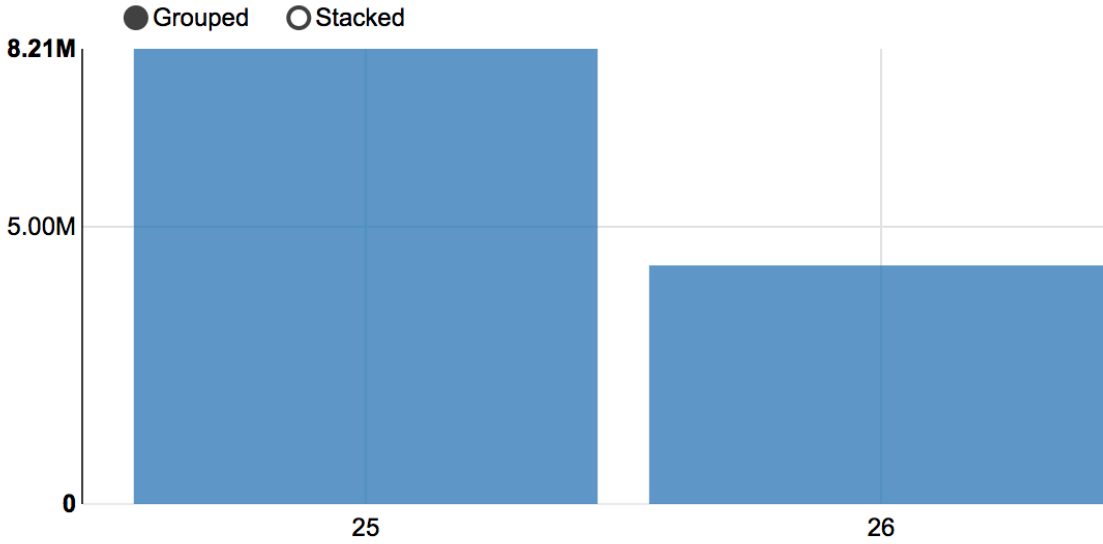
Zero to Hero

SparkSQL: Analytics

- Batch or Streaming queries are treated more or less the same*
- Use the DSL or query interpreter

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

settings



Took 11 sec. Last updated by anonymous at June 26 2019, 3:55:26 PM. (outdated)

Zero to Hero
SparkSQL: Analytics

- Hate SQL. Write SQL as a DAG!

```
%spark
val aggs = scoresData
    .select("account_sid","dimensions_country_code","score
    .filter(not(col("dimensions_country_code") === ""))
    .groupBy("account_sid","dimensions_country_code")
    .agg(
        count("*") as "total",
        min("score") as "min_score",
        avg("score") as "avg_score",
        max("score") as "max_score"
    )
    .sort(desc("total"))

aggs.createOrReplaceTempView("score_aggs")
```

```
aggs: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row
... 4 more fields]
```

Took 2 sec. Last updated by anonymous at June 26 2019, 4:08:13 PM.

Zero to Hero

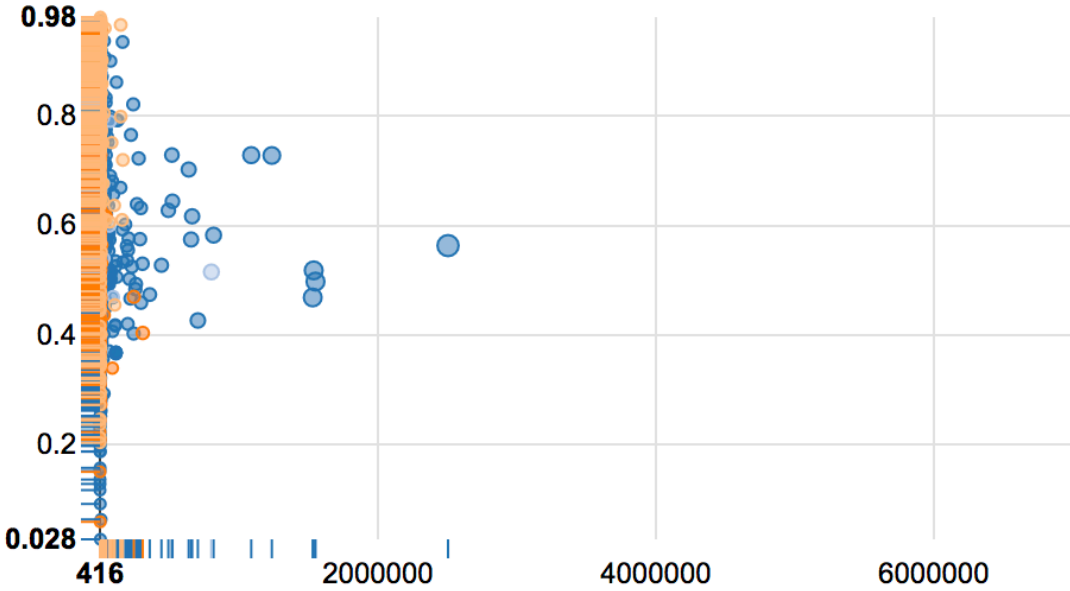
SparkSQL: Analytics

- Not Strictly SQL.
- Not Strictly MapReduce.
- Use what works.

```
%sql
select
account_sid, dimensions_country_code, total, min_score, avg_score,
from score_aggs
where dimensions_country_code in ("US", "MX", "GB", "CA")
```

SPARK JOB

settings



Zero to Hero
SparkSQL: Analytics

- Aggregate and groupBy with ease.

```
%spark
val carrierEdgebyCountry = scoresData
  .filter($"dimensions_edge_type" === "carrier_edge")
  .filter(not($"dimensions_country_code" === ""))
  .groupBy("dimensions_country_code","dayofyear")
  .agg(
    count("*") as "total",
    avg($"score") as "mean_score",
    sum("tag_high_packet_loss") as "pktloss_sum",
    sum("tag_silent") as "silent_sum",
    sum("tag_high_pdd") as "high_pdd_sum",
    sum("tag_high_jitter") as "high_jitter_sum",
    sum("tag_high_latency") as "high_latency_sum",
    avg($"metrics_connect_duration") as "avg_connect_duration"
  )

carrierEdgebyCountry.createOrReplaceTempView("carrier_by_countr
```

```
carrierEdgebyCountry: org.apache.spark.sql.DataFrame = [dimensio
dayofyear: int ... 8 more fields]
```

Zero to Hero

SparkSQL: Analytics

- Extend SparkSQL Aggregations with the **MutableAggregationBuffer**
- Chain Complex Queries into a Simple Flow (Spark will do the heavy lifting -> LogicalPlan)

```
val report = reporter.generate()
  .withColumn("is200",
    when(col("last_sip_response_num") equalTo(200), 1) otherwise(0))
  .groupBy("parent_account_sid",
  .agg(
    count("call_sid") as "total",
    sum("is200") as "normal",
    sum("connect_duration") as "t
    bround(asr(col("call_state"))
    bround(aloc(col("connect_dura
    insight_tags(col("insight_tag
  )
  .withColumn("connected_minutes"
    col("total_connected_seconds"
    , 2)
  )
  .withColumn("normal_sessions", bround(lit(100.0) * (col("normal") / col
  .drop("normal")
  .na.fill("", Seq("parent_account_sid"))
  .withColumn("reportId", lit(reportId))
```

```
/*
 Answer-seizure ratio
 ASR = 100 * ( answered / total ) - eg. compl
 */
def asr: AverageSeizureRatio.type = {
  AverageSeizureRatio
}

/*
 Average Length of Call
 sum(connect_duration) / N
 */
def aloc: AverageLengthOfCall.type = {
  AverageLengthOfCall
}
```

Zero to Hero

SparkSQL: Analytics

- Looking at creating an Distributed Aggregation - aka the Monoid

```scala
object AverageSeizureRatio extends UserDefinedAggregateFunction {

  private val mainKey: String = "completed"

  // note: The actual fieldName doesn't matter here it is the intent
  // - when you pass in (asr(col("call_state")) the schema is struct of string type
  override def inputSchema: StructType = {
    StructType(StructField("row", StringType) :: Nil)
  }

  // DataTypes of the values in the aggregation buffer
  override def bufferSchema: StructType = {
    StructType(
      StructField("answered", LongType) :: StructField("total", LongType) :: Nil)
  }

  // DataType of returned value
  override def dataType: DataType = DoubleType

  override def deterministic: Boolean = true

  override def initialize(buffer: MutableAggregationBuffer): Unit = {
    buffer(0) = 0L
    buffer(1) = 0L
  }

  override def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
    if (!input.isNullAt(0)) {
      if (input.getString(0) == mainKey) {
        buffer(0) = buffer.getLong(0) + 1L
      }
      buffer(1) = buffer.getLong(1) + 1L
    }
  }

  override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
    buffer1(0) = buffer1.getLong(0) + buffer2.getLong(0)
    buffer1(1) = buffer1.getLong(1) + buffer2.getLong(1)
  }
```

Zero to Hero

SparkSQL: Analytics

- Looking at creating an Distributed Aggregation - aka the Monoid

```scala
object AverageSeizureRatio extends UserDefinedAggregateFunction {

  private val mainKey: String = "completed"

  // note: The actual fieldName doesn't matter here it is the intent
  // - when you pass in (asr(col("call_state")) the schema is struct of string type
  override def inputSchema: StructType = {
    StructType(StructField("row", StringType) :: Nil)
  }

  // DataTypes of the values in the aggregation buffer
  override def bufferSchema: StructType = {
    StructType(
      StructField("answered", LongType) :: StructField("total", LongType) :: Nil)
  }

  // DataType of returned value
  override def dataType: DataType = DoubleType

  override def deterministic: Boolean = true

  override def initialize(buffer: MutableAggregationBuffer): Unit = {
    buffer(0) = 0L
    buffer(1) = 0L
  }

  override def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
    if (!input.isNullAt(0)) {
      if (input.getString(0) == mainKey) {
        buffer(0) = buffer.getLong(0) + 1L
      }
      buffer(1) = buffer.getLong(1) + 1L
    }
  }

  override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
    buffer1(0) = buffer1.getLong(0) + buffer2.getLong(0)
    buffer1(1) = buffer1.getLong(1) + buffer2.getLong(1)
  }
```

5 min break & QA then

INSIGHTS

SQL / NoSQL / NewSQL

Modern Data Architecture

Zero to Hero: SparkSQL

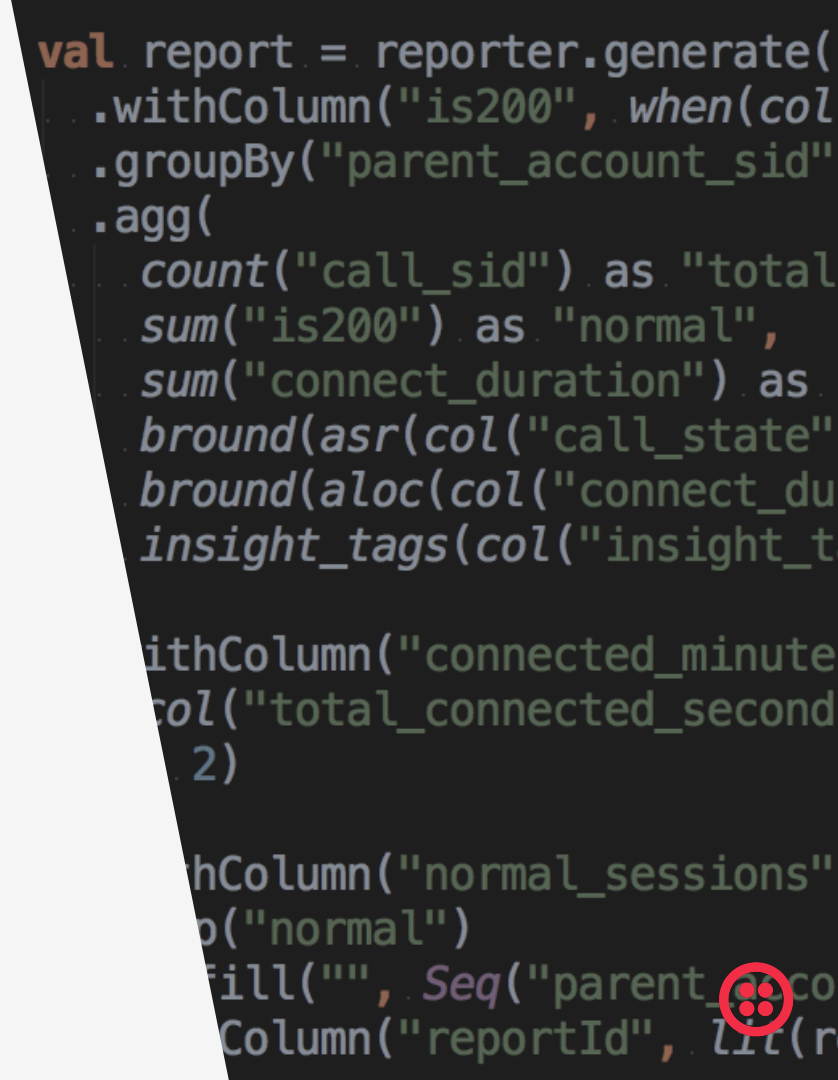
Beyond SQL

@newfront
@twilio

Zero to Hero: Insights

"Insights are the Why behind your Analytics"

- Find patterns in your data.
- Deliver value to customers



Beyond SQL
Insights

- **Percentiles** help provide a sense of what has been*
- **Explain** why things changed by triggering additional jobs - ML / Reports / etc
- **Triangulation**: Find a quorum in your data to answer a question

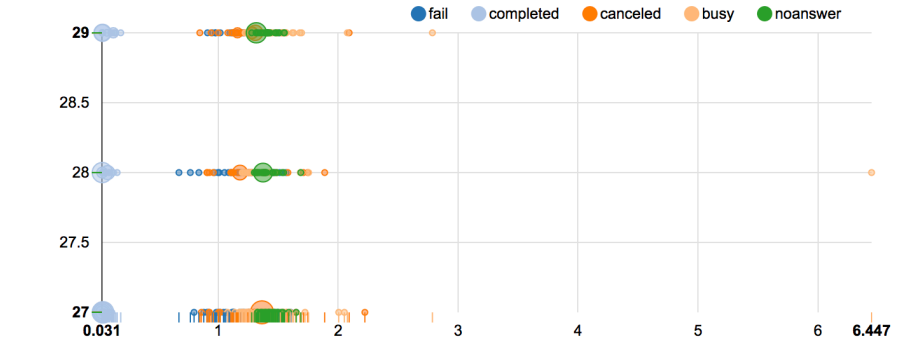
```
%spark
DataFrameUtils.percentiles(
    projectionOfCallStates
        .where(col("day").equalTo(27)
            .and(col("dimensions_call_state").equalTo("noanswer"))
        ),
        col("metrics_pdd"), Array(0.25, 0.5, 0.75, 0.9), 0.15)(spark).show

+----------+------------------+
|percentile|             value|
+----------+------------------+
|       p25| 1.0939
|       p50|3.07699
|       p75| 6.5069
|       p90|   2080
+----------+------------------+
```

```
%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 pdd_call_states where day in
```

SPARK JOBS FINISHED

settings



Took 23 sec. Last updated by anonymous at June 26 2019, 5:01:32 PM.

Beyond SQL
BYO-Logic

- **Window**: [start-end time] for a grouped operation
- **Watermark**: [throw away late arriving data]
- Use Spark to coordinate, you manage data in StateStore

```
// 1. Convert from the kafka data frame to an instance of our Metric
val groupedMetrics = df.mapPartitions(
  _.flatMap { kd =>
    bytesToMetricRow(kd.getAs[Array[Byte]]("value"), windowInterval)
  }
)(metricEncoder)
// 1b. apply watermark to ditch updates to said record state and allow for e
// 2. groupBy binary key
// 3. aggregate the value of the metric
// 4. output the final aggregations
groupedMetrics
  .withWatermark("timestamp", s"${config.watermarkIntervalMinutes} minutes")
  .groupByKey(_.metricAggregation)
  .flatMapGroupsWithState[Array[Byte], MetricAggregation](
    outputMode,
    GroupStateTimeout.EventTimeTimeout()
  )(metricAggregator)
```

FROM ZERO TO DATA HERO: HOW TO EFFECTIVELY USE VOICE INSIGHTS DATA TO SAVE THE DAY

How's it going? Learn from the experts on the Voice Insights team just how to answer common Business Intelligence questions over the course of their workshop. We will detail how organizations can start using the Voice Insights APIs in order to start answering their core Business Intelligence questions (in a streaming fashion) while leveraging expert insights from the team behind the data.

Tuesday, Aug 6
4:55 PM to 5:25 PM

- Intermediate
- Talk
- Best Practices
- Product Innovation
- Voice
- Developer / Engineer
- Product Management
- Solutions or Systems Architect
- Systems Administrator

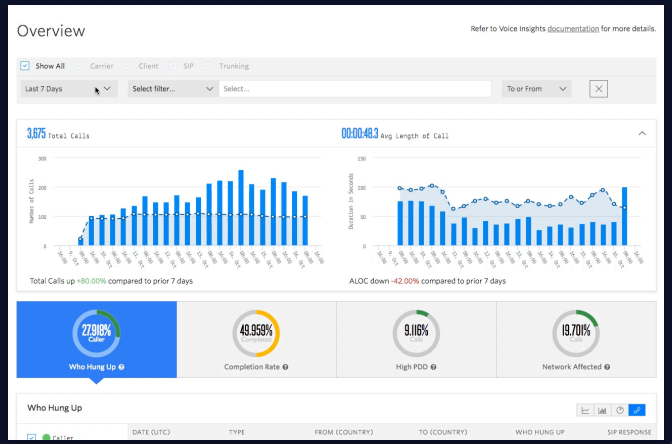
What I Do
Voice Insights

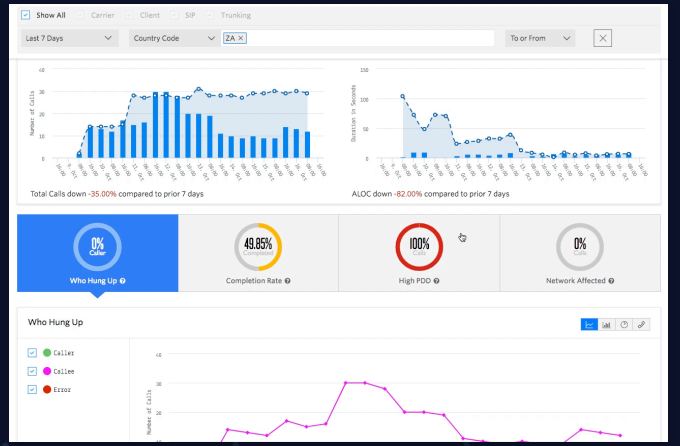
Voice Analytics Platform

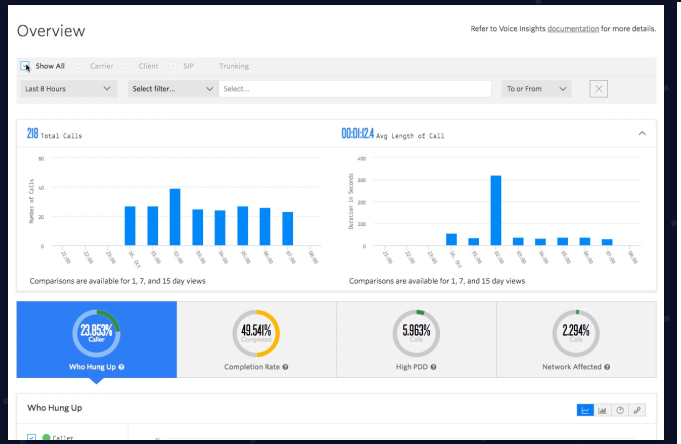
Millions of complex time series aggregations daily across billions of raw events

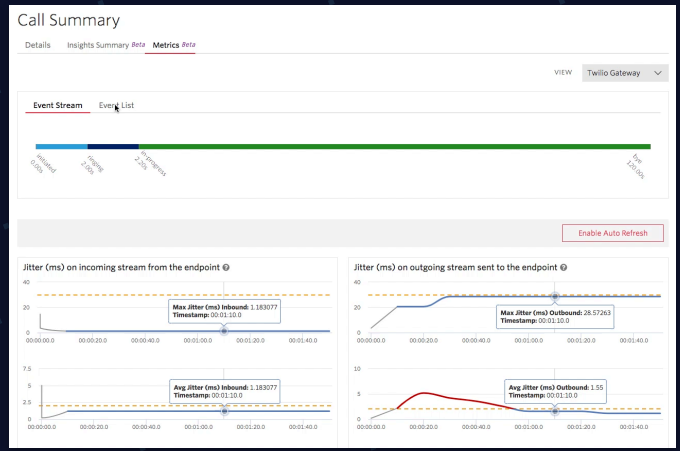
Supports internal and external customers.

*Near Real-Time









twitter: @newfront
blog: blog.twilio.com

THANK YOU