AUTOMATING TREND DISCOVERY

On Streaming Datasets with Apache Spark

#streamingdiscovery | @newfront | @odsc

Introductions

Scott Haines

[@newfront](https://twitter.com/newfront)

newfrontcreative@gmail.com

https://github.com/newfront/odsc-west-streaming-trends

Workshop Goals

Data Architecture / Fall back in love with your data

Workshop Goals

Use Spark to Clean and Explore Data

Workshop Goals

Learn how to harness Trend Discovery / Why unsupervised matters

Walk through a real application

Workshop Goals

Understand how to test Spark Applications and how this makes shipping to prod great!

Workshop Goals

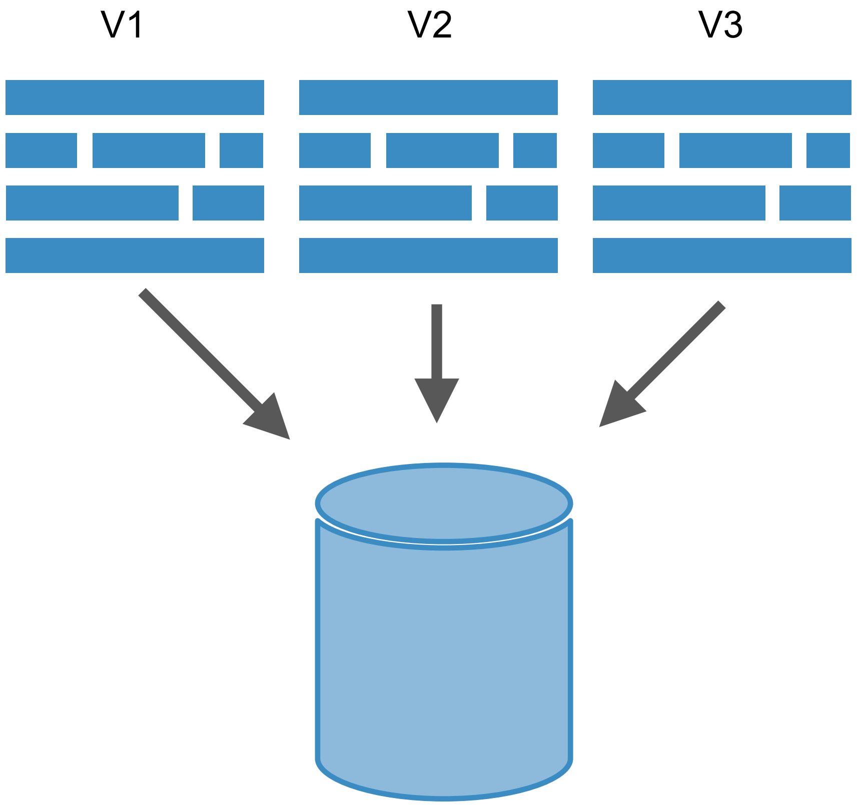
Simplifying Monitoring

PART ONE: DATA ARCHITECTURE

What is Data Architecture?

Data Architecture

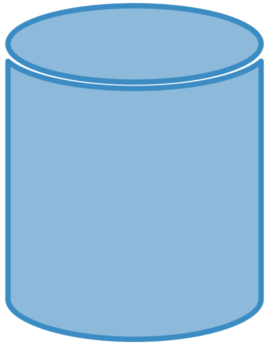
Data Lake Style



Makes Sense in Theory

Data Architecture

Data Lake Style



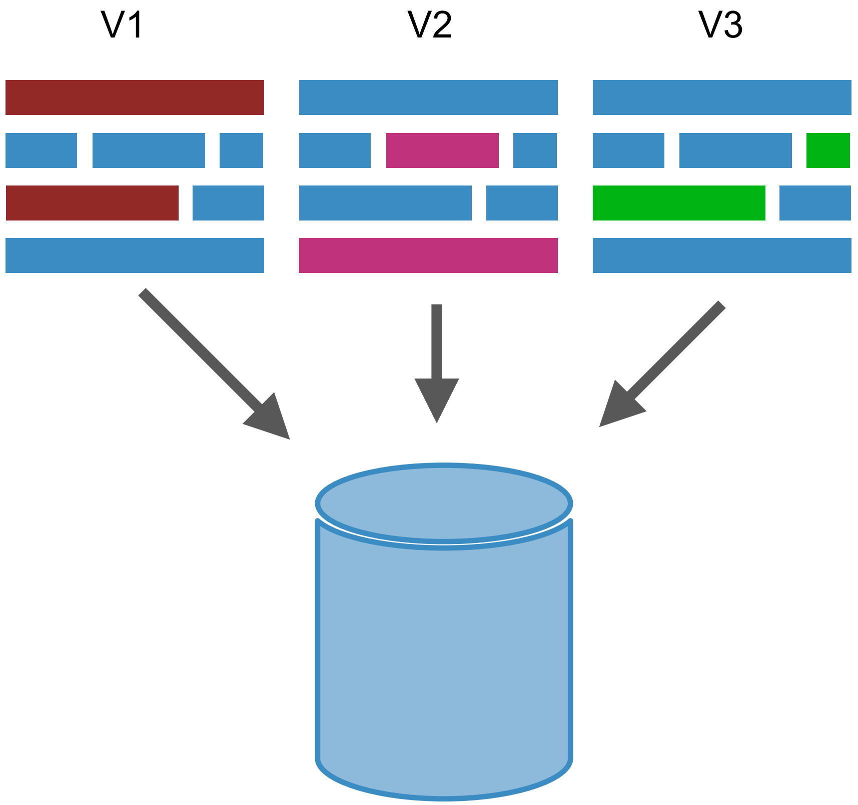
select * from table where **a IS NOT NULL** and **b IS NOT NULL**...

Extracting the data is a
different story all together

Data Architecture

Data Lake Style

1. Data is Volatile.
2. Field Types can change.
3. Older Data can becomes broken...



In Reality...

PART ONE: DATA ARCHITECTURE

Establish a Data Contract

Data Architecture

Data Contract

1. Define Data Rules
2. Define Types
3. Use Interoperable Data Formats
4. Validate Data Completeness

```
message UserEvent {
  required uint32 schema_version = 1;
  required bool gdpr_redacted      = 2;
  required string  user_id         = 3;
  required string  uuid            = 4;
  required uint64 event_ts         = 5;
  uint64 logged_event_ts           = 6;
  required UserEventType event     = 7;
  required UserEventData data      = 8;
}
```

Know what to expect

PART ONE: DATA ARCHITECTURE PIPELINE

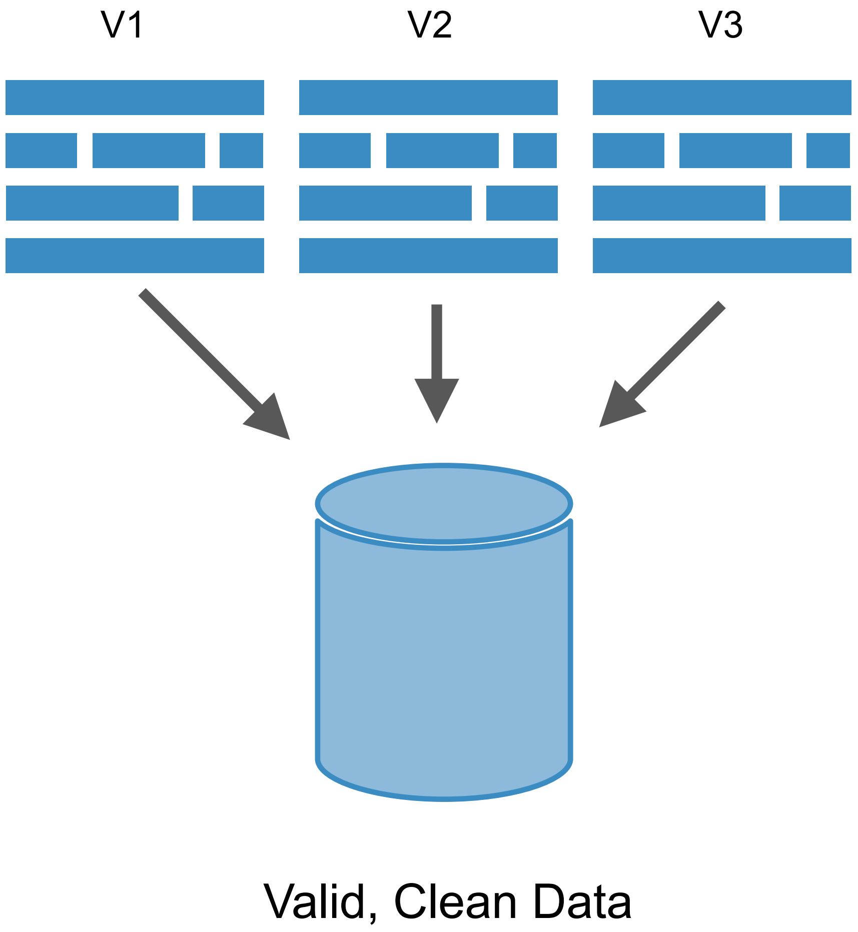
/api/v1/user/events

/api/v2/user/events

Data Architecture

Structured Data Store

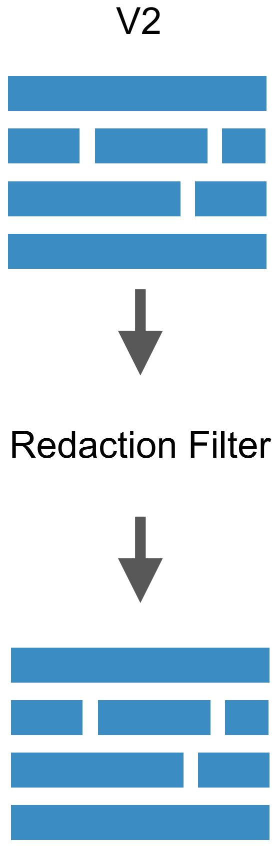
1. Clean Valid Data
2. Authenticated before ingestion
3. Backwards Compatible



Data Architecture

Structured Data Store

1. Can Sanitize and Store in GDPR compliant Democratized All-Access store too



Valid, Compliant, Clean Data

FALL BACK IN LOVE WITH YOUR DATA ARCHITECTURE

Questions?

First Steps

Quick Spark 101

1. DataFrames / Datasets

DataFrame's have Structure!

```
val us = Seq(
    ("Scott", "Teacher"),
    ("You", "Student")
    ).toDF("name", "role")

us.printSchema
/*
root
 |-- name: string (nullable = true)
 |-- role: string (nullable = true)
*/
```

First Steps

DataFrames/Datasets

Easily joined, mutated and aggregated

```
coffeeStand
  .join(coffeeRatings, coffeeStand("name") === coffeeRatings("coffeeName"))
  .drop("coffeeName")
  .groupBy("name")
  .agg(avg("score") as "rating")
  .sort(desc("rating"))
```

First Steps

DataFrames/Datasets

Did I mention that all of this can be done streaming? Let's take a peek.

/part2/coffee/streaming_coffee.scala

Your First Streaming App

```scala
import spark.implicits._

def asCoffeeRating(input: String): CoffeeRating = {
    val data = input.split(",")
    val coffeeName = data(0)
    val score = data(1).toInt
    val note = if (data.size > 2) Some(data(2)) else None
    CoffeeRating(coffeeName, score, note)
}

val coffeeStandDF = sparkSession.sparkContext.parallelize(availableCoffee, 3).toDF
val coffeeRatingsReader = sparkSession.readStream.format("socket").option("host", "localhost").option("port", 9999).load()
val rawRatingsData: Dataset[String] = coffeeRatingsReader.as[String]

val coffeeRatingsInput = rawRatingsData.map { asCoffeeRating }.toDF
val coffeeAndRatingsDF = coffeeStandDF.join(coffeeRatingsInput, coffeeStandDF("name") === coffeeRatingsInput("coffeeName"))
val averageRatings = coffeeAndRatingsDF.groupBy(col("name")).agg(avg("score") as "rating").sort(desc("rating"))
val query = averageRatings.writeStream.outputMode("complete").format("console").start()
```

/part2/coffee/streaming_coffee.scala

First Steps

Code Walk Through

/part2/coffee/

FIRST STEPS: DOING SOME THINGS WITH SPARK

Questions?

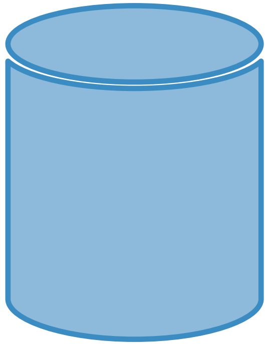
PART TWO: CLEAN AND EXPLORE YOUR DATA

Cause we don't all have Data Contracts!

Data Analysis

Core Concepts

1. Loading
2. Exploring
3. Cleaning
4. Filling
5. More Exploring
6. Apriori
7. KMeans



Let's Play with Wine Reviews

/part2/wine/hello-wine.scala
/part2/wine/wine_reviews.scala
/part2/wine/wine_reviews_json.scala

Data Cleaning and Analysis

Code Walk Through

/part2/wine/

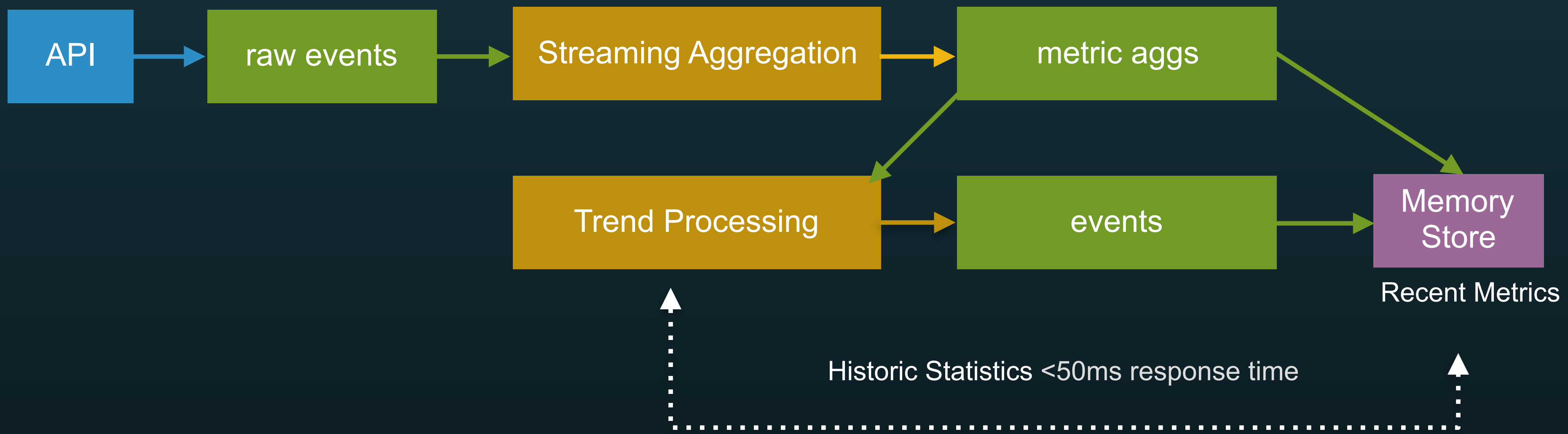
PART TWO: CLEAN AND EXPLORE YOUR DATA

Questions?

PART THREE: TREND DISCOVERY

Using statistics to understand unlabeled trends in streaming datasets

STREAMING TREND DISCOVERY: ARCHITECTURE

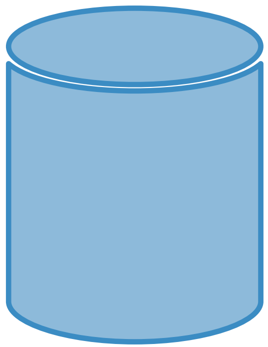


Trend Discovery

Core Concepts

1. Kafka Streaming
2. Windowing
3. Watermarking
4. FlatMapGroupsWithState
5. DataSketching
6. Monadic Systems
7. And More!

Let's graduate from the Shell...



/part3/streaming-trend-discovery

Trend Discovery

Data Model

1. Generic Structures are nice.

```
message MetricAggregation {
  optional string metric                  = 1;
  optional uint64 window_start            = 2;
  optional uint64 window_end              = 3;
  optional string window_interval         = 4;
  optional uint32 samples                 = 5;
  optional Stats stats                    = 6;
  optional Histogram histogram            = 7;
  optional Dimensions dimensions          = 8;
  optional string dimension_hash          = 10;
}
```

/part3/streaming-trend-discovery

Trend Discovery

Data Model

1. Standardize on conventions

```
message Window {
  optional string start     = 1;
  optional uint64 start_ms  = 2;
  optional string interval  = 3;
  optional string end       = 4;
  optional uint64 end_ms    = 5;
}
```

/part3/streaming-trend-discovery

Trend Discovery

Data Model

1. Solve common problems

```
message Stats {
  optional double min      = 1;
  optional double p25      = 2;
  optional double median   = 3;
  optional double p75      = 4;
  optional double p90      = 5;
  optional double p95      = 6;
  optional double p99      = 7;
  optional double max      = 8;
  optional double mean     = 9;
  optional double sd       = 10;
  optional double variance = 11;
}
```

/part3/streaming-trend-discovery

Trend Discovery

Data Model

1. Names should describe the data for humans.

```
message Metric {
  optional uint64 timestamp          = 1;
  optional string group_key          = 2;
  optional string dimensional_hash   = 3;
  optional string metric_name        = 4;
  optional string label              = 5;
  optional float value               = 6;
  optional string carrier            = 7;
  optional string country            = 8;
  optional string route              = 9;
  optional string direction          = 10;
}
```

/part3/streaming-trend-discovery

Trend Discovery

Windowing / Watermark

1. Names should describe the data for humans.

/part3/streaming-trend-discovery

```
callRecords
  .withWatermark("timestamp", s"$watermarkInterva
  .dropDuplicates("callSid", "timestamp")
  .groupBy($"accountSid", window($"timestamp", s"
  .agg(
    min("pdd") as "minPdd",
    round(avg("pdd"), 2) as "avgPdd",
    max("pdd") as "maxPdd",
    min("duration") as "minDuration",
    round(avg("duration"), 2) as "avgDuration",
    max("duration") as "maxDuration",
    count("callSid") as "calls",
    CallStateAggregation($"callState") as "callSt
    DirectionAggregation($"direction") as "direct
    DisconnectedByAggregation($"disconnectedBy")
    LastSipResponseAggregation($"lastSipResponseL
    ProvidersAggregation($"providerSid") as "prov
    CountriesAggregation($"callerCountry") as "ca
    CountriesAggregation($"calleeCountry") as "ca
  )
```

Trend Discovery

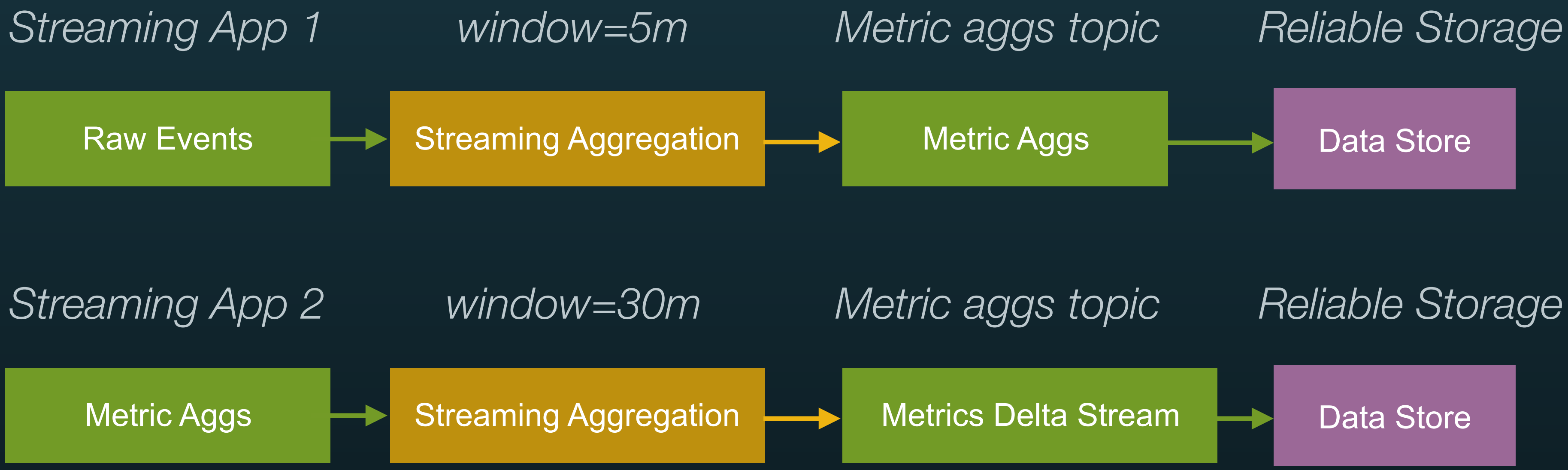
Monadic Systems

1. Distributed
2. Mergeable
3. Accurate

```scala
abstract class InsightsAggregation extends UserDefinedAggregateFunction {
  type T<: Product
  val log: Logger

  def productEncoder: Encoder[T]
  def productSchema: StructType = productEncoder.schema

  lazy val fieldIndex: Map[String, Int] = productSchema.fieldNames.map {
    fieldName -> productSchema.fieldIndex(fieldName)
  }.toMap

  val fieldLength: Int

  override def inputSchema: StructType = InsightsAggregation.inputStringR
  override def bufferSchema: StructType = productSchema
  override def dataType: DataType = productSchema
  override def deterministic: Boolean = true

  override def initialize(buffer: MutableAggregationBuffer): Unit = {
    (0 to fieldLength).foreach { num =>
      buffer(num) = 0L
    }
  }

  override def merge(cache: MutableAggregationBuffer, state: Row): Unit =
    (0 to fieldLength).foreach { num =>
      cache(num) = cache.getLong(num) + state.getLong(num)
    }
  }

}
```

Trend Discovery

Anatomy of the Discovery Engine



Trend Discovery

Code Walk Through

/part3/streaming-trend-discovery

[More Documentation]()

PART THREE: TREND DISCOVERY

Questions?

PART FOUR: TESTING AND OTHER PRODUCTION CONSIDERATIONS

How to test, ship, update and monitor massive streaming applications

Testing and Prod

Core Concepts

1. Testing Spark Apps

1. See `EventMemoryStreamSpec`

We want to ensure that we can test exactly HOW our aggregations will work.

We should have a repeatable pattern that can be employed on all spark applications.

We should be as confident with our Spark applications as we do traditional Enterprise apps

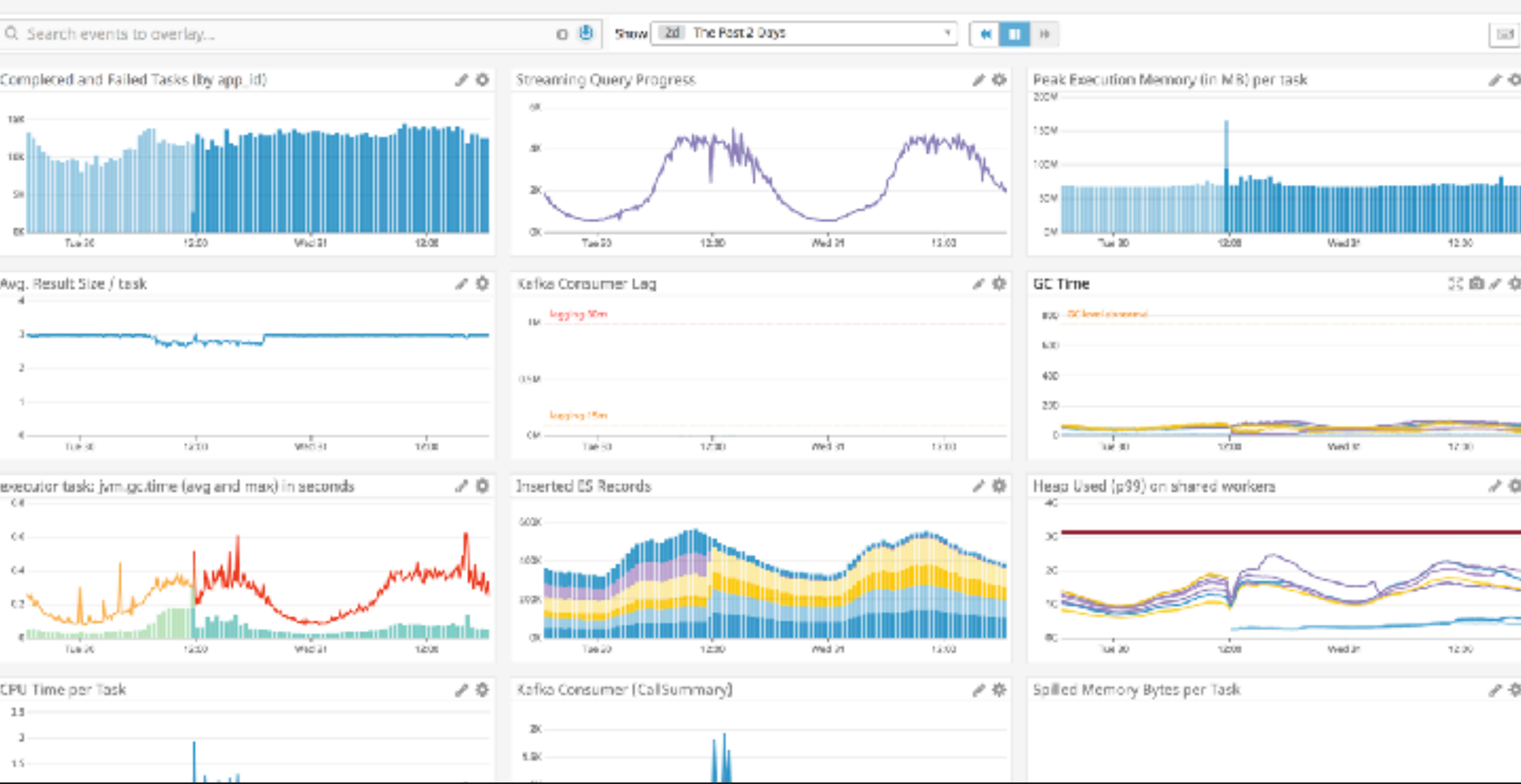
Testing and Prod

Core Concepts

1. SparkApplicationListener
2. StreamingQueryListener

1. *Emits all task, stage, job level details back to the driver.*
2. *Emits progress and statistics for the runtime behavior of a streaming query*

You can use these metrics to Emit to DataDog or other system to be able to track the performance and heath of your system



#streamingdiscovery | @newfront | @odsc

Testing and other production considerations

Code Walk Through

/part3/streaming-trend-discovery/.. /listeners/ .. /tests

PART FOUR: TESTING AND OTHER PRODUCTION CONSIDERATIONS

Questions?