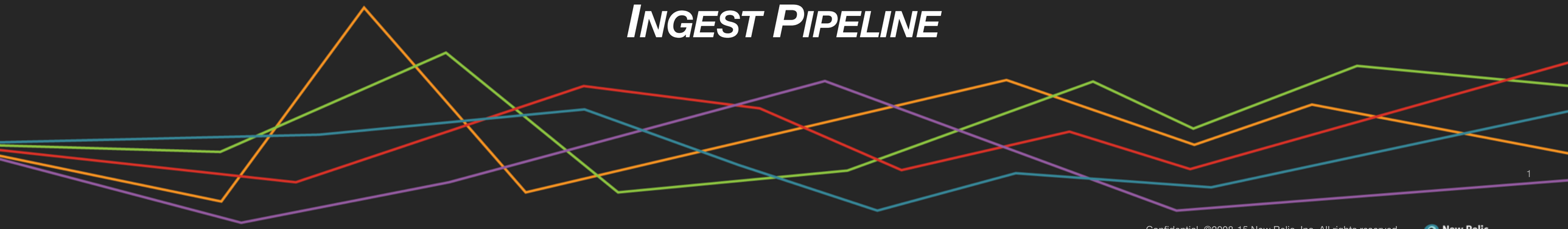
Evaluating Streaming Framework Performance for a Large-Scale Aggregation Pipeline

Ron Crocker

(rcrocker@newrelic.com)

Principal Engineer & Architect

Ingest Pipeline



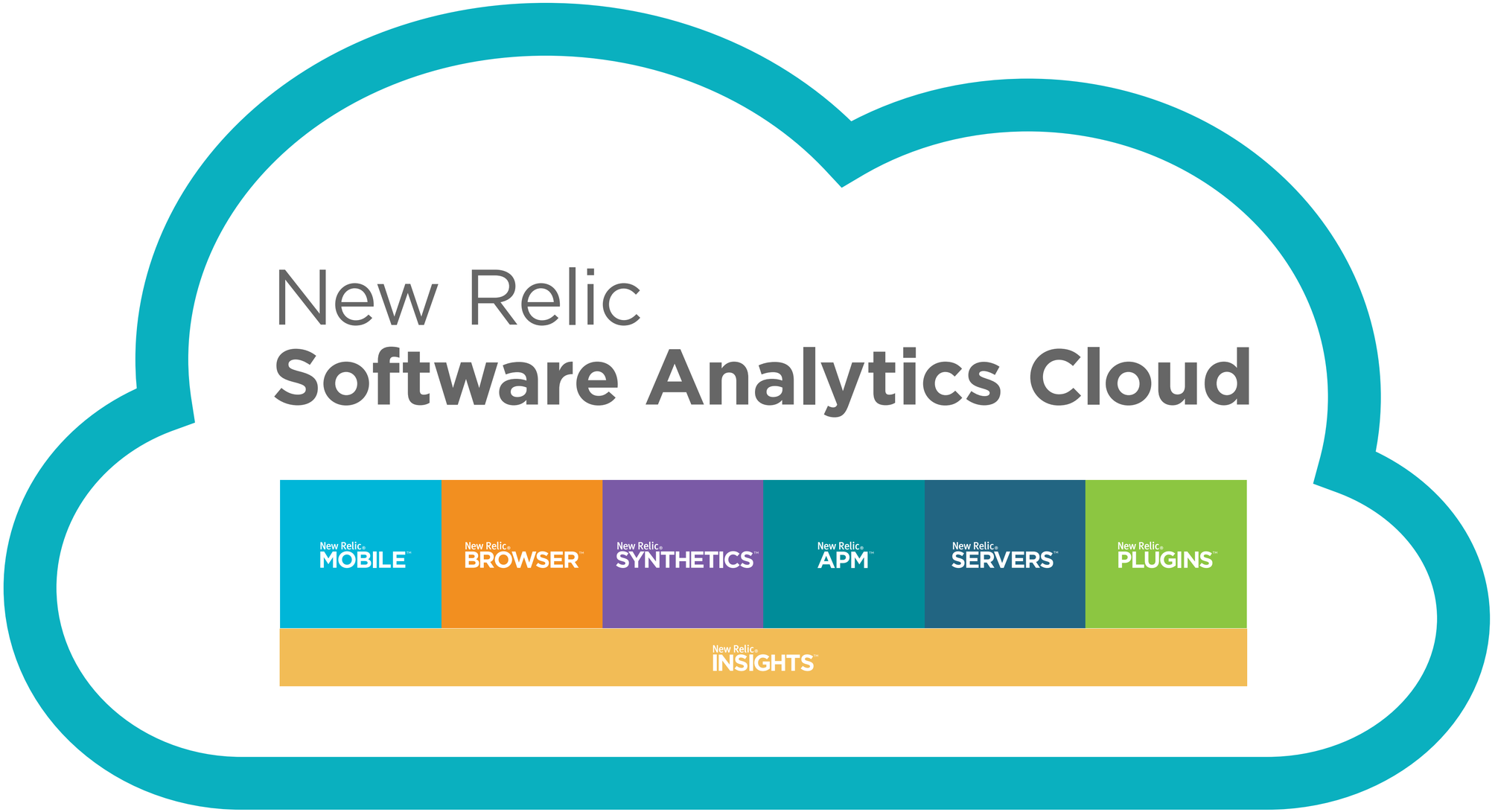
New Relic.

This document and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. ("New Relic") to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic's express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as "believes," "anticipates," "expects" or words of similar import.

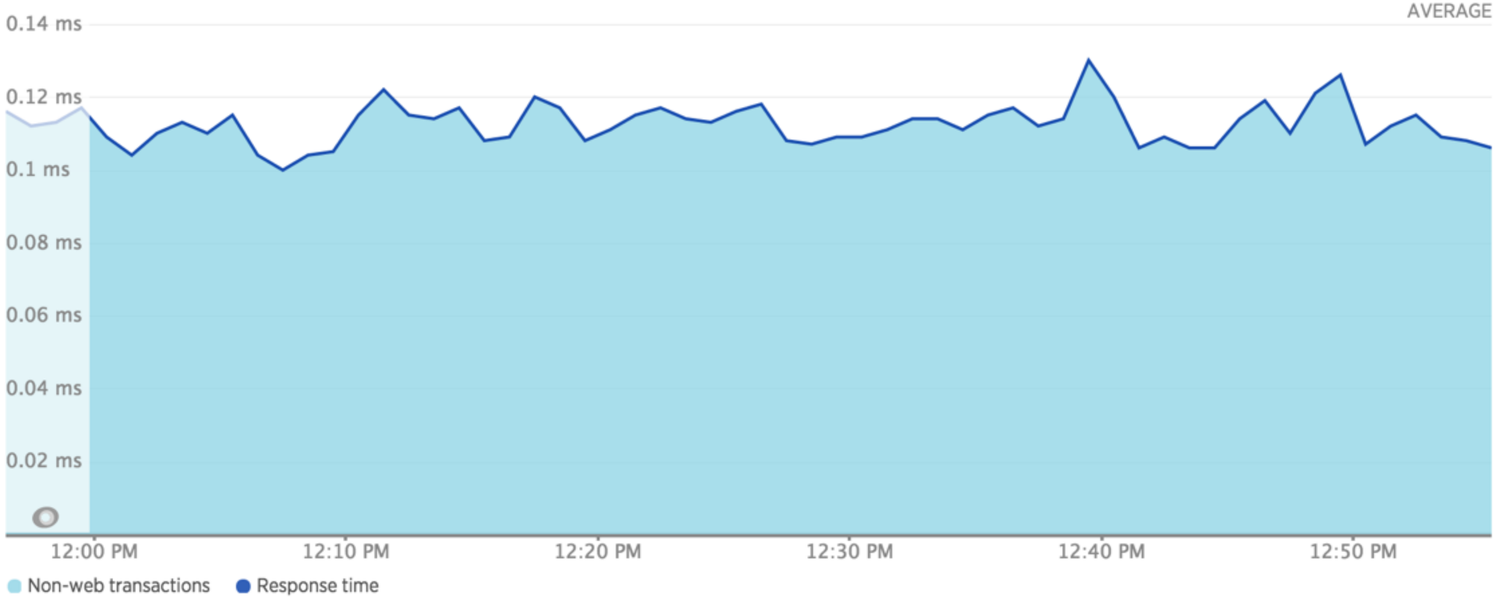
Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic's current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings we make with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic's Investor Relations website at ir.newrelic.com or the SEC's website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this document or otherwise, with respect to the information provided.


New Relic
Software Analytics Cloud
New Relic MOBILE
New Relic BROWSER
New Relic SYNTHETICS
New Relic APM
New Relic SERVERS
New Relic PLUGINS
New Relic INSIGHTS

Non-web transactions time

0.112 ms
AVERAGE

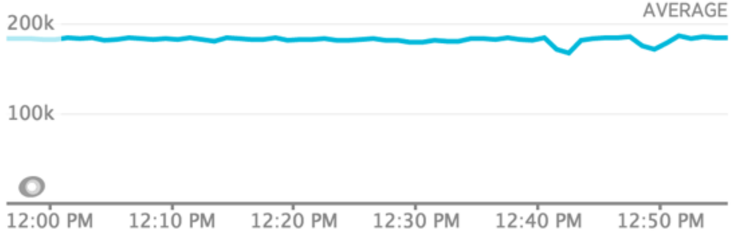


Apdex score

NS [0.5]*
APP SERVER

Throughput

182k rpm
AVERAGE



Transactions

	App server time
primaryTimesliceHarvest	609 ms
Transaction traces: 1 s 0.9 s 0.9 s	
publish	0.388 ms
Transaction traces: 0.5 s 0.5 s 0.4 s	
ResolvedTimeslices	0.144 ms
Transaction traces: 0.4 s 0.1 s 0.1 s	
publish	0.026 ms
Transaction traces: 0.1 s 0.1 s 0.1 s	
JettyMonitor	0.0134 ms
Transaction traces: n/a	

Error rate

0.00 %

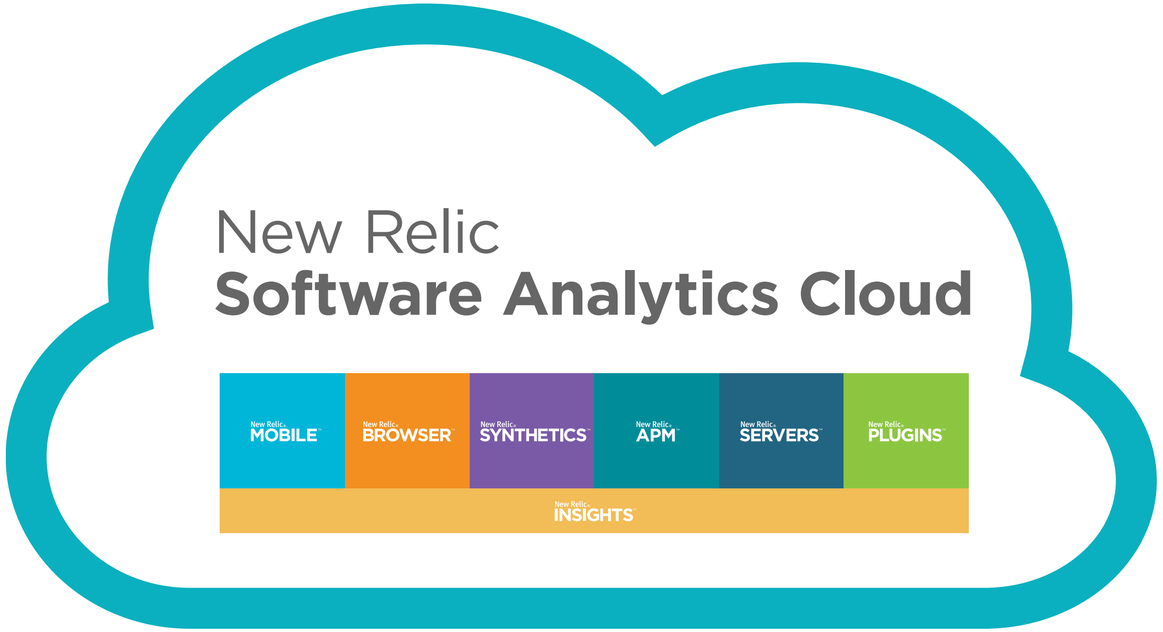
Application activity

In progress | Event log | Filter

There are no violations in progress right now.

4 servers

Server name	Apdex	Resp. time	Throughput	Error Rate	CPU usage	Memory
Minute Timeslice Aggregator	0 ms	184k rpm	0.00 err%			



accepts over **16M** requests

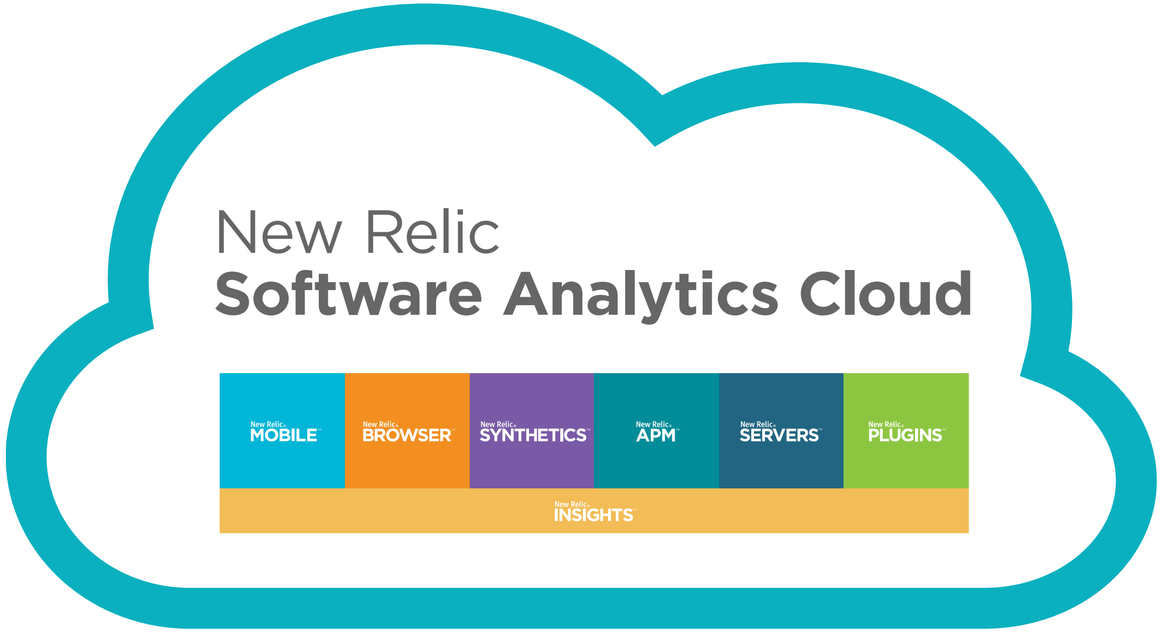
stores over **2M** analytic events

aggregates over **800M** metrics

queries over **3B** data points

EVERY MINUTE

New Relic.

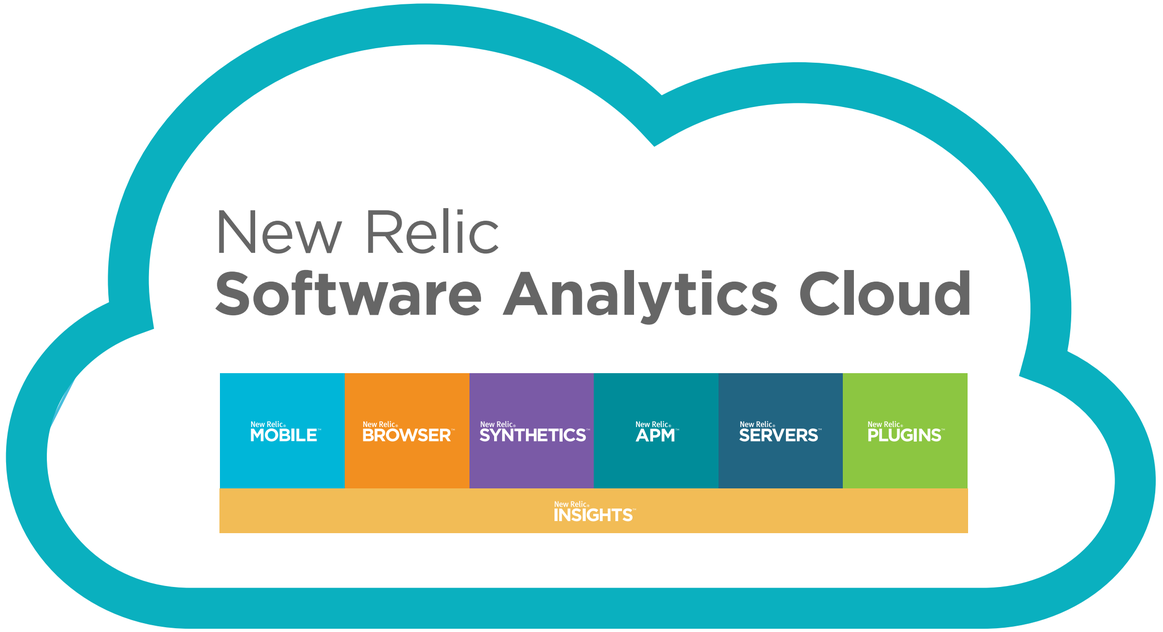

New Relic
Software Analytics Cloud
New Relic MOBILE
New Relic BROWSER
New Relic SYNTHETICS
New Relic APM
New Relic SERVERS
New Relic PLUGINS
New Relic INSIGHTS

New Relic

contains over **200** different services

more than **2.5** PETABYTES SSD storage

maintained/ built by **25+** engineering teams



Thanks for the pic! https://www.flickr.com/photos/stephenyeargin/7466608166

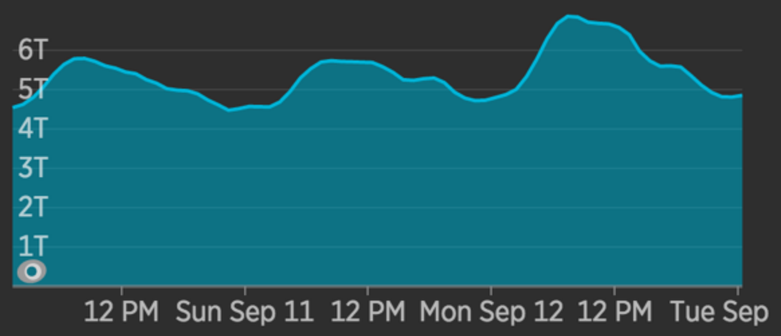
Production Kafka

Created by rcrocker@newrelic.com

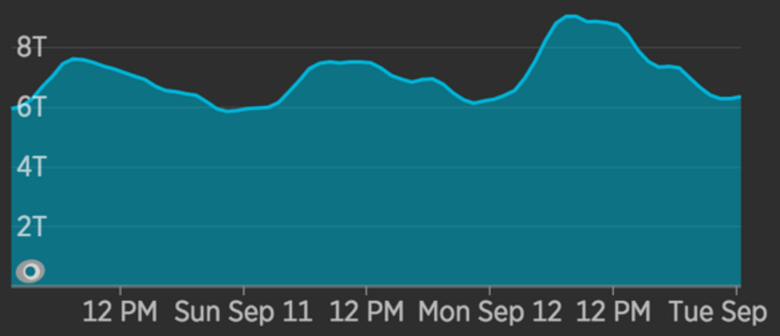
powered by New Relic INSIGHTS

Last edited 9/12/16

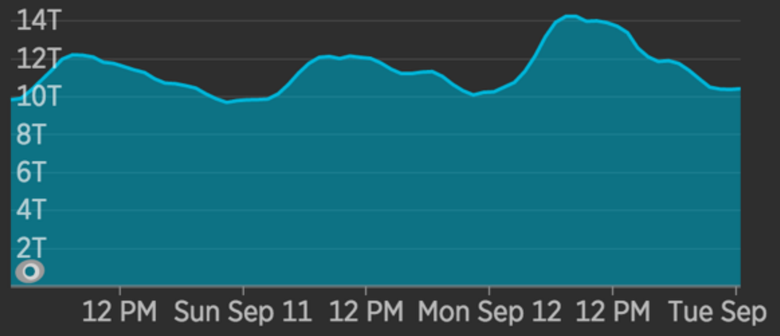
Kafka Producers
Since 72 hours ago



Kafka Consumers
Since 72 hours ago



Kafka replication
Since 72 hours ago



Kafka Producers
Since 24 hours ago

139

Total Byte Count (TB)

Kafka Consumers
Since 24 hours ago

175

Total Byte Count (TB)

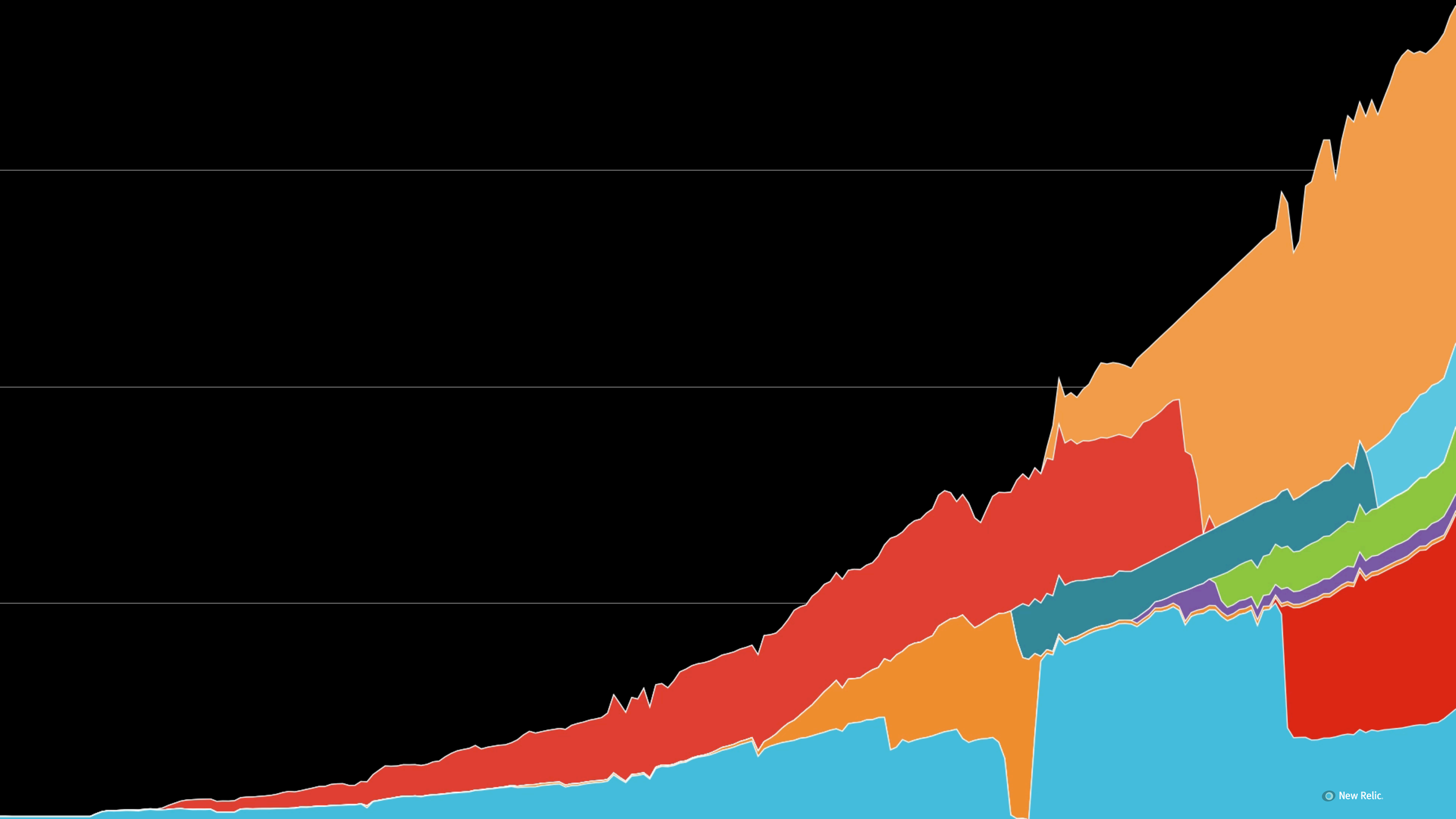
Kafka replication
Since 24 hours ago

294

Total Byte Count (TB)

NR Export

Permalink

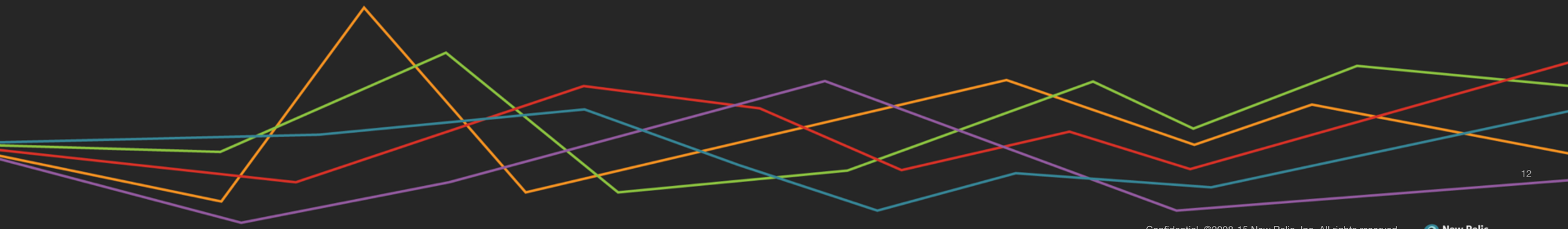

New Relic

Goals for evaluating streaming systems

- Understand performance characteristics
- Understand operations characteristics

How New Relic works…

… the cartoon version



A_1 An instance of your application running on a host

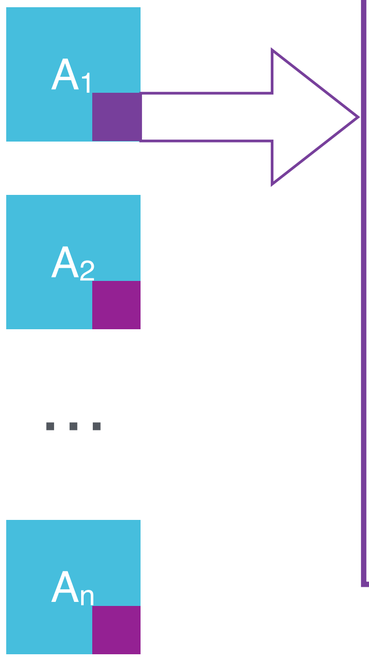
A_2 Another instance of your application running on another host

...

A_n More instances of your application running on more hosts

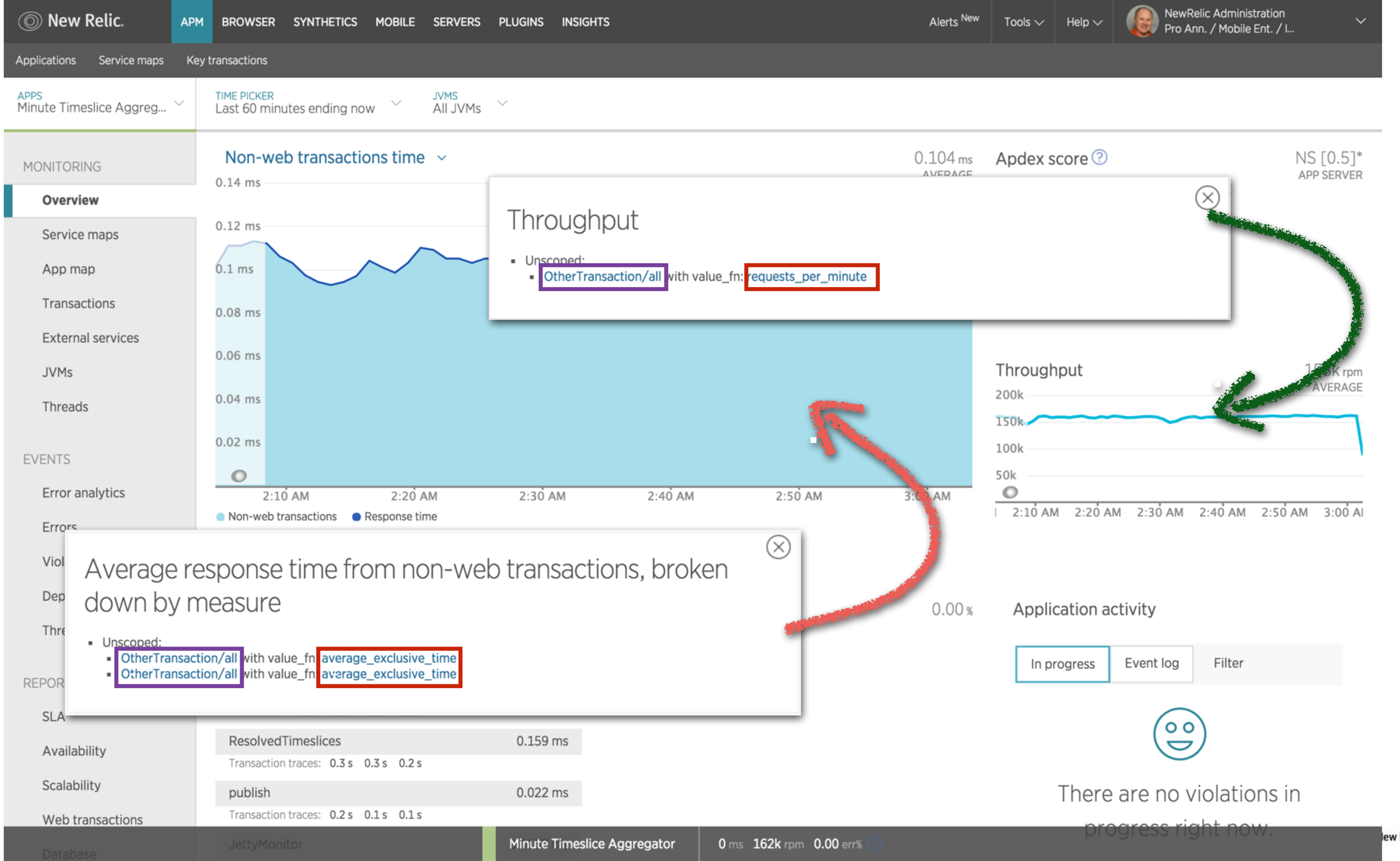
New Relic Agent reports data to New Relic

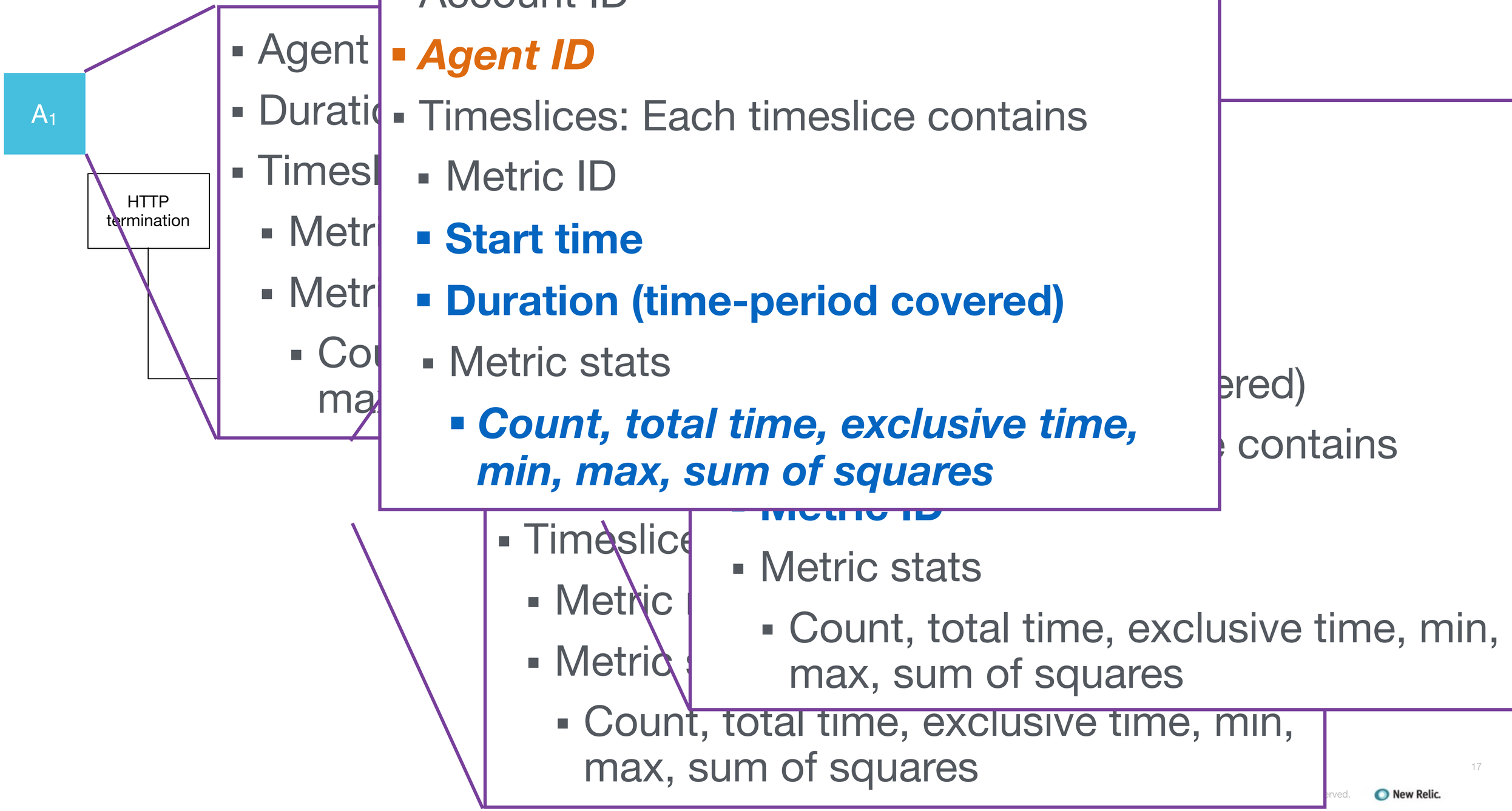
New Relic.



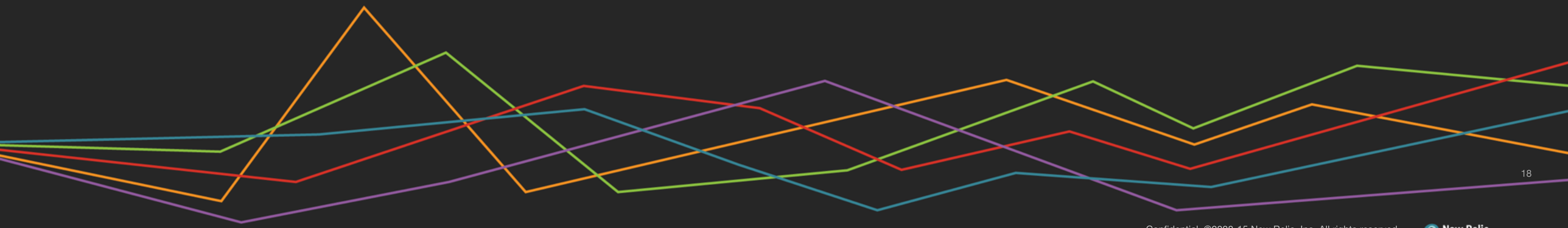
- Agent Token (≈ account ID, agent ID)
- Duration (time-period covered)
- Timeslices: Each timeslice contains
  - Metric name
  - Metric stats
    - Count, total time, exclusive time, min, max, sum of squares

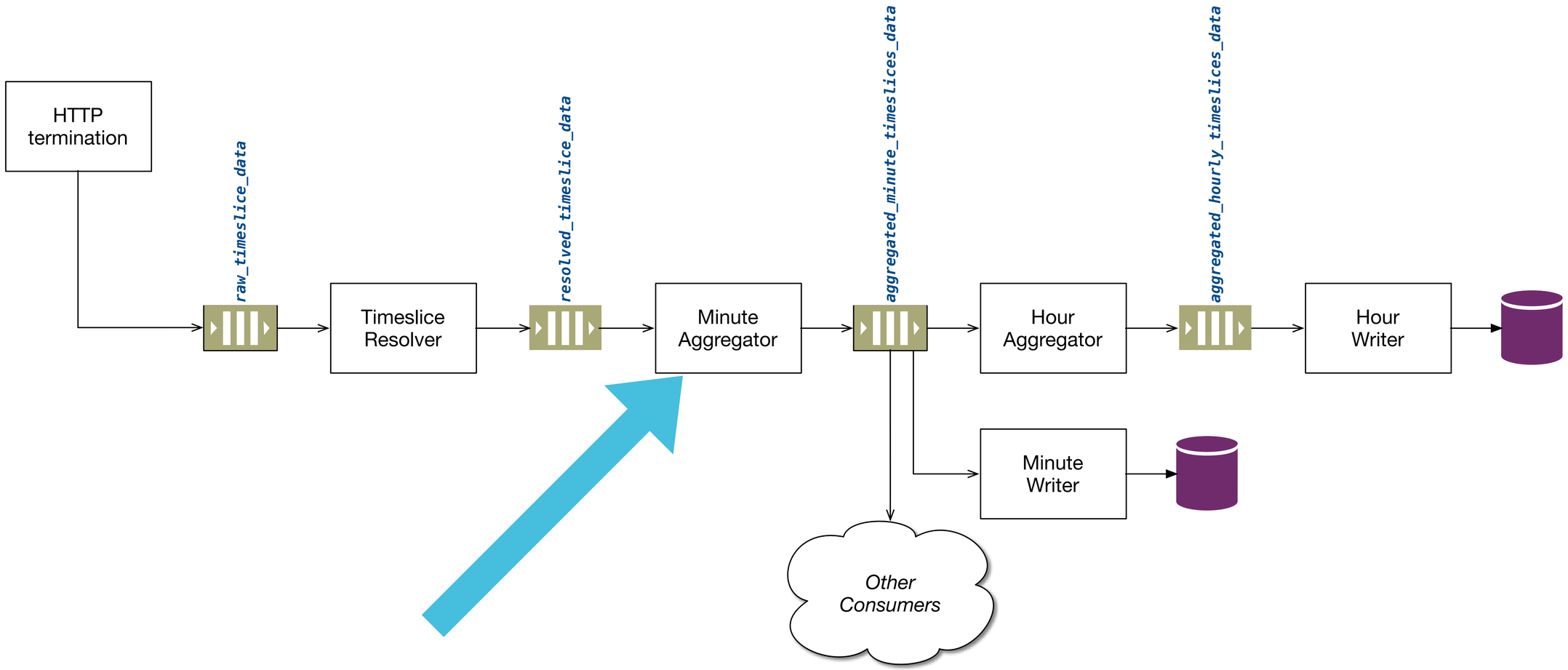
HTTP post to <something>.newrelic.com


New Relic.
APM
BROWSER
SYNTHETICS
MOBILE
SERVERS
PLUGINS
INSIGHTS
Alerts New
Tools
Help
NewRelic Administration
Pro Ann. / Mobile Ent. / I...
Applications
Service maps
Key transactions
APPS
Minute Timeslice Aggreg...
TIME PICKER
Last 60 minutes ending now
JVMS
All JVMs
MONITORING
Overview
Service maps
App map
Transactions
External services
JVMs
Threads
EVENTS
Error analytics
Errors
Violations
Deployments
Thresholds
REPORTING
SLA
Availability
Scalability
Web transactions
Database
Non-web transactions time
0.104 ms
AVERAGE
0.14 ms
0.12 ms
0.1 ms
0.08 ms
0.06 ms
0.04 ms
0.02 ms
2:10 AM
2:20 AM
2:30 AM
2:40 AM
2:50 AM
Non-web transactions
Response time
Throughput
Unscoped:
OtherTransaction/all with value_fn: requests_per_minute
Apdex score
NS [0.5]*
APP SERVER
Throughput
AVERAGE
200k
150k
100k
50k
2:10 AM
2:20 AM
2:30 AM
2:40 AM
2:50 AM
Average response time from non-web transactions, broken down by measure
Unscoped:
OtherTransaction/all with value_fn: average_exclusive_time
OtherTransaction/all with value_fn: average_exclusive_time
0.00 %
Application activity
In progress
Event log
Filter
There are no violations in progress right now.
ResolvedTimeslices
0.159 ms
Transaction traces: 0.3 s 0.3 s 0.2 s
publish
0.022 ms
Transaction traces: 0.2 s 0.1 s 0.1 s
JettyMonitor
Minute Timeslice Aggregator
0 ms 162k rpm 0.00 err%



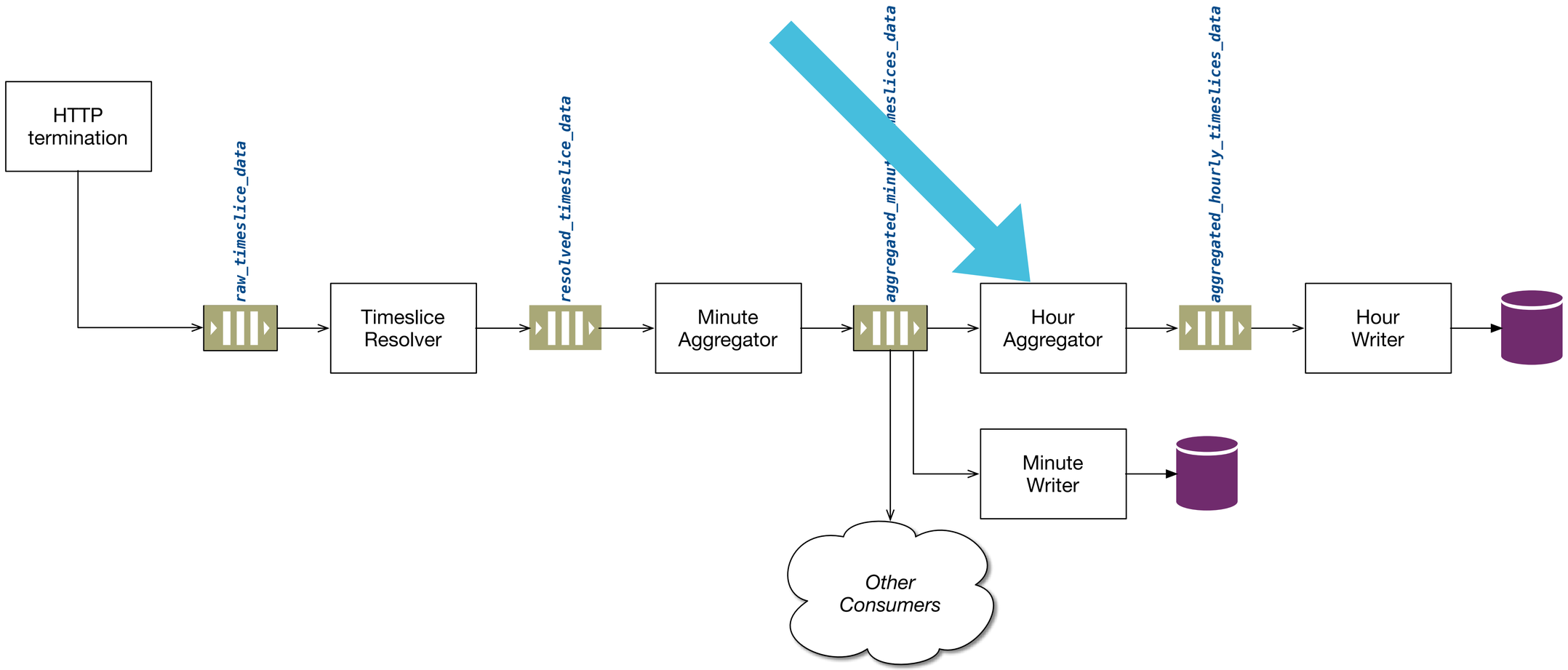
The Experiment





Why *Minute Aggregator*?

- **No external dependencies**
  - Performance comparisons solely focused on processing
- **Repeatable**
  - We can compare across technologies without needing to normalize
- **Important to our business**
  - ProvIDes aggregation across instances of your application
  - We could have benchmarked something else, like Yahoo benchmark or word count, but would it have mattered?



What about *Hour Aggregator*?

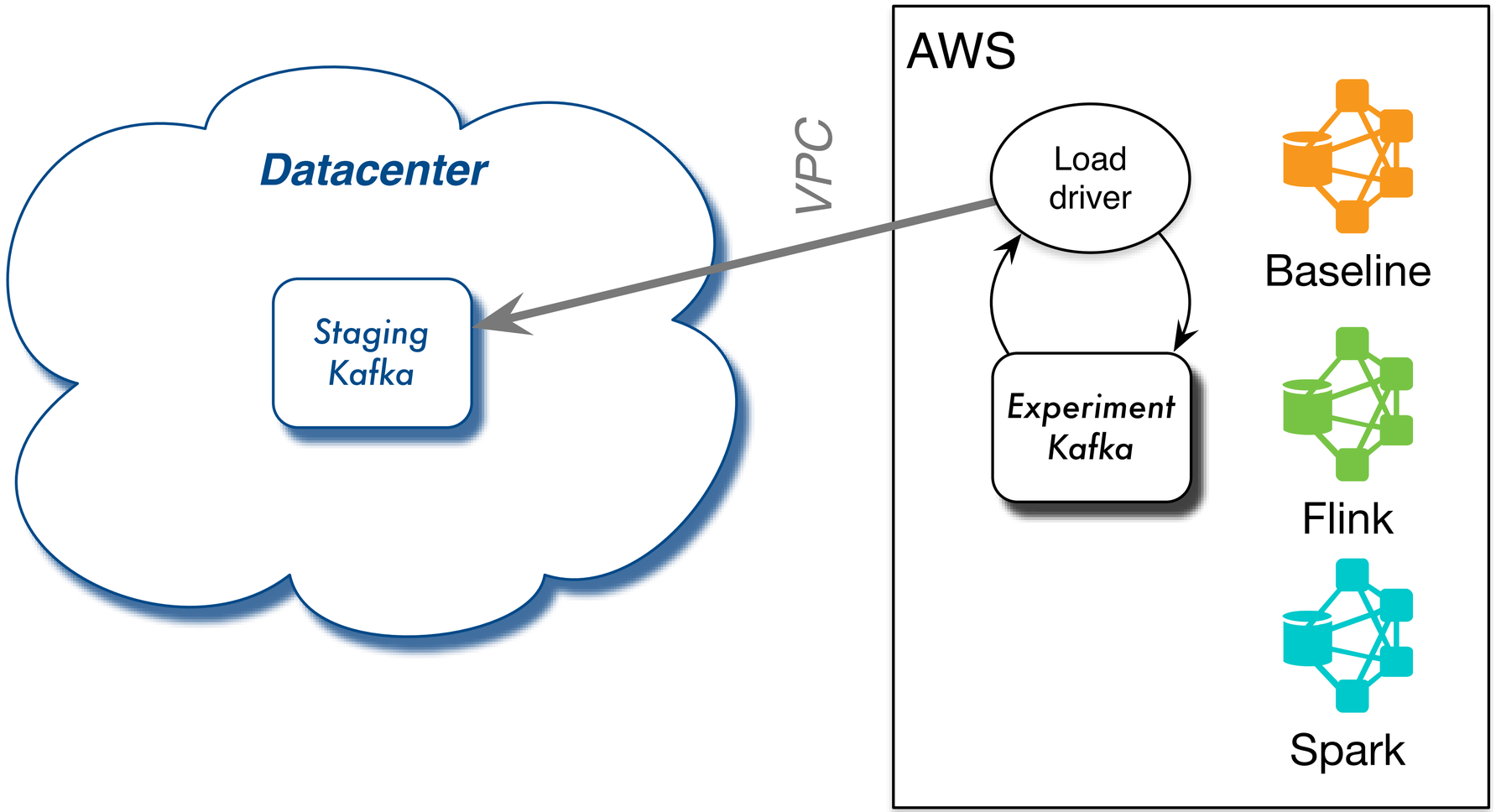
- **Similar to *Minute Aggregator***
  - *No external dependencies, Repeatable, Important to the business*
- **Needs to run for several hours to understand performance**
  - … and I'm that patient
- **Extra credit: Integrate into stream implementations**

Goals for evaluating streaming systems

- **Understand performance characteristics**
  - Performance at different arrival rates:
    - 100%
    - 6000%
    - *To infinity and beyond*

- **Understand operations characteristics**
  - *No explicit goal*

Evaluation Framework



AWS Configurations

- **Kafka + ZK**
  - 3 i2.8xlarge hosts
- **Baseline**
  - 3 m4.4xlarge hosts
- **Flink**
  - 4 m4.4xlarge hosts
- **Spark**
  - EMR - 1 master , 3 workers, all m4.4xlarge

AWS Configuration	i2.8xlarge	m4.4xlarge
Cores	32	16
RAM	244GB	64GB
Network Bandwidth	10Gbps	2Gbps

Experimental Kafka system

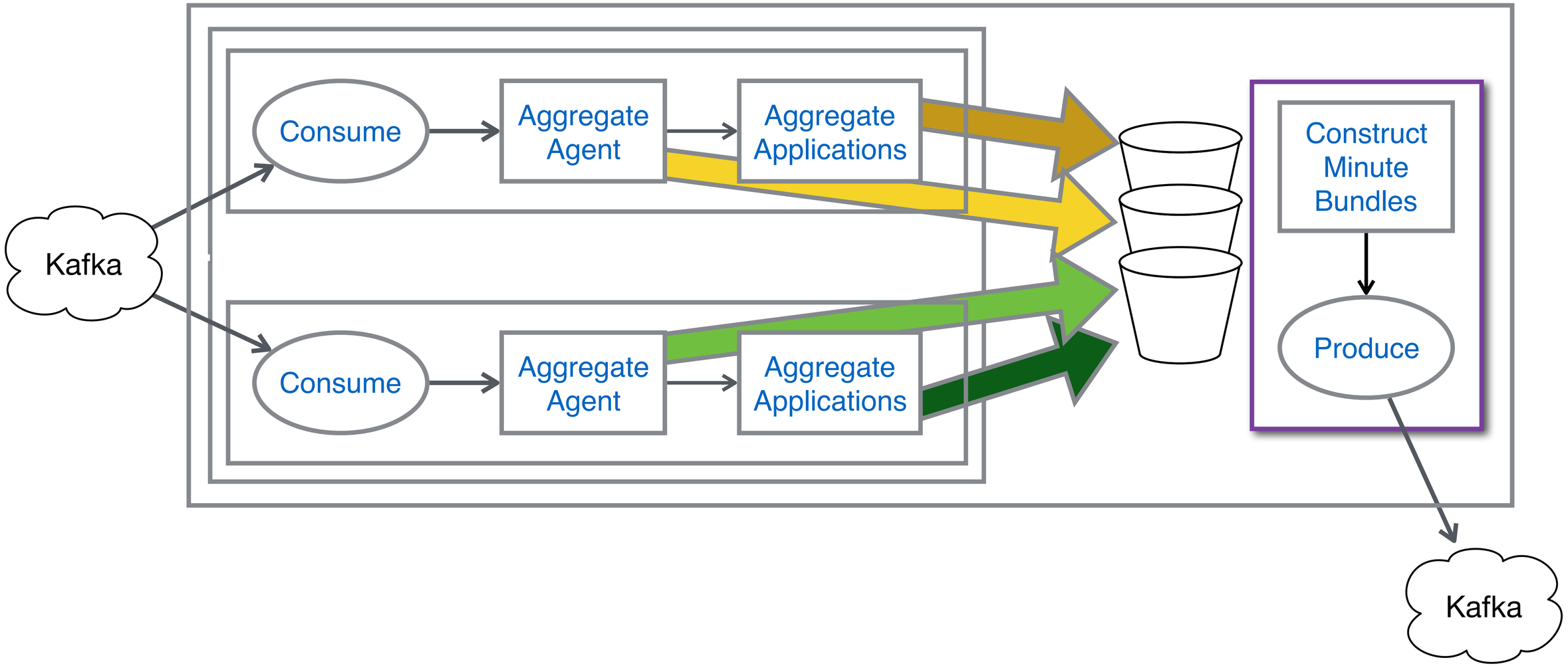
- **Kafka 0.8.2.2**
  - NR fork, includes back ports of some 0.9 features
- **# partitions: 16**
  - It's possible that this is too few partitions for the Baseline system

Load Driver

- **Generates simple synthetic load *based on real traffic***
  - *Real traffic = output of* Timeslice Resolver
  - *Load generated based on repeated messages*

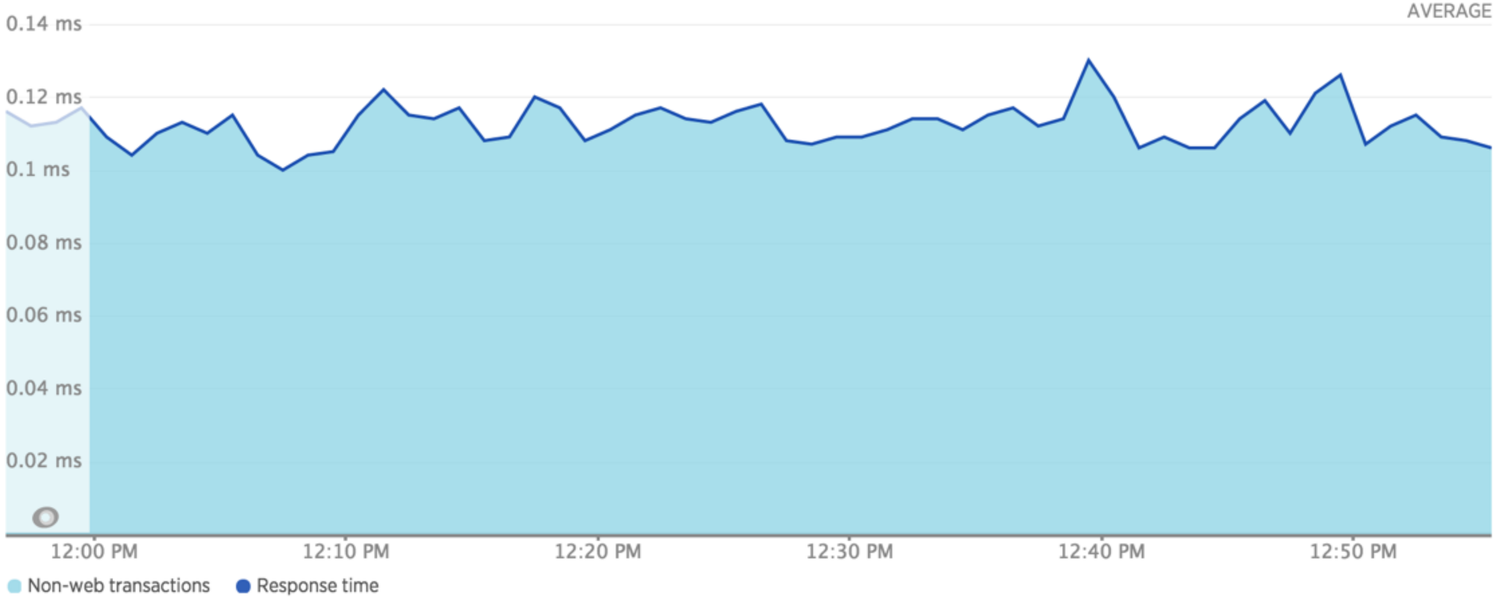
- **Synthesizing *interesting* load is challenging:**
  - Un-bundle timeslices
  - Generate re-bundled with new IDs - Agent, Account and/or Metrics
  - Repeat as necessary to get to load point

Baseline system - Our incumbent *Minute Aggregator*



Non-web transactions time

0.112 ms
AVERAGE

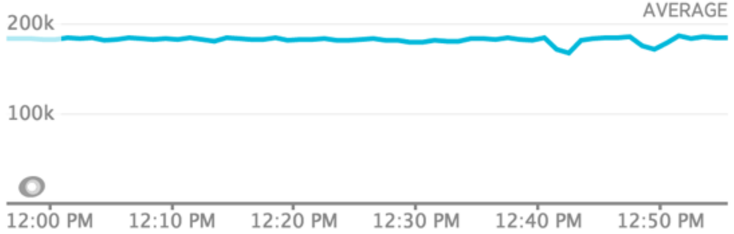


Apdex score

NS [0.5]*
APP SERVER

Throughput

182k rpm
AVERAGE



Transactions

	App server time
primaryTimesliceHarvest	609 ms
Transaction traces: 1 s 0.9 s 0.9 s	
publish	0.388 ms
Transaction traces: 0.5 s 0.5 s 0.4 s	
ResolvedTimeslices	0.144 ms
Transaction traces: 0.4 s 0.1 s 0.1 s	
publish	0.026 ms
Transaction traces: 0.1 s 0.1 s 0.1 s	
JettyMonitor	0.0134 ms
Transaction traces: n/a	

Error rate

0.00 %

Application activity

In progress | Event log | Filter

There are no violations in progress right now.

4 servers

Server name	Apdex	Resp. time	Throughput	Error Rate	CPU usage	Memory
Minute Timeslice Aggregator	0 ms	184k rpm	0.00 err%			

now.

JettyMonitor	0.0134 ms

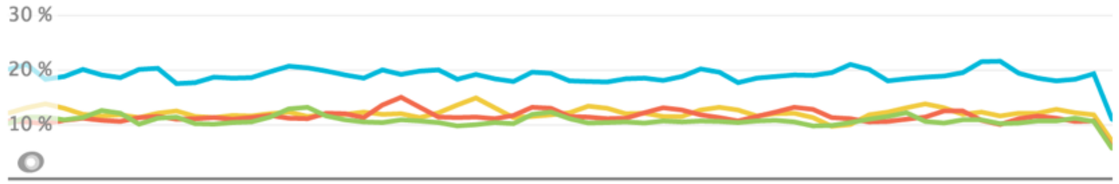
Transaction traces: n/a

4 servers

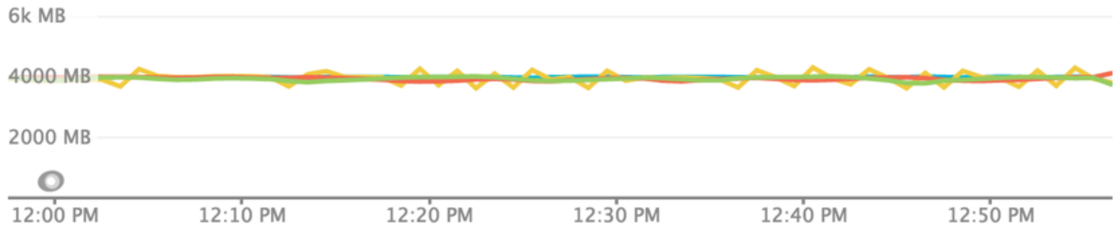
Response time

Throughput

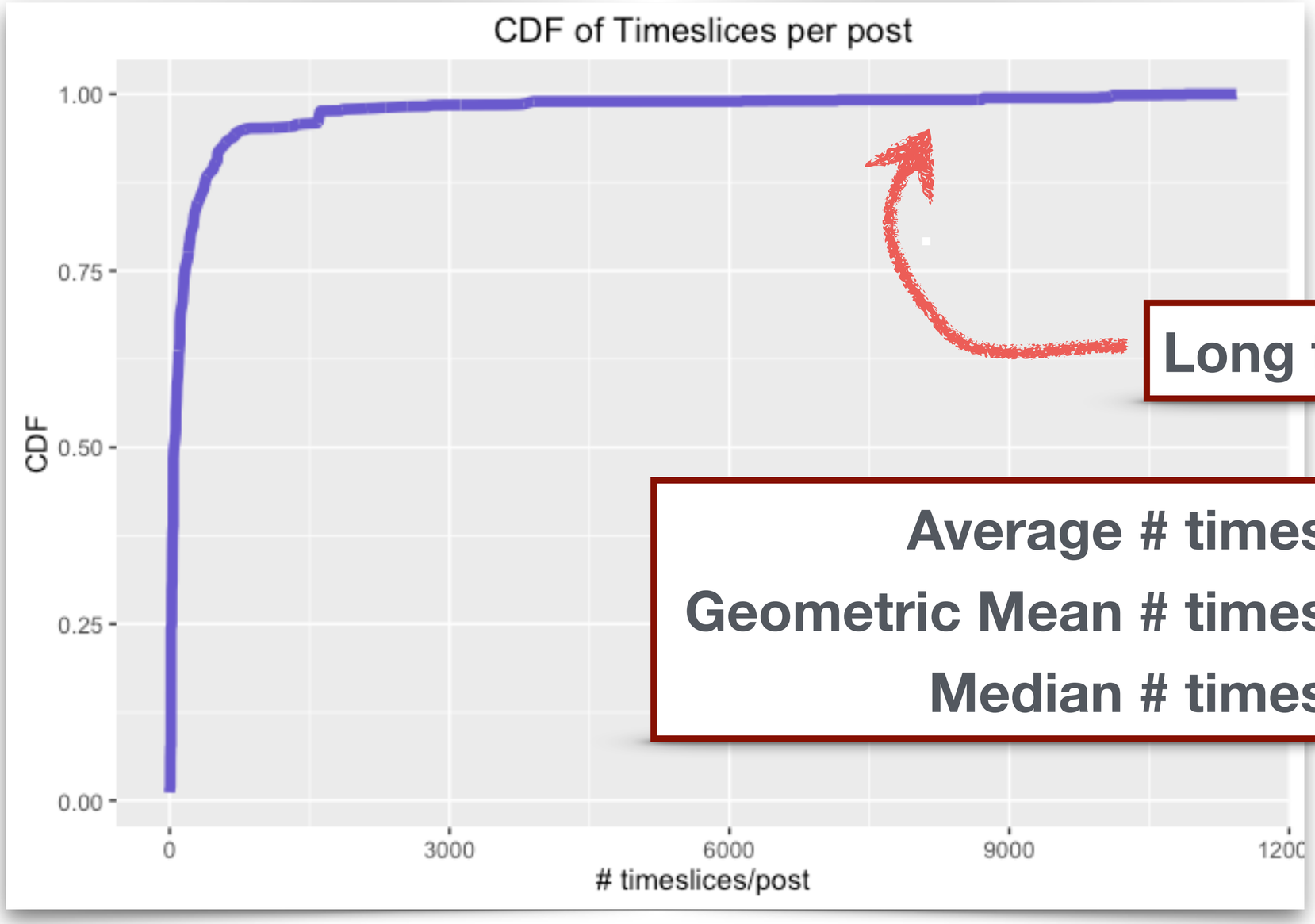
CPU usage



Memory



Distributions are not friendly…



Long tail…

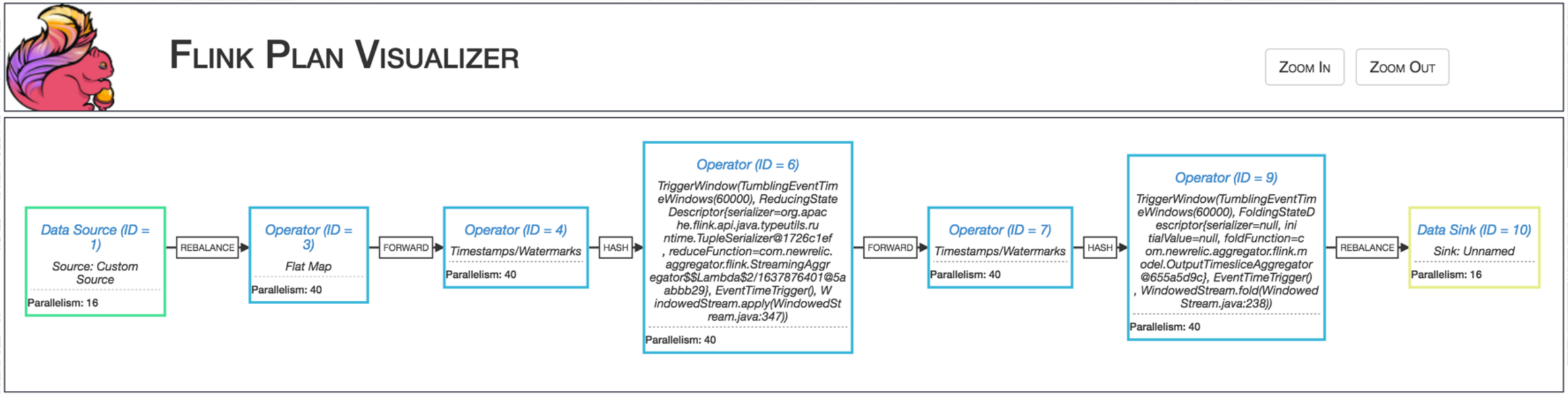
Average # timeslices: 279

Geometric Mean # timeslices: 64

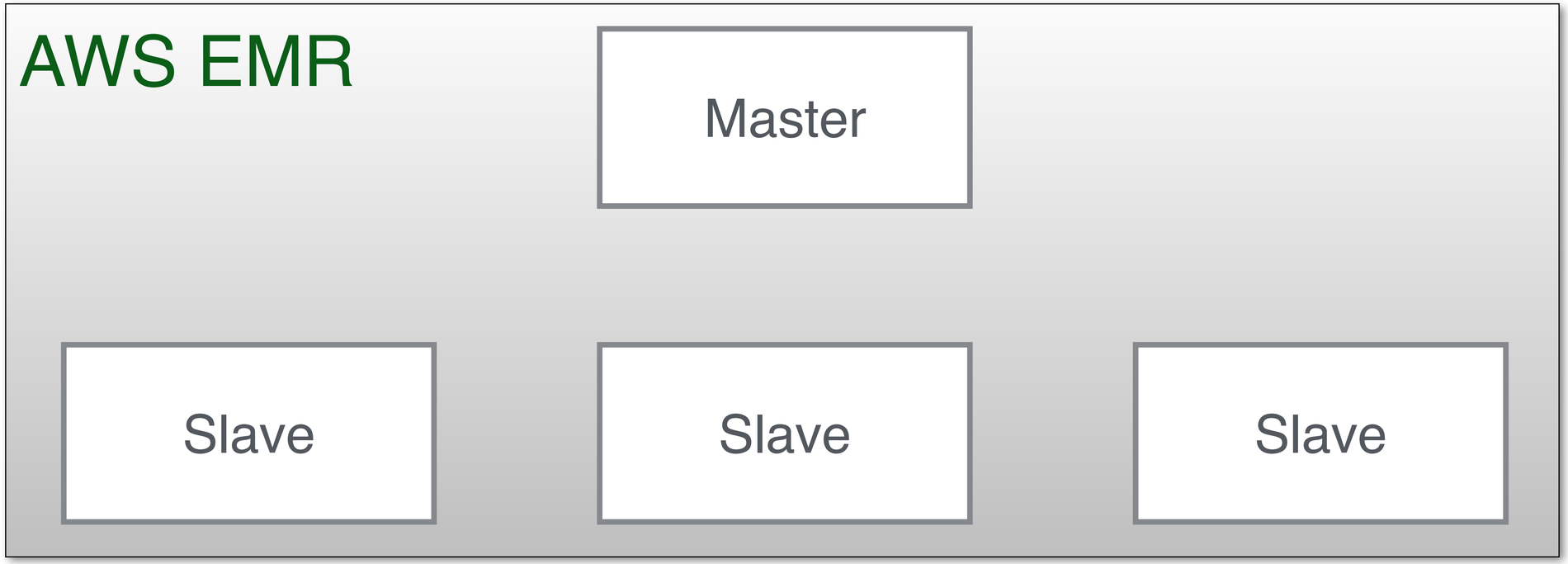
Median # timeslices: 44

Flink configuration

```java
public static void main(String[] args) throws Exception {
    final String confDir = System.getProperty("app.conf.dir");
    final String environment = System.getProperty("newrelic.environment");
    final DynamicConfiguration configuration = Configurations.configure(confDir, environment);

    Properties kafkaConsumerProperties = getKafkaConsumerProperties(configuration);

    logger.info("kafkaConsumerProperties: {}", kafkaConsumerProperties);

    StreamExecutionEnvironment see = StreamExecutionEnvironment.getExecutionEnvironment();
    see.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    see.getConfig().setAutoWatermarkInterval(1000);
    see.setParallelism(40);

    DataStream<TimesliceData> dataStream = see
            .addSource(new FlinkKafkaConsumer08<>(
                    timesliceConstants.RESOLVED_TIMESLICE_DATA_KAFKA_TOPIC_NAME,
                    new TimesliceDeserializer(),
                    kafkaConsumerProperties)).setParallelism(16);

    // aggregatedByMinuteStream contains events that are the aggregated timeslices by [MetricId,AgentId,AccountId]
    DataStream<TimesliceTuple> aggregatedByMinuteStream = dataStream
            .rebalance()
            .flatMap(new TimesliceTupleMapper())
            .assignTimestampsAndWatermarks(new TimesliceTupleTimestampExtractor())
            .keyBy(TimesliceTuple.WIDE_METRIC_ID,TimesliceTuple.AGENT_ID, TimesliceTuple.ACCOUNT_ID)
            .timeWindow(Time.seconds(60))
            .reduce(TimesliceTuple::aggregateWith);

    // packagedMinuteTimeslices are all the minute-aggregated timeslices for a given [AgentId,AccountId]
    DataStream<OutputTimesliceData> packagedMinuteTimeslices = aggregatedByMinuteStream
            .assignTimestampsAndWatermarks(new TimesliceTupleTimestampExtractor())
            .keyBy(TimesliceTuple.AGENT_ID, TimesliceTuple.ACCOUNT_ID)
            .timeWindow(Time.seconds(60))
            .fold(null, new OutputTimesliceAggregator());

    packagedMinuteTimeslices.addSink(new FlinkKafkaProducer08<OutputTimesliceData>(
            AGGREGATED_MINUTE_TIMESLICE_DATA,
            new OutputTimesliceDataSerializer(),
            getKafkaProducerProperties(configuration))).setParallelism(16);

    System.out.println(see.getExecutionPlan());

    see.execute();
}
```



Spark configuration

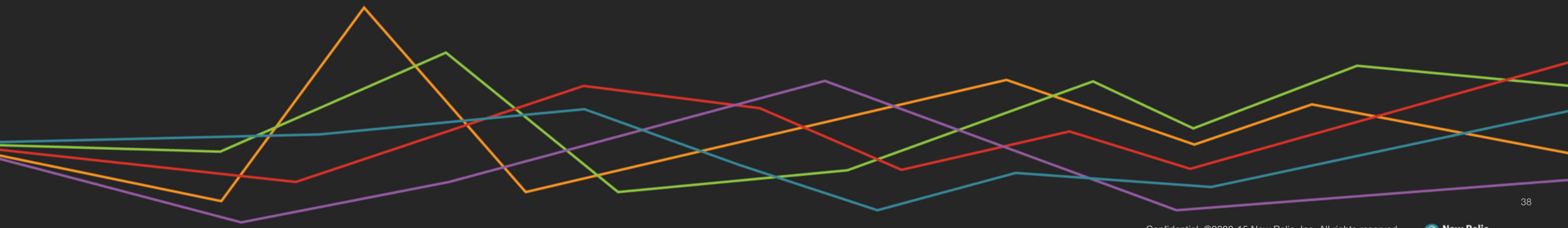


```
52          SparkConf conf = new SparkConf().setAppName("SparkTimesliceAggregator");
53          JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(1));
54
55          JavaPairInputDStream<byte[], TimesliceData> resolvedTimesliceStream = KafkaUtils.createDirectStream(jssc,
56                  byte[].class, TimesliceData.class,
57                  DefaultDecoder.class, TimesliceDeserializer.class,
58                  getKafkaConsumerProperties(configuration),
59                  (Set) ImmutableSet.of(RESOLVED_TIMESLICE_TOPIC));
60
61          JavaDStream<AggregatableTimeslice> resolvedTimeslicesFlattenedStream = resolvedTimesliceStream
62                  .flatMap(new TimesliceTupleMapper());
63
64          JavaPairDStream<Tuple3<Long, Integer, Integer>, AggregatableTimeslice>
65                  minuteAggregatedTimesliceStream = resolvedTimeslicesFlattenedStream
66                  .mapToPair(new AggregatableTimesliceKeyMapper())
67                  .reduceByKeyAndWindow(new TimesliceReducer(), Durations.minutes(1));
68
69          JavaMapWithStateDStream<Tuple2<Integer, Integer>, AggregatableTimeslice, OutputTimesliceData, OutputTimesliceData>
70                  packagedAggregatedMinuteTimeslices = minuteAggregatedTimesliceStream.mapToPair(new PairFunction<Tuple2<Tuple3<Long,
    Integer, Integer>, AggregatableTimeslice>, Tuple2<Integer, Integer>, AggregatableTimeslice>() {
71              @Override
72              public Tuple2<Tuple2<Integer, Integer>, AggregatableTimeslice> call(Tuple2<Tuple3<Long, Integer, Integer>,
    AggregatableTimeslice> keyValuePair) throws Exception {
73                  return new Tuple2<Tuple2<Integer, Integer>, AggregatableTimeslice>(
74                          new Tuple2<Integer, Integer>(keyValuePair._2.accountId, keyValuePair._2.agentId),
75                          keyValuePair._2);
76              }
77          }).mapWithState(StateSpec.function(new Function4<Time, Tuple2<Integer, Integer>, Optional<AggregatableTimeslice>,
    State<OutputTimesliceData>, Optional<OutputTimesliceData>>() {
78              @Override
79              public Optional<OutputTimesliceData> call(Time time, Tuple2<Integer, Integer> tuple, Optional<AggregatableTimeslice>
    wrappedValue, State<OutputTimesliceData> stateObject) throws Exception {
80                  AggregatableTimeslice value = wrappedValue.get();
81                  OutputTimesliceData newState = stateObject.exists() ?
82                          stateObject.get() :
83                          new OutputTimesliceData(value.getAccountId(),
84                                  value.getAgentId(),
85                                  value.getBeginTimeInSeconds(),
86                                  value.getEndTimeInSeconds(),
87                                  value.getMetricStoragePolicy());
88                  newState.appendTimeslice(value);
89                  stateObject.update(newState);
90                  return Optional.of(newState);
91              }
92          }));
93
94          packagedAggregatedMinuteTimeslices.foreachRDD(outputTimesliceDataJavaRDD -> produceToKafka(outputTimesliceDataJavaRDD));
```

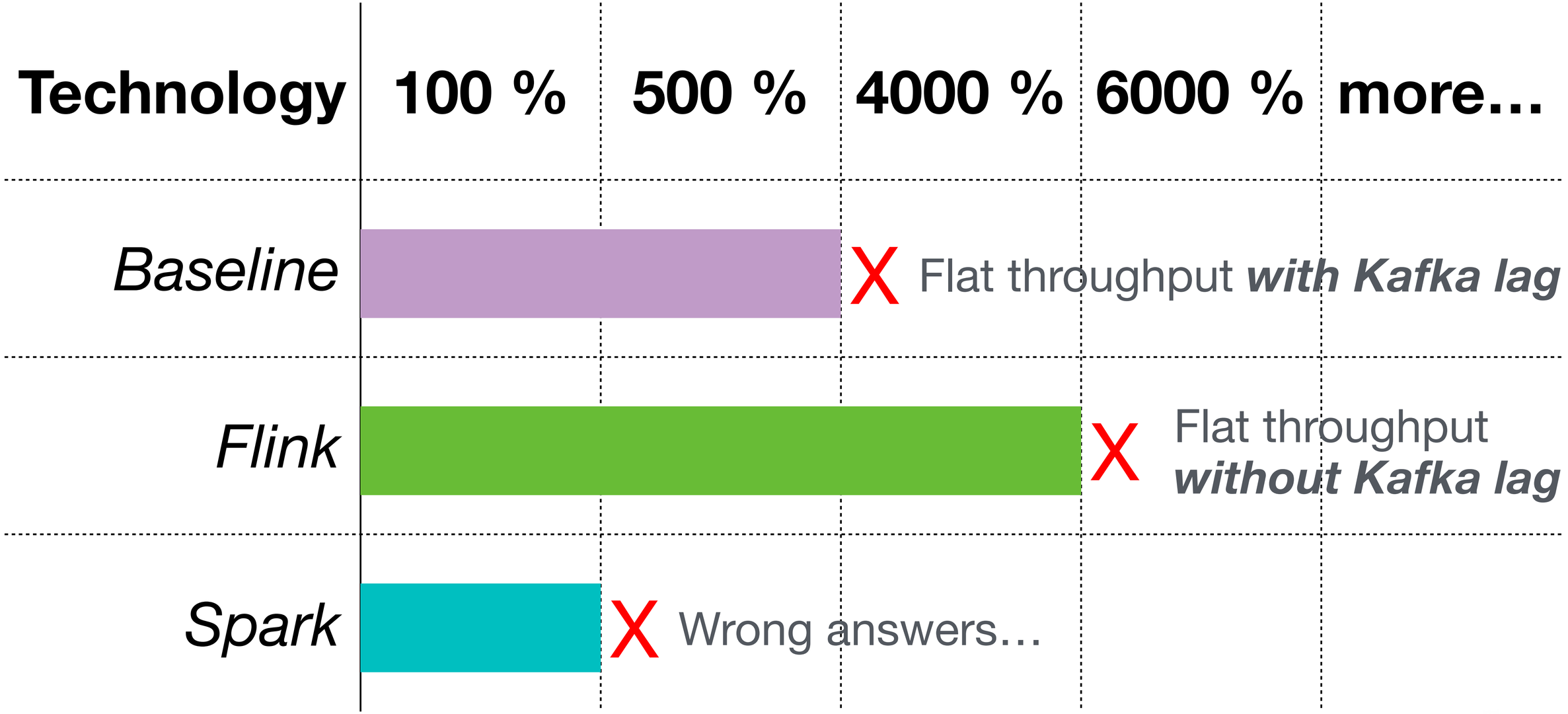
But the Spark Streaming solution generates WRONG results

- … because there is no Event Time windowing
- … leading to me abandoning Spark Streaming

Results



Results


Technology
100 %
500 %
4000 %
6000 %
more…
Baseline
X
Flat throughput with Kafka lag
Flink
X
Flat throughput
without Kafka lag
Spark
X
Wrong answers…

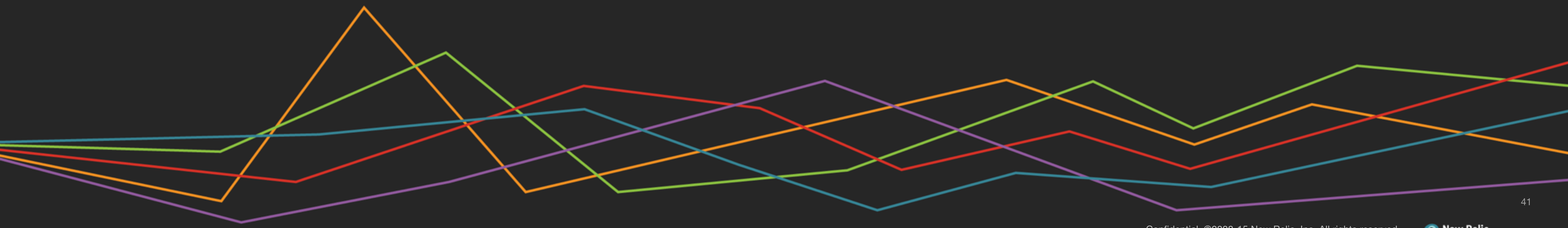
Opportunities to improve the experiment

- **MORE BANDWIDTH**
  - I don't know the limit of the Flink implementation

- **Key space domain expansion *[All]***
  - Scaling in rate domain only, with the same set of keys
    - This is *too easy* on the key-based systems ***[Flink, Spark]***
    - This may be hard on the baseline system as well

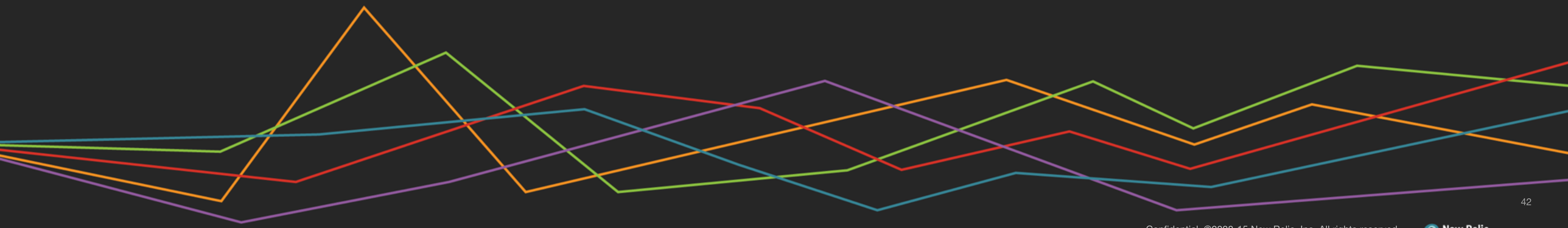
- **Inclusion of Database sinks *[Flink, Spark]***
  - Kafka sinks are still needed for downstream functions

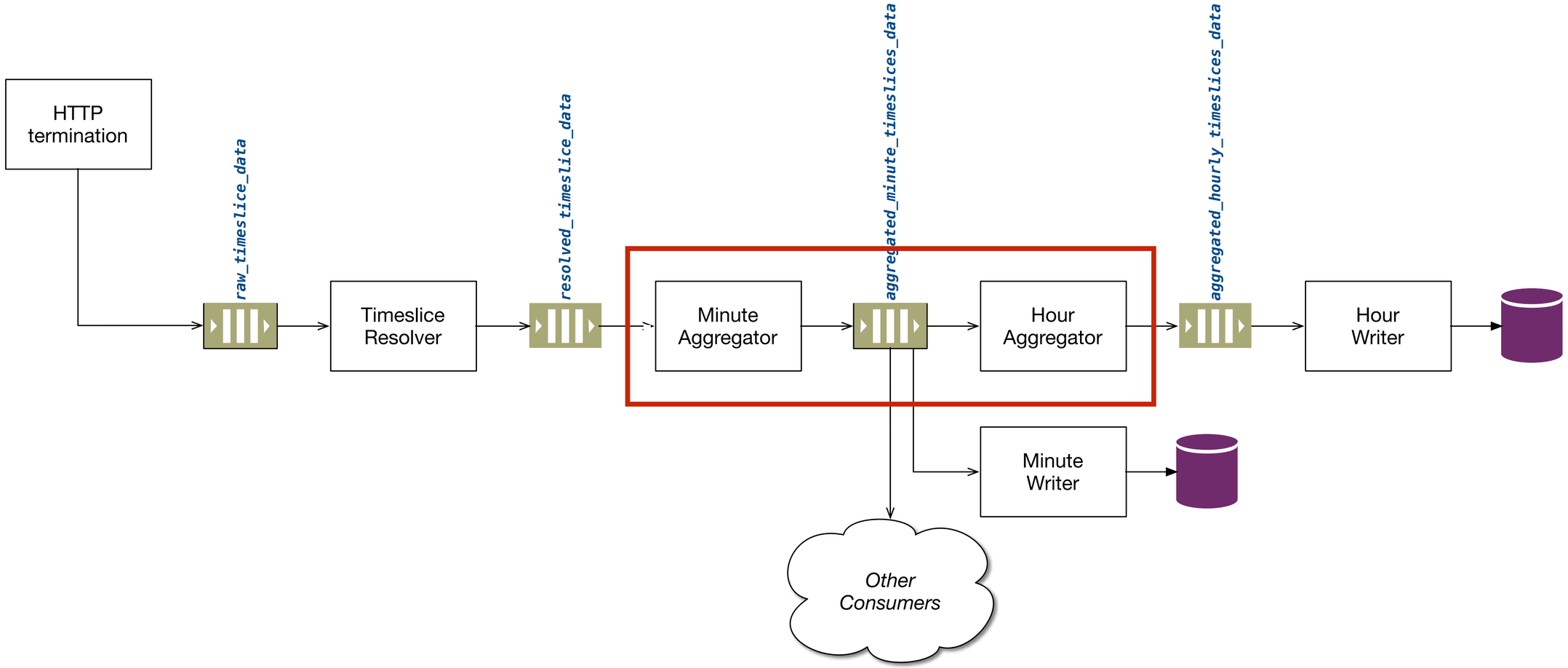
Thank you



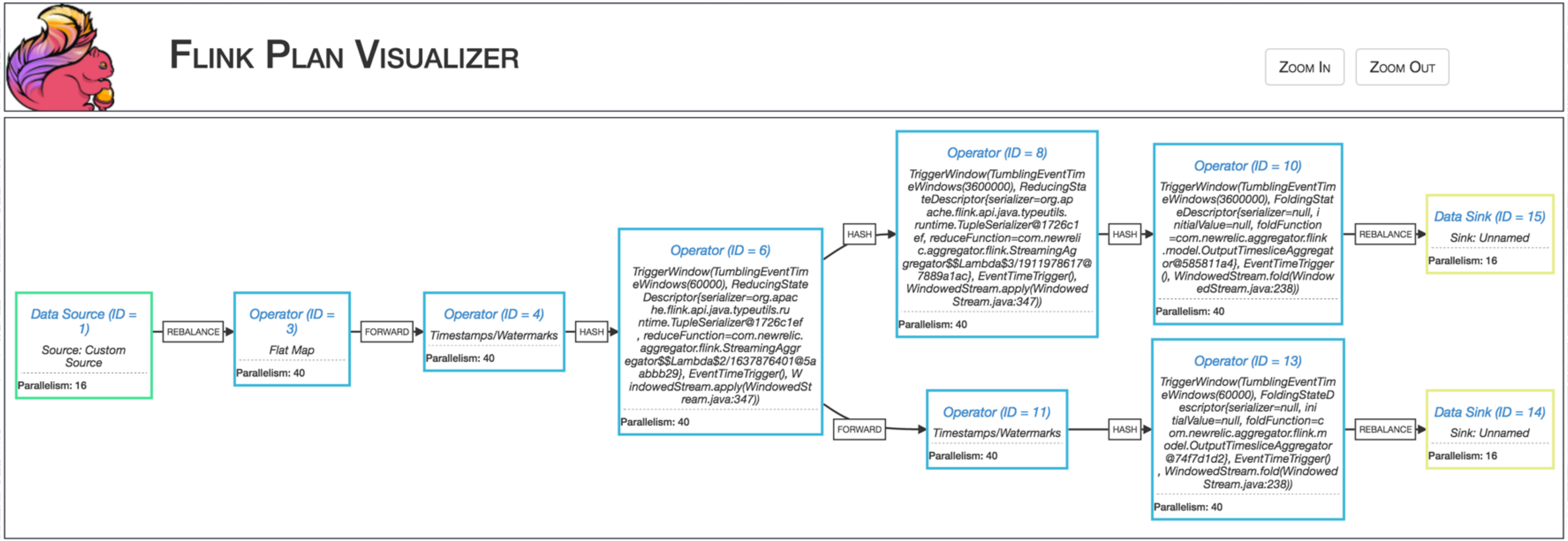
New Relic.

Extra credit

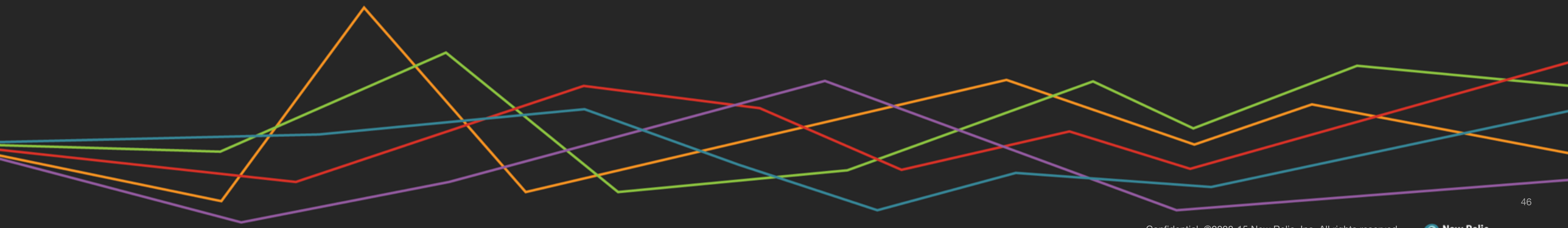




```
63
64      // aggregatedByMinuteStream contains events that are the aggregated timeslices by [MetricId,AgentId,AccountId]
65      DataStream<TimesliceTuple> aggregatedByMinuteStream = dataStream
66              .rebalance()
67              .flatMap(new TimesliceTupleMapper())
68              .assignTimestampsAndWatermarks(new TimesliceTupleTimestampExtractor())
69              .keyBy(TimesliceTuple.WIDE_METRIC_ID,TimesliceTuple.AGENT_ID, TimesliceTuple.ACCOUNT_ID)
70              .timeWindow(Time.seconds(60))
71              .reduce(TimesliceTuple::aggregateWith);
72
73      // packagedMinuteTimeslices are all the minute-aggregated timeslices for a given [AgentId,AccountId]
74      DataStream<OutputTimesliceData> packagedMinuteTimeslices = aggregatedByMinuteStream
75              .assignTimestampsAndWatermarks(new TimesliceTupleTimestampExtractor())
76              .keyBy(TimesliceTuple.AGENT_ID, TimesliceTuple.ACCOUNT_ID)
77              .timeWindow(Time.seconds(60))
78              .fold(null, new OutputTimesliceAggregator());
79
80      packagedMinuteTimeslices.addSink(new FlinkKafkaProducer08<OutputTimesliceData>(
81              AGGREGATED_MINUTE_TIMESLICE_DATA,
82              new OutputTimesliceDataSerializer(),
83              getKafkaProducerProperties(configuration))).setParallelism(16);
84
85      DataStream<TimesliceTuple> aggregatedByHourStream = aggregatedByMinuteStream
86              .keyBy(TimesliceTuple.WIDE_METRIC_ID, TimesliceTuple.AGENT_ID, TimesliceTuple.ACCOUNT_ID)
87              .timeWindow(Time.minutes(60))
88              .reduce(TimesliceTuple::aggregateWith);
89
90      DataStream<OutputTimesliceData> packagedHourTimeslices = aggregatedByHourStream
91              .keyBy(TimesliceTuple.ACCOUNT_ID, TimesliceTuple.AGENT_ID)
92              .timeWindow(Time.minutes(60))
93              .fold(null, new OutputTimesliceAggregator());
94
95      packagedHourTimeslices.addSink(new FlinkKafkaProducer08<OutputTimesliceData>(
96              AGGREGATED_HOUR_TIMESLICE_DATA,
97              new OutputTimesliceDataSerializer(),
98              getKafkaProducerProperties(configuration))).setParallelism(16);
99
```



Thank you



New Relic.