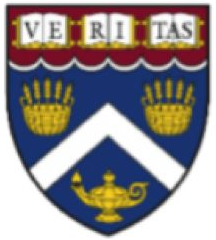
CSCI E-88 Principles Of Big Data Processing

Harvard University Extension, Fall 2019

Marina Popova



Lecture 14 Big Data Processing Use Cases

Agenda

- Boston - first snowstorm! :)
- Admin info
- BDP Use Case: Yottaa

Admin Info

Final Project:

- Final Project: Due: Sat, Dec 14, midnight, EST - **no Late Days, no extensions!**
- Final Projects Presentations: Dec 17
- **Teams of 2 people: identify yourself! Have to implement 4 tiers in your pipelines**

Next Week:

- More BDP use case reviews and presentations

Feedback!!

BigData Processing - Use Cases

Why review use cases?

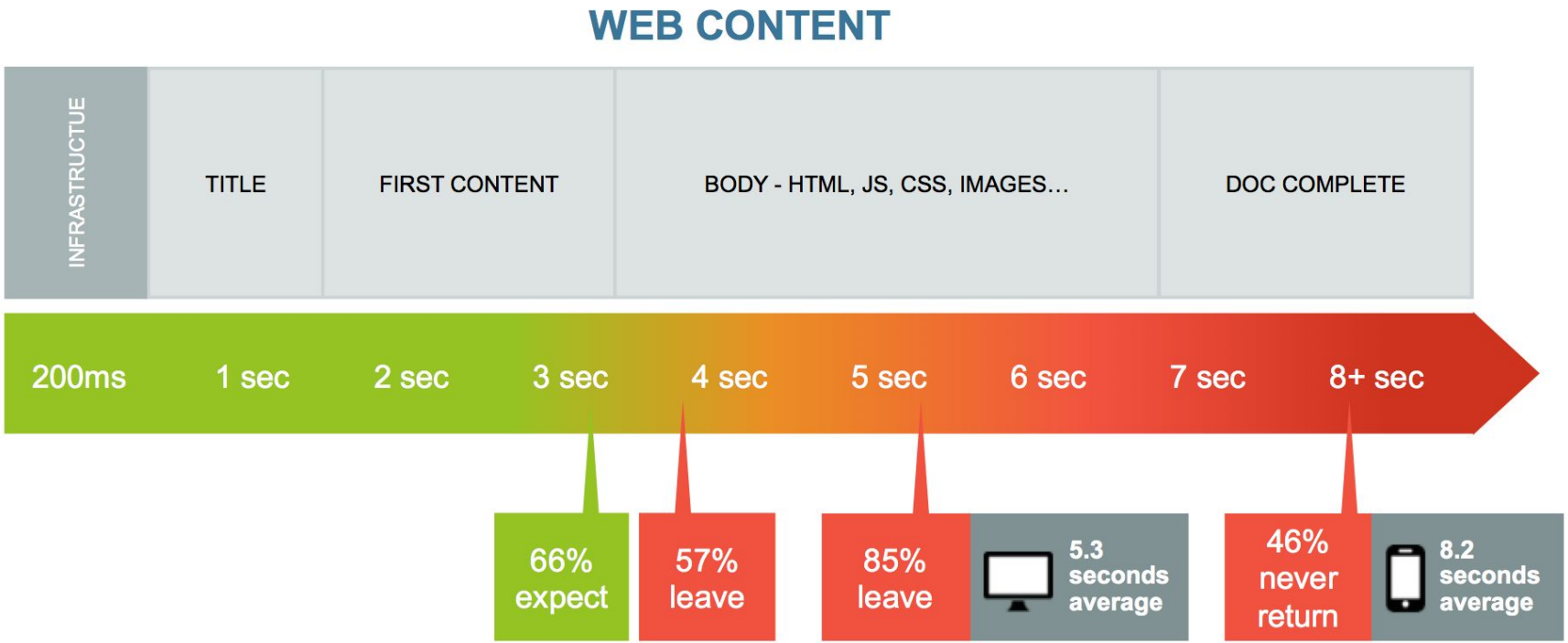
How will we review them?

- Architecture overview
- Identify processing tiers
- Identify common concerns/ problems they were trying to solve
- Identify solution approaches
- Identify what type of the system (Type 1 through 4) is implemented

BigData Processing Use Case: Yottaa

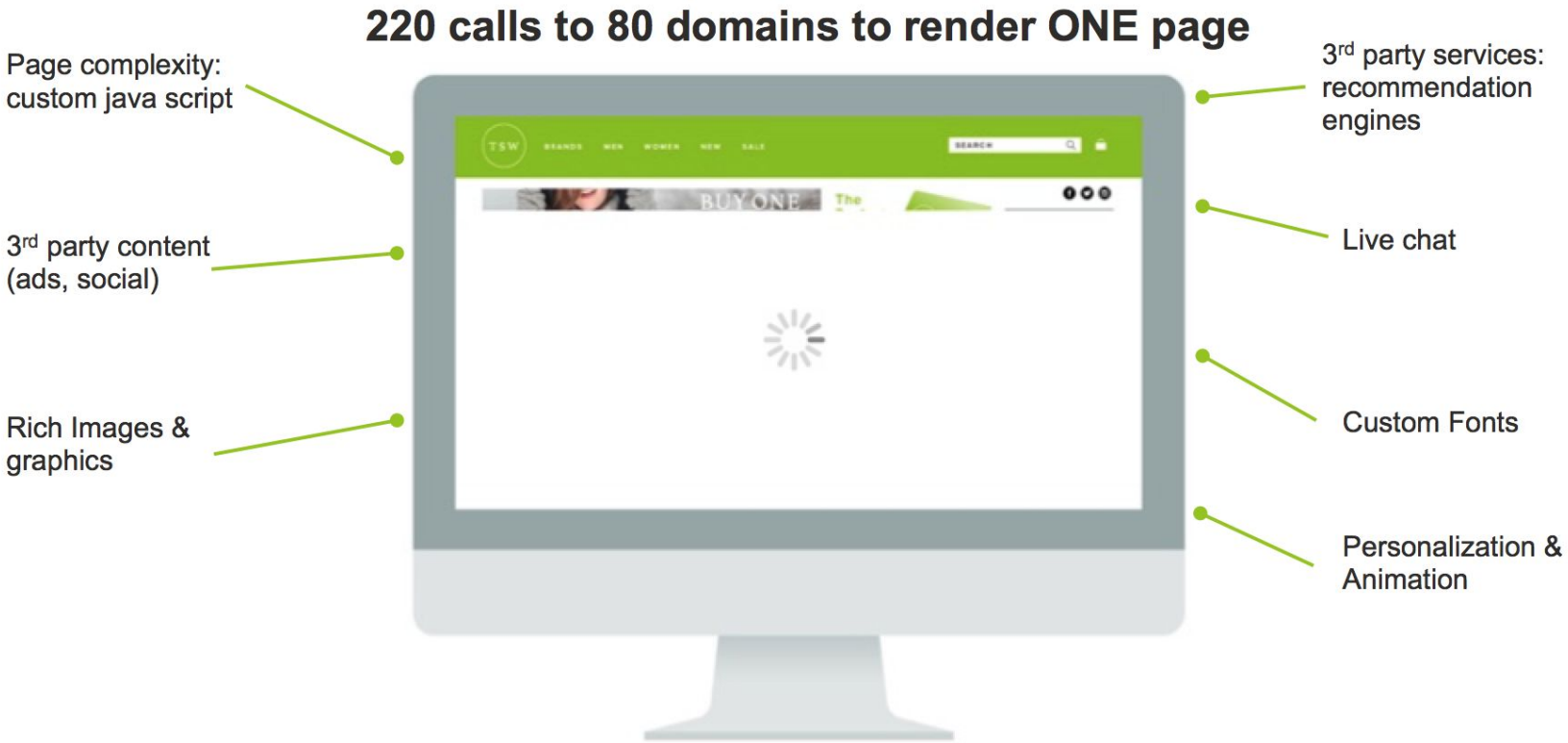
Problem Statement:

How Does a Typical Page Load? 8.117 secs for cb2.com

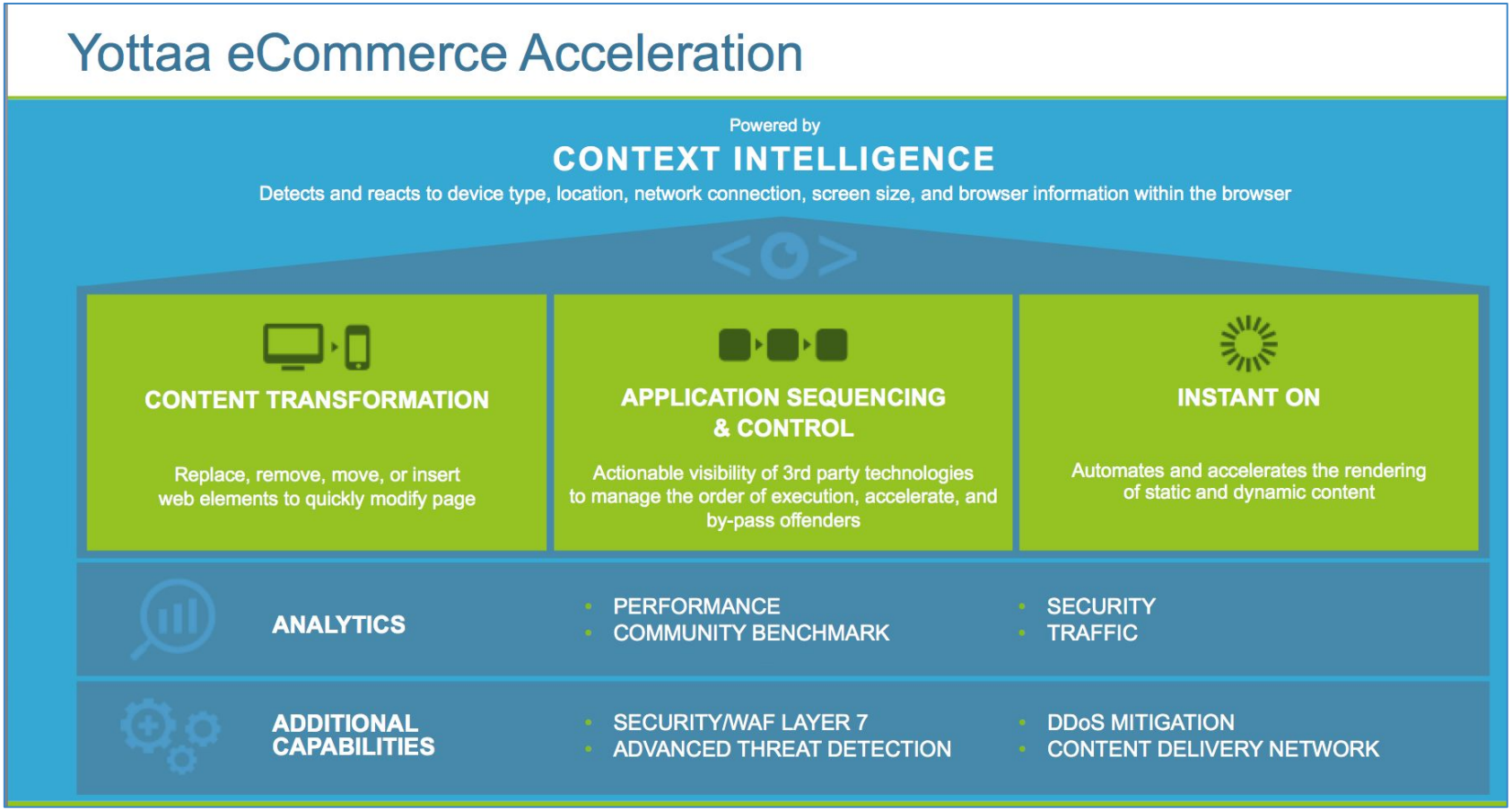


BigData Processing Use Case: Yottaa

Why are web pages load so slowly??



Yottaa - so what do we do?



@Marina Popova

Yottaa

Analytics Projects Goals:

- Customer and Internal visibility into:
  - General traffic patterns
  - Yottaa's Impact on their site
  - Opportunities for additional performance improvement
  - Insight into potential threats
- Needed to be
  - Real-time
  - Fast!
  - Scalable
  - Easy to use

BigData Processing Use Case: Yottaa

Two types of logs:

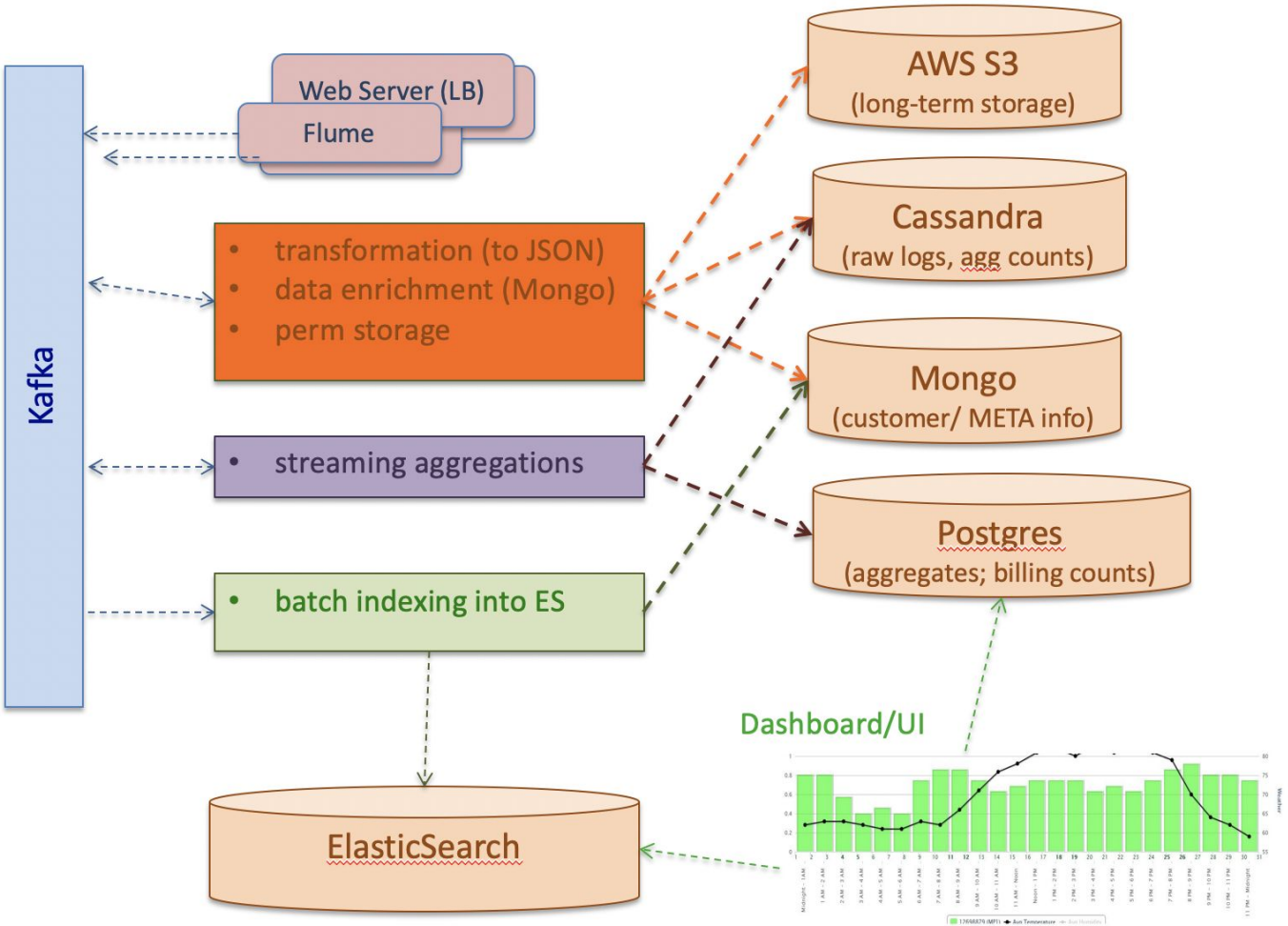
- Nginx (similar to Apache) event logs
  - *68.232.39.150 - [15/May/2017:23:01:50 +0000] "GET http://www.moosejaw.com/assetStore HTTP/1.1" 404 18591 ...*
- Browser generated events
  - DOM events
  - resource loading timings, etc.

BigData Processing Use Case: Yottaa

Main functionality/ pipelines:

- log collection from Flume into Kafka
- Nginx log storage into S3 - partitioned by customer and date/day/hour
- Nginx log indexing into ElasticSearch - for Real-Time analytics - last 7 days only
- Billing app:
  - Ongoing - from ElasticSearch
  - Long-term recovery: from S3
- Browser events: streaming aggregations by configurable dimensions
  - Storing hourly aggregates into Postgres
- Browser events: indexing of *some* events into ElasticSearch - last 24 hours only

Yottaa Log Processing High-Level Architecture



@Marina Popova

Yottaa

Main Questions to Answer:

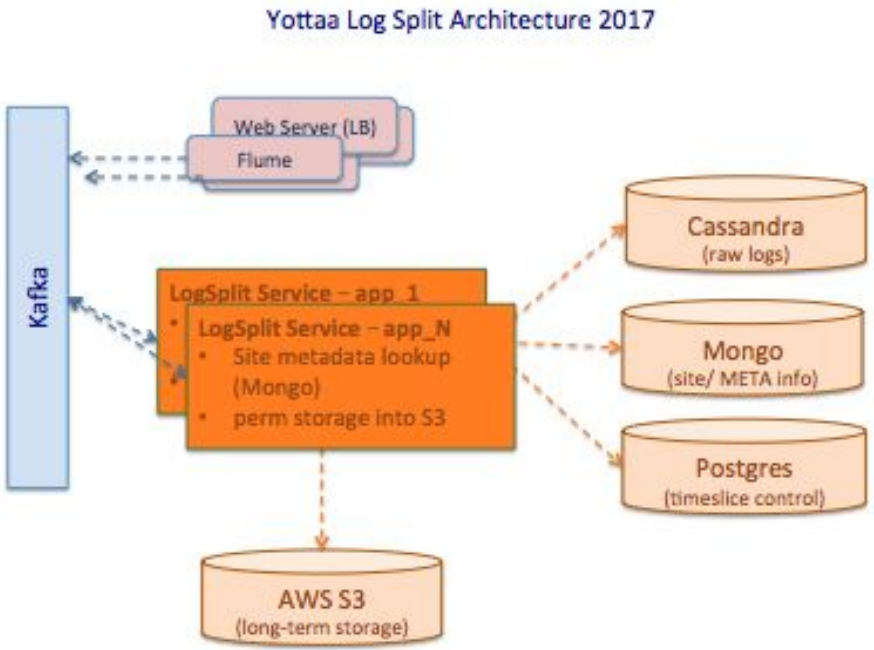
1. How are duplicate and out-of-order events handled ?
2. How is horizontal scalability achieved?
3. How is data consistency achieved?
    a. No data loss
    b. Correct aggregate metrics - idempotent computations

Log collection

- Flume agents sitting on each Nginx web server
- Flume source: 'tail -F'
    - Out of necessity - to not add any delays into Nginx request processing
- Flume sink: Kafka
- Flume channel: memory with disk spill-over

Log Storage into S3 - V1

- Each application is a Kafka Consumer with multiple threads
  - each thread is processing one or more Kafka partitions
  - each application is using the same Kafka consumer group_id
- events are stored into "timeslice" tables in Cassandra
- once timeslice (say 5-30 min) is done - a different process (threads) reads data from Cassandra , batches into 100K batches per customer and stores into S3



Log Storage into S3 - V1

68.232.39.150 - [04/Dec/2017:10:31:50 +0000] "GET http://www.moosejaw.com/assetStore HTTP/1.1" 404 18591 ...

S3 file names and dir structure:

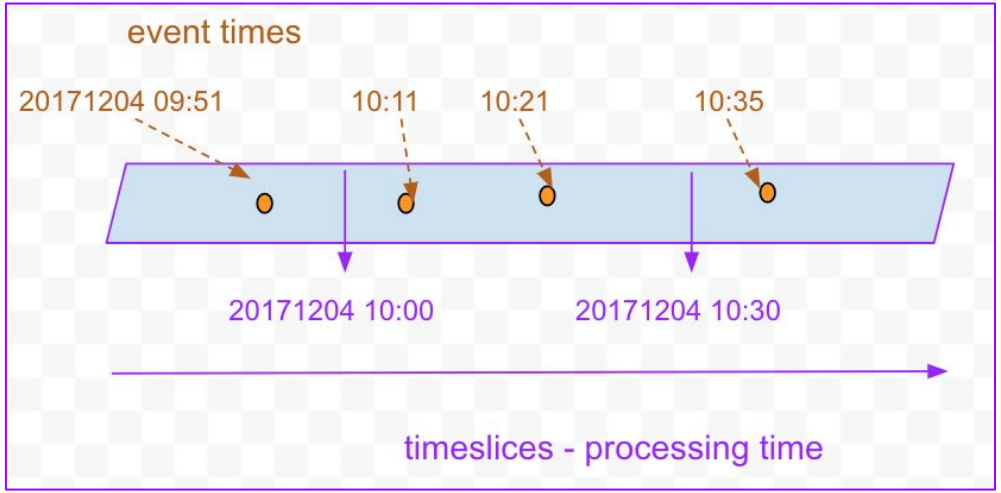
https://s3.amazonaws.com/<my_s3_bucket>

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1000_<app_1>_1.gz

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1000_<app_2>_1.gz

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1030_<app_1>_1.gz

to dedupe events per timeslice!



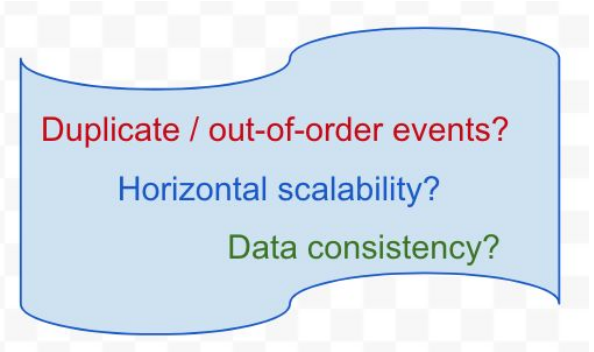
Cassandra: raw events:

```
TABLE logs_20171204_1000_<app_id> (
    timeslice timestamp,
    site_key text,
    event_uuid text,
    event_timestamp timestamp,
    raw_event text,
    PRIMARY KEY ((timeslice, site_key), event_uuid)
```

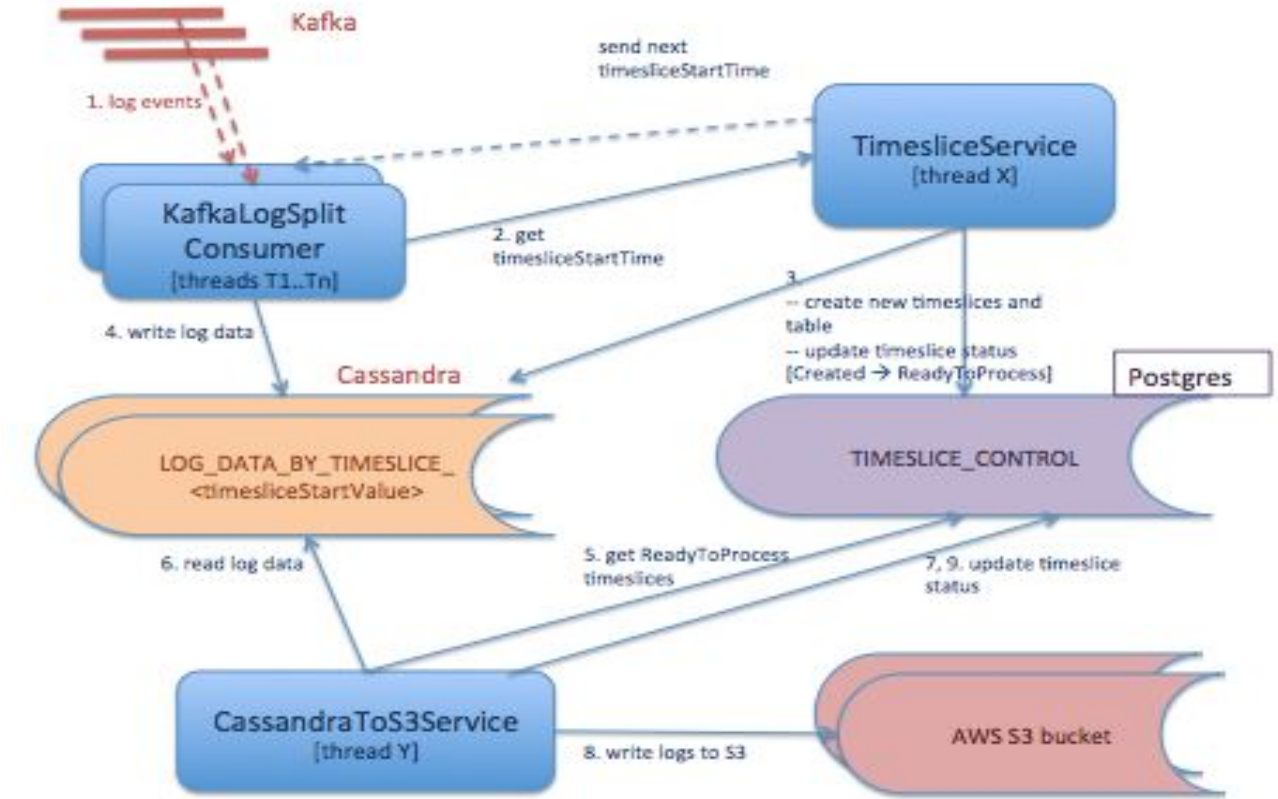
Log Storage into S3

Main design points:

- Can scale horizontally - all operations are idempotent:
  - Cassandra tables have 'timeslice' value as part of their name
  - S3 files have both 'timeslice' and 'batchId' values
- Each application instance uses its own Cassandra table
- S3 files are write-once only
  - Data consistency !



Yottaa:
S3 storage



Log Storage into S3 - V2

Scaling challenges: (after about 2 years of great performance):

- too high load on Cassandra - read/write latency was increasing with the increase of the event rates

Reason:

- tables with raw events are too often created/deleted - Cassandra is not designed to handle so many DDL changes efficiently

Solution:

- move to a single table for raw events
- preserve "timeslice/bucket" as part of the PK
- use separate keyspace for each application

Log Storage into S3 - V2

68.232.39.150 - [04/Dec/2017:10:31:50 +0000] "GET http://www.moosejaw.com/assetStore HTTP/1.1" 404 18591 ...

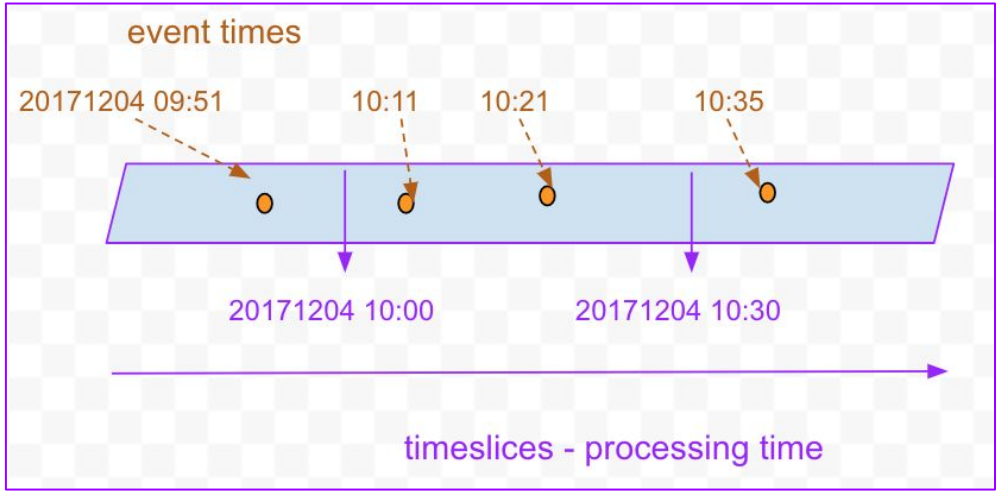
S3 file names and dir structure:

https://s3.amazonaws.com/<my_s3_bucket>

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1000_<app_1>_1.gz

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1000_<app_2>_1.gz

/<site_key>/2017/12/04/logs_20171204_1000_20171204_1030_<app_1>_1.gz

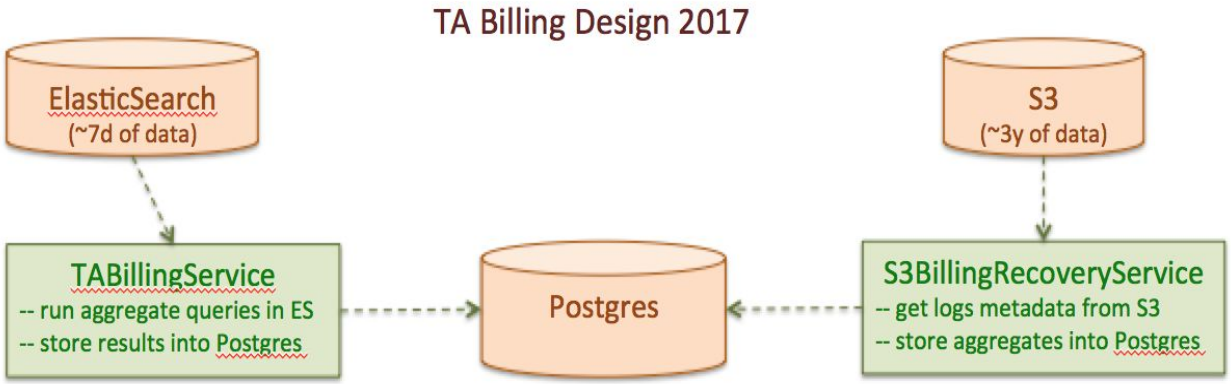


Cassandra raw events:
keyspace: raw_logs_<app_id>

```
TABLE raw_logs_<app_id>.raw_logs (
    site_key text,
    timeslice timestamp,
    event_uuid text,
    event_timestamp timestamp,
    raw_event text,
    PRIMARY KEY ((site_key, timeslice), event_uuid)
```

@Marina Popova

Billing app



- de-duped by event UUIDs - across sites/ days
- Java 8 streams - parallel processing by sites/S3 folders
- aggregates are never updated - only re-written

Duplicate / out-of-order events?

Horizontal scalability?

Data consistency?

S3 and Billing pipelines - architecture type?

what type of system is it?

??????

?

Type 1: Limited Real-time + Ad-hoc queries systems:
Systems that can answer any [ad-hoc] question based on the **limited raw data** (in-memory only) received during some small recent time window

Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:
systems that can answer pre-defined questions based on the **historical pre-aggregated data (metrics)** - not on all historical raw data - and on the limited latest real-time raw data

Type 3: Full Historical + No Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far - but no real-time up-to-the-second data

Type 4: Full Historical + Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far, including up-to-the-second data

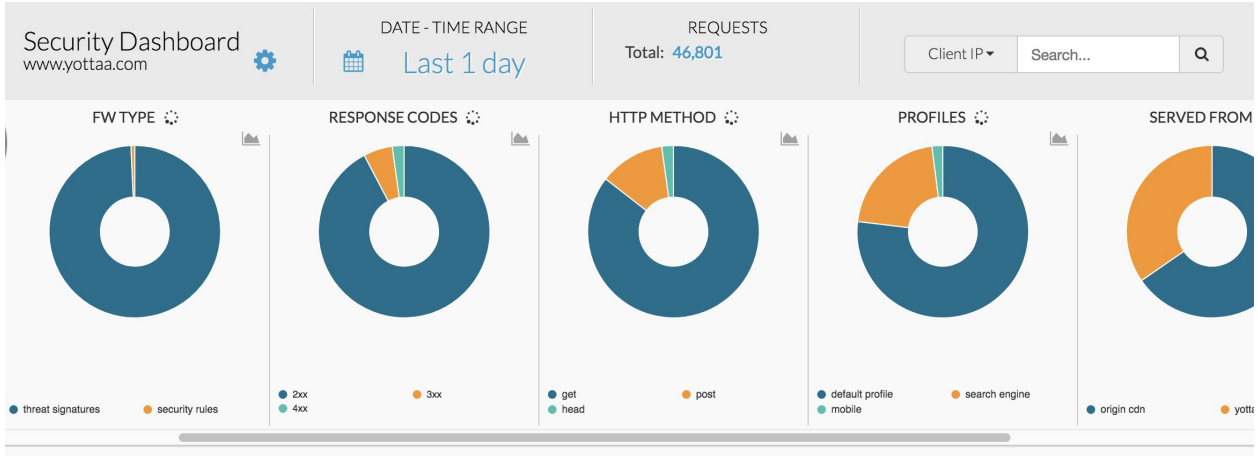
Answer: limited Type 3 System

@Marina Popova

Next Pipeline: Log Indexing and Real-Time Analytics

Goals:

- visualize and provide ad-hoc analysis for real-time traffic
- calculate and visualize continuous pre-defined queries - such as TopN

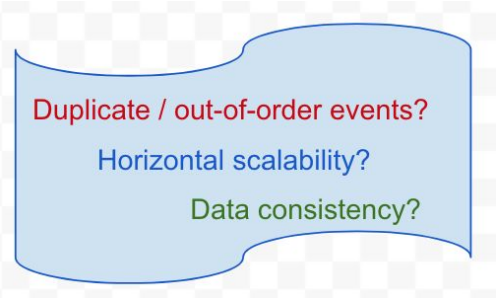


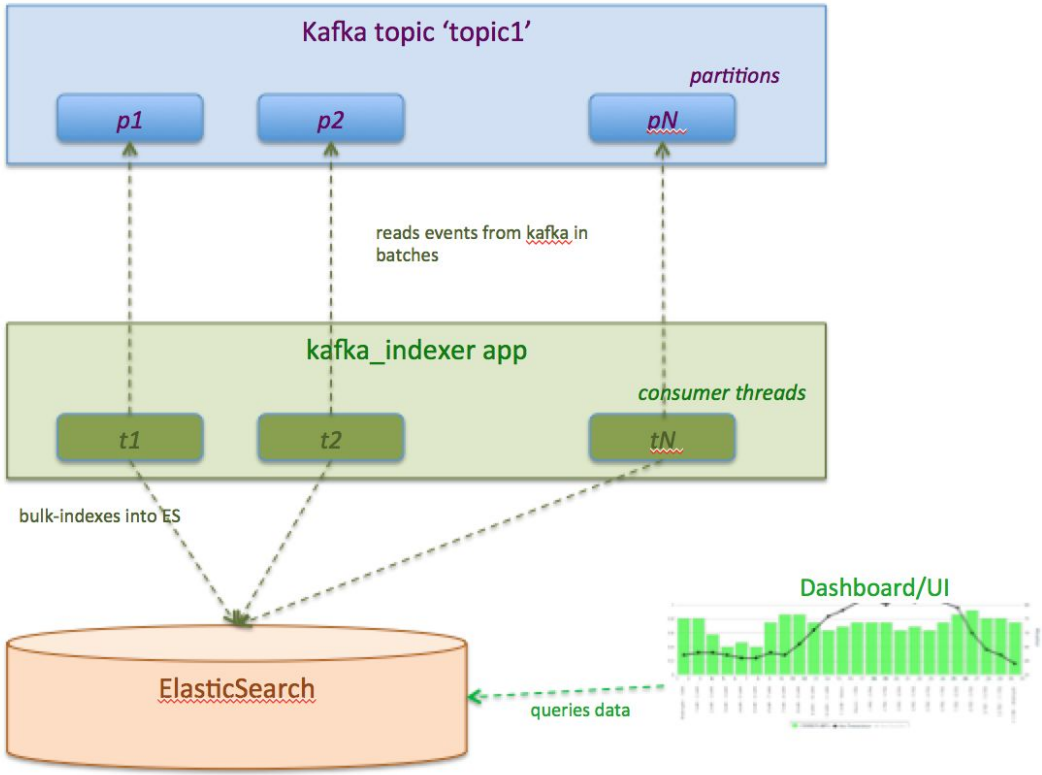
@Marina Popova

Log Indexing

Based on this Open Source project:
https://github.com/BigDataDevs/kafka-elasticsearch-consumer

Why?
-- embeddable
-- easily customizable
-- very light way
-- full control over offsets commits
-- exception handling





@Marina Popova

Log Indexing

Public Indexer project:

TravisCI --> Bintray -->
Build # xxx

kafka-elasticsearch-consumer-1.0.xxx.jar

Your project:

```
<repository>
    <id>bintray-bigdatadevs-bigdatadevs-repo</id>
    <name>bintray</name>
    <url>http://dl.bintray.com/bigdatadevs/bigdatadevs-repo</url>
</repository>
```

```
<dependency>
    <groupId>org.elasticsearch.kafka.indexer</groupId>
    <artifactId>kafka-elasticsearch-consumer</artifactId>
    <version>1.0.xxx</version>
</dependency>
```

```
<import resource="classpath*:spring/common.xml"/>
<context:component-scan base-package="org.elasticsearch.kafka.indexer.service"/>
<context:property-placeholder location="classpath:config/kafka-es-indexer.properties"
    ignore-resource-not-found="true" />

<!-- Yottaa's customizations -->
<bean id="messageHandler"
    class="com.yottaa.elasticsearch.kafka.indexer.VarnishMessageHandler"
    scope="prototype"/>
```

@Marina Popova

Log Indexing

Index Design

V1:

varnish_x5_56abbddddddddddddddd_20190518

[type]_[size]_[customerID]_[daydate]

1 index per day originally
1 index per customer (regular)
1 shared index for small customers per day
1 shared index for medium customers per day - but with larger number of shards

Routing for shared indexes:
-ES API: IndexRequestBuilder.setRouting(customerID)

Log Indexing

Index Design - Multi-tenancy & Templates

Customers have different traffic volumes → different requirements for index sizes & index sharding
How do we manage that?

MongoDB + custom MessageHandler + ES Templates

MongoDB

customerID	es_size
aaa111	x2
aaa222	x5

Yottaa MessageHandler

event: {customerID = “aaa111”)
→ index:
“varnish_x2_aaa111_<day>”

ES mappings:

```
"varnish_x2": {
  "order": 2,
  "template": "varnish_x2*",
  "settings": {
   "index": {
    "number_of_shards": "2"
   }
  }
```

@Marina Popova

Log Indexing

Index Design - Aliases

Aliases for shared indexes

alias	index
varnish_x0_4df23d040fcb15396000000a_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_4e95accc8c088e1eaf002835_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_4fac03bb4707e612cd01c73a_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_4fe361f24707e675d4001fb3_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_50577ae476111c7215066d3a_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_508a840b5b15f23ce1012db9_20191127am	varnish_xs_smallsites_20191127am
varnish_x0_509534ab4707e6246e00c3d3_20191127am	varnish_xs_smallsites_20191127am

Aliases for Kibana - to restrict searchable indexes to 2 days

alias	index
kibana_2_days	varnish_x10_578855e22bb0ac10350002d6_20191202am
kibana_2_days	varnish_x10_561559c52bb0ac72f500050e_20191202pm
kibana_2_days	varnish_x5_557ae6510b5344747a000d5b_20191202am
kibana_2_days	varnish_x5_56e84016312e58244c0009e6_20191202pm
kibana_2_days	varnish_x10_58542937312e580e4e000016_20191204pm
kibana_2_days	varnish_x5_570cf67b312e58262e00028d_20191202am
kibana_2_days	varnish_x10_58542937312e580e4e000016_20191203am

ES cluster evolution

First version:

- 3 node cluster: AWS c3.8xlarge, 32 cores, 60Gb RAM
- Marvel (then monitoring/dev tools) on all servers
- 1 T per node

Issues:

- cannot take one node down - cluster in Red state
- cluster re-balancing takes forever
- indexing is failing during the rebalancing
- monitoring is not available when cluster is in a red state
- large Kibana queries can affect the cluster performance drastically
- frequent master re-election
- segments merge affects indexing speed and over-loads ES cluster

ES cluster evolution

V2:

varnish_x5_56abbddddddddddddd_20190518AM [PM]

[type]_[size]_[customerID]_[daydate]AM [PM]

Lessons Learned and Fixes:

- Index design: V2:
  - move to 2 indices per day: AM and PM
  - this decreased impact of segments merge
- Index creation
  - Use a scheduled job instead of automatic creation
  - Indexes has to be pre-created for next day, with delay between creation

- Index deletion - to remove indexes older than 7 days
  - Spread deletion among wide period of time
  - Don’t delete data during index creation

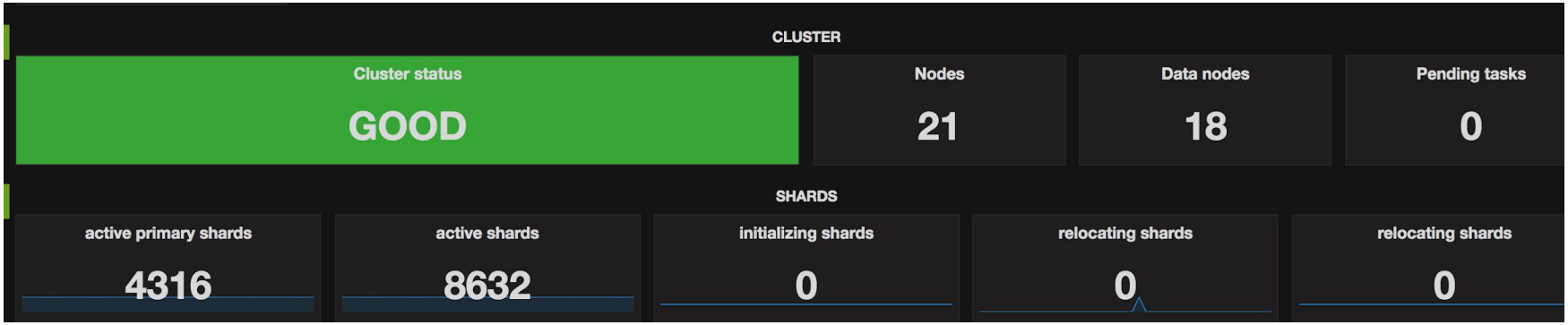
ES cluster evolution

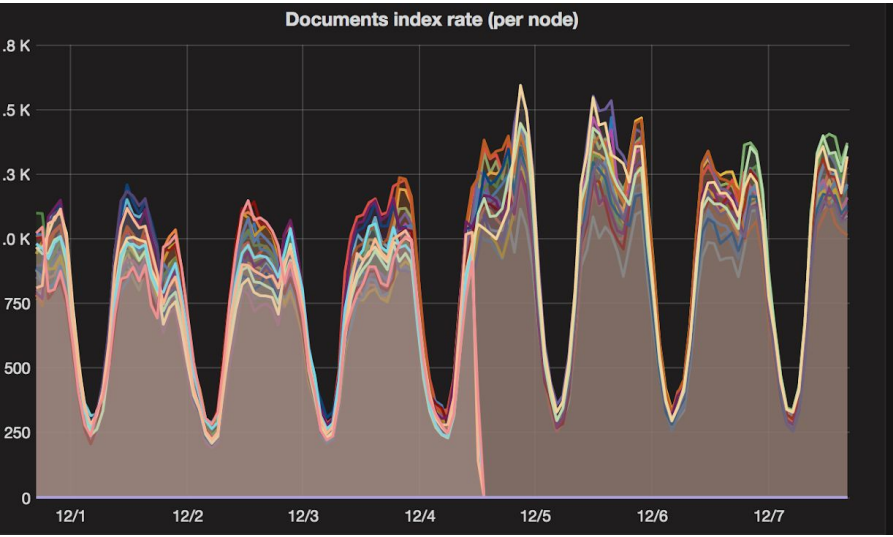
Current version:

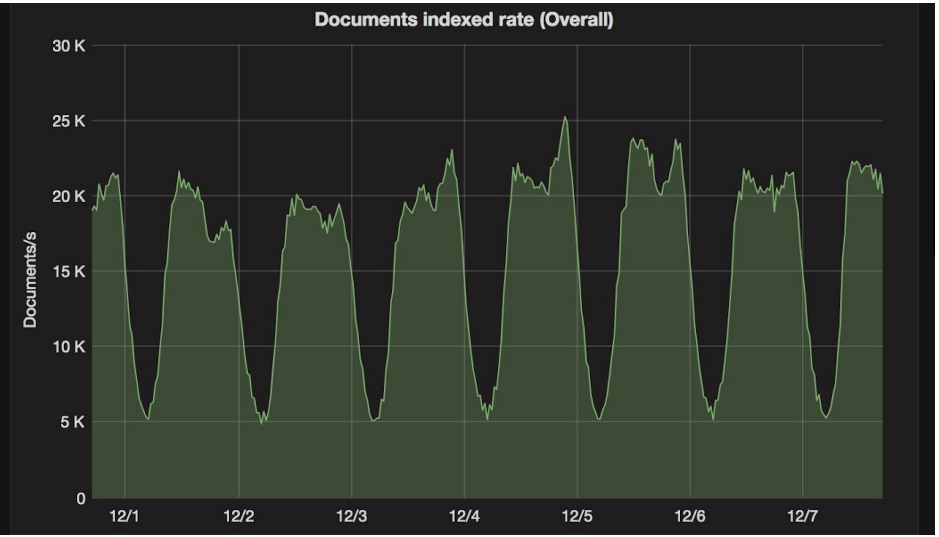
- 3 master nodes (m4.xlarge, 4cores, 16Gb)
- 18 data nodes (m4.2xlarge, 8cores, 32Gb) (in AWS) or 48 data nodes on premises (CentOS/Xen based)
  - Data nodes vs Master nodes
- 3-node Monitoring cluster
  - Doesn't affect performance of main cluster
  - If the main cluster is in red state, monitoring cluster still works
- Prometheus/Graphana for ES Monitoring

Scalability is the most important feature here!
During the Holidays - able to handle 3x traffic increase with linear horizontal scalability

ES Monitoring with Prometheus/Graphana

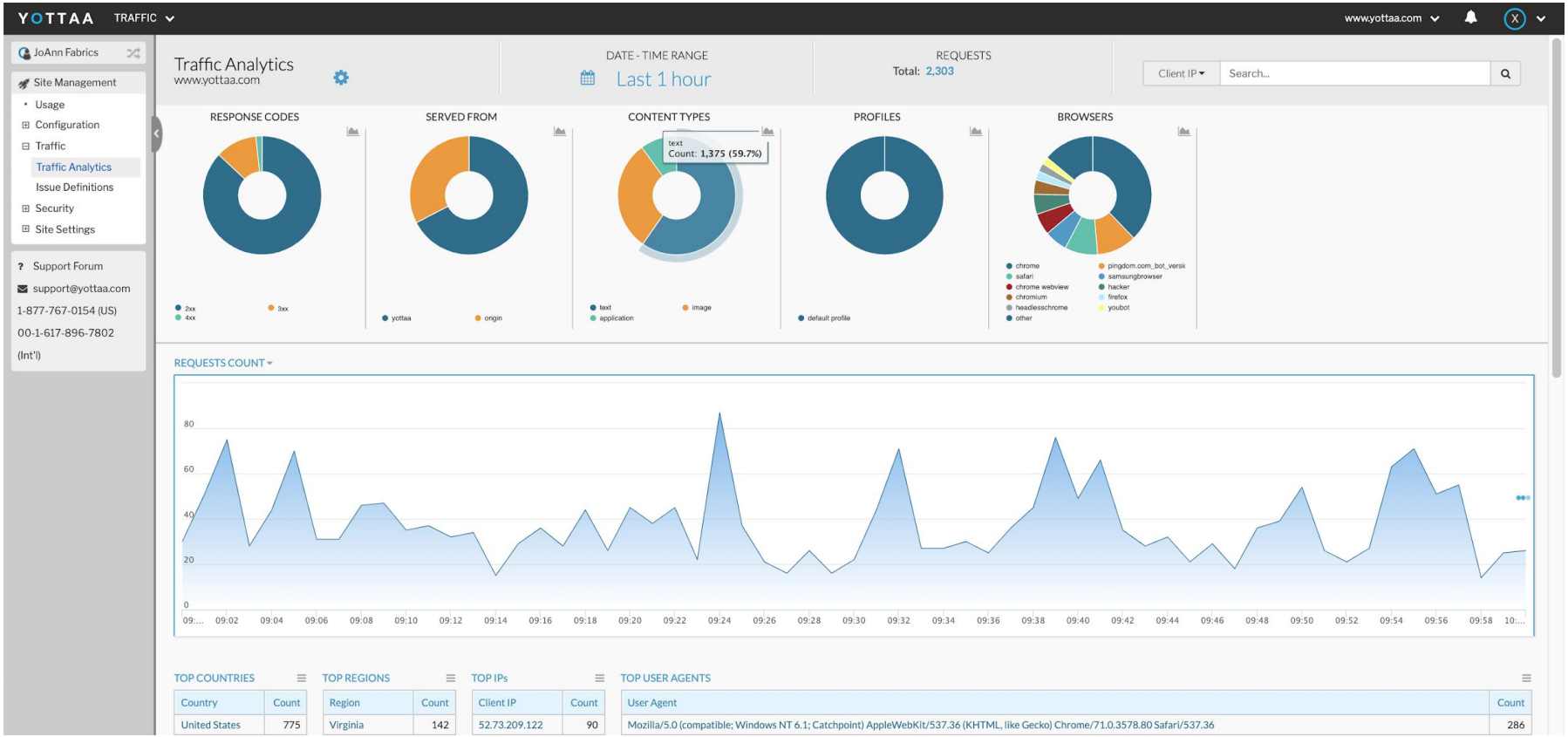






@Marina Popova

Log Indexing and Real-Time Analytics - Results



@Marina Popova

Log Indexing and Real-Time Analytics: TopN queries

REQUESTS COUNT

TOP FW RULES

FW Rule	Count
Threat Signature WAF	139
Redirect /assess to contact sales form	1

TOP COUNTRIES

Country	Count
United States	31,943
Hong Kong	2,300
China	1,659
Germany	1,362

TOP REGIONS

Region	Count
California	14,646
Colorado	6,896
Hong Kong (SAR)	2,300
Virginia	2,099

TOP IPs

Client IP	Count
130.211.138.158	2,747
204.2.133.210	2,223
204.2.131.110	2,220
64.71.161.22	2,217

@Marina Popova

Real-time indexing pipeline - architecture type?

what type of system is it?

??????

Type 1: Limited Real-time + Ad-hoc queries systems:
Systems that can answer any [ad-hoc] question based on the **limited raw data** (in-memory only) received during some small recent time window

Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:
systems that can answer pre-defined questions based on the **historical pre-aggregated data (metrics)** - not on all historical raw data - and on the limited latest real-time raw data

Type 3: Full Historical + No Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far - but no real-time up-to-the-second data

Type 4: Full Historical + Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far, including up-to-the-second data

Answer: Glorified Type 1 System

@Marina Popova

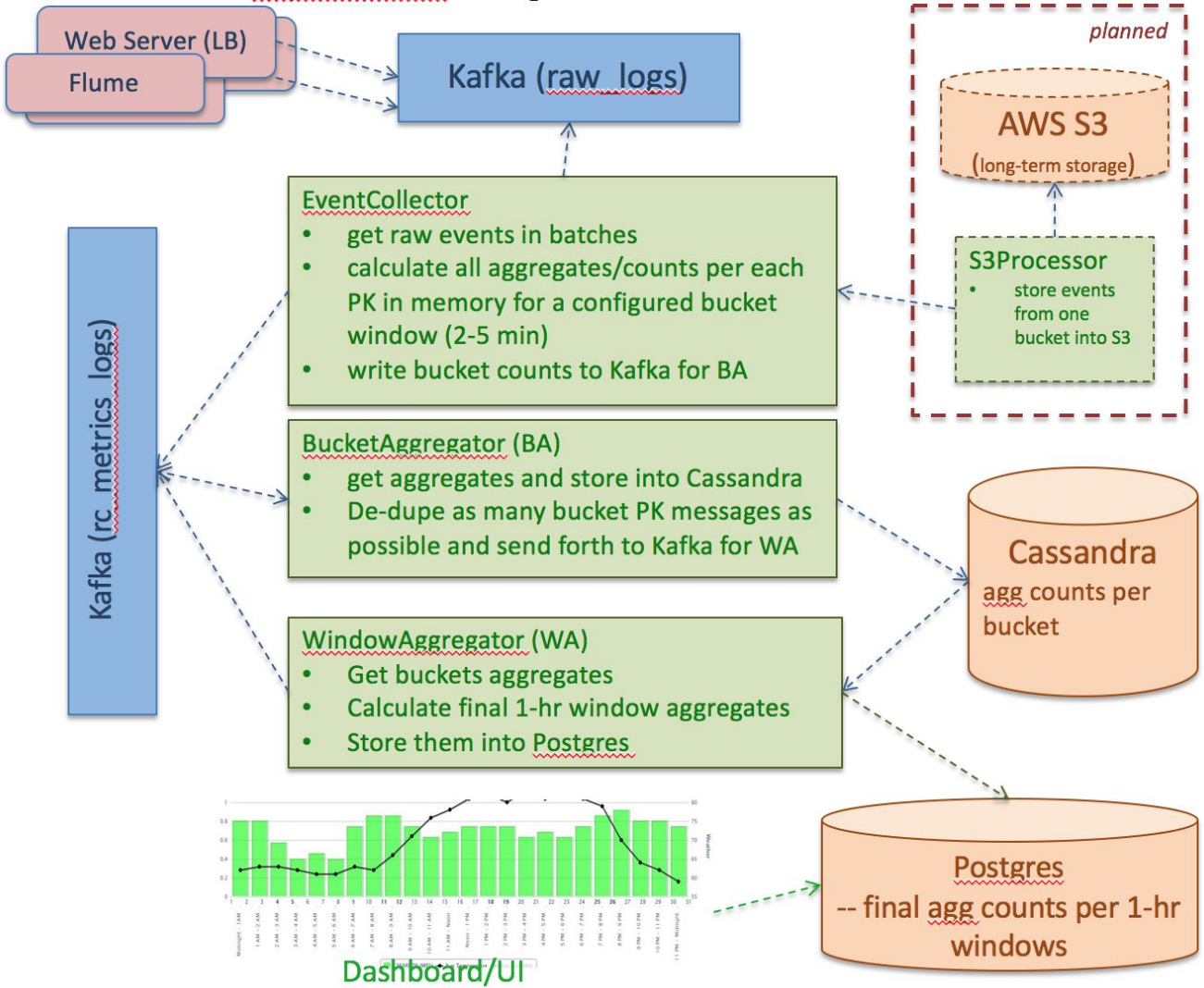
Next: aggregation pipelines

Two pipelines:

- persistent historical aggregates
- volatile tmp aggregates for Anomaly Detection

@Marina Popova

Streaming Metrics Aggregation



@Marina Popova

Yottaa Rapid: streaming aggregates

How do we define what to aggregate and by which dimensions/ keys ??

Rapid Counter Definitions

Example: counter definition for resource_timings (simplified)

rc_counter_pk:
what we are aggregating **by**:
dimensions + window:

```
{
  "siteKey" : "site1",
  "eventWindow" :  20190604_1000
  "ua_browser" : "chrome",
  "ua_device" : "tablet",
  "opt_state" : "active",
  ....
}
```

@Marina Popova

rc_metrics: what we are aggregating:

```
"counterMetrics":{
  "event_count": {
    "metricValue":124,
    "aggregationType":"INCREMENT"
  },
  "connect_time": {
    "metricValue":0.8,
    "aggregationType":"AVG"
  },
  "first_byte_time": {
    "metricValue":15.9,
    "aggregationType":"AVG"
  },
  ...
```


```
{
  "pgTableName":"rc_metrics_schema.agg_qoe_resource_timing",
  "dimensionRootElement":"payload.qoe.resourceTiming",
  "rcDimensions":[
    {
      "jsonElementName":"payload.parsedUserAgent.browser",
      "pgColumnName":"ua_browser",
      "pgColumnType":"VARCHAR",
      "caseSensitive":false
    },
    {
      "jsonElementName":"payload.parsedUserAgent.deviceType",
      "pgColumnName":"ua_device_type",
      "pgColumnType":"VARCHAR",
      "caseSensitive":false
    },
    {
      "jsonElementName":"payload.qoe.commonDimensions.optState",
      "pgColumnName":"opt_state",
      "pgColumnType":"VARCHAR",
      "caseSensitive":false
    },
  ],
  "rcMetrics":[
    {
      "pgColumnType":"BIGINT",
      "decimalPrecision":0,
      "jsonElementName":null,
      "pgColumnName":"event_count",
      "aggregationType":"INCREMENT"
    },
    {
      "pgColumnType":"REAL",
      "decimalPrecision":3,
      "jsonElementName":"payload.qoe.resourceTiming.connectTime",
      "pgColumnName":"connect_time",
      "aggregationType":"AVG"
    },
    {
      "pgColumnType":"REAL",
      "decimalPrecision":3,
      "jsonElementName":"payload.qoe.resourceTiming.firstByteTime",
      "pgColumnName":"first_byte_time",
      "aggregationType":"AVG"
    },
  ]
```

Rapid streaming aggregates

How do we keep windowed aggregated state??
Cassandra:

example TABLE: agg_metrics_**20190604**

- rc_counter_md5 - used as the single PK
- timeslice and app_id - enable idempotency of operations
- timeslice - processing time
- event_window - event time

Cassandra: timeslice aggregates

```
TABLE agg_metrics_20190604 (
    rc_counter_md5 text,
    timeslice timestamp,
    application_id text,
    event_window timestamp,
    rc_counter_pk text,
    rc_metrics text,
    site_key text,
    PRIMARY KEY (rc_counter_md5, timeslice, application_id)
)
```

@Marina Popova

rc_counter_md5	timeslice	app_id	event_window	rc_counter_pk	rc_metrics	site_key
56a8dc6697cf97d8f8b...	2019-06-04 10:00:00	app_1	20190604_1000	{....}	{....}	site1
56a8dc6697cf97d8f8b...	2019-06-04 10:00:00	app_2	20190604_1000	{....}	{....}	site1
56a8dc6697cf97d8f8b...	2019-06-04 10:30:00	app_1	20190604_1000	{....}	{....}	site1
56a8dc6697cf97d8f8b...	2019-06-04 10:30:00	app_2	20190604_1000	{....}	{....}	site1
aaaaaaaaaa7cf97d8f8b...	2019-06-04 10:00:00	app_1	20190604_1000	{....}	{....}	site1
aaaaaaaaaa7cf97d8f8b...	2019-06-04 10:00:00	app_2	20190604_1000	{....}	{....}	site1
...						

Streaming Metrics Aggregation

Postgres: final 1-hr metrics: no timeslice, no application_id anymore! Why??

```
TABLE agg_resource_timing (
        time_window ,
        site_key ,
        ua_browser ,
        ua_device ,
        opt_state ,
        event_count ,
        connect_time ,
        first_byte_time ,
        ....
PRIMARY KEY (
        time_window,
        site_key,
        ua_browser,
        ua_device,
        opt_state ...)
)
```

time_window	site_key	ua_browser	ua_device	opt_state	event_count	connect_time	first_time_byte
20190604_1000	site1	chrome	mobile	active	124	3.5	120.45
20190604_1000	site1	chrome	tablet	active	12	7.5	1111.45
20190604_1100	site1	chrome	tablet	none	12	7.5	1111.45
...							

@Marina Popova

Streaming Metrics Aggregation

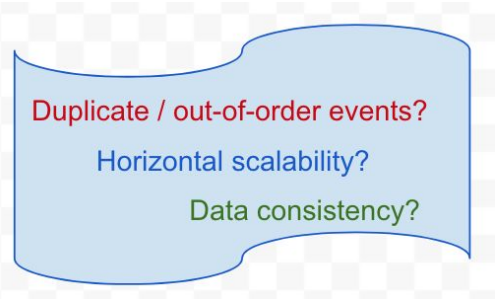
Critical Design Decisions:

- all Kafka consumer threads count their portion of events for the *same* PKs
  - they calculate aggregated metrics per timeslice per app_id
  - all threads use ***shared in-memory*** storage for the counters
  - when batch of events is successfully processed - send message for each unique PK to BA
- Kafka consumers manage offset commitment explicitly, using:
  - consumer.commitAsync(partitionOffsetMap) Kafka APIs
  - if something fails - offsets are NOT saved in Kafka

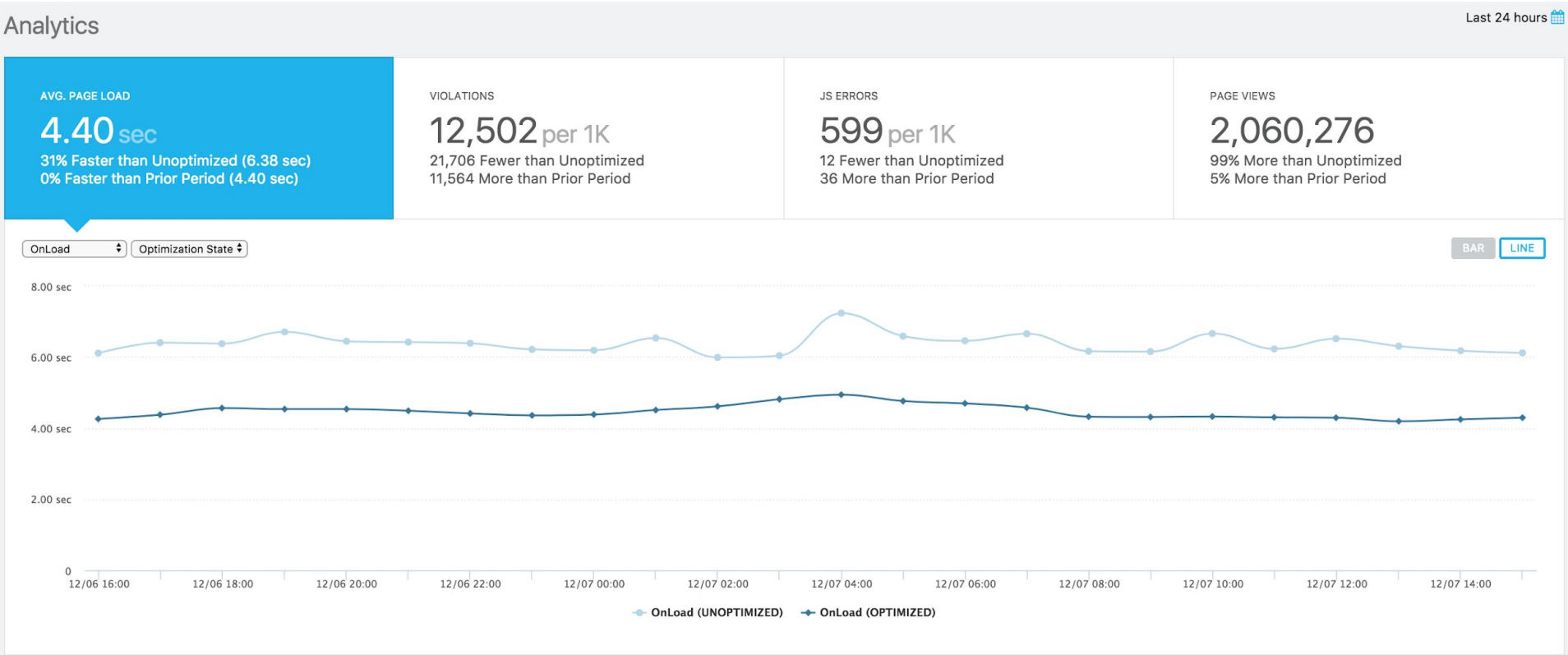
- BucketAggregator:
  - stores each counter key/value into Cassandra
  - send message for each unique PK to WindowAggregator
- WindowAggregator:
  - aggregates metrics across all timeslices and app_IDs for one hour

Nothing is ever updated - only fully re-counted - true idempotent operations !

What core principles does it remind you of??

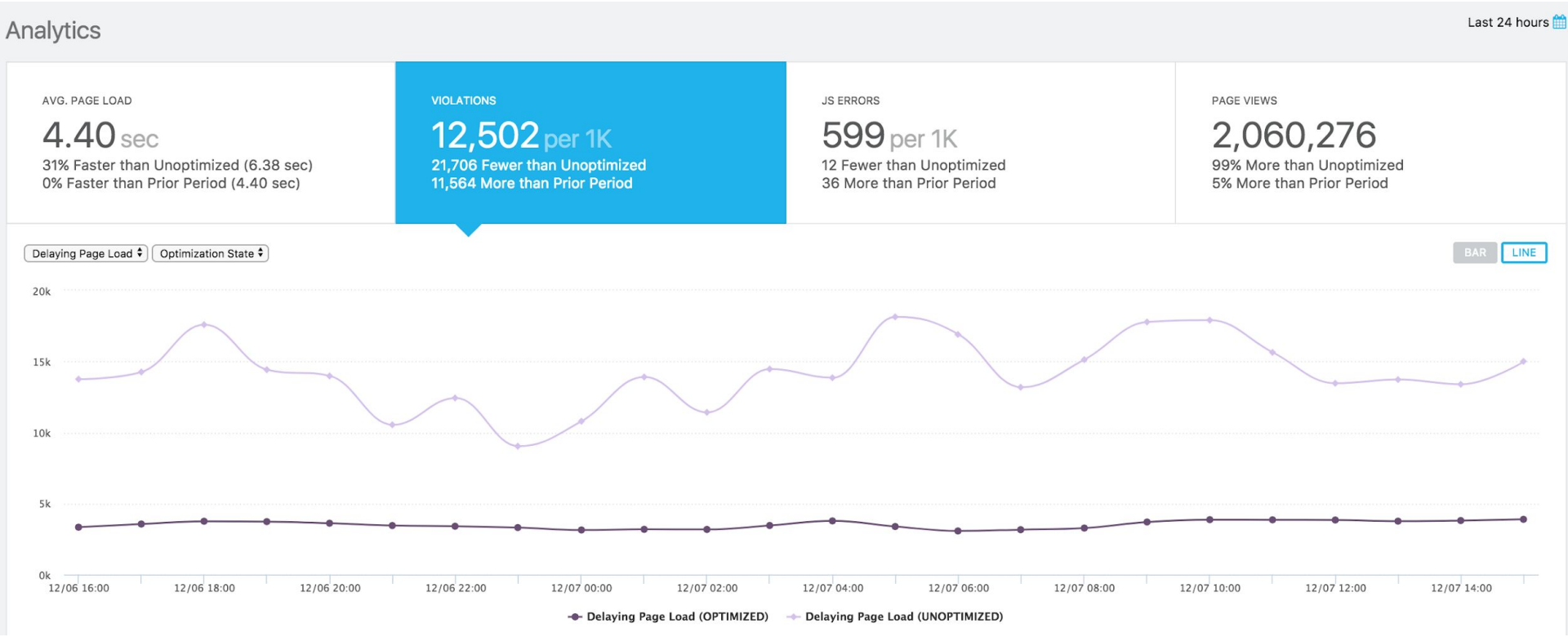


Streaming Metrics Aggregation - Results



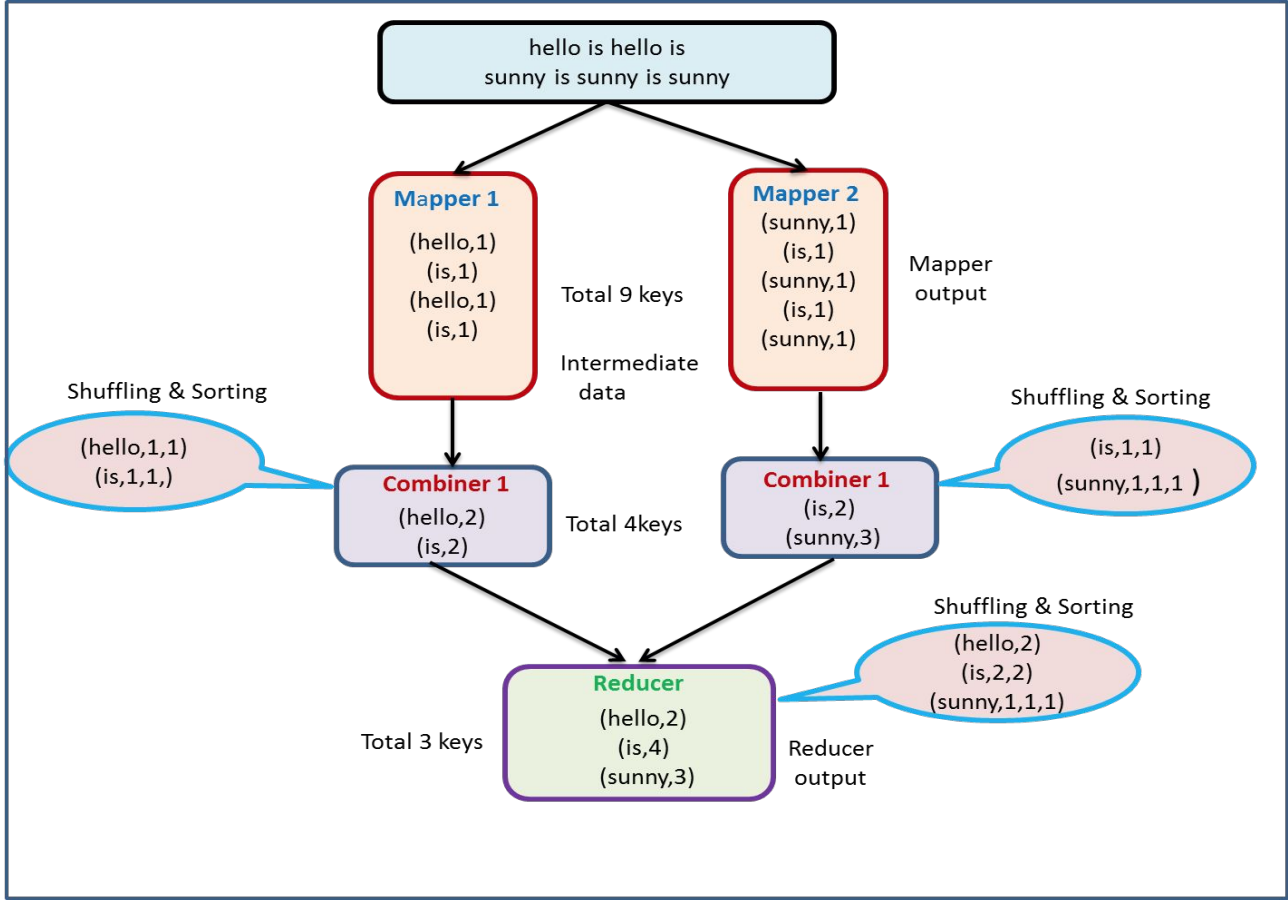
@Marina Popova

Streaming Metrics Aggregation - Results



@Marina Popova

Recap: MR Again!



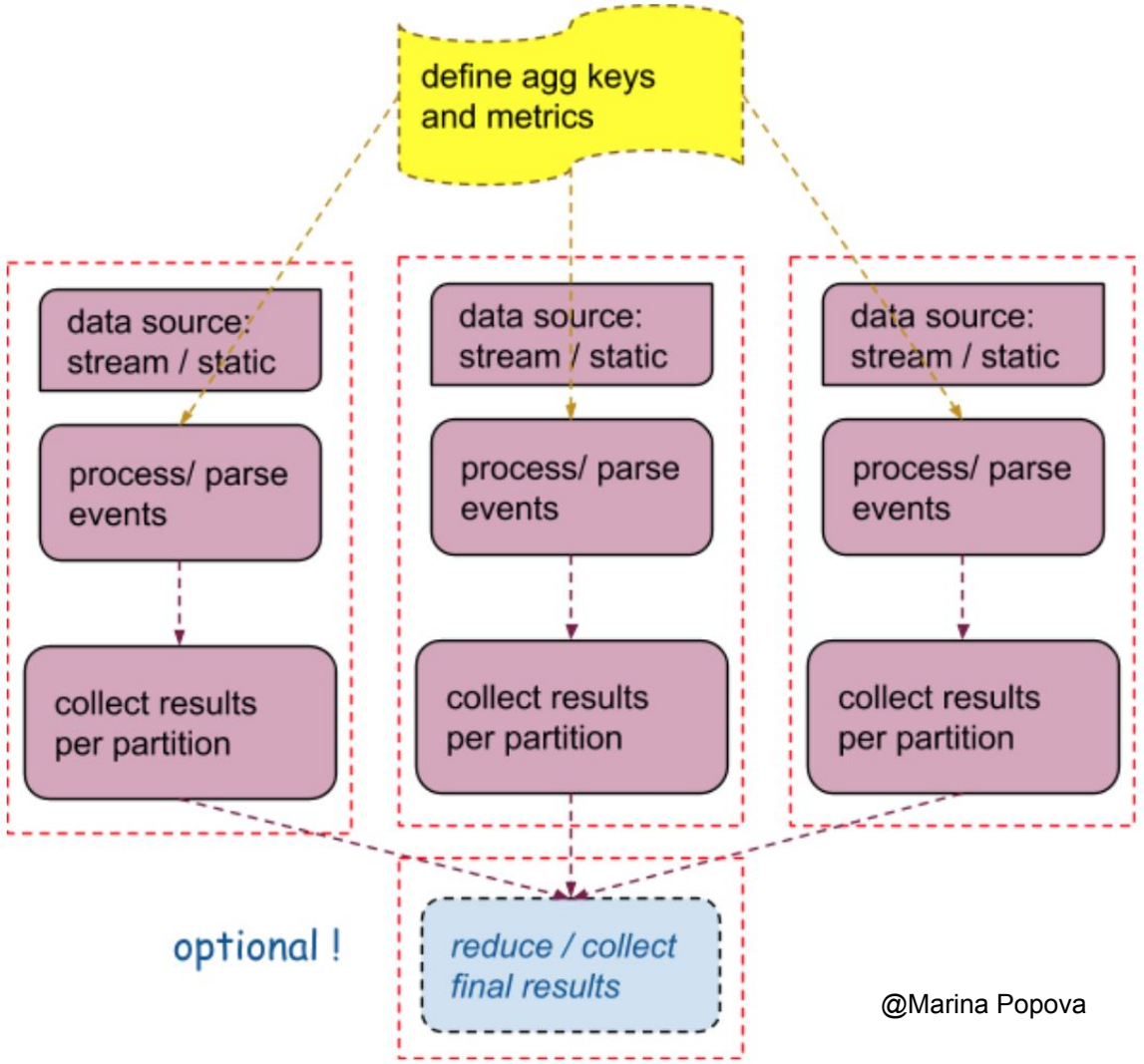
References: http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/

@Marina Popova

Stateless operations

- filter
- transform
- select ...
- uniques (approx - HLL)

Processing topology is a bit different for Stateless vs Stateful operations !

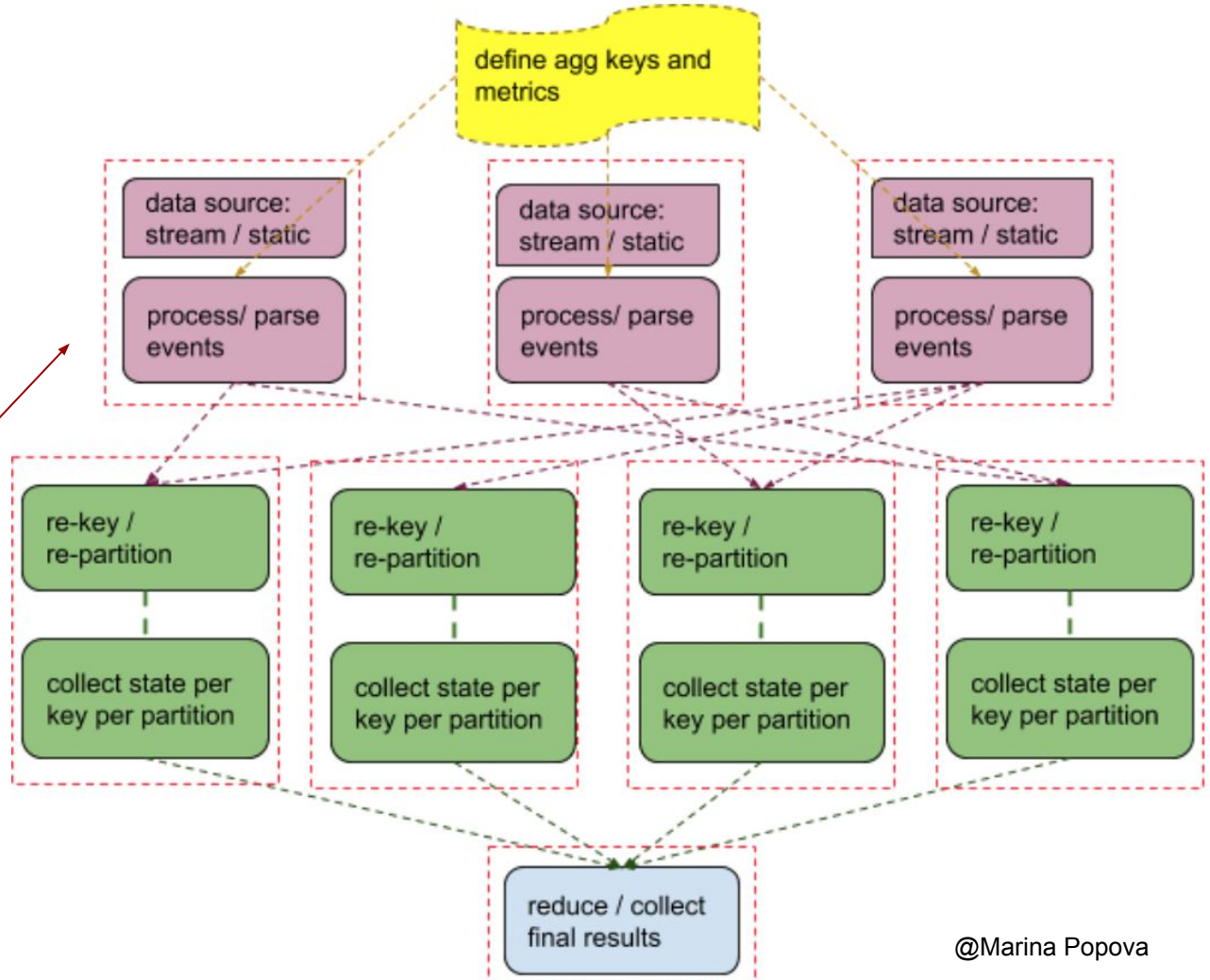


@Marina Popova

Stateful operations

- aggregates:
  - MIN(), MAX()
  - TOPK
  - CountBy
- uniques (exact)

different physical servers!



@Marina Popova

Yottaa Rapid: streaming aggregates

```
Rapid Counter Definitions:
{
    rc_dimensions: { ... },
    rc_metrics: { ... }
}
```

data: Kafka - partitions

processing: rc-event-collectors

re-keyed aggregates in Kafka:
rc-bucket-aggregator,
rc-window-aggregator

partial state: Cassandra

partial results: PG QOE table (overwritten when new events contribute to the 1hr window)

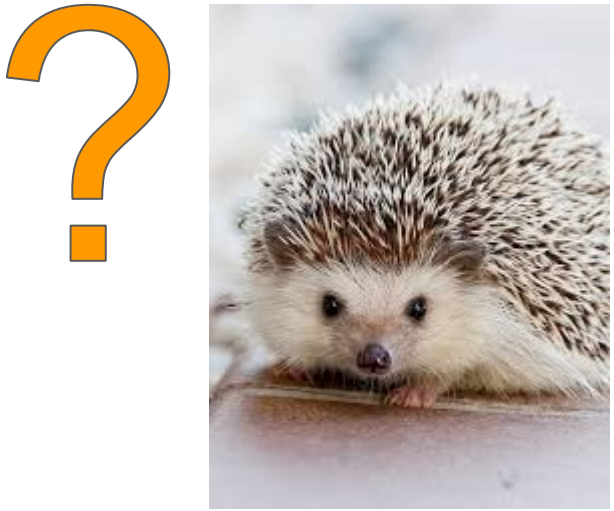
results: Postgres QOE DB - 1 hr counts

@Marina Popova

Rapid Aggregates - architecture type?

what type of system is it?

??????



Type 1: Limited Real-time + Ad-hoc queries systems:
Systems that can answer any [ad-hoc] question based on the **limited raw data** (in-memory only) received during some small recent time window

Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:
systems that can answer pre-defined questions based on the **historical pre-aggregated data (metrics)** - not on all historical raw data - and on the limited latest real-time raw data

Type 3: Full Historical + No Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far - but no real-time up-to-the-second data

Type 4: Full Historical + Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far, including up-to-the-second data

Answer: Type 2 System - real-time is coming soon!

@Marina Popova

Next: Real-Time Alerts

Goals:

- define "anomalies" in real-time - using any of the available event attributes and flexible match criteria
- flexible alert window durations
- send alert notifications when "anomalies" criteria are met
- horizontally scalable (of course!)

Real-Time Alerts: Critical Design Decisions

Alert Definitions and Match Criteria are defined as meta-data in Postgres:

table **ALERT_DEFINITIONS**:

- ID
- name
- threshold - BIGINT
- window_duration_seconds
- operator - EQ, GT. LT (wish list)

table **MATCH_CRITERIA**:

- ID
- alert_definition_id
- field - [status_code, os_browser,..]
- operator - IN, CONTAINS, NOT, ...
- match_expr - RegEx

The counts for each alert definition are incremented/stored in Redis and are disposed of after the time window is expired.

Thsi is the code where this magic happens!

```
if (alert.getAlertDefinition().getThreshold() == counterValue) {
    // this counter exceeded its configured MAX value - send an alert message to Kafka;
    // this will happen in one thread only - since incr() is an atomic operation
    // and will return new value equal to the MAX to one thread only
    sendAlertMessage(alert);
}
```

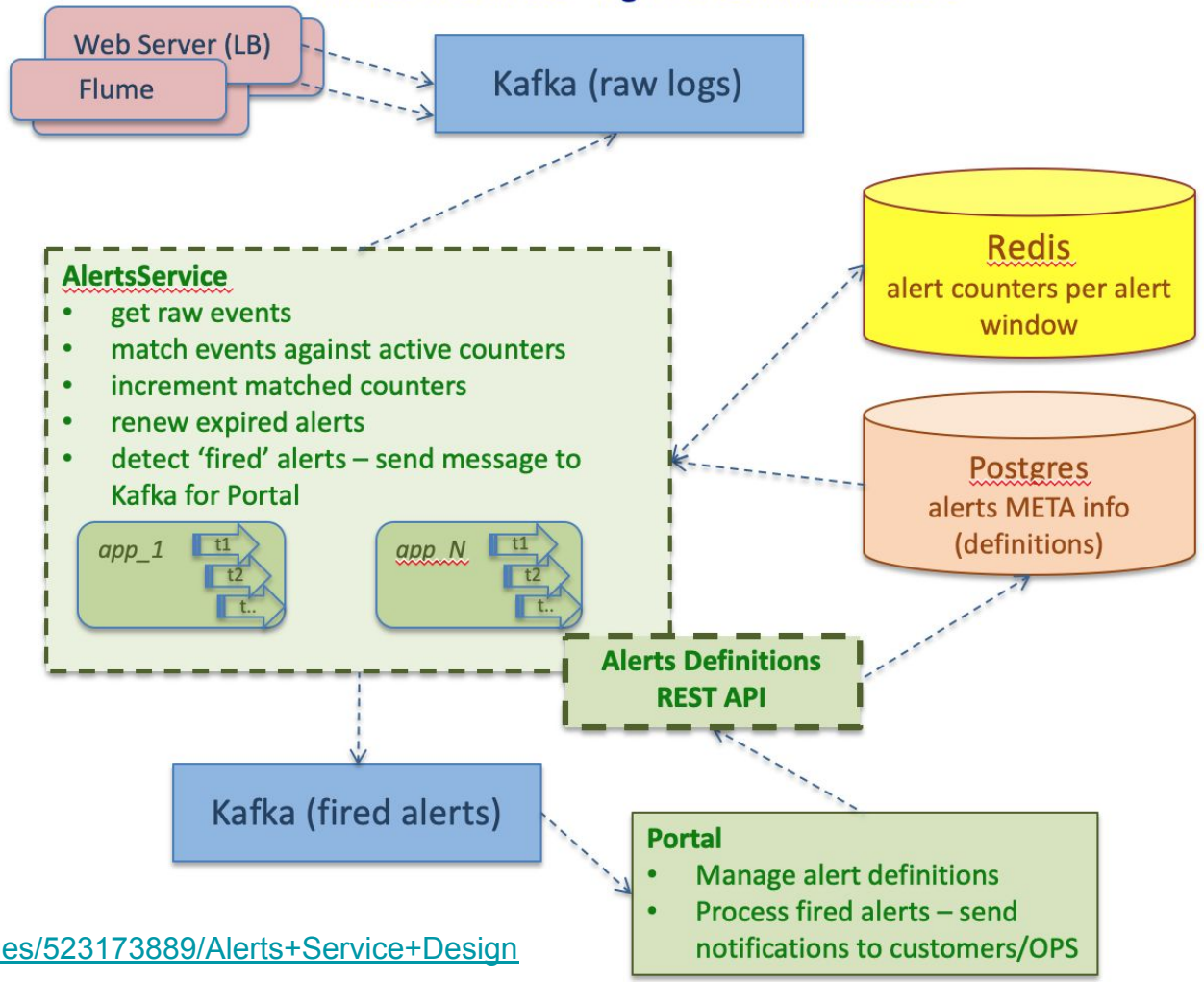
@Marina Popova

Yottaa Alert Service

Redis counters:

key: timeWindow_siteKey_alertID
value: count

example:
siteKey = s1
alertID = a1
Redis counter key:
20190604_1005_s3_a1



Alert Definitions:

https://yottaa.jira.com/wiki/spaces/AN/pages/523173889/Alerts+Service+Design

@Marina Popova

Real-Time Alerts

Alert Definitions and match criteria:

```
{
    siteID: { ... },
    field: { ... },
    operator:
}
```

data: Kafka - partitions

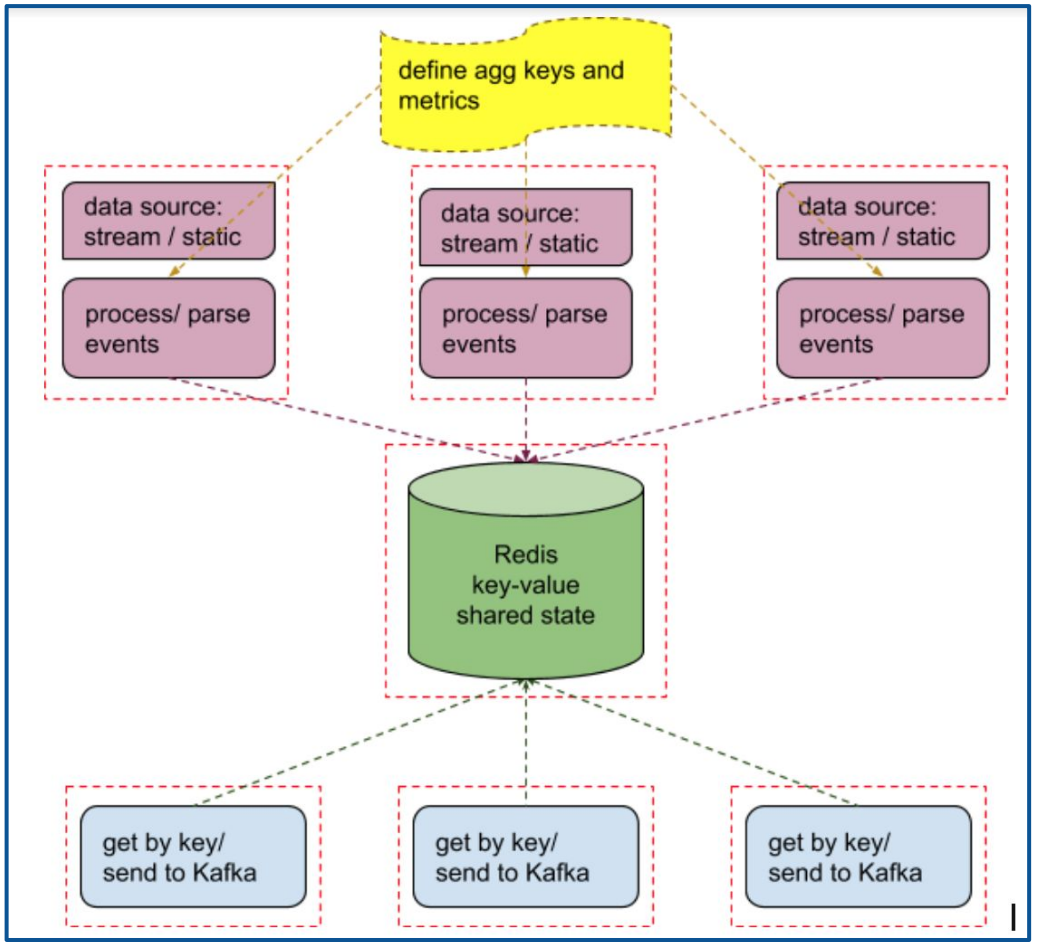
processing: alert-services

re-keyed aggregates: Redis

shared non-distributed state: Redis

NO partial results:

no Reduction of results: alert-service - messages to Kafka



Real-Time Alerts - architecture type?

what type of system is it?

??????

Type 1: Limited Real-time + Ad-hoc queries systems:
Systems that can answer any [ad-hoc] question based on the **limited raw data** (in-memory only) received during some small recent time window

Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:
systems that can answer pre-defined questions based on the **historical pre-aggregated data (metrics)** - not on all historical raw data - and on the limited latest real-time raw data

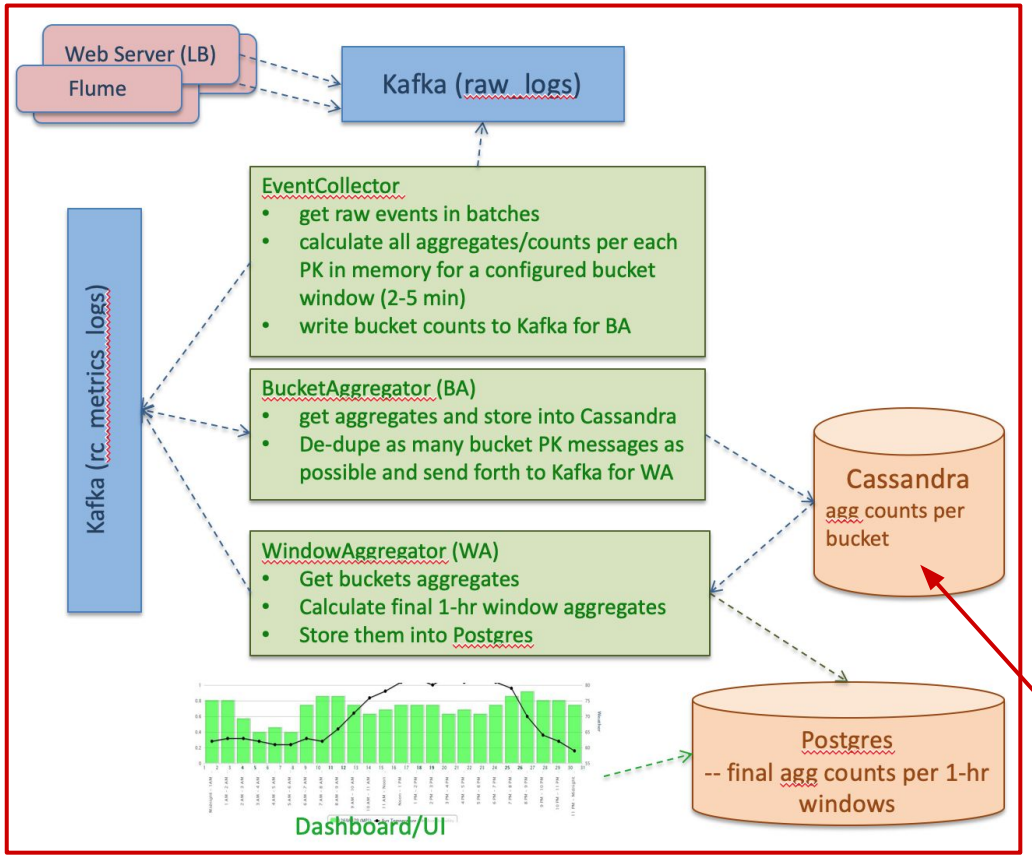
Type 3: Full Historical + No Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far - but no real-time up-to-the-second data

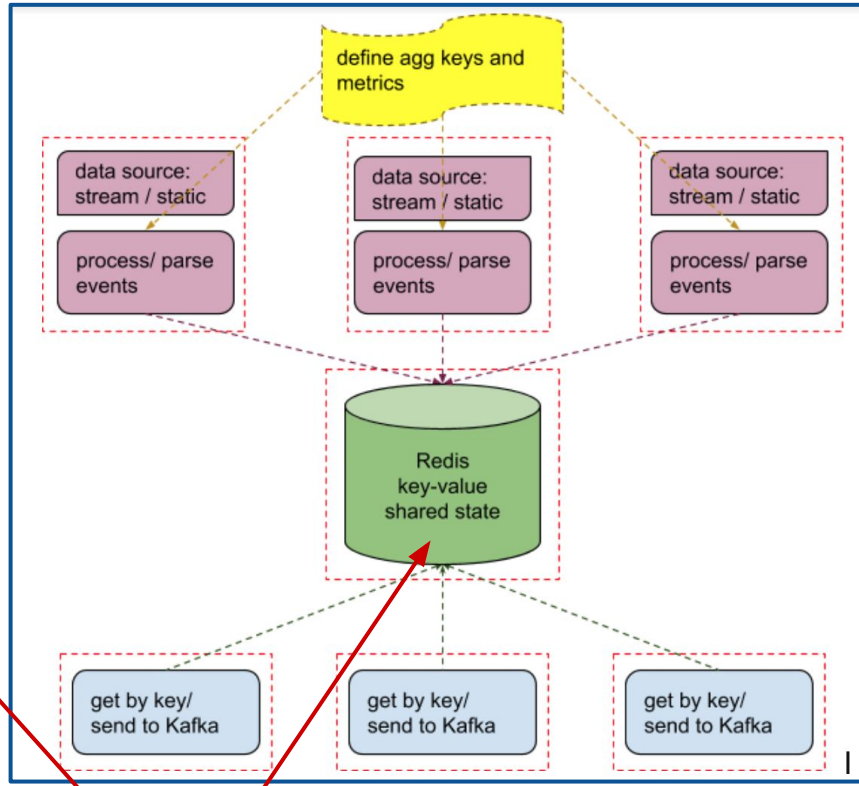
Type 4: Full Historical + Real-Time + Ad-hoc queries systems:
systems that can answer any [ad-hoc] question on **ALL raw data** collected so far, including up-to-the-second data

Answer: pure Type 1 System

@Marina Popova

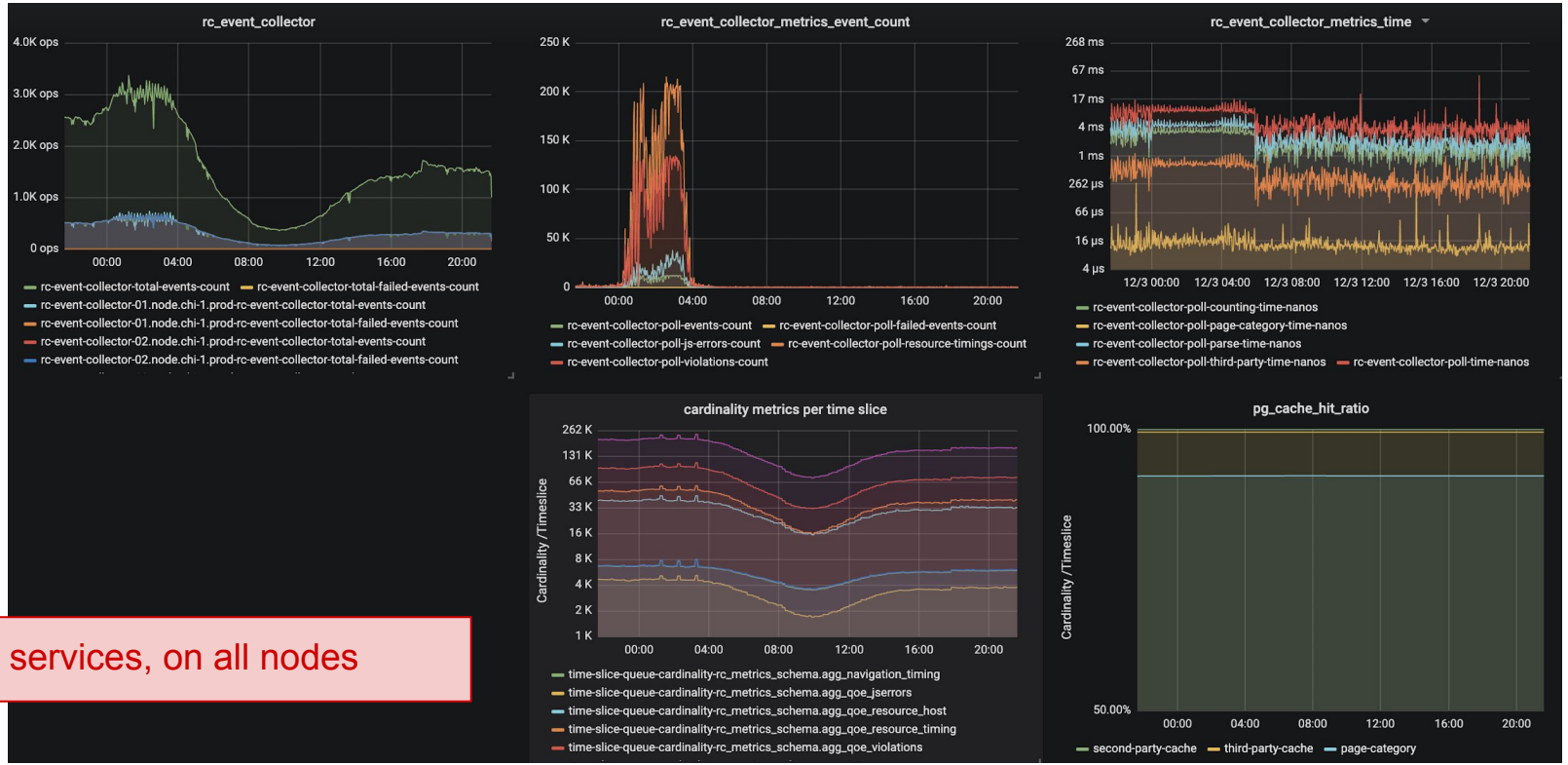
Rapid aggregates vs Alerts Service - why not the same?





State!!! 12 hrs vs 5 min ...
Durable/distributed vs non-durable

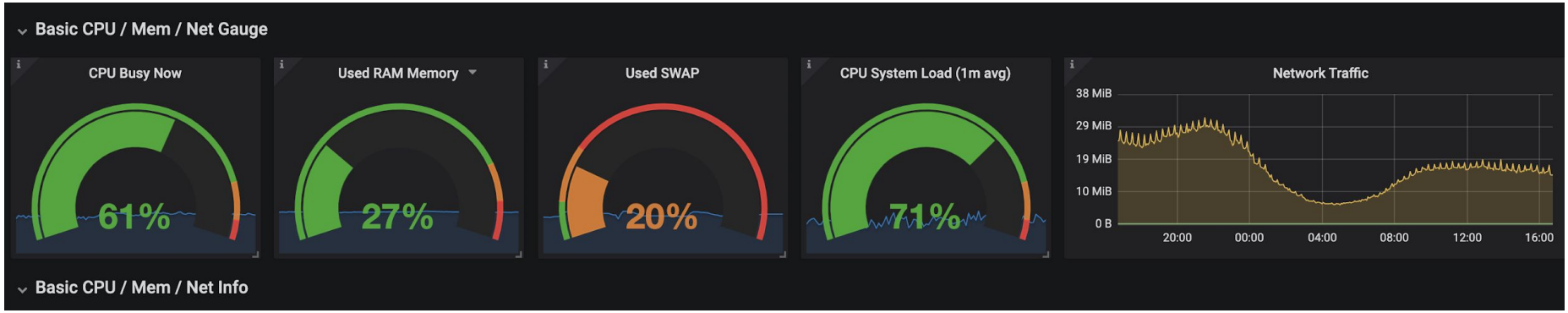
Most important lesson learned: Montor, Monitor, Monitor!!! Everything!

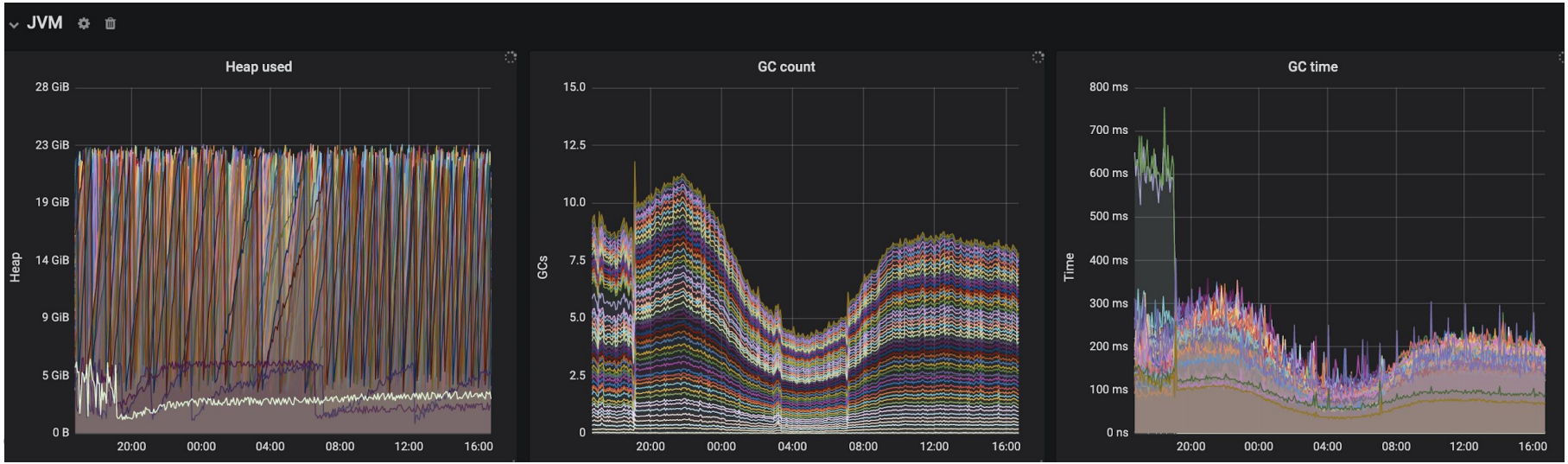


Pipeline services, on all nodes

@Marina Popova

Monitoring: server and JVM-level





Yottaa Analytics Architecture - Type?

Type 1: Limited Real-time Ad-hoc query-ability systems
This are systems that can answer questions about the latest state of the pre-defined metrics, based on the data received during some limited, small recent time window

Type 2: Full historical pre-defined query-ability systems:
systems that allows you to ask questions based on the historical pre-calculated data points (metrics), but not on raw data

Type 3: Full historical Ad-hoc query-ability systems:
systems that allow you to ask any question (ad-hoc query) using the stored raw data points, over the whole set of stored data - but no real-time up-to-the-second data

Type 4: Full Historical and Real-time ad-hoc query-ability systems: systems that can answer ANY question based on ALL data collected so far, including up-to-the-second data

Answer: Type 1 + Type 2 + Type 3 Pipelines!

Yottaa Analytics Team

Marina
[eng]

Vitalii
[eng]

Xitij
[QA]

Dhyan
[eng]

Andriy
[eng]

@Marina Popova