800,000,000 events/day

Lars Marius Garshol, lars.marius.garshol@schibsted.com
http://twitter.com/larsga

SCHIBSTED
MEDIA GROUP

2017–02–08, JFokus 2017

Schibsted?
Collecting events?
Why?

Schibsted

SCHIBSTED
MEDIA GROUP

Three parts

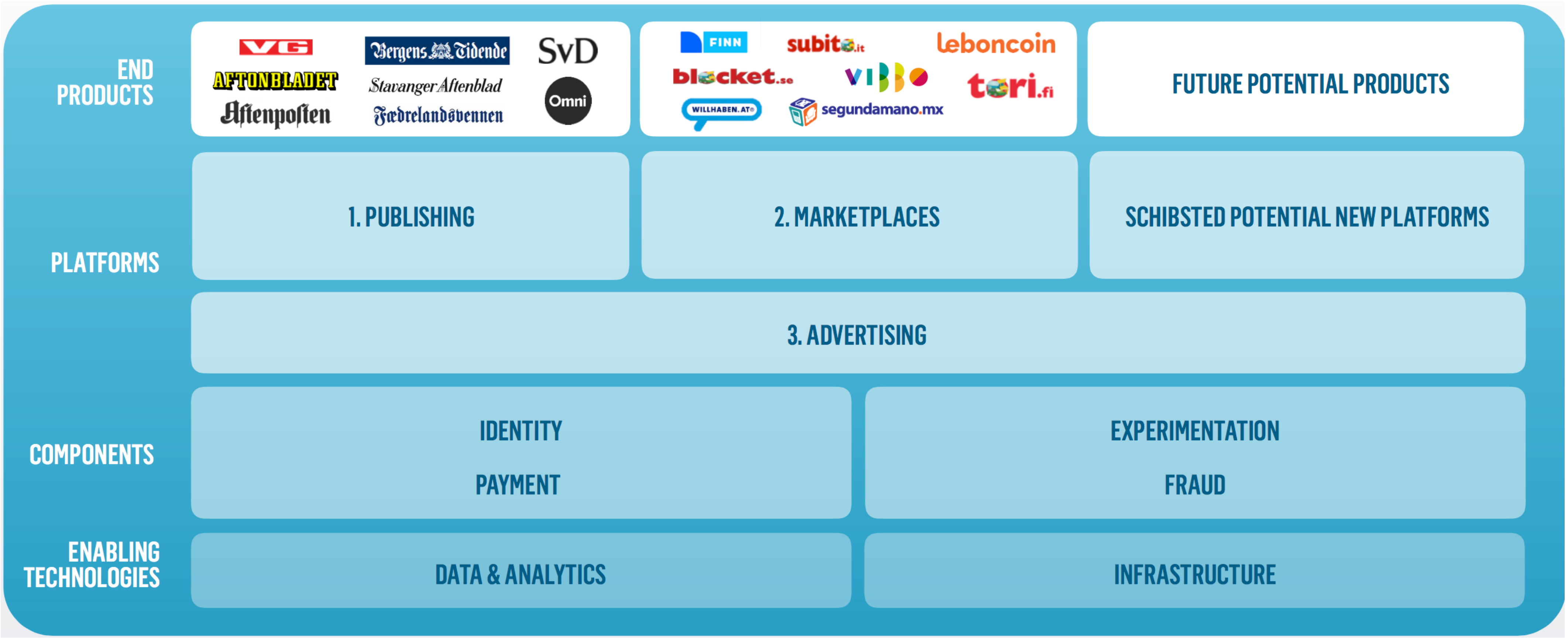
SCHIBSTED MEDIA GROUP

Tinius

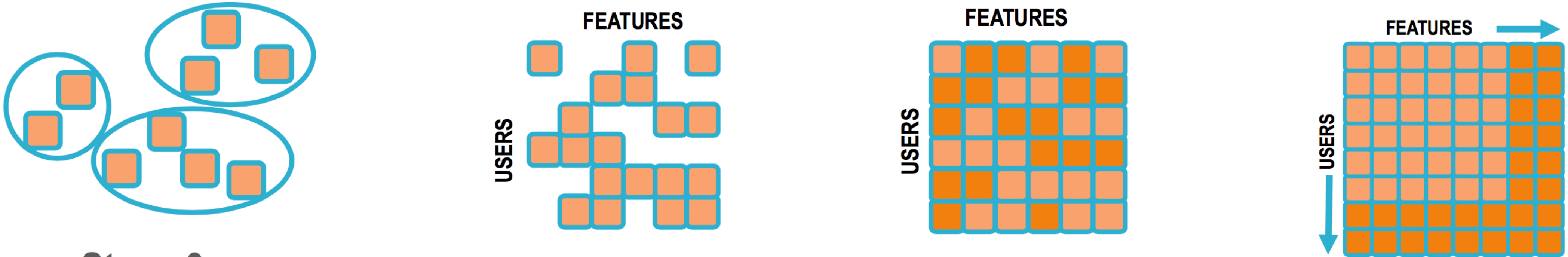
SCHIBSTED
MEDIA GROUP

Restructuring

SCHIBSTED
MEDIA GROUP



User data is central



Stage 0:
Silos of
User Data

Stage 1:
Sums of
User Data

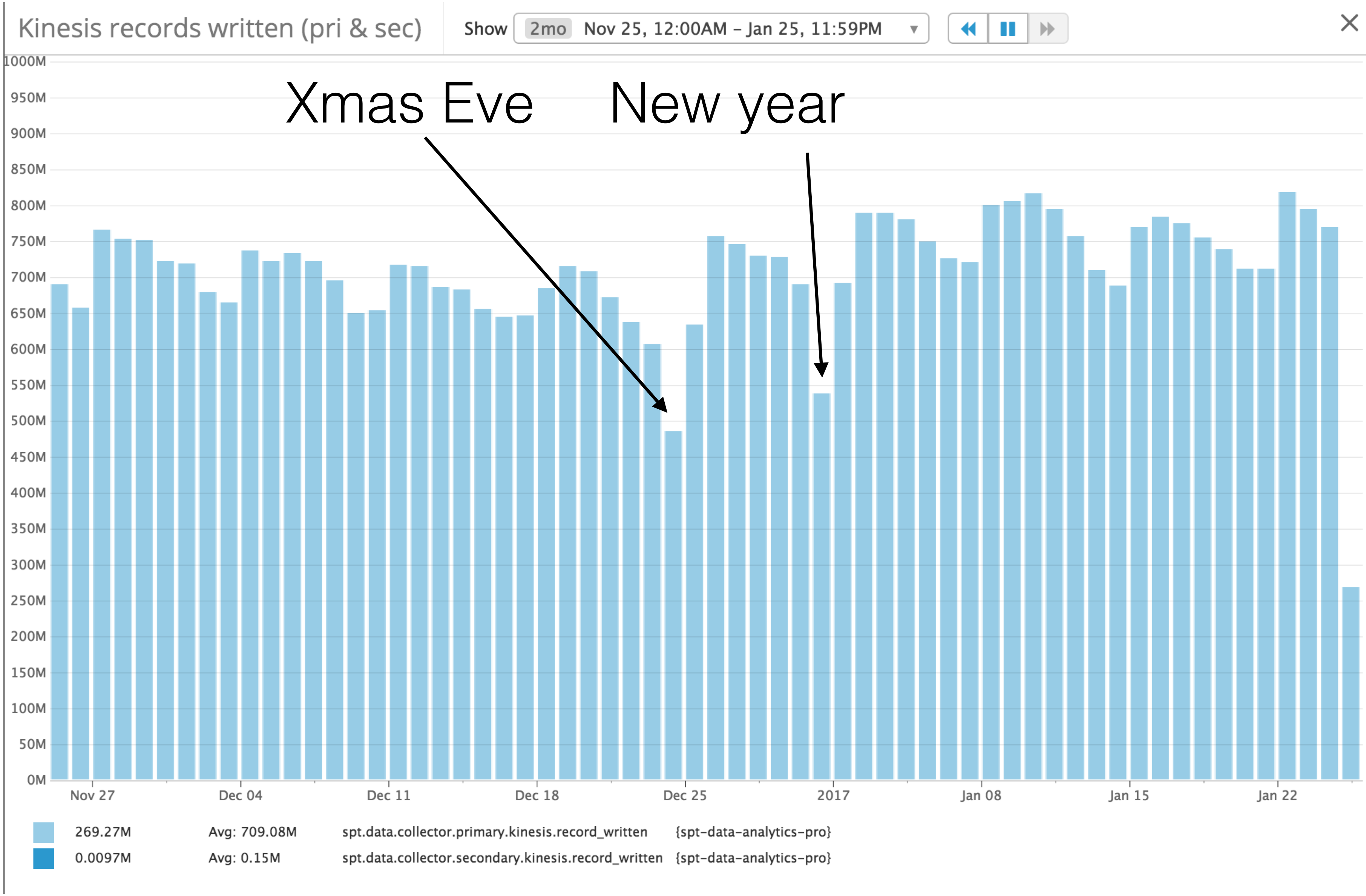
Stage 2:
Enriched
User Data

Stage 3:
Self Reinforcing
User Data

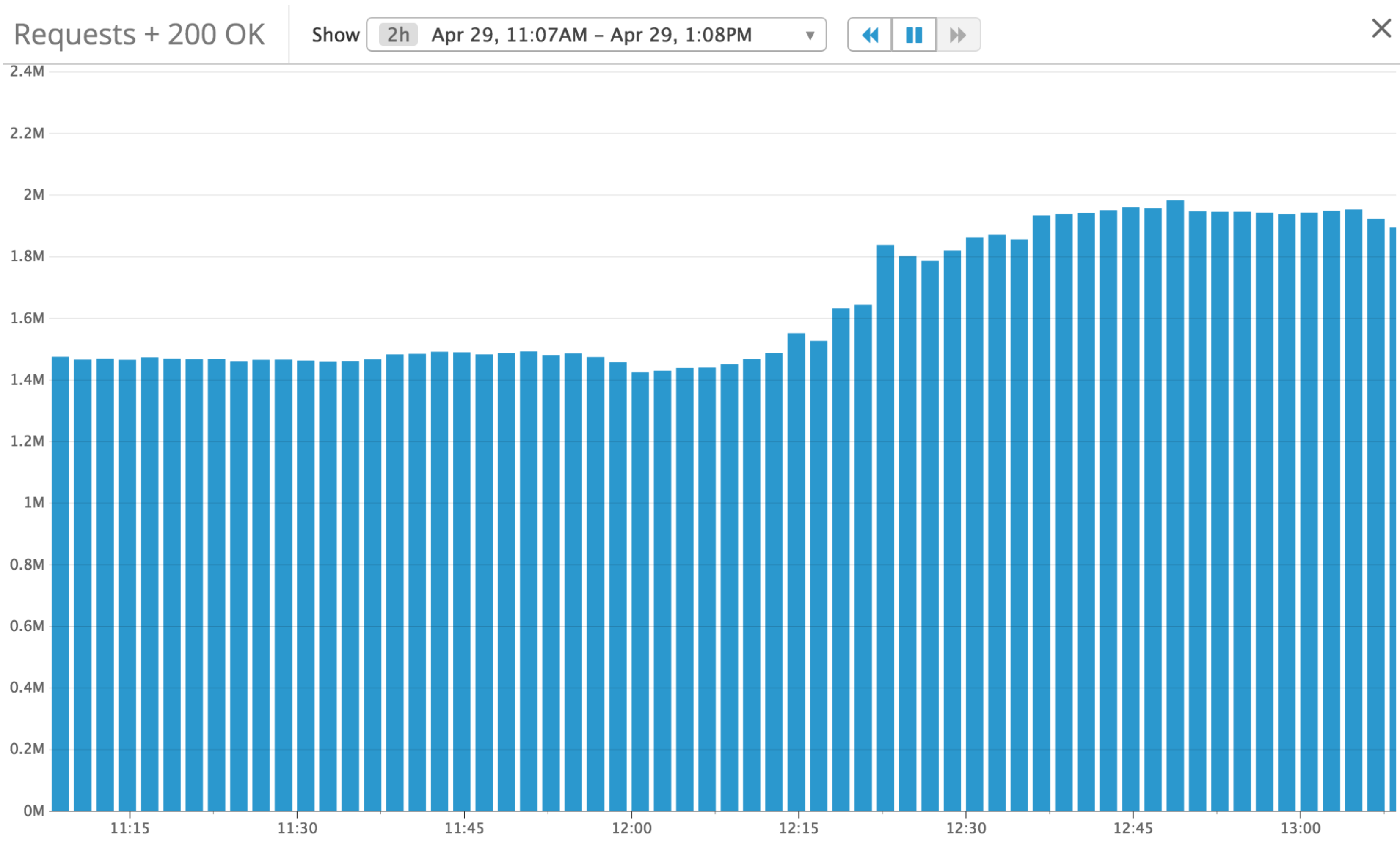
Event collection

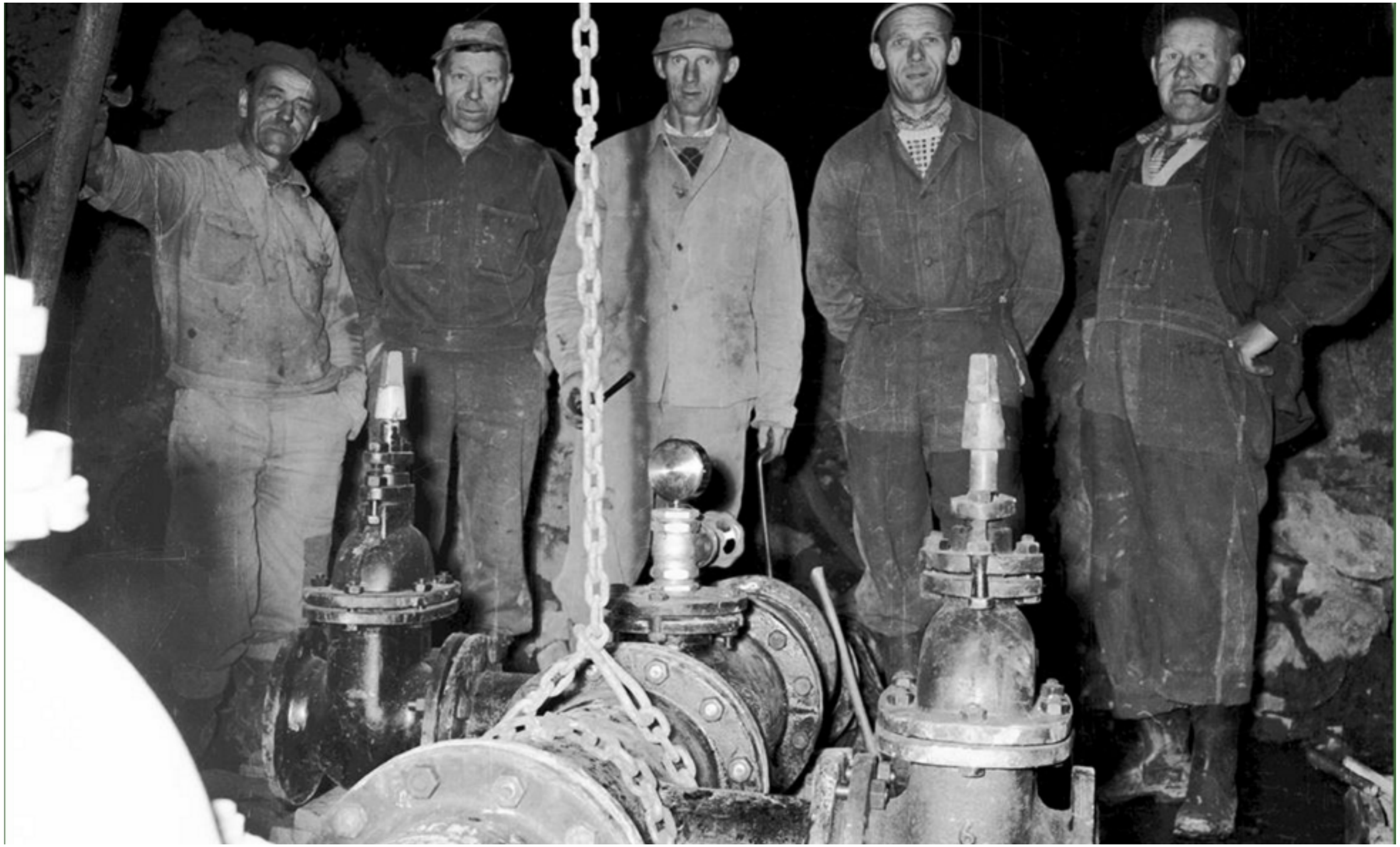
- Event collection pipeline exists to enable strategy
- Collect information on user behaviour
  - across all web properties
- Use to
  - target ads
  - improve products
  - make recommendations
  - understand users

Events/day


Kinesis records written (pri & sec)
Show 2mo Nov 25, 12:00AM – Jan 25, 11:59PM
Xmas Eve
New year
1000M
950M
900M
850M
800M
750M
700M
650M
600M
550M
500M
450M
400M
350M
300M
250M
200M
150M
100M
50M
0M
Nov 27
Dec 04
Dec 11
Dec 18
Dec 25
2017
Jan 08
Jan 15
Jan 22
269.27M Avg: 709.08M spt.data.collector.primary.kinesis.record_written {spt-data-analytics-pro}
0.0097M Avg: 0.15M spt.data.collector.secondary.kinesis.record_written {spt-data-analytics-pro}

Helicopter crash

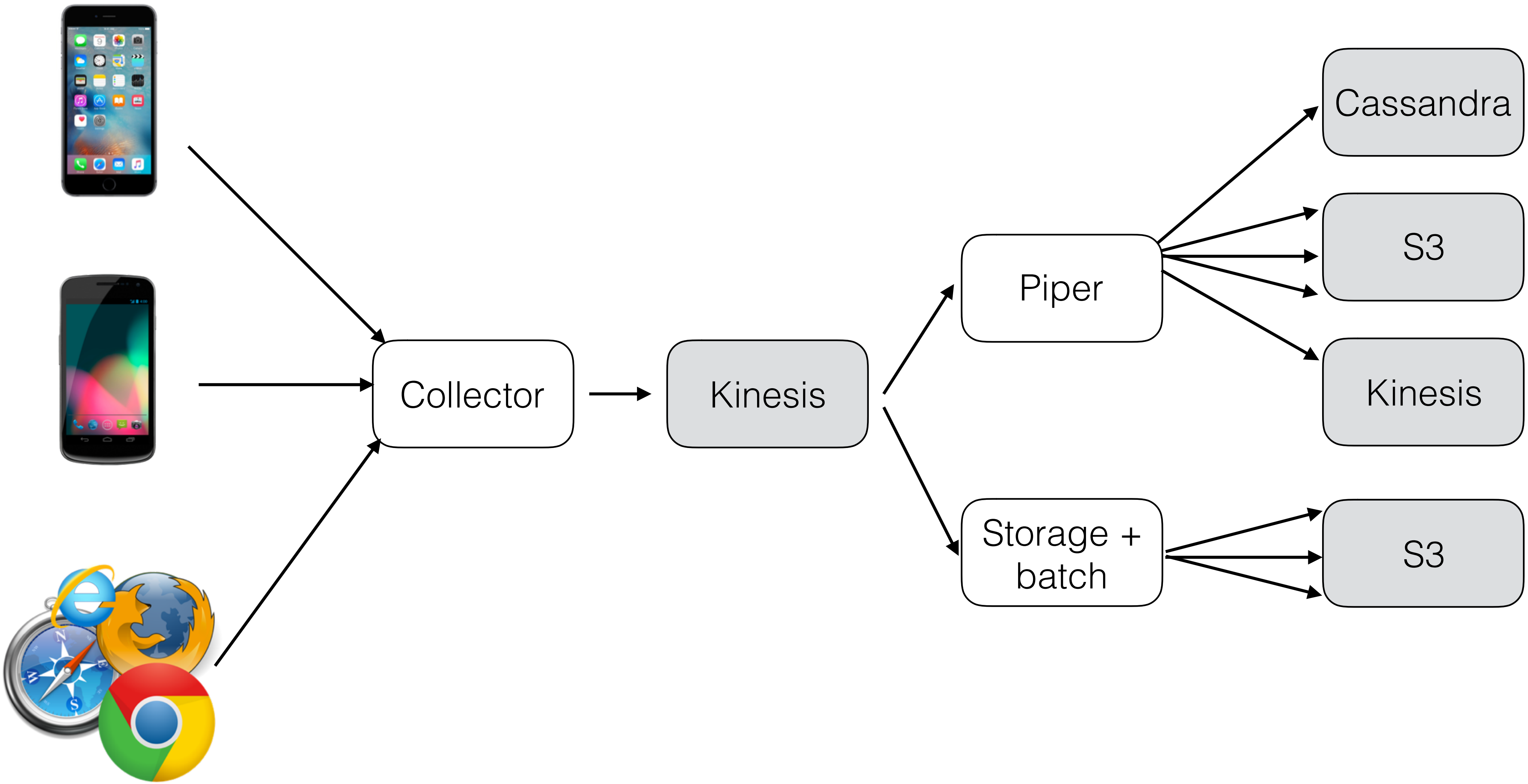




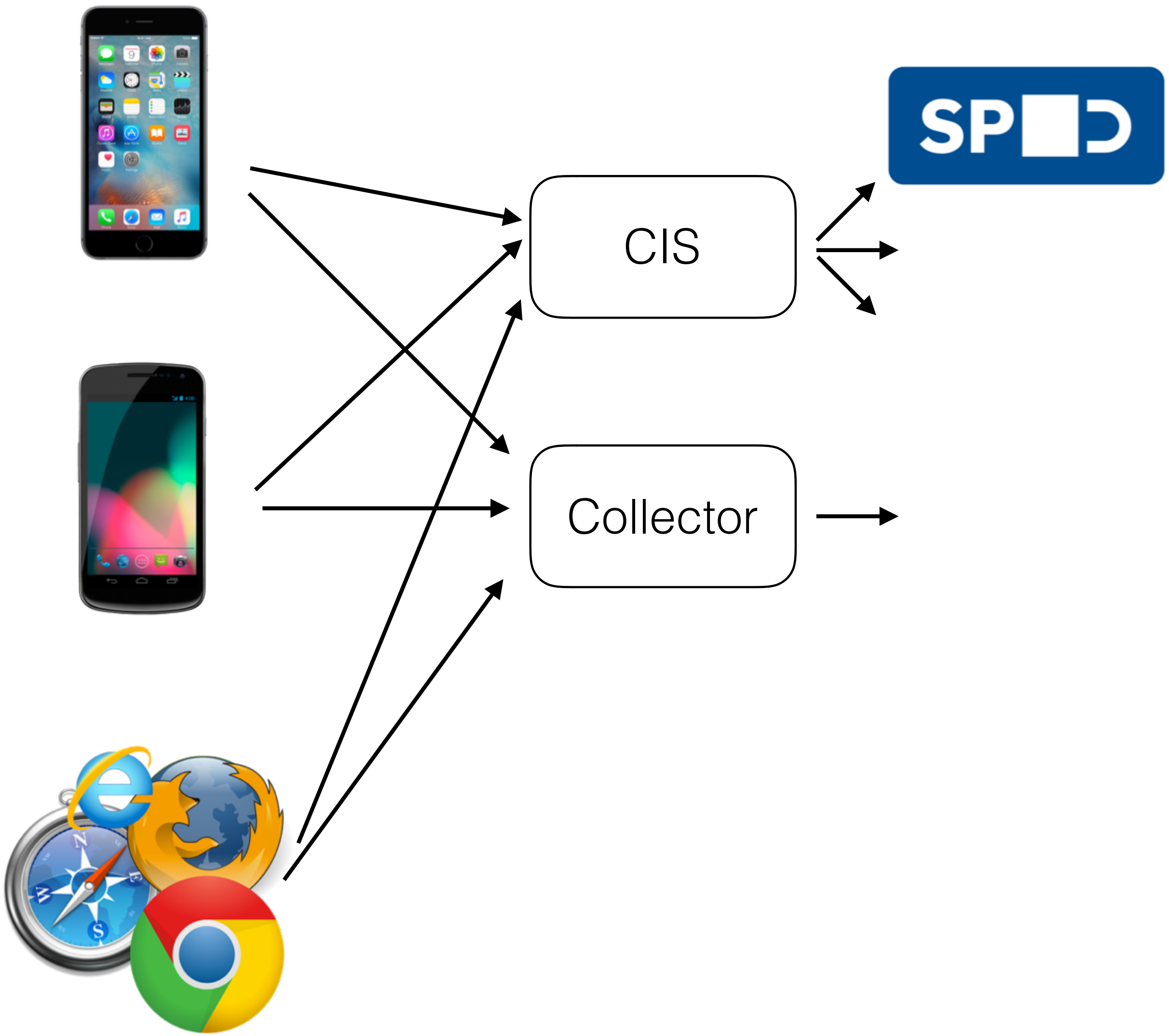
Rælingen, 1960. Photo by Per Solrud

The pipeline

Architecture



Complications



Steps:

1. Start batching events
2. Get ID from CIS
3. If opt out, stop
4. Send events

Storage

SCHIBSTED MEDIA GROUP

- The second step writes events into S3 as JSON
- S3: AWS storage system
  - kind of like a file system over HTTP
- You can write to s3://bucket/path/to/file
  - files are blobs
  - have to write entire file when writing, no random access
  - eventually consistent
- Very strong guarantee on durability
  - but substantial latency

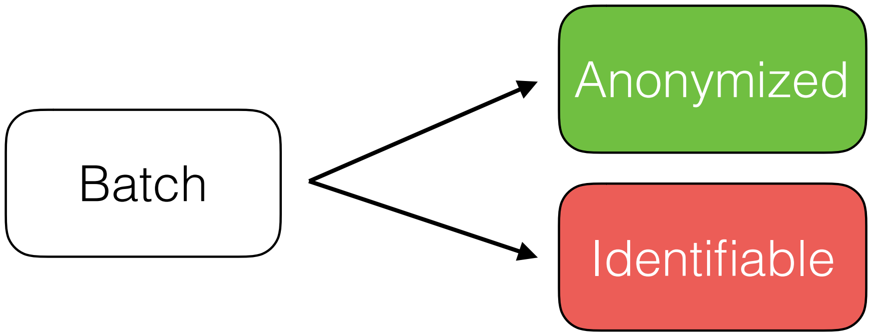
Why not S3 directly?

- Writing to S3 takes ~2-4 seconds
  - a long time to wait before the events are safe
- Reading takes a while, too
- Not good for “realtime” usages
  - of which we have a few

Storage

- Each event ~2200 bytes of JSON = ~1760 GB/day
    - this turns out to be very slow to load in Spark
- Switched over to using batch jobs coalescing JSON into Parquet
    - some difficulties with nested -> flat structure
    - unpredictable schema variations from site to site
    - huge performance gain

Consumers



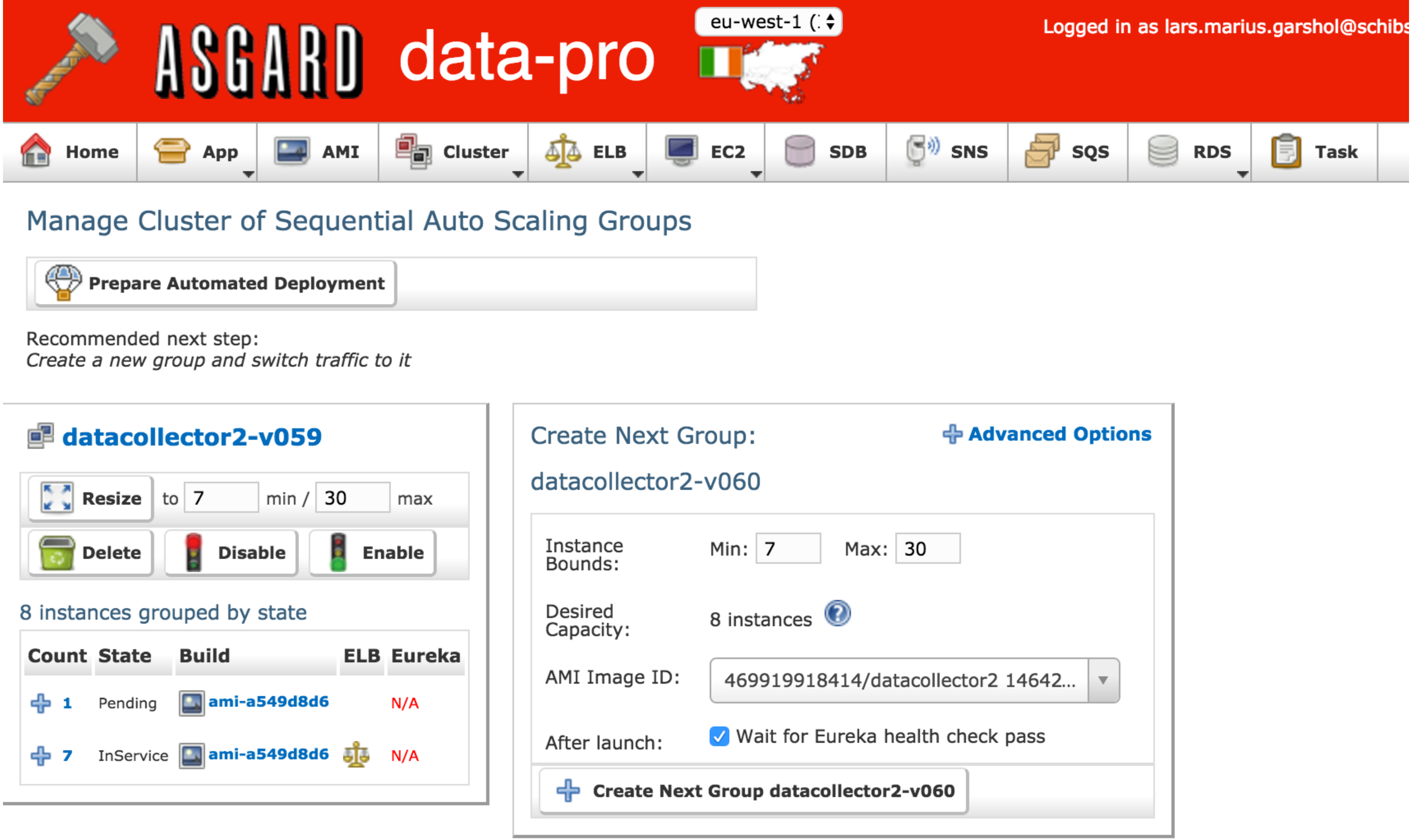
- Ad targeting
- Personalization
- User modelling
- Business intelligence
- Individual sites
- Schibsted internal CMS
- …

amazon web services™

Working in the cloud

Amazon web services

- For each application, build a Linux image (AMI)
- Set up an auto-scaling group
  - define min and max number of nodes
  - define rules to scale up/down based on metrics (CPU, …)
- Amazon ELB in front
  - talks to health check endpoint on each node
- Also provides Kinesis, S3, …



ASGARD data-pro
eu-west-1
Logged in as lars.marius.garshol@schibs
Home
App
AMI
Cluster
ELB
EC2
SDB
SNS
SQS
RDS
Task
Manage Cluster of Sequential Auto Scaling Groups
Prepare Automated Deployment
Recommended next step:
Create a new group and switch traffic to it
datacollector2-v059
Resize to 7 min / 30 max
Delete
Disable
Enable
8 instances grouped by state
Count State Build ELB Eureka
1 Pending ami-a549d8d6 N/A
7 InService ami-a549d8d6 N/A
Create Next Group:
Advanced Options
datacollector2-v060
Instance Bounds: Min: 7 Max: 30
Desired Capacity: 8 instances
AMI Image ID: 469919918414/datacollector2 14642...
After launch: Wait for Eureka health check pass
Create Next Group datacollector2-v060

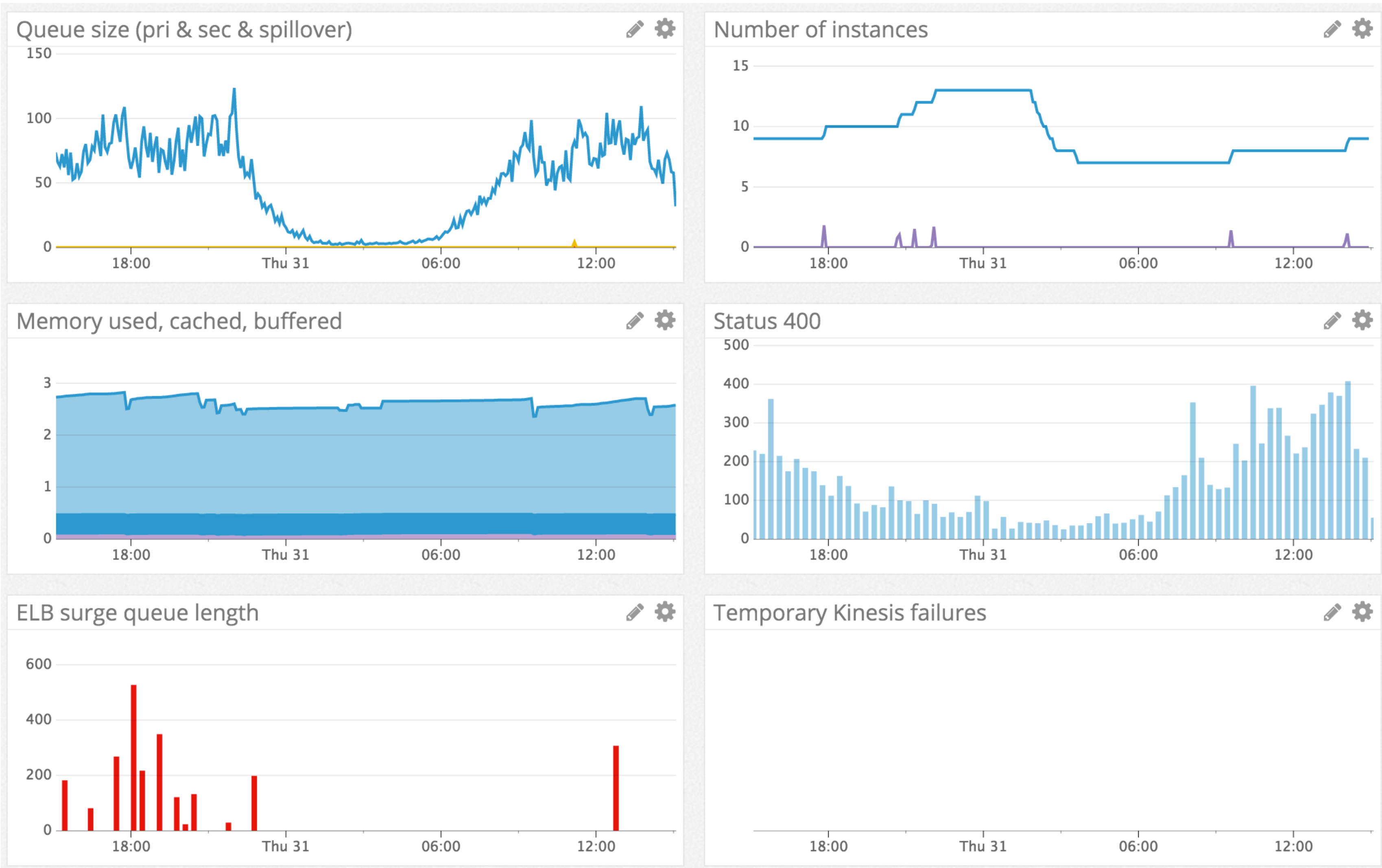
Scaling policies

Scaling Policies

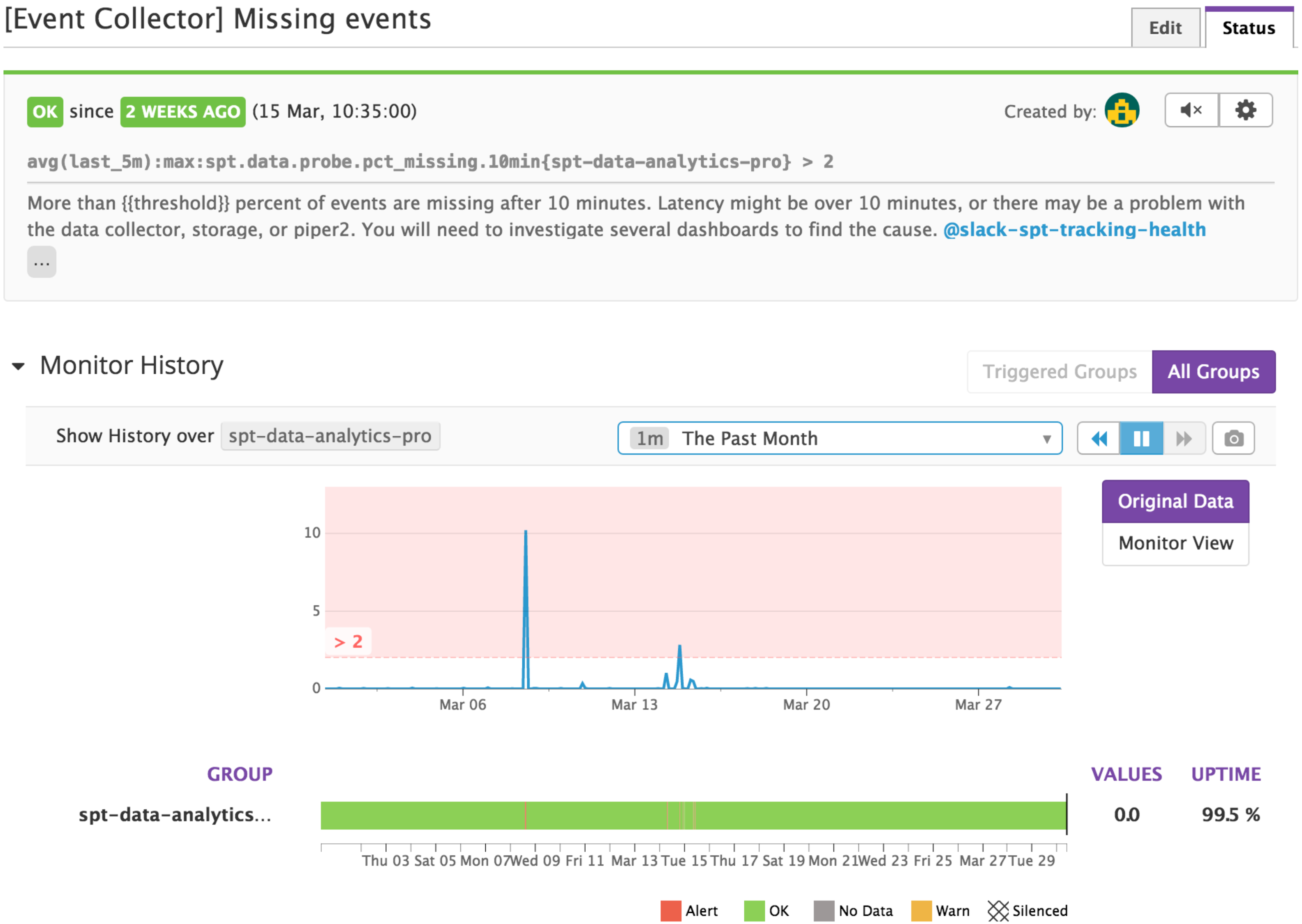
Total Policies: 3 | Create New Scaling Policy

Policy Name	Scaling Adjustment	Adjustment Type	Cooldown	Alarms
datacollector2-v059-537	-1	ChangeInCapacity	600	CPUUtilization < 20.0
datacollector2-v059-538	1	ChangeInCapacity	300	CPUUtilization >= 65.0
datacollector2-v059-539	15	ExactCapacity	600	HealthyHostCount < 4.0

Metrics

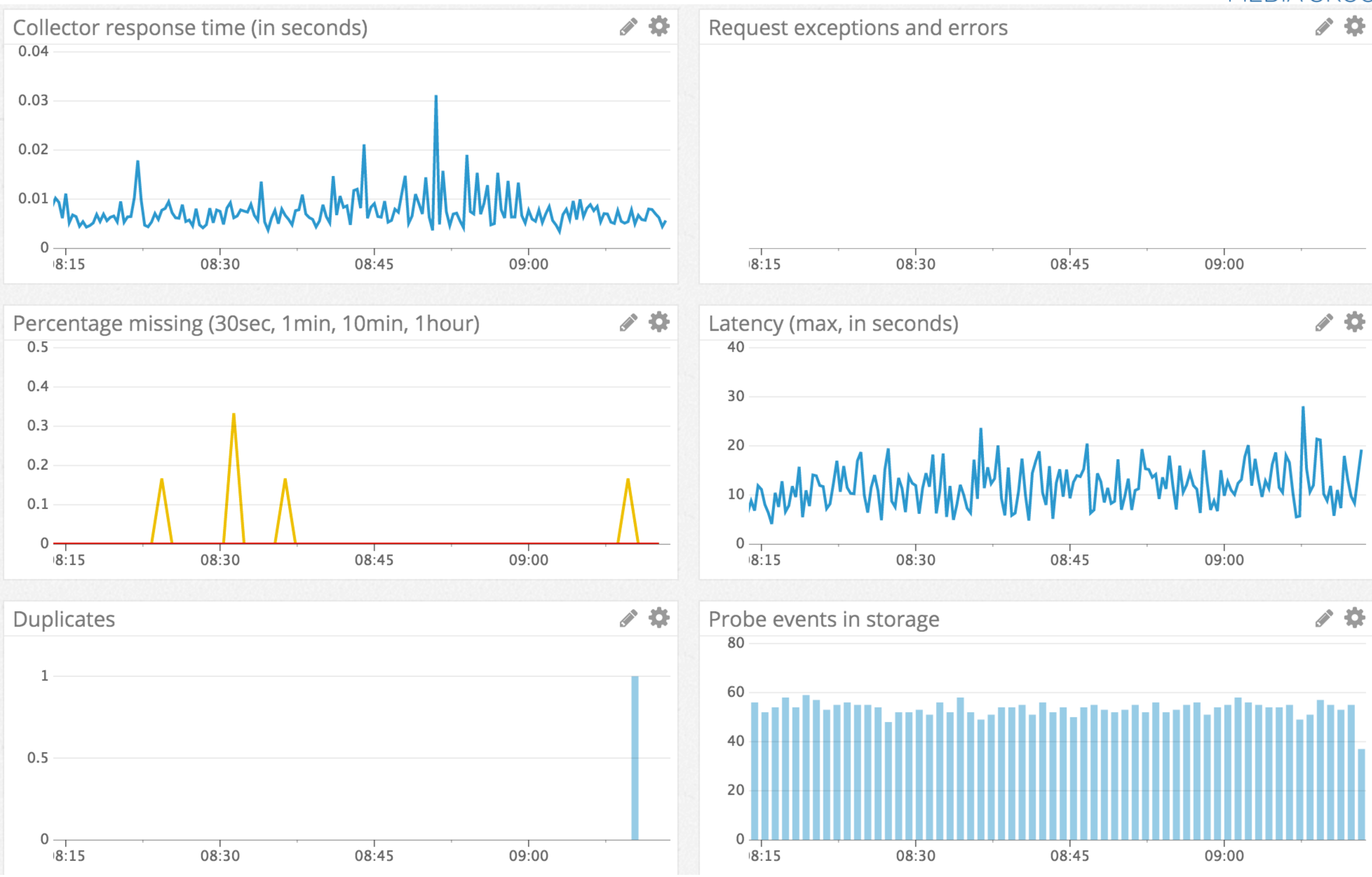

Queue size (pri & sec & spillover)
150
100
50
0
18:00
Thu 31
06:00
12:00
Number of instances
15
10
5
0
18:00
Thu 31
06:00
12:00
Memory used, cached, buffered
3
2
1
0
18:00
Thu 31
06:00
12:00
Status 400
500
400
300
200
100
0
18:00
Thu 31
06:00
12:00
ELB surge queue length
600
400
200
0
18:00
Thu 31
06:00
12:00
Temporary Kinesis failures
18:00
Thu 31
06:00
12:00

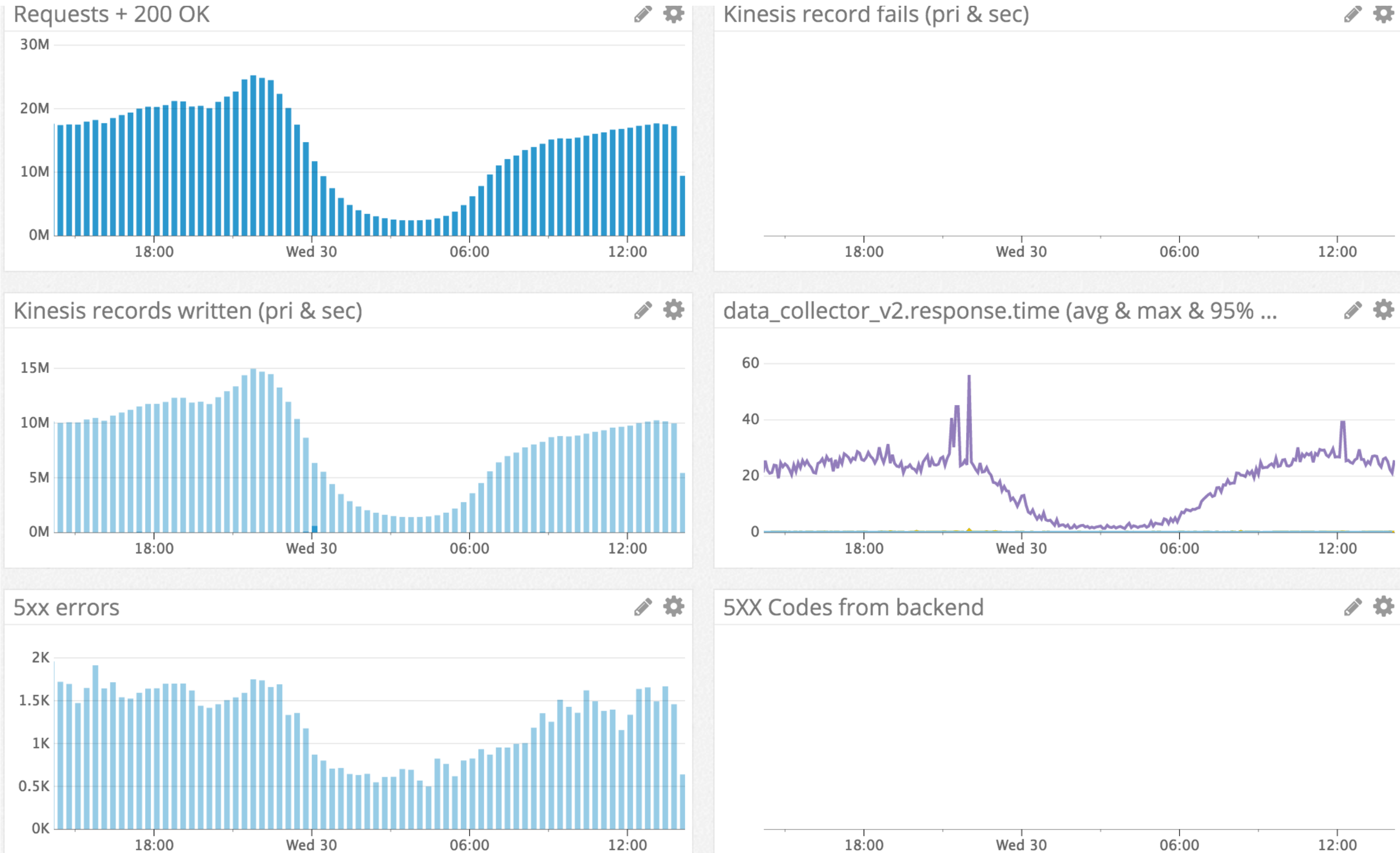
Monitors


[Event Collector] Missing events
Edit
Status
OK since 2 WEEKS AGO (15 Mar, 10:35:00)
Created by:
avg(last_5m):max:spt.data.probe.pct_missing.10min{spt-data-analytics-pro} > 2
More than {{threshold}} percent of events are missing after 10 minutes. Latency might be over 10 minutes, or there may be a problem with the data collector, storage, or piper2. You will need to investigate several dashboards to find the cause. @slack-spt-tracking-health
...
Monitor History
Triggered Groups
All Groups
Show History over spt-data-analytics-pro
1m The Past Month
Original Data
Monitor View
10
5
0
> 2
Mar 06
Mar 13
Mar 20
Mar 27
GROUP
VALUES
UPTIME
spt-data-analytics...
0.0
99.5 %
Thu 03 Sat 05 Mon 07 Wed 09 Fri 11 Mar 13 Tue 15 Thu 17 Sat 19 Mon 21 Wed 23 Fri 25 Mar 27 Tue 29
Alert
OK
No Data
Warn
Silenced

Probe

SCHIBSTED
MEDIA GROUP





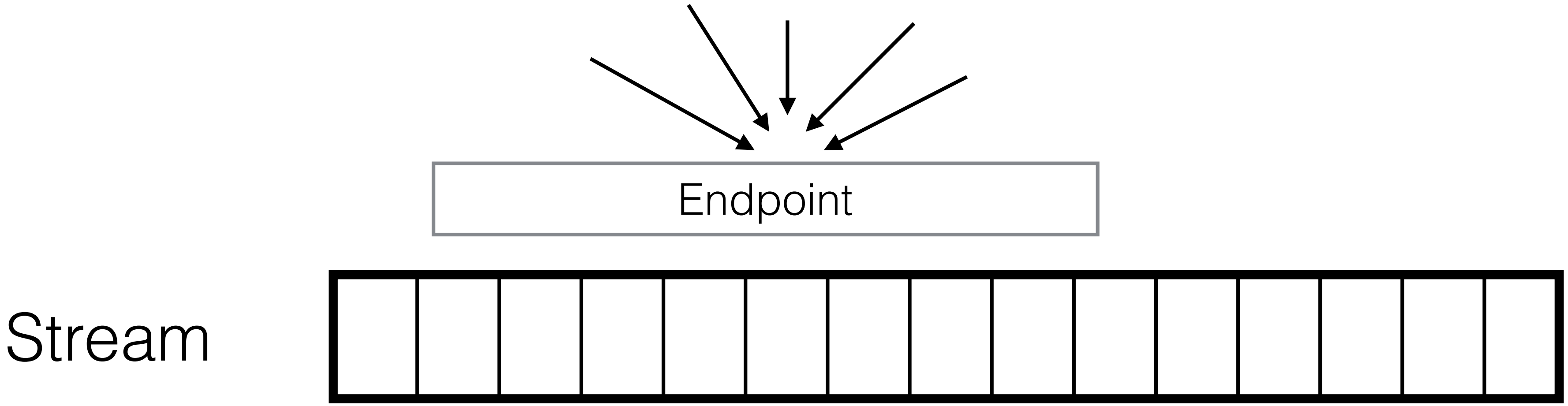
Data collector

Overview

- Vulnerable
  - data not yet persisted
  - can’t expect clients to resend if collector fails
  - traffic comes with sudden spikes
- A very thin layer in front of Kinesis
  - persist the data as quickly as possible
  - we can work on the data later, once it’s stored
- Use Kinesis because it has very low write latency
  - generally <100 ms

Kinesis

SCHIBSTED
MEDIA GROUP

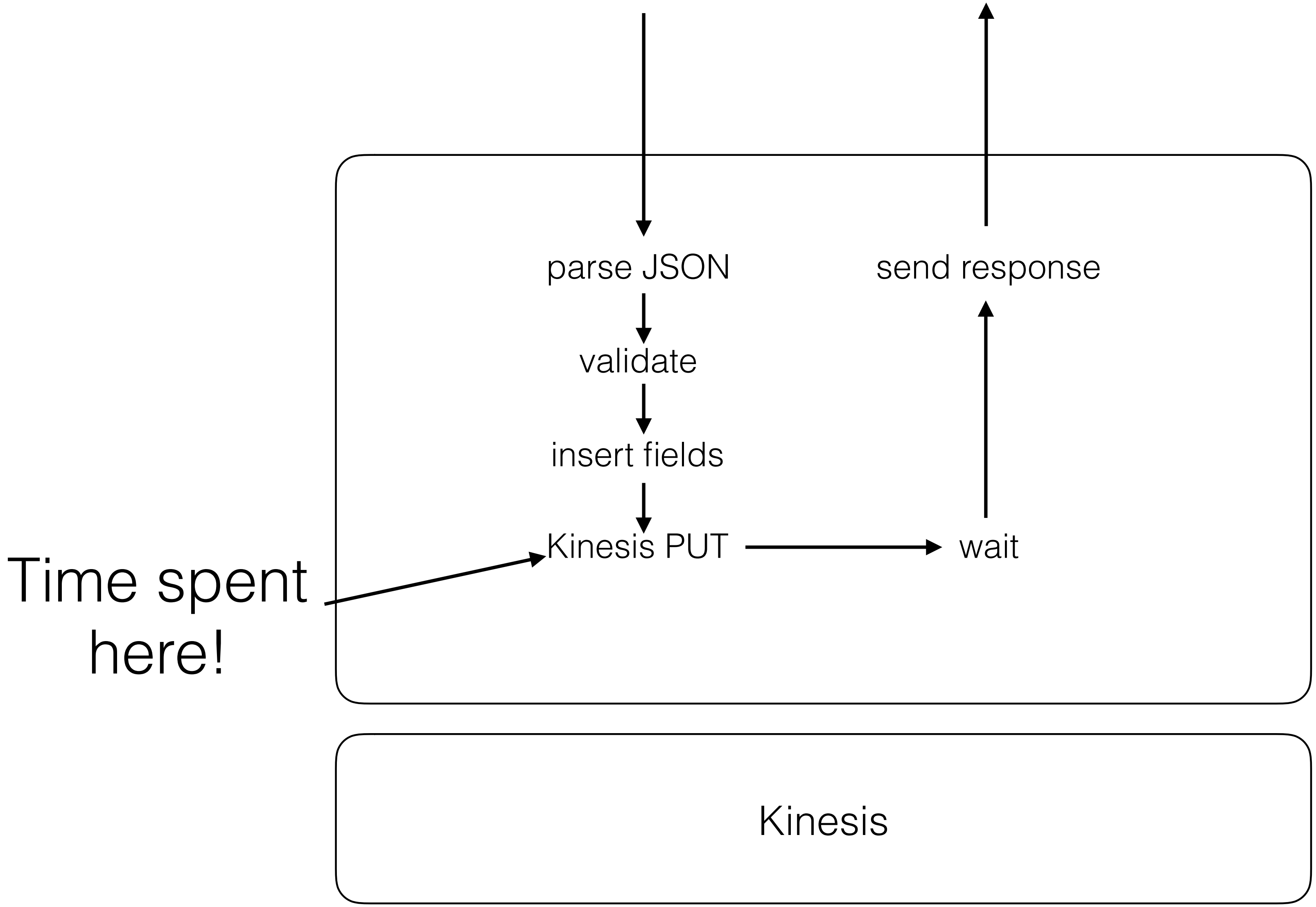


Apache Kafka
A high-throughput distributed messaging system.

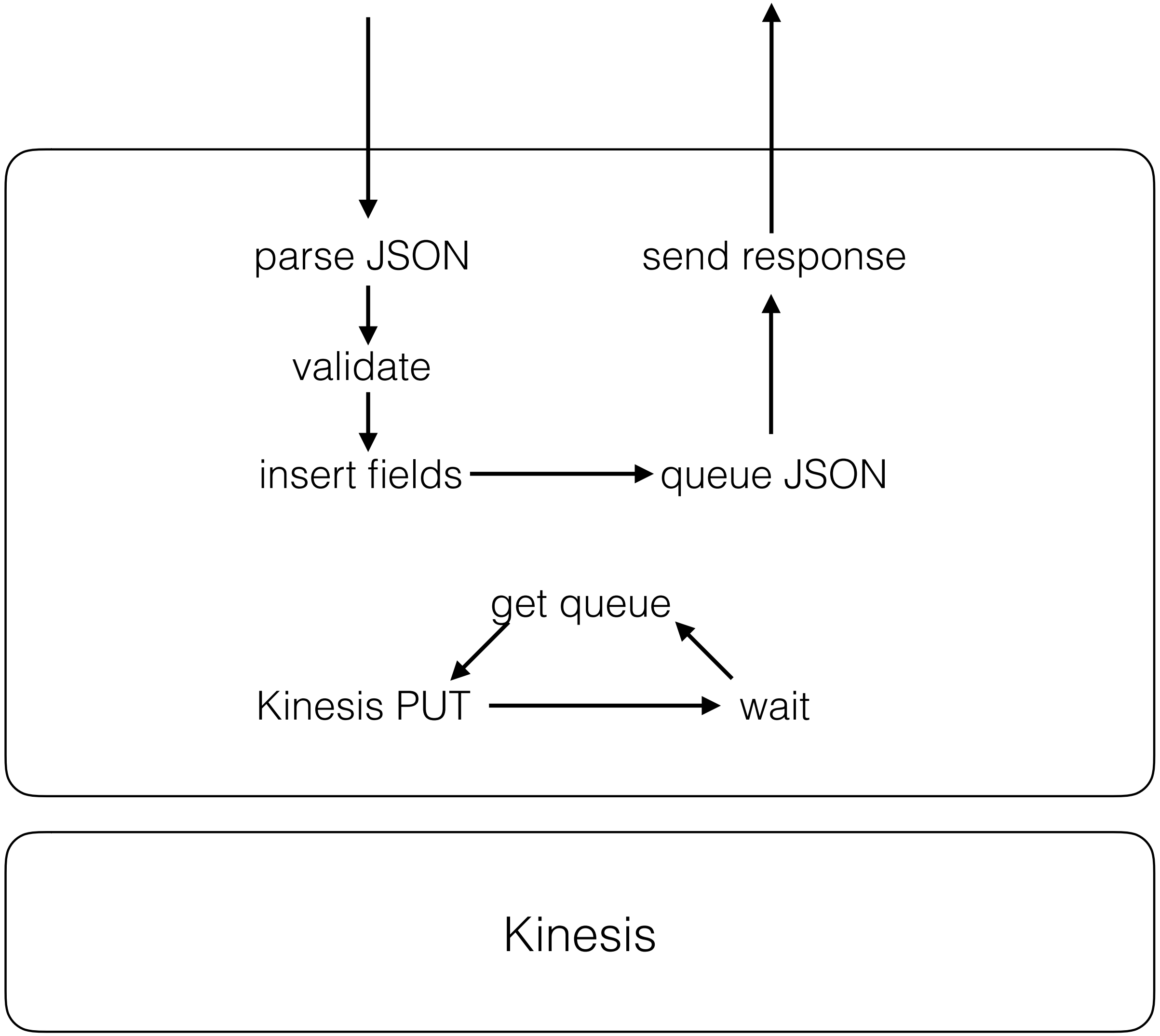
Basics

- Scala application
- Based on Netty/Finagle/Finatra
- Framework scales very well
    - if app is overloaded, the only thing that happens is number of open connections piles up
    - eventually hits max open files, causing failures

First design



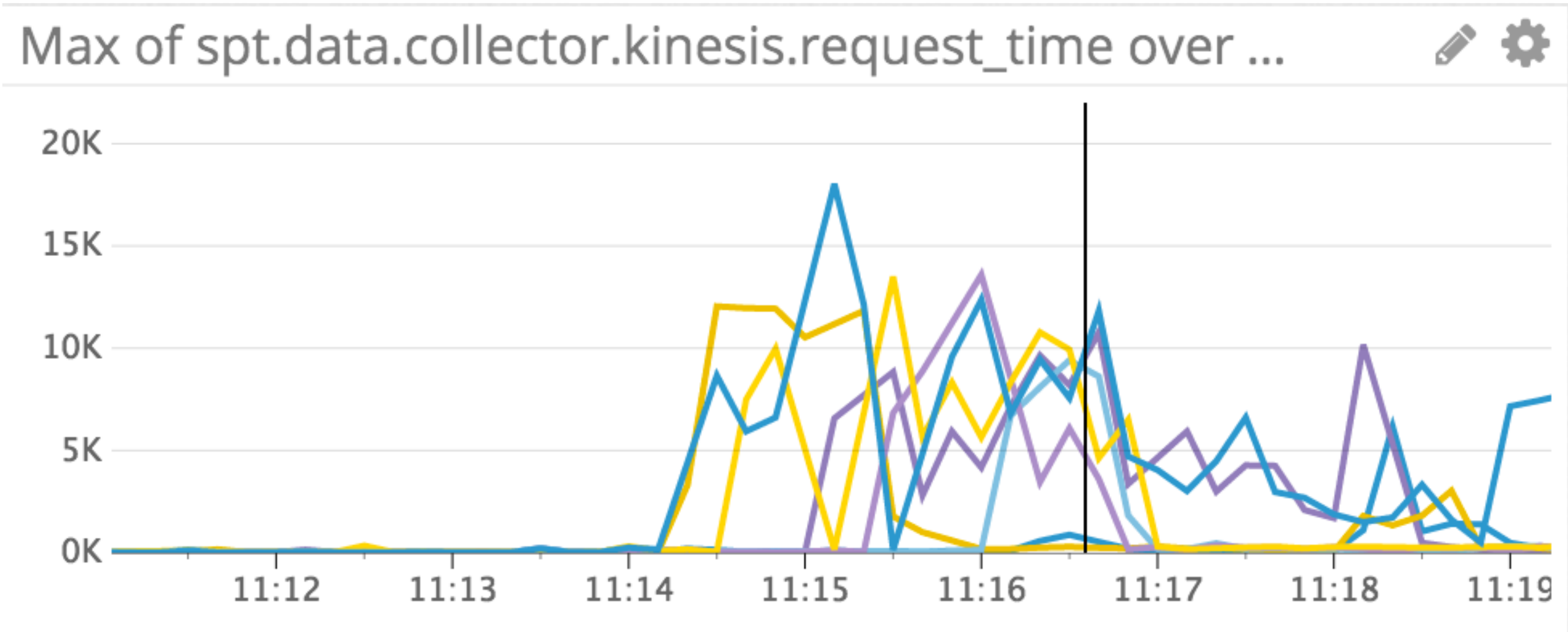
First version



Error handling?

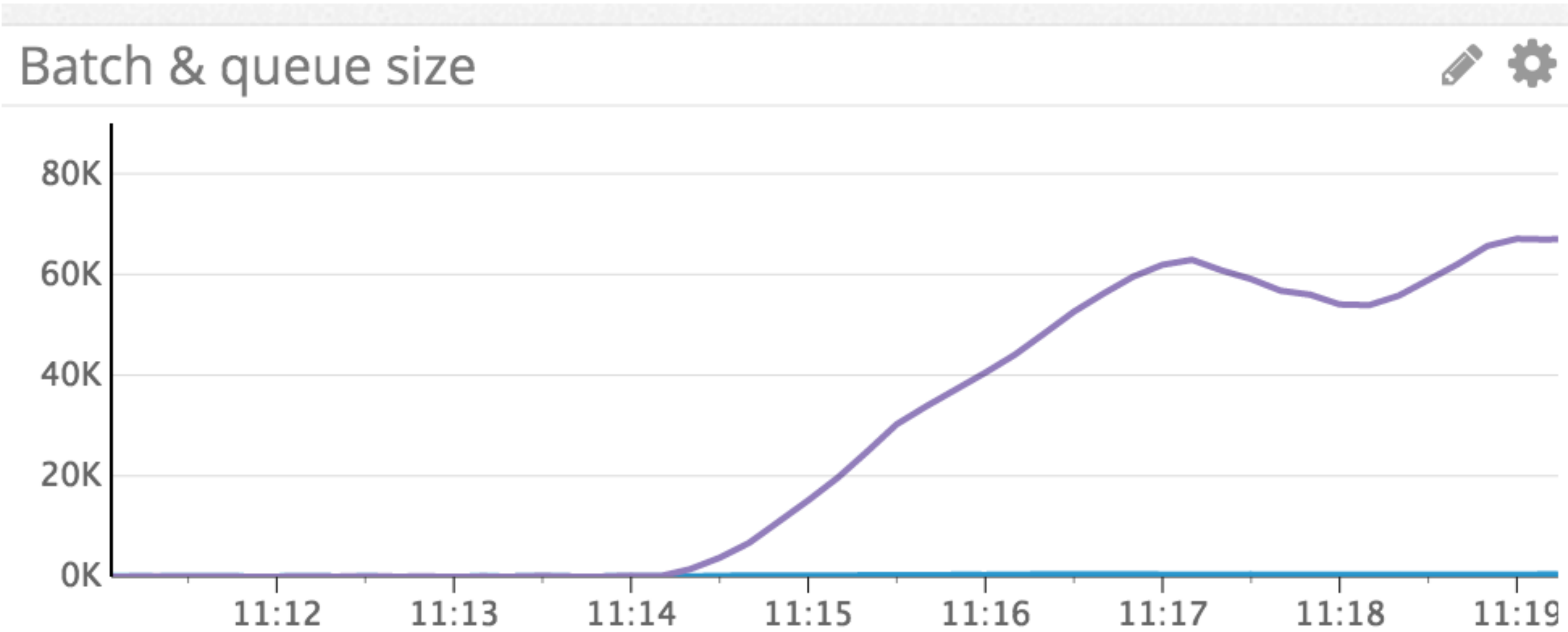
- I had no experience with AWS - didn't know what to expect
    - could Kinesis go down for 1 minute? 1 hour? 1 day?
    - anecdotal reports of single nodes being unable to contact Kinesis while others working fine
- Considered
    - batching on disk with fallback to S3
    - feeding batched events back to collector
    - …
- In the end decided to wait for more experience

Ooops!



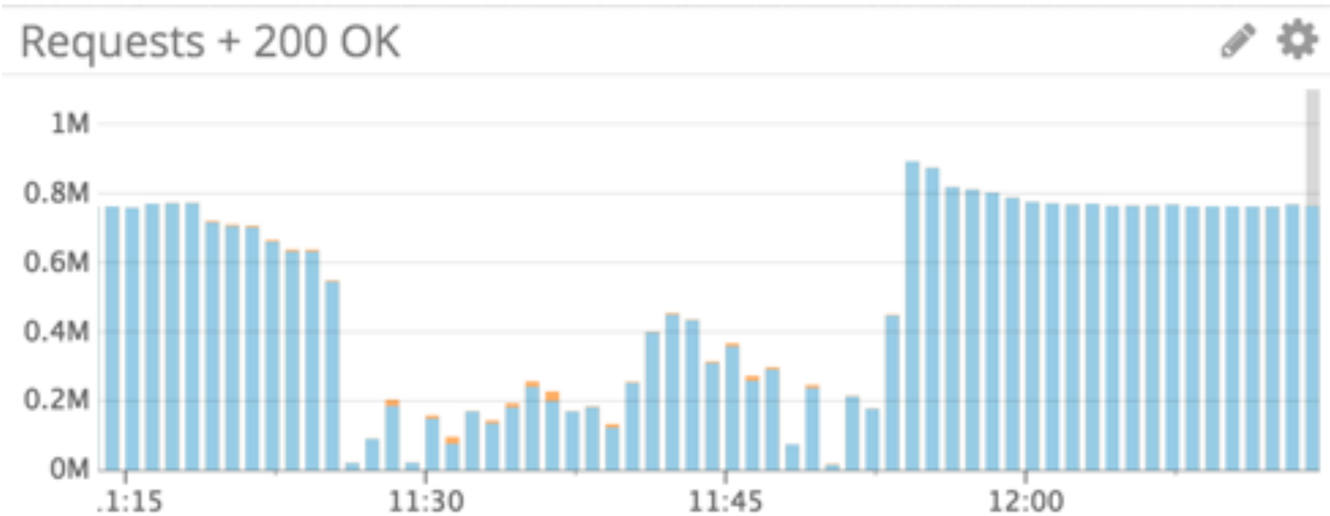
Max of spt.data.collector.kinesis.request_time over ...
20K
15K
10K
5K
0K
11:12
11:13
11:14
11:15
11:16
11:17
11:18
11:19

Queue grows

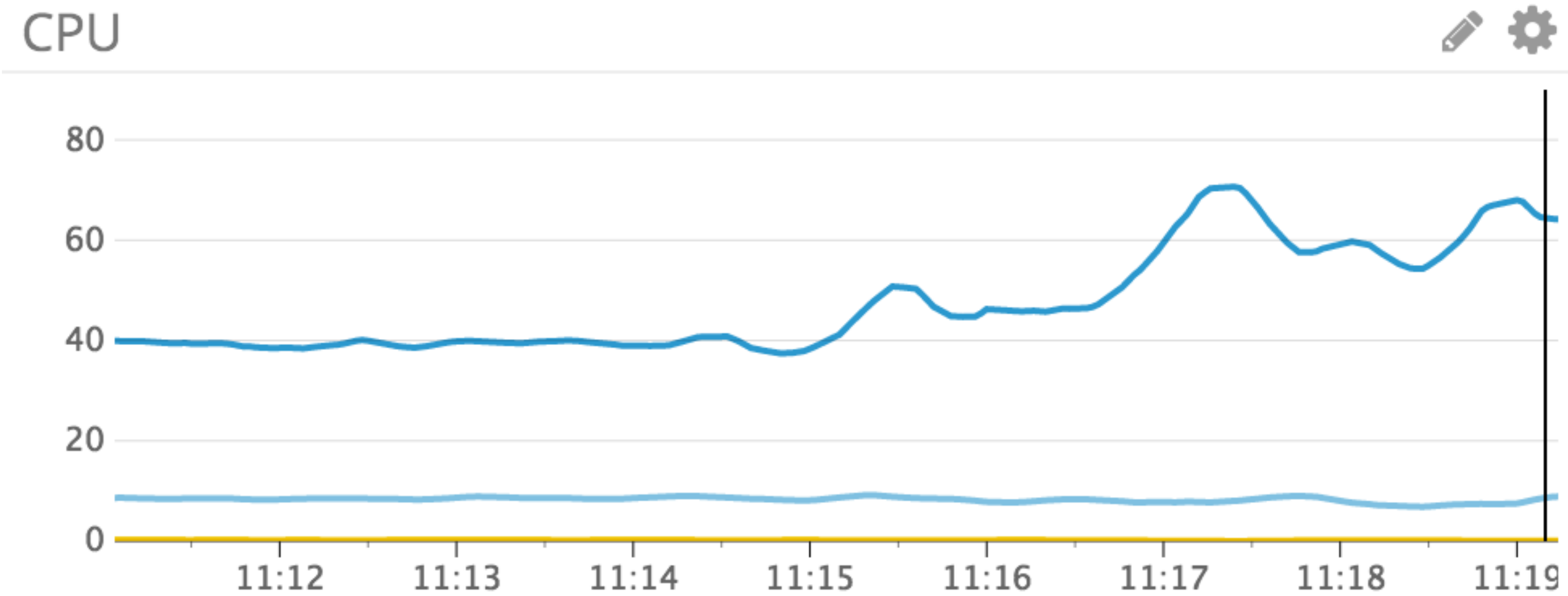

Batch & queue size
80K
60K
40K
20K
0K
11:12
11:13
11:14
11:15
11:16
11:17
11:18
11:19

Kaboom

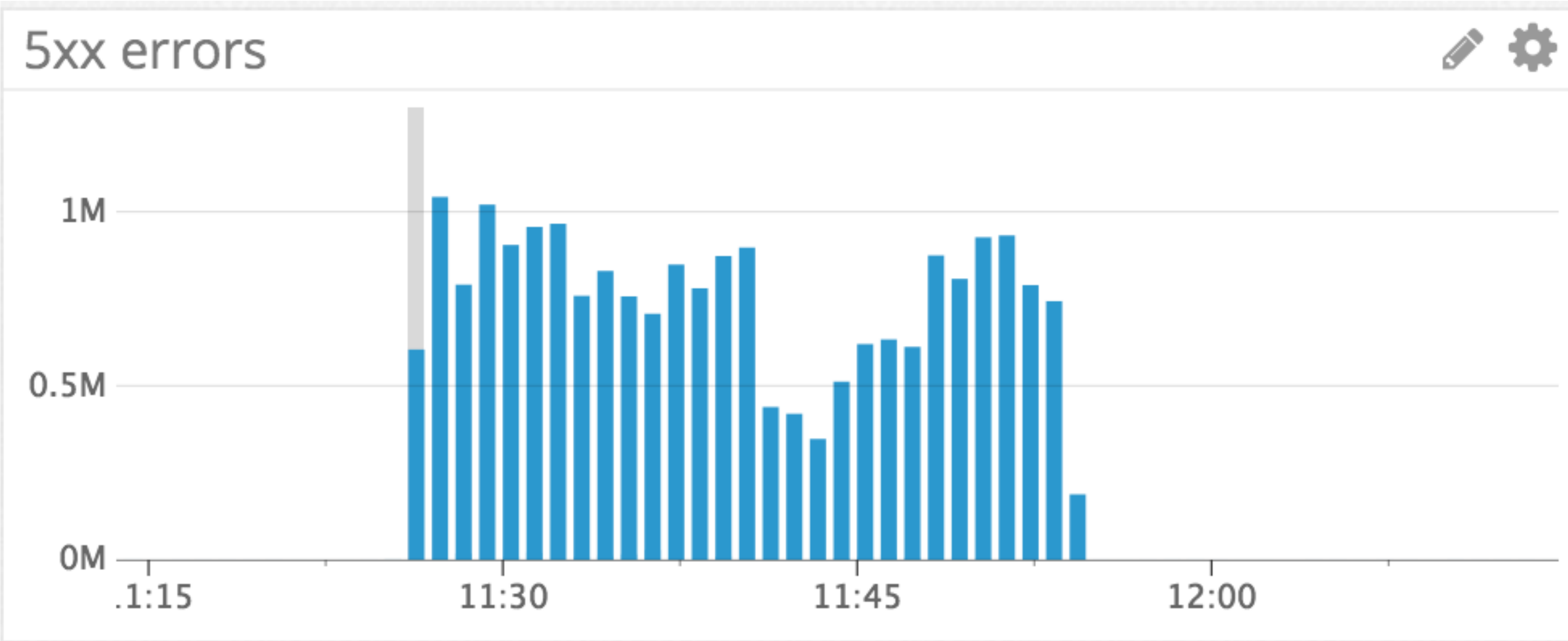
SCHIBSTED
MEDIA GROUP



GC eats the CPU



Failure to recover



2016-01-06 Data collector outage

Created by Lars Marius Garshol, last modified on Feb 04, 2016

Summary

The data collector was partially unresponsive for about 40 minutes, causing about 12 million records to be lost. The cause was a temporary increase in Kinesis write latency.

Background

In order to understand the incident it's helpful to have a minimal understanding of the data collector internals.

Incoming data collector requests have their JSON parsed and two fields inserted. The JSON object is then added to a list (a buffer), and the request is responded to.

A background thread takes the list (and replaces it with an empty one), then converts the list to Kinesis records and writes it to Kinesis. Once Kinesis responds satisfactorily, the thread repeats, without sleeping.

High-level timeline

The detailed timeline is quite complex, so we start with a timeline focusing only on the most important events.

December 14, 2015

The data-collector AMI is built and deployed. It runs unchanged in production from this point on.

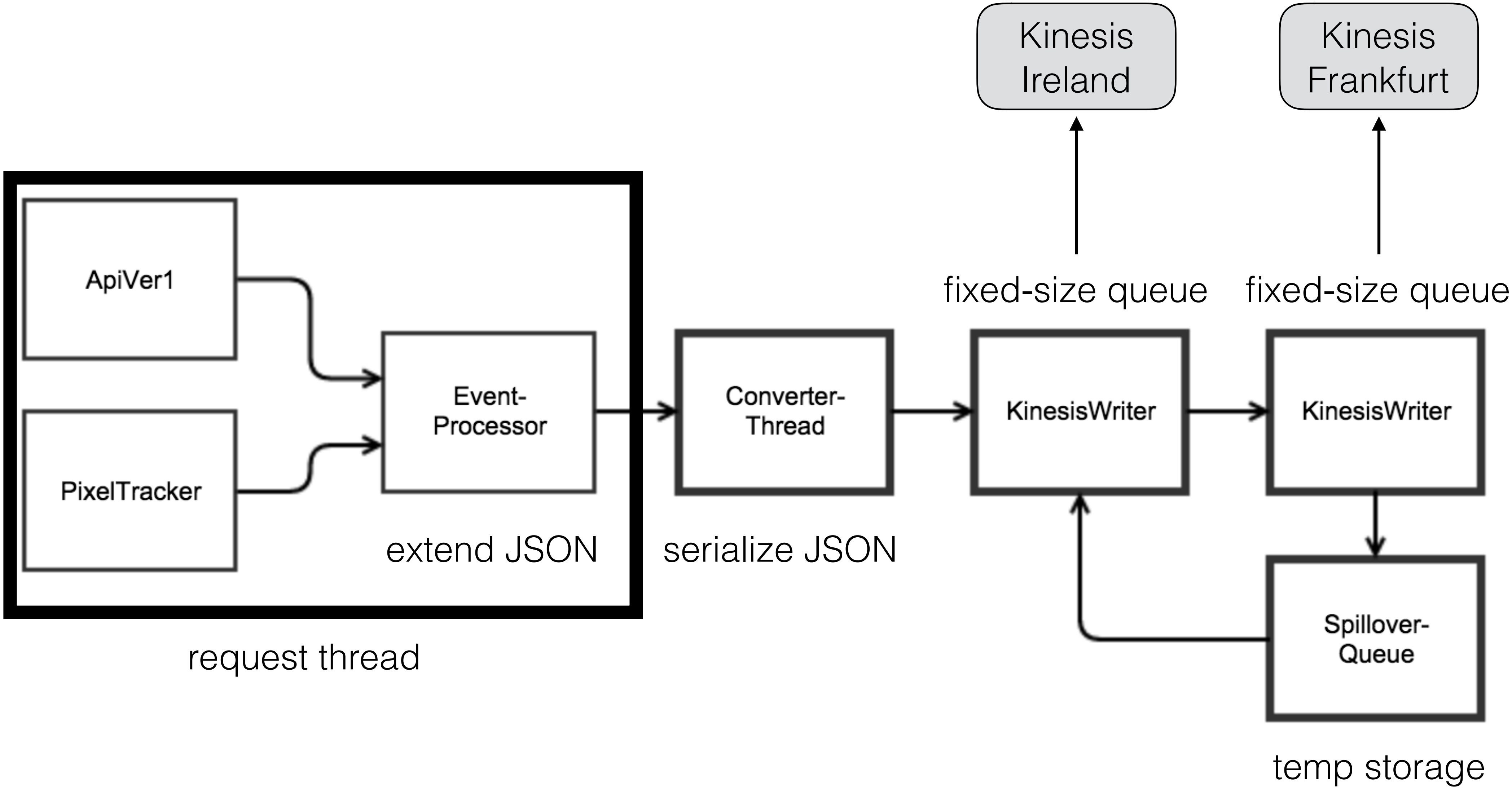
January 5, 2015

Auto-scaling group scaling policy is changed to scale down when the CPU usage is 20%, instead of the previous 15%.

January 6, 2015

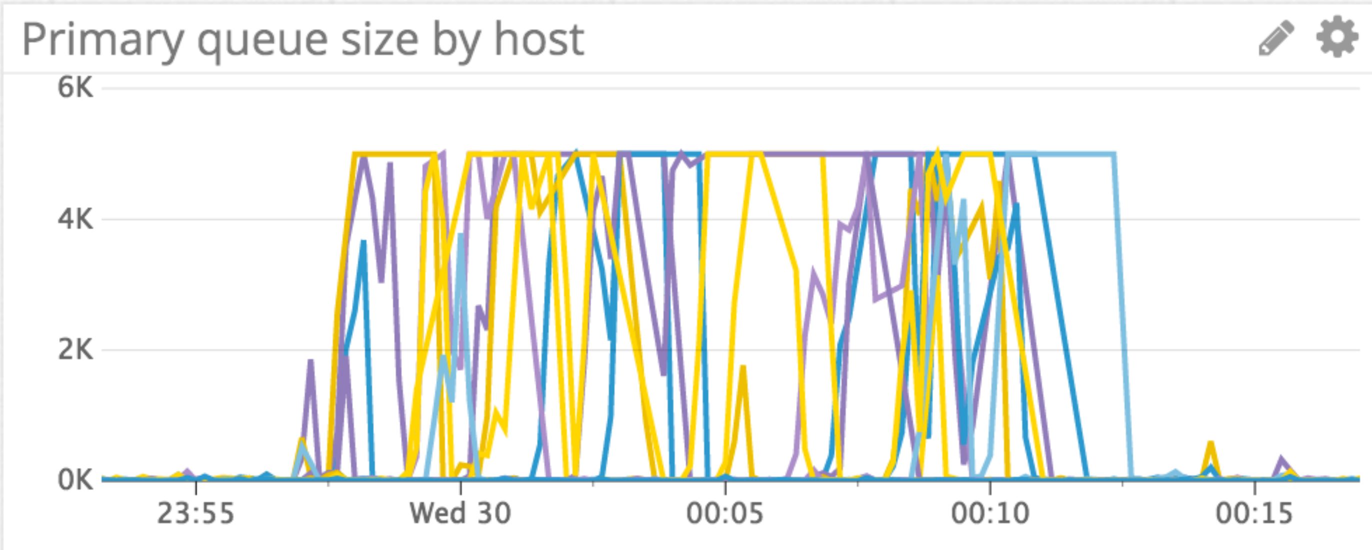
- **11:14** Kinesis request latencies spike dramatically, and events start queueing up within the nodes.

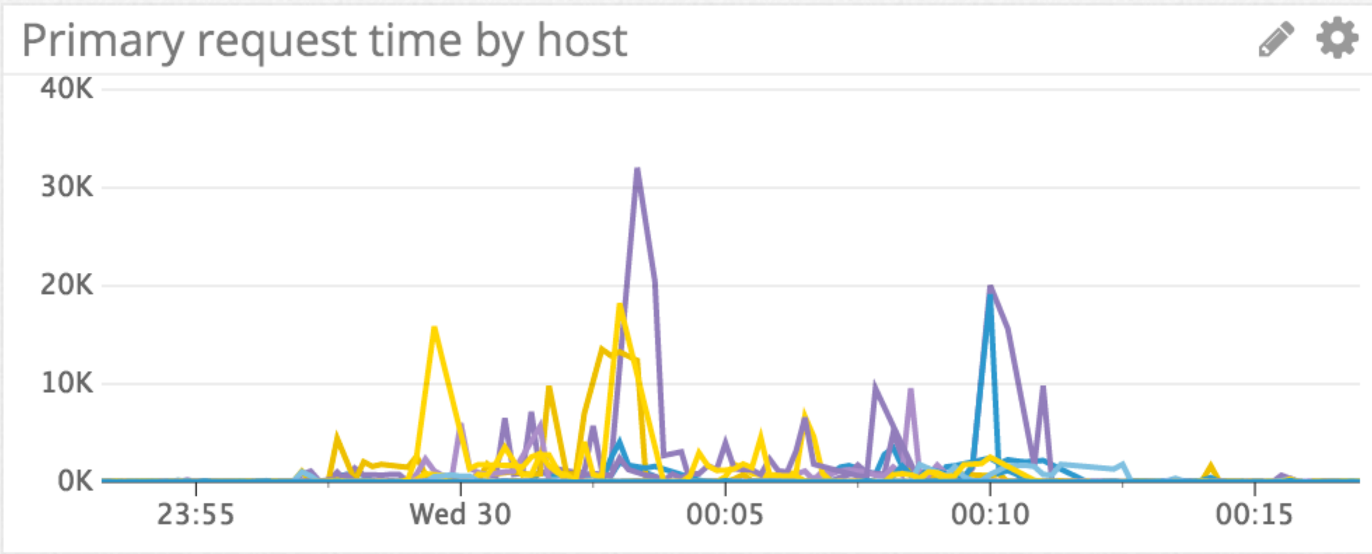
Redesign


Kinesis Ireland
Kinesis Frankfurt
fixed-size queue
fixed-size queue
ApiVer1
PixelTracker
Event-Processor
Converter-Thread
KinesisWriter
KinesisWriter
Spillover-Queue
extend JSON
serialize JSON
request thread
temp storage

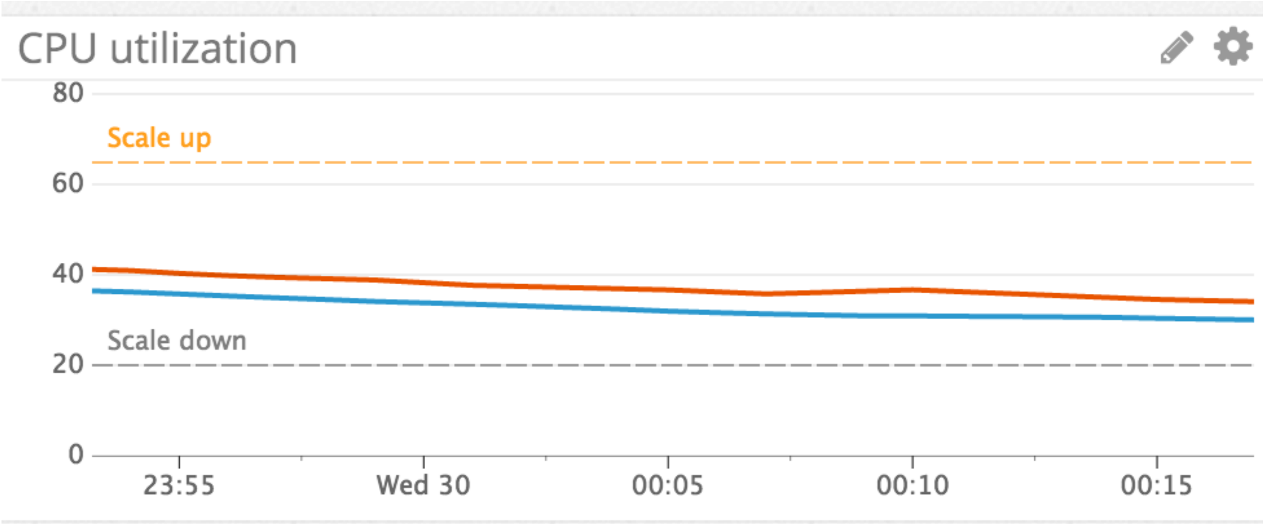
Latency spike!

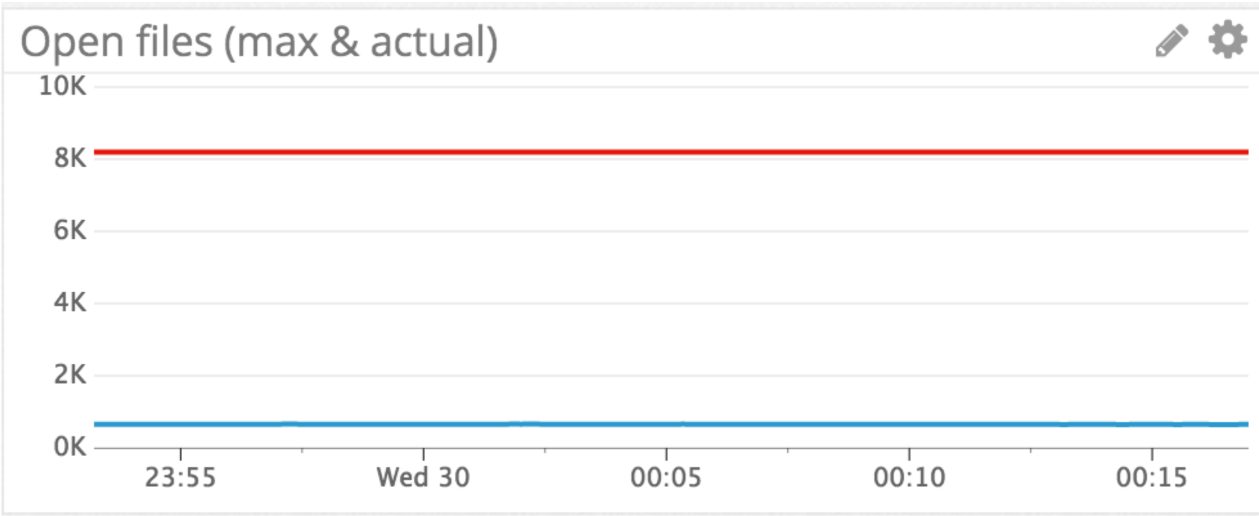
SCHIBSTED
MEDIA GROUP



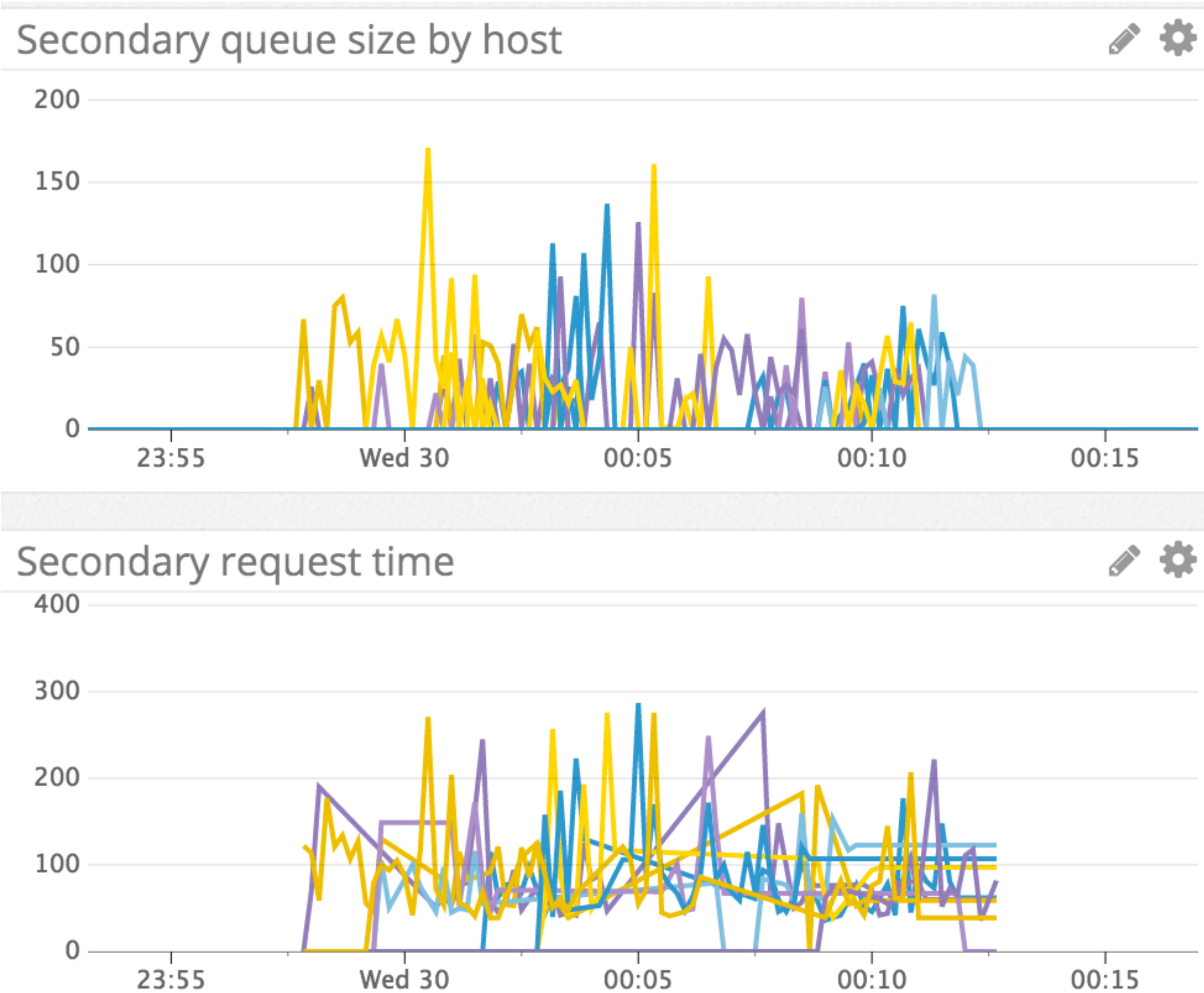


No increase in CPU



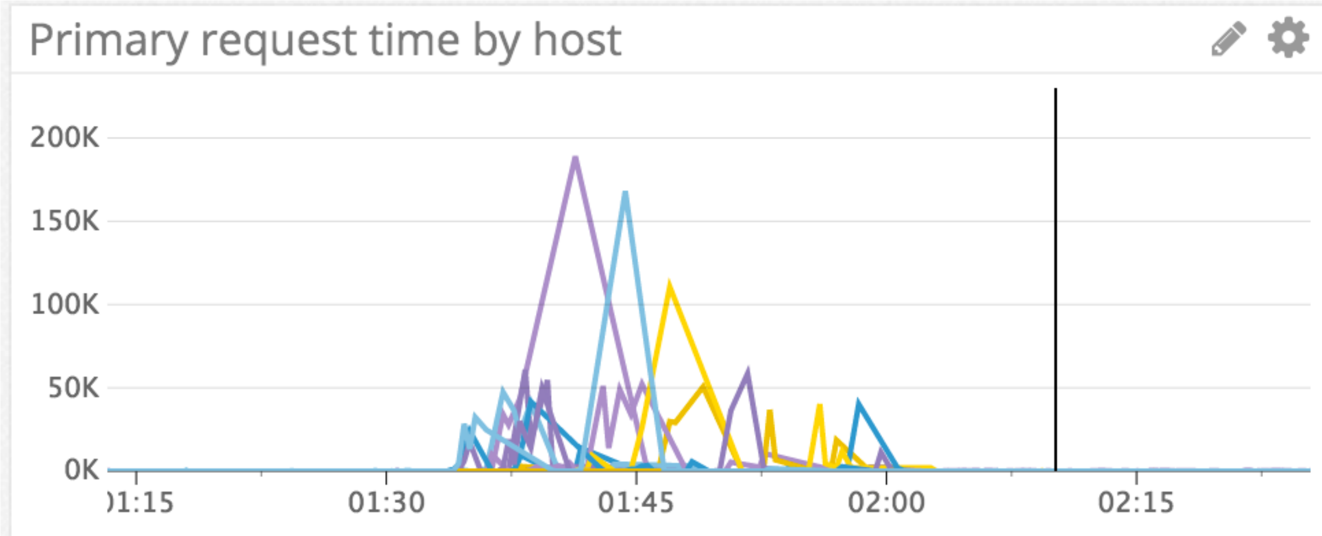


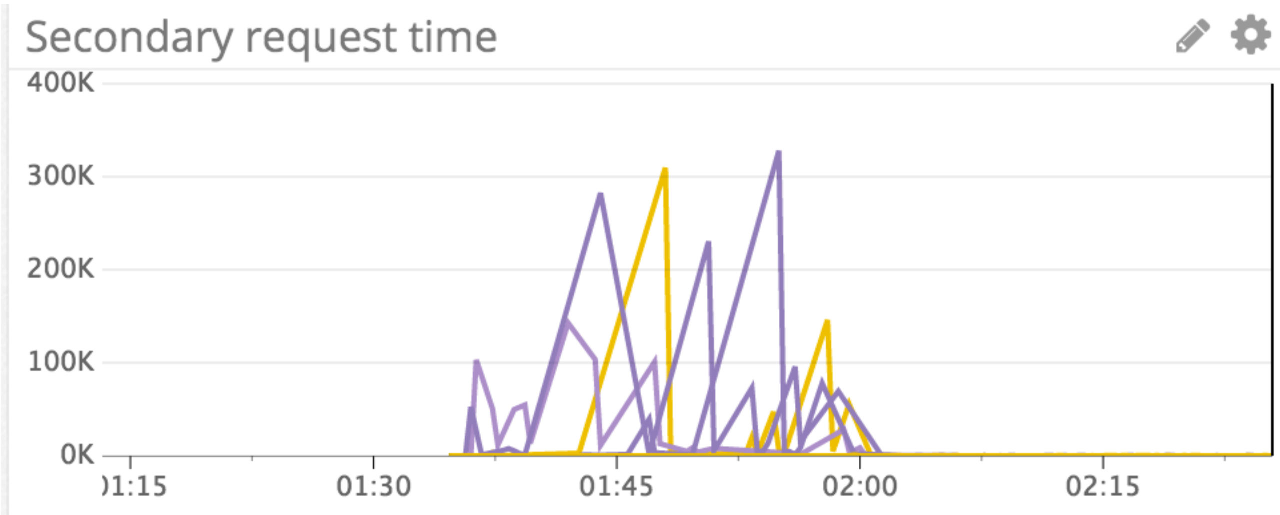
Frankfurt saves the day



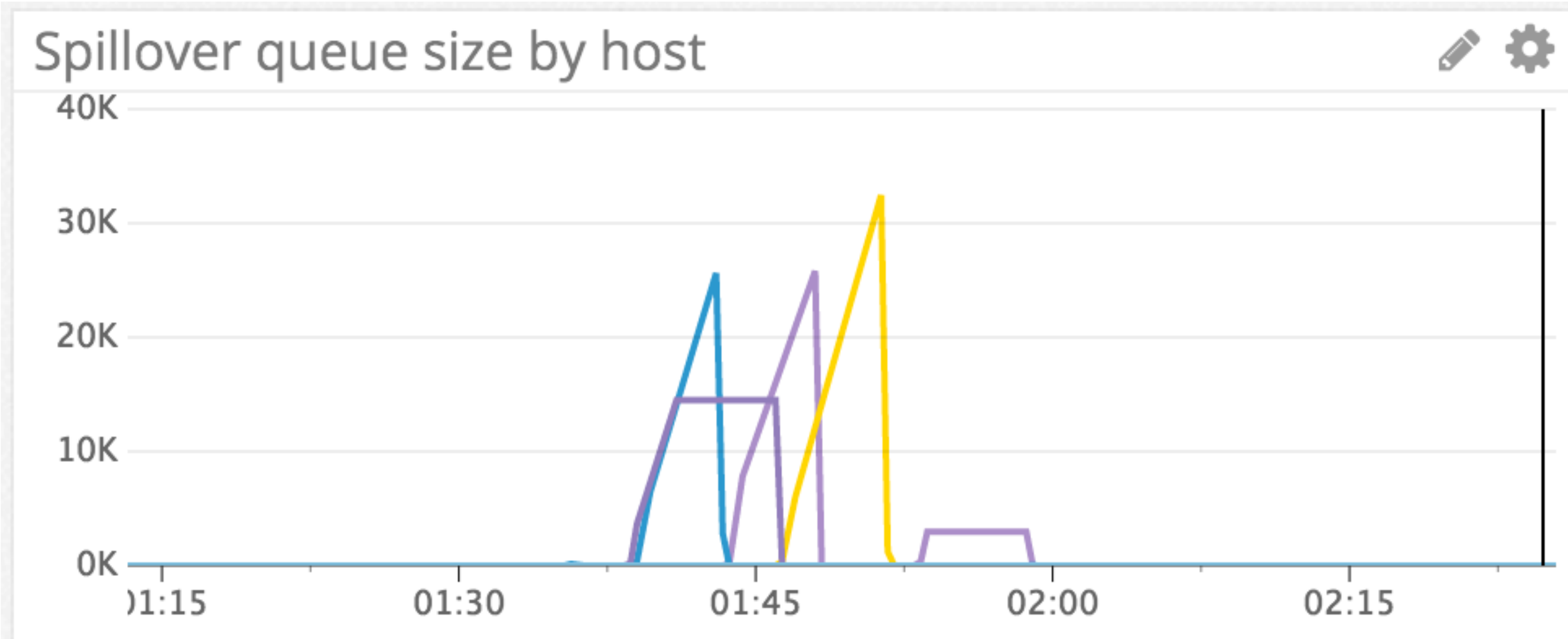
Network outage

Network out is slow

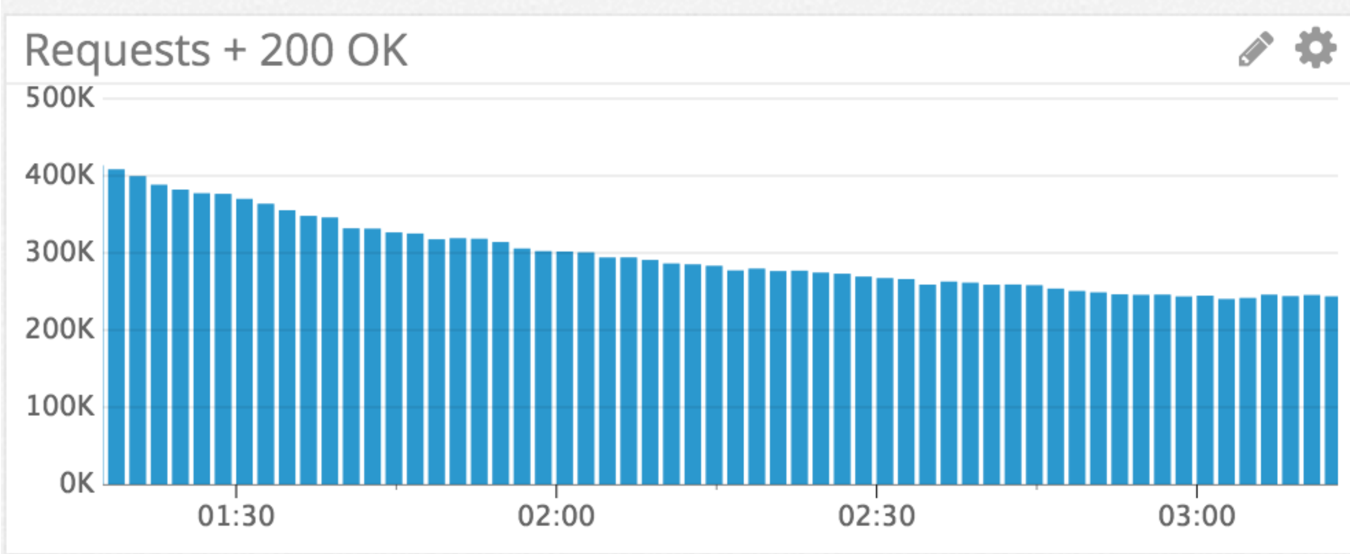


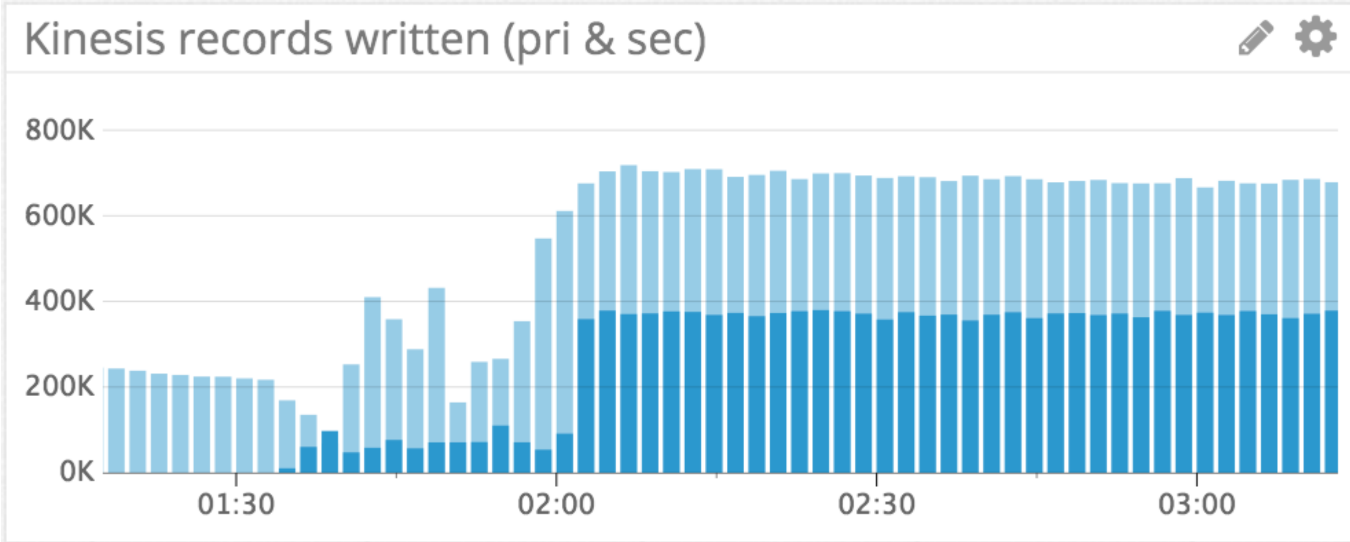


Events spill to disk



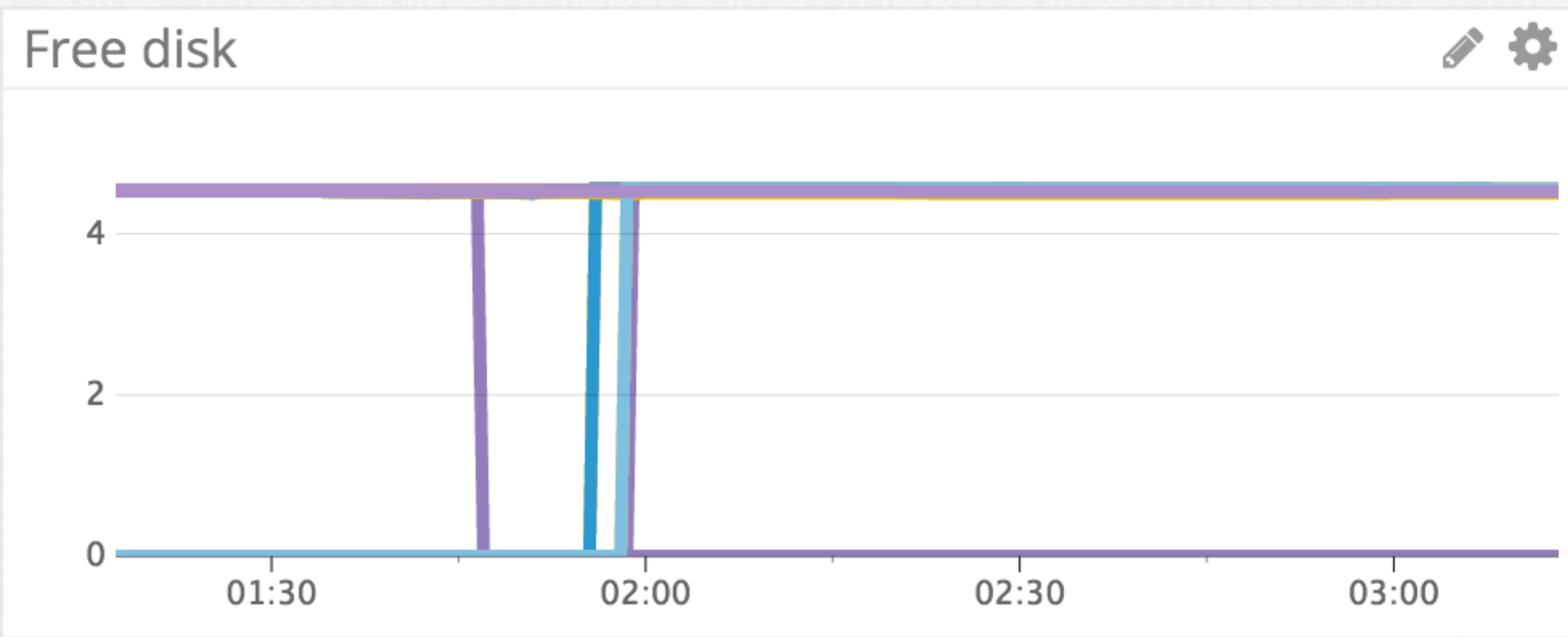
Application survives





Disk usage

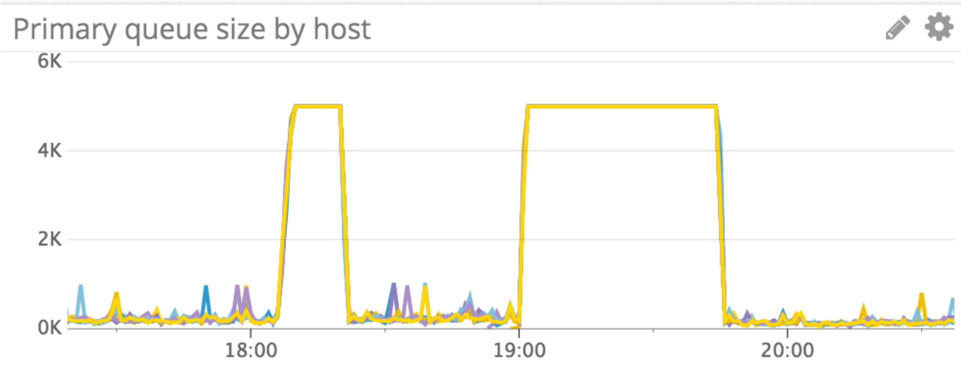
SCHIBSTED
MEDIA GROUP

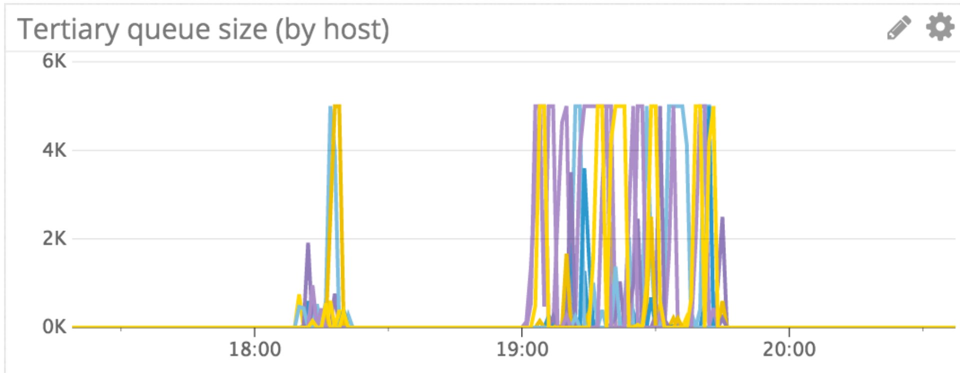


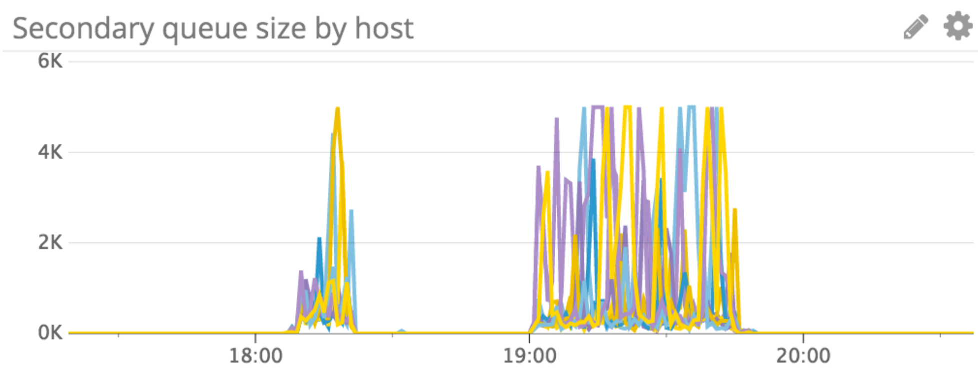
More rework

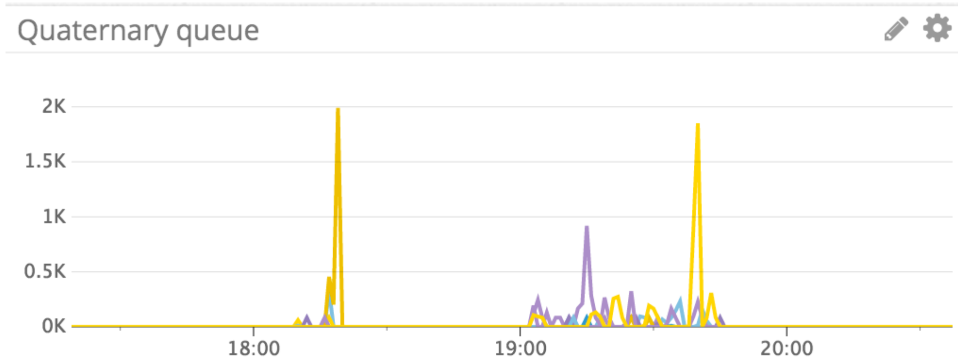
- It turned out that it was still possible for the two Kinesises to be too slow
  - more records would come in than we could write out
  - events spilling to disk, high latency
  - set off alarms and violate SLAs
  - solution: *four* writers, to two streams
- Networking issues causing absurd write latency
  - writes taking several minutes
  - solution: set a timeout of 15 seconds

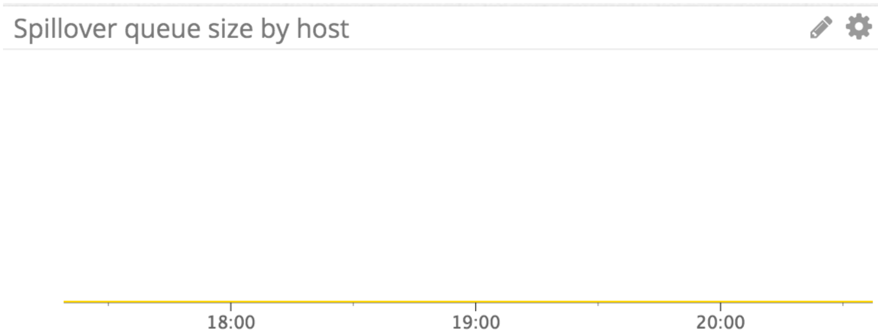
Nov 11 non-incident











Design flaws

- Collector shouldn't parse the JSON
  - this is a waste of CPU resources
- Collector should just pack JSON plus extra fields into some efficient serialization (Avro? Thrift? …)
  - then write to Kinesis
  - perhaps also gzip the data
- Let later stages deal with the tidying up
  - not done yet, because requires changes to several components
  - quite possibly also a custom Spark reader

Kinesis -> S3

Storage

- Very simple application
  - data in Kinesis lives 24 hours
  - therefore want something simple and fool-proof
- Stores all the data to S3
  - does nothing else
  - uses Kinesis Client Library (KCL) to read from Kinesis

Kinesis read limits

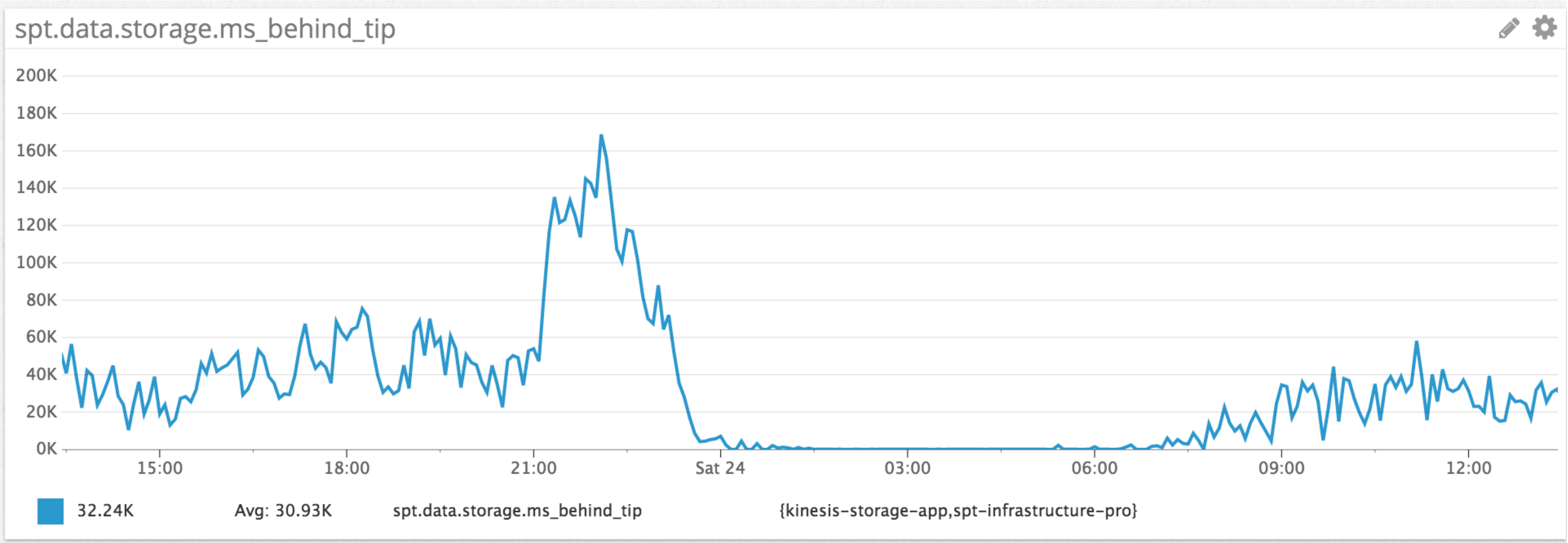
- KCL can give us max 10,000 records per read
  - but it never does
  - even if the stream contains many more records
- Experiment
  - write lots of 80-byte records into a test stream
  - then add lots of 2000-byte records
  - read the stream with KCL, observe results

Result

```
Wrote 10000 records (815960 bytes) in 2409 ms, 4 records/ms
Wrote 10000 records (816980 bytes) in 790 ms, 12 records/ms
Wrote 10000 records (816270 bytes) in 750 ms, 13 records/ms
Wrote 10000 records (817690 bytes) in 742 ms, 13 records/ms
Wrote 10000 records (817990 bytes) in 929 ms, 10 records/ms
Wrote 10000 records (817990 bytes) in 798 ms, 12 records/ms
Wrote 10000 records (819000 bytes) in 720 ms, 13 records/ms
Wrote 10000 records (816980 bytes) in 724 ms, 13 records/ms
Wrote 10000 records (817990 bytes) in 833 ms, 12 records/ms
Wrote 10000 records (818080 bytes) in 726 ms, 13 records/ms
Wrote 10000 records (818000 bytes) in 730 ms, 13 records/ms
Wrote 10000 records (818180 bytes) in 721 ms, 13 records/ms
Wrote 9535 records (6176176 bytes) in 2432 ms, 3 records/ms
Wrote 3309 records (6934426 bytes) in 1991 ms, 1 records/ms
Wrote 3309 records (6933172 bytes) in 1578 ms, 2 records/ms
Wrote 3309 records (6934878 bytes) in 1667 ms, 1 records/ms
Wrote 3310 records (6934916 bytes) in 1599 ms, 2 records/ms
Wrote 3309 records (6934319 bytes) in 1614 ms, 2 records/ms
Wrote 3309 records (6933975 bytes) in 2054 ms, 1 records/ms
```

Bigger records = fewer per batch

Falling behind

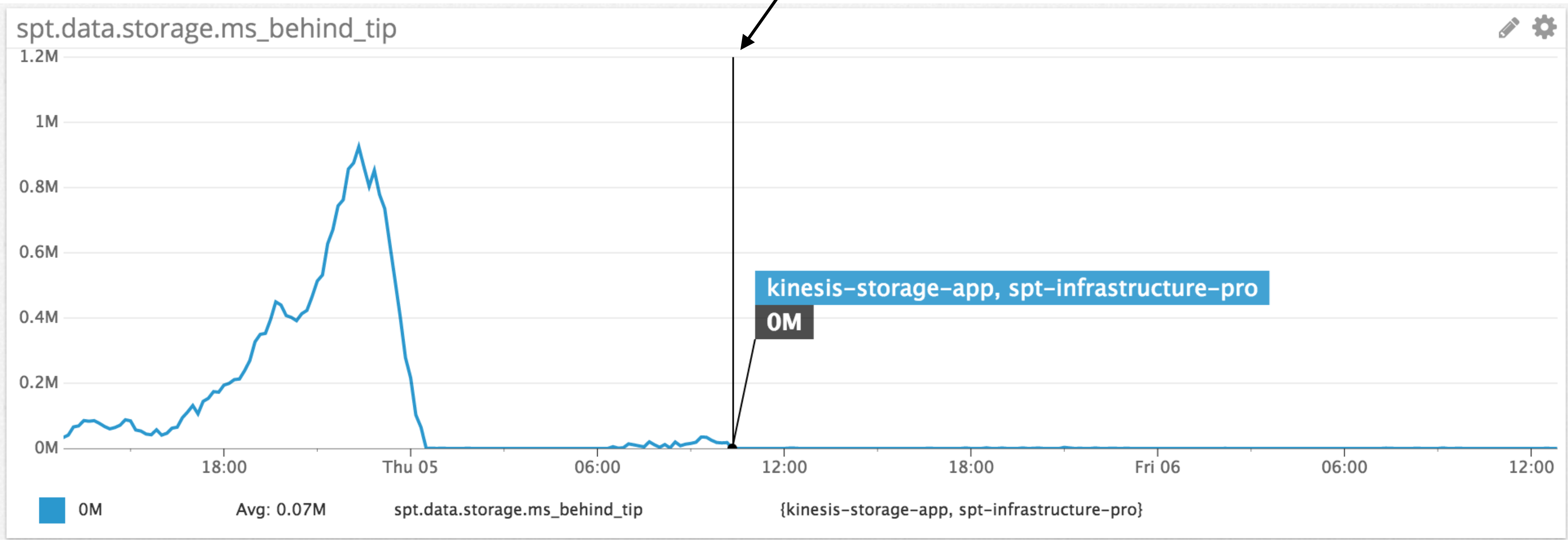


The relevant knob

- KCL has a setting for sleep between reads
- Used to have this at 10,000 ms
  - this in order to not get so many small JSON files
  - these are slow to read back out of S3
- As a result of this investigation, reduced to 5000ms
  - much later, reduced further to 3000ms
- Another knob is the number of shards

Results

New setting deployed



Analytics platform

- Analytics jobs are written in Apache Spark
  - *much* easier to write code for than Hadoop
  - also more efficient
- Used to be deployed on separate clusters
  - this was very expensive
  - now switching over to a shared cluster
  - this is somewhat painful, as we’re still learning

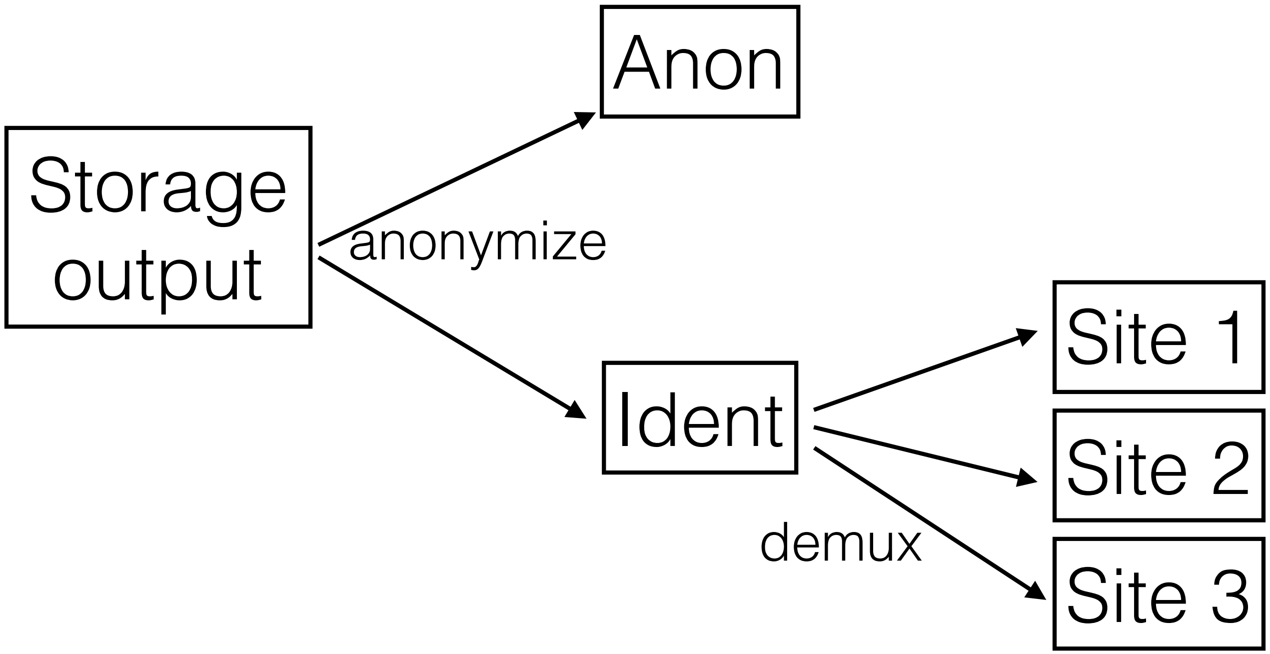
Spark

SCHIBSTED
MEDIA GROUP

```scala
val conf = new SparkConf().setAppName("basestats")
val sc = new SparkContext(conf)

try {
  implicit val sqlContext = new SQLContext(sc)
  val df = sqlContext.read.parquet(args(0))
  df.select("provider.@id")
    .map(id => (id, 1))
    .reduceByKey((a, b) => a + b)
    .saveAsTextFile(args(1))
} finally {
  sc.stop()
}
```

Dependencies



- A job scheduler developed by Spotify
  - use Python code to declare parameters, dependencies, and outputs
  - Luigi will schedule jobs accordingly
  - locking to ensure only one of each job running at a time
- Python code also for actually starting the job
  - many ready-made wrappers for known job types

```python
# this is our actual description
class BaseStatsTask(luigi.contrib.spark.SparkSubmitTask):
    date = luigi.DateParameter(default = yesterday())

    # for SparkSubmitTask
    entry_class = 'com.schibsted.spt.data.helpers.coalesce.BaseStats'

    # the jar file is in 'basestats-x.x.xx/jars/coalescer.jar', so we need
    # to compute that path
    thepath = glob.glob('basestats-*/jars/coalescer.jar')
    assert len(thepath) == 1
    app = os.path.abspath(thepath[0])

    def requires(self):
        # we need the anonymized events for each hour of the day
        return [AnonymizeEvents(self.date, hour) for hour in range(0, 24)]

    def output(self):
        return s3.S3FlagTarget(self._make_target_path(), client = client)

    # we assume SparkSubmitTask already implements run

    def app_options(self):
        # where we translate the Luigi parameters into the actual command-line
        # parameters of the job. a task that Luigi quite frankly ought to do
        # for us...
        return [make_input_date_path(self.date), self._make_target_path()]

    def _make_target_path(self):
        return ('s3://schibsted-spt-common-dev/lmg/basestats/'+
                make_date_suffix(self.date))
```

HIBSTED
IA GROUP

A Luigi task

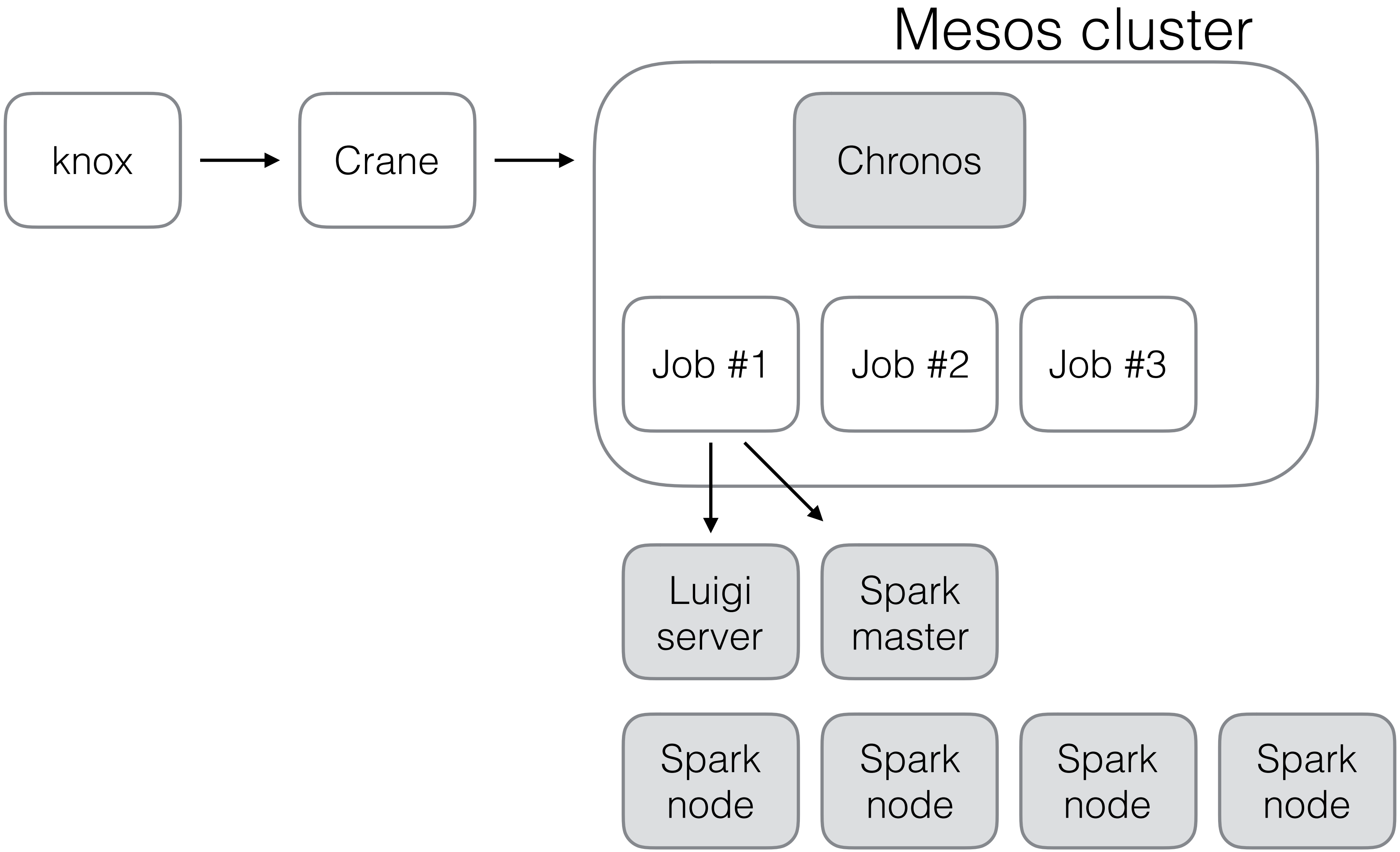
Luigi issues

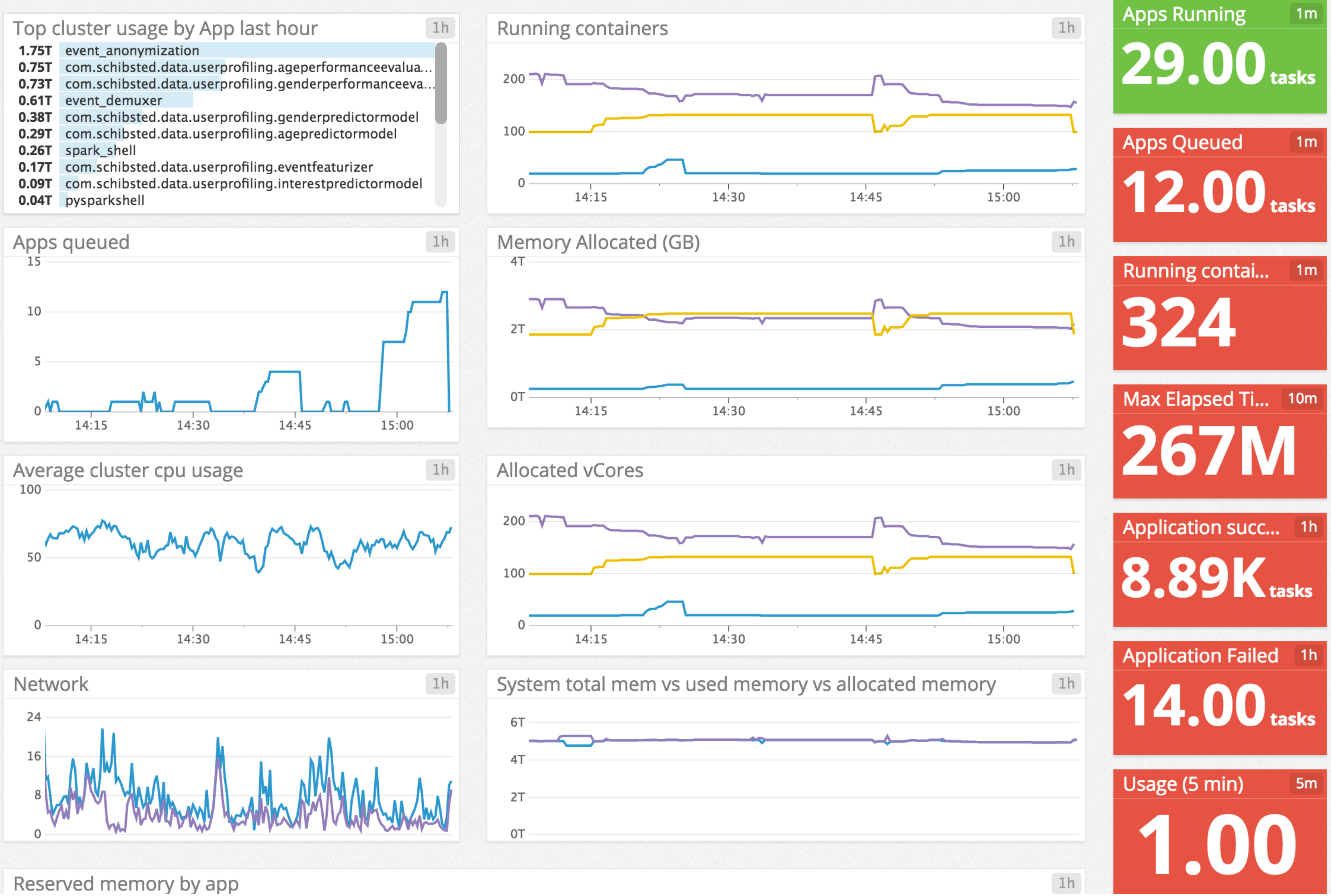
- No cron-like functionality
  - has to be handled by other means (like cron)
- Single master only
  - uses file locking on disk to prevent simultaneous execution of tasks
- No resource planning
  - it has no idea what resources are available
  - cannot queue jobs waiting for resources

knox

- Schibsted internal tool for working with data
- **knox job deploy**: deploy job in cluster
- **knox job status**: what’s the status of my job?
- **knox job kill**: stop running job
- **knox job disable**: don’t run this job again
- **knox job list**: what jobs exist?

Cluster architecture





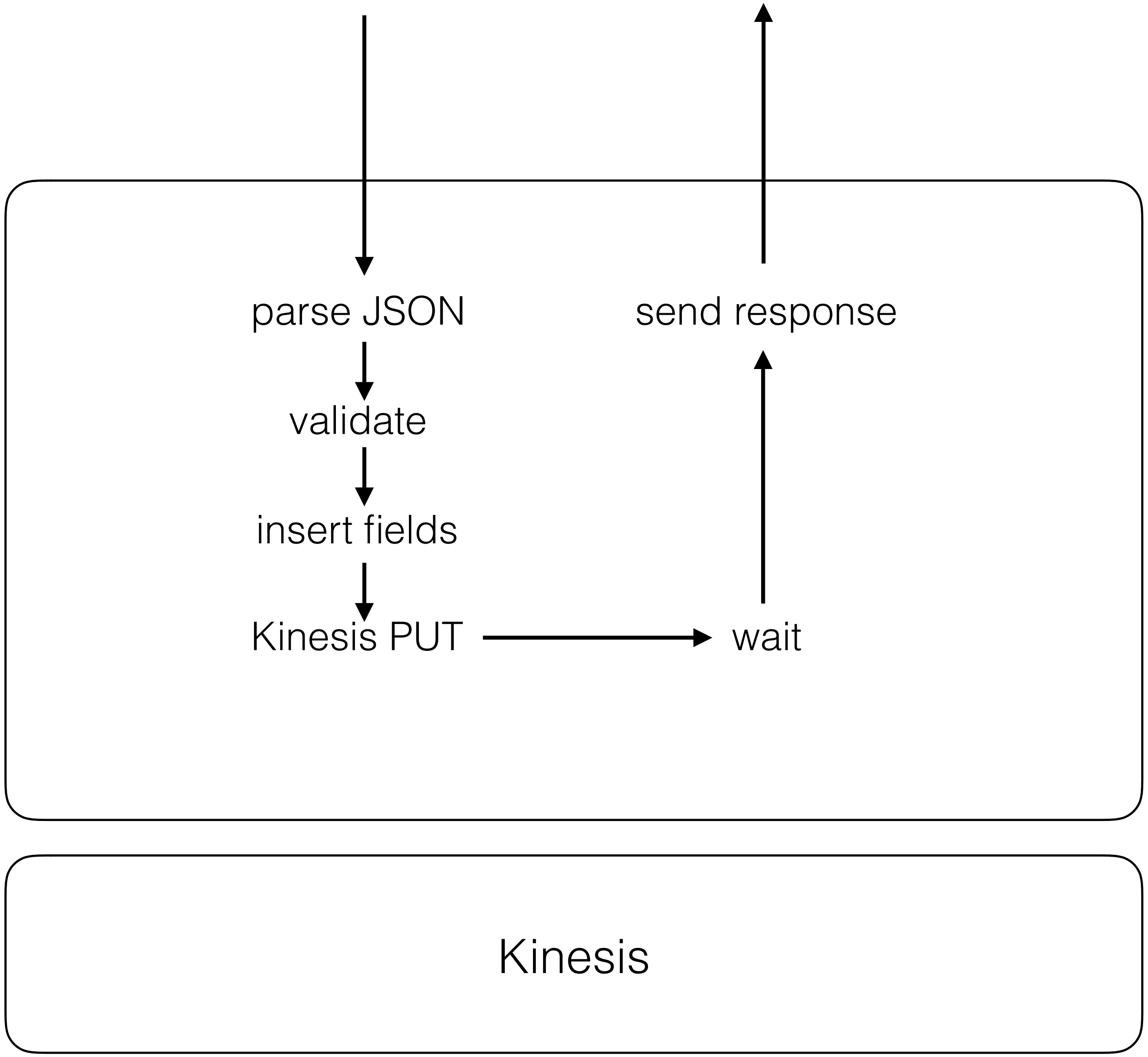
Top cluster usage by App last hour
1h
1.75T event_anonymization
0.75T com.schibsted.data.userprofiling.ageperformanceevalua...
0.73T com.schibsted.data.userprofiling.genderperformanceeva...
0.61T event_demuxer
0.38T com.schibsted.data.userprofiling.genderpredictormodel
0.29T com.schibsted.data.userprofiling.agepredictormodel
0.26T spark_shell
0.17T com.schibsted.data.userprofiling.eventfeaturizer
0.09T com.schibsted.data.userprofiling.interestpredictormodel
0.04T pysparkshell
Running containers
1h
Apps Running
1m
29.00 tasks
Apps Queued
1m
12.00 tasks
Apps queued
1h
Memory Allocated (GB)
1h
Running contai...
1m
324
Max Elapsed Ti...
10m
267M
Average cluster cpu usage
1h
Allocated vCores
1h
Application succ...
1h
8.89K tasks
Application Failed
1h
14.00 tasks
Network
1h
System total mem vs used memory vs allocated memory
1h
Usage (5 min)
5m
1.00
Reserved memory by app
1h

Synchronous events

New requirements

- Receive backend events from other applications
  - new ad, ad updated, …
- Can’t lose events
  - want to be able to tie business logic to them
- Must confirm event written to Kinesis with 200 OK
  - these clients can resend

Remember this?



Throttling

SCHIBSTED
MEDIA GROUP

```
try {
  openRequests++
  if (openRequests > MAX_SIMULTANEOUS_REQUESTS)
    response.status(509)
  else if (kinesis.sendToKinesis(request.content))
    response.status(200)
  else
    response.status(500)
} finally {
  openRequests--
}
```

openRequests never bigger than 2…

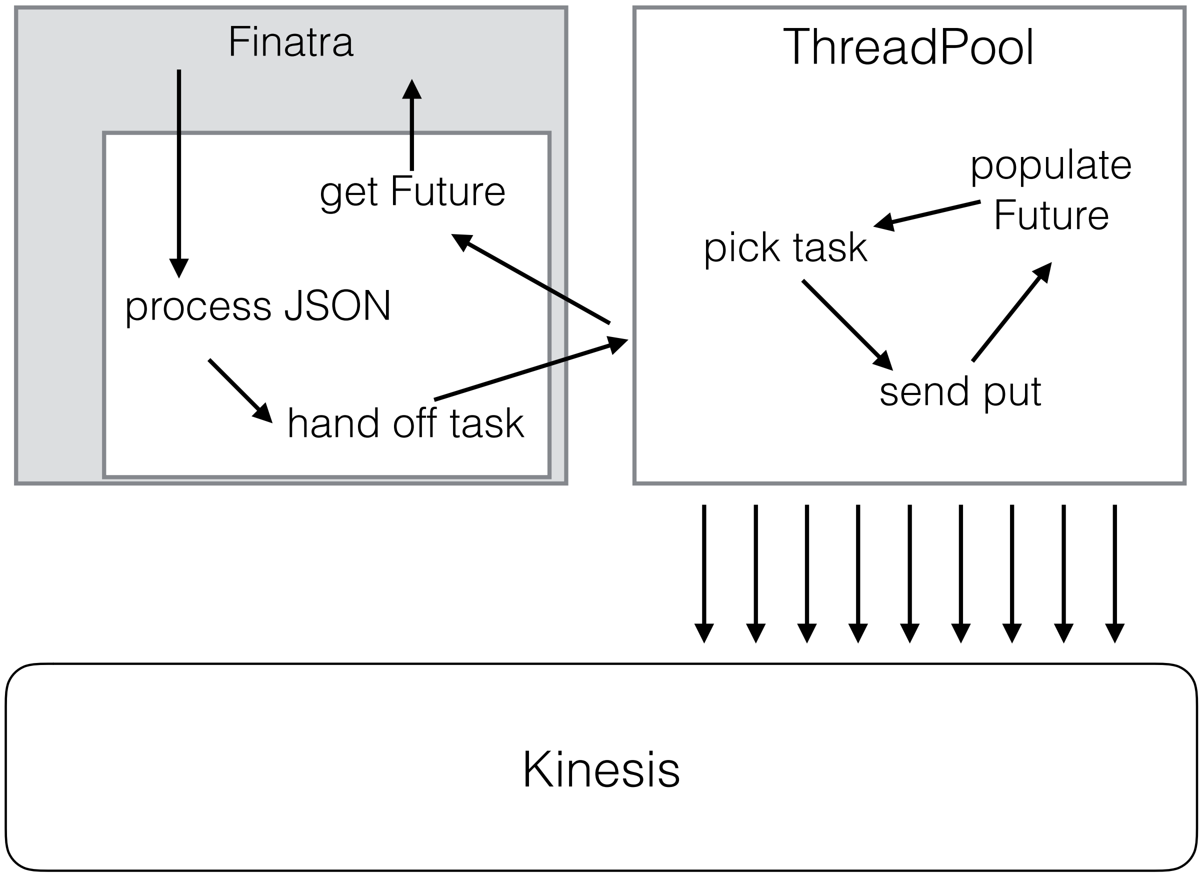
What's going on?

- Worker-based web servers
  - allocate a fixed number of worker threads/processes
  - each worker picks one request, finishes processing that, then picks the next request
  - have enough that while some may block on I/O there are always some threads making progress
- Event-based web servers
  - small number of worker threads
  - use polling interfaces to multiplex between connections
  - less context switching => more efficient

Finatra

- Event-based framework
  - response.status(200) doesn't return a Response
  - it returns Future[Response] that's already populated
- This means, if we're blocked we can return a Future[Response] that completes when we're done
  - allows Finatra to continue on another request that's not blocked

Canonical solution



Weaknesses

SCHIBSTED MEDIA GROUP

- Every event is a separate request to Kinesis
  - very inefficient
- Now suddenly we have a lot of threads again
  - back to the context-switching we were supposed to avoid
- Hard, low limit on the number of simultaneous requests
  - limit = number of threads

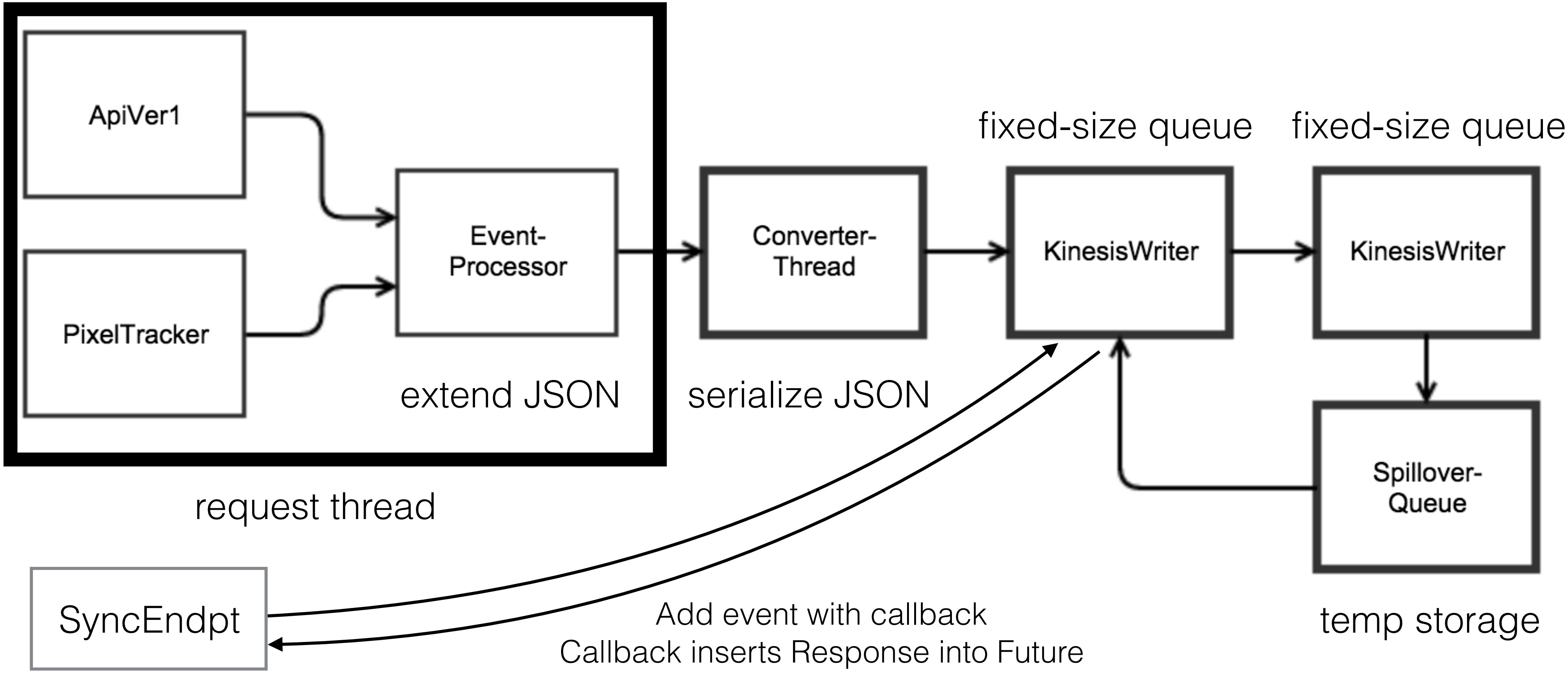
What's a Future?

```
public interface Future<V> {
  public boolean isDone()
    // true iff the value is there, false otherwise

  public V get()
    // loop until the value is there, then return it
    // if computing the value failed, throw the exception
}
```

Redesign

SCHIBSTED
MEDIA GROUP



Inside KinesisWriter

SCHIBSTED
MEDIA GROUP

Event	Future
JSON	
JSON	
JSON	Callback
JSON	
JSON	
JSON	
JSON	
JSON	Callback
JSON	
...	

Actual code

```
val promise = new Promise[Response]
kinesis.add(request.contentString.getBytes("utf-8"),
  // the writer will call this function with the outcome, which
  // causes Finatra to send the response to the client
  (success : Boolean) => if (success)
    promise.setValue(response.status(200))
  else
    promise.setValue(response.status(509))
)
promise
```

Benefits

- Number of threads is kept minimal
  - not all that context-switching
- Hard limit on requests is much higher (5000)
- No extra moving parts
- Synchronous requests sent together with other requests
  - much more efficient
  - much simpler

Winding up

Conclusion

SCHIBSTED
MEDIA GROUP

- Schibsted Tech is still only just getting started
  - the basics now in place
  - starting to generate money
  - a lot more work to do
- Use of AWS saves us from a lot of hassle
- Working at this scale
  - causes different challenges from what I'm used to
  - a lot more error handling/retries/scaling

SCHIBSTED
MEDIA GROUP

Back-End Software Engineer
OSLO ENGINEERING FULL-TIME

APPLY

Back-End Software Engineer
LONDON ENGINEERING FULL-TIME

APPLY

Back-End Software Engineer
BARCELONA ENGINEERING FULL-TIME

APPLY

CIO
OSLO, STOCKHOLM, LONDON, BARCELONA ENGINEERING FULL-TIME

APPLY

Data Scientist
BARCELONA ENGINEERING FULL-TIME

APPLY

Data Scientist
STOCKHOLM ENGINEERING FULL-TIME

APPLY

Data Scientist
LONDON ENGINEERING FULL-TIME

APPLY

Developer Relations Lead (Infrastructure)
BARCELONA ENGINEERING FULL-TIME

APPLY

DevOps Engineer
LONDON ENGINEERING FULL-TIME

APPLY

SCHIBSTED
MEDIA GROUP