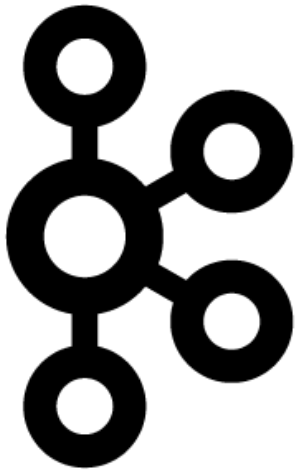


kafka

@

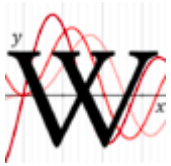
WIKIMEDIA FOUNDATION

November 2014

Introduction

"Imagine a world in which every single human being can freely share in the sum of all knowledge."

Introduction



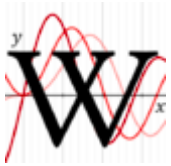
Andrew Otto

Systems/Operations Engineer at The Wikimedia Foundation working mainly on Analytics Infrastructure. (2012 - present)
http://www.mediawiki.org/wiki/User:Ottomata

Previously Lead SysAdmin at CouchSurfing.org (2008-2012)

http://linkedin.com/in/ottomata

Wikipedia is purty big, at least in total numbers of HTTP requests.



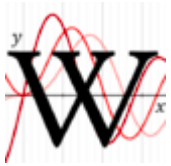
Wikipedia is the 5th largest website globally [*comScore*].

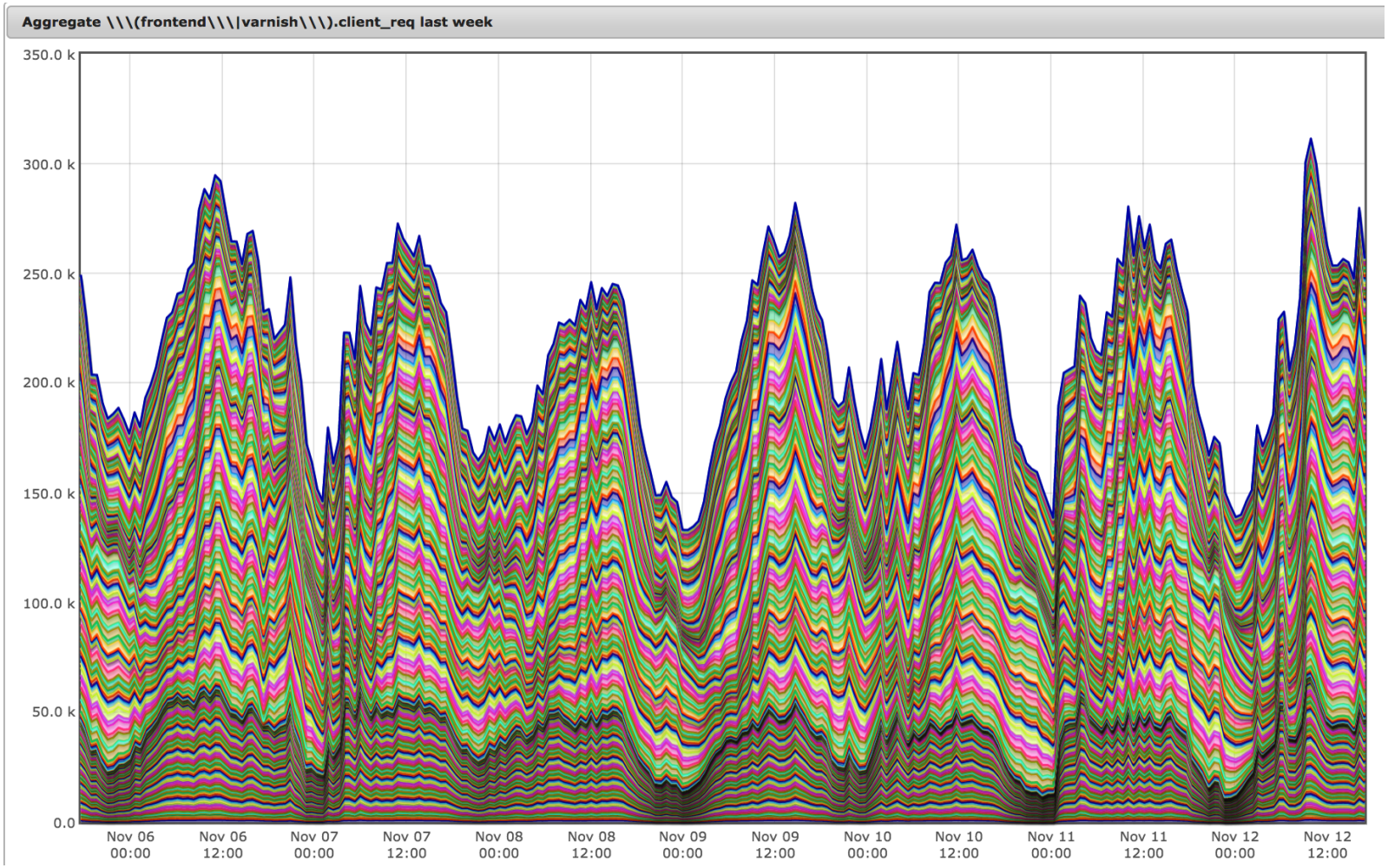
~500 million uniques / month

~20 billion pageviews / month

>200,000 HTTP requests / second (*at peak*)

WMF HTTP requests/second





Note: This graph is an overestimate of real HTTP requests due to annoying technical reasons, but you get the idea. :)

That's a lot of requests with a lot of yummy data.

How do we move it around?

Wait wait!

First, some history and a little Wikimedia architecture…

Data Sources

History

MediaWiki databases

webrequest logs

History

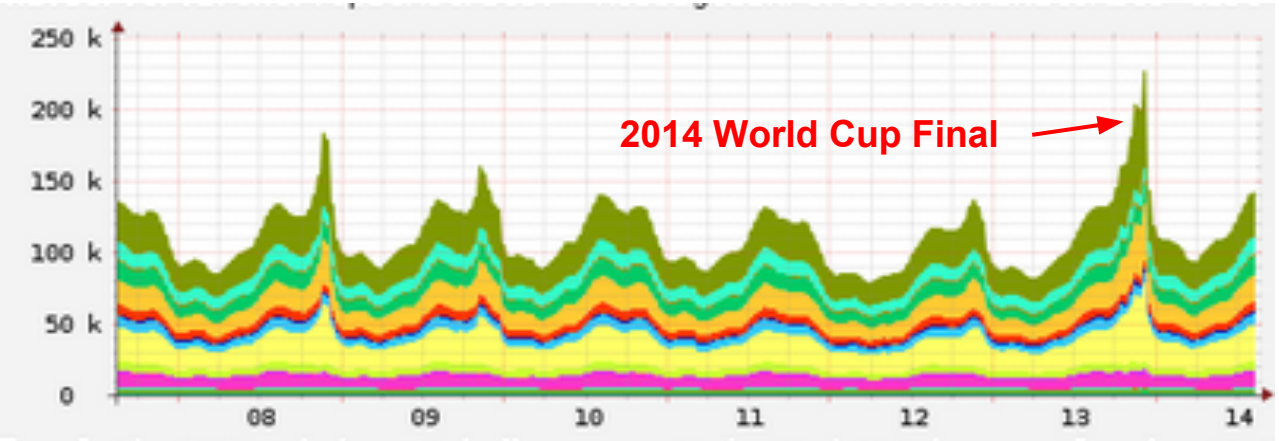
MediaWiki databases

Queryable slaves already available for analysts, this works (mostly) great!

webrequest logs

A log line for every WMF HTTP request.

This can max at > 200,000 requests per second.



History

Varnish

Webrequests handled by Varnish
in multiple datacenters.

Shared memory log

varnishlog apps can access in Varnish's logs in memory.

Varnishncsa

Varnishlog -> stdout formatter

Wikimedia patched this to send logs over UDP.

History

udp2log

History

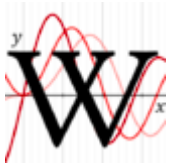
udp2log

Listens for UDP traffic stream.
Delimits messages by newlines.
Tees out and samples traffic to custom filters.

multicast relay

socat relay sends varnishncsa traffic to a multicast group, allowing for multiple udp2log consumers.

History



udp2log

works great but ...

History

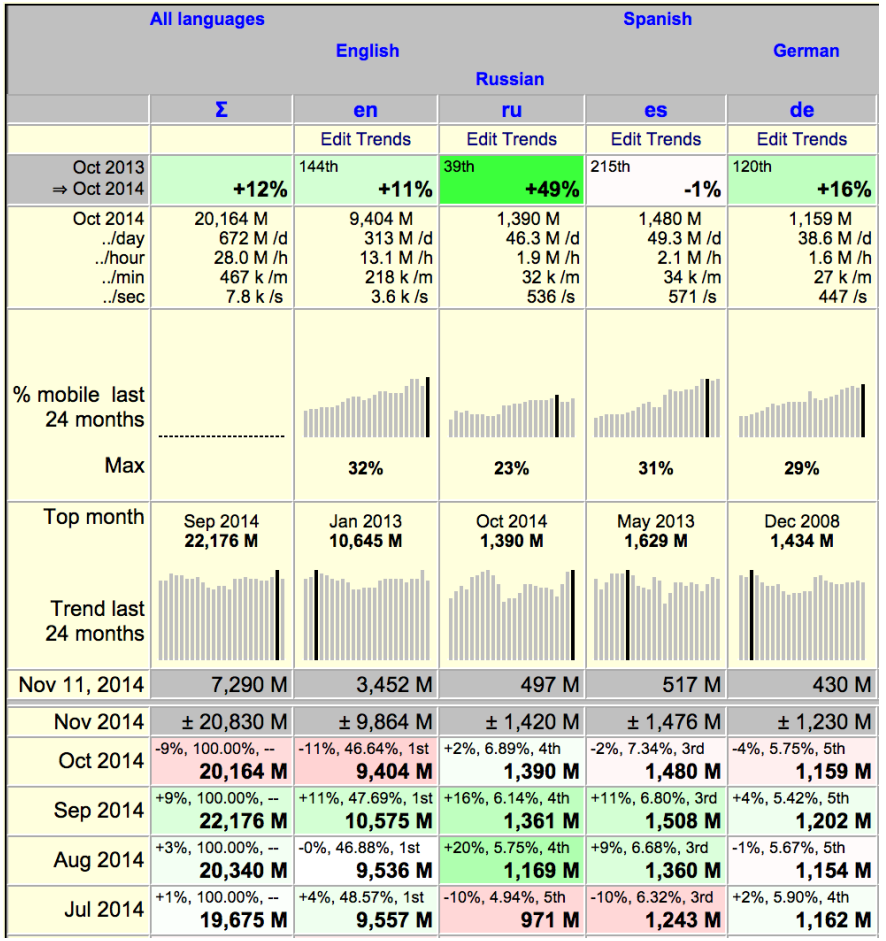
doesn't scale - every udp2log instance must see every network packet.

Works for simple use cases and **lower traffic** scenarios.

History

http://stats.wikimedia.org

udp2log
(and other)
sampled logs
saved and
post-processed
by analysts.



	All languages	English	Russian	Spanish	German
	Σ	en	ru	es	de
		Edit Trends	Edit Trends	Edit Trends	Edit Trends
Oct 2013 ⇒ Oct 2014	+12%	144th +11%	39th +49%	215th -1%	120th +16%
Oct 2014	20,164 M	9,404 M	1,390 M	1,480 M	1,159 M
../day	672 M /d	313 M /d	46.3 M /d	49.3 M /d	38.6 M /d
../hour	28.0 M /h	13.1 M /h	1.9 M /h	2.1 M /h	1.6 M /h
../min	467 k /m	218 k /m	32 k /m	34 k /m	27 k /m
../sec	7.8 k /s	3.6 k /s	536 /s	571 /s	447 /s
% mobile last 24 months					
Max		32%	23%	31%	29%
Top month	Sep 2014 22,176 M	Jan 2013 10,645 M	Oct 2014 1,390 M	May 2013 1,629 M	Dec 2008 1,434 M
Trend last 24 months					
Nov 11, 2014	7,290 M	3,452 M	497 M	517 M	430 M
Nov 2014	± 20,830 M	± 9,864 M	± 1,420 M	± 1,476 M	± 1,230 M
Oct 2014	-9%, 100.00%, -- 20,164 M	-11%, 46.64%, 1st 9,404 M	+2%, 6.89%, 4th 1,390 M	-2%, 7.34%, 3rd 1,480 M	-4%, 5.75%, 5th 1,159 M
Sep 2014	+9%, 100.00%, -- 22,176 M	+11%, 47.69%, 1st 10,575 M	+16%, 6.14%, 4th 1,361 M	+11%, 6.80%, 3rd 1,508 M	+4%, 5.42%, 5th 1,202 M
Aug 2014	+3%, 100.00%, -- 20,340 M	-0%, 46.88%, 1st 9,536 M	+20%, 5.75%, 4th 1,169 M	+9%, 6.68%, 3rd 1,360 M	-1%, 5.67%, 5th 1,154 M
Jul 2014	+1%, 100.00%, -- 19,675 M	+4%, 48.57%, 1st 9,557 M	-10%, 4.94%, 5th 971 M	-10%, 6.32%, 3rd 1,243 M	+2%, 5.90%, 4th 1,162 M

http://stats.wikimedia.org/EN/TablesPageViewsMonthlyCombined.htm

What we want

All requests saved for easy and fast analysis.

What we need

Scalable

log

transport


kafka

A **high throughput** distributed messaging system.

Apache Kafka

Distributed

Partitions messages across multiple nodes.

Reliable

Messages replicated across multiple nodes.

All Brokers are peers.

Performant

> 460,000 writes/second at LinkedIn [1]

> 2,300,000 reads/second

Kafka Terms

Broker

A Kafka Server.

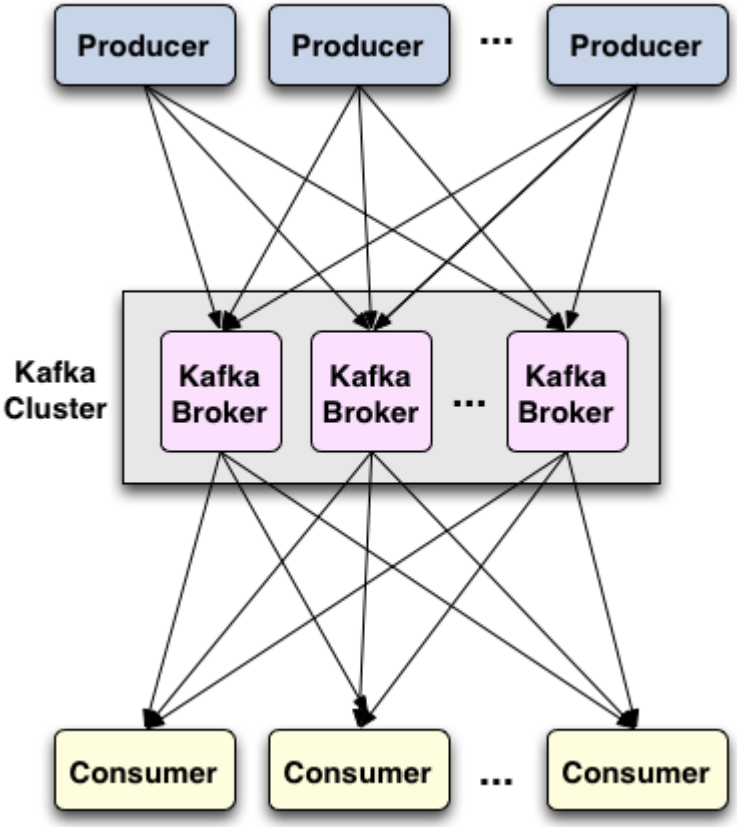
Producer

N producers send messages to Brokers.

Consumer

N consumers read messages from Brokers.

Apache Kafka



Kafka Terms

Topic

Logical delineation of messages.

Partition

Combined with topic, this is a physical delineation of messages.

Each topic is made up of N partitions.

Replication

Each partition will be replicated to N brokers.

Kafka Terms

Leader

Current broker in charge of a partition. All producers to a particular partition produce here.

Follower

A broker that consumes (replicates) a partition from a leader.

In Sync Replicas (ISR)

List of broker replicas that are up to date for a given partition. Any of these can be consumed from.

Analytics Cluster at Wikimedia

Hadoop for storage and **batch processing**

Hive tables for **easy SQL** querying of webrequest logs

Analytics Cluster at Wikimedia

Kafka

Camus

Hue

Hadoop

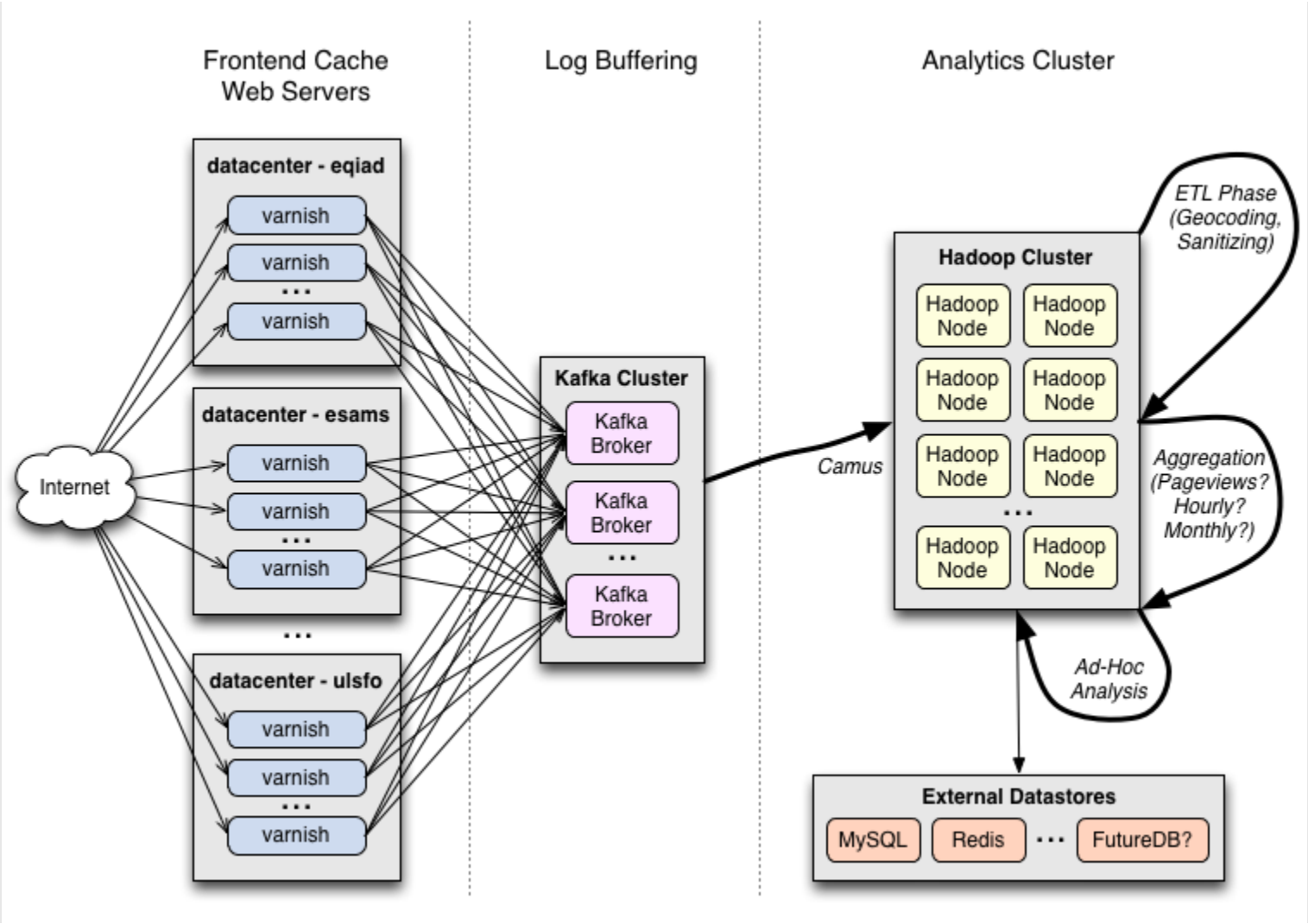
Hive

HDFS

Oozie

MapReduce

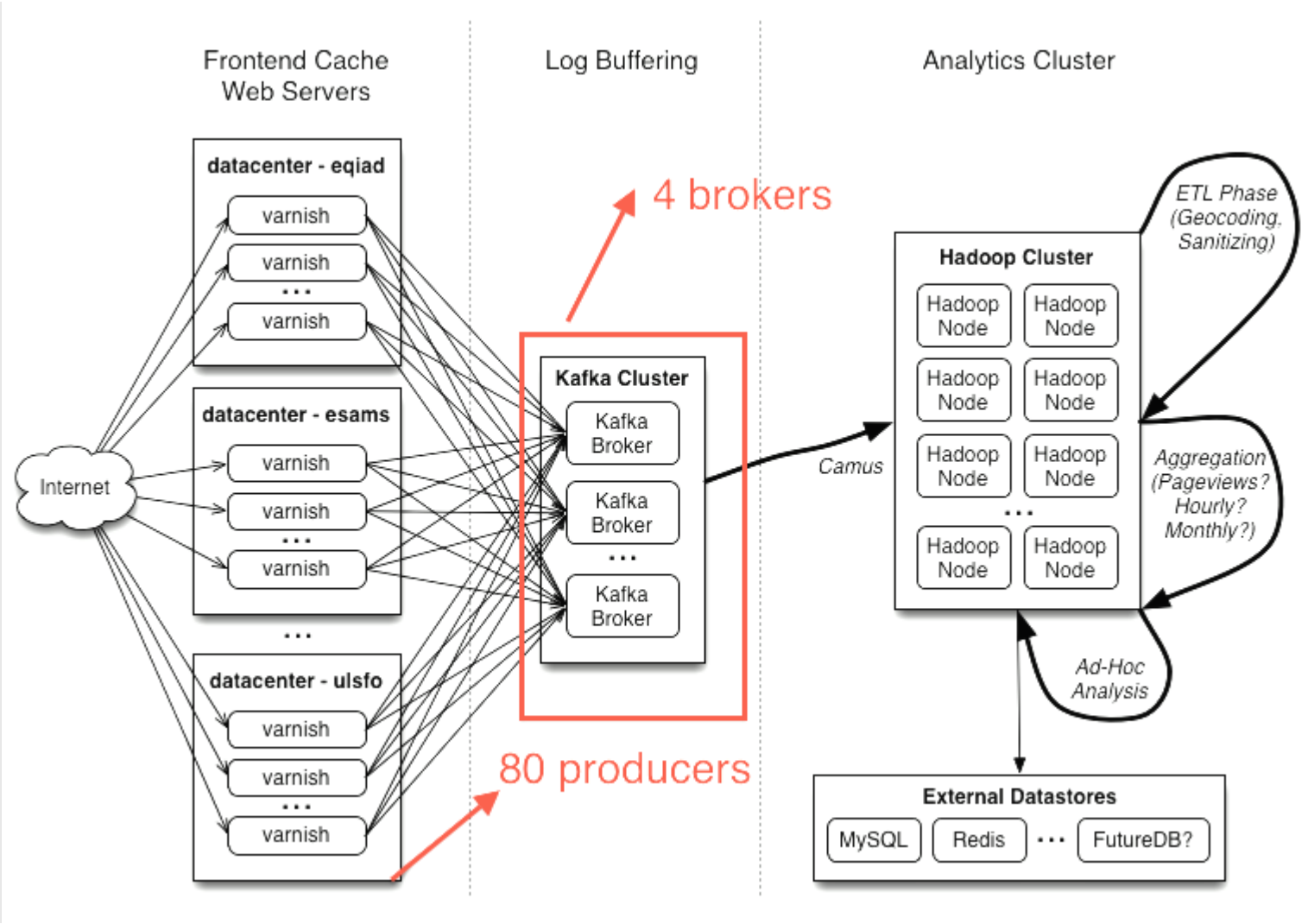
Analytics Cluster at Wikimedia



Kafka at Wikimedia

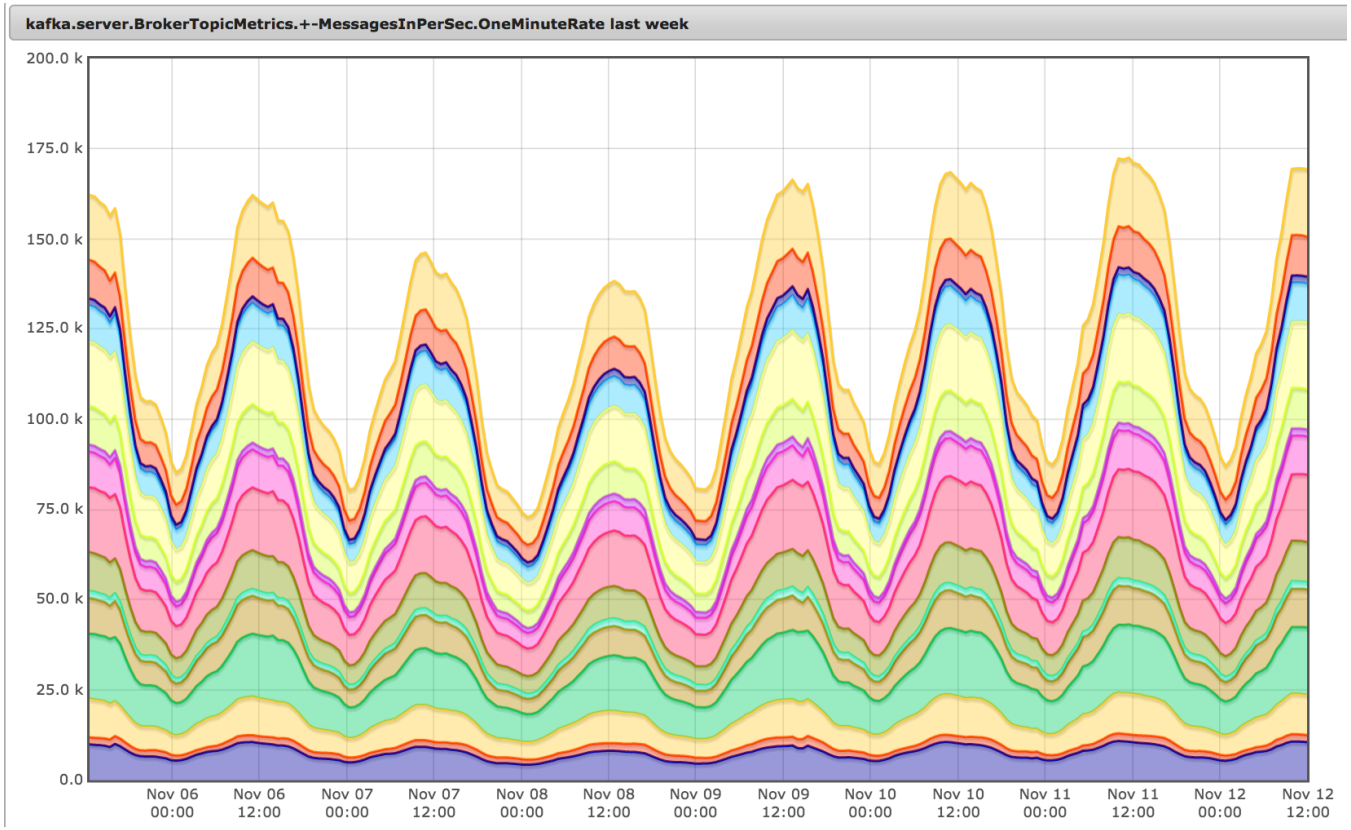
>200,000 messages per second | 30 MB per second, consumed **every 10 minutes into HDFS**

Kafka at Wikimedia

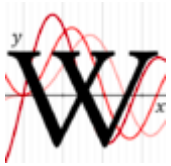


Kafka at Wikimedia - brokers

- 4 brokers, 4 (webrequest) topics, 12 partitions, replication factor = 3.



Kafka at Wikimedia - producer



Requirement from our ops team:

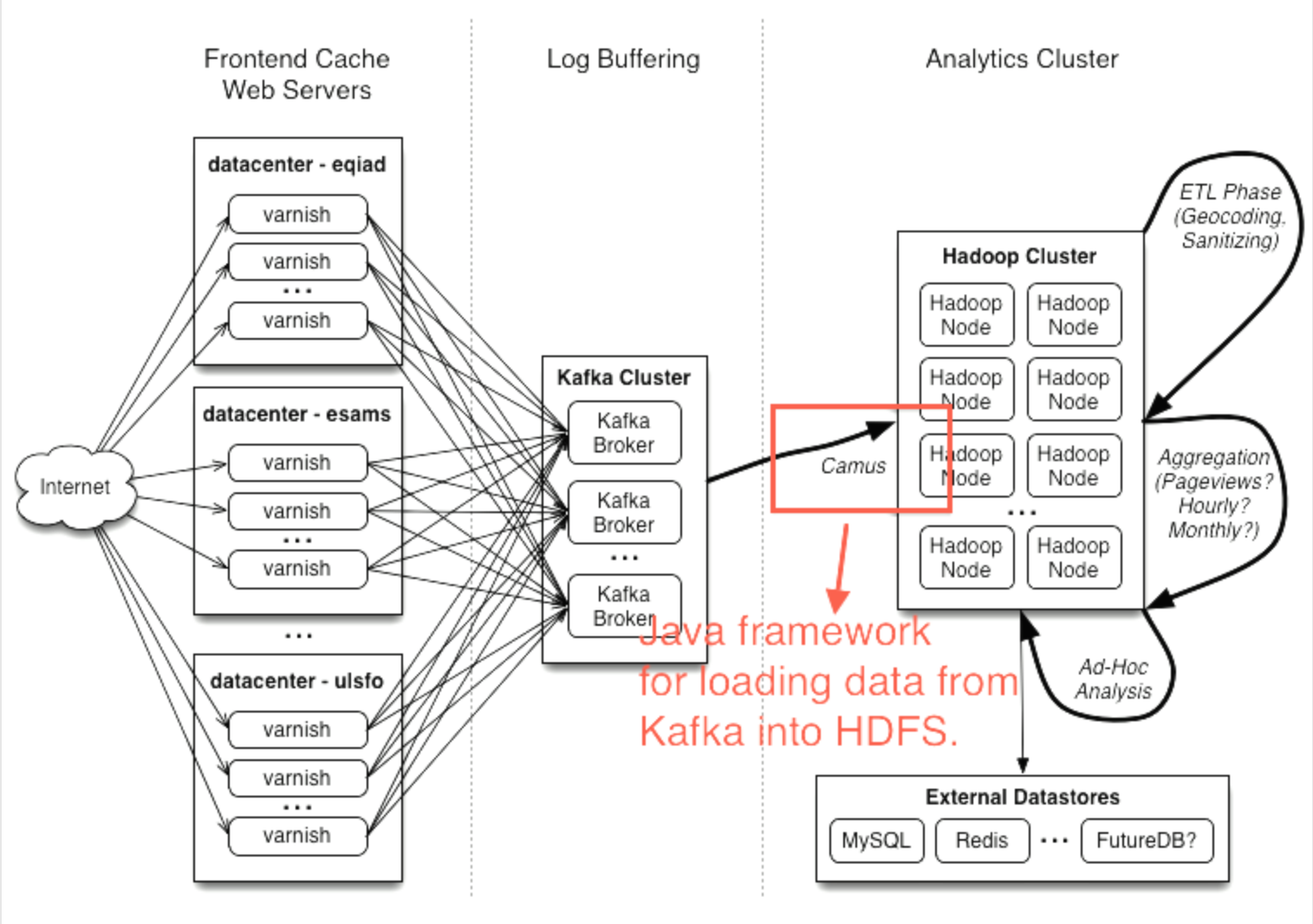
No JVM on frontend varnish nodes.

Producer: varnishkafka

We hired author of librdkafka (C client) to build varnishkafka.

Reads varnish shared logs, formats into JSON, and produces to Kafka brokers.

Kafka at Wikimedia - consumers



Kafka at Wikimedia - consumers

Consumer: Camus

- A MapReduce job to for distributed parallel loads of Kafka topics.

- Stores data in content based time bucketed data.

- e.g. A request from **2014-07-14 23:59:59** will be in ... **/2014/07/14/23**, and not accidentally in ... **/2014/07/15/00**.

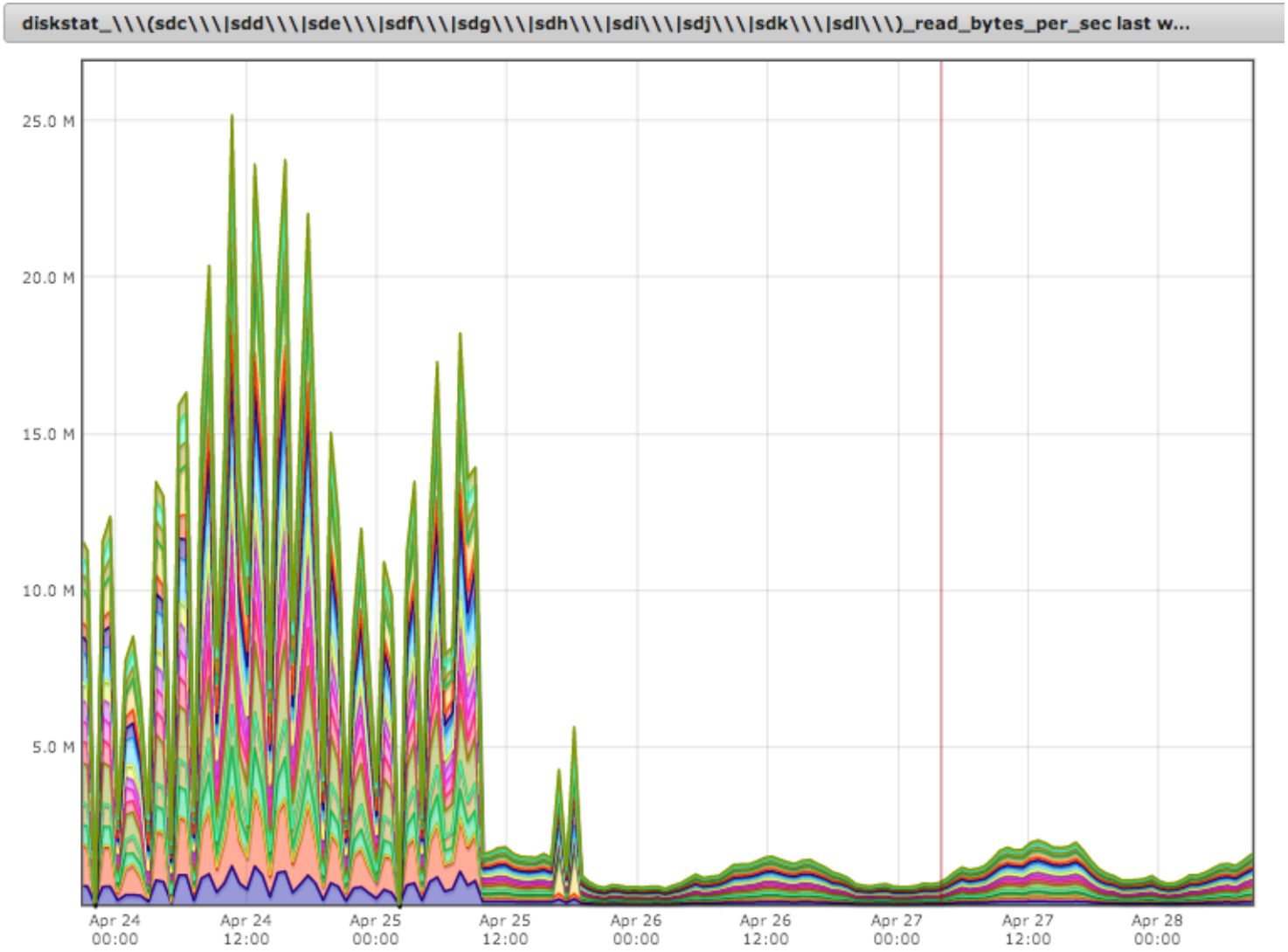
- Consuming more frequently is better for brokers — data more is more likely to be in memory if it was recently written (see next slide).

Kafka at Wikimedia - consumers

Broker disk bytes read per second.

Before:
Camus consuming every hour

After:
Camus consuming every 10 minutes



Kafka at Wikimedia - consumers

Consumer: kafkatee

Non-distributed process to:

- consume from multiple Kafka topics
- optionally sample
- optionally re-format (JSON -> tsv, etc.)
- output to multiple files and/or piped processes

Also written by author of **librdkafka**.

Kafka at Wikimedia - consumers

Consumer: kafkatee

```
output.format = %{hostname}   %{sequence}  %{dt}    %{time_firstbyte} \
%{ip}    %{cache_status}/%{http_status}    %{response_size} \
%{http_method}   http://%{uri_host}%{uri_path}%{uri_query}

input [encoding=json] kafka topic webrequest_upload \
partition 0-11 from stored

output file 1000 \
/srv/log/webrequest/sampled-1000.tsv.log

output pipe 10 /bin/grep -P 'zero=\d' \
>> /srv/log/webrequest/zero.tsv.log
```

Kafka at Wikimedia - Issues

Inter-datacenter production

- Works most of the time, but we do sometimes have problems with latency across the Atlantic Ocean, especially when link provider is not reliable.

Flaky Zookeeper connection

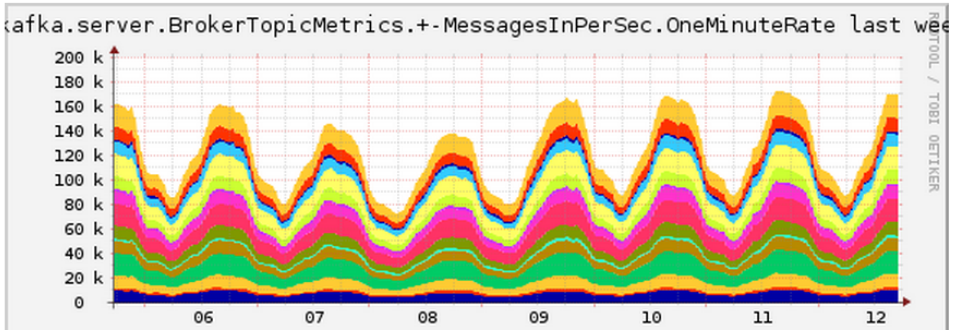
- Have occasional issues with a Broker dropping out of ISR due to expired Zookeeper connection.
- We suspect this is hardware or network related.
- Don't lose any messages if `request.required.acks` > 1

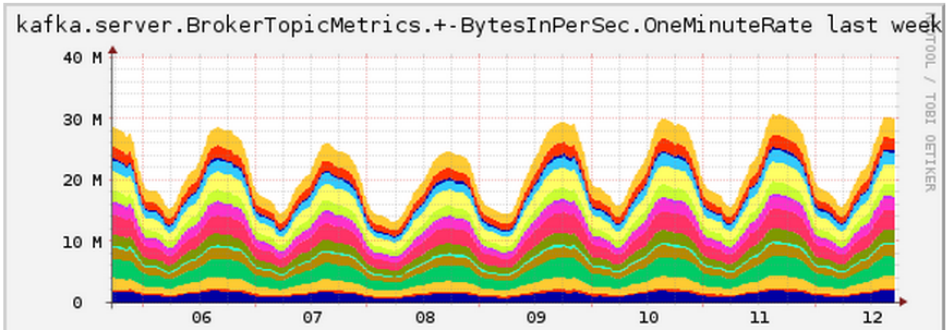
Other Tooling

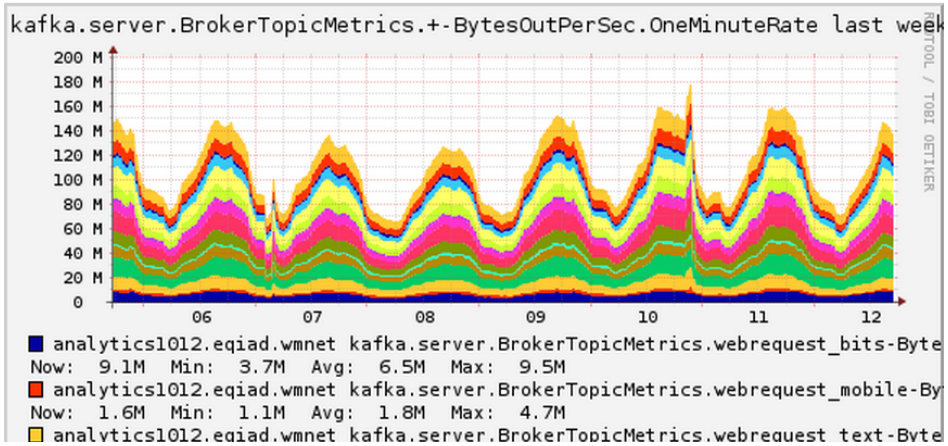
Kafka at Wikimedia - Monitoring

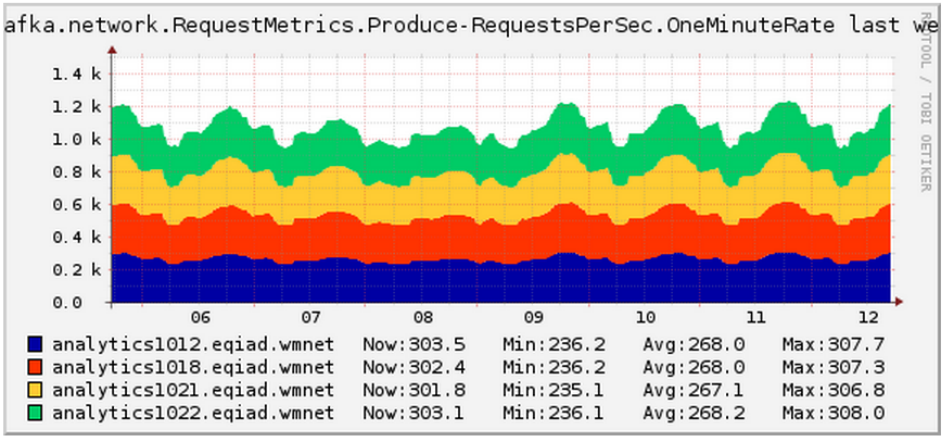
JMXtrans

Pulls JMX stats from Brokers into Ganglia





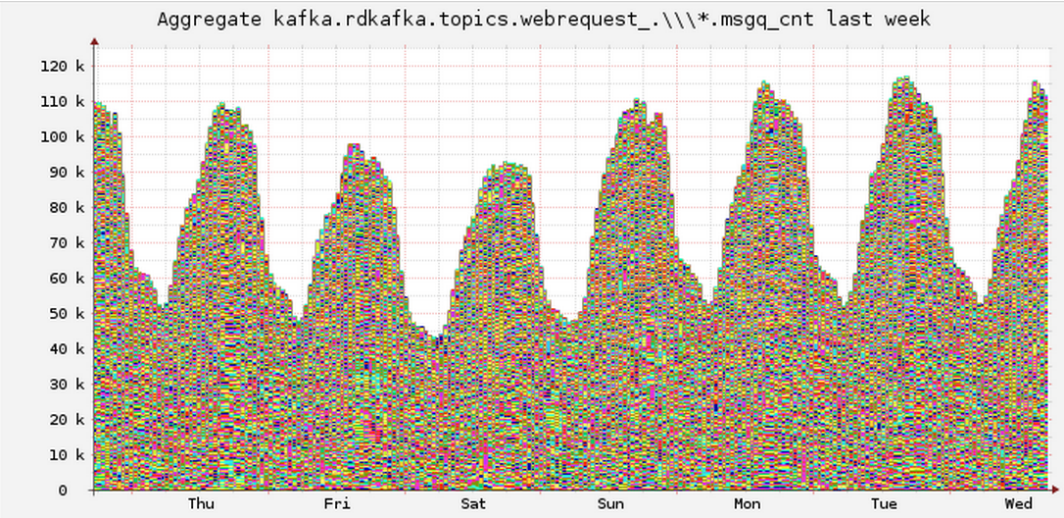




Kafka at Wikimedia - Monitoring

librdkafka's stats.json output

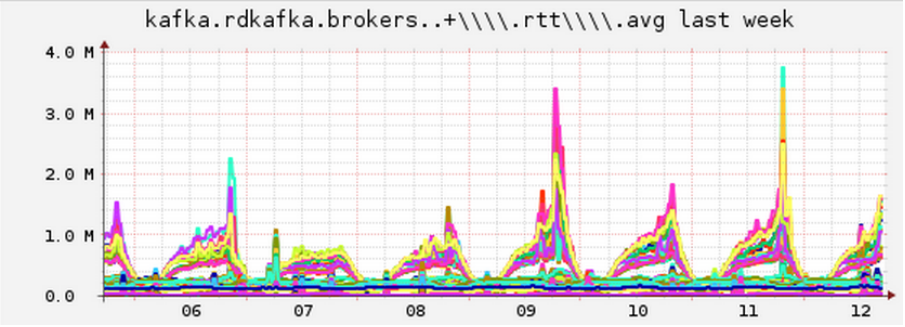
Used to send varnishkafka metrics to Ganglia:



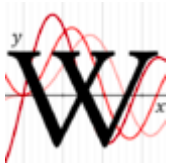
The number of messages queued to be sent by varnishkafka at any given time (measured per second).

(AHHH THE COLORS!
4 brokers * ~95 varnishkafkas * 12 partitions each = 4560 data points.)

Average produce request latency. The peaks are from varnishes in our Amsterdam datacenter.



Kafka at Wikimedia - Debian Package



Debian package

Wikimedia likes to follow Debian guidelines.

Requirement that .debs can be built without talking to the internet.

Ditched sbt and gradle in favor of custom Makefiles.

Includes (a better?) Kafka CLI than the bin/*.sh scripts.

Kafka at Wikimedia - Debian Package

```
Usage: kafka <command> [opts]

Commands:
  kafka topic                          [opts]

  kafka console-producer               [opts]
  kafka console-consumer               [opts]
  kafka simple-consumer-shell          [opts]
  kafka replay-log-producer            [opts]

  kafka mirror-maker                   [opts]
  kafka consumer-offset-checker        [opts]

  kafka add-partitions                 [opts]
  kafka reassign-partitions            [opts]
  kafka check-reassignment-status      [opts]
  kafka preferred-replica-election     [opts]
  kafka controlled-shutdown            [opts]

...
```

```
  kafka producer-perf-test             [opts]
  kafka consumer-perf-test             [opts]
  kafka simple-consumer-perf-test      [opts]

  kafka server-start           [-daemon] [<server.properties>]
  kafka server-stop

  kafka zookeeper-start        [-daemon] [<zookeeper.properties>]
  kafka zookeeper-stop
  kafka zookeeper-shell         [opts]

Environment Variables:
  ZOOKEEPER_URL               - If this is set, any commands that take
                                a --zookeeper flag will be passed with
                                this value.
  KAFKA_CONFIG                - location of Kafka config files.
                                 Default: /etc/kafka
  JMX_PORT                    - Set this to expose JMX.  This is set
                                by default for brokers and producers.

...
```

Kafka at Wikimedia - Puppet

puppet-kafka - works with Debian Package.

```
# Include Kafka Broker Server.
class { 'kafka::server':
    log_dirs         => ['/var/spool/kafka/a', '/var/spool/kafka/b'],
    brokers          => {
        'kafka-node01.example.com' => { 'id' => 1, 'port' => 12345 },
        'kafka-node02.example.com' => { 'id' => 2 },
    },
    zookeeper_hosts  => ['zk-node01:2181', 'zk-node02:2181', 'zk-node03:2181'],
    zookeeper_chroot => '/kafka/cluster_name',
}
```

kafka::server

```
# Configure kafka-mirror to produce to Kafka Brokers which are
# part of our kafka aggregator cluster.
class { 'kafka::mirror':
    destination_brokers => {
        'kafka-aggregator01.example.com' => { 'id' => 11 },
        'kafka-aggregator02.example.com' => { 'id' => 12 },
    },
    topic_whitelist => 'webrequest.*',
}

# Configure kafka-mirror to consume from both clusterA and clusterB
kafka::mirror::consumer { 'clusterA':
    zookeeper_hosts  => ['zk-node01:2181', 'zk-node02:2181', 'zk-node03:2181'],
    zookeeper_chroot => ['/kafka/clusterA'],
}
kafka::mirror::consumer { 'clusterB':
    zookeeper_hosts  => ['zk-node01:2181', 'zk-node02:2181', 'zk-node03:2181'],
    zookeeper_chroot => ['/kafka/clusterB'],
}
```

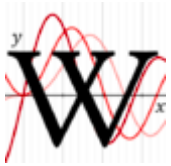
kafka::mirror
and
kafka::mirror::consumer

Kafka at Wikimedia - Puppet

puppet-varnishkafka

```
# varnishkafka instance using custom JSON output format
class { 'varnishkafka':
    brokers     => ['kafka1.example.org:9092', 'kafka2.example.org:9092'],
    format_type => 'json',
    format      => '%{@hostname}l %{@sequence!num}n %{%FT%T@dt}t %{Varnish:time_first
}
```

Kafka at Wikimedia - Puppet



puppet-kafkatee

```
# Install kafkatee and point it at a Kafka Cluster.
class { 'kafkatee':
    kafka_brokers => ['kafka1.example.org:9092', 'kafka2.example.org:9092'],
}

# Consume from the JSON formatted input topic 'mytopic'.
kafkatee::input { 'mytopic':
    topic       => 'mytopic',
    partitions  => '0-11',
    options     => { 'encoding' => 'json' },
    offset      => 'stored',
}

# Output sampling 1/100 to a file:
kafkatee::output { 'mytopic-sampled-100':
    destination => '/path/to/mytopic-sampled-100.json',
    type        => 'file',
    sample      => 100,
}

# Output with no sampling to a piped filtering process:
kafkatee::output { 'mytopic-filtered':
    destination => 'grep -v "whocares" >> /path/to/mytopic-filtered.json'
    type        => 'pipe',
}
```

Thanks! Questions?