Netflix Data Pipeline with Kafka

Allen Wang & Steven Wu

Agenda

- Introduction
- Evolution of Netflix data pipeline
- How do we use Kafka

What is Netflix?

Netflix is a logging company

that occasionally streams video

Numbers

- 400 billion events per day
- 8 million events & 17 GB per second during peak
- hundreds of event types

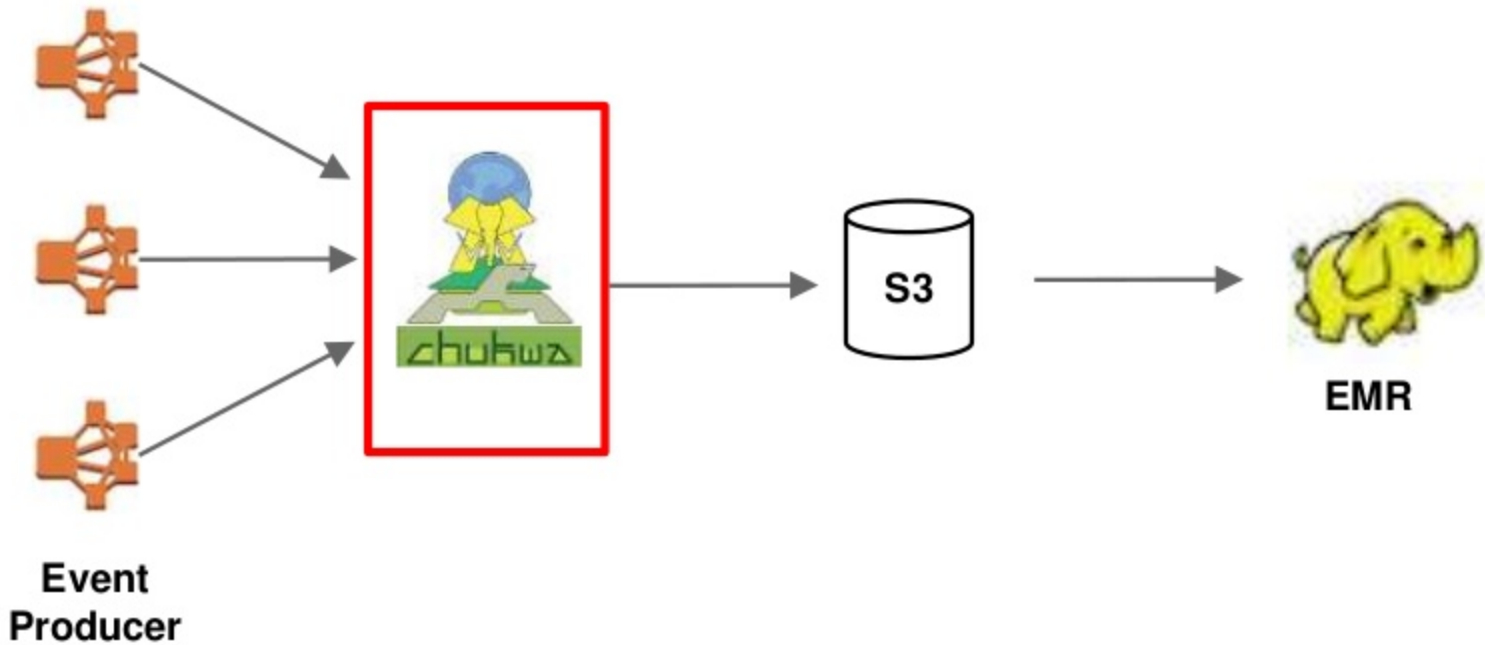
Agenda

- Introduction
- Evolution of Netflix data pipeline
- How do we use Kafka

Mission of Data Pipeline

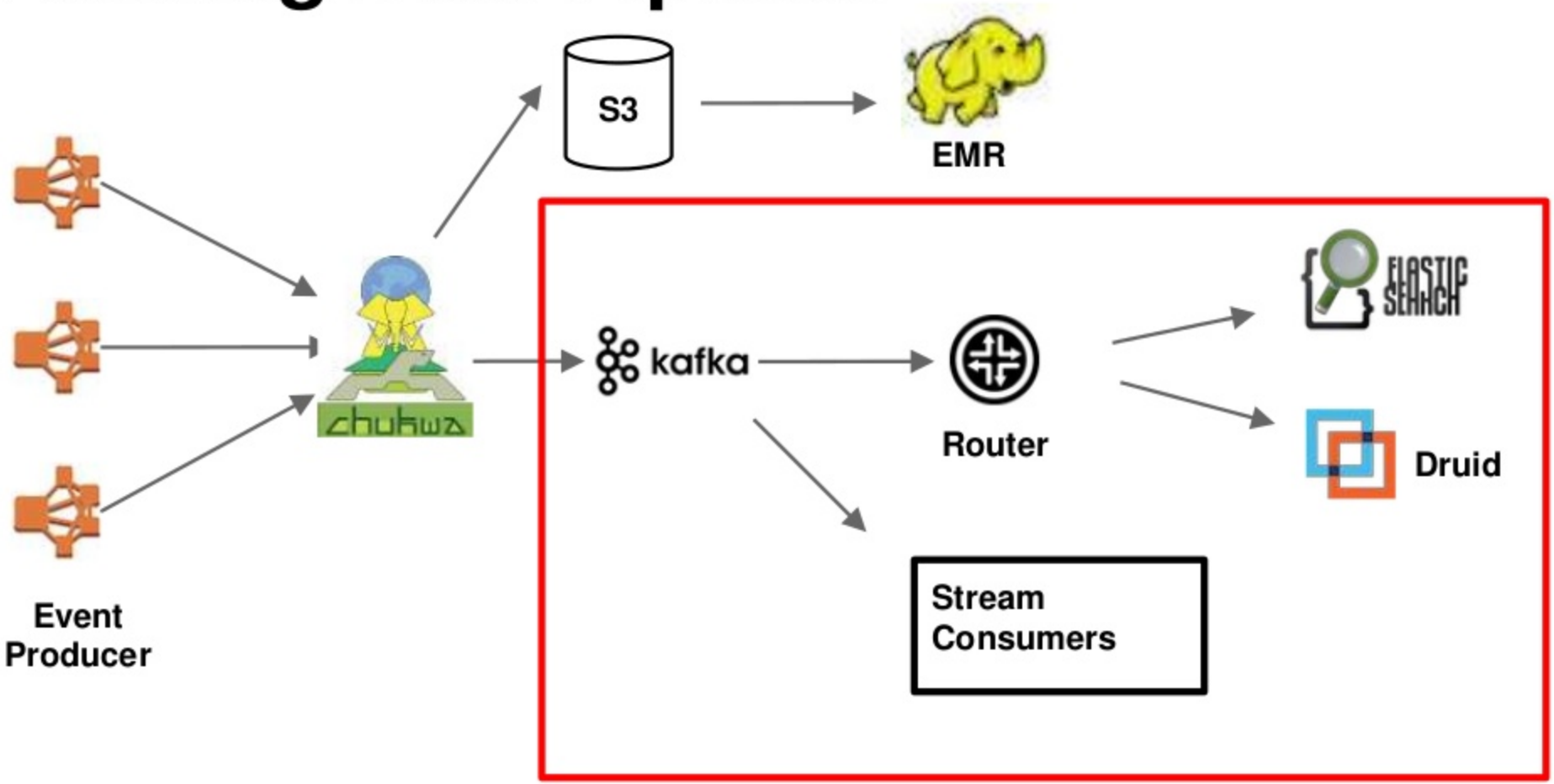
Publish, Collect, Aggregate, Move Data @ Cloud Scale

In the old days ...


chukwa
S3
EMR
Event
Producer

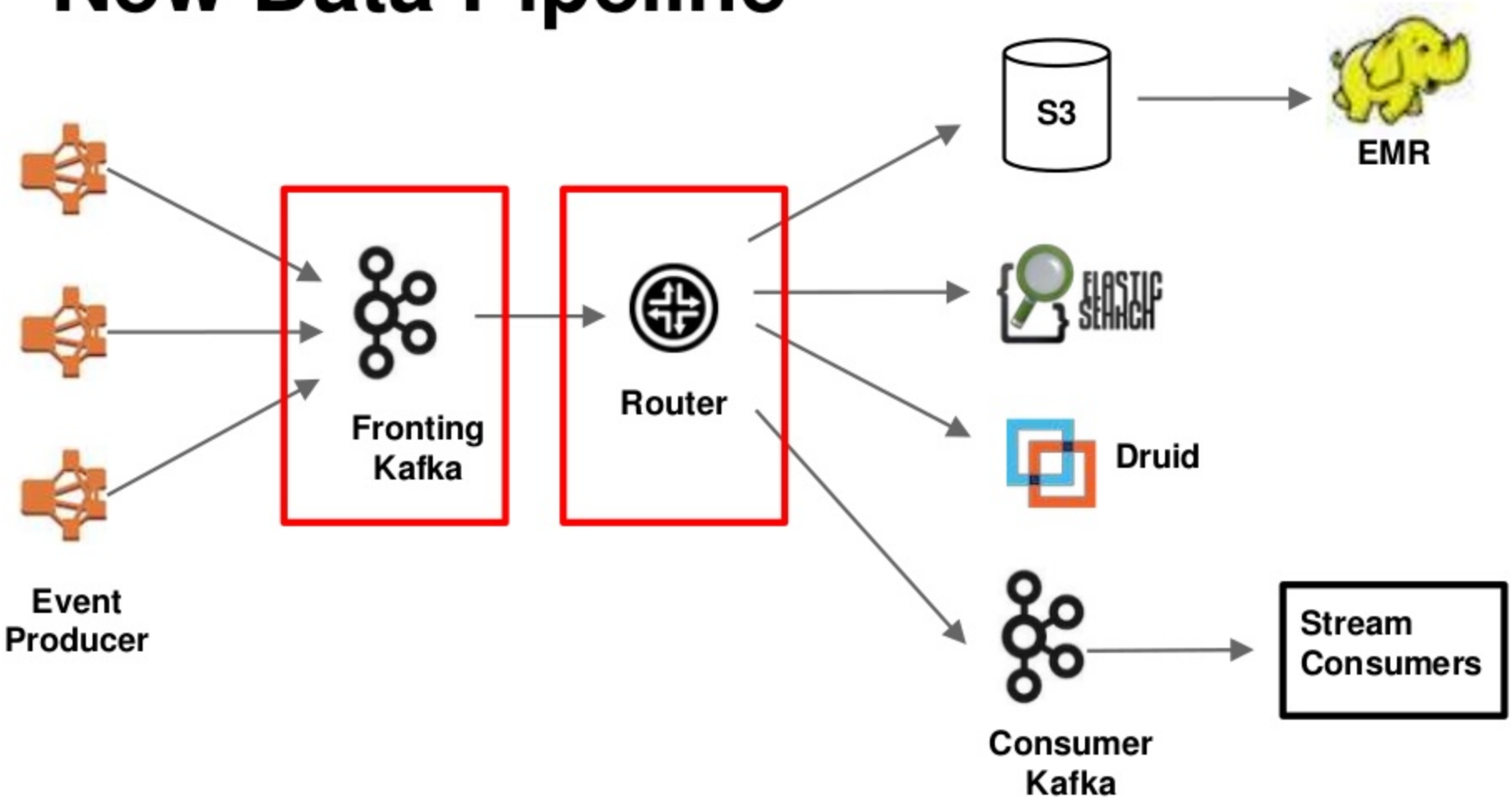
Nowadays ...

Existing Data Pipeline

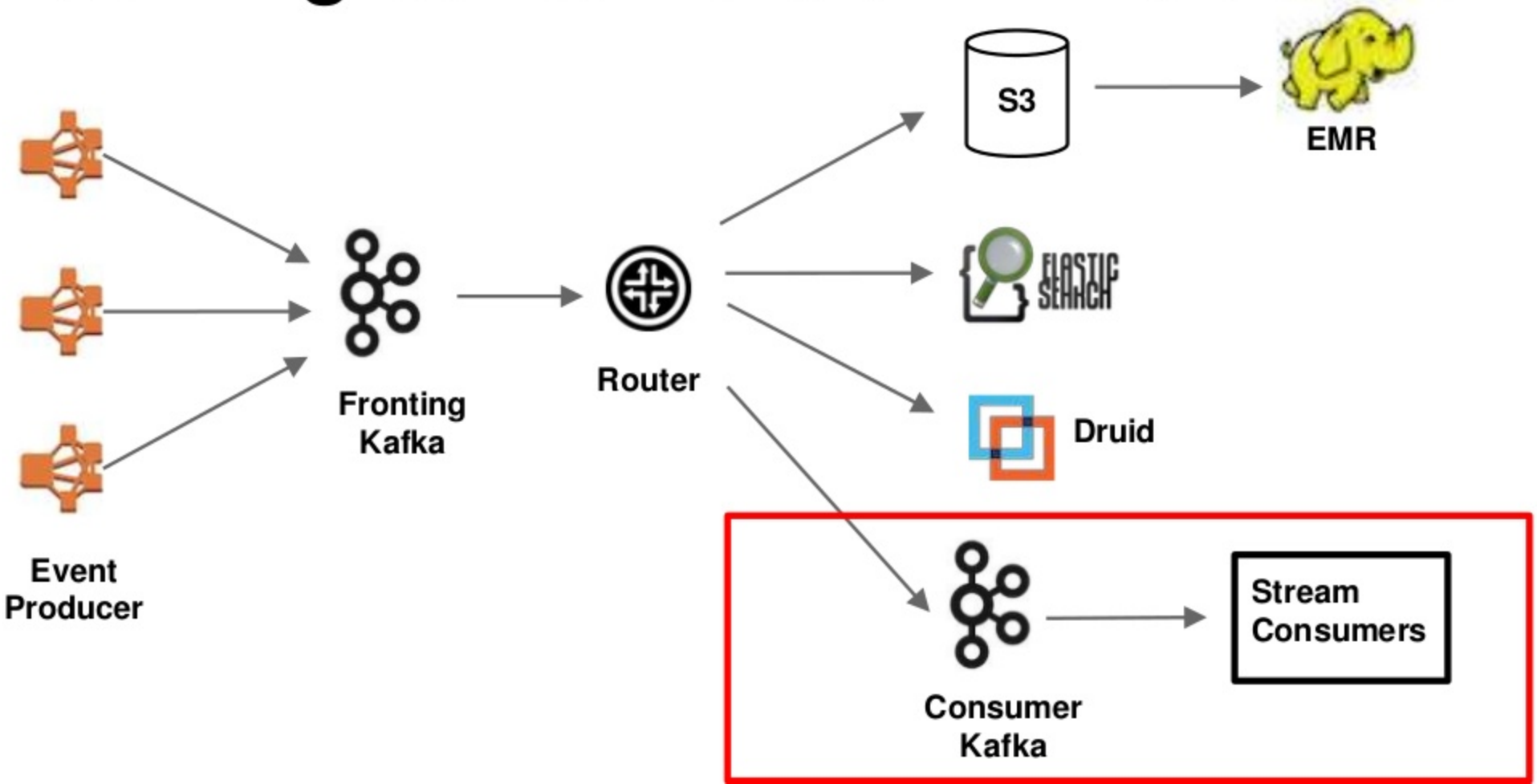


In to the Future ...

New Data Pipeline



Serving Consumers off Diff Clusters



Split Fronting Kafka Clusters

- Low-priority (error log, request trace, etc.)
  - 2 copies, 1-2 hour retention
- Medium-priority (majority)
  - 2 copies, 4 hour retention
- High-priority (streaming activities etc.)
  - 3 copies, 12-24 hour retention

Producer Resilience

- Kafka outage should never disrupt existing instances from serving business purpose
- Kafka outage should never prevent new instances from starting up
- After kafka cluster restored, event producing should resume automatically

Fail but Never Block

- block.on.buffer.full=false
- handle potential blocking of first meta data request
- Periodical check whether KafkaProducer was opened successfully

Agenda

- Introduction
- Evolution of Netflix data pipeline
- How do we use Kafka

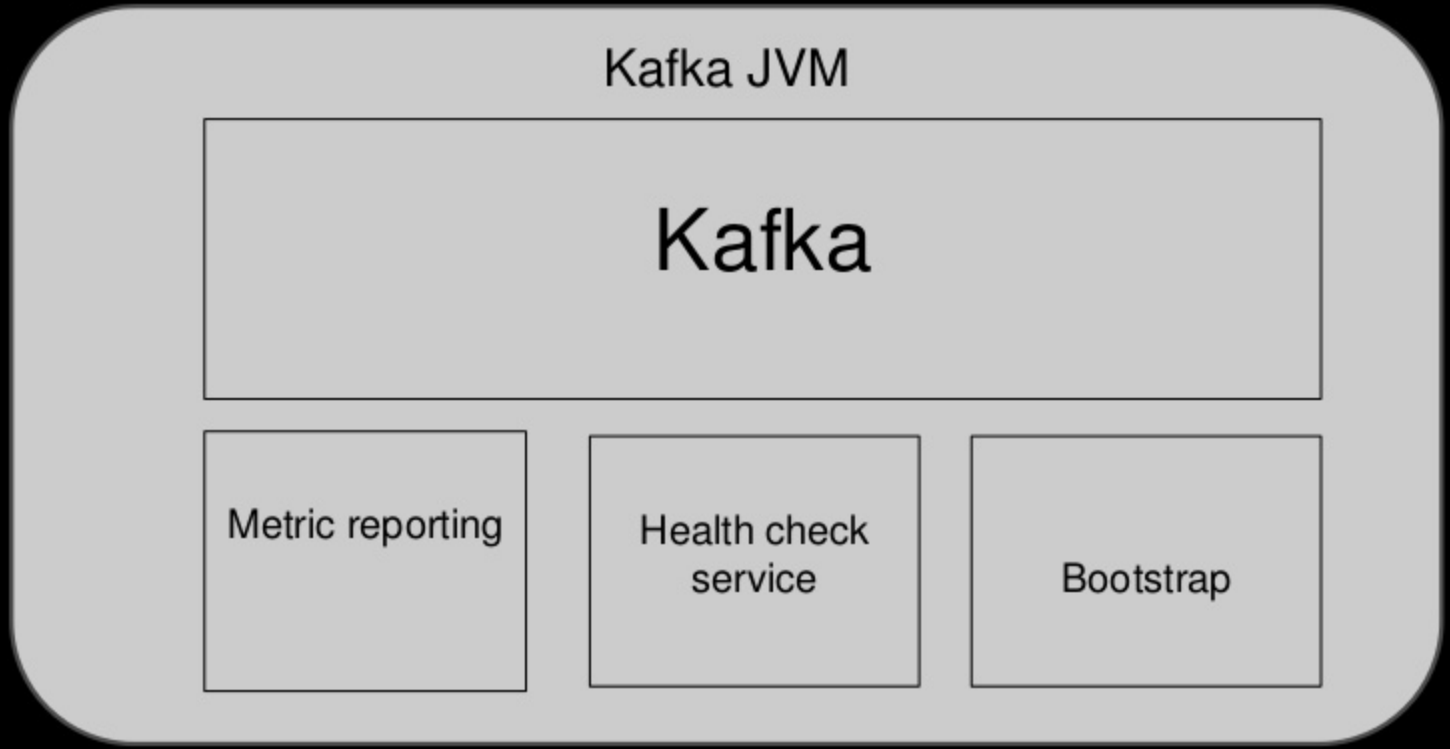
What Does It Take to Run In Cloud

- Support elasticity
- Respond to scaling events
- Resilience to failures
  - Favors architecture without single point of failure
  - Retries, smart routing, fallback ...

Kafka in AWS - How do we make it happen

- Inside our Kafka JVM
- Services supporting Kafka
- Challenges/Solutions
- Our roadmap

Netflix Kafka Container

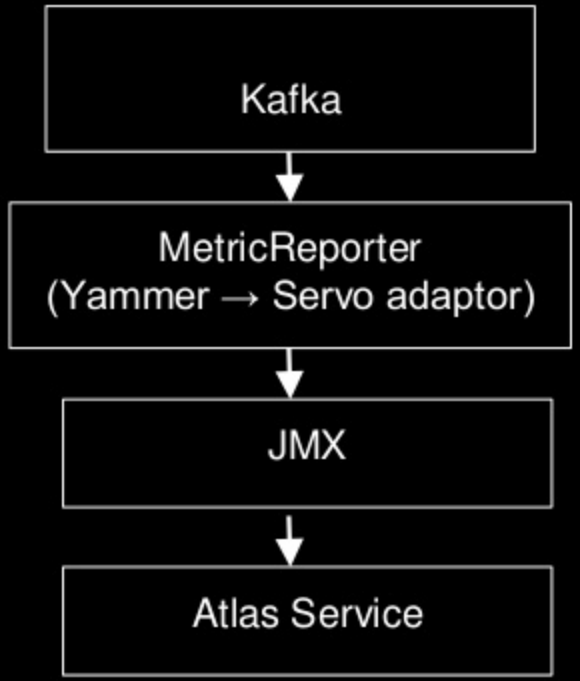


Bootstrap

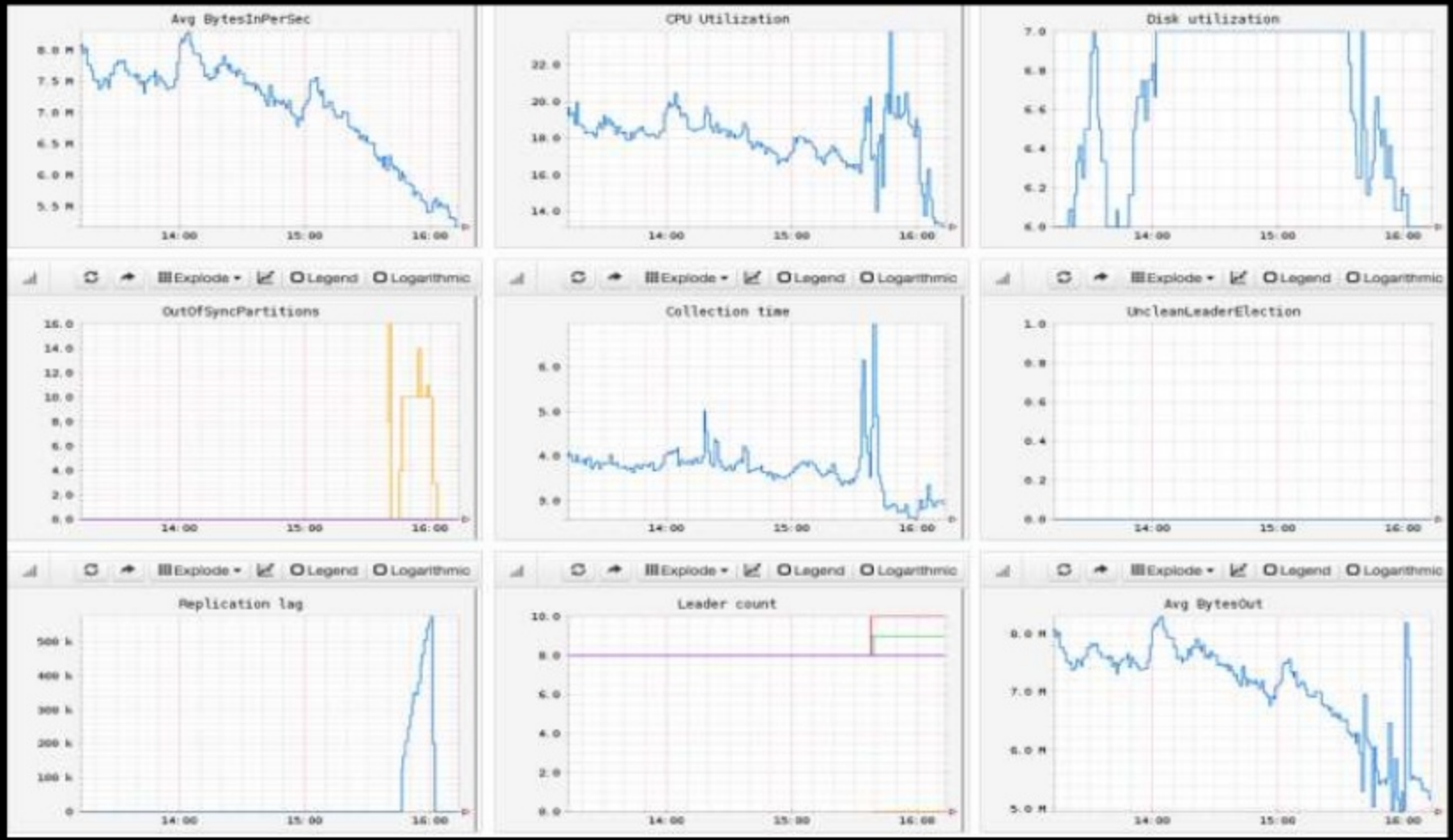
- Broker ID assignment
  - Instances obtain sequential numeric IDs using Curator's locks recipe persisted in ZK
  - Cleans up entry for terminated instances and reuse its ID
  - Same ID upon restart
- Bootstrap Kafka properties from Archaius
  - Files
  - System properties/Environment variables
  - Persisted properties service
- Service registration
  - Register with Eureka for internal service discovery
  - Register with AWS Route53 DNS service

Metric Reporting

- We use Servo and Atlas from NetflixOSS



Kafka Atlas Dashboard



Health check service

- Use Curator to periodically read ZooKeeper data to find signs of unhealthiness
- Export metrics to Servo/Atlas
- Expose the service via embedded Jetty

Kafka in AWS - How do we make it happen

- Inside our Kafka JVM
- Services supporting Kafka
- Challenges/Solutions
- Our roadmap

ZooKeeper

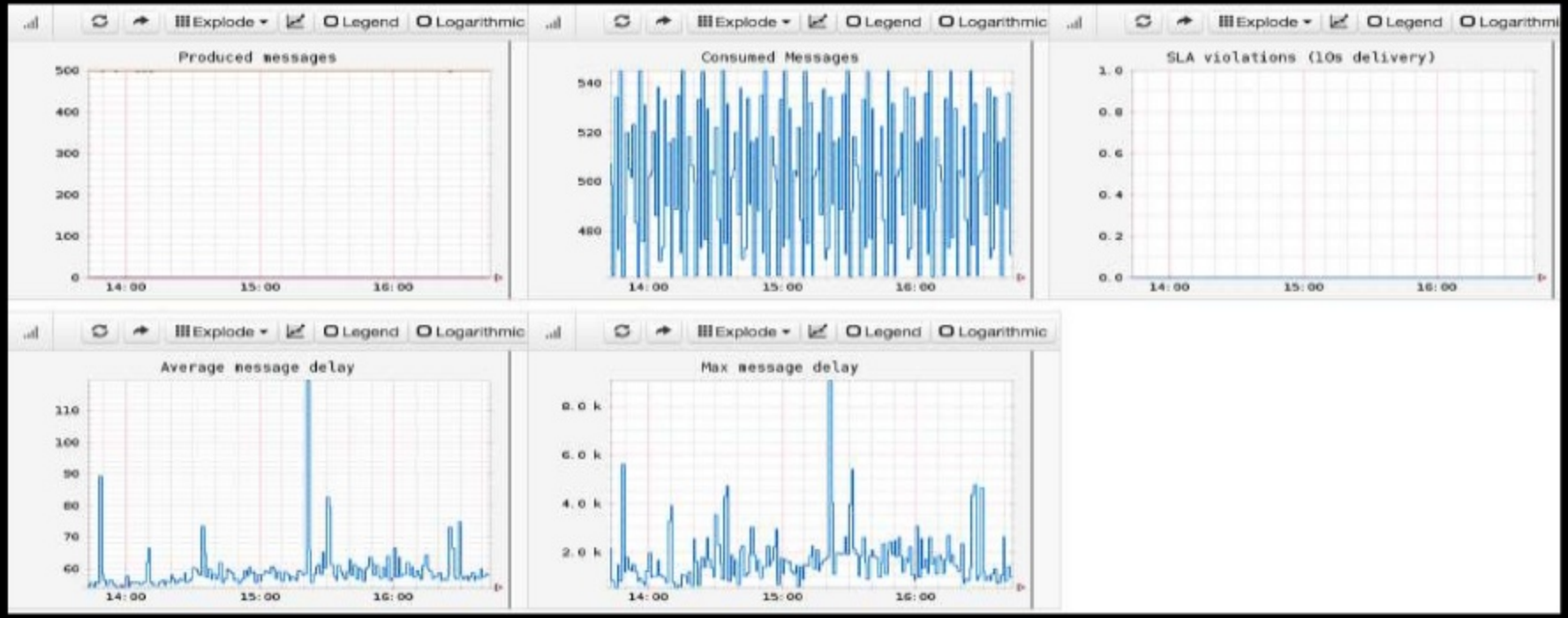
- Dedicated 5 node cluster for our data pipeline services
- EIP based
- SSD instance

Auditor

- Highly configurable producers and consumers with their own set of topics and metadata in messages
- Built as a service deployable on single or multiple instances
- Runs as producer, consumer or both
- Supports replay of preconfigured set of messages

Auditor

- Broker monitoring (Heartbeating)

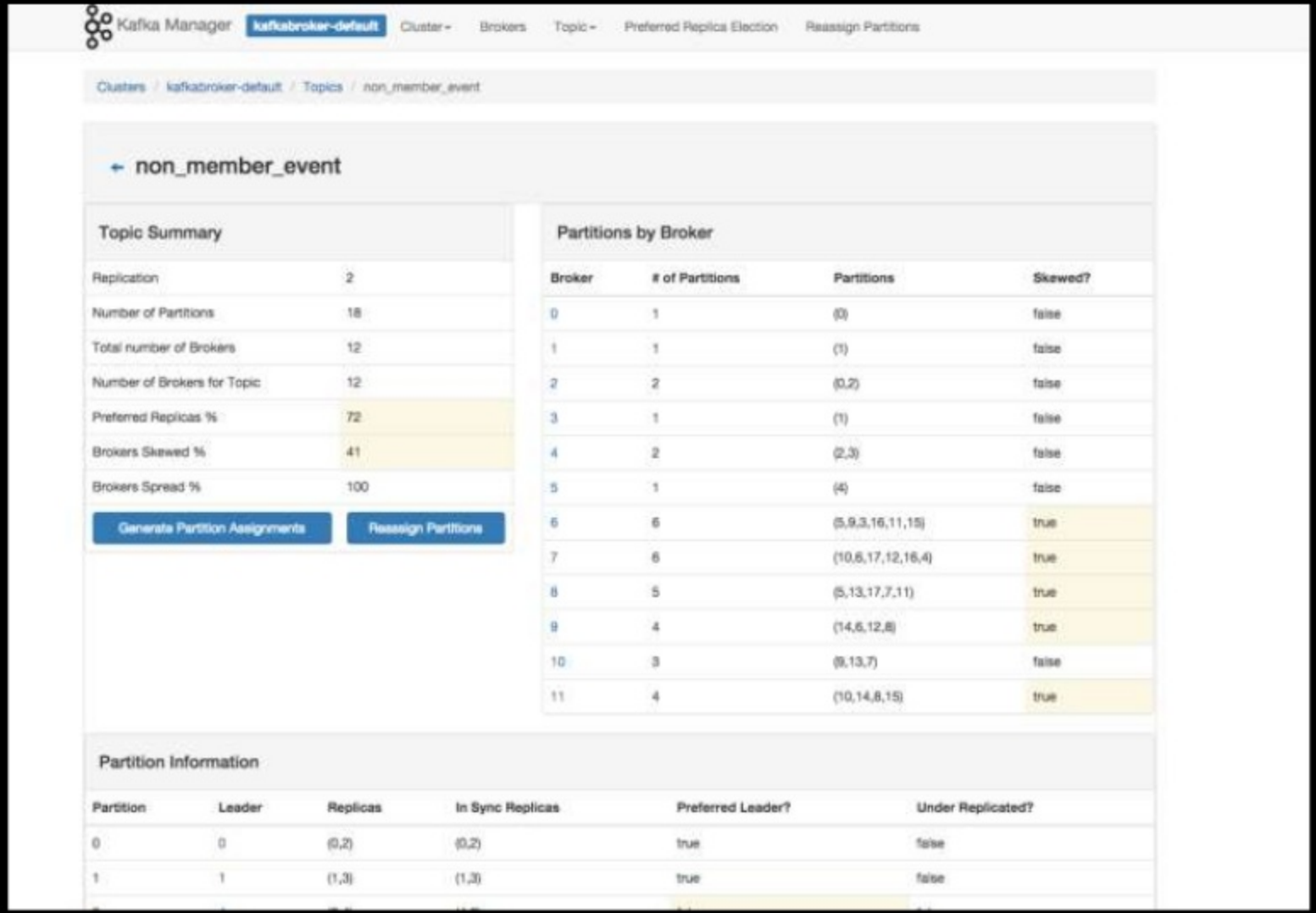


Auditor

- Broker performance testing
  - Produce tens of thousands messages per second on single instance
  - As consumers to test consumer impact

Kafka admin UI

- Still searching …
- Currently trying out KafkaManager


Kafka Manager
kafkabroker-default
Cluster
Brokers
Topic
Preferred Replica Election
Reassign Partitions
Clusters / kafkabroker-default / Topics / non_member_event
non_member_event
Topic Summary
Replication 2
Number of Partitions 18
Total number of Brokers 12
Number of Brokers for Topic 12
Preferred Replicas % 72
Brokers Skewed % 41
Brokers Spread % 100
Generate Partition Assignments
Reassign Partitions
Partitions by Broker
Broker # of Partitions Partitions Skewed?
0 1 (0) false
1 1 (1) false
2 2 (0,2) false
3 1 (1) false
4 2 (2,3) false
5 1 (4) false
6 6 (5,9,3,16,11,15) true
7 6 (10,6,17,12,16,4) true
8 5 (5,13,17,7,11) true
9 4 (14,6,12,8) true
10 3 (9,13,7) false
11 4 (10,14,8,15) true
Partition Information
Partition Leader Replicas In Sync Replicas Preferred Leader? Under Replicated?
0 0 (0,2) (0,2) true false
1 1 (1,3) (1,3) true false

Kafka in AWS - How do we make it happen

- Inside our Kafka JVM
- Services supporting Kafka
- Challenges/Solutions
- Our roadmap

Challenges

- ZooKeeper client issues
- Cluster scaling
- Producer/consumer/broker tuning

ZooKeeper Client

- Challenges
  - Broker/consumer cannot survive ZooKeeper cluster rolling push due to caching of private IP
  - Temporary DNS lookup failure at new session initialization kills future communication

ZooKeeper Client

- Solutions
  - Created our internal fork of Apache ZooKeeper client
  - Periodically refresh private IP resolution
  - Fallback to last good private IP resolution upon DNS lookup failure

Scaling

- Provisioned for peak traffic
  - … and we have regional fail-over

Strategy #1 Add Partitions to New Brokers

- Caveat
  - Most of our topics do not use keyed messages
  - Number of partitions is still small
  - Require high level consumer

Strategy #1 Add Partitions to new brokers

- Challenges: existing admin tools does not support atomic adding partitions and assigning to new brokers

Strategy #1 Add Partitions to new brokers

- Solutions: created our own tool to do it in one ZooKeeper change and repeat for all or selected topics
- Reduced the time to scale up from a few hours to a few minutes

Strategy #2 Move Partitions

- Should work without precondition, but ...
- Huge increase of network I/O affecting incoming traffic
- A much longer process than adding partitions
- Sometimes confusing error messages
- Would work if pace of replication can be controlled

Scale down strategy

- There is none
- Look for more support to automatically move all partitions from a set of brokers to a different set

Client tuning

- Producer
  - Batching is important to reduce CPU and network I/O on brokers
  - Stick to one partition for a while when producing for non-keyed messages
  - “linger.ms” works well with sticky partitioner
- Consumer
  - With huge number of consumers, set proper fetch.wait.max.ms to reduce polling traffic on broker

Effect of batching

partitioner	**batched records per request**	**broker cpu util** [1]
random without lingering	1.25	75%
sticky without lingering	2.0	50%
sticky with 100ms lingering	15	33%

[1] 10 MB & 10K msgs / second per broker, 1KB per message

Broker tuning

- Use G1 collector
- Use large page cache and memory
- Increase max file descriptor if you have thousands of producers or consumers

Kafka in AWS - How do we make it happen

- Inside our Kafka JVM
- Services supporting Kafka
- Challenges/Solutions
- Our roadmap

Road map

- Work with Kafka community on rack/zone aware replica assignment
- Failure resilience testing
  - Chaos Monkey
  - Chaos Gorilla
- Contribute to open source
  - Kafka
  - Schlep -- our messaging library including SQS and Kafka support
  - Auditor


Thank you!
http://netflix.github.io/
http://techblog.netflix.com/
@NetflixOSS
@allenxwang
@stevenzwu
NETFLIX