AWS re:Invent

Unmeltable Infrastructure at Scale:
Using Apache Kafka, Twitter Storm, and ElasticSearch on AWS

Jim Nisbet
CTO and VP of Engineering, Loggly

Philip O'Toole
Lead Developer, Infrastructure, Loggly

November 2013

amazon web services

What Loggly Does

- Log Management as a service
  - Near real-time indexing of events
- Distributed architecture, built on AWS
- Initial production services in 2010
  - Loggly Generation 2 released in Sept 2013
- Thousands of customers

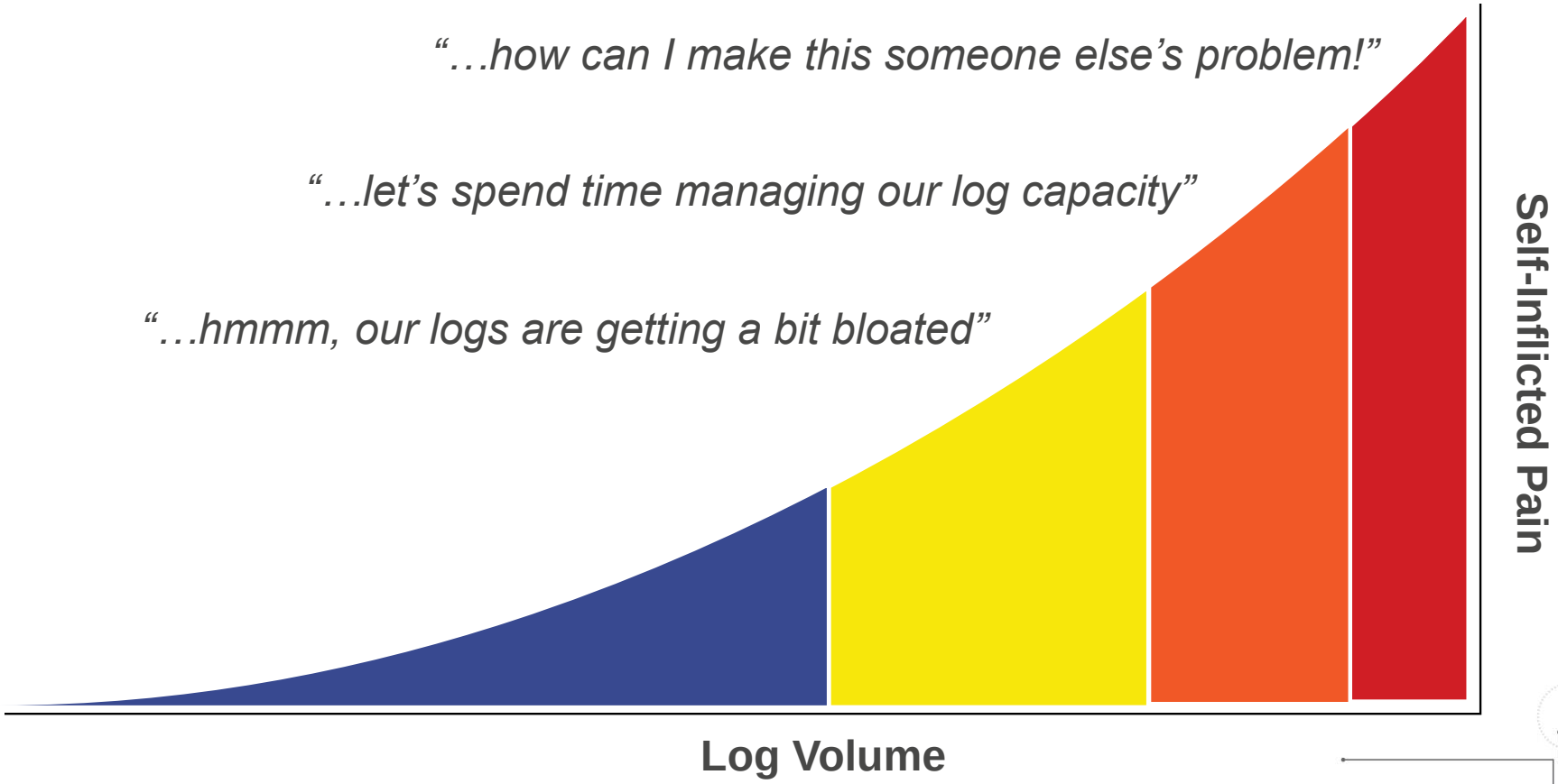
Agenda for this Presentation

- A bit about logging
- Lessons learned from our first generation
- How we leverage AWS services
- Our use of Kafka, Storm, ElasticSearch
- What worked well for us and what did not

Log Management

- Everyone starts with …
  - A bunch of log files (syslog, application specific)
  - On a bunch of machines

- Management consists of doing the simple stuff
  - Rotate files, compress and delete
  - Information is there but awkward to find specific events
  - Weird log retention policies evolve over time

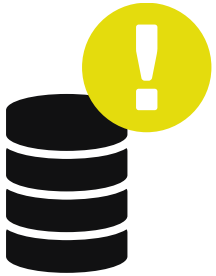

"…how can I make this someone else's problem!"
"…let's spend time managing our log capacity"
"…hmmm, our logs are getting a bit bloated"
Self-Inflicted Pain
Log Volume

Best Practices in Log Management

- Use existing logging infrastructure
  - Real time syslog forwarding is built in
  - Application log file watching
- Store logs externally
  - Accessible when there is a system failure
- Log messages in machine parsable format
  - JSON encoding when logging structured information
  - Key-value pairs

From the Trenches…

- Managing Applications vs. Managing Logs
  - Do not make this is an either/or proposition!



If you get a disk space alert, first login…

```
% sudo rm –rf /var/log/apache2/*
```

Admit it, we've all seen this kind of thing!

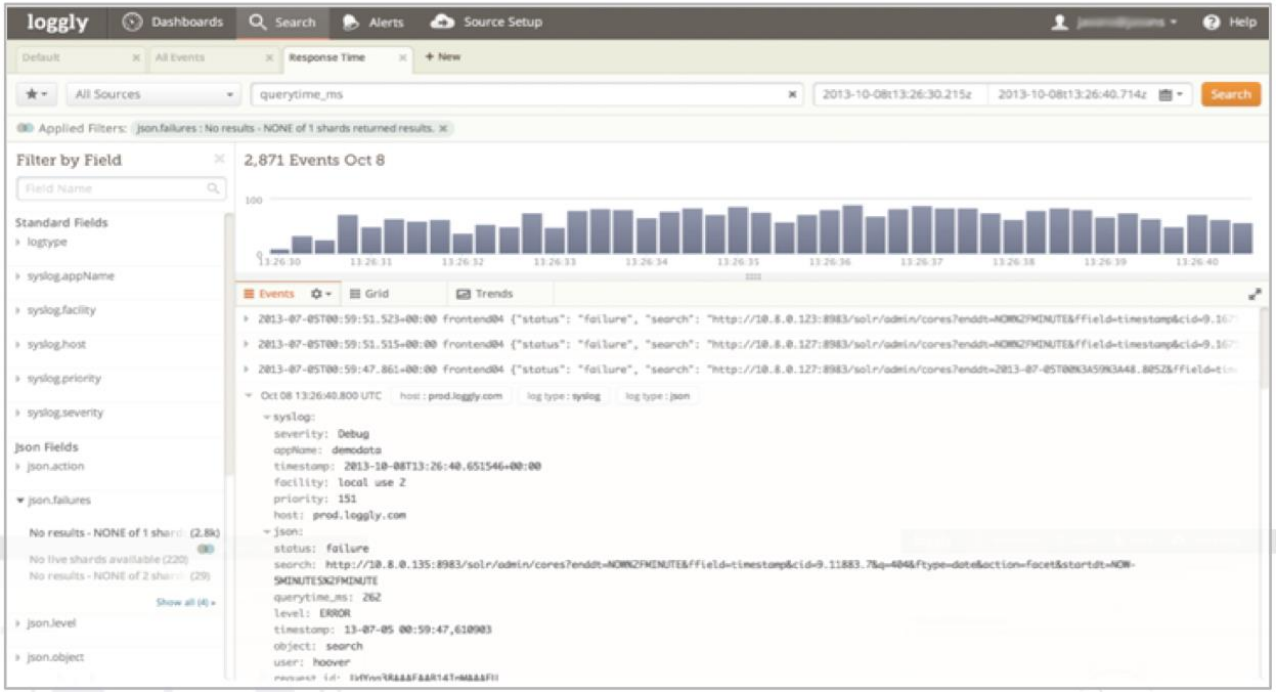
You Have Logs...

```
2013-10-25T18:35:43.387+0000: 441.482: [GC [PSYoungGen: 2430541K->268617K(2484544K)] 7687523K->5660738K(8076992K), 0.3266870 secs] [Times: user=1.05 sys=0.17, real=0.33 secs]
2013-10-25T18:35:43.714+0000: 441.809: [Full GC [PSYoungGen: 268617K->0K(2484544K)] [ParOldGen: 5392121K->354965K(5592448K)] 5660738K->354965K(8076992K) [PSPermGen: 44444K->44395K(83968K)], 0.9225290 secs] [Times: user=2.22 sys=0.26, real=0.92 secs]
```

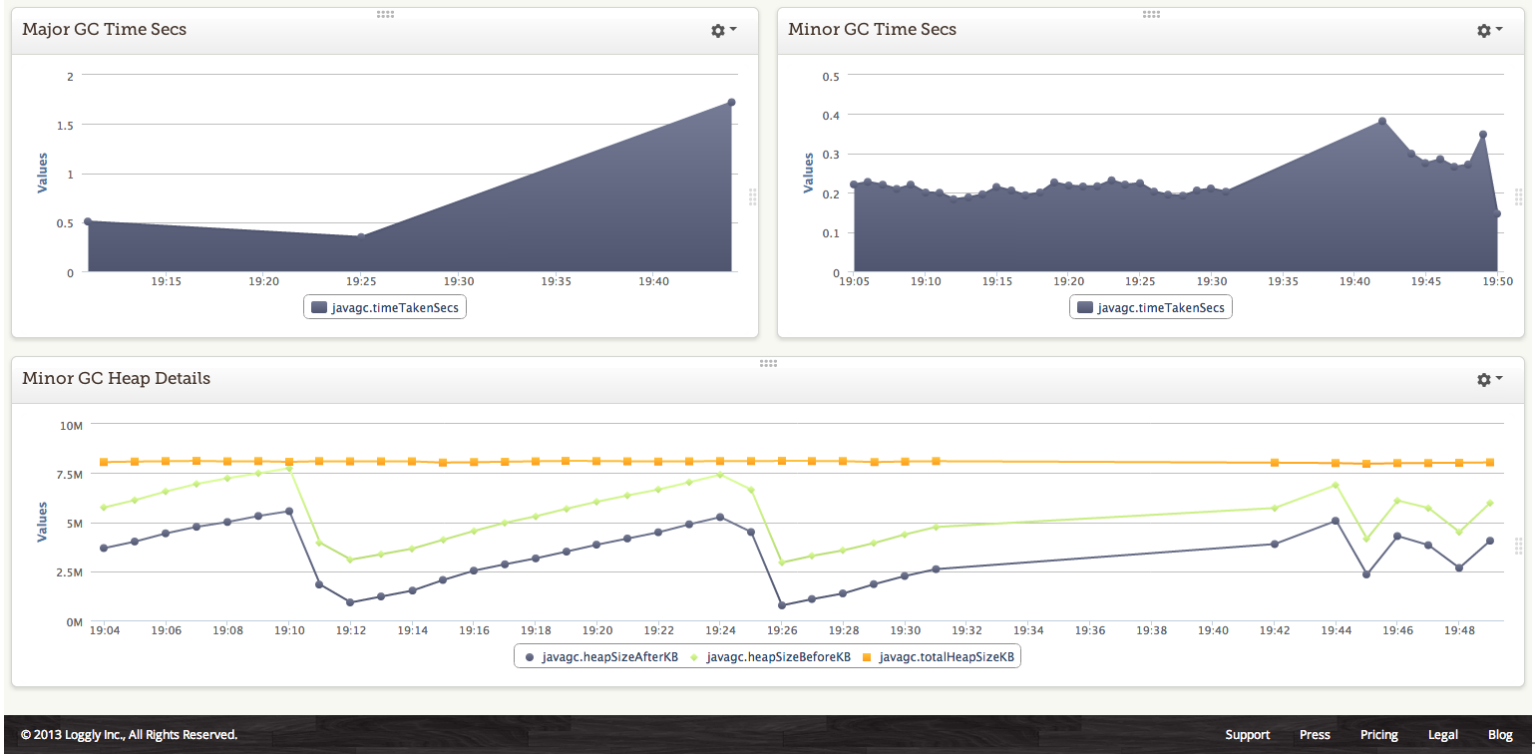
- In this case, JVM garbage collection logs enabled with…

```
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps
```

Yes, you need to search these logs



But you also need to to spot trends



Loggly Offers Logging as a Service

Loggly First Generation

- Logging as a service
  - Near real-time searchable logs
- Thousands of customers
  - Transmission rates from 10 events/sec to 100k events/sec
  - When customers systems are busy they send more logs
  - Log traffic has distinct bursts; bursts can last for several hours
- Amazon EC2 deployment
  - We used EC2 Instance storage
- SOLR Cloud
  - Full power of Lucene search
  - Tens of thousands of shards (with special ‘sleep shard’ logic)
- ZeroMQ for message queue

First Generation Lessons Learned

- Event ingestion too tightly coupled to indexing
  - Manual re-indexing for temporary SOLR issues
- Multiple Indexing strategies needed
  - 4 orders of magnitude difference between our high volume users and our low volume users (10 eps vs. 100,000+ eps)
  - Too much system overhead for low volume users
  - Difficult to support changing indexing strategies for a customer

Big Data Infrastructure Solutions

We are not alone…

- Our challenges
  - Massive incoming event stream
  - Fundamentally multi-tenant
  - Scalable framework for analysis
  - Near real-time indexing
  - Time series index management

Apache Kafka

- Overview
  - An Apache project initially developed at LinkedIn
  - Distributed publish-subscribe messaging system
  - Specifically designed for real time activity streams
  - Does not follow JMS Standards nor uses JMS APIs

- Key Features
  - Persistent messaging
  - High throughput, low overhead
  - Uses ZooKeeper for forming a cluster of nodes
  - Supports both queue and topic semantics

Message Queue Performance

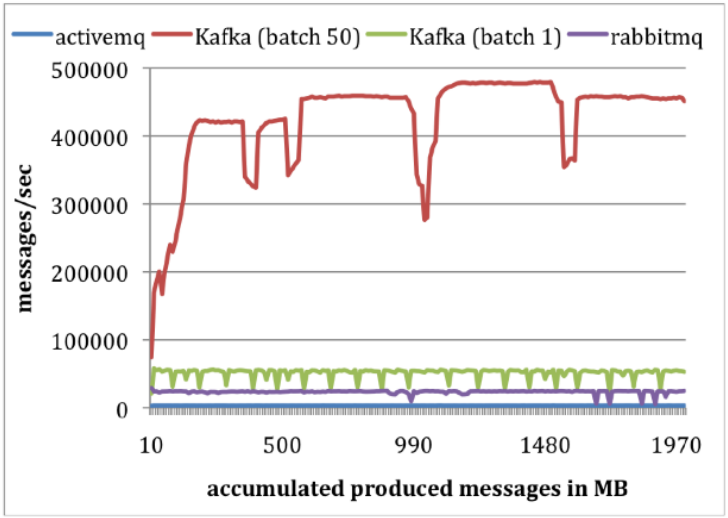


Figure 4. Producer Performance

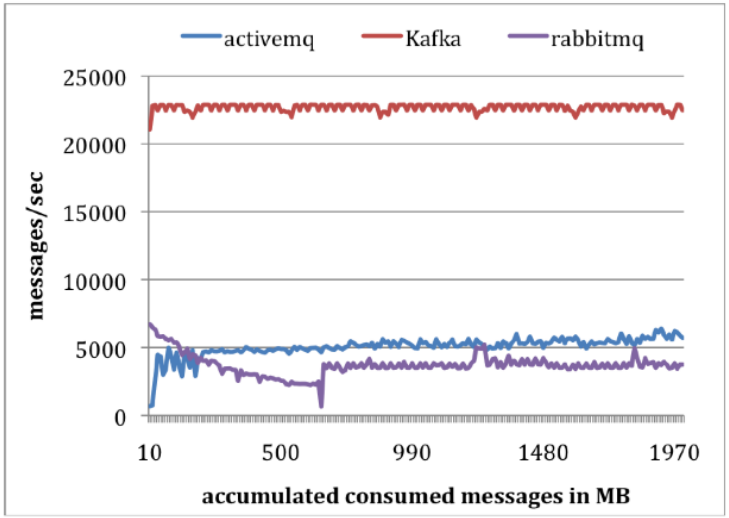


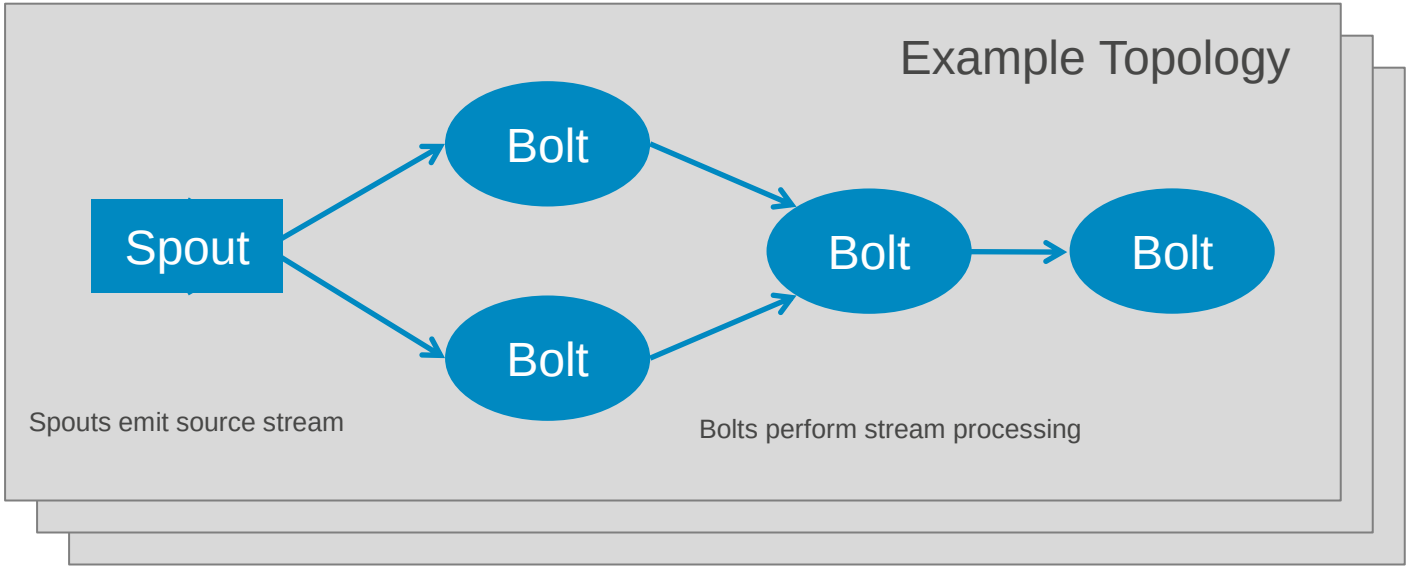
Figure 5. Consumer Performance

http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

Storm Framework

- Storm (open sourced by Twitter)
  - Open sourced September 2011
  - Now an Apache Software Foundation project
    - Currently Incubator Status
- Framework is for stream processing
  - Distributed
  - Fault tolerant
  - Computation
  - Fail-fast components

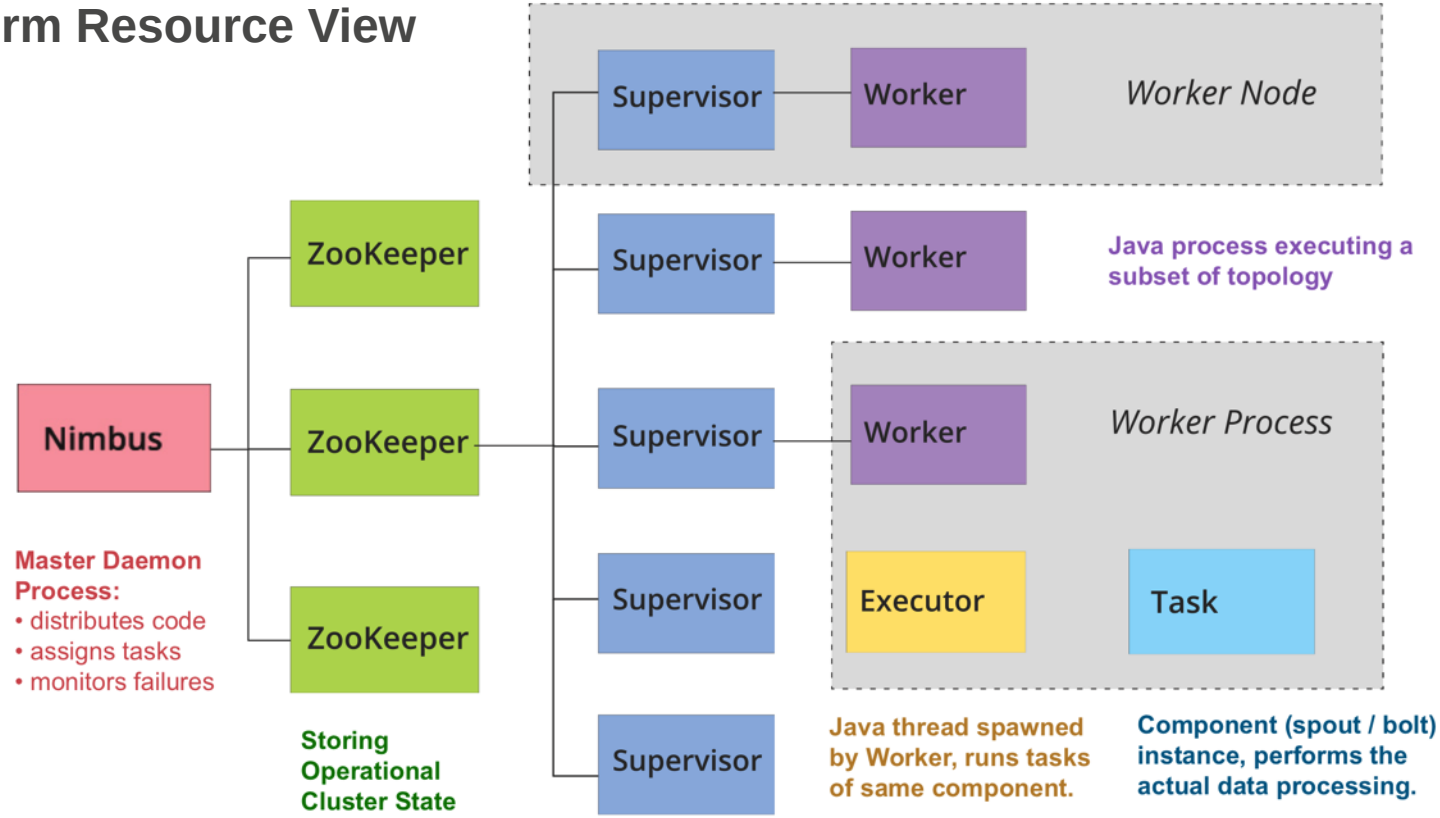
Storm Logical View



Storm terminology

- Streams, Spouts, Bolts, Tasks, Workers, Stream Groups and Topologies

Storm Resource View



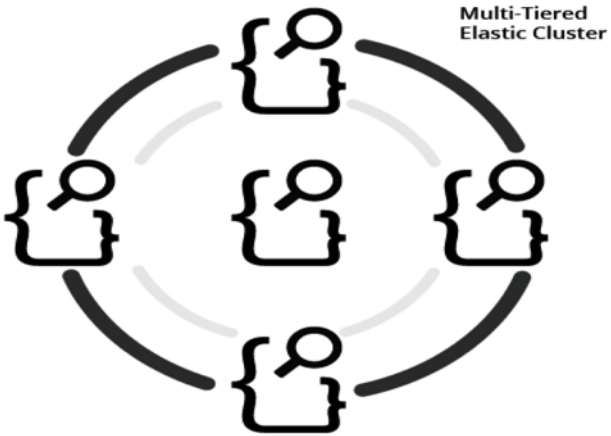
ElasticSearch

- Open source
  - Commercial support available from ElasticSearch.com
  - Growing open-source community

- Distributed search engine
- Fully exposes Lucene search functionality
- Built for clustering from the ground-up
- High availability
- Multi-tenancy

ElasticSearch In Action

- Add/delete nodes dynamically
- Add indices with REST API
- Indices and Nodes have attributes
  - Indices automatically moved to best Nodes
- Indices can be sharded
- Supports bulk insertion of events
- Plugins for monitoring cluster



Our Second Generation

Generation 2 – The Challenge

- Always accept log data
  - Never make a customer's incident worse
- Never drop log data
  - A single log message could be critical
- True Elasticity

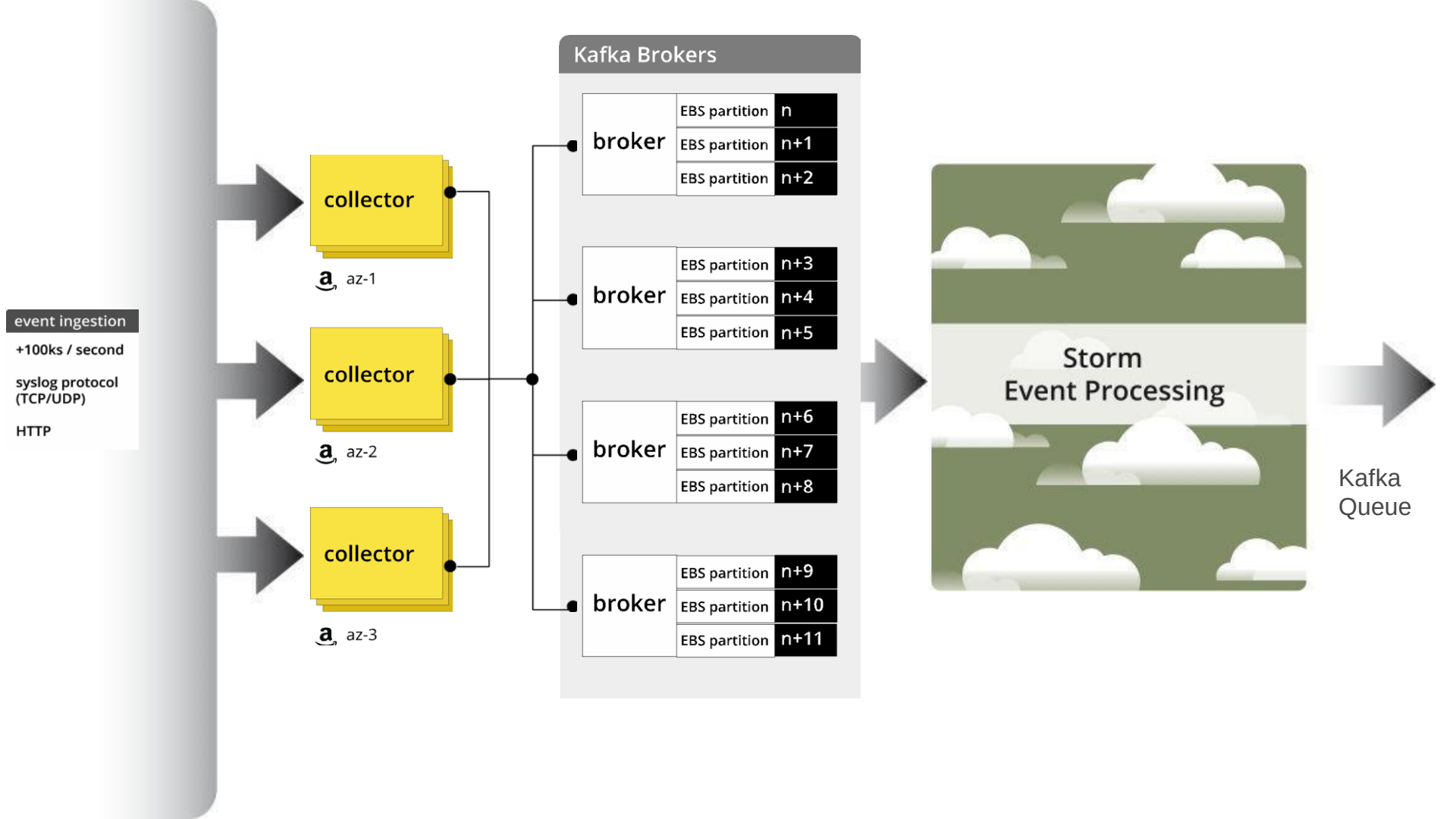
Perfect Match For Real Time Log Events

- Apache Kafka
  - Extremely high-performance pub-sub persistent queue
- Consumer tracks their location in queue
  - A good fit for our use cases
- Multiple Kafka brokers
  - Good match for AWS
    - Multiple brokers per region
    - Availability Zone separation

Real Time Event Processing

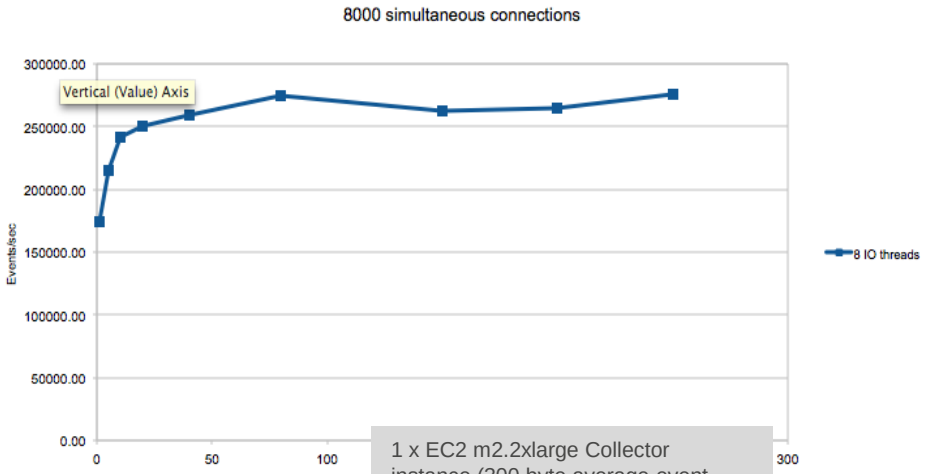
- Twitter Storm
  - Scalable real-time computation system
- Storm used as a “pull” system
  - Provisioned for average load, not peak load
  - Input from Kafka queue
    - Worker nodes can be scaled dynamically
- Elasticity is key
  - Another good match for AWS
    - Able to scale workers up and down dynamically

Log Event Ingestion


event ingestion
+100ks / second
syslog protocol (TCP/UDP)
HTTP
collector
az-1
collector
az-2
collector
az-3
Kafka Brokers
broker
EBS partition n
EBS partition n+1
EBS partition n+2
broker
EBS partition n+3
EBS partition n+4
EBS partition n+5
broker
EBS partition n+6
EBS partition n+7
EBS partition n+8
broker
EBS partition n+9
EBS partition n+10
EBS partition n+11
Storm
Event Processing
Kafka
Queue

Loggly Collector Performance

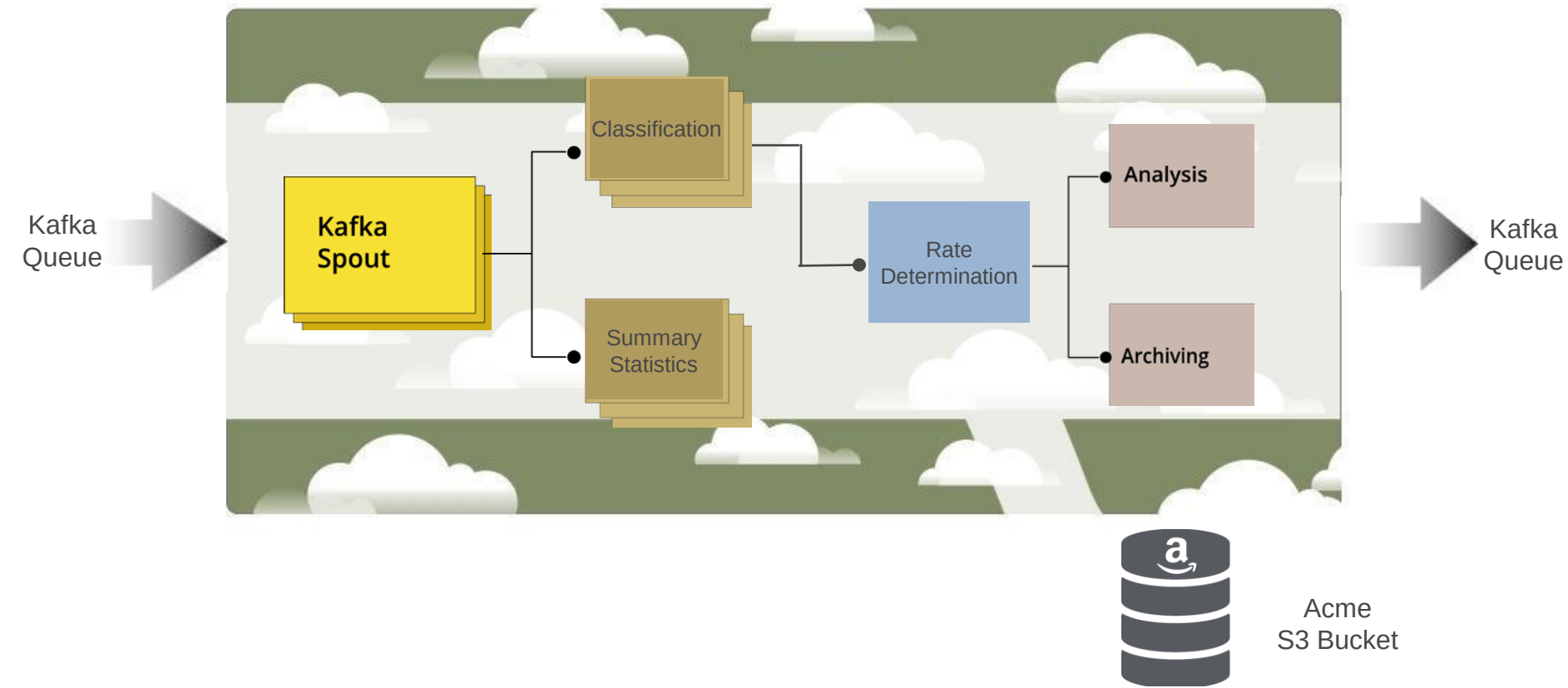
- C++ multi-threaded
- Boost ASIO framework
- Each Collector can handle 250k+ events per second
  - Per m2.2xlarge instance



1 x EC2 m2.2xlarge Collector instance (300 byte average event size).

Processing Events

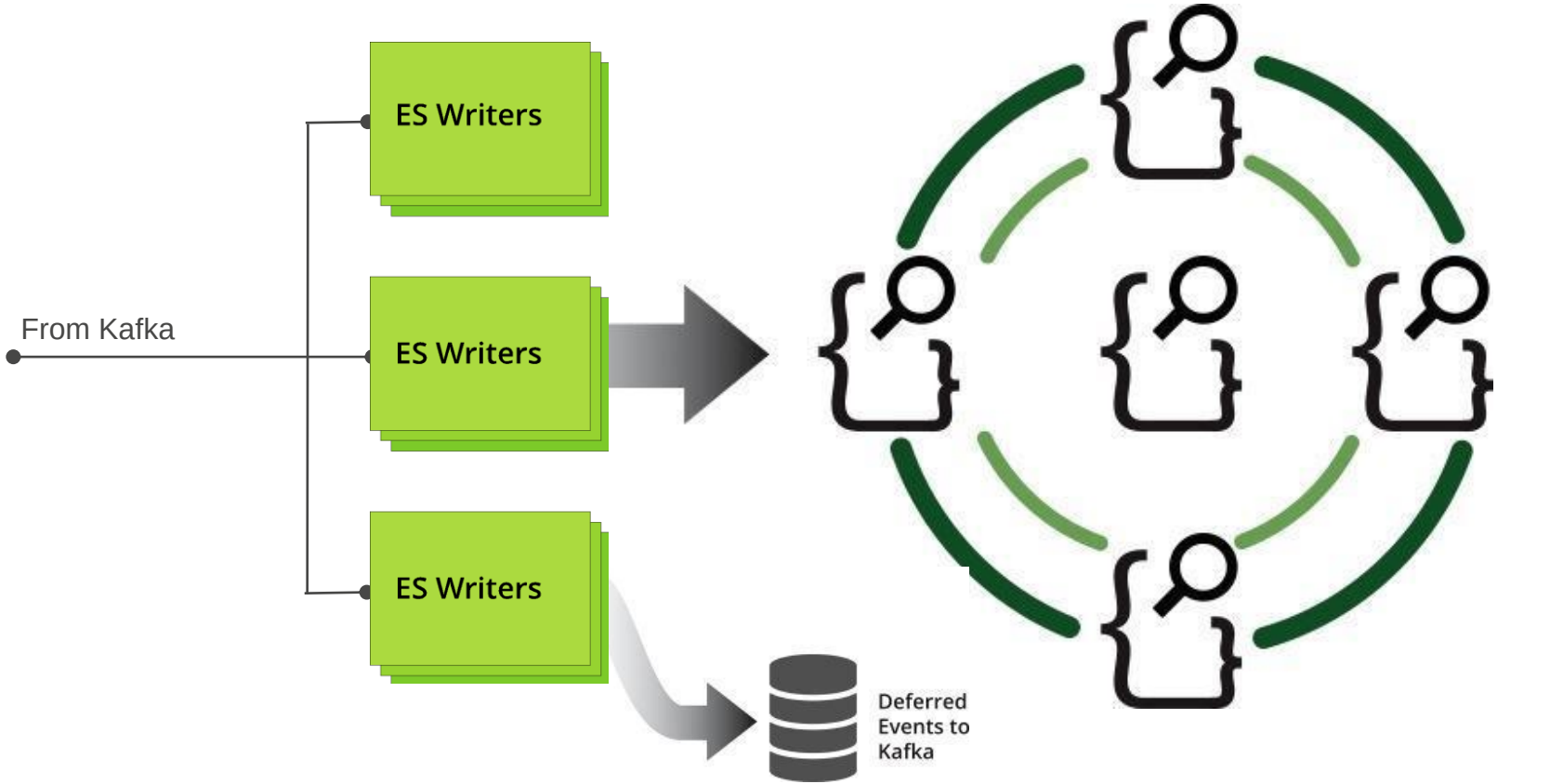
Storm Event Processing



Event Pipeline in Summary

- Storm provides *Complex Event Processing*
  - Where we run much of our secret-sauce
- Kafka contains both raw *and* processed Events
- Snapshot the last day of Kafka events to S3

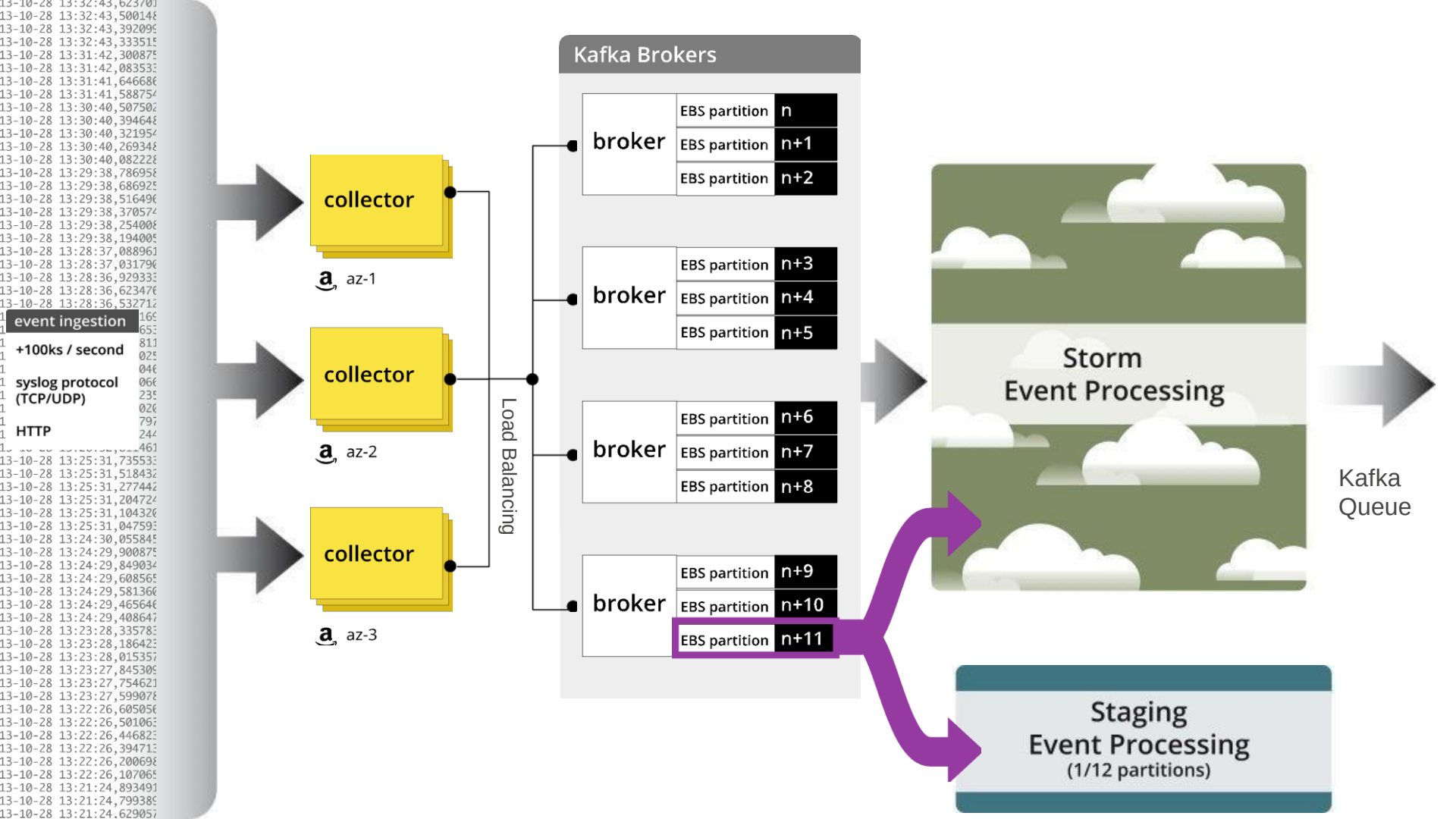
Elastic Search Clusters



Loggly and Index Management

- Indices are time-series data
  - Separated by customer
  - Represent slices of time
    - Higher volume index will have shorter time slice
- Multi-tier architecture for efficient indexing
  - Multiple indexing tiers mapped to different AWS instance types
- Efficient use of AWS resources

Staging Pre-Production System


event ingestion
+100ks / second
syslog protocol (TCP/UDP)
HTTP
collector
az-1
collector
az-2
collector
az-3
Load Balancing
Kafka Brokers
broker
EBS partition n
EBS partition n+1
EBS partition n+2
broker
EBS partition n+3
EBS partition n+4
EBS partition n+5
broker
EBS partition n+6
EBS partition n+7
EBS partition n+8
broker
EBS partition n+9
EBS partition n+10
EBS partition n+11
Storm
Event Processing
Kafka
Queue
Staging
Event Processing
(1/12 partitions)

Kafka enables Staging Architecture

- Kafka Broker doesn’t care if there are multiple consumers
- Staging system runs pre-production code
- Pub-sub allows us to randomly index a fraction of our production load
- A highly-effective pre-production system

AWS Deployment Details

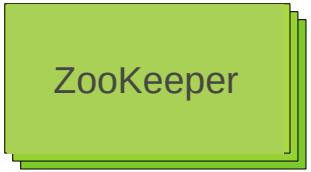
AWS Deployment Instances – Collection

c1.xlarge
- Compute-optimized
- High-traffic ingestion points
- Disk not important

m2.2xlarge
- Memory-optimized
- Disk buffer caching

4K Provisioned IOPs EBS
- Ensures consistent IO
- No noisy-neighbors
- Persistent storage

AWS Deployment Instances – Processing



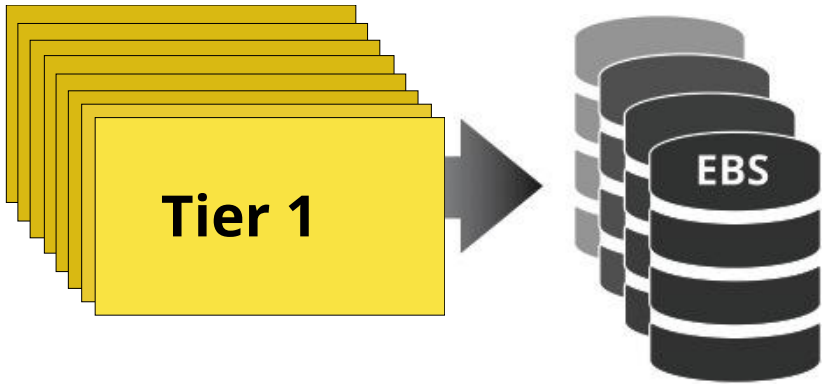
c1.xlarge

- Compute-optimized
- CPU-intensive processing
- Network IO

m1.xlarge

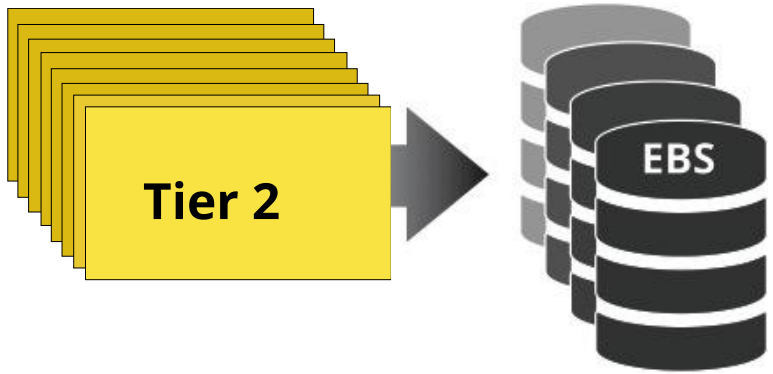
- General-purpose
- Configuration
- Management

AWS Deployment Instances – Indexing



cc2.8xlarge

- 4K Provisioned IOPs EBS



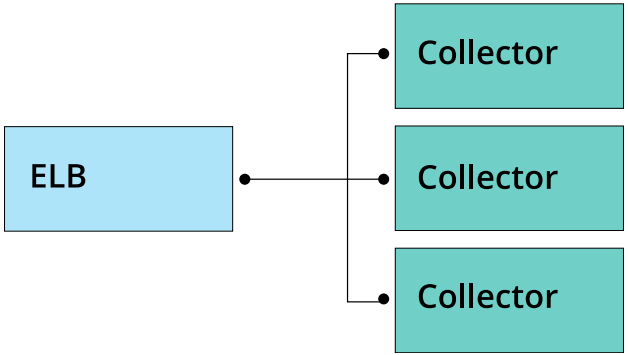
m2.4xlarge

- 4K Provisioned IOPs EBS

A Few False Starts

Elastic Load Balancing in front of Collector Had Limitations

- Initial testing used Elastic Load Balancing for incoming events:



- Elastic Load Balancing doesn't allow forwarding port 514 (syslog)
- Elastic Load Balancing doesn't support forwarding UDP
- Event traffic can burst and hit Elastic Load Balancing performance limits

Amazon Route 53 DNS Round Robin a Win

- DNS Round Robin is pretty basic load balancing
  - Not a bump in the wire
- Take advantage of AWS failover health checks
  - When a collector goes out of service, it will be out of the DNS rotation
- Round Robin across multiple regions, AZs
  - Latency based resolution optimizes inbound traffic

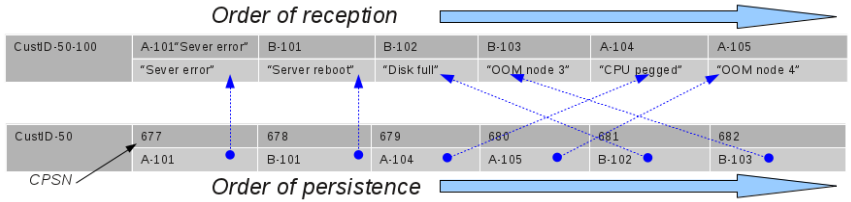
Our First Plan for Log Events

- Cassandra
  - Highly scalable key-value store
  - Impressive write performance a good match for us
  - Apache project plus commercial support with DataStax

- Use Cassandra for both our Event Queue and Persistent Store
  - Our strategy was to get the raw event in to Cassandra
  - …then perform workflow processing on events

Design meets Reality

- Cassandra not designed to be a message queue
  - Hard to

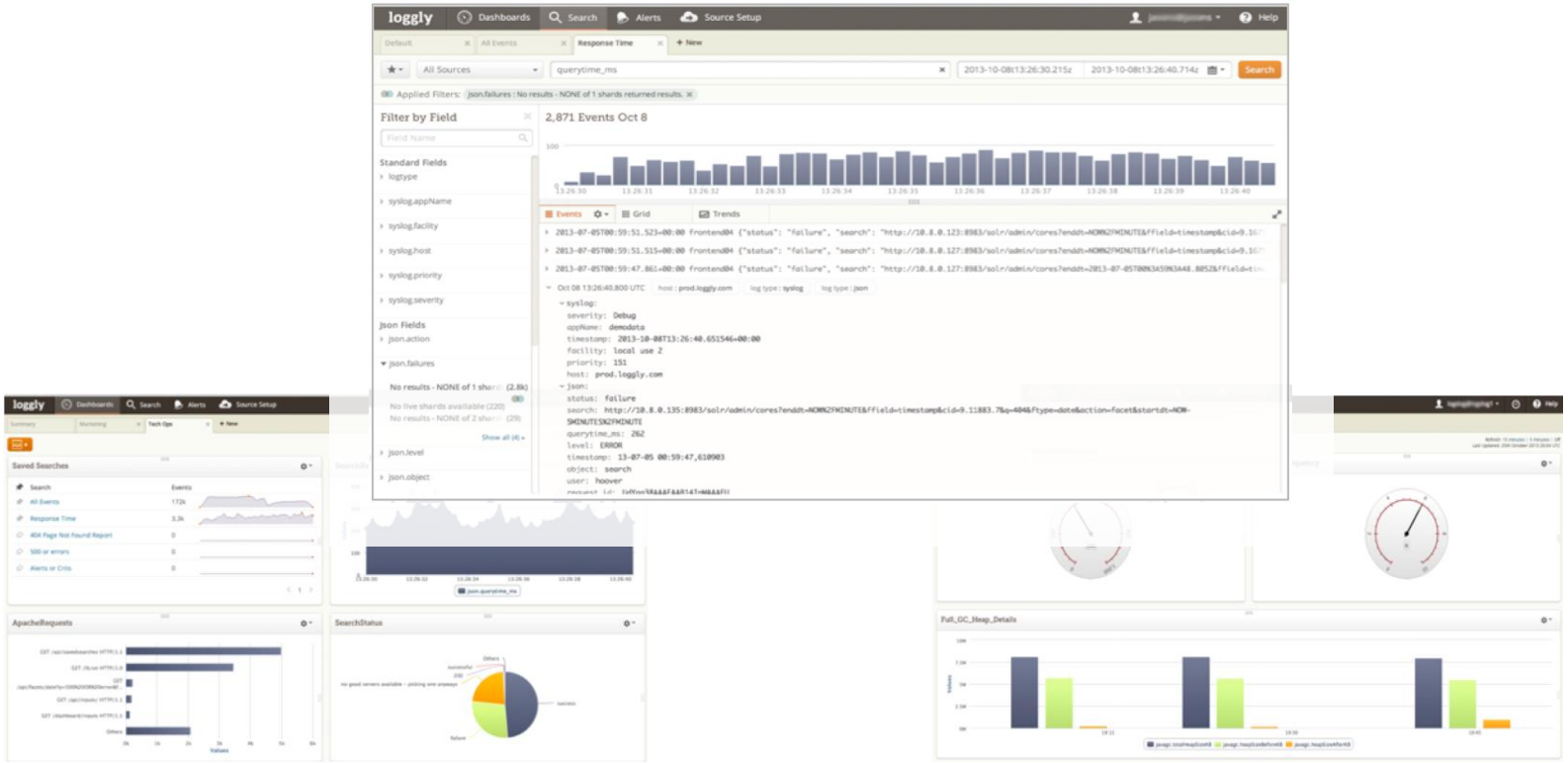


- Multi-tenancy requires handling data bursts
  - Collectors still needed to be able to buffer to disk
  - Added complexity and became a point of failure

Big Wins

- Leveraging AWS services
  - Multi-region, multi-AZ
  - Provisioned IOPS for availability and scale
  - Amazon Route 53 DNS support with latency resolution
  - Easy to increase and decrease Storm resources
- Leveraging Open Source infrastructure
  - Apache Kafka
  - Twitter Storm
  - ElasticSearch
- Pre-production “Staging” system

The Means to an End



Feedback

- Questions?

Jim Nisbet (niz@loggly.com)
CTO and VP of Engineering, Loggly

Philip O'Toole (philip@loggly.com)
Lead Developer, Infrastructure, Loggly

Follow us @loggly!

AWS re:Invent

Please give us your feedback on this presentation

ARC303

As a thank you, we will select prize winners daily for completed surveys!

amazon web services