Thriving Under a Continuous Self-Inflicted DDoS Attack

Kevin Beck
kbeck@newrelic.com
Safe Harbor

This presentation and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. (“New Relic”) to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic’s express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as “believes,” “anticipates,” “expects” or words of similar import.

Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic’s current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings New Relic makes with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic’s Investor Relations website at ir.newrelic.com or the SEC’s website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this presentation or otherwise, with respect to the information provided.
Agenda

- Introductions
- New Relic and High Volume Ingest
- Using Apache Kafka
- Techniques for Resilience
- Alerting
- Disaster Recovery
About New Relic
Exponential Growth of Incoming Data
Challenges

- Millions of agents send monitoring data once a minute
  - Autonomous
  - Customer driven
- Customers expect data to be processed and visible in real time
  - Alerts
  - Current view of performance
- Complex aggregation and processing of data
  - Stitch together metrics spread across multiple requests
  - Interpolate for missing and time-skewed data
Scope of Problem

- 177 K http requests / second
- 34 Gbps
- Data rate is increasing
  - Growing business
  - New features
- Multiplier: 34 Gbps in front door becomes 300 Gbps in Kafka cluster
Quality of Service Ideals

- No data loss
  - Ingest continues while pipeline is stalled
  - Deploying an update doesn’t disrupt service
  - Tolerate hardware failures
- Data visible within <N> seconds
- Resilient to common failures
- Survive loss of a data center
Apache Kafka + Microservices

Diagram:
- Producer
- Producer / Consumer
- Consumer
- Topics
Apache Kafka Components

Topic
“metric_data”
Apache Kafka Components

Topic "metric_data"

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitioned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Apache Kafka Components

Topic "metric_data"

partitioned

P0
P1
P2

replicated

P0
P1
P2
Apache Kafka Components

Topic "metric_data"
Apache Kafka Components

Topic “metric_data”

partitioned

replicated

ZK

P0

P1

P2

ZK

ZK

ZK

P0

P1

P2

ZK
High Level Architecture

Load Balancers → Vortex → Aggregators + Kafka Topics → New Relic Charts

Agents
Rent Versus Buy - Hardware

- Results of cost analysis evaluating use of a well known cloud provider
  - Buy and manage our own hardware
  - Lease space in data centers

- Gives us control to mitigate sources of latency
Standardized Hardware

- Use one common hardware configuration
  - Easy to repurpose machines
  - Simplified troubleshooting
- In the process of containerizing services

- Standard New Relic server
  - 2 sockets, 12 cores, 48 “threads”
  - 256 GB RAM
  - 2 x 10G NICs
  - RAID 6, up to 24 SSDs 800 GB
Kafka Best Practices – Fat Brokers

- Fat broker – entirety of a larger server dedicated to each broker
  - Fewer brokers => less to manage
  - Bigger brokers => easier to balance load
  - Kafka is a noisy neighbor - especially wrt network
  - More processing power in each broker avoids problems with e.g. coordinator overload
Kafka Best Practices – Low Retention

- Set data retention as low as practical
  - Faster admin operations including rebalance, adding partitions
  - Quicker to bring up a broker on new hardware
- Default retention of 1 hour
- One hour is enough time to respond to an incident, not necessarily resolve it
  - Provide a self-serve process to bump retention
  - Track and alert on undersized topics
    i.e. size limit hit before 1 hour time limit
- Dedicate as much memory as possible to filesystem cache
  - Big cache + low consumer lag => all reads serviced from cache
High Availability – N + 2

- Apply N + 2 rule
  - Replication factor 3 for Kafka topics
  - Three instances of critical services
- Load balancer + status checks to route traffic to available instances
- Go wide on all services
  - More instances => more capacity
  - N + 2 over capacity => resilience
- Canary + rolling deployments
High Availability – Key Alerts

- Status check on every service
  - Alert when no response, error response
- Throughput for relevant services
  - Alert when below minimum
- Consumer lag
  - Alert for lag > N seconds, varies by service
- Replication Factor 2, 1, unpreferred leader
- Zookeeper unable to respond to client requests, request queuing
High Availability – More on Alerts

- Appropriate level of alerting
  - One Kafka broker offline (replication factor 2 for those topics) => soft alert
  - Two Kafka brokers offline (no replication for shared topics) => page on-call person
- Tune thresholds for throughput, consumer lag
- Future: baselines
Resilience in Agents

Agents

Load Balancers

Vortex

Aggregators + Kafka Topics

Σ

λ

New Relic Charts

#VelocityConf
Resilience at the Edge

Agents → Load Balancers → Vortex → Aggregators + Kafka Topics → New Relic Charts
Vortex – Driving Principles

- Separate **persistence** of data from **processing**
  “separate ingest concerns from business concerns”
  - Absolute minimum of processing of incoming data before writing to disk
- Buffer between agents and collectors / aggregators
  - Problems in downstream processing result in data buffered preferable to back pressure against agents
  - Ingest should be able to continue in spite of failures of any/all downstream processing
- One app to rule them all
  - No more snowflake endpoints
    Gets rid of snowflake endpoints in previous incarnation
Vortex – Actions

- Reject bad requests
  - Garbage
  - Oversize payloads
- Authenticate user (license check)
  - Includes dealing with failures in account service
- Write to Kafka
- HTTP status returned to agent
Vortex – Architecture

Account Emitter

Load Balancers -> HTTP -> Vortex -> Local Kafka -> Mirror -> Main Kafka
Vortex – Components

- Single Kafka broker + Zookeeper
- Dedicated box for each instance
- Vortex uses Netty framework
  - Many threads for HTTP response
  - Lightweight, fast context switching
  - Deal with e.g. slow senders
- Mirror to copy topics to main cluster
  - Uses LMAX disruptor
- What about replication?
  - Messages fly through in a few ms during normal operation
Vortex – Problem Mitigation (Back Pressure)

- Main Kafka cluster stops accepting data
  - Messages accumulate in Vortex clusters
  - Alerts fire, people are paged
  - Afterwards, messages produced as fast as main cluster can take them
Vortex – Problem Mitigation (Hardware Failure)

- Server dies (e.g. hardware failure)
  - Agents time-out waiting for HTTP status
  - Load balancers detect failure, route to remaining Vortex instances
  - Failing status checks => alert
  - Agents retry and succeed
  - Tiny amount of data lost:
    - Messages written to local Kafka, not yet produced to main cluster
    - Data loss proportional to consumer lag, ~ 15 ms
Disaster Recovery

- **Plan A**
  - Re-instantiate everything in a cloud provider
  - Annual exercise to practice

- **Plan B (in progress)**
  - Three data centers in same city
  - Linked by dedicated fiber
  - All services spread across data centers

- **Goal:** services continue after loss of a data center
Stretch Cluster - Overview

Agents → Vortex

Data Center 1

Data Center 2

Data Center 3
Stretch Cluster - Data Replication

Agents → Vortex → Data Center 1 → Data Center 2 → Data Center 3

Produce → Replicate

#VelocityConf
Stretch Cluster - Network Partition

Agents  Vortex

Data Center 1

Data Center 2

Data Center 3
Summary

- New Relic ingests a lot of data quickly and reliably
- Keys to success
  - Microservices built on Apache Kafka
  - Monitoring and alerting
  - Architecture for resilience
- Disaster recovery