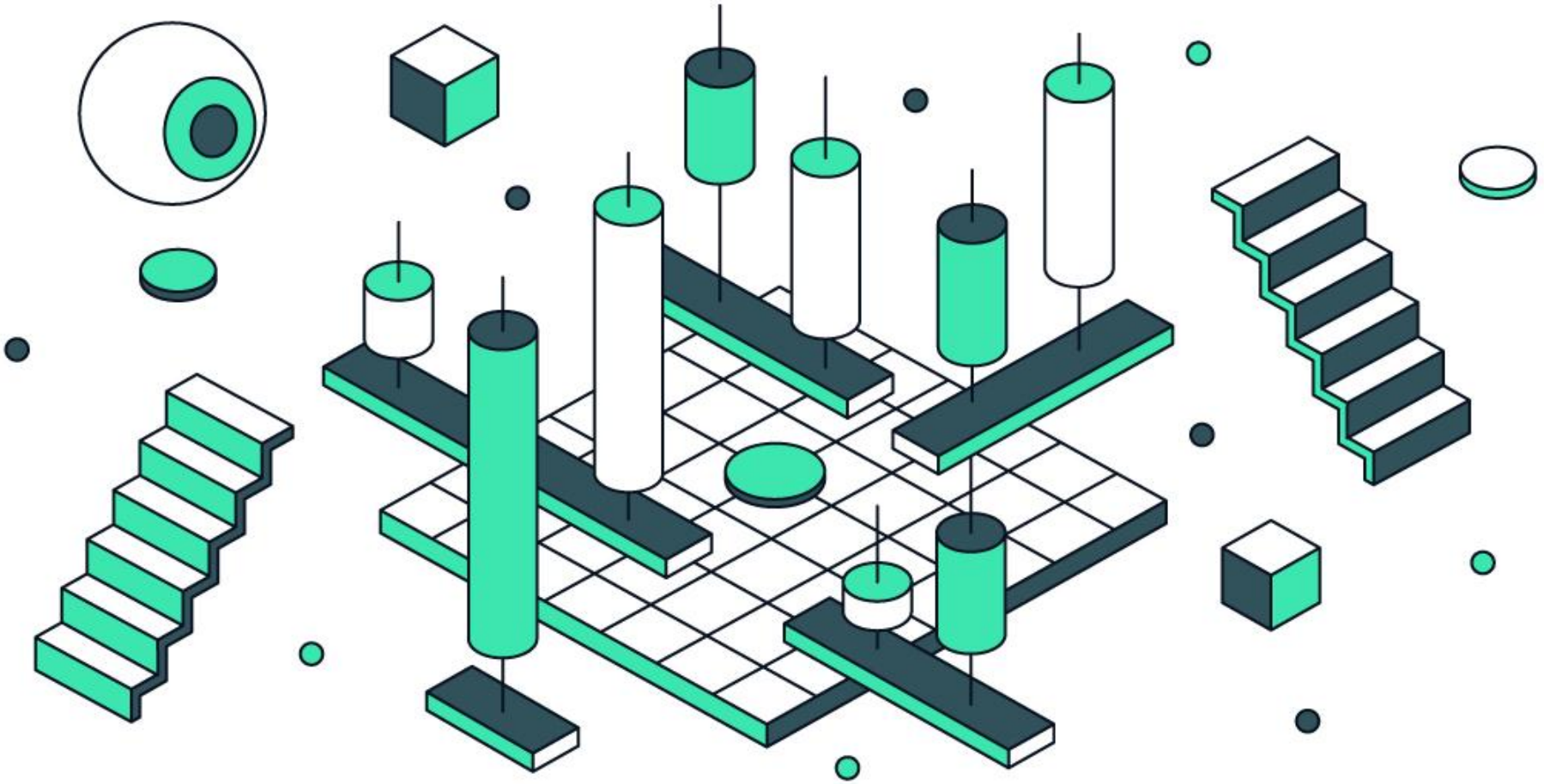
Tackling Kafka, with a Small Team

@JarenGlover



Organization Evolution  Engineering Trade Offs

circa 2015

Product Analytics

We wanted to build our own event pipeline.

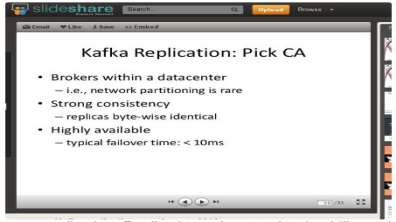
Kafka Concerns

Jepsen: Kafka

2013/09/24
Software Network Distributed Systems Jepsen Kafka

In the last Jepsen post, we learned about NuoDB. Now it's time to switch gears and discuss Kafka. Up next: Cassandra.

Kafka is a messaging system which provides an immutable, linearizable, sharded log of messages. Throughput and storage capacity scale linearly with nodes, and thanks to some impressive engineering tricks, Kafka can push astonishingly high volume through each node; often saturating disk, network, or both. Consumers use Zookeeper to coordinate their reads over the message log, providing efficient at-least-once delivery—and some other nice properties, like replayability.



In the upcoming 0.8 release, Kafka is introducing a new feature: replication. Replication enhances the durability and availability of Kafka by duplicating each shard's data across multiple nodes. In this post, we'll explore how Kafka's proposed replication system works, and see a new type of failure.

Here's a slide from Jun Rao's overview of the replication architecture. In the context of the CAP theorem, Kafka claims to provide both serializability and availability by sacrificing partition tolerance. Kafka can do this because LinkedIn's brokers run in a datacenter, where partitions are rare.

Note that the claimed behavior isn't impossible: Kafka could be a CP system, providing "bytewise identical replicas" and remaining available whenever, say, a majority of nodes are connected. It just can't be fully available if a partition occurs. On the other hand, we saw that NuoDB, in purporting to refute the CAP theorem, actually sacrificed availability. What happens to Kafka during a network partition?

Real Time Business Decisions

Run Experiments

Actual Application Visibility


PD

Application Engineer’s Perspective

Application Engineer’s Perspective

- **Implicit Trust**

Application Engineer’s Perspective

- Implicit Trust
- Retry Logic

Application Engineer’s Perspective

- Implicit Trust
- Retry Logic
- Library Ownership

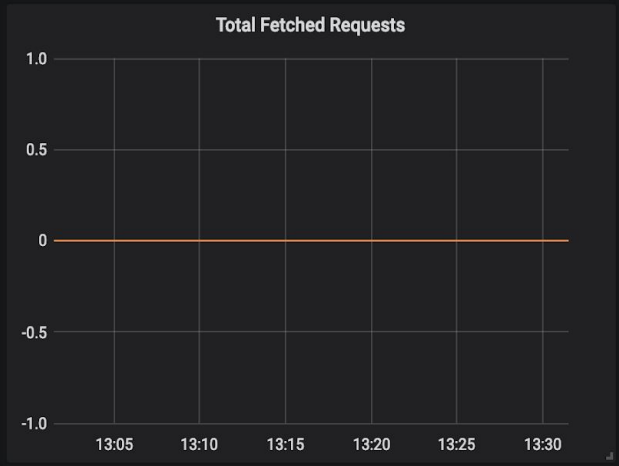
Infrastructure Engineer’s Perspective

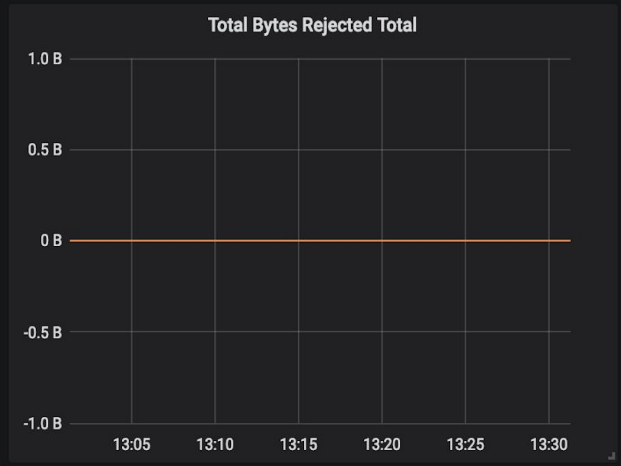

Congratulations. You played yourself.

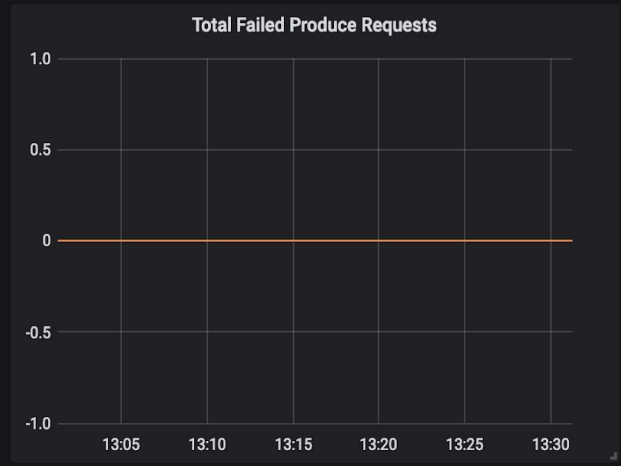
Confluent Broker

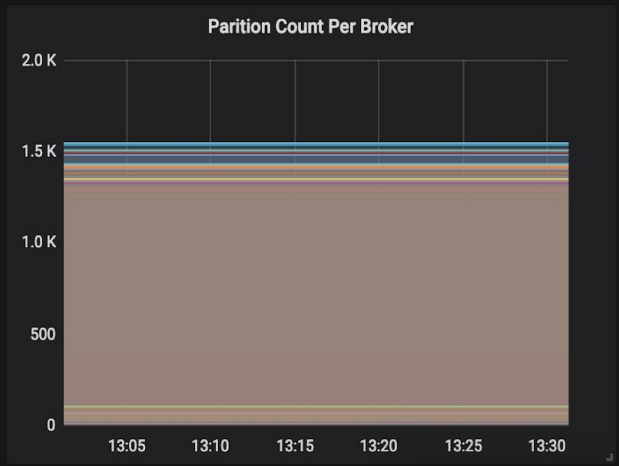
Last 30 minutes

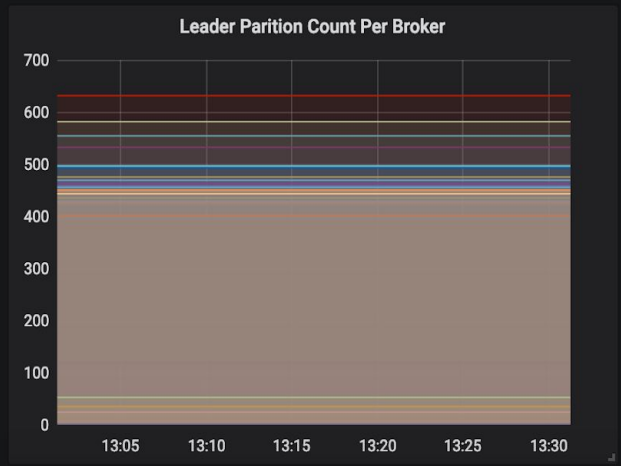
role confluent-broker hosts All

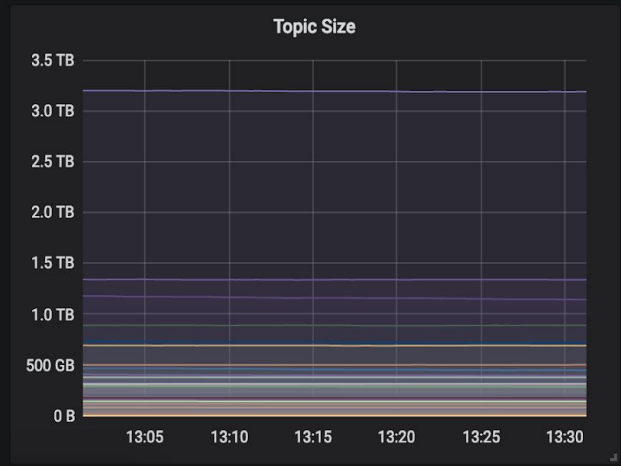












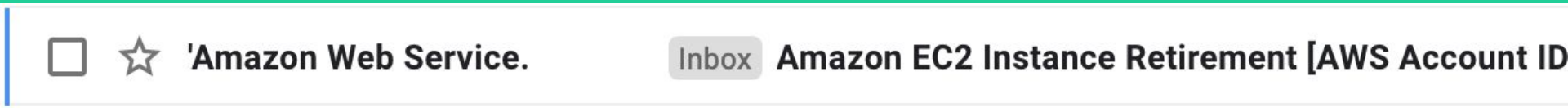
Infrastructure Engineer’s Perspective

- Suffering from Success
- Technical Whack-A-Mole
- Library Ownership

Engineering Trade Offs

What We Got Right

Replacing Unhealthy Brokers


'Amazon Web Service.
Inbox
Amazon EC2 Instance Retirement [AWS Account ID

Brokers

Id	Host	Port
0	10.2.4.64	PLAINTEXT:9092
1	10.2.4.32	PLAINTEXT:9092
2	10.2.4.12	PLAINTEXT:9092
3	10.2.4.80	PLAINTEXT:9092
4	10.2.4.9	PLAINTEXT:9092
5	10.2.4.60	PLAINTEXT:9092

… Recently Got Right

Tooling for Rolling Restarts


NICKTOONS
NETWORK

Thanks Prometheus and Fabric

```
1       def restart_service(self):
2           self.ensure_healthy_cluster()
3           for host in self.get_hosts():
4               self.restart_process(host)
5               self.after_restart_wait(host)
6               self.ensure_healthy_cluster()
```

Soon to be Right ...

Overhaul the Client Library

Shared Library Improvements

- More Implementation Structure

Shared Library Improvements

- More Implementation Structure
- Observability for Free

Shared Library Improvements

- More Implementation Structure
- Observability for Free
- Integration and Regression Testing

“You can do it too!”