#ASLI ENGINEERING

GitHub Outage: Chaos in the Zookeeper Cluster

SWIPE

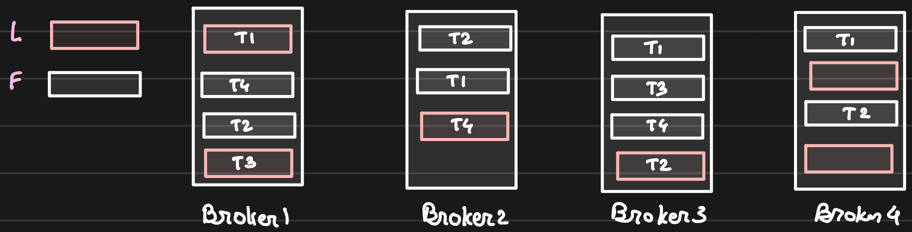
BY

ARPIT BHAYANI

Dissecting GitHub Outage

The second leader problem

Zookeeper and Kafka



1. Controller Election : leader / Follower for all partitions. ZK ensures that if leader goes down, Follower becomes the leader

2. Cluster Membership : Managing which nodes are part of cluster

3. Topic Configuration : list of topics, # partitions for each topic, location of replica, leader node location, and many more

4 ACL and Quota : who is allowed to read / write and how much

Zookeeper is an extremely important component for Kafka. It is the Brain that holds the most important info

* Newer version of Kafka does not rely in Zookeeper

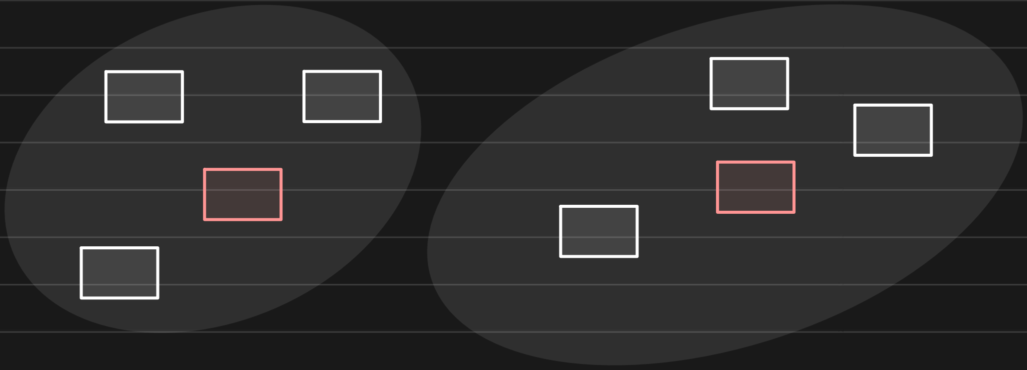
ARPIT BHAYANI

Version upgrade,
OS patch, Security Patch,

Zookeeper routine maintenance

Nodes of Zookeeper cluster needs to be upgraded
hence new nodes are added and then old ones are removed

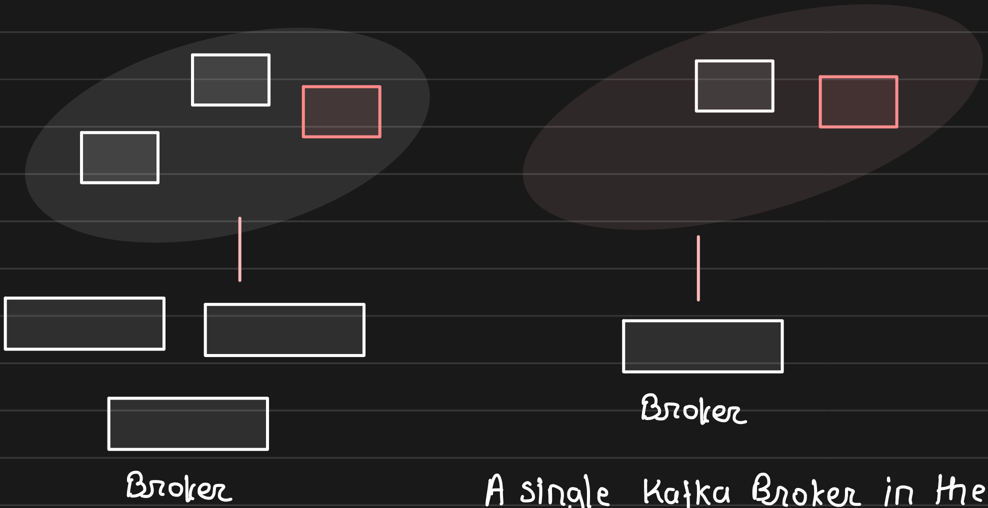
New nodes were added "too quickly" to zookeper cluster



When new nodes were added "quickly", it
resulted in another leader Election

Zookeeper is near-autonomous. When a new node is added to the cluster it tries to self-discover. If a lot of new nodes are added, they could not 'discover' leader and thought they are leaderless and hence triggered a leader Election

Split Brain



A single Kafka Broker in the cluster connected to the newly formed Zookeeper cluster and elected itself as Controller [controller for a topic]

* When clients are connecting to Zookeeper for Kafka Details, they are getting Conflicting Information

This led to failures of writes,

until the clients discover, some writes would have happened through new node controller.

Recovery

Zookeeper auto-detects this inconsistency over-time and auto-heals

we can also manually take actions to fix it

$\downarrow$

killing the second logical cluster

What affected?

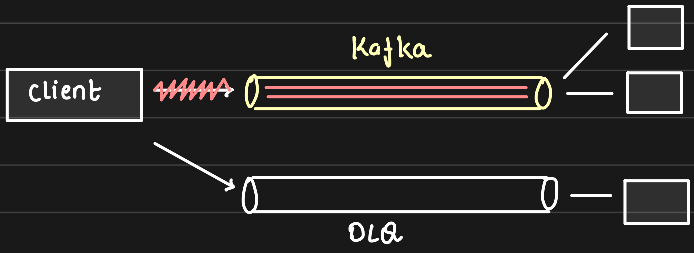
The Kafka Cluster where this happened, handled internal background jobs

So, did they lose any jobs? No!

Fallback Queues

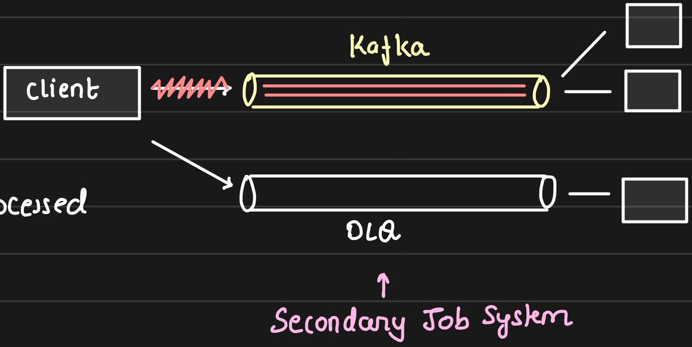
A standard architecture requires you to have Dead letter Queues.

Idea: if unable to write message in the main queue, we put the same message in DLQ which are then later processed



For a high write ingestion, DLQ would overload.

But with retries and one time processing, it could be processed quickly



Important learnings

1. Having a DLQ is a must
2. Consumers should be idempotent
3. Clients and consumers should have retries
4. Automate cluster provision with jitter