# Stories from the Trenches, Apache Kafka episode 11

Pere Urbon-Bayes

@purbon

Technology Architect

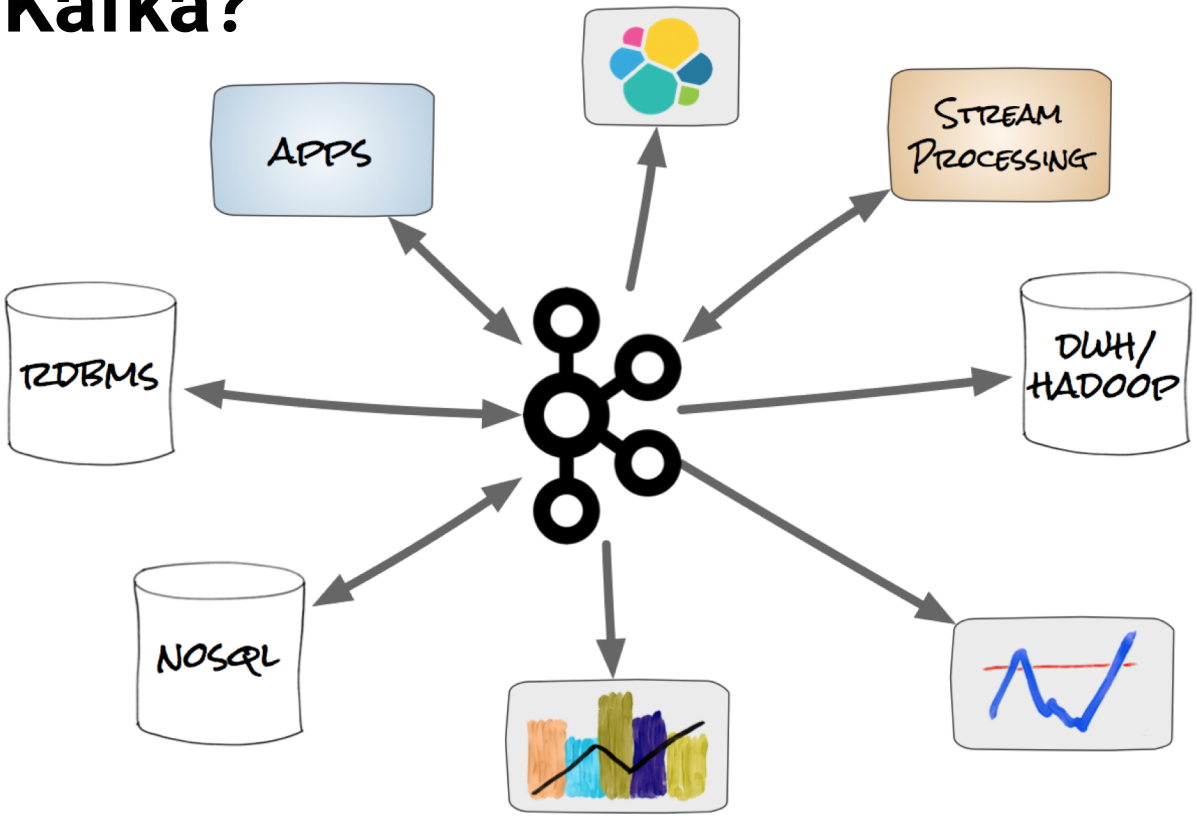
Confluent

# Topics for today

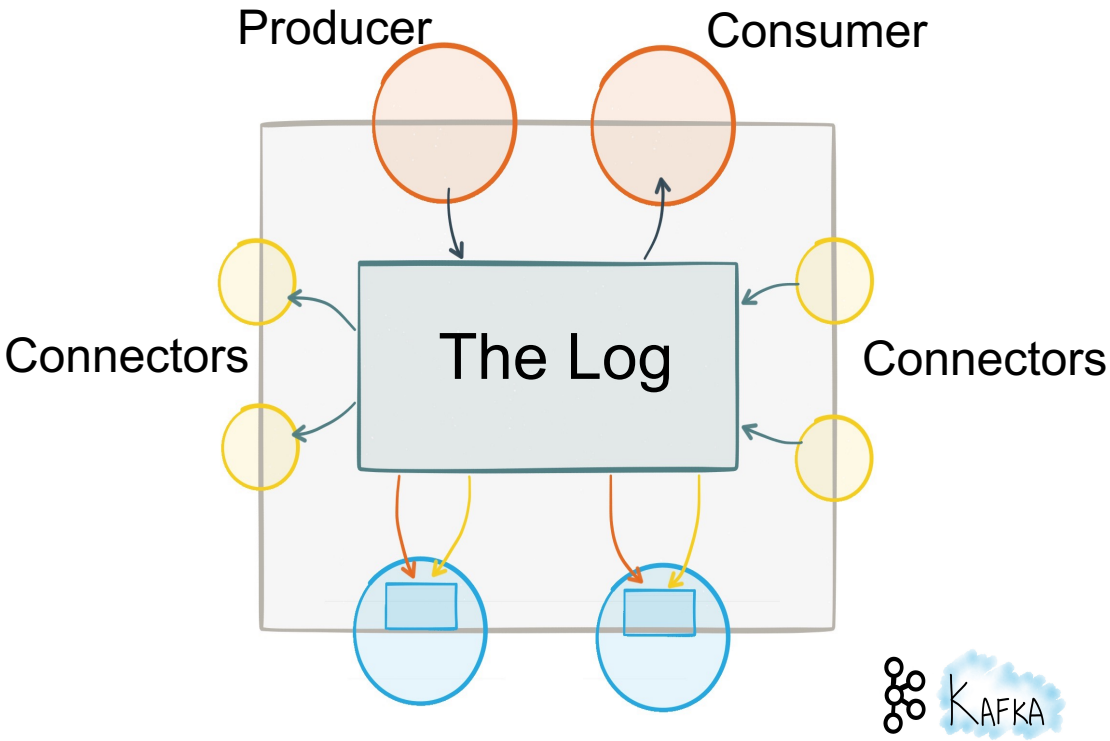
1. Apache Kafka and it’s internals.
2. Stuff that usually makes your head around
    a. Understanding data durability
    b. Under Replicated Partitions
    c. Message ordering in Apache Kafka
    d. Partition reassignment storm?
    e. Taking care of Zookeeper
    f. Monitoring
    g. Security
3. Recap

# Apache Kafka internals report

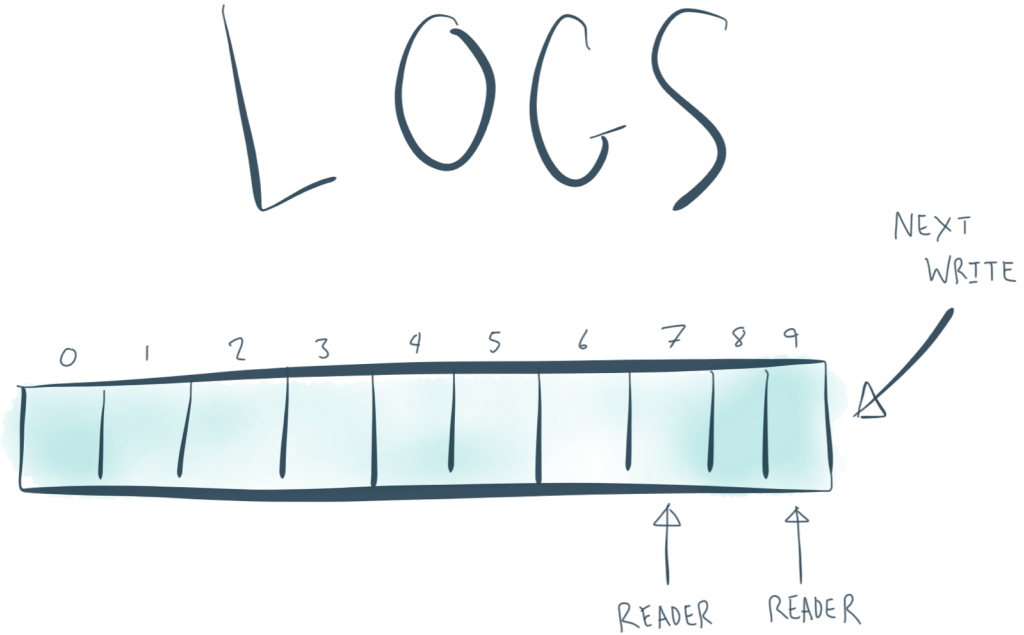
# What is Kafka?



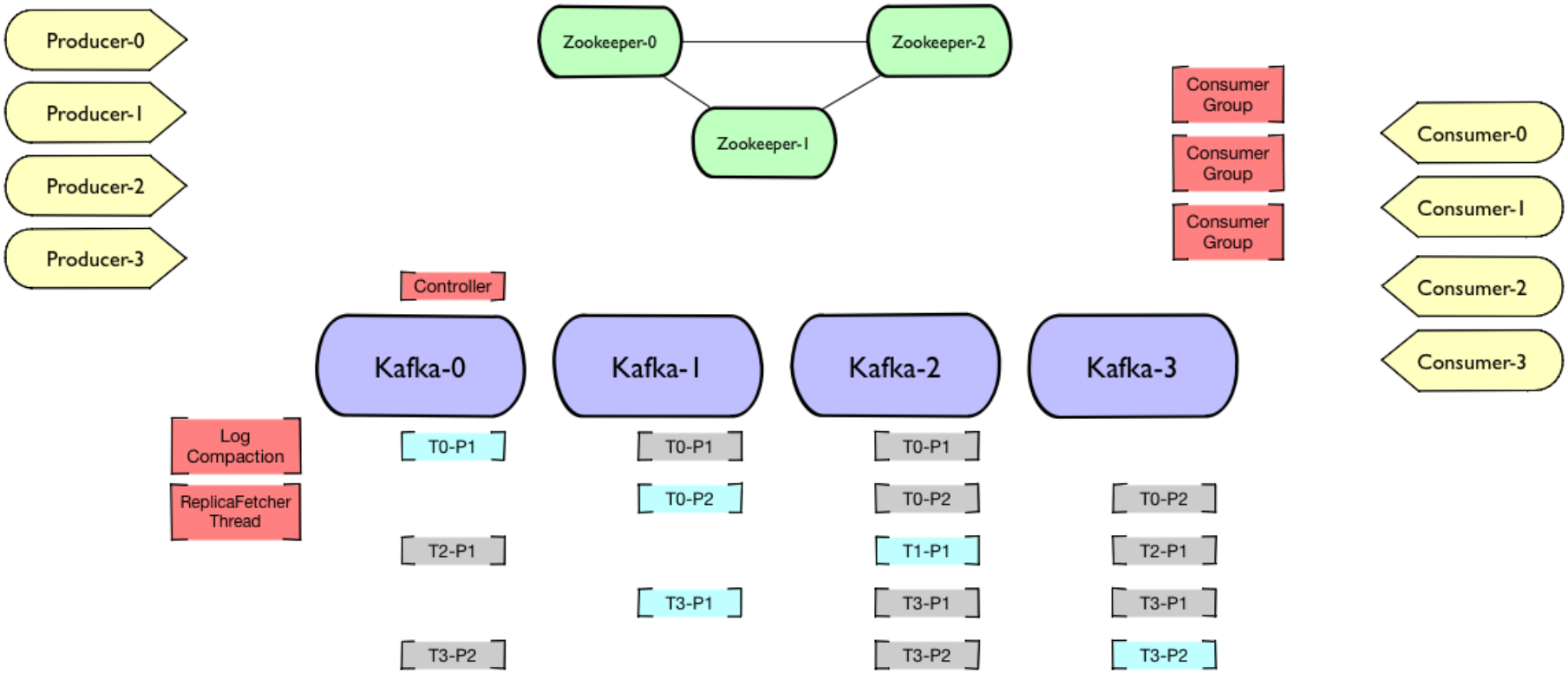
# An Streaming Platform



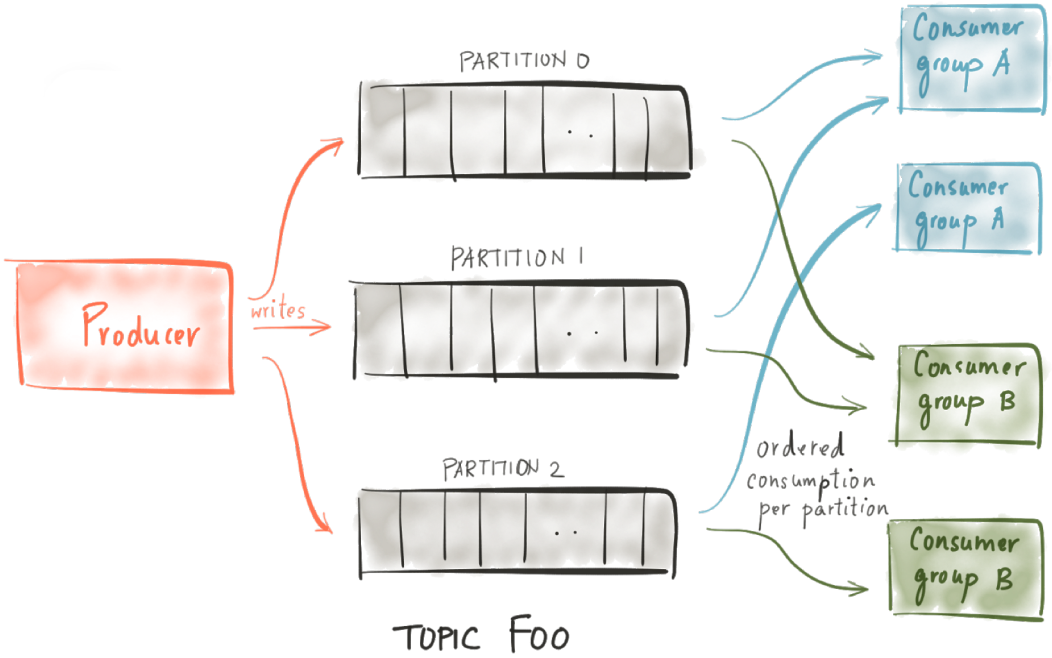
# The Distributed Log



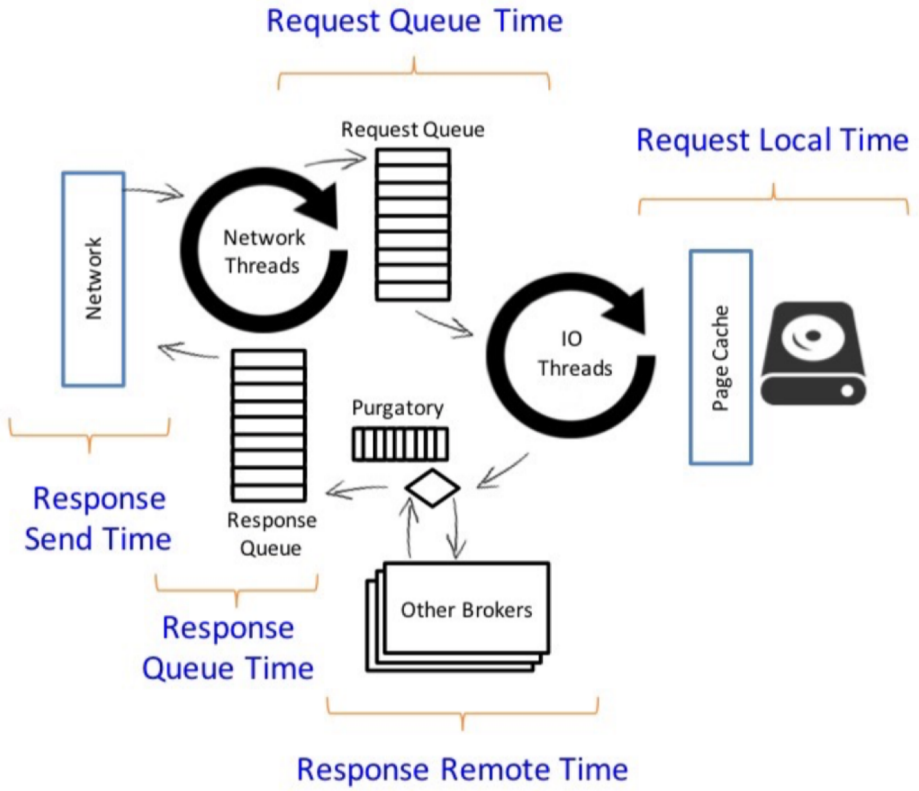
# Apache Kafka, a distributed system



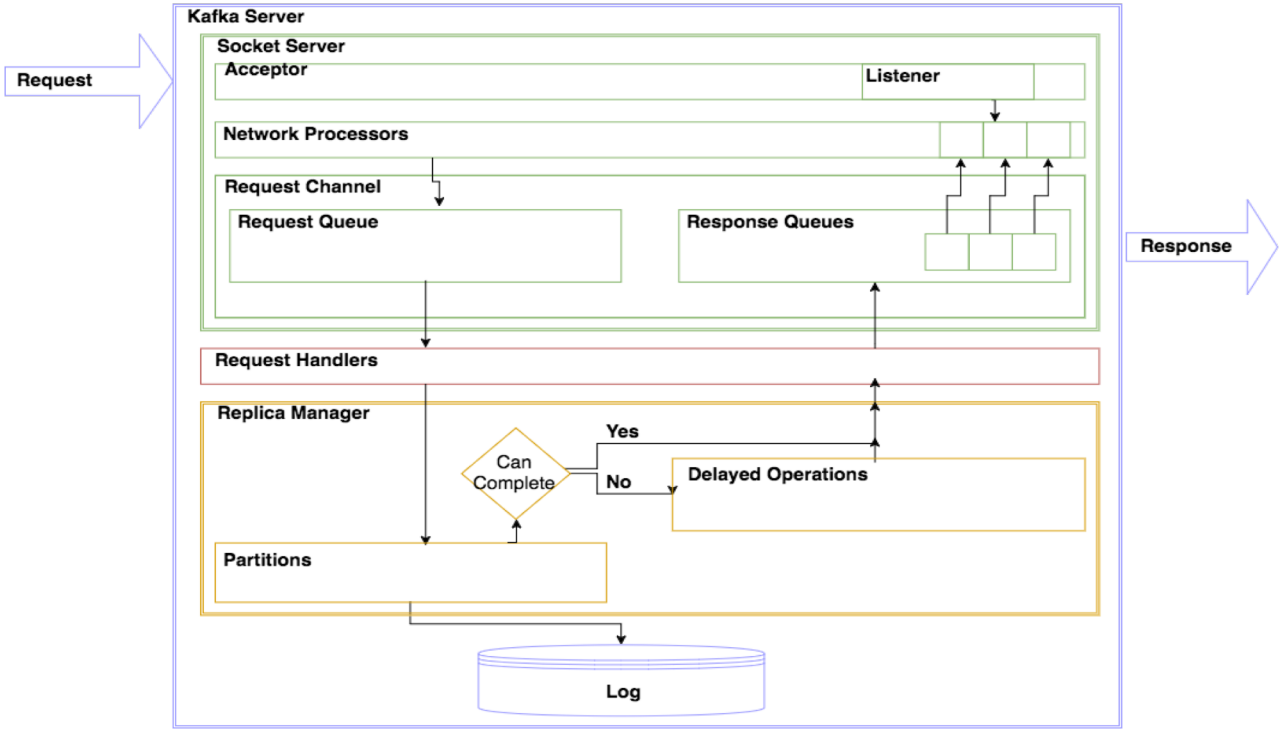
# Scalable Consumption



# Understanding the process of a Request



# The Kafka Broker Key Resources



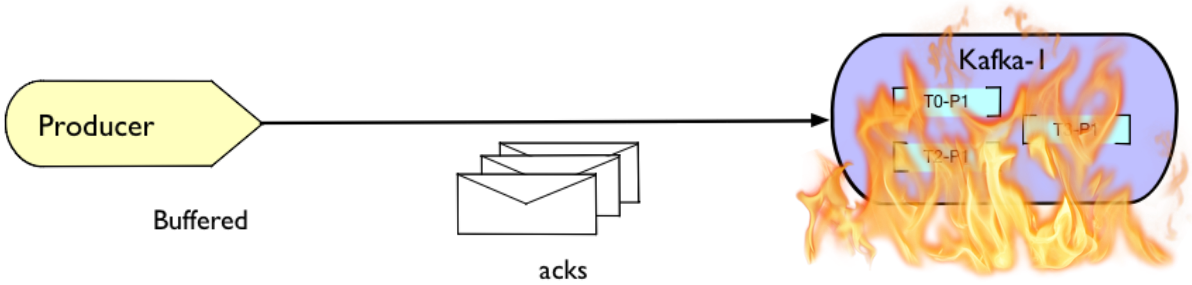
# Challenge 1
# Understanding Durability

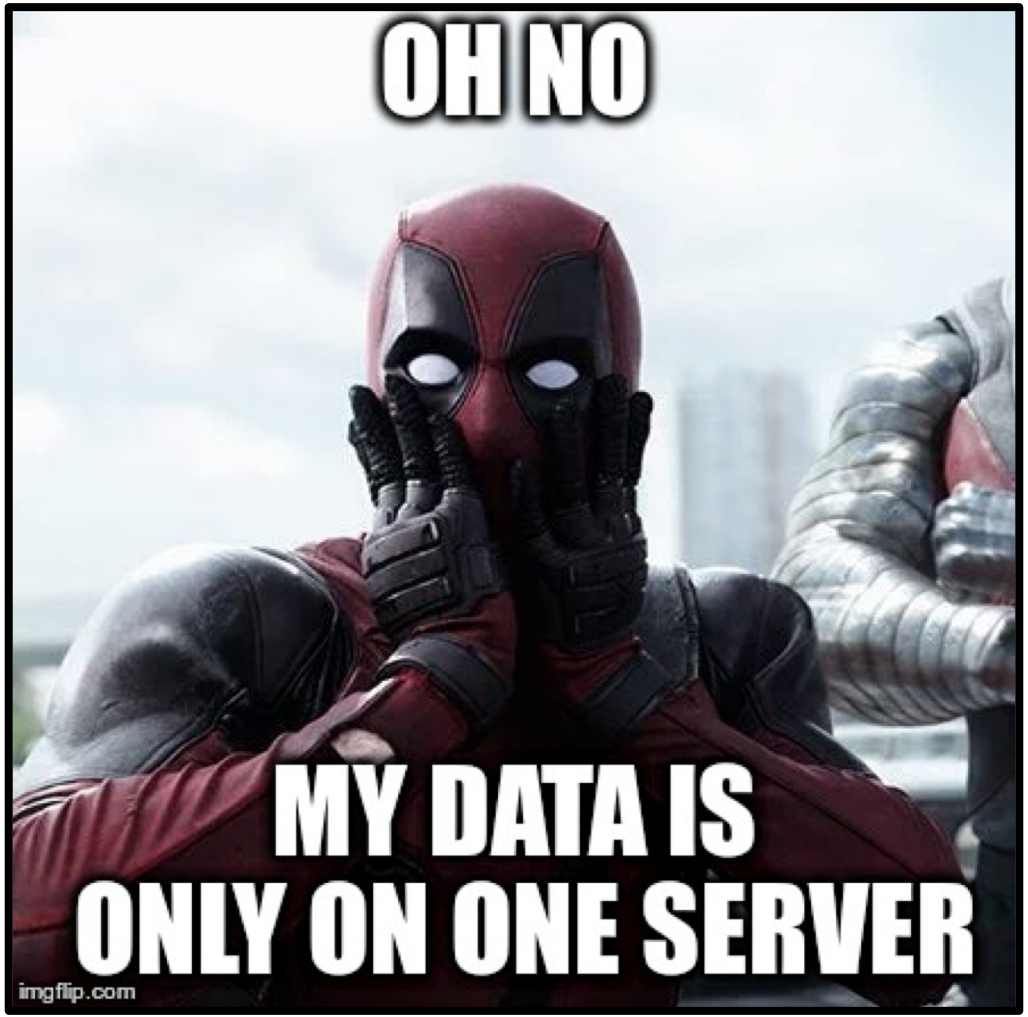
# In a wonderful world scenario?

- A producers send a bunch of messages asynchronously
- The partition leader receive the message and update the open segment.
  - If more than 1 partitions, replication and high watermark protocol kick in.
  - All the new data is replicated to all the replicas.
- No exception (KafkaException) is returned to the client, the producer continue without interruption

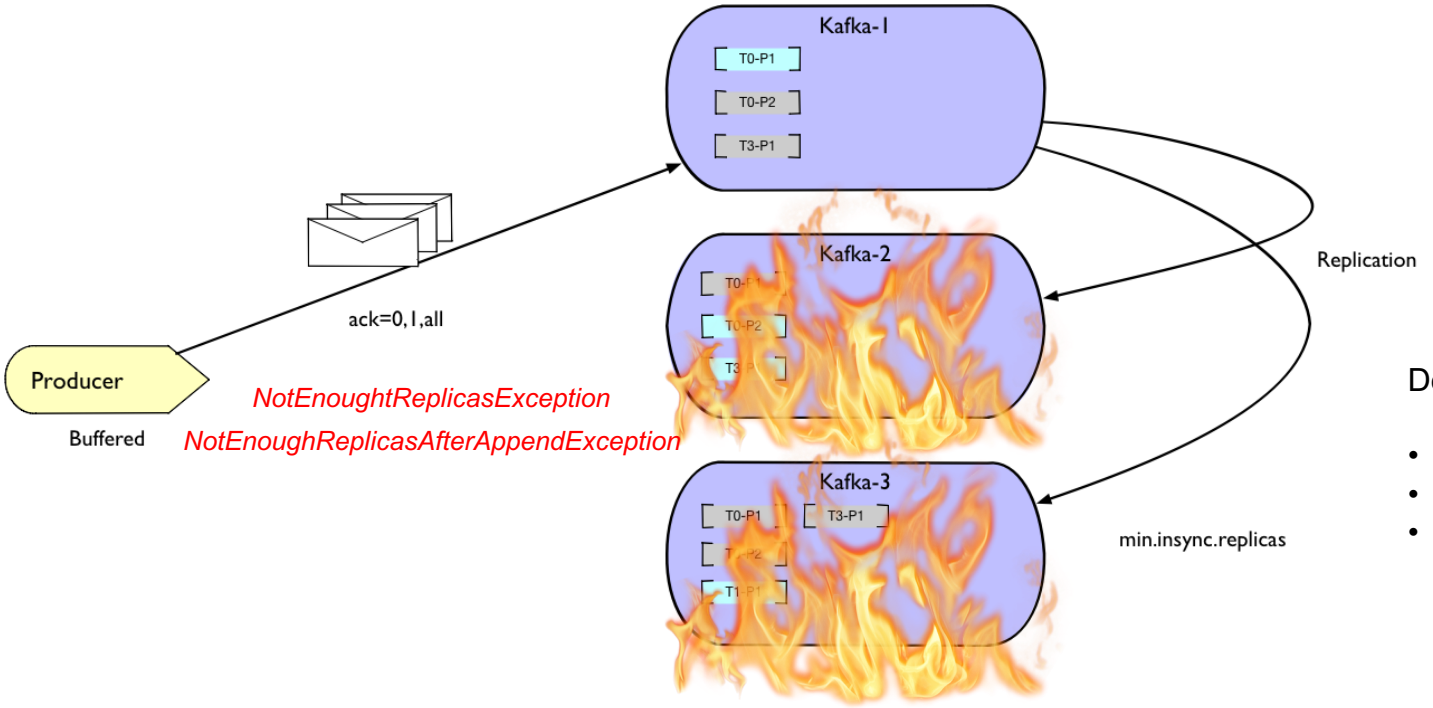
But **bad things** happen in distributed systems

# The Route of a Message in Apache Kafka




OH NO
MY DATA IS
ONLY ON ONE SERVER
imgflip.com

# The Route of a Message in Apache Kafka




Defaults:

- acks=1
- replication.factor=1
- min.insync.replicas=1

# Optimized for Availability and Latency over Durability

# Durability is achieved through replication

# Understanding durability

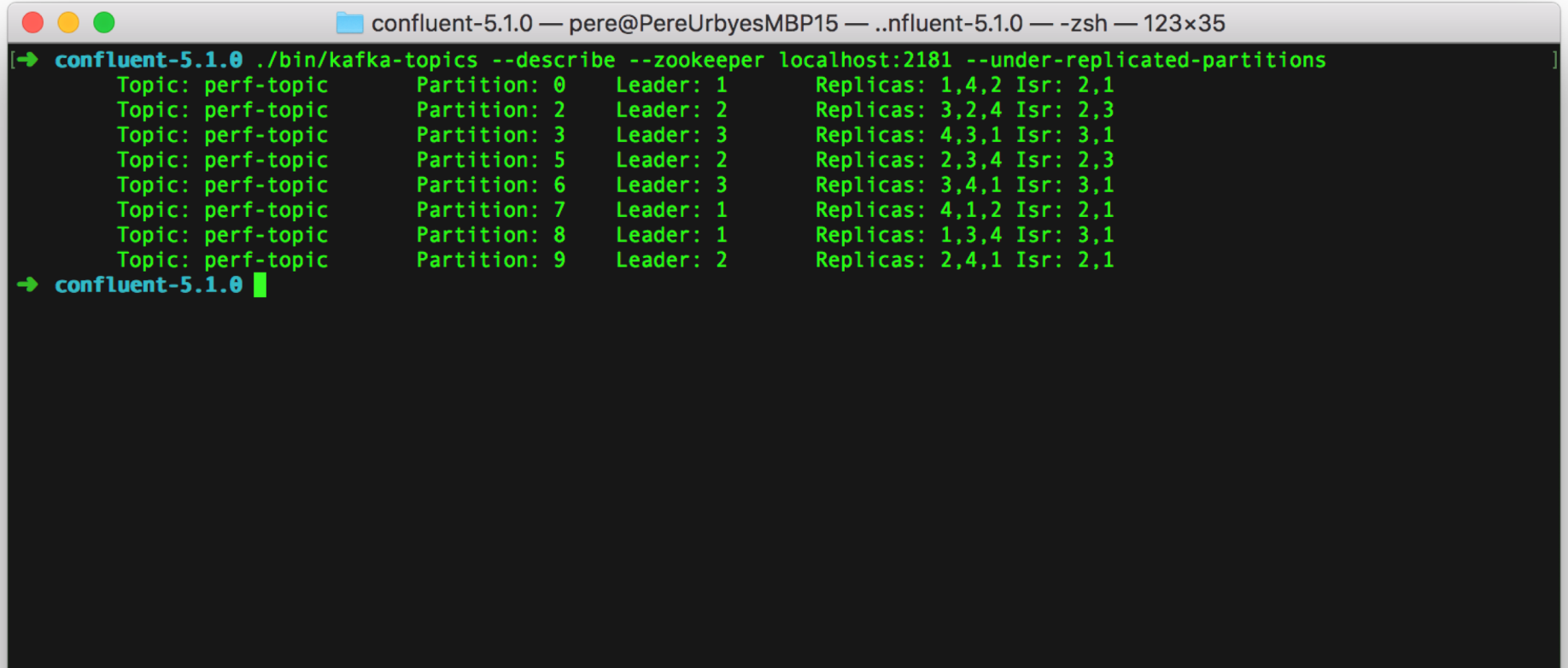
- Durability is achieved trough replication
  - In the Producers
    - Using Acks=0 is equivalent to fire and forget (fast but could be unreliable)
    - Using Acks=all is resilient, but you will achieve less performance
  - In the brokers
    - For a topic with N replicas, use min.insync.replicas = N-1 (strict) or min.insync.replicas = N-1 (less)
    - The min.insync.replicas should be 2 to keep always more than 1 data copy.
    - The replication factor should be minimum of 3.
      - replication factor should be 4 ( or in multiples of 2) in scenarios of 2 DC.

# Challenge 2
# Under replicated partitions

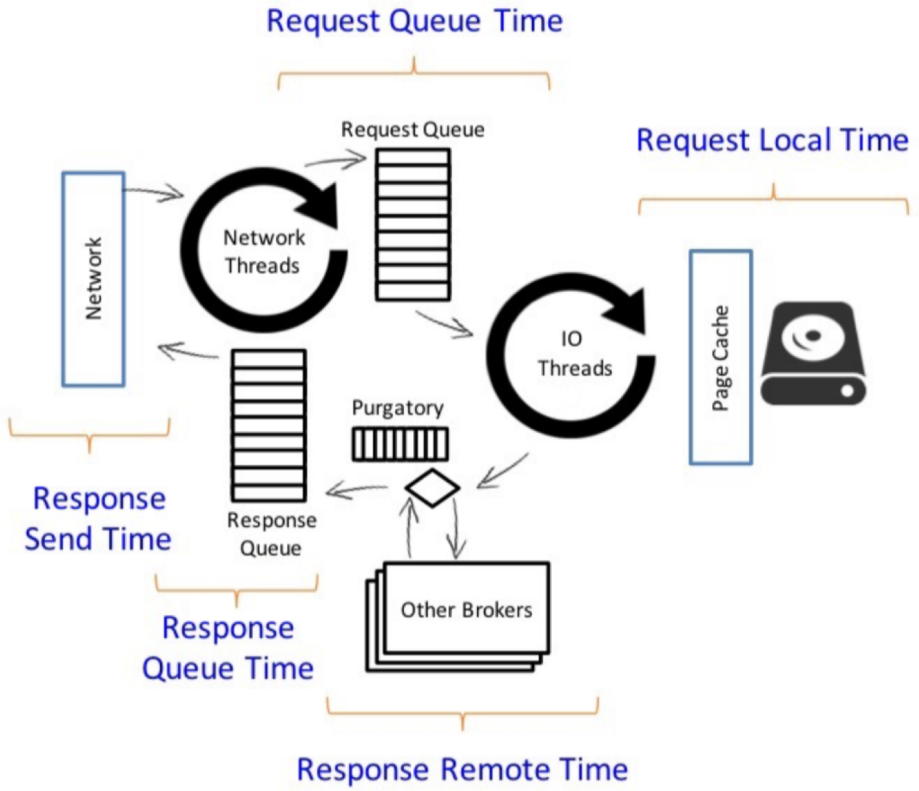
# What is under replicated partitions?

- All writes and reads goes to the primary partition.
  - The primary partition is elected using zookeeper.
- Once the data is received the replication process starts by the ReplicaFetcher thread.
  - The **high watermark offset** is moved around.
    - A consumer can only read up to the high watermark offset to prevent reading under replicated messages
- When all acks and min.insync.replicas are copied over, positive response is back.
- There are situations were URP is normal, but usually if you have URP is a sign something wrong is going on.

# What is under replicated partitions?



# The Anatomy of a Request



# Under replicated partitions

- Description:
  - You start seeing an increased number of under replicated
    - Even a few of your topics could be offline for now
  - If you stop your producers, the cluster does not heal over time.
  - If you restart the problematic nodes, everything works again.
  - When you start your producers, the cluster goes back to URP.

# Under replicated partitions

- Scenario:
  - Your Kafka cluster is version 0.11.x
  - Network and  IO utilization is normal
  - The issue does not heal itself (remember URP might be transitory)
  - You're seeing in your logs: "*OffsetOutOfRangeException*" or "*FATAL [ReplicaFetcherThread-0-3]: Exiting because log truncation is not allowed for partition*"

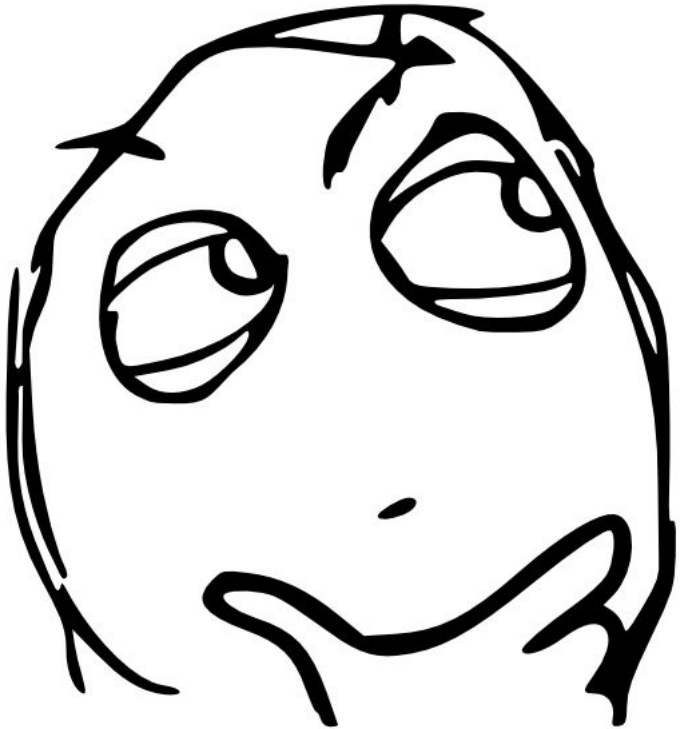
# Under replicated partitions

- Cause:
  - Update to a new version is necessary
  - This bug occurs when a fetch response contains a high watermark lower than the log start offset
  - Easily reproducible by creating a replicated topic configured with compact+delete and a low retention value, and writing data older than the retention value quickly from a producer
  - You hit an instance of https://issues.apache.org/jira/browse/KAFKA-5634
  - The cluster will not recover as data is watermarks are broken

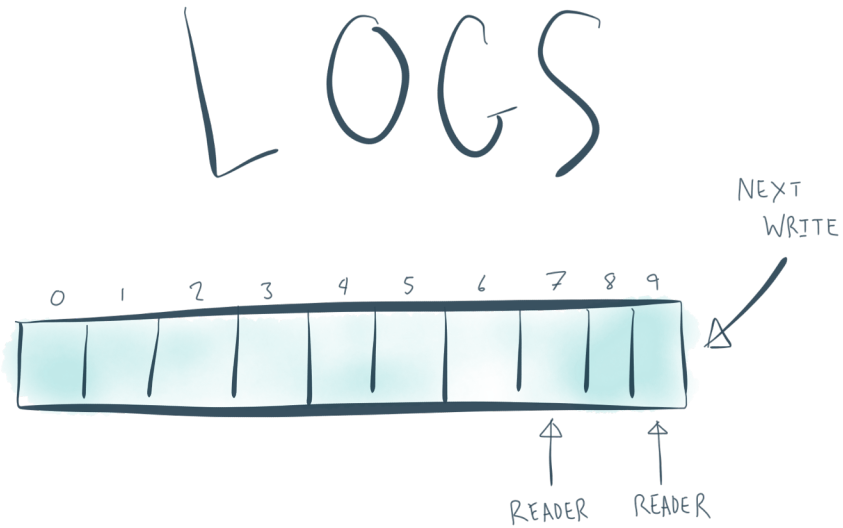
# Challenge 3
# Interested on keeping order?

# Keeping order in Apache Kafka?

- Does this sounds familiar so you?
  - Your producers are sending message to Apache Kafka without problems
  - The consumers are not processing the message in the expected order
  - This could happen for many reasons, so your start wondering….



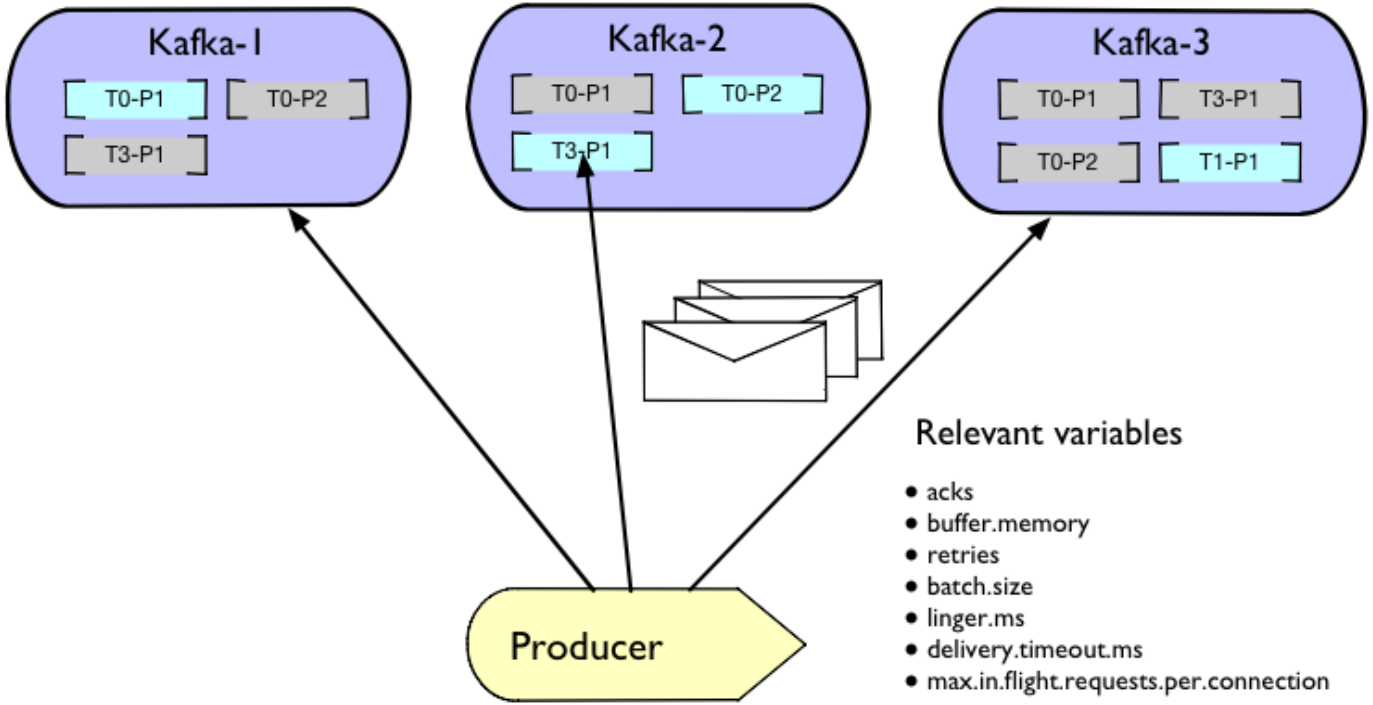
# What might be happening? Many moving parts




# Keeping order in Apache Kafka

- Apache Kafka guarantees write order per partition.
  - 1 topic will have N partition where N is >=1
- Partition offsets are always monotonically increasing

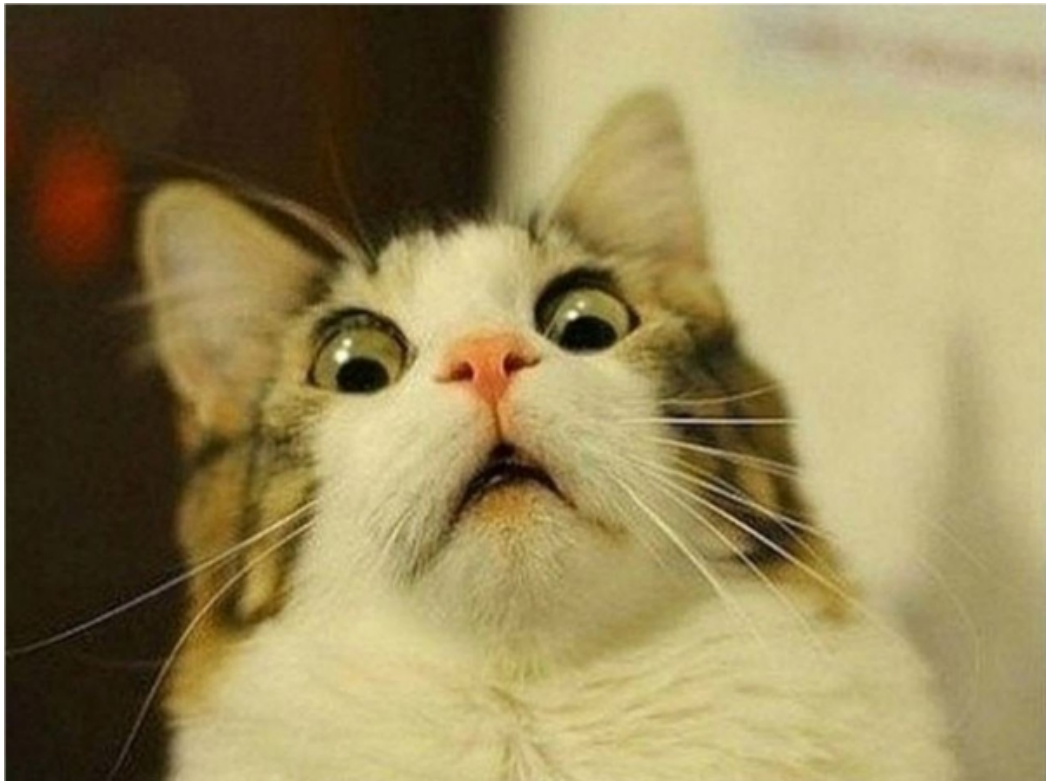
# Producing messages in order?



# Produce messages in order?

- If you are interested to keep order in your messages:
  - For reliability keep retries > 0 (make sure messages are delivered in case or problems)
  - Ensure max.in.flight.requests.per.connection == 1 (only one request is in.flight per connection)
- Understand and play with your key to ensure data is send to the expected partition.

# Now your might be wondering?



This is a distributed system, have I missed any important part?

Yes, the consumer's ;-)

# Consuming messages in Apache Kafka

- Your system could have 1 or more consumers
  - The consumer group protocol will organize which consumer gets which partitions
- Consumers are responsible of committing consumed offset
  - A committed offsets is not going to be processed again
  - Committing messages at reading (after the poll) is different that committing them after processing.
  - *enable.auto.commit* works based on a timer.
- Consumers will only read committed data (high watermark level)

# Consuming messages in Apache Kafka

- When do you are committing offsets?
  - Understand pros and cons of *enable.auto.commit*
  - Commit offsets when messages are proceed
  - Handle retries, ie target system is offline. Embrace DLQ pattern, second consumer.
    - Becareful with keeping them in memory.
- Prepare your application to handle duplicates, embrace at least once
- Committing aggressive does not provide exactly once semantics
  - It ads as well high workload to Apache Kafka

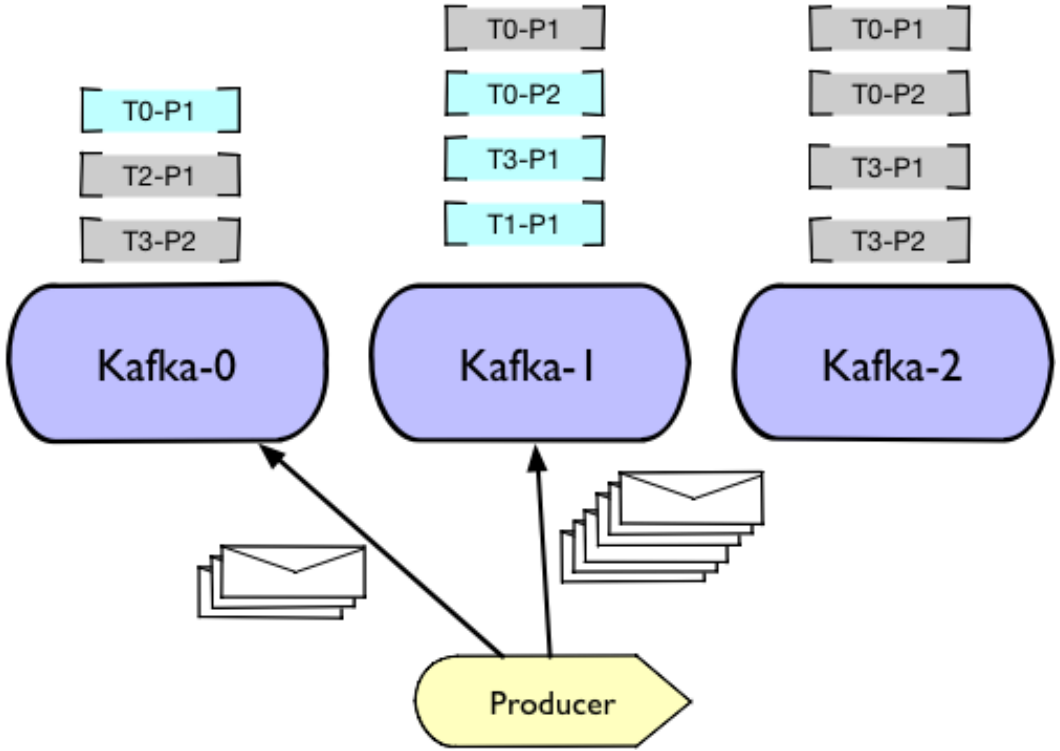
# Challenge 4
# Having a partition reassignment storm ?

# Is throughput low?

- Does this scenario rings a bell to you?
  - Your expected consumption throughput is degrading over time
  - Your production throughput as well is going down
  - You decide to create new partitions

But the problem seems to persist

# Is throughput low?



# Is throughput low?

- The natural reaction to this situation is to
  - Might be to add new broker
  - Reassign the partitions (*./bin/kafka-reassign-partitions*)
- However this scenario done wildly could
  - Overwhelm the broker network processors
  - If the network processors are crashing it, everything slows down
  - In old versions, this process could not be throttled

# Having a partition reassignment storm?

- **The Solution**:
  - Move an small number of partitions at time
  - Take advantage of replica throttling
  - Use tools like Confluent Rebalancer to automate this
- **The Moral** of this is:
  - Monitor your cluster using JMX!
  - Every time you change how your data is flowing, please test it in your staging environment

# Challenge 5
# Taking care of Zookeeper

# Taking special care of Zookeeper

- Zookeeper is used as a coordinator for decision and as an internal key value store for Apache Kafka. It’s performance is very important for the overall system.
- For example, if you lost the Kafka data in Zookeeper, the mapping of replicas to Brokers and topic configurations would be lost as well, making your Kafka cluster no longer functional and potentially resulting in total data loss.

# Taking special care of Zookeeper

- Does your Zookeeper have an odd number of nodes? 3 or 5 ?
  - Any election process needs an even 2n+1 nodes keep quorum in decision
  - With 2n+1 nodes, there could be n failed servers at any given time
- For production clusters, better have five zookeeper nodes in your ensemble

# Taking special care of Zookeeper

- Is Zookeeper running in dedicated hardware, this is the ideal.
- Does it has a dedicated disk for the transaction log?
  - While Apache Kafka does not benefit much of SSD (64Gb min), Zookeeper does a lot. Latency matters.
  - Use *autopurge.purgeInterval* and *autopurge.snapRetainCount* to ensure data cleanup.
- Not memory intensive usually 8Gb are enough.
- You should ensure Zookeeper is not competing for CPU. Latency again!

# Taking special care of Zookeeper


Zookeeper is your grandmother, you put it by the fireside, you pamper it, and you put SSD

https://twitter.com/framiere/status/1037614270299680769

# Challenge 6
# Monitoring

# Monitoring

- There seems to be a unanimous agreement in the community
- Running a distributed system is easy
- There is no need to observe how the system is doing!

Sarcasm Alert!

# Monitoring


WE KNOW WHAT
WE ARE DOING
WE DON'T
NEED MONITORING
imgflip.com

# Monitoring

- The reality is without observability your eyes into the system are blind
- A distributed system is form of many parts that need to work together, few things could go wrong that will disturb the overall system
- Apache Kafka is a very chatty system in terms of monitoring (over JMX)

Serious alert!!

# Monitoring

- Detailed list of metrics:
  http://kafka.apache.org/documentation.html#monitoring
- Set up alerts in different thresholds to help you react to the situations

# Monitor your system

- Don’t do only Apache Kafka, your system is important.
  - CPU, DISK, IO, Network, file handlers etc
- Set alerts for:
  - 60%: You must act upon it, but you will have time to react.
  - 80%: Run, you better fix the situation now!.

# Monitor your Apache Kafka

- Lots of interesting metrics such as:

| | |
|---|---|
| kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec | Number of incoming messages per second. Useful for understanding broker load |
| kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce/FetchConsumer/FetchFollower} | Number of requests per second. Useful for understanding broker load. |
| kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions | Should always be 0 |

# Monitor your Apache Kafka

- Or:

| | |
|---|---|
| *kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs* | Rate and time of leader election |
| *kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent* | The average fraction of time the I/O threads are idle. |
| *kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent* | The average fraction of time the network threads are idle. |
| *kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce or Fetch}* | Time in milliseconds spent on message format conversions. |

# Monitoring

- Pull this metrics into a central solution that will allow you get an overall cluster health view and manage your alerts
- Prometheus, jmx_reporter and Graphana are an excellent open source solution
- Jolokia, MetricBeat and Elasticsearch is another common solution
- See for more details:
  - https://github.com/purbon/monitoring-kafka-with-prometheus
  - https://www.elastic.co/blog/monitoring-java-applications-with-metricbeat-and-jolokia

# Challenge 6
# Security

# Kafka Security

- If you are willing to screw things up in your Apache Kafka setup, not having security and quotas in place is certainly a useful approach.
- Apache Kafka has support for:
  - Encryption and Authentication over SSL
  - Authentication with SASL
  - Authorization with ACL’s
  - Quotas and Throttle (for produce and fetch request)
- Kafka uses the JAAS mechanism to configure security

# Kafka Security overview

- Very useful for multi tenant deployments
- But not only for this, as well recommended for smaller deployments where accountability and control is encourage
- You can use as well SSL to communicate between brokers
- Clients can access the cluster using multiple protocols
    - PLAINTEXT within the secure area, SSL for outside clients

# Authentication with SASL

- SASL mechanism supported are:
  - Kerberos (I know you are brave!)
  - OAuthBearer: Unless you know what you are doing, better not use it in production
  - Scram (credentials are stored in Zookeeper, secure Zookeeper!)
  - Plain (user password over TLS)
- You can have more than one mechanism at the same time
- There is even LDAP integration

https://docs.confluent.io/current/kafka/authentication_sasl/authentication_sasl_oauth.html#production-use-of-sasl-oauthbearer

# Kafka niceties: ACL’s, Quotas and Throttle

- Not everyone should be able to access your Apache Kafka cluster, use ACL’s!
- Operations under ACL’s:
  - AlterConfig, CreateTopics, DeleteTopics, ….
  - Fetch, LeaderAndIsr, OffsetForLeaderEpoch,…
  - Metadata, OffsetFetch, FindCoordinator,…
- Leave enough “food” for all your dinner guest
  - Use quotas, basically byte-rate thresholds per client.id (producers or consumers)
  - Moving data from cluster to cluster, use throttle
  - Your cluster will appreciate!

https://docs.confluent.io/current/kafka/authorization.html#acl-format

# Success with Apache Kafka will require

- Understanding data durability
- Getting comfortable with the replication mechanism
- How to handle message ordering
- Load balancing your data access
- Taking care of Zookeeper
- Monitoring and Security

# If all of this sounds terrible

# Consider using a cloud service!

# Can do that with your eyes closed?

# We’re Hiring! Talk to me!

# Thanks!
# Questions?

Pere Urbon-Bayes

@purbon

Technology Architect

Confluent