

<https://newrelic.com>[WHY NEW RELIC](#)[PRODUCTS & PRICING](#)[SOLUTIONS](#)[SERVICES & SUPPORT](#)[SEARCH \(/SEARCH\)](#)<https://rpm.newrelic.com>[Sign Up \(https://newrelic.com/signup\)](https://newrelic.com/signup)[Blog Home \(https://blog.newrelic.com/\)](https://blog.newrelic.com/)[News and Products \(https://blog.newrelic.com/category/product-news/\)](https://blog.newrelic.com/category/product-news/)[Software Engineering \(https://blog.newrelic.com/category/engineering/\)](https://blog.newrelic.com/category/engineering/)[Cloud \(https://blog.newrelic.com/tag/cloud/\)](https://blog.newrelic.com/tag/cloud/)[A Leader in Gartner's 2019 APM Magic Quadrant > \(https://newrelic.com/gartner-magic-quadrant-19\)](https://newrelic.com/gartner-magic-quadrant-19)

# Effective Strategies for Kafka Topic Partitioning

By Amy Boyle (<https://blog.newrelic.com/author/amyboyle/>) • Mar. 13th, 2018 • [Software Engineering \(https://blog.newrelic.com/category/engineering/\)](https://blog.newrelic.com/category/engineering/)🔗 [Apache Kafka \(https://blog.newrelic.com/tag/apache-kafka/\)](https://blog.newrelic.com/tag/apache-kafka/), [event data \(https://blog.newrelic.com/tag/event-data/\)](https://blog.newrelic.com/tag/event-data/), [streaming \(https://blog.newrelic.com/tag/streaming/\)](https://blog.newrelic.com/tag/streaming/)[G+](#)[f](#)[in](#)

*Don't miss part one in this series: [Using Apache Kafka for Real-Time Event Processing at New Relic \(https://blog.newrelic.com/engineering/apache-kafka-event-processing/\)](https://blog.newrelic.com/engineering/apache-kafka-event-processing/).*

If you're a recent adopter of Apache Kafka (<https://kafka.apache.org/>), you're undoubtedly trying to determine how to handle all the data streaming through your system. The Events Pipeline team at New Relic processes a huge amount of "event data (<https://docs.newrelic.com/docs/using-new-relic/metrics/analyze-your-metrics/data-collection-metric-timeslice-event-data#event-data>)" on an hourly basis, so we've thought about this question a lot. Unless you're processing only a small amount of data, you need to distribute your data onto separate partitions.

In part one of this series—[Using Apache Kafka for Real-Time Event Processing at New Relic \(https://blog.newrelic.com/engineering/apache-kafka-event-processing/\)](https://blog.newrelic.com/engineering/apache-kafka-event-processing/)—we explained how we built the underlying architecture of our event processing streams using Kafka. In this post, we explain how the partitioning strategy for your producers depends on what your consumers will do with the data.

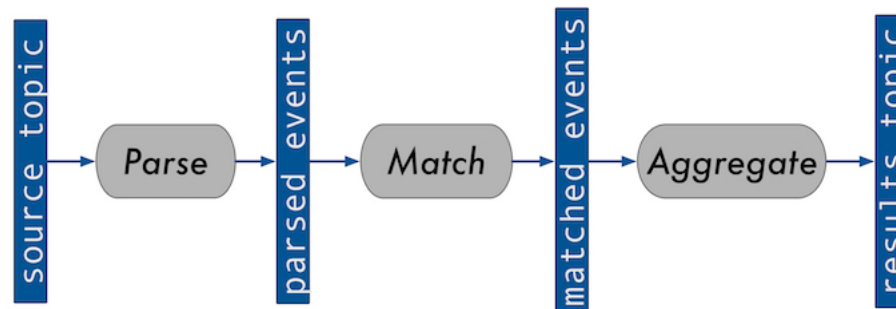
## Why partition your data in Kafka?

If you have enough load that you need more than a single instance of your application, you need to partition your data. The producer clients decide which topic partition data ends up in, but it's what the consumer applications will do with that data that drives the decision logic. If possible, the best partitioning strategy to use is random.

However, you may need to partition on an attribute of the data if

- The consumers of the topic need to aggregate by some attribute of the data
- The consumers need some sort of ordering guarantee
- Another resource is a bottleneck and you need to shard data
- You want to concentrate data for efficiency of storage and/or indexing

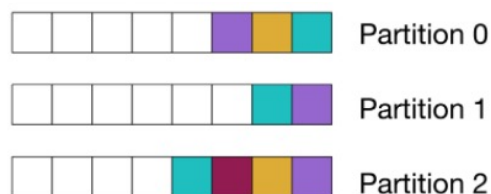
In part one (<https://blog.newrelic.com/engineering/apache-kafka-event-processing/>), we used the diagram below to illustrate a simplification of a system we run for processing ongoing queries on event data:



## Random partitioning of Kafka data

We use this system on the input topic for our most CPU-intensive application—the match service. This means that all instances of the match service must know about all registered queries to be able to match *any* event. While the event volume is large, the number of registered queries is *relatively* small, and thus a single application instance can handle holding all of them in memory, for now at least.

The following diagram uses colored squares to represent events that match to the same query. It shows messages randomly allocated to partitions:

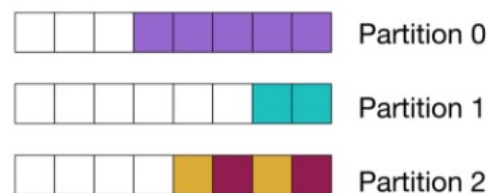


Random partitioning results in the most even spread of load for consumers, and thus makes scaling the consumers easier.

## Partition by aggregate

On the topic consumed by the service that does the query aggregation, however, we must partition according to the query identifier since we need all of the events that we're aggregating to end up at the same place.

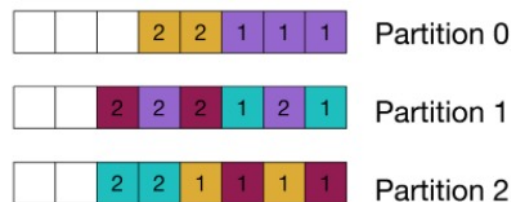
This diagram shows that events matching to the same query are all co-located on the same partition. The colors represent which query each event matches to:



After releasing the original version of the service, we discovered that the top 1.5% queries accounted for approximately 90% of the events processed for aggregation. As you can imagine, this resulted in some pretty bad hot spots on the unlucky partitions.

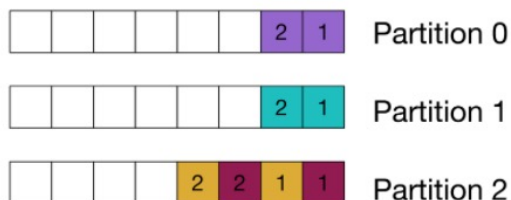
To mitigate the hot spots, we needed a more sophisticated partitioning strategy, so we also partitioned by time window to move the hot spots around. We hashed together the query identifier with the time window begin time. This spread the “hot” queries across the partitions in chunks.

In the diagram below, the numbers indicate what time window each message belongs to:



## Ordering guarantee

We partition our final results by the query identifier, as the clients that consume from the results topic expect the windows to be provided in order:



## Planning for resource bottlenecks and storage efficiency

When choosing a partition strategy, it's important to plan for resource bottlenecks and storage efficiency.

*(Note that the examples in this section reference other services that are not a part of the streaming query system I've been discussing.)*

**Resource bottleneck:** We have another service that has a dependency on some databases that have been split into shards. We partition its topic according to the how the shards are split in the databases. This approach produces a result similar to the diagram in our partition by aggregate example. Each consumer will be dependent only on the database shard it is linked with. Thus, issues with other database shards will not affect the instance or its ability to keep consuming from its partition. Also, if the application needs to keep state in memory related to the database, it will be a smaller share. Of course, this method of partitioning data is also prone to hotspots.

**Storage efficiency:** The source topic in our query processing system shares a topic with the system that permanently stores the event data. It reads in all the same data using a separate consumer group. The data on this topic is partitioned by which customer account the data belongs to. For efficiency of storage and access, we concentrate an account's data into as few nodes as possible. While many accounts are small enough to fit on a single node, some accounts must be spread across multiple nodes. If an account becomes too large, we have custom logic to spread it across nodes, and, when needed, we can shrink the node count back down.

## Rebalance or statically assign partitions?

By default, whenever a consumer enters or leaves a consumer group, the brokers rebalance the partitions across consumers ([https://kafka.apache.org/documentation/#impl\\_consumerrebalance](https://kafka.apache.org/documentation/#impl_consumerrebalance)), meaning Kafka handles load balancing with respect to the number of partitions per application instance for you. This is great—it's a major feature of Kafka. We use this default on nearly all our services.

When a rebalance happens, all consumers drop their partitions and are reassigned new ones. If you have an application that has state associated with the consumed data, such as our aggregator service, you need to drop that state and start fresh with data from the new partition.

However, if dropping state isn't an option, an alternative is to not use a consumer group and instead use the Kafka API (<https://kafka.apache.org/10/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html>) to statically assign partitions, which does not trigger rebalances. Of course, in that case, you must balance the partitions yourself and also make sure that all partitions are consumed.

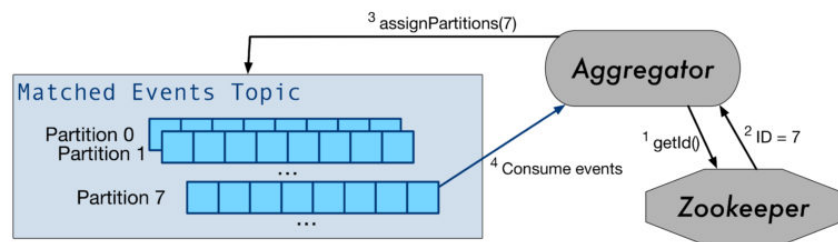
Since New Relic deals with high-availability real-time systems, we cannot tolerate any downtime for deploys, so we do rolling deploys.

For our aggregator service, which collects events into aggregates over the course of several minutes, we use statically assigned partitions to avoid unnecessarily dropping this state when other application instances restart. The aggregator builds up state that it must drop at every rebalance/restart/deploy. It has to backtrack and rebuild the state it had from the last recorded publish or snapshot. Before we used statically assigned partitions, we had to wait for every application instance to recover before they could restart.

If you use static partitions, then you must manage the consumer partition assignment in your application manually. Here is how we do this in our aggregator service:

We set a configuration value for the number of partitions each application instance should attempt to grab. When each instance starts up, it gets assigned an ID through our Apache ZooKeeper (<https://zookeeper.apache.org/>) cluster, and it calculates which partition numbers to assign itself. The instance holds onto those partitions for its lifetime. We always keep a couple extra idle instances running—waiting to pick up partitions in the event that another instance goes down (either due to failure or because of a normal restart/deploy). All of our instances run in containers, and we orchestrate them with Marathon (<https://mesosphere.github.io/marathon/>) to always keep a minimum number of instances running.

The diagram below shows the process of a partition being assigned to an aggregator instance. In this example, we have configured 1 partition per instance:



([http://blog.newrelic.com/wp-content/uploads/kafka\\_static\\_partitions.jpg](http://blog.newrelic.com/wp-content/uploads/kafka_static_partitions.jpg))

## Conclusion

Your partitioning strategies will depend on the shape of your data and what type of processing your applications do. As you scale, you may need to adapt your strategies to handle new volume and shape of data. Consider what the resource bottlenecks are in your architecture, and spread load accordingly across your data pipelines. It may be CPU, database traffic, or disk space, but the principle is the same. Be efficient with your most limited/expensive resources.

🔗 Apache Kafka (<https://blog.newrelic.com/tag/apache-kafka/>), event data (<https://blog.newrelic.com/tag/event-data/>), streaming (<https://blog.newrelic.com/tag/streaming/>)



Amy Boyle is a senior software engineer at New Relic, working on the core data platform. Her interests include distributed systems, readable code, and puppies. View posts by Amy Boyle (<https://blog.newrelic.com/author/amyboyle/>).

*The views expressed on this blog are those of the author and do not necessarily reflect the views of New Relic. This blog may contain links to content on third-party sites. By providing such links, New Relic does not adopt, guarantee, approve or endorse the information, views or products available on such sites.*

Interested in writing for New Relic Blog? Send us a pitch ([mailto:blog@newrelic.com?subject=Guest post submission request&body=Please provide a description of what you'd like to write about for New Relic's blog:](mailto:blog@newrelic.com?subject=Guest post submission request&body=Please provide a description of what you'd like to write about for New Relic's blog:)))!

## RELATED POSTS

### THE PLATFORM

Digital Intelligence Platform (<https://newrelic.com/products>) New Relic APM (<https://newrelic.com/application-monitoring>) New Relic for AWS (<https://newrelic.com/partner/aws-monitoring>)

New Relic Synthetics (<https://newrelic.com/synthetics>) New Relic Infrastructure (<https://newrelic.com/infrastructure>) New Relic Mobile (<https://newrelic.com/mobile-monitoring>) New Relic Insights (<https://newrelic.com/insights>)

### ADDITIONAL FEATURES

New Relic Alerts (<https://newrelic.com/alerts>) New Relic Plugins (<https://newrelic.com/plugins>) New Relic for iOS and Android (<https://newrelic.com/mobility>) Enterprise Security (<https://newrelic.com/security>)

### WHY NEW RELIC

Overview (<https://newrelic.com/why-new-relic>) Our Customers (<https://newrelic.com/why-new-relic/customers>) Your Success Plan (<https://newrelic.com/why-new-relic/customer-success>)

### SUPPORT

Get Support (<https://support.newrelic.com/>) New Relic Status (<https://status.newrelic.com>) Documentation (<https://docs.newrelic.com>) Community Forum (<https://discuss.newrelic.com>)

New Relic University (<https://learn.newrelic.com>) Deploying New Relic (<https://newrelic.com/start>)

### THE COMPANY

About Us (<https://newrelic.com/about>) Careers (<https://newrelic.com/about/culture>) Partner Program (<https://newrelic.com/about/partners>) Investor Relations (<https://ir.newrelic.com/>)

Leadership (<https://newrelic.com/about/leadership>) New Relic Community (<https://newrelic.com/community>) New Relic Nonprofit Program (<https://newrelic.com/nonprofit>) Selfie Stories (<https://newrelic.com/selfies>)

Contact Us (<https://newrelic.com/about/contact-us>)

### UPDATES

Blog (<https://blog.newrelic.com>) Newsroom (<https://newrelic.com/about/newsroom>)

### TOPICS

DevOps (<https://newrelic.com/devops>) Software Analytics (<https://newrelic.com/software-analytics>) Mobile (<https://newrelic.com/mobile-app-development>)

### SOLUTIONS

Digital Customer Experience (<https://newrelic.com/solutions/digital-customer-experience>) Application Development (<https://newrelic.com/solutions/application-development>)

Production Monitoring (<https://newrelic.com/solutions/production-monitoring>) Real-Time Analytics (<https://newrelic.com/solutions/real-time-analytics>)

Mobile Application Management (<https://newrelic.com/solutions/mobile-application-management>) Digital Transformation (<https://newrelic.com/solutions/digital-transformation>)

Cloud Migration (<https://newrelic.com/solutions/cloud-migration>)

### RESOURCES



Case Studies (<https://newrelic.com/resources/case-studies>) Videos (<https://newrelic.com/resources/videos>) White Papers (<https://newrelic.com/resources/white-papers>) eBooks (<https://newrelic.com/resources/ebooks>)

Analyst Reports (<https://newrelic.com/resources/analyst-reports>) Infographics (<https://newrelic.com/resources/infographics>) Tutorials (<https://newrelic.com/resources/tutorials>)

Webinars (<https://newrelic.com/resources/webinars>) Latest Resources (<https://newrelic.com/resources>)

#### INTERNATIONAL

[newrelic.de](https://www.newrelic.de) (German) (<https://www.newrelic.de>) [br.newrelic.com](https://br.newrelic.com) (Portuguese) (<https://br.newrelic.com>)

 (<http://www.facebook.com/NewRelic>)  (<http://www.twitter.com/NewRelic>)

 (<https://www.linkedin.com/company/new-relic-inc->)

 (<https://www.youtube.com/user/NewRelicInc/featured>)

 (<https://www.instagram.com/newrelic/>)



(<https://plus.google.com/b/108825274074563270301/10882527407456327030>)

 (<http://feeds2.feedburner.com/NewRelic>)



©2008-17 New Relic, Inc. All rights reserved

[Terms of Service \(https://newrelic.com/terms\)](https://newrelic.com/terms) [DMCA Policy \(https://newrelic.com/dmca\)](https://newrelic.com/dmca) [Privacy Policy \(https://newrelic.com/privacy\)](https://newrelic.com/privacy) [Cookie Policy \(https://newrelic.com/cookie-policy\)](https://newrelic.com/cookie-policy)

[UK Slavery Act of 2015 \(https://newrelic.com/uk-slavery-act\)](https://newrelic.com/uk-slavery-act)