Ten Rules For Managing Kafka

Overview

Much like its namesake, Apache Kafka® can be an inscrutable beast for the uninitiated. If you dive in and just try to "wing it," you are likely to make mistakes. Kafka is not difficult to use but it is tricky to optimize. With time, Kafka has evolved from essentially a message queue to a versatile streaming platform.

In this white paper we cover ten rules that will help you perfect your Kafka system to get ahead.

instaclustr

1. Logs

Kafka has a lot of log configurations. The defaults are generally sane but most users will have at least a few things they will need to tweak for their particular use case. You need to think about retention policy, cleanups, compaction, and compression.

The three parameters to worry about are:

- log.segment.bytes,
- log.segment.ms, and
- log.cleanup.policy (or the equivalent settings at the topic level).

If you don't need to worry about preserving logs then you can set cleanup.policy to 'delete' and let Kafka remove log files after a set time or after they reach a certain size. If you do need to keep logs around for a while, use the 'compact' policy. The defaults are generally sane but if your use case is unique you may need to adjust the frequency of compactions. Remember that log cleanup isn't free—it eats CPU and RAM every time it runs. If you are relying on Kafka to serve as a kind semi-long-term commit log, be sure your compactions run frequently enough but do not harm performance.

2. Hardware Requirements

When tech teams first start playing with Kafka there is a tendency to just sort of 'ballpark' the hardware—just spin up a big server and hope it works. In actuality Kafka does not necessarily need a ton of resources. It is designed for horizontal scaling thus you can get away with using relatively cheap commodity hardware.

Here some basic points to remember:

CPU: Doesn't need to be very powerful unless you're using SSL and compressing logs. The more cores, the better for parallelization. If you do need compression we recommend using LZ4 codec for best performance in most cases.

Memory: Kafka works best when it has at least 6 GB of memory for heap space. The rest will go to OS page cache which is key for client throughput. Kafka can run with less RAM, but don't expect it to handle much load. For heavy production use cases go for at least 32 GB.

Disk: Because of Kafka's sequential disk I/O paradigm SSD's will not offer much benefit. Do not use NAS. Multiple drives in a RAID setup can work well.

Network and Filesystem: Use XFS and keep your cluster in a single data center if possible. The higher the network bandwidth the better.

3. ZooKeeper

We could do an entire white paper just on ZooKeeper. It is a versatile piece of software that works great for both service discovery and a range of distributed configuration use cases. For brevity's sake, we'll offer just three key points to keep in mind when setting it up for Kafka.

Avoid co-locating ZooKeeper in any major production environment

This is a shortcut many companies take thanks to the spread of Docker. They figure they can run Kafka and ZooKeeper as separate Docker containers on one instance by carefully locking in memory and CPU limits. This is actually fine for a development environments or even smaller production deployments assuming you take the right precautions. The risk with larger systems is that you lose more of your infrastructure if a single server goes down. It's also suboptimal for your security setup because Kafka and ZooKeeper are likely going to have a very different set of clients and you will not be able to isolate them as well.

Do not use more than five ZooKeeper nodes without a really great reason

For a dev environment, one node is fine. For your staging environment you should use the same number of nodes as production. In general three ZooKeeper nodes will suffice for a typical Kafka cluster. If you have a very large Kafka deployment, it may be worth going to five ZooKeeper nodes to improve latency, but be aware this will put more strain on the nodes. ZooKeeper tends to be bound by CPU and network throughput. It is rarely a good idea to go to seven or more nodes as you will end up with a huge amount of load from all seven nodes trying to stay in sync and handle Kafka requests (not as much of an issue in the later Kafka versions that rely less on ZooKeeper).

Tune for minimal latency

Use servers with really good network bandwidth. Use appropriate disks and keep logs on a separate disk. Isolate the ZooKeeper process and ensure that swap is disabled. Be sure to track latency in your instrumentation dashboards.

4. Replication and Redundancy

There are a few dimensions to consider when thinking about redundancy with Kafka. The first and most obvious is just the replication factor. We believe Kafka defaults at 2 but for most production uses 3 is best. It will allow you to lose a broker and not freak out. If, improbably, a second broker also independently fails, your system is still running. Alongside replication factor you also have to think about datacenter racks zones. In AWS for example, you would not want to have your Kafka servers in different regions, but putting them in different availability zones is a good idea for sake of redundancy. Single AZ failures have happened often enough in Amazon.

5. Topic Config

Your Kafka cluster's performance will depend greatly on how you configure your topics. In general you want to treat topic configuration as immutable since making changes to things like partition count or replication factor can cause a lot of pain. If you find that you need to make a major change to a topic, often the best solution is to just create a new one. Always test new topics in a staging environment first.

As mentioned above, start at 3 for replication factor. If you need to handle large messages, see if you can either break them up into ordered pieces (easy to do with partition keys) or just send pointers to the actual data (links to S3 for example). If you absolutely have to handle larger messages be sure to enable compression on the producer's side. The default log segment size of 1 GB should be fine (if you are sending messages larger than 1 GB, reconsider your use case). Partition count, possibly the most important setting, is addressed in the next section.

6. Parallelization

Kafka is built for parallel processing. Partition count is set at the topic level. The more partitions, the more throughput you can get through greater parallelization. The downside is that it will lead to more replication latency, more painful rebalances, and more open files on your servers. Keep these tradeoffs in mind. The most accurate way to determine optimal partition settings is to actually calculate desired throughput against your hardware. Assume a single partition on a single topic can handle ~10 MB/s (producers can actually produce faster than this but it's a safe baseline) and then figure out what your desired total throughput is for your system.

If you want to dive in and start testing faster, a good rule of thumb is to start with 1 partition

per broker per topic. If that works smoothly and you want more throughput, double that number, but try to keep the total number of partitions for a single topic on a single broker below 10. So for example, if you have 24 partitions and three brokers, each broker will be responsible for 8 partitions, which is generally fine. If you have dozens of topics an individual broker could easily end up handling hundreds of partitions. If your cluster's total number of partitions is north of 10,000 then be sure you have really good monitoring because rebalances and outages could get really thorny.

7. Security

There are two fronts in the war to secure a Kafka deployment:

- Kafka's internal configuration, and
- The infrastructure on which Kafka is running.

Starting with the latter, the first goal is isolating Kafka and ZooKeeper. ZooKeeper should never be exposed to the public internet (except for unusual use cases). If you are only using ZooKeeper for Kafka, then only Kafka should be able to talk to it. Restrict your firewalls/security groups accordingly. Kafka should be isolated similarly. Ideally there is some middleware or load balancing layer between any clients connecting from the public internet and Kafka itself. Your brokers should reside within a single private network and by default reject all connections from outside.

As for Kafka's configuration, the .9 release added a number of useful features. Kafka now supports authentication between itself and clients as well as between itself and ZooKeeper. Kafka also now supports TLS, which we recommend using if you have clients connecting directly from the public internet. Be advised that using TLS will impact throughput performance. If you can't spare the CPU cycles then you will need to find some other way to isolate and secure traffic hitting your Kafka brokers.

8. Open File Config

Ulimit configuration is one of those things that can sneak up on you with a lot of different programs. Devops engineers have all been there before. A Pagerduty alert fires late at night. Seems at first like a load issue but then you notice one or more of your brokers is just totally down. You dig through some logs and get one of these: "java.io.IOException: Too many open files."

It's an easy fix. Edit /etc/sysctl.conf with a larger value for max open files and restart. Save yourself an outage and ensure that your deployment system (Chef, CloudFormation, etc.) is setting a hard Ulimit of at least 128,000.

9. Network Latency

This one is pretty simple. Certainly low latency is going to be your goal with Kafka. Ideally you have your brokers geographically located near their clients. If your producers and consumers are located in the United States, best not to have your Kafka brokers in Europe. Also be aware of network performance when choosing instance types with cloud providers. It may be worthwhile to go for the bigger servers with AWS that have greater bandwidth if that becomes your bottleneck.

10. Monitoring (to catch all of the above)

All of the above issues can be anticipated at the time of cluster creation. However conditions change, and without a proper monitoring and alerting strategy, you can get bit by one of these problems down the road. With Kafka you want to prioritize two basic types of monitoring: system metrics and JVM stats. For the former you need to ensure you track open file handles, network throughput, load, memory, and disk usage at a minimum. For the latter, be mindful of things like GC pauses and heap usage. Ideally you will keep a good amount of history and set up dashboards for quickly debugging issues.

For alerting, you will want to configure your system (Nagios, PagerDuty, etc.) to warn you about system issues like low disk space or latency spikes. Better to get an annoying alert about reaching 90% of your open file limit than getting an alert that your whole system has crashed.

Conclusion

Kafka is a powerful piece of software that can solve a lot of problems. Like most libraries and frameworks, you get out of it what you put into it. If you have a solid infrastructure and dev team that can devote sufficient time, you can do amazing things with Kafka. Lacking that, Kafka can be risky. Fortunately there is a growing ecosystem of managed offerings. With the right provider you can get all of the performance and scaling benefits of Kafka without having to go it alone. To that end, check out Instaclustr's newest offerings in the Kafka space.

Ready to Experience Instaclustr Managed Apache Kafka®?

Reach out to our Sales team today.

About Instaclustr

Instaclustr delivers reliability at scale through our integrated data platform of open source technologies such as **Apache Cassandra®**, **Apache Kafka®**, **Apache Spark™**, **Elasticsearch™**, **Redis™**, and **PostgreSQL®**.

Our expertise stems from delivering more than 100 million node hours under management, allowing us to run the world's most powerful data technologies effortlessly.

We provide a range of managed, consulting, and support services to help our customers develop and deploy solutions around open source technologies. Our integrated data platform, built on open source technologies, powers mission critical, highly available applications for our customers and help them achieve scalability, reliability, and performance for their applications.