NetApp Instaclustr

White Paper

10 RULES FOR MANAGING *Apache Kafka®*

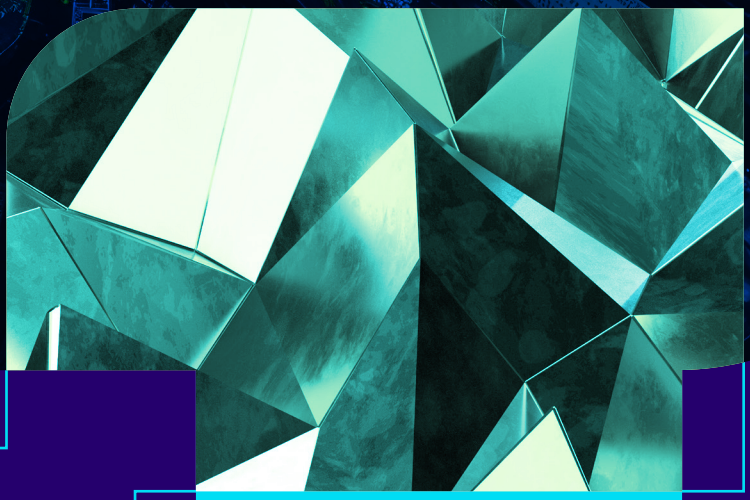

Table of contents

OVERVIEW

For real time streaming and queuing technology, Apache Kafka® is truly unrivaled, but it can be an inscrutable beast for newcomers. Without proper guidance, it's easy to miss out on Kafka's full capabilities. While not the easiest technology to optimize, Kafka rewards those willing to explore its depths. Under the hood, it is an elegant system for stream processing, event sourcing, and data integration.

In this white paper, we cover the 10 critical rules that will help you optimize your Kafka system and unlock its full potential.

1. *Logs*

Kafka comes equipped with a multitude of log configurations and additional tools to complement Kafka's capabilities. The defaults are generally suitable for a broad selection of use-cases but most users will have at least a few things they will need to tweak for their particular use case. It's important to consider elements like retention policy, cleanups, compaction, and compression.

The 3 parameters to focus on are:

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

For non-critical data, 'delete' cleanup policies let Kafka remove logs after a time or size threshold.

However, for long-term retention, cloud object storage like Amazon S3 combined with analytics tools like OpenSearch® are now commonly used alongside Kafka's native storage.
The 'compact' policy can reduce on-disk footprint for long-term Kafka log retention if object storage isn't used.

Adjust log cleanup frequencies carefully, keeping in mind the impact on CPU and RAM during cleanup. If relying on Kafka as a commit log, ensure compactions occur often enough without jeopardizing performance.

2. Hardware requirements

When first exploring Kafka, tech teams often take a rough estimate approach to hardware sizing—spinning up a large server and hoping it can handle the workload. In reality, Kafka doesn't necessarily require a significant amount of resources. While Kafka can run on commodity hardware, managed services like Instaclustr for Apache Kafka can help simplify infrastructure management.

If you are taking the 'do it yourself' approach, here are 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.

Memory: Kafka works best when it has at least 6 GB of memory for heap space. The rest will go to the 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 larger RAM would be needed.

Disk: Similar to the point above about RAM, storage capacity needed is use-case dependent. The storage you need would be directly proportional to the message size, the throughput and the retention period for your Kafka cluster.

3. Apache ZooKeeper™ and KRaft

Earlier versions of Apache Kafka relied on Apache ZooKeeper for managing meta-data on broker information, topic configurations, and partitions. ZooKeeper provides distributed coordination and consensus for Kafka but introduces an external dependency.

Recent Kafka versions starting from 3.3 include a new consensus protocol called Kafka Raft (KRaft) that allows Kafka to manage cluster meta-data internally without ZooKeeper. KRaft provides a distributed "quorum controller" natively within Kafka using the Raft consensus algorithm. Meta-data is stored in an internal Raft log and periodically snapshotted.

Compared to ZooKeeper, KRaft improves scalability, recovery time, monitoring, and security by eliminating the external dependency. Now that KRaft has reached production-ready status in Kafka 3.3, from version 4.0 onwards the Kafka project will fully drop the ZooKeeper dependency and make KRaft the default meta-data management method.

By consolidating Kafka's meta-data management internally, KRaft delivers simplified operations, better resilience, and removes the need to run a separate ZooKeeper cluster. Teams currently relying on older Kafka versions with ZooKeeper can realize significant benefits by migrating to KRaft when feasible.

4. *Replication and redundancy*

There are a few key dimensions to consider when thinking about redundancy with Kafka. The most basic is the replication factor—a setting of 3 is recommended for most production uses to allow for broker failures. Alongside the replication factor, you also have to think about availability zones. You would not want to have your Kafka brokers in different regions but putting them in different availability zones is a good idea for the sake of redundancy. Single AZ failures have happened often enough in AWS.

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. Partition count can be increased, but not decreased. 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 earlier, 3 is a common production replication factor. If you need to handle large messages, see if you can either break them up into ordered pieces (with partition keys) or just send pointers to the actual data (for example, links to S3). If you need 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 calculate desired throughput against your hardware. Assume a single partition on a single topic can handle ~10 MB/s (producers can produce faster than this but it's a safe baseline) and then figure out what the 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.

For example, if you have 24 partitions and 3 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 good monitoring because rebalances and outages could get thorny.

7. Security

Securing Kafka deployments is critical on 2 fronts: infrastructure and configuration.

Starting with the former, the first goal is isolating Kafka and ZooKeeper. ZooKeeper should never be exposed to the public internet (except in 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.

Kafka 3.0 and later editions simplify security by removing the ZooKeeper dependency.

Kafka has evolved its native capabilities around encryption, authentication, and authorization, though enabling security capabilities is optional. Make sure to enable SSL/ TLS for encryption in transit across all clients and authentication to verify client identities. 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. Managed services can also help with offloading security management.

8. *Open file config*

Kafka brokers require substantial open file handles for network connections and log segments. In older versions, careful manual tuning of Ulimits was needed to prevent brokers from crashing due to too many open files. However, recent Kafka releases can now dynamically scale file handles based on actual server loads and usage patterns. Brokers automatically increase Ulimits if configured to do so. This self-tuning reduces the need for massive precautionary Ulimit settings.

Of course, continued monitoring of open file usage is advised to catch any constraints. But the days of pre-emptively setting Ulimits to high values like 128K+ just to be safe are largely behind us given Kafka's improved self-scaling capabilities. Teams can now rely on smarter default policies and dynamic tuning rather than intensive manual limits and restarts.

9. *Network latency*

This one is pretty simple: low latency is going to be your goal with Kafka. Ideally, you have your brokers geographically located near their clients. For example, if your producers and consumers are located in the United States, it's best not to have your Kafka brokers in Europe. Leverage Kafka rack awareness features to optimize replication traffic across data centers.

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 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 bitten by one of these problems down the road. With Kafka, you want to prioritize 2 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. It is better to get an annoying alert about reaching 90% of your open file limit than get an alert that your whole system has crashed.

CONCLUSION

Apache Kafka is a powerful distributed streaming platform, but like most enterprise infrastructure, it requires extensive expertise to maximize results. While Kafka enables impressive scalability and resilience when implemented properly, it carries risks if infrastructure and operations fall short.

Fortunately, maturity within the Kafka ecosystem provides options beyond just open source Kafka. Managed services from vendors like Instaclustr allow organizations to realize Kafka's benefits without managing operations themselves.

Ready to Experience Instaclustr Managed Apache Kafka®?

Reach out to our **Sales team** today.

NetApp® Instaclustr specializes in open source technologies for enterprises. Our managed platform streamlines data infrastructure management, backed by experts who ensure ongoing performance, scalability, and optimization. This enables companies to focus on building cutting edge applications at lower costs.

1-16apr25