Things nobody will tell you setting up a Kafka cluster

medium.com/@ankurrana/things-nobody-will-tell-you-setting-up-a-kafka-cluster-3a7a7fd1c92d

Setting up a Kafka cluster for a production environment is no trivial task. There are so many configurations, settings, and decisions to be taken that it's quite difficult to understand all of them up front. Even though after going through entire confluent documentation and reading everything available about the best practices, you are bound to make mistakes and spend days, if not weeks pondering over what is going wrong and why is the cluster so unstable.

Documentation provided by confluent is indeed very good and contains everything you'll ever need to set up a cluster. But sometimes you do miss a few things and end up learning them the hard way.

In this article, I am going to discuss some of the key points to keep in mind when setting up a Kafka cluster. Few things which we missed and had a hard time fixing up and some other things that I feel anybody can miss when setting up a cluster for the first time.

I am assuming that you already know the basics of Kafka and generally understand how to set up a cluster with Zookeeper and all.

Points :

- *Set vm.swappiness to the minimum possible value for your server type for both Kafka brokers as well as the zookeeper servers*.
  Swappiness is basically Linux kernel's tendency to copy RAM contents to swap. For latency critical application such as Kafka, it's very important that unnecessary swapping of memory contents doesn't affect its performance. You can go here to know how to do it in Ubuntu.
- *Increase the file descriptors limits to a very high value as much as 100,000.*
  Kafka uses a very large number of open files when communicating with Kafka clients and the default limit per process is very low in most Linux distributions. Depending on the way you are starting your Kafka broker process, you may require different ways to

increase this limit. We suffered a Kafka broker failure as the limit set was not applied to our Kafka process. Check this thread to know why. It's best to make sure that the limit is indeed applied to your process using the steps below:
ps ax | grep <Name of the Kafka process>
cat /proc/{{process_id}}/limits | grep "Max open files"

- *Give at least 32 GB of RAM to your Kafka brokers.*
  Kafka brokers may not require much heap as it uses memory very efficiently (maximum 5 GB), but it very well leverages Linux kernel’s page cache to keep data in RAM, instead of flushing messages to disk immediately. So typically most of your RAM would be utilized by Kafka indirectly through the kernel’s Page Cache. To know how much RAM is currently used as page cache check out the buffer/cache column of the report produced by the *free -m -h* command in your terminal.
- *Use multiple data log directories mounted on multiple disks.*
  Kafka allows you to distribute data in multiple directories. Topic partitions are assigned to log directories in round-robin fashion during topic creation. Having multiple log directories increases parallelism and provides better throughput. Also, Increase the value of broker property ***num.recovery.threads.per.data.dir*** to at least the number of data directories for faster brokers shutdown and startup.
- *Do not share Kafka data drives with any other application or OS filesystem.*
  A typical Kafka cluster deployed would have Kafka brokers mounted with multiple throughput optimized st1 HDD drives. These disks are specially optimized for large IO requests (at least 1 Mb) and generally perform very bad with a large number of small IO requests. Adding logs or any kind of other OS filesystem will affect it’s performance adversely. It's better to use SSD for such operations.
- *Garbage Collection taking too much time may indicate other problems with the cluster. Keep it in check.*
  I personally struggled a lot with GC taking too much time on our Kafka brokers, sometimes as much as 20–25%. This terribly affected cluster’s performance as producers were failing to flush messages

and consumers were lagging drastically even when the load was relatively low. Since this lag increased with increase in GC, I wrongfully mapped these two and started experimenting with GC configuration without finding the root cause. I felt that maybe the default GC settings were not tuned for a production environment and kept trying different configs without much benefit. Although the complete story of struggle with this issue requires its own post, but in a gist, the real problem was that our Kafka broker was spending too much time in disk IO, and the maximum throughput we were getting was just 400 Kbps in a 40 Mbps st1 disk. Many IO requests were getting queued and thus hogging the application. All because I didn't understand the working of an st1 drive.

- *Choose the number of topic partitions carefully.*
  There are tradeoffs of choosing the number of partitions for your Kafka topics. Apart from the points discussed in the article by confluent, keep in mind that topics which could be joined in a Kafka Streams application must be co-partitioned, which means that they must have the same number of partitions to work correctly. More on this *here*.

We just walked through a very small set of key points that one must keep in mind when setting up a Kafka cluster. I am pretty sure that you'll face many other weird issues of your own which will make you scratch your head just like we did. Hopefully, this article will help you avoid some of those.