7 Effective Ways to Improve Your Elasticsearch Performance

March 9, 2021
by SentinelOne

Every business directly or indirectly depends on data. We live in a fast-paced world. And to keep up with its speed, we need data on the fly. Hence, handling and processing data efficiently has become very crucial. Elasticsearch is a great search engine that helps retrieve data in near-real time and also helps store data efficiently to do so. High performance of Elasticsearch is significant in order to meet business needs. Especially as data grows and complexity increases, you start seeing adverse effects if performance isn't high. And to help you with that, let's go through different ways of improving Elasticsearch performance, metrics to monitor, and tips on scaling. To close, we'll also cover an Elasticsearch alternative.

Improving Your Elasticsearch Performance

You might be either someone who's just decided to use Elasticsearch or someone who already uses Elasticsearch, sees your business growing, and wants to be prepared for what may come. Planning ahead always has benefits. So if you want to build and configure a high-performing Elasticsearch, here are the most important points to focus on.

1. Hardware

You can do all the optimization possible, but if you don't have enough hardware, you'll still fall short on performance. For high performance of Elasticsearch, you should mainly focus on cache, disk space, CPUs, and RAM. The reason you've chosen Elasticsearch instead of a traditional database is probably that you're dealing with a humongous amount of data and you want quick access. And hardware plays a very important role.

Do your homework, and understand your business needs. Think of how much and how often you'll have to store, query, and manipulate data. This will help you understand how much hardware you would need for smooth working. Along with quantity, quality also matters. For example, using SSDs would increase performance far more than when using HDDs.

2. Load Balancing

In production, you most probably have multiple nodes running. If you have a large number of requests, you'll have to divide this load between the nodes to decrease strain on a particular node. Load balancing is a straightforward way to do this. Load balancing is a feature that distributes the load coming to an endpoint across multiple nodes. This reduces the load on each node, thus increasing performance.

Load balancing in Elasticsearch is rather easy. Load balancers are a part of the Elasticsearch cluster by default. You just have to enable them. Configuring a node as a coordinating only node will enable smart load balancing. It will distribute the requests, collect results after processing, and merge these results to form and return a final result. You can also decide the number of load balancers and configure accordingly.

3. Indices

Indices are where data is stored in Elasticsearch. You can have one or more indices to store data. You don't necessarily have to store all your data in one index. Depending on the use case, you can set an index to store data for a month, a day, or an hour. If your nodes are heavy-indexing nodes, then you should have a high number for index buffer size. Index buffer size is the size of data stored in a buffer before it's written to the disk. By default, this value is 10% of the heap size. But if you have an index-heavy use case, then you might want to consider increasing this size.

Data in Elasticsearch is immutable. So when you create a document, it's stored in an index. But when you want to update the values of this document, you can't just modify the values in the existing document. In such cases, Elasticsearch creates a new document with the latest values. And to keep track of the latest documents, Elasticsearch uses version numbers. Therefore, the document with the highest version number is considered to be the latest. The problem here is that, now, the index has both old and new documents, increasing index size. The solution is to reindex. When you reindex your indices, they will have only the latest data and will save memory.

4. Designing Documents

Each index has documents where an instance of data is stored. Some businesses may have read-heavy documents and some might have write-heavy. Designing documents properly will reduce the time to process requests. Nested fields and parent-child structures make queries slower. You should aim to make the documents as flat as possible to make queries faster.

In many cases, the data making it to Elasticsearch is in raw format. Especially for system or application logs. Preprocessing data into proper fields can eliminate the need for processing data using ES queries. You can also have additional fields with values that you would frequently use. For example, if a document has five integer values and a sum of these values is something that you frequently need, you can find the sum while creating documents and store it in a separate field. So instead of finding the sum every time, you just need to fetch the sum field.

5. Sharding

Due to the large size of data, it's not recommended to store the whole set of data as one in an index. To make querying easy, the data in an index is divided into multiple parts known as shards. In Elasticsearch, every query runs in a single thread per shard. But multiple shards can be executed in parallel. So if you have multiple shards, you would have multiple threads running simultaneously. You may be thinking, "Well, that's great! I'll set the number of shards to maximum." But hold up. Because an excess number of shards has its own disadvantages.

Each shard utilizes some resources for mapping, storing cluster state, querying, etc. The higher the number of shards, the greater the resource utilization. And this would again decrease the performance. You need to find the right balance: not too few, not too many shards. There's no universal value for the number of shards that works for all use cases. You'll have to figure out the right number for your use case. A common approach is to start with one shard and keep increasing the number until you achieve the highest performance.

You should also take care of the shard size. The suggested shard size is between 30 GB to 40 GB. So if you think your index would store 300 GB of data, you should assign around eight to ten shards for each index.

6. Refresh Interval

When data comes to Elasticsearch, it's not immediately available after indexing. After indexing, data is present in an in-memory buffer. This data is written to segment when a refresh takes place. It's like refreshing a website to get the latest results. The refresh interval defines how often you refresh. By default, the refresh interval is one second, which means you have updated results available every second.

In most cases, the default refresh interval works fine. But you need to think about what works best for you. Refreshing indices use resources. If you don't need near-real-time data—for example, if you work only on data from the previous day—you can have a refresh once every day. So depending on your use case, choose your refresh interval.

7. Optimized Requests: Bulk Requests, Fields, Filters

Elasticsearch is mostly used to query bulk data. And it was created keeping that in mind. While running requests, you might think querying on individual indices might increase performance. But that's not true. If you're looking for data in one particular index, then, of course, run the request on that index only. But if you want to find data in all indices, then do it all at once. Bulk requests perform better than multiple individual requests.

In most cases, you want particular data from indices or documents. So there's no point in fetching all the data and then using only what you want. Instead, you can just fetch the data that you need. You can use the **_source** keyword to fetch the required fields or use terms aggregations to get unique values.

In cases where you're looking for something specific, make use of filters. Filters narrow down your search and help increase performance. You can use **range**, **match**, **terms**, and various features to narrow down your search and finally fetch only the data that you require.

This brings us to the end of the improving Elasticsearch performance section. These are approaches that would generally work for all use cases. But for specific use cases, you'll still have to monitor what's affecting the performance and then solve those issues. So how would you identify these issues? How would you know where the bottleneck lies? By monitoring performance.

Elasticsearch Performance Metrics to Monitor

An Elasticsearch setup is made up of a lot of different components. And all of these contribute to its performance. If you want to see how your Elasticsearch setup is performing, here are some of the important metrics to monitor.

Cluster Health

If you want a straightforward and short answer to "Is my Elasticsearch setup fine?", cluster health will give you the answer. You can use cluster health to check the health of the overall cluster or particular index or shard. You have three types of health status—red, yellow, and green—as stated in the official docs:

> On the shard level, a `red` status indicates that the specific shard is not allocated in the cluster, `yellow` means that the primary shard is allocated but replicas are not, and `green` means that all shards are allocated.

Indexing Rate

This metric tells you about the number of documents being indexed at a given point in time. A higher indexing rate is good. It means that the setup is able to keep up with the requirements.

Query Rate and Latency

Query rate tells you how many queries the Elasticsearch setup is able to execute. And query latency talks about delays. Delays could be due to queuing, poor configuration, etc. A high query rate and low latency are good.

Refresh Time

We've already gone through what refresh interval is. Refresh time talks about how much time a refresh takes. The lower the refresh time, the better. Spikes and high values of refresh time are of serious concern.

CPU Usage and Disk Space

With time, data keeps growing and more disk space is utilized. And with more data, you'll also need more CPU power to execute requests. You should monitor both CPU usage and disk space to see whether they're enough for the requirements. If either of these falls short, you'll want to add more hardware resources.

Scaling Elasticsearch

Scaling Elasticsearch isn't just adding more hardware. There's more than that. If you have an Elasticsearch setup and want to scale it, here are a few tips:

- Understand your business and its growth to avoid frequent upgrades.
- Analyze if your index is write-heavy or read-heavy and design indices and documents accordingly.
- Disable replicas for initial loads. And index before replicating.
- Configure Elasticsearch for cross-cluster replication.
- Reindex and merge indices.
- Allocate additional shards to cater if additional nodes are added in the future.

And apart from these, use your knowledge from monitoring to identify possible problems and address them. You can read more on Elasticsearch scaling here.

What, If Not Elasticsearch?

Elasticsearch is great at what it does. Although it has a lot of benefits, it's not perfect. Most important of all, it's not beginner-friendly. Likely every person who has worked on Elasticsearch has at least once thought "I wish this feature was better." So if you're looking for something that's not Elasticsearch but does everything Elasticsearch does and even more, it's time to check out Scalyr.

Scalyr is an event data analytics and log management tool that could replace Elasticsearch. The way Scalyr is designed, you can still use Kibana and the same old queries. But instead of Elasticsearch as the search engine, you'd be using Scalyr. Scalyr is really fast, even at petabyte scale. It provides complete log analytics, observability, scaled event data ingestion and storage, sub-second query response, and a lot more. If this makes you curious, we encourage you to take Scalyr for a test drive.

This post was written by Omkar Hiremath. Omkar is a cybersecurity analyst who is enthusiastic about cybersecurity, ethical hacking, data science, and Python. He's a part time bug bounty hunter and is keenly interested in vulnerability and malware analysis.

Like this article? Follow us on LinkedIn, Twitter, YouTube or Facebook to see the content we post.

Read more about Cyber Security

- Horizontal Scalability in Your Software: The What, Why, and How