The most important part of an application’s user experience is its search response time. The longer users wait for a response to their request, the quicker they lose interest and exit the application.

That’s why Elasticsearch has become a go-to search and analytics engine used for building applications. At the core of [Elastic Stack](https://www.elastic.co/elastic-stack), Elasticsearch is used for indexing, searching, and analyzing large sets of data in near real-time. It is the perfect tool for real estate applications that require prompt searches across a variety of different use cases.

At Xome®, we utilize Elasticsearch in areas of our digital properties where fast and relevant search responses are required. Even though Elasticsearch is already known for its efficiency, we follow a set of best practices to help optimize search performance and provide the best experience possible for our users.

## **What is Elasticsearch?**

Elasticsearch is a distributed open-source search and analytics engine built on top of [Apache Lucene](https://lucene.apache.org/), a full-text search engine library written in Java. It was developed by Elastic as a part of their Elastic Stack, a collection of open-source tools that Elasticsearch can integrate with seamlessly.

Other tools in the stack include Beats for data shippers, Logstash for data processing, and Kibana for data visualization and exploration.

Elasticsearch is commonly used for various purposes including full-text search, monitoring and alerting, log and event data analysis, and business analytics, among many others.

When used to build real estate applications, a few specific use cases include:

* Property search based on various criteria, such as price range and property type
* Geospatial search for properties based on their geographic location and proximity to places like schools and public transportation
* Geospatial mapping for integrating interactive maps
* Dynamic filtering for users to refine search results based on specific attributes
* Relevant property recommendations to users based on their search history, preferences, and behavior

## **Seven best practices for improving Elasticsearch search performance**

There are various ways to optimize the implementation of Elasticsearch when developing applications.

Search speed depends on factors such as query complexity, data volume, cluster configuration, and hardware resources allocated to the search operation. Since Elasticsearch depends significantly on the filesystem cache to speed up search performance, increasing the size of that cache and using upgraded hardware are basic best practices.

There are other best practices that are less obvious but just as important.

### **1. Search fewer fields**

Searching fewer fields reduces the amount of data that Elasticsearch needs to access and analyze during search operations. By limiting the search scope to only the most relevant fields, Elasticsearch can execute queries more efficiently, leading to faster search response times while reducing resource utilization.

Using a copy\_to directive is one way to improve search time. For example, with an index that contains properties that searches over both the property type and details of the property, the values are indexed into a single field.

This field can be called something like a type\_and\_details field:

“copy\_to” = “type\_and\_details”

When a user searches for home or auction properties, they may experience a more intuitive search with less latency.

### **2. Use preference to optimize cache utilization**

Elasticsearch uses three different caches to help improve search performance. These include the filesystem cache, request cache, and query cache. However, these caches may not help much when multiple search requests are run in a row.

By leveraging the preference parameter in search requests, subsequent searches can be directed to the same shard or replica where caching is more effective. This ensures that frequently accessed data and query results are cached at the shard level, increasing cache hit rates and reducing the need for redundant computations.

### **3. Use term query for exact matching and on keyword fields**

A term query is a type of query used to search for exact terms in the indexed data. When using the term query, Elasticsearch looks for exact matches of the search term in the inverted index without any text analysis or tokenization.

The term query can be used to find documents based on values such as price~~,~~ but should be avoided for text fields where the match query should be used instead.

### **4. Implement features like Elasticsearch Percolator**

Implementing other Elasticsearch features like [Percolator](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-percolate-query.html) can help to further optimize a user’s search experience. Elasticsearch Percolator is a reverse search where, when given a document, it finds all the search queries that match the document.

When a user’s search criteria does not return listings that interest them, they may set up alerts for those specific criteria. They will then be notified when new listings come on the market that match those criteria.

### **5. Optimize necessary index settings**

Configuring certain index settings can significantly impact indexing efficiency, such as the number of shards and replication settings.

* As a best practice, the number of shards should be a multiple of the number of data nodes in the cluster. For example, if there are 3 data nodes, it is better to have the number of shards be a multiple of 3 (6, 9, etc.).

For replication settings, it is better to have at least one replica for each index, so that data is always available even if one data node goes down. The number of replicas should be less than the number of data nodes. If there are three data nodes, you should not have more than two replicas.

### **6. Pre-index data**

Pre-indexing data eliminates the need for real-time indexing during search requests because it involves indexing data before it is queried. In this approach, data is indexed and prepared in advance. This helps ensure that it is readily available for search operations without the need for additional indexing tasks at the time of the search request.

### **7. Tune for indexing speed**

Increasing indexing speed means that newly ingested data becomes available for search queries more quickly, improving overall search performance.

There are a few practices recommended when tuning for indexing speed. Although indexing speed does not directly affect how fast searches are executed, there are indirect ways it improves search speed.

A few ways to tune for indexing speed include:

* Using bulk requests
* Using multiple workers/threads to send data to Elasticsearch
* Unsetting or increasing the refresh interval

Improvements in indexing speed can alleviate resource bottlenecks and improve the overall stability of the Elasticsearch cluster, indirectly benefiting property search performance for application users.

## **Indexing performance vs. search performance**

There is an inherent tradeoff between reducing indexing latency and solving for query latency. Optimizing for faster indexing may involve sacrificing query performance, as resources are prioritized for indexing tasks. On the other hand, optimizing search queries may require limiting indexing activity to avoid resource contention and ensure query responsiveness.

Compromising between the two depends on the use case and its priorities. One use case may prioritize fresh data while another requires faster queries. For example, real-time analytics applications like Domo may prioritize data freshness, while interactive real estate search applications like Xome may prioritize query responsiveness.

Improving Elasticsearch performance requires careful consideration of these tradeoffs.

## **Optimizing Elasticsearch for low latency search with less downtime at Xome**

High performance property searches are essential when building powerful real estate applications. This continues to be a high priority of the Xome technical teams as we try to solve users’ problems and promptly present them with the best-fitting properties that fit their search.