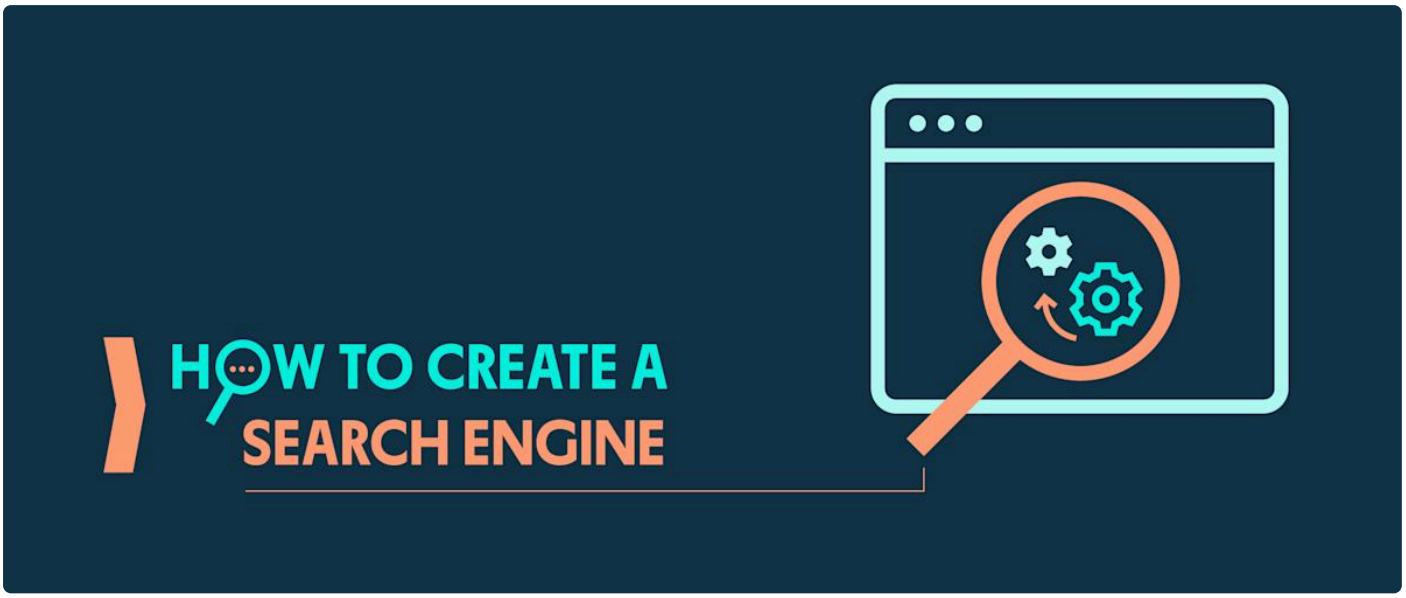
How to Create a Search Engine

Daniel Gleim · 2025-04-14



Search engines are a key part of the modern digital experience. Whether you're searching through web pages, product listings, research papers, or internal documents, a search engine helps you find relevant results quickly and efficiently.

But how do they work under the hood? What components are involved, and how do they process and retrieve information?

In this blog post, we'll walk through the fundamentals of creating a search engine — from the way data is stored and analyzed to how search queries are handled. We're not building something on the scale of Google, but the core concepts are the same, and they can be applied to many real-world use cases like site search, e-commerce filtering, or knowledge base navigation.

Toward the end, we'll also look at how security fits into the picture and how a plugin like **Search Guard** can help secure your search infrastructure when using tools like **Elasticsearch**.

Core Components of a Search Engine

At its core, a search engine is designed to help users find relevant information. To do this effectively, it relies on a few fundamental components:

Indexing

Before data can be searched, it needs to be processed and organized. Indexing is the process of converting raw content — such as text files, product data, or web pages — into a format that can be efficiently searched.

An index is essentially a specialized data structure that allows fast lookups. Each piece of content is typically stored as a **document**, with relevant fields like titles, descriptions, and timestamps. These

© 2026 floragunn GmbH · search-guard.com

documents are structured in a way that allows the search engine to quickly identify matching results based on user queries.

Document Structure

Most modern search engines work with structured or semi-structured data, often using a flexible document model. A document represents a single unit of searchable content, with fields that the engine can analyze and retrieve.

This model allows for easy filtering, sorting, and relevance tuning based on different parts of the content.

Text Analysis

Human language is complex — people use synonyms, abbreviations, and different phrasing for the same ideas. To handle this, search engines perform **text analysis**, which breaks down text into smaller units (tokens) and normalizes them.

This might include:

- Removing stopwords like “the” or “and”
- Lowercasing all terms
- Applying stemming or lemmatization (reducing words to their base or dictionary form, like “running” to “run”)
- Handling synonyms or spelling variations

Good text analysis improves the quality and relevance of search results, especially in full-text search scenarios.

How Search Works

Once your content is indexed and prepared, the next step is actually retrieving it — and that’s where the search process begins. At a high level, a search engine takes in a user query, compares it against the indexed content, and returns a ranked list of results. Here’s a closer look at how that happens:

Query Processing

When a user enters a search term, the engine doesn’t just look for exact matches. Instead, it processes the query much like it did with the indexed data: applying text analysis, breaking it into tokens, and interpreting its intent.

Depending on the engine’s capabilities, it might also recognize filters (like a date range or category), support advanced syntax (like wildcards or Boolean operators), or even handle natural language queries.

Matching and Scoring

After processing the query, the search engine compares it to the indexed content to find matches. But not all matches are created equal — some documents are more relevant than others.

To determine relevance, most search engines use a **scoring algorithm** that takes various factors into account, such as:

© 2026 floragunn GmbH · search-guard.com

- How often the search terms appear in a document
- Whether the terms appear in important fields (like a title)
- The length of the document
- Field-specific weights or boosts

The result is a **relevance score** for each document, which is used to rank the results from most to least relevant.

Full-Text vs. Keyword Search

Not all search behaves the same way. Depending on your data and what users are looking for, you might use one or both of the following approaches:

Keyword Search

Keyword search is about finding exact matches for specific terms, often within structured fields. It's typically used for precise filtering or when users know exactly what they're looking for.

Example

If you're searching a product database for `category:"laptop"` and `brand:"Dell"`, keyword search will only return items that match those exact terms in the respective fields. Keyword search is useful in cases like:

- Filtering search results by tags or categories
- Searching by IDs or codes (e.g. order numbers, SKU values)
- Combining filters with Boolean logic (AND, OR, NOT)

Full-Text Search

Full-text search is designed for natural language queries and unstructured content. It breaks down both the query and the documents into tokens, then uses scoring algorithms to rank results based on relevance rather than exact matches.

Example:

A user searches for *"best lightweight laptop for travel"*. Full-text search will analyze that phrase, identify key terms like "lightweight" and "laptop," and return results that match those concepts — even if the exact phrase doesn't appear in the documents.

Full-text search is ideal for:

- Articles, blog posts, or product descriptions
- User-generated content or reviews
- Any scenario where flexibility and relevance matter more than exact matches

Expanding Your Search Capabilities

Once you've got the basics of indexing and querying in place, you can start enhancing your search engine to provide a richer and more user-friendly experience. Here are some common features that take

© 2026 floragunn GmbH · search-guard.com

a search engine from functional to truly useful:

Filters and Facets

Filters let users narrow down results based on structured fields like category, price, or date. Facets — often used in e-commerce or documentation sites — show users which filter options are available and how many results match each one.

Example:

On an online store, users might filter laptops by brand, screen size, or price range, and see how many results fall under each filter.

Sorting and Pagination

Sorting allows results to be ordered by relevance, date, rating, or any other field. Pagination helps manage large result sets by showing results in pages rather than all at once.

Example:

A blog search might sort posts by “most recent” or “most relevant,” and display 10 posts per page.

Autocomplete and Suggestions

Autocomplete improves the search experience by predicting what the user is typing and offering suggestions. This can help reduce typos, speed up searches, and guide users toward popular or valid queries.

Example:

Typing “lap” into a search box might prompt suggestions like “laptop,” “laptop bag,” or “laptop charger.”

Spell Correction and Synonyms

Handling user errors or variations in wording is another key enhancement. Search engines can suggest corrections for misspelled words or match different terms with similar meaning.

Example:

A search for “notebok” might return results for “notebook,” or a search for “cell phone” could also match documents containing “smartphone.”

Securing Your Search Engine with Search Guard

A search engine isn’t just about performance and relevance — it’s also about protecting the data behind it. Whether you’re indexing internal company documents, customer information, or proprietary product data, securing that content is critical.

This is especially important when you’re using a powerful search platform like **Elasticsearch**, which is often deployed in enterprise environments and used to process sensitive information.

That’s where **Search Guard** comes in.

© 2026 floragunn GmbH · search-guard.com

Search Guard is a security plugin for Elasticsearch that helps you protect your search engine at multiple levels:

Authentication and access control

Ensure that only authorized users can access your search data. Search Guard supports a wide range of authentication methods, including LDAP, Kerberos, JSON web tokens, and more. You can also define fine-grained permissions to control what each user or role can see or do.

Encryption

Search Guard provides TLS encryption for both data in transit (between nodes and clients) and for REST/HTTP traffic, helping to prevent eavesdropping or tampering.

Audit logging

Track who accessed what, and when. This is especially useful for compliance and security audits.

By integrating Search Guard into your Elasticsearch setup, you ensure that your search engine is not only powerful, but also secure — from the moment data is indexed to the moment it's retrieved.

Conclusion

Creating a search engine is about more than just building an index and executing queries. It involves structuring your data, choosing the right analysis techniques, and designing features that enhance the user experience, like filters, autocomplete, and relevant results.

But as your search engine grows and handles more sensitive or mission-critical data, security becomes just as important as functionality. That’s where tools like **Search Guard** come in, providing authentication, encryption, and audit logging to ensure your search infrastructure remains safe and compliant.

Whether you’re creating a search engine for a website, an internal knowledge base, or an e-commerce platform, these core concepts and security practices will help you build a robust and efficient solution.

This article was published on the Search Guard blog: https://search-guard.com/blog/how-to-create-a-search-engine/
Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · Start a free trial