**WHAT IS ELASTIC SEARCH?**

By

MAKOUNDOU Paule-Theodora

DE TROGOFF Guilhem

SHARMA Karan

LEDROLE Pierre

Data Management and Corporate finance 24-2025

October 24, 2024

**I - Elasticsearch: A powerful distributed search and analytics engine for handling big data in real-time.**

## Introduction to Elasticsearch

Elasticsearch is a distributed, open-source search and analytics engine designed to handle large amounts of structured, unstructured, and semi-structured data in near real-time. It was initially released by Elastic in 2010 and is based on the Apache Lucene library, which provides the core functionality for full-text search.

1. **Data Storage and Indexing in Elasticsearch**

Elasticsearch works by storing data in JSON documents, which are indexed for fast search and retrieval. These documents are organized into *indexes*, and each index is split into shards to distribute across multiple nodes for scalability and fault tolerance. In Elasticsearch, an index is a fundamental data structure used to organize and store documents. It is conceptually similar to a database in relational databases but has its own unique properties suited for search and analytic.

1. **Data ingestion and search capabilities**

Elasticsearch allows both retrieving data and ingesting new data. You can index (add) data by sending JSON documents to Elasticsearch via its REST API. This data is automatically indexed to enable fast searches. Elasticsearch provides powerful search and query capabilities for this data. You can perform simple searches, full-text searches, or complex queries (filters, aggregations) to extract and analyze information in near real-time. Thus, it is designed to efficiently and rapidly handle both data ingestion and searching.

1. **BASE vs. ACID in Elasticsearch**

Elasticsearch follows the BASE (Basically Available, Soft state, Eventually consistent) model rather than the ACID (Atomicity, Consistency, Isolation, Durability) model.  Elasticsearch is a BASE system because it prioritizes availability and scalability over strict consistency. Here's a simple breakdown of why:

* Basically Available (B): Elasticsearch stays available even if some parts of the system (nodes) fail. It can still handle read and write operations, ensuring that the system keeps running without full outages.
* Soft State (S): The data stored in Elasticsearch may not be immediately consistent across all nodes. This is because data gets replicated and synchronized over time between nodes, meaning there could be a brief period when not all copies are the same.
* Eventually Consistent (E): Elasticsearch guarantees that the data will eventually be the same (consistent) across all nodes, but it doesn't happen instantly. Over time, the system ensures that all copies of the data will match.

Elasticsearch is BASE because it focuses on staying available and handling large amounts of data, even if the system isn't immediately consistent. This is useful when high availability and fast performance are more important than strict, instant accuracy.

1. **Types of Data stored in Elasticsearch**

Elasticsearch is versatile and can store and manage various types of data. Here are some common types of data you can find in Elasticsearch:

* Logs and Event Data:
* Examples: System logs, application logs, access logs, error logs.
* Use Case: Centralized logging and monitoring for troubleshooting and analysis.
* Full-Text Documents:
* Examples: Articles, blog posts, research papers, product descriptions.
* Use Case: Full-text search capabilities for content management systems and document retrieval.
* Structured Data:
* Examples: Tables with fields such as user information (name, email, age), product details (price, category, stock).
* Use Case: Storing and querying structured data for e-commerce platforms and user databases.
* Geospatial Data:
* Examples: Geographic coordinates, location data, maps.
* Use Case: Applications that require geolocation-based searches, such as restaurant finders or logistics tracking.
* Metrics and Time-Series Data:
* Examples: Sensor data, application performance metrics, financial data over time.
* Use Case: Real-time analytics and monitoring for performance and trends.
* Social Media Data:
* Examples: Posts, comments, likes, user interactions.
* Use Case: Analyzing user engagement and sentiment in social media platforms.
* Metadata:
* Examples: Metadata for images, videos, and other media files (e.g., titles, tags, descriptions).
* Use Case: Enabling search and retrieval of media based on their metadata.
* Machine Learning Data:
* Examples: Training datasets, model predictions, feature vectors.
* Use Case: Storing and analyzing machine learning results and outputs for various applications.

1. **Position on the CAP triangle**

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquementHere is a little reminder about what the CAP triangle is (below).

*Figure – CAP Triangle*

Elasticsearch is positioned as a CP (Consistency and Partition Tolerance) system on the CAP theorem triangle. Let’s explain why by breaking down the CAP Theorem:

* Consistency (C): Elasticsearch prioritizes consistency, meaning that every read reflects the most recent write. By default, it ensures that data across different nodes is synchronized before responding to a query. However, there is some flexibility in configuring consistency levels, but it generally ensures that all replicas have the same data.
* Partition Tolerance (P): Elasticsearch is designed to tolerate network partitions. It can operate and continue to function even if some nodes in the cluster are unreachable, ensuring that data remains available and consistent across the available nodes.
* Availability (A): While Elasticsearch does offer high availability by distributing data across multiple nodes, it may sacrifice immediate availability during network partitioning to maintain consistency. This means that if a network partition occurs, Elasticsearch might reject some requests temporarily until consistency is restored.

In Elasticsearch, when there is a network partition, it focuses on ensuring that data remains consistent rather than always being available. Some queries may fail during a partition until the system can ensure consistency again. This is a typical trade-off for a CP system under the CAP theorem.

Thus, Elasticsearch is positioned as CP on the CAP triangle, prioritizing Consistency and Partition Tolerance over Availability in the event of a partition.

1. **Why use Elasticsearch?**

Elasticsearch is a powerful, flexible database and search engine that is widely used in various scenarios due to its unique strengths. Here are several reasons why you would want to use Elasticsearch:

1. Fast Full-Text Search

Elasticsearch excels at performing real-time, full-text searches across large datasets. If you need to search through text-based data (like documents, logs, product descriptions, etc.), Elasticsearch is one of the best solutions. Its underlying inverted index structure ensures that search operations are extremely fast and efficient.

Use Case: E-commerce platforms for product searches, document management systems.

1. Scalability and Distributed Nature

Elasticsearch is horizontally scalable by design, allowing you to handle huge volumes of data by simply adding more nodes to the cluster. This makes it suitable for use cases where data grows rapidly and needs to be processed across multiple servers.

Use Case: Large-scale logging, monitoring, or analytics platforms.

1. Real-Time Analytics

Elasticsearch is not just a search engine but also supports real-time data analytics. You can query and aggregate data in near real-time, making it ideal for monitoring systems or dashboards where immediate insights are crucial.

Use Case: Log analysis, application performance monitoring, and security event tracking.

1. Document-Based Storage

Elasticsearch uses a document-oriented approach with data stored as JSON documents, making it flexible for various types of structured, unstructured, or semi-structured data. This makes it easy to adapt for applications where traditional relational schemas might be too rigid.

Use Case: Storing logs, events, and other types of semi-structured data.

1. Handling Large Volumes of Data

Elasticsearch is built to handle big data efficiently. It can index and query vast amounts of data quickly, which is critical for systems that generate high volumes of information, such as logging and monitoring solutions.

Use Case: Centralized logging systems like the ELK stack (Elasticsearch, Logstash, Kibana), data lakes.

1. Advanced Query and Aggregation Capabilities

It allows for highly sophisticated querying, including filtering, sorting, aggregation, and faceted searches. You can perform complex queries combining multiple fields and conditions, making it ideal for data exploration and ad-hoc reporting.

Use Case: Business intelligence, advanced analytics, and search-based applications.

1. Geospatial Queries

Elasticsearch supports geospatial queries, which is crucial for applications involving location-based data.

Use Case: Applications that rely on geolocation, such as ride-sharing apps, mapping tools, or local search engines.

1. Integration with Other Tools

Elasticsearch integrates well with various other tools, particularly in the ELK Stack (Elasticsearch, Logstash, and Kibana) and Beats for log shipping. It’s often used in combination with tools like Logstash for ingesting data and Kibana for data visualization and dashboards.

Use Case: Real-time data pipelines, centralized logging systems, data dashboards.

1. Fault Tolerance and High Availability

Elasticsearch is designed with fault tolerance and high availability in mind. It replicates data across multiple nodes, ensuring that if one node fails, the system can continue functioning without data loss.

Use Case: Mission-critical applications where downtime and data loss need to be minimized.

1. Flexible Deployment

Elasticsearch can be deployed in various environments, whether on-premises or in the cloud. It also has managed solutions like Elasticsearch Service on AWS and GCP, simplifying deployment and scaling.

Use Case: Cloud-based solutions for businesses that prefer managed services.

However, there are few situations where you should not use ElasticSearch. For instance, when doing complex transactions or working with relational data. When it comes to complex transactions Elasticsearch is not designed for ACID-compliant transactions, so it’s not ideal for applications where you need strict data consistency or complex transactions like in traditional relational databases. For relation data, if your data is highly relational and requires complex joins, a relational database (e.g., PostgreSQL, MySQL) might be more appropriate.

Elasticsearch is a great choice when you need to handle fast, full-text searches across large datasets, perform real-time analytics, or manage big data. It is also excellent for log analysis, handling geospatial queries, and powering search-heavy applications that demand speed and scalability.

**II – Use Cases, Challenges, and Future Prospects of Elasticsearch**

* 1. **Examples of uses, issues and cost of Elastic Search**

ElasticSearch is widely used for various purposes, primarily as a central component of the ELK stack (ElasticSearch, Logstash, Kibana) for log and event management. It enables aggregation, searching, and visualization of log data, making it ideal for monitoring systems, troubleshooting, and tracking operational metrics in real-time. Major companies like Netflix and GitHub rely on ElasticSearch for these purposes. It also powers search engines that provide high-performance search capabilities for websites, e-commerce platforms, and applications requiring fast, accurate search functionality. Platforms like Wikipedia and Shopify use ElasticSearch to handle large datasets and return relevant search results swiftly. Additionally, it is leveraged in real-time analytics scenarios, such as social media monitoring, where it processes and analyzes trends or sentiment in real-time from platforms like Twitter.

However, ElasticSearch has some issues. It is resource-intensive, requiring significant memory (RAM) and CPU, especially when handling large datasets, which can make scaling complex and costly. The risk of data loss can arise from improper configuration, particularly in cluster setup or replication, and this risk is amplified under heavy loads or during node failures. Moreover, management complexity can be challenging at scale, with cluster management, sharding, and query tuning requiring specialized expertise.

In terms of cost, ElasticSearch is open-source, and the base version is free to use. However, there are paid versions that offer additional features, such as security, machine learning capabilities, and alerting. Additionally, Elastic Cloud offers a managed, hosted version where the cost varies based on usage, such as the number of nodes, data volume, and performance requirements. While this option is more expensive, it simplifies the setup and scaling process.

* 1. **Benefits and limitations of Elastic Search**

ElasticSearch offers several benefits, particularly its ability to scale effectively. It is designed to handle large volumes of data across distributed systems, making it a suitable choice for big data use cases such as log aggregation and real-time analytics. One of its key advantages is real-time performance. ElasticSearch can return search results almost instantly, providing excellent user experience for real-time data analytics and search applications. Its full-text search capabilities are another major benefit, allowing it to handle complex search queries with features like fuzzy matching, autocompletion, and result ranking. This makes it ideal for applications that require flexible search functionality. Additionally, as an open-source solution, ElasticSearch offers significant flexibility, enabling organizations to customize and integrate it with other tools in the stack, such as Kibana and Logstash, for data visualization and ingestion.

However, ElasticSearch also has some limitations. It does not provide ACID compliance, meaning it doesn't offer strict transactional guarantees. Instead, it follows the BASE model, which prioritizes availability and partition tolerance over consistency, making it less suitable for applications that require transactional integrity, such as financial systems. Another limitation is the challenge of cluster management. Running large ElasticSearch clusters efficiently can be difficult, as it requires expertise in managing sharding, replication, and dealing with failures. Finally, ElasticSearch has high resource demands, requiring substantial amounts of RAM and processing power, which can drive up operational costs, particularly in high-performance environments with large datasets.

* 1. **Popularity of Elastic Search**

ElasticSearch is highly popular, particularly in sectors such as IT operations, e-commerce, and media. It has become the go-to tool for managing and querying log data, largely due to its widespread use in the ELK Stack (ElasticSearch, Logstash, Kibana), which is employed for log aggregation and visualization. The tool’s adoption across industries is significant, with large organizations and tech companies like Netflix, Slack, Uber, Shopify, Amazon, and eBay relying on it for their search and data analytics needs. Its ability to scale horizontally has made it a preferred solution for enterprises dealing with large and complex datasets.

The popularity of ElasticSearch is also driven by its strong open-source community. Frequent updates and a wide range of plugins ensure that it remains a flexible and adaptable solution for various use cases. As a result, ElasticSearch continues to be a critical tool for developers and data engineers, especially in environments that demand real-time processing and search-heavy applications.

* 1. **Competitors**

ElasticSearch faces competition from several key tools in the search and data analytics space. Its closest open-source competitor is Apache Solr, which offers similar search and indexing capabilities. However, Solr is less focused on real-time analytics and is typically more suited for traditional enterprise search needs. Another major competitor is Splunk, a commercial alternative known for log management and monitoring. While Splunk provides a more polished user experience, it comes at a much higher cost. Amazon CloudSearch is another rival, offering a fully managed search service through AWS. Although CloudSearch is easier to use than ElasticSearch, it sacrifices some flexibility and control. Additionally, some developers choose to work directly with Lucene, the underlying technology on which ElasticSearch is built, for more granular control over indexing and search functionality.

ElasticSearch was first released in 2010 and has since undergone significant evolution. Its integration with the ELK Stack has been particularly impactful, boosting its prominence in log management and real-time analytics. Over the years, ElasticSearch has expanded its capabilities to include cloud-based offerings, enhanced security features, and machine learning integrations, further solidifying its position as a leading tool in the industry.

* 1. **Future of Elasticsearch**

Elasticsearch is expected to continue expanding into areas such as machine learning and predictive analytics, where it can be used to detect anomalies, forecast trends, and analyze complex datasets in real-time. Its capabilities in these fields will likely grow as organizations demand more intelligent, data-driven insights. Additionally, cloud-based growth is a key aspect of ElasticSearch's future. With the rise of Elastic Cloud, its managed service offering, ElasticSearch’s cloud-native capabilities will become even more important as enterprises increasingly shift from on-premise solutions to cloud infrastructures.

The market for ElasticSearch is also experiencing significant changes. There is a clear shift toward fully managed services, particularly in the cloud, as many companies prefer to avoid the complexity of managing clusters in-house. As a result, major cloud providers like Amazon, Google Cloud, and Microsoft Azure have integrated ElasticSearch into their platforms. However, the open-source market has been impacted by changes to ElasticSearch's licensing. In 2021, Elastic changed its license model to limit how cloud providers could use its software, pushing some users towards alternatives like OpenSearch (an AWS fork of ElasticSearch) or other open-source search solutions.

Looking ahead, competition in the data analytics space is expected to increase, particularly as more real-time processing and AI integration become central to enterprise data strategies. This could lead ElasticSearch to further diversify its offerings to meet evolving market demands.

**III – Example Indexing and Querying in Elasticsearch: A Comparison with SQL and Scalability Features**

* 1. **Indexing in Elasticsearch**

Indexing in Elasticsearch is essentially the process of storing and organizing data so that it can be efficiently searched and retrieved. When we index a document, we add it to an index (like a table in SQL) where it becomes searchable.

* Index: A collection of documents (like a table in SQL).
* Document: The basic unit of information (like a row in SQL) containing various fields (key-value pairs).
* Field: Each key-value pair in a document (like columns in SQL).
* Shards: Indexes are split into smaller units called shards, which allow Elasticsearch to distribute data across multiple nodes for scalability and performance.

The Indexing Process:

1. Parsing: Elasticsearch breaks down a document into its individual fields during indexing.
2. Mapping: Elasticsearch automatically maps field types (or you can define your own mappings), telling Elasticsearch how to store and search the data.
3. Inverted Index: For text fields, Elasticsearch creates an inverted index that allows for fast full-text search by mapping terms to the documents that contain them.

Code Example (Python):

Indexing a document into Elasticsearch:

|  |
| --- |
| from elasticsearch import Elasticsearch    # Initialize the Elasticsearch client  es = Elasticsearch("http://localhost:007")    # Index a document  doc = {  'name': 'Karan',  'age': 24,  'occupation': 'Developer'  }  es.index(index='users', id=1, document=doc) |

Here, Elasticsearch function initializes the client and connects it to your Elasticsearch instance (in this case, running locally at “localhost:007”). Then we create a python dictionary (in this case, it is named as “doc”) representing the document we want to index, with fields like name, age, and occupation. After that we use es.index function to index the document into the specified index (in this case, it is named as “users”) which is like a table in SQL, with “id=1” being the unique identifier like a primary key for this document.

* 1. **Similarities with SQL and Code Comparison**

Elasticsearch shares a lot of similarities with SQL, making it easy to understand.

* Indexes vs. Tables: In Elasticsearch, indexes are like tables in SQL.
* Documents vs. Rows: Documents in Elasticsearch are the equivalent of rows in SQL.
* Fields vs. Columns: Fields in Elasticsearch correspond to columns in SQL tables.
* Queries: Elasticsearch supports querying, filtering, and even aggregations like SQL.
* Aggregations: Aggregations in Elasticsearch are like SQL's aggregate functions (e.g., “COUNT”, “SUM”, “AVG”).

The key Differences are:

* Full-Text Search: Elasticsearch is optimized for full-text search, whereas SQL is not built for this out of the box.
* Schema: Elasticsearch is schema-less, while SQL requires a predefined schema.
* Joins: SQL supports relational joins, while Elasticsearch does not natively support joins (though it has alternatives like nested documents and parent-child relationships).

Code Comparison (Examples):

* Basic Insert Operation:
* SQL: Insert a row into a table:

|  |
| --- |
| INSERT INTO users (id, name, age, occupation)  VALUES (1, 'Karan', 24, 'Developer'); |

* Elasticsearch (Python):

|  |
| --- |
| es.index(index='users', id=1, document={'name': 'John Doe', 'age': 30, 'occupation': 'Developer'}) |

* Search with Conditions (WHERE clause):
* SQL:

|  |
| --- |
| SELECT \* FROM users WHERE age > 25 AND occupation = 'Developer'; |

* Elasticsearch (Python):

|  |
| --- |
| es.search(index='users', query={  "bool": {  "must": [  {"range": {"age": {"gt": 25}}},  {"match": {"occupation": "Developer"}}  ]  }  }) |

* Aggregating Data (GROUP BY and COUNT):
* SQL: Count the number of users by occupation:

|  |
| --- |
| SELECT occupation, COUNT(\*) FROM users GROUP BY occupation; |

* Elasticsearch (Python):

|  |
| --- |
| es.search(index='users', size=0, aggs={  "group\_by\_occupation": {  "terms": {"field": "occupation.keyword"}  }  }) |

* 1. **Scalability of Elasticsearch**

Elasticsearch is designed to be highly scalable. As your data grows, Elasticsearch can scale horizontally by adding more nodes to your cluster. Key features like sharding and replication make it resilient and able to handle large volumes of data efficiently.

* Sharding: Each index is divided into shards, which are distributed across multiple nodes.
* Replication: Shards are replicated to provide high availability. If a node goes down, Elasticsearch will use replica shards to ensure data availability.

* 1. **Flexibility**

One of Elasticsearch's greatest strengths is its flexibility. It handles structured, semi-structured, and unstructured data seamlessly. You can store:

* Text and perform full-text search.
* Numbers, dates, and booleans for structured queries.
* Arrays and nested objects for more complex data models.
* Geospatial data for location-based searches.

Elasticsearch is also schema-less by default, meaning you don’t need to define a rigid structure upfront. You can index and search a wide variety of data types without setting up predefined mappings (though you can define mappings if needed).

* 1. **Usability**

Elasticsearch is relatively user-friendly, especially with features like:

* RESTful API: You can interact with Elasticsearch using simple HTTP requests (e.g., GET, POST), making it easy to use and integrate with different tools.
* Kibana: Kibana, part of the Elastic Stack, allows you to visualize and explore your data using a graphical interface. This makes Elasticsearch accessible to technical and non-technical users alike.
* Real-Time Data Ingestion: Elasticsearch indexes data in near real-time, meaning new data becomes searchable almost immediately after it's ingested.

**IV. Conclusion**

Elasticsearch stands as a robust and versatile tool for managing big data in real time. Its distributed architecture, scalability, and speed make it a powerful search and analytics engine, addressing the growing needs of modern data-driven organizations. Firstly, we explored Elasticsearch's core capabilities, demonstrating its effectiveness in handling large volumes of data efficiently, with rapid indexing and querying features.

Then, we saw the diverse use cases where Elasticsearch excels, from log and event data management to e-commerce and business intelligence applications. We also highlighted challenges, such as managing complex queries and optimizing resource consumption, while noting that its prospects remain strong, especially with continuous improvements and integrations with other technologies.

Finally, we provided a comparison between Elasticsearch and SQL, emphasizing the differences in querying capabilities and scalability. Elasticsearch offers more flexibility and performance when dealing with unstructured data, though it requires a learning curve compared to SQL's more familiar syntax. Nevertheless, its scalability features allow it to outperform traditional relational databases in certain large-scale scenarios, making it an invaluable tool for organizations looking to process and analyze data at scale.

In conclusion, Elasticsearch proves to be an essential engine in the modern data landscape, capable of delivering real-time search and analytics solutions with high scalability, addressing both current demands and future challenges.