ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Elasticsearch as a NoSQL Database for Fast Distributed Financial Service Applications

Ananth Majumdar

USA

ABSTRACT

This paper explores the implementation of Elasticsearch as a primary data store for fast, distributed financial service applications. It examines the advantages of Elasticsearch in the context of financial services, including its scalability, high performance, flexible schema, and powerful search and analytics capabilities. The study addresses key challenges encountered during implementation, such as eventual consistency, limited join capabilities, and field limits, providing practical solutions based on real-world experience. The paper also discusses implementation strategies, focusing on data modeling for complex financial instruments, query optimization, and performance tuning. By balancing Elasticsearch's strengths against its limitations, this case study demonstrates how financial service organizations can leverage this technology to build more scalable, performant, and adaptable systems capable of handling the increasing data volumes and real-time processing demands of modern finance.

***Corresponding author**

Ananth Majumdar, USA.

Received: June 03, 2022; **Accepted:** June 10, 2022; **Published:** June 22, 2022

Introduction

The Need for Advanced Database Solutions in Finance

In the rapidly evolving landscape of financial services, the need for fast, scalable, and flexible database solutions has never been more critical. Traditional relational databases, while robust and reliable, often struggle to meet the demands of modern financial applications that require real-time processing of vast amounts of data with flexible schemas. Enter Elasticsearch, a distributed, RESTful search and analytics engine that has gained significant traction as a NoSQL database solution.

This paper explores the use of Elasticsearch as a primary data store for fast, distributed financial service applications. We will examine its key advantages, address the challenges encountered during implementation, and provide insights from our real-world experience in deploying Elasticsearch in a financial services context.

Introduction to Elasticsearch

Elasticsearch is a distributed, open-source search and analytics engine built on Apache Lucene. Originally developed by Shay Banon in 2010, it has since become a core component of the Elastic Stack, alongside Logstash and Kibana. While Elasticsearch was initially designed as a search engine, its capabilities have expanded significantly, making it a powerful solution for various use cases, including as a primary data store for applications requiring fast, scalable data access and analysis.

At its core, Elasticsearch is a document-oriented database that stores complex data structures serialized as JSON documents. It offers near real-time search and analytics capabilities, typically with latency of one second or less from the time a document is indexed until it becomes searchable. This speed is achieved through its distributed nature and its use of inverted indices, which allow for fast full-text searches.

Key Features of Elasticsearch Include

- **Distributed Architecture:** Elasticsearch is designed to be horizontally scalable, allowing it to handle large volumes of data by distributing it across multiple nodes in a cluster.
- **RESTful API:** It provides a comprehensive RESTful API for indexing, searching, and managing data, making it easy to integrate with various applications and programming languages.
- **Powerful Query DSL:** Elasticsearch offers a flexible, JSON-based query domain-specific language (DSL) that supports complex queries, filters, and aggregations.
- **Schema-less:** While it allows for explicit mapping definitions, Elasticsearch can also operate in a schema-less mode, automatically detecting and mapping fields based on the data it receives.
- **Near Real-Time Operations:** It provides near real-time search and analytics capabilities, making it suitable for applications that require quick access to large datasets.
- **Built-in Analyzers:** Elasticsearch includes a variety of built-in analyzers for text processing, enabling powerful full-text search capabilities.
- **Aggregations:** It offers a rich set of aggregation capabilities for data analysis, including metrics, bucket, and pipeline aggregations.

While traditionally associated with search-heavy applications like e-commerce platforms or content management systems, Elasticsearch has found its way into various other domains, including log and event data analysis, application performance monitoring, and, as this paper explores, financial services.

The combination of Elasticsearch's speed, scalability, and flexibility makes it an attractive option for financial institutions dealing with large volumes of complex, rapidly changing data. As we will discuss in the following sections, these characteristics align well with many of the data management challenges faced in the financial sector, from real-time market data analysis to risk assessment and regulatory reporting.

Advantages of Elasticsearch for Financial Services
Scalability
Elasticsearch offers several compelling advantages that make it particularly well-suited for financial service applications. First and foremost is its scalability. In an industry where data volumes are constantly growing, Elasticsearch's ability to scale horizontally by adding more nodes to distribute data and processing is invaluable. This scalability ensures that as our financial data and query loads increase, our system can easily adapt without significant architectural changes.

High Performance
Another crucial advantage is Elasticsearch's high performance. Its distributed architecture and inverted index structure enable lightning-fast search and aggregation operations, even on large datasets. This speed is essential in financial services, where milliseconds can make a difference in decision-making processes.

Flexible Schema
The flexible schema of Elasticsearch, a hallmark of NoSQL databases, proved to be a significant asset in our implementation. Financial data structures can be complex and varied, and requirements often change rapidly. Elasticsearch's schema-less nature allowed us to adapt quickly to these changes without the need for time-consuming database migrations.

A prime example of this flexibility was our implementation of a task management system for Portfolio Administrators (PAs). This system needed to handle a wide variety of tasks, including monitoring cash flows, addressing guideline failures, processing new account openings, and managing overdrafts. Each of these task types required different information to be stored and displayed, which would typically necessitate separate tables or complex schema designs in a traditional relational database.

With Elasticsearch, we were able to store all these diverse tasks in a single index. The flexible schema allowed us to add new fields as required for each task type, without affecting the structure of existing documents. This flexibility was crucial as it enabled us to:

- Maintain a consistent high-level workflow for all tasks (claiming, actioning, and closing) while allowing for task-specific data fields.
- Easily introduce new task types without schema modifications or migrations.
- Implement subtasks for certain business needs, such as aggregating cashflows for an account, by simply adding new fields to the relevant documents.
- Adapt to evolving business requirements by adding or modifying fields as needed, without disrupting existing functionality.

This approach significantly reduced development time and increased our agility in responding to new business needs. It allowed us to focus on building features rather than managing complex data structures, all while maintaining the ability to perform powerful searches and analytics across the entire task dataset.

```
{
  "index1_tasks": {
    "mappings": {
      "properties": {
        "id": { "type": "keyword" },
        "type": { "type": "keyword" },
        "status": { "type": "keyword" },
        "createdAt": { "type": "date" },
        "updatedAt": { "type": "date" },
        "cfDetails": {
          "type": "object",
          "properties": {
            "flowAmount": { "type": "float" },
            "flowDate": { "type": "date" },
            "flowStatus": { "type": "keyword" },
            "flowAccount": { "type": "keyword" }
          }
        },
        "odDetails": { "type": "object" }
      }
    }
  },
```

```
  "index2_tasks": {
    "mappings": {
      "properties": {
        "id": { "type": "keyword" },
        "type": { "type": "keyword" },
        "status": { "type": "keyword" },
        "createdAt": { "type": "date" },
        "updatedAt": { "type": "date" },
        "cfDetails": { "type": "object" },
        "odDetails": {
          "type": "object",
          "properties": {
            "amount": { "type": "float" },
            "currency": { "type": "keyword" },
            "account": { "type": "keyword" }
          }
        }
      }
    }
  }
}
```

Figure 1: Elasticsearch mapping for different tasks with same Java representation but two different indices

Full-text Search Capabilities

While not initially a primary consideration, Elasticsearch's full-text search capabilities have proven unexpectedly valuable.

Moreover, we found a particularly impactful use case for full-text search in enhancing the user experience for our Portfolio Administrators (PAs) and Portfolio Managers (PMs). We leveraged Elasticsearch's powerful search capabilities to implement a typeahead functionality for security searches. This feature proved to be instrumental in the success of our application for several reasons:

- **Rapid Security Entry:** Financial professionals often need to quickly enter security information in their workflow. The typeahead functionality allowed them to find and select the correct security with minimal typing, significantly speeding up their processes.
- **Reduced Errors:** By providing suggestions as users type, we minimized the risk of data entry errors, which is crucial when dealing with financial securities.
- **Improved User Satisfaction:** PAs and PMs expressed a strong dislike for waiting and typing full security names or identifiers. The quick, responsive search functionality directly addressed this pain point, leading to higher user satisfaction and adoption of the application.
- **Handling Complex Security Names:** Financial securities often have long or complex names. Full-text search allows users to find the correct security by typing any part of its name or identifier, making the process more intuitive and flexible.
- **Performance at Scale:** Despite having a large database of securities, Elasticsearch's efficient indexing and search algorithms ensured that the typeahead functionality remained fast and responsive, even as our data grew.

This implementation showcases how Elasticsearch's full-text search capabilities can go beyond traditional document search use cases and directly contribute to core functionality in financial applications. By improving the speed and accuracy of security lookups, we were able to streamline operations for our financial professionals, demonstrating that seemingly small features can have a significant impact on the overall success and adoption of financial software [1].

Aggregations and Analytics

Elasticsearch's aggregation capabilities have been an added bonus, allowing for complex data analysis essential for financial reporting and risk assessment. The ability to perform advanced analytics directly within the database, rather than exporting data to separate analytics tools, has streamlined our processes considerably.

It was very helpful to improve the user experience by providing aggregate stats for the tasks and task details.

Challenges and Solutions

Eventual Consistency

Despite its many advantages, implementing Elasticsearch as our primary data store was not without challenges. The eventual consistency model, while beneficial for performance, posed potential issues in a financial context where data accuracy is paramount. We addressed this by leveraging Elasticsearch's version field and implementing optimistic locking at the application level. This approach ensures that if an update operation is attempted with an older version of a document, it fails, prompting the end-user to update with the latest data [2,3].

Limited Join Capabilities

Another significant challenge was Elasticsearch's limited join capabilities compared to traditional relational databases. To overcome this, we adopted a denormalization strategy, aiming to fit related data into single documents wherever possible.

Mapping and Field Limits

We also developed an innovative approach using multiple indices for the same object. In one index, we defined fields in detail, while in another, we defined them as objects. This strategy allowed us to maintain a single Java representation while circumventing Elasticsearch's limit of 1000 fields per index [4].

Reindexing for Mapping Changes

The need to reindex data when changing mappings was another hurdle we had to overcome. We mitigated this by carefully planning our data models upfront and using index aliases to switch between different mappings. Reindexing operations were scheduled during release windows to minimize disruption to our services.

Challenges	Solutions
Eventual Consistency	Version fields and optimistic locking
Limited Join Capabilities	Denormalization and multiple indices approach
Field Limits	Strategic use of object fields and multiple indices
Reindexing for Mapping Changes	Use of aliases and planned reindexing during release windows

Implementation Strategies

Our implementation strategy focused on careful data modeling to balance normalization and performance needs. We paid particular attention to handling complex financial instruments, often using nested objects to represent intricate relationships within a single document.

Query optimization was another key focus area. We implemented various techniques to improve query performance, including the strategic use of filter contexts, appropriate field mappings, and regular performance testing and tuning.

For scaling and performance tuning, we adopted a set of best practices including regular monitoring, optimizing cluster settings, and implementing custom routing where appropriate to improve shard usage.

Conclusion

In conclusion, our experience with Elasticsearch as a primary data store for financial services applications has been largely positive. The advantages of scalability, performance, and flexible schema have more than compensated for the challenges we encountered. While Elasticsearch required us to rethink some of our data modeling and consistency approaches, it has enabled us to build a more scalable, performant, and adaptable system.

As financial services continue to evolve and data volumes grow, we believe that solutions like Elasticsearch will play an increasingly important role. However, it's crucial to approach such implementations with a clear understanding of both the strengths and limitations of the technology. With careful planning and appropriate strategies, Elasticsearch can serve as a powerful

foundation for modern, high-performance financial service applications.

References

1. Request Body Search, ElasticSearch Guide [6.8], Elastic. (n.d.). Elastic https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-request-body.html.
2. Optimistic concurrency control, Elasticsearch Guide [6.8], Elastic. (n.d.). Elastic. https://www.elastic.co/guide/en/elasticsearch/reference/6.8/optimistic-concurrency-control.html.
3. Creating, indexing, and deleting a document, ElasticSearch: The Definitive Guide [2.X], Elastic. (n.d.). Elastic https://www.elastic.co/guide/en/elasticsearch/guide/current/distrib-write.html.
4. Mapping, ElasticSearch Guide [6.8], Elastic. (n.d.). Elastic https://www.elastic.co/guide/en/elasticsearch/reference/6.8/mapping.html.

Copyright:©2022 Ananth Majumdar This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.