FLOWX.AI

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / intro-to-elasticsearch

Contents

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to Elasticsearch

Elasticsearch itself is not inherently event-driven, it can be integrated into event-driven architectures or workflows. External components or frameworks detect and trigger events, and Elasticsearch is utilized to efficiently index and make the event data searchable. This integration allows event-driven systems to leverage Elasticsearch's powerful search and analytics capabilities for real-time processing and retrieval of event data.

What is Elasticsearch?

Elasticsearch is a powerful and highly scalable open-source search and analytics engine built on top of the Apache Lucene library. It is designed to handle a wide range of data types and is particularly well-suited for real-time search and data analysis use cases. Elasticsearch provides a distributed, document-oriented architecture, making it capable of handling large volumes of structured, semi-structured, and unstructured data.

How it works?

At its core, Elasticsearch operates as a distributed search engine, allowing you to store, search, and retrieve large amounts of data in near real-time. It uses a schema-less JSON-based document model, where data is organized into indices, which can be thought of as databases. Within an index, documents are stored, indexed, and made searchable based on their fields. Elasticsearch also provides powerful querying capabilities, allowing you to perform complex searches, filter data, and aggregate results.

Why it is useful?

One of the key features of Elasticsearch is its distributed nature. It supports automatic data sharding, replication, and node clustering, which enables it to handle massive amounts of data across multiple servers or nodes. This distributed architecture provides high availability and fault tolerance, ensuring that data remains accessible even in the event of hardware failures or network issues.

Elasticsearch integrates with various programming languages and frameworks through its comprehensive RESTful API. It also provides official clients for popular

languages like Java, Python, and JavaScript, making it easy to interact with the search engine in your preferred development environment.

Indexing & sharding

Indexing

Indexing refers to the process of adding, updating, or deleting documents in Elasticsearch. It involves taking data, typically in JSON format, and transforming it into indexed documents within an index. Each document represents a data record and contains fields with corresponding values. Elasticsearch uses an inverted index data structure to efficiently map terms or keywords to the documents containing those terms. This enables fast full-text search capabilities and retrieval of relevant documents.

Sharding

Sharding, on the other hand, is the practice of dividing index data into multiple smaller subsets called shards. Each shard is an independent, self-contained index that holds a portion of the data. By distributing data across multiple shards, Elasticsearch achieves horizontal scalability and improved performance. Sharding allows Elasticsearch to handle large amounts of data by parallelizing search and indexing operations across multiple nodes or servers.

Shards can be configured as primary or replica shards. Primary shards contain the original data, while replica shards are exact copies of the primary shards, providing redundancy and high availability. By having multiple replicas, Elasticsearch ensures data durability and fault tolerance. Replicas also enable parallel search operations, increasing search throughput.

Sharding offers several advantages. It allows data to be distributed across multiple nodes, enabling parallel processing and faster search operations. It also provides fault tolerance, as data is replicated across multiple shards. Additionally, sharding allows Elasticsearch to scale horizontally by adding more nodes and distributing the data across them.

The number of shards and their allocation can be determined during index creation or modified later. It is important to consider factors such as the size of the dataset, hardware resources, and search performance requirements when deciding on the number of shards.

For more details, check Elasticsearch documentation:

» Elasticsearch

Leveraging Elasticsearch for advanced indexing with FLOWX.AI

The integration between FLOWX.AI and Elasticsearch involves the indexing of specific keys or data from the

The fallback content to display on prerendering
to
The fallback content to display on prerendering
using Elasticsearch. This indexing process is initiated by the
The fallback content to display on prerendering
in a synchronous manner, sending the data to Elasticsearch. The data is then indexed or updated in the "process_instance" index.

To ensure effective indexing of process instances' details, a crucial step involves defining a mapping that specifies how Elasticsearch should index the received messages. This mapping is essential as the process instances' details often have specific formats. The process-engine takes care of this by automatically creating an index template during startup if it doesn't already exist. The index template acts as a blueprint, providing Elasticsearch with the necessary instructions on how to index and organize the incoming data accurately. By establishing and maintaining an appropriate index template, the integration between FLOWX.AI and Elasticsearch can seamlessly index and retrieve process instance information in a structured manner.

Kafka transport strategy

The fallback content to display on prerendering
transport strategy implies process-engine sending messages to a Kafka topic whenever there is data from a process instance to be indexed. Kafka Connect is then configured to read these messages from the topic and forward them to Elasticsearch for indexing.
This approach offers benefits such as fire-and-forget communication, where the process-engine no longer needs to spend time handling indexing requests. By decoupling the process-engine from the indexing process and leveraging Kafka as a messaging system, the overall system becomes more efficient and scalable. The process-engine can focus on its core responsibilities, while Kafka Connect takes care of transferring the messages to Elasticsearch for indexing.

To optimize indexing response time, Elasticsearch utilizes multiple indices created dynamically by the Kafka Connect connector. The creation of indices is based on the timestamp of the messages received in the Kafka topic. The frequency of

index creation, such as per minute, hour, week, or month, is determined by the timestamp format configuration of the Kafka connector.

It's important to note that the timestamp used for indexing is the process instance's start date. This means that subsequent updates received for the same object will be directed to the original index for that process instance. To ensure proper identification and indexing, it is crucial that the timestamp of the message in the Kafka topic corresponds to the process instance's start date, while the key of the message aligns with the process instance's UUID. These two elements serve as unique identifiers for determining the index in which a process instance object was originally indexed.

For more details on how to configure process instance indexing through Kakfa transport, check the following section:

» Configuring elasticsearch indexing

» Configuration guidelines

Was this page helpful?