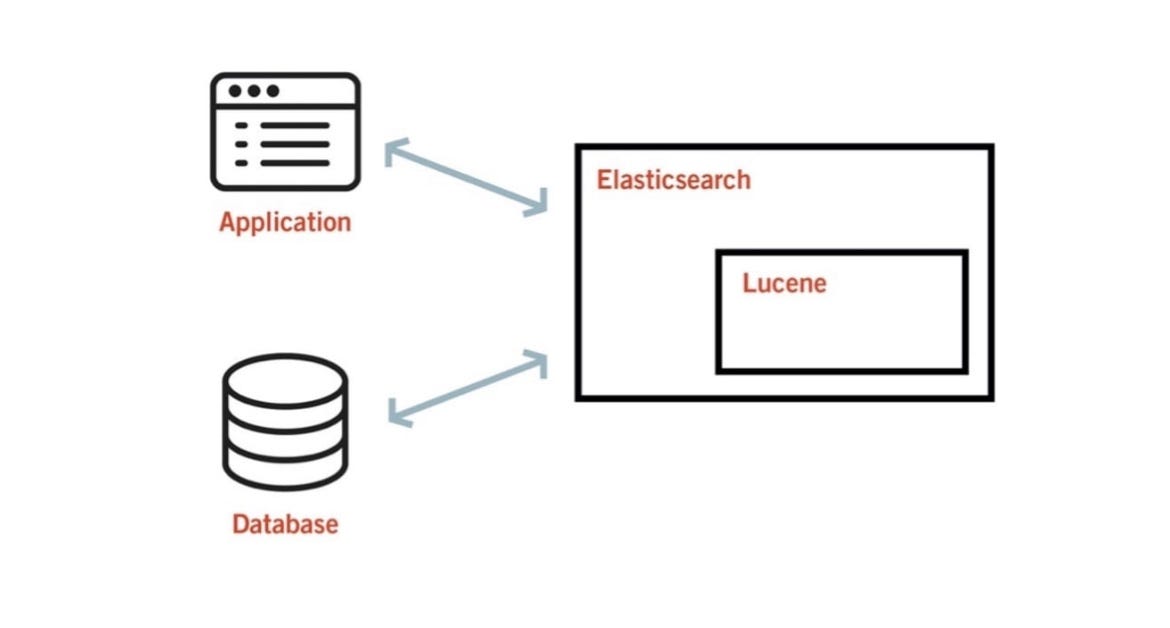
**Elasticsearch Overview**

Elasticsearch is a distributed, RESTful search and analytics engine based on Apache Lucene. It is designed to handle large volumes of data and provide real-time search capabilities. Elasticsearch is document-oriented, meaning it stores and manages data as documents in a flexible, schema-free manner.

**Lucene Engine**

Elastic search is uses Lucene Engine to create inverted index and to search the value based on index. On top of this Elasticsearch exposes a RESTful API to interact with the index, including inserting data, searching.



**Elasticsearch Index:**

Elasticsearch Index is collection of documents (data). Like a table in RDBMS or a collection in MongoDB. (***This index is not inverted index)***

**Inverted Index**

**Inverted Index** is a fundamental data structure used by Elasticsearch to enable fast and efficient full-text search capabilities.

* **Normal Index**: Maps indexed columns to row locations.
* **Inverted Index**: Maps words (terms) to documents (object inserted) containing those words. Terms are typically stored in ascending order for efficient searching.

**Creating an Inverted Index**

1. **Tokenize Documents**:

* Split documents into individual words (tokens).
* Remove stop words, punctuation, and apply stemming.

1. **Build the Inverted Index**:
   * Iterate through each token and document ID.
   * Create a mapping of tokens to document IDs.

**Example**:

* Document ID 1: "Elasticsearch is a search engine"
* Document ID 2: "Search engines like Elasticsearch are powerful"

The inverted index would look like this:

* **Term: "Elasticsearch"**

Documents: [Document ID 1, Document ID 2]

* **Term: "search"**

Documents: [Document ID 1, Document ID 2]

* **Term: "engine"**

Documents: [Document ID 1]

* **Term: "engines"**

Documents: [Document ID 2]

* **Term: "are"**

Documents: [Document ID 2]

* **Term: "powerful"**

Documents: [Document ID 2]

**Components of Inverted Index**:

Note: Inverted Index not only have Term-Document mapping

1. **Term-Document Mapping**: Maps each term to a list of documents containing that term.
2. **Posting Lists**: Contains document IDs and additional information like term frequency (TF) and positional data (optional).
3. **Term Dictionary**: Collection of unique terms with metadata such as Term ID, Term Frequency (TF), and Document Frequency (DF).

**Searching Architecture:**

When searching for a term, Elasticsearch ranks documents based on various factors:

* **Term Frequency (TF)**: How often the term appears in the document.
* **Inverse Document Frequency (IDF)**: How rare the term is across all documents.
* **Document Length**: Longer documents may be less relevant.
* **Other Factors**: Proximity of the term to the beginning of the document and document quality.

**Example**: Given the documents:

* Document ID 1: "Elasticsearch is a search engine"
* Document ID 2: "Search engines like Elasticsearch are powerful"

Searching for "search":

**Document ID 1** might appear first even though relevance of ‘search’ is only once in both the documents (note: it will not be searched as substring):

**Reason**:

* The term "search" appears close to the beginning.
* The document is shorter and more concise.

**Sharding**

Sharding involves partitioning a large dataset into smaller chunks called shards, distributed across different servers. In Elastic search, **Shard** is a Lucene index

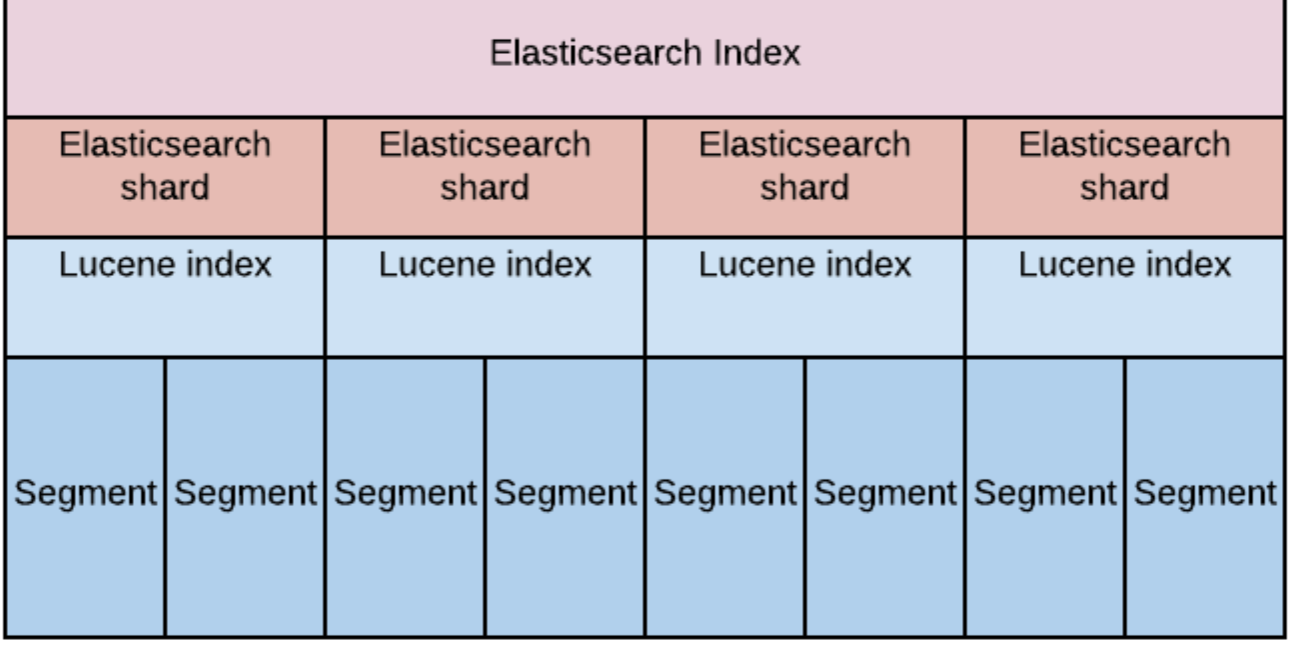
Here sharding is done based on elastic search index (not inverted index).

* **Shard**: A Lucene index that holds a portion of the data. Each shard contains:
* **Shard Metadata**: Information about the shard such as Shard ID and range of values.
* **Index Segments**: Portions of the index, each with document IDs, term dictionary, and posting lists.
* **Term Dictionary**: Contains unique terms.
* **Posting Lists**: Lists of documents with term information.
* **Stored Fields**: Document data stored for retrieval.

***Data subset (or a portion of our data after sharding) is present inside a Lucene index, specifically within each index segment (shard). Each index segment is a inverted index.***

***So, based on this we can know which shard have the searched data***

**Architecture:**



In above image, Index(table) is made as subset(based on range or hash or some algorithm) and shared between shards. Each shard act as a Lucene index. Index segment present inside Lucene index will have the inverted index(which have the data).

Elasticsearch operates in a cluster setup:

* **Cluster**: A collection of nodes (servers) that hold shards.
* **Primary Shard**: The original shard responsible for data storage.
* **Replica Shard**: Copies of the primary shard for redundancy and fault tolerance.
* **Master Node**: Manages metadata, node coordination, and cluster state.

***Master node will act like zookeeper on metadata, leader election and other things.***

A diagram of a diagram of a cylinder

Description automatically generated with medium confidence

**CRUD Operation**

**Writing Data**

Elasticsearch writes data using the following process:

1. **Indexing Operations** → **Translog File** → **RAM Buffer** → **Disk**
2. **Translog**: Records all changes made to the data (writes, updates, deletes).

* Data is first recorded in the translog.
* Periodically, data is flushed from the translog to the RAM buffer.
* When the RAM buffer is full, it is flushed to disk as a new segment.
* Each commit operation creates a new segment and is resource-intensive.

**Handling Node Failures**:

* During a failure, translog data is replayed from disk upon node restart.
* Replicas ensure data availability by promoting a replica to primary in case of node failure.

**Updating and Deleting Documents**

* **Updating**: Documents are immutable. Updating a document involves adding a new version with the same document ID and an incremented version number.
* **Deleting**: Documents are marked as deleted but not physically removed immediately. During segment merging, deleted documents are not included.

**Reading Data**

The search process involves:

1. Identifying relevant index(es) and shards.
2. Executing the search query on each selected shard.
3. Retrieving matching documents from each shard.
4. Calculating relevance scores and sorting results.
5. Returning the search response to the client.

**Communication with Elasticsearch Cluster**

* **Kibana**: A visualization tool for interacting with Elasticsearch data.
* **REST API**: Provides endpoints for performing operations on Elasticsearch.
* **Client Libraries**: Libraries for various programming languages to interact with Elasticsearch.

**Hands On:**

