What is Elasticsearch?

You know, for search (and analysis)

Elasticsearch is the distributed search and analytics engine at the heart of the Elastic Stack. Logstash and Beats facilitate collecting, aggregating, and enriching your data and storing it in Elasticsearch. Kibana enables you to interactively explore, visualize, and share insights into your data and manage and monitor the stack. Elasticsearch is where the indexing, search, and analysis magic happens.

Elasticsearch provides near real-time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data, or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches. You can go far beyond simple data retrieval and aggregate information to discover trends and patterns in your data. And as your data and query volume grows, the distributed nature of Elasticsearch enables your deployment to grow seamlessly right along with it.

While not *every* problem is a search problem, Elasticsearch offers speed and flexibility to handle data in a wide variety of use cases:

- Add a search box to an app or website
- Store and analyze logs, metrics, and security event data
- Use machine learning to automatically model the behavior of your data in real time
- Use Elasticsearch as a vector database to create, store, and search vector embeddings
- Automate business workflows using Elasticsearch as a storage engine
- Manage, integrate, and analyze spatial information using Elasticsearch as a geographic information system (GIS)
- Store and process genetic data using Elasticsearch as a bioinformatics research tool

We're continually amazed by the novel ways people use search. But whether your use case is similar to one of these, or you're using Elasticsearch to tackle a new problem, the way you work with your data, documents, and indices in Elasticsearch is the same.

Information out: search and analyze

While you can use Elasticsearch as a document store and retrieve documents and their metadata, the real power comes from being able to easily access the full suite of search capabilities built on the Apache Lucene search engine library.

Elasticsearch provides a simple, coherent REST API for managing your cluster and indexing and searching your data. For testing purposes, you can easily submit requests directly from the command line or through the Developer Console in Kibana. From your applications, you can use the Elasticsearch client for your language of choice: Java, JavaScript, Go, .NET, PHP, Perl, Python or Ruby.

Searching your data

The Elasticsearch REST APIs support structured queries, full text queries, and complex queries that combine the two. Structured queries are similar to the types of queries you can construct in SQL. For example, you could search the `gender` and `age` fields in your `employee` index and sort the matches by the `hire_date` field. Full-text queries find all documents that match the query string and return them sorted by *relevance*—how good a match they are for your search terms.

In addition to searching for individual terms, you can perform phrase searches, similarity searches, and prefix searches, and get autocomplete suggestions.

Have geospatial or other numerical data that you want to search? Elasticsearch indexes non-textual data in optimized data structures that support high-performance geo and numerical queries.

You can access all of these search capabilities using Elasticsearch's comprehensive JSON-style query language (Query DSL). You can also construct SQL-style queries to search and aggregate data natively inside Elasticsearch, and JDBC and ODBC drivers enable a broad range of third-party applications to interact with Elasticsearch via SQL.

Analyzing your data

Elasticsearch aggregations enable you to build complex summaries of your data and gain insight into key metrics, patterns, and trends. Instead of just finding the proverbial "needle in a haystack", aggregations enable you to answer questions like:

- How many needles are in the haystack?
- What is the average length of the needles?
- What is the median length of the needles, broken down by manufacturer?
- How many needles were added to the haystack in each of the last six months?

You can also use aggregations to answer more subtle questions, such as:

- What are your most popular needle manufacturers?
- Are there any unusual or anomalous clumps of needles?

Because aggregations leverage the same data-structures used for search, they are also very fast. This enables you to analyze and visualize your data in real time. Your reports and dashboards update as your data changes so you can take action based on the latest information.

What's more, aggregations operate alongside search requests. You can search documents, filter results, and perform analytics at the same time, on the same data, in a single request. And because aggregations are calculated in the context of a particular search, you're not just displaying a count of all size 70 needles, you're displaying a count of the size 70 needles that match your users' search criteria—for example, all size 70 *non-stick embroidery* needles.

But wait, there’s more

Want to automate the analysis of your time series data? You can use machine learning features to create accurate baselines of normal behavior in your data and identify anomalous patterns. With machine learning, you can detect:

- Anomalies related to temporal deviations in values, counts, or frequencies
- Statistical rarity
- Unusual behaviors for a member of a population

And the best part? You can do this without having to specify algorithms, models, or other data science-related configurations.

Scalability and resilience: clusters, nodes, and shards

Elasticsearch is built to be always available and to scale with your needs. It does this by being distributed by nature. You can add servers (nodes) to a cluster to increase capacity and Elasticsearch automatically distributes your data and query load across all of the available nodes. No need to overhaul your application, Elasticsearch knows how to balance multi-node clusters to provide scale and high availability. The more nodes, the merrier.

How does this work? Under the covers, an Elasticsearch index is really just a logical grouping of one or more physical shards, where each shard is actually a self-contained index. By distributing the documents in an index across multiple shards, and distributing those shards across multiple nodes, Elasticsearch can ensure redundancy, which both protects against hardware failures and increases query capacity as nodes are added to a cluster. As the cluster grows (or shrinks), Elasticsearch automatically migrates shards to rebalance the cluster.

There are two types of shards: primaries and replicas. Each document in an index belongs to one primary shard. A replica shard is a copy of a primary shard. Replicas provide redundant copies of your data to protect against hardware failure and increase capacity to serve read requests like searching or retrieving a document.

The number of primary shards in an index is fixed at the time that an index is created, but the number of replica shards can be changed at any time, without interrupting indexing or query operations.

It depends...

There are a number of performance considerations and trade offs with respect to shard size and the number of primary shards configured for an index. The more shards, the more overhead there is simply in maintaining those indices. The larger the shard size, the longer it takes to move shards around when Elasticsearch needs to rebalance a cluster.

Querying lots of small shards makes the processing per shard faster, but more queries means more overhead, so querying a smaller number of larger shards might be faster. In short...it depends.

As a starting point:

- Aim to keep the average shard size between a few GB and a few tens of GB. For use cases with time-based data, it is common to see shards in the 20GB to 40GB range.
- Avoid the gazillion shards problem. The number of shards a node can hold is proportional to the available heap space. As a general rule, the number of shards per GB of heap space should be less than 20.

The best way to determine the optimal configuration for your use case is through testing with your own data and queries.

In case of disaster

A cluster’s nodes need good, reliable connections to each other. To provide better connections, you typically co-locate the nodes in the same data center or nearby data centers. However, to maintain high availability, you also need to avoid any single point of failure. In the event of a major outage in one location, servers in another location need to be able to take over. The answer? Cross-cluster replication (CCR).

CCR provides a way to automatically synchronize indices from your primary cluster to a secondary remote cluster that can serve as a hot backup. If the primary cluster fails, the secondary cluster can take over. You can also use CCR to create secondary clusters to serve read requests in geo-proximity to your users.

Cross-cluster replication is active-passive. The index on the primary cluster is the active leader index and handles all write requests. Indices replicated to secondary clusters are read-only followers.

Care and feeding

As with any enterprise system, you need tools to secure, manage, and monitor your Elasticsearch clusters. Security, monitoring, and administrative features that are integrated into Elasticsearch enable you to use Kibana as a control center for managing a cluster. Features like downsampling and index lifecycle management help you intelligently manage your data over time.