Elasticsearch
Sizing and Capacity Planning

Dave Moore, Principal Solutions Architect
November 2019
Let's make sizing simple!
Housekeeping & Logistics

- Attendees are automatically muted when joining Zoom
- **Q+A** will be at the end of the webinar
- Ask questions for us in the **Zoom chat** during the webinar
  - Chat settings to: **All panelists and attendees**
  - Ask more questions on our discuss forum: **discuss.elastic.co**
- **Recording** will be available after the webinar and emailed to all registrants
Why Elastic?
Why Elastic?

SCALE
Distributed by design

SPEED
Find matches in milliseconds

RELEVANCE
Get highly relevant results
Search is a Foundation
Elastic Deployment Models

- Elastic Managed + Orchestration
- Self-Managed + Orchestration
- Self-Managed

**Elastic Cloud**
Elasticsearch Service

- The official fully managed Elastic Stack solution.
- Available on AWS, GCP, and Azure.

**Elastic Cloud Enterprise**

- Orchestrate the Elastic Stack on your infrastructure.
- Deploy anywhere.

**Self-Managed**

- Download and administer the Elastic Stack on your infrastructure.
- Deploy anywhere.
Webinar Overview
Overview

Let’s master the art of capacity planning for Elasticsearch.

Elasticsearch is the heart of the Elastic Stack.

Any production deployment of the Elastic Stack should be guided by capacity planning for Elasticsearch. Whether you use it for logs, metrics, traces, or search, and whether you run it yourself or in our cloud, you need to plan the infrastructure and configuration of Elasticsearch to ensure the health and performance of your deployment.
Let’s master the art of capacity planning for Elasticsearch.

Webinar Goals

Capacity planning is about estimating the type and amount of resources required to operate an Elasticsearch deployment. By the end of this webinar you will know:

• Basic computing resources
• Architecture, behaviors, and resource demands of Elasticsearch
• Methodologies to estimate the requirements of an Elasticsearch deployment
The Four Basic Computing Resources

Storage
- Where data persists
- e.g. Words in a book

Memory
- Where data is buffered
- e.g. Words you read

Compute
- Where data is processed
- e.g. Analyzing the words

Network
- Where data is transferred
- e.g. Speaking the words
Storage Resources

Storage

Where data persists

e.g. Words in a book

Storage Media

Solid State Drives (SSDs) offer best performance for “hot” workloads.

Hard Disk Drives (HDDs) are economic for “warm” and “frozen” storage.

RAID0 can improve performance.

RAID is optional as Elastic defaults to N+1 shard replication.

Standard performant RAID configurations are acceptable for hardware level high-availability (e.g. RAID 1/10/50 etc.)

Storage Attachment

Recommndations

- Direct Attached Storage (DAS)
- Storage Area Network (SAN)
- Hyperconverged
  (Recommended minimum ~ 3Gb/s, 250Mb/s)

Avoid

- Network Attached Storage (NAS)
  e.g. SMB, NFS, AFP. Network protocol overhead, latency, and costly storage abstraction layers make this a poor choice for Elasticsearch.
Memory Resources

How Elasticsearch Uses Memory

**JVM Heap**
Stores metadata about the cluster, indices, shards, segments, and fielddata. This should be **50% of available RAM**, and up to a **maximum of 30GB RAM** to avoid garbage collection.

**OS Cache**
Elasticsearch will use the remainder of available memory to cache data, improving performance dramatically by avoiding disk reads during full-text search, aggregations on doc values, and sorts.
Elasticsearch processes data in many ways that can be computationally expensive. Elasticsearch nodes have thread pools and thread queues that utilize the available compute resources. The quantity and performance of CPU cores governs the average speed and peak throughput of data operations in Elasticsearch.
How Elasticsearch Uses Network

**Bandwidth** is rarely a resource that constrains Elasticsearch. For very large deployments, the amount of data transfer for ingest, search, or replication between nodes can cause **network saturation**. In these cases, network connectivity can be upgraded to higher speeds, or the Elastic deployment can be split into two or more clusters and then searched as a single logical unit using **cross-cluster search** (CCS).
Elasticsearch Architecture
2.0 Elasticsearch Architecture

Terminology

Cluster  A **group of nodes** that work together to operate Elasticsearch.

Node  A **Java process** that runs the Elasticsearch software.

Index  A **group of shards** that form a logical data store.

Shard  A **Lucene index** that stores and processes a portion of an Elasticsearch index.

Segment  A **Lucene segment** that immutably stores a portion of a Lucene index.

Document  A **record** that is submitted to and retrieved from an Elasticsearch index.
2.1 Elasticsearch Architecture

**Elastic Stack**
- Kibana
- Elasticsearch
- Beats
- Logstash

**Elasticsearch Cluster**
- Coordinator
- Ingest
- Data
- Data
- Data
- Data
- Data
- Data
- Machine Learning
- Machine Learning
- Machine Learning

**Data Nodes**
- /var/lib/elasticsearch/data
  - Index 1 Shard 1
  - Index 2 Replica 1
  - Index 1 Replica 2
- /var/lib/elasticsearch/data
  - Index 1 Replica 1
  - Index 2 Shard 1
  - Index 1 Shard 2
# Nodes

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>Indexes, stores, and searches data</td>
<td>Extreme</td>
</tr>
<tr>
<td><strong>Master</strong></td>
<td>Manages cluster state</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Ingest</strong></td>
<td>Transforms inbound data</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Machine Learning</strong></td>
<td>Processes machine learning models</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Coordinator</strong></td>
<td>Delegates requests and merges search results</td>
<td>Low</td>
</tr>
</tbody>
</table>

## 2.2 Elasticsearch Architecture
Elasticsearch

Data Operations
There are four basic data operations in Elasticsearch. Each operation has its own resource demands. Every use case makes use of these operations, but they will favor some operations over others.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Processing a document and storing it in an index for future retrieval.</td>
</tr>
<tr>
<td>Delete</td>
<td>Removing a document from an index.</td>
</tr>
<tr>
<td>Update</td>
<td>Removing a document and indexing a replacement document.</td>
</tr>
<tr>
<td>Search</td>
<td>Retrieving one or more documents or aggregates from one or more indices.</td>
</tr>
</tbody>
</table>
3.1 Elasticsearch Data Operations

Index Operations: Data Processing Flow

Elasticsearch Cluster

Client → PUT → Coordinator

Ingest Node

- PUT
- PUT

- No

Coordinator → Route

Data Node

- Yes

- Ingest Pipeline?

- No

- Route

- Yes

- Text?

- No

Ingest Node

- Replicate

Lucene Shard

- Analyze

- Buffer

- Flush

- Segment

- Index 1 Shard 1

- Index 2 Replica 1

- Index 1 Replica 2

- /var/lib/elasticsearch/data

Lucene Replica

- Analyze

- Buffer

- Flush

- Segment

- Index 1 Replica 1

- Index 2 Shard 1

- Index 1 Shard 2

- /var/lib/elasticsearch/data

Network

Compute

Network

Compute

Storage
Delete Operations: Data Processing Flow

Elasticsearch Cluster

Client ➔ Coordinator ➔ Data Node ➔ Network ➔ Storage

- Client sends a DELETE request to the Coordinator.
- The Coordinator routes the request to the Data Node(s).
- The Data Node marks the data as deleted in its Lucene Shard.
- The data is then flushed to the Segment.
- The Segment is committed to the Storage.

Paths:
- Index 1 Shard 1
  - /var/lib/elasticsearch/data
  - Index 2 Replica 1
- Index 1 Shard 2
  - /var/lib/elasticsearch/data
- Index 1 Shard 1
  - /var/lib/elasticsearch/data
- Index 1 Shard 2
  - /var/lib/elasticsearch/data
- Index 1 Replica 2
  - /var/lib/elasticsearch/data
- Index 2 Replica 1
  - /var/lib/elasticsearch/data

Network:
- Mark as Deleted
- Buffer
- Flush
- Commit

Storage:
- Lucene Shard
- Lucene Replica
- Segment
- Index 1 Shard 1
- Index 2 Replica 1
- Index 1 Replica 2
- Index 1 Shard 2
- Index 2 Shard 1
- Index 1 Shard 2
Documents are immutable in Elasticsearch. When Elasticsearch updates a document, it deletes the original document and indexes the new, updated document. The two operations are performed atomically in each Lucene shard. This incurs the costs of a delete and index operation, except it does not invoke any ingest pipelines.

Update = Delete + (Index - Ingest Pipeline)
“Search” is a generic term for information retrieval. Elasticsearch has various retrieval capabilities, including but not limited to full-text searches, range searches, scripted searches, and aggregations. Search speed and throughput are affected by many variables including the configurations of the cluster, indices, queries, and hardware. Realistic capacity planning depends on empirical testing after applying the best practices for optimizing those configurations.

Elasticsearch executes searches in phases known informally as scatter, search, gather, and merge.
Search Operations: Data Processing Flow

Elasticsearch Cluster

Client ➔ GET ➔ Coordinator ➔ Route ➔ Scatter ➔ Data Node ➔ Parse ➔ Text? ➔ No ➔ Search ➔ Segment ➔ Analyze ➔ Segment ➔ Segment ➔ Segment

Data Node ➔ Parse ➔ Text? ➔ Yes ➔ Analyze ➔ Segment ➔ Segment ➔ Segment

Network ➔ Memory ➔ Network ➔ Compute ➔ Storage

Text?:
- Yes ➔ Analyze
- No ➔ Search

Network:
- Scatter ➔ Coordinator
- Route ➔ Coordinator

Memory:
- Parse ➔ Data Node
- Analyze ➔ Data Node

Compute:
- Segment ➔ Data Node
- Segment ➔ Data Node
- Segment ➔ Data Node

Storage:
- /var/lib/elasticsearch/data

Index 1 Shard 1
- Replica 1
- Replica 2

Index 2 Replica 1
- Shard 1
- Shard 2

Index 1 Shard 2
- Replica 1
- Replica 2

Index 2 Shard 1
- Replica 1
- Replica 2

Index 1 Shard 1
- Replica 1
- Replica 2

Index 2 Shard 2
- Replica 1
- Replica 2
## Use Cases

There are a few conventional usage patterns of Elasticsearch. Each favors one of the basic operations.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Use cases that favor index operations</th>
<th>Logging, Metrics, Security, APM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Heavy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search Heavy</td>
<td></td>
<td>App Search, Site Search, Analytics</td>
</tr>
<tr>
<td>Update Heavy</td>
<td></td>
<td>Caching, Systems of Record</td>
</tr>
<tr>
<td>Hybrid</td>
<td></td>
<td>Transactions Search</td>
</tr>
</tbody>
</table>

We will review the sizing methodologies for these use cases later in the workshop.
Overview

The following processes are applied to data on ingest.

**JSON Conversion**  Data can be larger or smaller on disk due to the format it is stored in.

**Indexing**  Data can be processed and indexed in various structures.

**Compression**  Data can be compressed for greater storage efficiency.

**Replication**  Data can be copied for greater fault tolerance and search throughput.
JSON Conversion

A Verbose Syntax

Elasticsearch stores the original document in the _source field in JSON format. JSON is more verbose than common delimited formats such as CSV, because each value is paired with the name of the field. The size of a log record from a delimited file could double or more. By contrast, JSON is less verbose than some formats such as XML.

It's Optional

Logging use cases require the _source field to return the source of truth for an event. Metrics use cases can discard the _source field because analysis is always done on aggregations of indexed fields, with no single record being important to look at.
## Indexing

### Data Structures

Elasticsearch indexes values in various data structures. Each data type has its own storage characteristics.

### Many Ways to Index

Some values can be indexed in multiple ways. String values are often indexed twice – once as a **keyword** for aggregations and once as **text** for full-text search. Values prone to error and ambiguity such as names and addresses can be indexed in multiple ways to support different search strategies.

<table>
<thead>
<tr>
<th>Original 4 Values</th>
<th>Indexed 6 Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-02-14T12:30:45 192.168.1.1 200 /index.html</td>
<td>date 2018-02-14T12:30:45</td>
</tr>
<tr>
<td></td>
<td>keyword 192.168.1.1</td>
</tr>
<tr>
<td></td>
<td>text 1:2 168:1 192:1</td>
</tr>
<tr>
<td></td>
<td>integer 200</td>
</tr>
<tr>
<td></td>
<td>keyword /index.html</td>
</tr>
<tr>
<td></td>
<td>text index:1 html:1</td>
</tr>
</tbody>
</table>
Compression

Elasticsearch can compress data using one of two different compression algorithms: LZ4 (the default) and DEFLATE, which saves up to 15% additional space at the expense of added compute time compared to LZ4. Typically Elasticsearch can compress data by 20 – 30%.
Shard Replication

Storage

Elasticsearch can replicate shards once or multiple times across data nodes to improve fault tolerance and search throughput. Each replica shard is a full copy of its primary shard.

Index and Search Throughput

Logging and metrics use cases typically have one replica shard, which is the minimum to ensure fault tolerance while minimizing the number of writes. Search use cases often have more replica shards to increase search throughput.
Complete Example

What you sent

2018-02-14T12:30:45 192.168.1.1 200 /index.html

What was stored

Primary

```json
"_source": {"timestamp":"2018-02-14T12:30:45","ip":"192.168.1.1","response":200,"url":"/index.html"}
```

Indexed values

```
2018-02-14T12:30:45|192.168.1.1|1:2 168:1 192:1|200|/index.html|index:1 html:1
```

Replica 1

```json
"_source": {"timestamp":"2018-02-14T12:30:45","ip":"192.168.1.1","response":200,"url":"/index.html"}
```

Indexed values

```
2018-02-14T12:30:45|192.168.1.1|1:2 168:1 192:1|200|/index.html|index:1 html:1
```

Replica n

...
Elasticsearch

Sizing Methodologies
Sizing Methodologies

There are two basic sizing methodologies that span the major use cases of Elasticsearch.

**Volume** Estimating the storage and memory resources required to store the expected amount of data and shards for each tier of the cluster.

**Throughput** Estimating the memory, compute, and network resources required to process the expected operations at the expected latencies and throughput for each operation and for each tier of the cluster.
Volume Sizing: Data Volume

**Discovery Questions**

- How much raw data (GB) will you index per day?
- How many days will you retain the data?
- What is the net expansion factor of the data?
  
  \[ \text{JSON Factor} \times \text{Indexing Factor} \times \text{Compression Factor} \]
- How many replica shards will you enforce?
- How much memory will you allocate per data node?
- What will be your memory: data ratio?

**Constants**

- Reserve +15% to stay under the disk watermarks.
- Reserve +5% for margin of error and background activities.
- Reserve the equivalent of a data node to handle failure.

\[
\begin{align*}
\text{Total Data (GB)} & = \text{Raw data (GB) per day} \times \text{Number of days retained} \times \text{Net expansion factor} \times (\text{Number of replicas} + 1) \\
\text{Total Storage (GB)} & = \text{Total Data (GB)} \times (1 + 0.15 \text{ Disk watermark threshold} + 0.05 \text{ Margin of error}) \\
\text{Total Data Nodes} & = \text{ROUNDUP}(\text{Total Storage (GB)} / \text{Memory per data node} / \text{Memory: data ratio}) + 1 \text{ Data node for failover capacity}
\end{align*}
\]
**Volume Sizing: Shard Volume**

**Discovery Questions**

- How many index patterns will you create?
- How many primary and replica shards will you configure?
- At what time interval will you rotate the indices, if at all?
- How long will you retain the indices?
- How much memory will you allocate per data node?

**Constants**

- Do not exceed 20 shards per GB of JVM Heap.
- Do not exceed 50GB per shard.

**Tip**

Collapse small daily indices into weekly or monthly indices to reduce shard count. Split large (>50GB) daily indices into hourly indices or increase the number of primary shards.

\[
\text{Total Shards} = \text{Number of index patterns} \times \text{Number of primaries} \times (\text{Number of replicas} + 1) \times \text{Total intervals of retention}
\]

\[
\text{Total Data Nodes} = \text{ROUNDUP}((\text{Total shards} / (20 \times \text{Memory per data node})))
\]
Throughput Sizing: Search Operations

Search use cases have targets for **search response time** and **search throughput** in addition to the storage capacity. These targets can demand more memory and compute resources.

Too many variables affect search response time to predict how any given capacity plan will affect it. But by empirically testing search response time and estimating the expected search throughput, we can estimate the required resources of the cluster to meet those demands.
Throughput Sizing: Search Operations

Discovery Questions

• What is your peak number of searches per second?
• What is your average search response time in milliseconds?
• How many cores and threads per core are on your data nodes?

Theory of the Approach

Rather than determine how resources will affect search speed, treat search speed as a constant by measuring it on your planned hardware. Then determine how many cores are needed in the cluster to process the expected peak search throughput. Ultimately the goal is to prevent the thread pool queues from growing faster than they are consumed. With insufficient compute resources, search requests risk being dropped.

Peak Threads = \( \text{ROUNDUP} (\text{Peak searches per second} \times \text{Average search response time in milliseconds} / 1000 \text{ Milliseconds}) \)

Thread Pool Size = \( \text{ROUNDUP} ((\text{Physical cores per node} \times \text{Threads per core} \times 3 / 2) + 1) \)

Total Data Nodes = \( \text{ROUNDUP} (\text{Peak threads} / \text{Thread pool size}) \)
Elasticsearch can use **shard allocation awareness** to allocate shards on specific hardware.

Index heavy use cases often use this to store indices on **Hot**, **Warm**, and **Frozen** tiers of hardware, and then schedule the migration of those indices from hot to warm to frozen to deleted or archived.

This is an economical way to store lots of data while optimizing performance for more recent data. During capacity planning, each tier must be sized independently and then combined.

<table>
<thead>
<tr>
<th>Tier</th>
<th>Goal</th>
<th>Example Storage</th>
<th>Example Memory:Storage Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot</td>
<td>Optimize for search</td>
<td>SSD DAS/SAN (&gt;200Gb/s)</td>
<td>1:30</td>
</tr>
<tr>
<td>Warm</td>
<td>Optimize for storage</td>
<td>HDD DAS/SAN (~100Gb/s)</td>
<td>1:160</td>
</tr>
<tr>
<td>Frozen</td>
<td>Optimize for archives</td>
<td>Cheapest DAS/SAN (&lt;100Gb/s)</td>
<td>1:1000+</td>
</tr>
</tbody>
</table>

Beware of recovery failures with this much data per node.
Dedicated Nodes

Elasticsearch nodes perform one or multiple roles. Often it makes sense to assign one role per node. You can optimize the hardware for each role and prevent nodes from competing for resources.

**Master**
Dedicated master nodes help ensure the stability of clusters by preventing other nodes from consuming any of their resources.

**Ingest**
Ingest nodes that run many pipelines or use many processors will demand extra compute resources.

**Machine Learning**
Machine learning nodes that run many jobs or use many splits, buckets, or complex aggregations will demand extra memory and compute resources.

**Coordinator**
Dedicated coordinating nodes can benefit hybrid use cases by offloading the merge phase of searches from data nodes that are constantly indexing.
A proper sizing takes the following steps:

1. For each applicable tier – Hot, Warm, Frozen – determine the largest of the following sizes:
   - Data volume
   - Shard volume
   - Indexing throughput
   - Search throughput

2. Combine the sizes of each tier

3. Make decisions on any dedicated nodes – Master, Coordinator, Ingest, Machine Learning
Additional Resources
Elastic Training
Empowering Your People

**Immersive Learning**
Lab-based exercises and knowledge checks to help master new skills

**Solution-based Curriculum**
Real-world examples and common use cases

**Experienced Instructors**
Expertly trained and deeply rooted in everything Elastic

**Performance-based Certification**
Apply practical knowledge to real-world use cases, in real-time
Elastic Consulting Services

**ACCELERATING YOUR PROJECT SUCCESS**

**PHASE-BASED PACKAGES**
Align to project milestones at any stage in your journey

**FLEXIBLE SCOPING**
Shifts resource as your requirements change

**GLOBAL CAPABILITY**
Provide expert, trusted services worldwide

**EXPERT ADVISORS**
Understand your specific use cases

**PROJECT GUIDANCE**
Ensures your goals and accelerate timelines
Q+A

Additional Resources

- Forums: https://discuss.elastic.co
- Cloud: https://www.elastic.co/products/elasticsearch/service
- Products + Solutions: https://www.elastic.co/products