## **Clustering, Shards, and High Availability**

### 1. **Elasticsearch Clustering**

A **cluster** is a collection of nodes (servers) that together hold your entire data and provide indexing and search capabilities.

Cluster health:

GET /\_cluster/health

View all nodes in a cluster:

GET /\_cat/nodes?v

### 2. **Sharding**

Shards split your index into parts. Each **primary shard** holds a subset of data.

Real-World:

An index with 1 billion docs is split into 5 shards, each with 200 million docs.

### 3. **Routing**

Routing determines how Elasticsearch places and finds documents across shards.

Example:

PUT /blog/\_doc/1?routing=user1

{

"title": "Custom Routing"

}

Useful when documents belong to a user, tenant, or group.

### 4. **Replica Sharding**

Each primary shard can have **replicas** for fault tolerance and scaling reads.

Real-World:

You can handle node failure without data loss.

### 5. **Index Creation with Shard and Replication Settings**

PUT /test-index

{

"settings": {

"number\_of\_shards": 3,

"number\_of\_replicas": 2

}

}

Total shards: 3 primary × (1 + 2 replicas) = 9 shards.

### 6. **Node Roles**

* **Master**: Manages cluster state and metadata.
* **Data**: Stores and queries data.
* **Ingest**: Preprocesses data.
* **Coordinating (client)**: Routes requests.

View node roles:

GET /\_nodes

### 7. **Three Node Elasticsearch Clustering**

Docker example for 3-node setup with static IPs and different roles. Use docker-compose.yml.

Structure:

* Node 1: master + data
* Node 2: master + data
* Node 3: data only

### 8. **Master Election**

The master node is elected by a majority vote among master-eligible nodes.

Set using:

node.master: true

discovery.seed\_hosts: ["es1", "es2", "es3"]

cluster.initial\_master\_nodes: ["es1", "es2"]

### 9. **High Availability Demo**

Create index with 1 shard, 1 replica:

PUT /ha-demo

{

"settings": {

"number\_of\_shards": 1,

"number\_of\_replicas": 1

}

}

Now kill one node and search again:

GET /ha-demo/\_search

Still works! Replicas ensure continuity.

### 10. **Two Shards Demo**

PUT /two-shard-demo

{

"settings": {

"number\_of\_shards": 2,

"number\_of\_replicas": 1

}

}

Inserts will be distributed across 2 shards.

### 11. **Initial Master Demo**

Set during first-time cluster setup to avoid split-brain:

cluster.initial\_master\_nodes: ["es1", "es2"]

### 12. **Node Roles Demo**

See roles for each node:

GET /\_cat/nodes?v&h=name,ip,node.role

Output example:

name ip node.role

es1 172.17.0.2 mdi

es2 172.17.0.3 mdi

es3 172.17.0.4 di

### 13. **Coordination-Only Node**

Purpose: Load balancer, no storage or computation.

node.master: false

node.data: false

node.ingest: false

### 14. **Optimistic Concurrency Control (OCC)**

Used to avoid overwriting during updates by checking \_seq\_no and \_primary\_term.

Step 1: Get doc metadata

GET /blog/\_doc/1

Step 2: Update with OCC

PUT /blog/\_doc/1?if\_seq\_no=5&if\_primary\_term=1

{

"title": "Safe Update"

}

### 15. **Cluster Settings – Persistent / Transient**

# Transient (temporary)

PUT /\_cluster/settings

{

"transient": {

"cluster.routing.allocation.enable": "none"

}

}

# Persistent (saved in cluster state)

PUT /\_cluster/settings

{

"persistent": {

"indices.recovery.max\_bytes\_per\_sec": "100mb"

}

}

### FAQ – How many Primary Shards do I need?

Depends on:

* Size of data (5–50GB/shard ideal)
* Query throughput
* Number of nodes

Rule: Start with fewer shards (e.g., 3–5). Split later with **index lifecycle management (ILM)** or **reindexing**.

### FAQ – How to Change Shard Count?

Not allowed after index creation.

Workaround:

* Reindex into a new index with new shard settings.

PUT /new-index

{

"settings": {

"number\_of\_shards": 2

}

}

POST /\_reindex

{

"source": {

"index": "old-index"

},

"dest": {

"index": "new-index"

}

}

### FAQ – How many Replica Shards do I need?

* **1 replica** = can handle 1 node failure.
* Add more replicas for:
  + More redundancy
  + Better read scaling

## Summary

| Concept | Summary |
| --- | --- |
| Clustering | Group of nodes for data/storage/search |
| Sharding | Split data across multiple parts for scalability |
| Replicas | Backup shards for fault tolerance |
| Routing | Routes documents to specific shards |
| Node Roles | Specialized nodes: master, data, ingest, coordination |
| High Availability | Achieved with multiple replicas and nodes |
| Concurrency Control | Prevents accidental overwrites using seq\_no/primary\_term |
| Cluster Settings | Tuned via transient or persistent APIs |

## QUIZ – Test Your Elasticsearch Cluster Knowledge

**1. What is the purpose of replica shards?**

a. Improve write performance  
b. Enable horizontal scaling  
c. Provide fault tolerance and scale read operations  
d. Reduce memory usage

**Correct:** c

**2. Which API adjusts the maximum shard recovery bandwidth persistently?**

a. \_cluster/stats  
b. \_cluster/health  
c. \_cluster/settings  
d. \_cat/shards

**Correct:** c

**3. What does a coordinating-only node do?**

a. Stores data  
b. Routes and aggregates queries only  
c. Acts as the elected master  
d. Processes ingest pipelines

**Correct:** b

**4. Can you change the number of primary shards of an existing index?**

a. Yes  
b. No  
c. Only for system indices  
d. Only if the cluster has < 5 nodes

**Correct:** b

**5. How does Elasticsearch handle concurrent updates?**

a. Uses version number  
b. Uses timestamps  
c. Uses optimistic concurrency control with \_seq\_no and \_primary\_term  
d. Locks documents

**Correct:** c