Size your shards

ECE ECK ELASTIC CLOUD HOSTED SELF MANAGED

A shard is a basic unit of storage in Elasticsearch. Every index is divided into one or more shards to help distribute data and workload across nodes in a cluster. This division allows Elasticsearch to handle large datasets and perform operations like searches and indexing efficiently. For more detailed information on shards, refer to nodes and shards.

General guidelines

Balancing the number and size of your shards is important for the performance and stability of an Elasticsearch cluster:

- Too many shards can degrade search performance and make the cluster unstable. This is referred to as *oversharding*.
- Very large shards can slow down search operations and prolong recovery times after failures.

To avoid either of these states, implement the following guidelines:

General sizing guidelines

- Aim for shard sizes between 10GB and 50GB
- Keep the number of documents on each shard below 200 million

Shard distribution guidelines

To ensure that each node is working optimally, distribute shards evenly across nodes. Uneven distribution can cause some nodes to work harder than others, leading to performance degradation and instability.

While Elasticsearch automatically balances shards, you need to configure indices with an appropriate number of shards and replicas to allow for even distribution across nodes.

If you are using data streams, each data stream is backed by a sequence of indices, each index potentially having multiple shards.

Despite these general guidelines, it is good to develop a tailored sharding strategy that considers your specific infrastructure, use case, and performance expectations.

Create a sharding strategy

The best way to prevent oversharding and other shard-related issues is to create a sharding strategy. A sharding strategy helps you determine and maintain the optimal number of shards for your cluster while limiting the size of those shards.

Unfortunately, there is no one-size-fits-all sharding strategy. A strategy that works in one environment may not scale in another. A good sharding strategy must account for your infrastructure, use case, and performance expectations.

The best way to create a sharding strategy is to benchmark your production data on production hardware using the same queries and indexing loads you'd see in production. For our recommended methodology, watch the quantitative cluster sizing video. As you test different shard configurations, use Kibana's Elasticsearch monitoring tools to track your cluster's stability and performance.

The performance of an Elasticsearch node is often limited by the performance of the underlying storage. Review our recommendations for optimizing your storage for indexing and search.

The following sections provide some reminders and guidelines you should consider when designing your sharding strategy. If your cluster is already oversharded, see Reduce a cluster's shard count.

Sizing considerations

Keep the following things in mind when building your sharding strategy.

Searches run on a single thread per shard

Most searches hit multiple shards. Each shard runs the search on a single CPU thread. While a shard can run multiple concurrent searches, searches across a large number of shards can deplete a node's search thread pool. This can result in low throughput and slow search speeds.

Each index, shard, segment and field has overhead

Every index and every shard requires some memory and CPU resources. In most cases, a small set of large shards uses fewer resources than many small shards.

Segments play a big role in a shard's resource usage. Most shards contain several segments, which store its index data. Elasticsearch keeps some segment metadata in heap memory so it can be quickly retrieved for searches. As a shard grows, its segments are merged into fewer, larger segments. This decreases the number of segments, which means less metadata is kept in heap memory.

Every mapped field also carries some overhead in terms of memory usage and disk space. By default Elasticsearch will automatically create a mapping for every field in every document it indexes, but you can switch off this behavior to take control of your mappings.

Moreover every segment requires a small amount of heap memory for each mapped field. This per-segment-per-field heap overhead includes a copy of the field name, encoded using ISO-8859-1 if applicable or UTF-16 otherwise. Usually this is not noticeable, but you may need to account for this overhead if your shards have high segment counts and the corresponding mappings contain high field counts and/or very long field names.

Elasticsearch automatically balances shards within a data tier

A cluster's nodes are grouped into data tiers. Within each tier, Elasticsearch attempts to spread an index's shards across as many nodes as possible. When you add a new node or a node fails, Elasticsearch automatically rebalances the index's shards across the tier's remaining nodes.

Best practices

Where applicable, use the following best practices as starting points for your sharding strategy.

Delete indices, not documents

Deleted documents aren't immediately removed from Elasticsearch's file system. Instead, Elasticsearch marks the document as deleted on each related shard. The marked document will continue to use resources until it's removed during a periodic segment merge.

When possible, delete entire indices instead. Elasticsearch can immediately remove deleted indices directly from the file system and free up resources.

Use data streams and ILM for time series data

Data streams let you store time series data across multiple, time-based backing indices. You can use index lifecycle management (ILM) to automatically manage these backing indices.

One advantage of this setup is automatic rollover, which creates a new write index when the current one meets a defined `max_primary_shard_size`, `max_age`, `max_docs`, or `max_size` threshold. When an index is no longer needed, you can use ILM to automatically delete it and free up resources.

ILM also makes it easy to change your sharding strategy over time:

- **Want to decrease the shard count for new indices?**
  Change the `index.number_of_shards` setting in the data stream's matching index template.
- **Want larger shards or fewer backing indices?**
  Increase your ILM policy's rollover threshold.

- **Need indices that span shorter intervals?**
  Offset the increased shard count by deleting older indices sooner. You can do this by lowering the `min_age` threshold for your policy's delete phase.

Every new backing index is an opportunity to further tune your strategy.

Aim for shards of up to 200M documents, or with sizes between 10GB and 50GB

There is some overhead associated with each shard, both in terms of cluster management and search performance. Searching a thousand 50MB shards will be substantially more expensive than searching a single 50GB shard containing the same data. However, very large shards can also cause slower searches and will take longer to recover after a failure.

There is no hard limit on the physical size of a shard, and each shard can in theory contain up to just over two billion documents. However, experience shows that shards between 10GB and 50GB typically work well for many use cases, as long as the per-shard document count is kept below 200 million.

You may be able to use larger shards depending on your network and use case, and smaller shards may be appropriate for certain use cases.

If you use ILM, set the rollover action's `max_primary_shard_size` threshold to `50gb` to avoid shards larger than 50GB and `min_primary_shard_size` threshold to `10gb` to avoid shards smaller than 10GB.

To see the current size of your shards, use the cat shards API.

```
GET _cat/shards?v=true&h=index,prirep,shard,store&s=prirep,store&bytes=gb
```

The `pri.store.size` value shows the combined size of all primary shards for the index.

```
index                               prirep shard store
.ds-my-data-stream-2099.05.06-000001  p      0      50gb
...
```

If an index's shard is experiencing degraded performance from surpassing the recommended 50GB size, you may consider fixing the index's shards' sizing. Shards are immutable and therefore their size is fixed in place, so indices must be copied with corrected settings. This requires first ensuring sufficient disk to copy the data. Afterwards, you can copy the index's data with corrected settings via one of the following options:

- running Split Index to increase number of primary shards
- creating a destination index with corrected settings and then running Reindex

Kindly note performing a Restore Snapshot and/or Clone Index would be insufficient to resolve shards' sizing.

Once a source index's data is copied into its destination index, the source index can be removed. You may then consider setting Create Alias against the destination index for the source index's name to point to it for continuity.

See this fixing shard sizes video for an example troubleshooting walkthrough.

Master-eligible nodes should have at least 1GB of heap per 3000 indices

The number of indices a master node can manage is proportional to its heap size. The exact amount of heap memory needed for each index depends on various factors such as the size of the mapping and the number of shards per index.

As a general rule of thumb, you should have fewer than 3000 indices per GB of heap on master nodes. For example, if your cluster has dedicated master nodes with 4GB of heap each then you should have fewer than 12000 indices. If your master nodes are not dedicated master nodes then the same sizing guidance applies: you should reserve at least 1GB of heap on each master-eligible node for every 3000 indices in your cluster.

Note that this rule defines the absolute maximum number of indices that a master node can manage, but does not guarantee the performance of searches or indexing involving this many indices. You must also ensure that your data nodes have adequate resources for your workload and that your overall sharding strategy meets all your performance requirements. See also Searches run on a single thread per shard and Each index, shard, segment and field has overhead.

To check the configured size of each node's heap, use the cat nodes API.

```
GET _cat/nodes?v=true&h=heap.max
```

You can use the cat shards API to check the number of shards per node.

```
GET _cat/shards?v=true
```

Add enough nodes to stay within the cluster shard limits

Cluster shard limits prevent creation of more than 1000 non-frozen shards per node, and 3000 frozen shards per dedicated frozen node. Make sure you have enough nodes of each type in your cluster to handle the number of shards you need.

Allow enough heap for field mappers and overheads

Mapped fields consume some heap memory on each node, and require extra heap on data nodes. Ensure each node has enough heap for mappings, and also allow extra space for overheads associated with its workload. The following sections show how to determine these heap requirements.

Mapping metadata in the cluster state

Each node in the cluster has a copy of the cluster state. The cluster state includes information about the field mappings for each index. This information has heap overhead. You can use the Cluster stats API to get the heap overhead of the total size of all mappings after deduplication and compression.

```
GET _cluster/stats?human&filter_path=indices.mappings.total_deduplicated_mapping_size*
```

This will show you information like in this example output:

```
{
  "indices": {
    "mappings": {
      "total_deduplicated_mapping_size": "1gb",
      "total_deduplicated_mapping_size_in_bytes": 1073741824
    }
  }
}
```

Retrieving heap size and field mapper overheads

You can use the Nodes stats API to get two relevant metrics for each node:

- The size of the heap on each node.
- Any additional estimated heap overhead for the fields per node. This is specific to data nodes, where apart from the cluster state field information mentioned above, there is additional heap overhead for each mapped field of an index held by the data node. For nodes which are not data nodes, this field may be zero.

```
GET _nodes/stats?human&filter_path=nodes.*.name,nodes.*.indices.mappings.total_estimated_overhe
```

For each node, this will show you information like in this example output:

```
{
  "nodes": {
    "USpTGYaBSIKbgSUJR2Z9lg": {
      "name": "node-0",
      "indices": {
```

```
        "mappings": {
          "total_estimated_overhead": "1gb",
          "total_estimated_overhead_in_bytes": 1073741824
        }
      },
      "jvm": {
        "mem": {
          "heap_max": "4gb",
          "heap_max_in_bytes": 4294967296
        }
      }
    }
  }
}
```

Consider additional heap overheads

Apart from the two field overhead metrics above, you must additionally allow enough heap for Elasticsearch's baseline usage as well as your workload such as indexing, searches and aggregations. 0.5GB of extra heap will suffice for many reasonable workloads, and you may need even less if your workload is very light while heavy workloads may require more.

Example

As an example, consider the outputs above for a data node. The heap of the node will need at least:

- 1 GB for the cluster state field information.
- 1 GB for the additional estimated heap overhead for the fields of the data node.
- 0.5 GB of extra heap for other overheads.

Since the node has a 4GB heap max size in the example, it is thus sufficient for the total required heap of 2.5GB.

If the heap max size for a node is not sufficient, consider avoiding unnecessary fields, or scaling up the cluster, or redistributing index shards.

Note that the above rules do not necessarily guarantee the performance of searches or indexing involving a very high number of indices. You must also ensure that your data nodes have adequate resources for your workload and that your overall sharding strategy meets all your performance requirements. See also Searches run on a single thread per shard and Each index, shard, segment and field has overhead.

Avoid node hotspots

If too many shards are allocated to a specific node, the node can become a hotspot. For example, if a single node contains too many shards for an index with a high indexing volume, the node is likely to have issues.

To prevent hotspots, use the `index.routing.allocation.total_shards_per_node` index setting to explicitly limit the number of shards on a single node. You can configure `index.routing.allocation.total_shards_per_node` using the update index settings API.

```
PUT my-index-000001/_settings
```

```
{
  "index" : {
    "routing.allocation.total_shards_per_node" : 5
  }
}
```

Avoid unnecessary mapped fields

By default Elasticsearch automatically creates a mapping for every field in every document it indexes. Every mapped field corresponds to some data structures on disk which are needed for efficient search, retrieval, and aggregations on this field. Details about each mapped field are also held in memory. In many cases this overhead is unnecessary because a field is not used in any searches or aggregations. Use *Explicit mapping* instead of dynamic mapping to avoid creating fields that are never used. If a collection of fields are typically used together, consider using `copy_to` to consolidate them at index time. If a field is only rarely used, it may be better to make it a Runtime field instead.

You can get information about which fields are being used with the Field usage stats API, and you can analyze the disk usage of mapped fields using the Analyze index disk usage API. Note however that unnecessary mapped fields also carry some memory overhead as well as their disk usage.

Reduce a cluster's shard count

If your cluster is already oversharded, you can use one or more of the following methods to reduce its shard count.

Create indices that cover longer time periods

If you use ILM and your retention policy allows it, avoid using a `max_age` threshold for the rollover action. Instead, use `max_primary_shard_size` to avoid creating empty indices or many small shards.

If your retention policy requires a `max_age` threshold, increase it to create indices that cover longer time intervals. For example, instead of creating daily indices, you can create indices on a weekly or monthly basis.

Delete empty or unneeded indices

If you're using ILM and roll over indices based on a `max_age` threshold, you can inadvertently create indices with no documents. These empty indices provide no benefit but still consume resources.

You can find these empty indices using the cat count API.

```
GET _cat/count/my-index-000001?v=true
```

Once you have a list of empty indices, you can delete them using the delete index API. You can also delete any other unneeded indices.

```
DELETE my-index-000001
```

Force merge during off-peak hours

If you no longer write to an index, you can use the force merge API to merge smaller segments into larger ones. This can reduce shard overhead and improve search speeds. However, force merges are resource-intensive. If possible, run the force merge during off-peak hours.

```
POST my-index-000001/_forcemerge
```

Shrink an existing index to fewer shards

If you no longer write to an index, you can use the shrink index API to reduce its shard count.

ILM also has a shrink action for indices in the warm phase.

Combine smaller indices

You can also use the reindex API to combine indices with similar mappings into a single large index. For time series data, you could reindex indices for short time periods into a new index covering a longer period. For example, you could reindex daily indices from October with a shared index pattern, such as `my-index-2099.10.11`, into a monthly `my-index-2099.10` index. After the reindex, delete the smaller indices.

```
POST _reindex
```

```
{
  "source": {
```

```
  "index": "my-index-2099.10.*"
 },
 "dest": {
  "index": "my-index-2099.10"
 }
}
```

Troubleshoot shard-related errors

Here's how to resolve common shard-related errors.

this action would add [x] total shards, but this cluster currently has [y]/[z] maximum shards open;

The `cluster.max_shards_per_node` cluster setting limits the maximum number of open shards for a cluster. This error indicates an action would exceed this limit.

If you're confident your changes won't destabilize the cluster, you can temporarily increase the limit using the cluster update settings API and retry the action.

```
PUT _cluster/settings
```

```
{
 "persistent" : {
  "cluster.max_shards_per_node": 1200
 }
}
```

This increase should only be temporary. As a long-term solution, we recommend you add nodes to the oversharded data tier or reduce your cluster's shard count. To get a cluster's current shard count after making changes, use the cluster stats API.

```
GET _cluster/stats?filter_path=indices.shards.total
```

When a long-term solution is in place, we recommend you reset the `cluster.max_shards_per_node` limit.

```
PUT _cluster/settings
```

```
{
 "persistent" : {
  "cluster.max_shards_per_node": null
 }
}
```

See this fixing "max shards open" video for an example troubleshooting walkthrough. For more information, see Troubleshooting shards capacity.

Number of documents in the shard cannot exceed [2147483519]

Each Elasticsearch shard is a separate Lucene index, so it shares Lucene's `MAX_DOC` limit of having at most 2,147,483,519 (`(2^31)-129`) documents. This per-shard limit applies to the sum of `docs.count` plus `docs.deleted` as reported by the Index stats API. Exceeding this limit will result in errors like the following:

```
Elasticsearch exception [type=illegal_argument_exception, reason=Number of documents in the sha
```

Tip

This calculation may differ from the Count API's calculation, because the Count API does not include nested documents and does not count deleted documents.

This limit is much higher than the recommended maximum document count of approximately 200M documents per shard.

If you encounter this problem, try to mitigate it by using the Force Merge API to merge away some deleted docs. For example:

```
POST my-index-000001/_forcemerge?only_expunge_deletes=true
```

This will launch an asynchronous task which can be monitored via the Task Management API.

It may also be helpful to delete unneeded documents, or to split or reindex the index into one with a larger number of shards.