Size your shards

IMPORTANT: This documentation is no longer updated. Refer to Elastic's version policy and the latest documentation.

To protect against hardware failure and increase capacity, Elasticsearch stores copies of an index's data across multiple shards on multiple nodes. The number and size of these shards can have a significant impact on your cluster's health. One common problem is *oversharding*, a situation in which a cluster with a large number of shards becomes unstable.

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 following sections provide some reminders and guidelines you should consider when designing your sharding strategy. If your cluster has shard-related problems, see Fix an oversharded cluster.

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 shard has overhead

Every shard uses 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 segment metadata in JVM 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.

Best practices

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

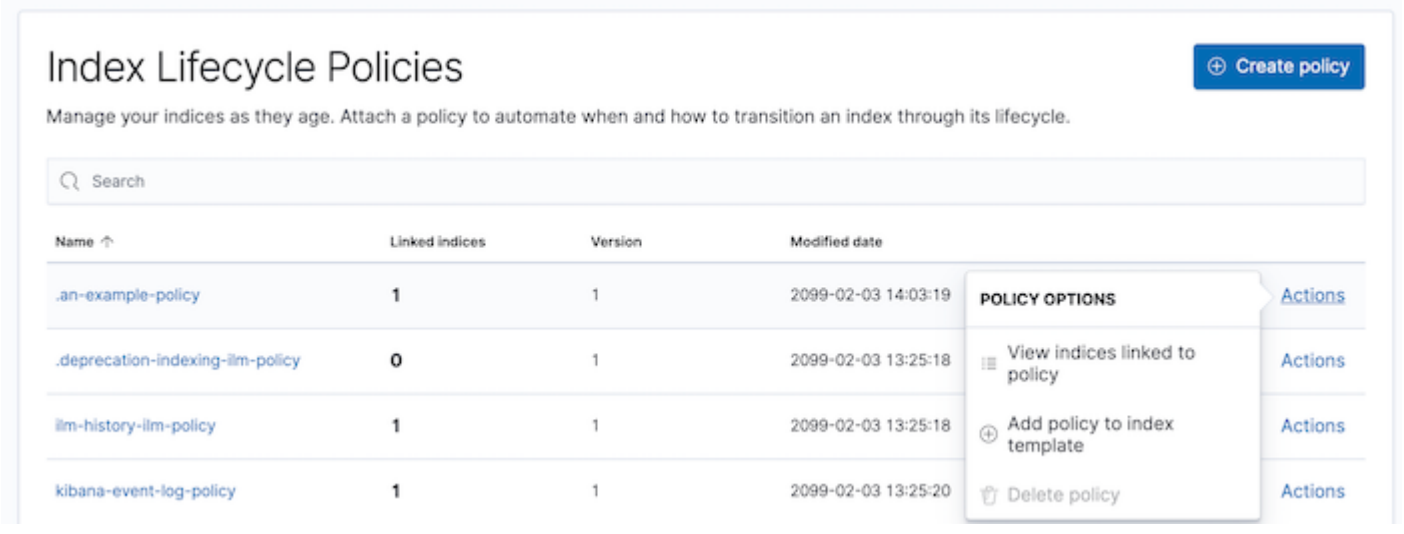
Delete indices, not documents

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?**
  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 shard sizes between 10GB and 50GB

Large shards may make a cluster less likely to recover from failure. When a node fails, Elasticsearch rebalances the node's shards across the data tier's remaining nodes. Large shards can be harder to move across a network and may tax node resources.

While not a hard limit, shards between 10GB and 50GB tend to work well. You may be able to use larger shards depending on your network and use case.

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

Copy as curl Try in Elastic

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
...
```

Aim for 20 shards or fewer per GB of heap memory

The number of shards a node can hold is proportional to the node's heap memory. For example, a node with 30GB of heap memory should have at most 600 shards. The further below this limit you can keep your nodes, the better. If you find your nodes exceeding more than 20 shards per GB, consider adding another node.

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

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

Copy as curl Try in Elastic

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

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

Copy as curl Try in Elastic

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
  }
}
```

Copy as curl Try in Elastic

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
```

Copy as curl Try in Elastic

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-*
```

Copy as curl Try in Elastic

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
```

Copy as curl Try in Elastic

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.

```
POST my-index-000001/_shrink/my-shrunken-index-000001
```

Copy as curl Try in Elastic

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

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"
  }
}
```

Copy as curl Try in Elastic

« Fix common cluster issues

Use Elasticsearch for time series data »