Size your shards 调整分片大小

ECE ECK ELASTIC CLOUD HOSTED SELF MANAGED

ECE ECK ELASTIC CLOUD 托管 自我管理

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.

分片是 Elasticsearch 中的基本存储单位。每个索引都分为一个或多个分片，以帮助在集群中的节点之间分配数据和工作负载。这种划分使 Elasticsearch 能够处理大型数据集并高效执行搜索和索引等。有关分片的更多详细信息，请参阅节点和分片 。

General guidelines 一般准则

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

平衡分片的数量和大小对于 Elasticsearch 集群的性能和稳定性非常重要：

- Too many shards can degrade search performance and make the cluster unstable. This is referred to as *oversharding*.

  分片过多会降低搜索性能并使集群不稳定。这称为*过度分片* 。

- Very large shards can slow down search operations and prolong recovery times after failures.

  非常大的分片可能会减慢搜索作速度并延长失败后的恢复时间。

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

要避免上述任一状态，请遵循以下准则：

General sizing guidelines
一般大小调整准则

- Aim for shard sizes between 10GB and 50GB

  分区大小尽量在 10GB 到 50GB 之间

- Keep the number of documents on each shard below 200 million

  将每个分片上的文档数控制在 2 亿以下

Shard distribution guidelines
分片分配准则

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.

为确保每个节点都以最佳方式工作，请在节点之间均匀分配分片。分布不均匀会导致某些节点比其他节点工作更努力，从而导致性能下降和不稳定。

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.

虽然 Elasticsearch 会自动平衡分片，但您需要为索引配置适当数量的分片和副本，以实现节点之间的均匀分布。

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

如果您使用的是数据流，则每个数据流都由一系列索引提供支持，每个索引可能具有多个分片。

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

尽管有这些一般准则，但最好制定一个量身定制的分片策略，以考虑您的特定基础设施、用例和性能预期。

Create a sharding strategy
创建分片策略

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.

防止过度分片和其他分片相关问题的最佳方法是创建分片策略。分片策略可帮助您确定和维护集群的最佳分片数量，同时限制这些分片的大小。

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.

遗憾的是，没有万能的分片策略。在一个环境中有效的策略可能无法在另一个环境中扩展。一个好的分片策略必须考虑您的基础设施、使用案例和性能预期。

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.

创建分片策略的最佳方法是使用您在生产中看到的相同查询和索引负载，在生产硬件上对生产数据进行基准测试。有关我们推荐的方法，请观看定量集群大小调整视频 。在测试不同的分片配置时，请使用 Kibana 的 Elasticsearch 监控工具来跟踪集群的稳定性和性能。

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.

Elasticsearch 节点的性能通常受到底层存储性能的限制。查看我们的建议，以优化您的存储以进行索引和搜索 。

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.

以下部分提供了您在设计分片策略时应考虑的一些提醒和准则。如果您的集群已过度分片，请参阅减少集群的分片计数 。

Sizing considerations 大小调整注意事项

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

在构建分片策略时，请记住以下事项。

Searches run on a single thread per shard
搜索在每个分片的单个线程上运行

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.

大多数搜索都会找到多个分片。每个分片在单个 CPU 线程上运行搜索。虽然一个分片可以运行多个并发搜索，但跨大量分片进行搜索可能会耗尽节点的搜索线程池 。这可能会导致吞吐量低和搜索速度慢。

Each index, shard, segment and field has overhead
每个索引、分片、分段和字段都有开销

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.

每个索引和每个分片都需要一些内存和 CPU 资源。在大多数情况下，一小部分大分片使用的资源比许多小分片少。

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.

分段在分片的资源使用中起着重要作用。大多数分片包含多个分段，用于存储其索引数据。Elasticsearch 将一些 segment 元数据保存在堆内存中，以便可以快速检索以进行搜索。随着分片的增长，其分段会合并为更少、更大的分段。这减少了 Segment 的数量，这意味着保存在堆内存中的元数据更少。

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.

每个映射的字段在内存使用和磁盘空间方面也会带来一些开销。默认情况下，Elasticsearch 会自动为其索引的每个文档中的每个字段创建一个映射，但您可以关闭此行为以控制您的映射 。

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.

此外，每个 segment 都需要每个 mapped 字段的少量堆内存。此每个字段每个分段的堆开销包括字段名称的副本，使用 ISO-8859-1 （如果适用）或 UTF-16 编码。通常这并不明显，但如果您的分片具有较高的分段计数，并且相应的映射包含较高的字段计数和/或非常长的字段名称，则可能需要考虑此开销。

Elasticsearch automatically balances shards within a data tier
Elasticsearch 会自动平衡数据层内的分片

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.

集群的节点分为数据层 。在每个层中，Elasticsearch 会尝试将索引的分片分布到尽可能多的节点。当您添加新节点或节点失败时，Elasticsearch 会自动在该层的其余节点之间重新平衡索引的分片。

Best practices 最佳实践

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

在适用的情况下，使用以下最佳实践作为分片策略的起点。

Delete indices, not documents
删除索引，而不是文档

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.

已删除的文档不会立即从 Elasticsearch 的文件系统中删除。相反，Elasticsearch 会在每个相关分片上将文档标记为已删除。标记的文档将继续使用资源，直到在定期分段合并期间将其删除。

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

如果可能，请改为删除整个索引。Elasticsearch 可以立即直接从文件系统中删除已删除的索引并释放资源。

Use data streams and ILM for time series data
将 Data Streams 和 ILM 用于时间序列数据

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.

Data Streams 允许您跨多个基于时间的支持索引存储时间序列数据。您可以使用索引生命周期管理 （ILM） 自动管理这些支持索引。

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.

此设置的一个优点是自动翻转，当当前写入索引满足定义的 `max_primary_shard_size`、`max_age`、`max_docs` 或 max_size 阈值时，它会创建一个新的写入索引。当不再需要某个索引时，您可以使用 ILM 自动删除它并释放资源。

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

ILM 还使您能够轻松地随时间更改分片策略：

- **Want to decrease the shard count for new indices?**

  想要减少新索引的分片计数?

  Change the `index.number_of_shards` setting in the data stream's matching index template.

  更改数据流的匹配索引模板中的 `index.number_of_shards` 设置。

- **Want larger shards or fewer backing indices?**

  想要更大的分片或更少的支持索引?

  Increase your ILM policy's rollover threshold.

  提高 ILM 策略的滚动更新阈值 。

- **Need indices that span shorter intervals?**

  需要跨较短间隔的索引?

  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.

  通过更快地删除较旧的索引来抵消增加的分片计数。您可以通过降低策略删除阶段的 `min_age` 阈值来实现此目的。

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

每个新的支持指数都是进一步调整您的策略的机会。

Aim for shards of up to 200M documents, or with sizes between 10GB and 50GB
目标是最大 200M 文档的分片，或大小在 10GB 到 50GB 之间的分片

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.

每个分片都会产生一些相关的开销，包括集群管理和搜索性能。搜索 1000 个 50MB 分片的成本要比搜索包含相同数据的单个 50GB 分片要贵得多。但是，非常大的分片也可能导致搜索速度变慢，并且在失败后需要更长的时间才能恢复。

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.

分片的物理大小没有硬性限制，理论上每个分片最多可以包含超过 20 亿个文档 。但是，经验表明，只要每个分片的文档计数保持在 2 亿个以下，10GB 到 50GB 之间的分片通常适用于许多使用案例。

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.

根据您的网络和使用案例，您可能能够使用较大的分片，而较小的分片可能适用于某些使用案例。

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.

如果您使用 ILM，请将滚动更新作的 `max_primary_shard_size` 阈值设置为 `50 GB` 以避免大于 50 GB 的分片，`min_primary_shard_size` 阈值设置为 `10 GB` 以避免小于 10 GB 的分片。

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

要查看分片的当前大小，请使用 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.

`pri.store.size` 值显示索引的所有主分片的总大小。

```
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:

如果索引的分片因超过建议的 50GB 大小而导致性能下降，您可以考虑修复索引分片的大小。分片是不可变的，因此它们的大小是固定的，因此必须使用更正的设置来复制索引。这需要首先确保有足够的磁盘来复制数据。之后，您可以通过以下选项之一复制具有更正设置的索引数据：

- running Split Index to increase number of primary shards

  运行 Split Index 以增加主分片的数量

- creating a destination index with corrected settings and then running Reindex

  使用更正的设置创建目标索引，然后运行 Reindex

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

请注意，执行还原快照和/或克隆索引不足以解决分片的大小问题。

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.

将源索引的数据复制到其目标索引后，可以删除源索引。然后，您可以考虑针对源索引名称的目标索引设置 Create Alias（创建别名），以指向该索引以实现连续性。

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

有关故障排除演练示例，请参阅此修复分区大小视频 。

Master-eligible nodes should have at least 1GB of heap per 3000 indices
符合 Master 条件的节点每 3000 个索引应至少有 1GB 的堆

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.

主节点可以管理的索引数量与其堆大小成正比。每个索引所需的确切堆内存量取决于各种因素，例如映射的大小和每个索引的分片数。

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.

作为一般经验法则，主节点上每 GB 堆的索引数应少于 3000 个。例如，如果您的集群具有专用主节点，每个节点具有 4GB 的堆，则索引应少于 12000 个。如果您的主节点不是专用主节点，则适用相同的大小调整指南：对于集群中的每 3000 个索引，您应该在每个符合主节点条件的节点上至少保留 1GB 的堆。

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.

请注意，此规则定义了主节点可以管理的绝对最大索引数，但不保证涉及这么多索引的搜索或索引的性能。您还必须确保您的数据节点有足够的资源来支持您的工作负载，并且您的整体分片策略满足您的所有性能要求。另请参阅 搜索在每个分片的单个线程上运行 和 每个索引、分片、分段和字段都有开销 。

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

要检查每个节点堆的配置大小，请使用 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.

您可以使用 cat shards API 检查每个节点的分片数量。

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

Add enough nodes to stay within the cluster shard limits
添加足够的节点以保持在集群分片限制范围内

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.

集群分片限制可防止每个节点创建超过 1000 个非冻结分片，以及每个专用冻结节点创建 3000 个冻结分片。确保集群中有足够的每种类型的节点来处理所需的分片数量。

Allow enough heap for field mappers and overheads
为字段映射器和开销提供足够的堆

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.

映射字段在每个节点上消耗一些堆内存，并且需要在数据节点上使用额外的堆。确保每个节点都有足够的堆用于映射，并为与其工作负载关联的开销留出额外的空间。以下部分说明如何确定这些堆要求。

Mapping metadata in the cluster state
在集群状态中映射元数据

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.

集群中的每个节点都有集群状态的副本。cluster state （集群状态） 包括有关每个索引的字段映射的信息。此信息具有堆开销。您可以使用集群统计 API 来获取去重和压缩后所有映射总大小的堆开销。

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

This will show you information like in this example output:

这将向您显示如下例输出中的信息：

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

Retrieving heap size and field mapper overheads
检索堆大小和字段映射器开销

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

您可以使用 Nodes stats API 获取每个节点的两个相关指标：

- The size of the heap on each node.

  每个节点上的堆大小。

- 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.

  每个节点的字段的任何其他估计堆开销。这是特定于数据节点的，其中除了上面提到的 cluster state 字段信息外，数据节点持有的索引的每个 mapped 字段都有额外的堆开销。对于不是数据节点的节点，此字段可能为零。

```
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:

对于每个节点，这将向您显示如下例输出中的信息：

```
{
  "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
考虑额外的堆开销

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.

除了上述两个字段开销指标外，您还必须为 Elasticsearch 的基准使用情况以及您的工作负载（如索引、搜索和聚合）留出足够的堆。0.5GB 的额外堆足以处理许多合理的工作负载，如果您的工作负载非常轻，则可能需要更少，而繁重的工作负载可能需要更多。

Example 例

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

例如，考虑上述数据节点的输出。节点的堆至少需要：

- 1 GB for the cluster state field information.

  1 GB 用于集群状态字段信息。

- 1 GB for the additional estimated heap overhead for the fields of the data node.

  1 GB 用于数据节点字段的额外估计堆开销。

- 0.5 GB of extra heap for other overheads.

  0.5 GB 的额外堆用于其他开销。

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

由于示例中节点的最大堆大小为 4GB，因此它足以满足所需的 2.5GB 总堆。

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

如果节点的堆最大大小不足，请考虑避免不必要的字段，或纵向扩展集群，或重新分配索引分片。

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.

请注意，上述规则不一定保证涉及大量索引的搜索或索引的性能。您还必须确保您的数据节点有足够的资源来支持您的工作负载，并且您的整体分片策略满足您的所有性能要求。另请参阅 搜索在每个分片的单个线程上运行 和 每个索引、分片、分段和字段都有开销 。

Avoid node hotspots 避免节点热点

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.

如果为特定节点分配的分片过多，则该节点可能会成为热点。例如，如果单个节点包含的分片过多，而索引量较高，则该节点可能会出现问题。

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.

要防止出现热点，请使用 `index.routing.allocation.total_shards_per_node` index 设置显式限制单个节点上的分片数。您可以使用更新索引设置 API 进行配置 `index.routing.allocation.total_shards_per_node` 。

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

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

Avoid unnecessary mapped fields
避免不必要的映射字段

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.

默认情况下，Elasticsearch 会自动为其索引的每个文档中的每个字段创建一个映射。每个映射的字段都对应于磁盘上的一些数据结构，这些数据结构是在此字段上进行高效搜索、检索和聚合所必需的。有关每个映射字段的详细信息也保存在内存中。在许多情况下，此开销是不必要的，因为字段未用于任何搜索或聚合。使用 *Explicit mapping 而不是 dynamic* mapping 以避免创建从未使用的字段。如果字段集合通常一起使用，请考虑在索引时使用 `copy_to` 来合并它们。如果某个字段很少使用，则最好将其设为 Runtime 字段。

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.

您可以通过 Field usage stats API 获取有关正在使用哪些字段的信息，并且可以使用 Analyze index disk usage API 分析映射字段的磁盘使用情况。但请注意，不必要的映射字段也会带来一些内存开销及其磁盘使用率。

Reduce a cluster's shard count
减少集群的分片计数

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

如果您的集群已经过度分片，您可以使用以下一种或多种方法来减少其分片计数。

Create indices that cover longer time periods
创建涵盖较长时间段的索引

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.

如果您使用 ILM 并且您的保留策略允许，请避免对滚动更新操作使用 `max_age` 阈值。相反，请使用 `max_primary_shard_size` 以避免创建空索引或许多小分片。

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.

如果您的保留策略需要 `max_age` 阈值，请增加该阈值以创建涵盖较长时间间隔的索引。例如，您可以每周或每月创建索引，而不是创建每日索引。

Delete empty or unneeded indices
删除空索引或不需要的索引

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.

如果您使用 ILM 并根据 `max_age` 阈值滚动索引，则可能会无意中创建没有文档的索引。这些空索引没有任何好处，但仍会消耗资源。

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

您可以使用 cat count API 找到这些空索引。

```
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.

获得空索引列表后，您可以使用 delete index API 删除它们。您还可以删除任何其他不需要的索引。

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

Force merge during off-peak hours
在非高峰时段强制合并

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.

如果您不再写入索引，则可以使用强制合并 API 将较小的分段合并为较大的分段。这可以减少分片开销并提高搜索速度。但是，强制合并会占用大量资源。如果可能，请在非高峰时段运行强制合并。

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

Shrink an existing index to fewer shards
将现有索引缩减为更少的分片

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

如果您不再写入索引，则可以使用收缩索引 API 来减少其分片计数。

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

ILM 还对处于热阶段的索引执行收缩作 。

Combine smaller indices 合并较小的索引

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.

您还可以使用 reindex API 将具有类似映射的索引合并到单个大型索引中。对于时间序列数据，您可以将短时间段的索引重新索引为涵盖较长时间段的新索引。例如，您可以使用共享索引模式（如 `my-index-2099.10.11`）将 10 月的每日索引重新索引为每月 `my-index-2099.10` 索引。重新编制索引后，删除较小的索引。

```
POST _reindex

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

Troubleshoot shard-related errors
排查分片相关错误

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

下面介绍了如何解决常见的分片相关错误。

this action would add [x] total shards, but this cluster currently has [y]/[z] maximum shards open;
此作将添加 [x] 个分片，但此集群当前打开的最大分片数为 [y]/[z];

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.

`cluster.max_shards_per_node` 集群设置限制集群的最大打开分片数。此错误表示作将超过此限制。

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.

如果您确信您的更改不会破坏集群的稳定性，则可以使用集群更新设置 API 临时增加限制，然后重试该作。

```
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.

这种增加应该只是暂时的。作为长期解决方案，我们建议您将节点添加到过度分片数据层或减少集群的分片计数 。要在进行更改后获取集群的当前分片计数，请使用集群统计 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.

当长期解决方案到位时，我们建议您重置 `cluster.max_shards_per_node` 限制。

```
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.

有关故障排除演练示例，请参阅此修复"最大打开分片数"视频 。有关更多信息，请参阅分区容量故障排除 。

Number of documents in the shard cannot exceed [2147483519]
分片中的文档数不能超过 [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:

每个 Elasticsearch 分片都是一个单独的 Lucene 索引，因此它与 Lucene 的 `MAX_DOC` 限制相同，即最多包含 2,147,483,519 （ `(2^31) -129` ） 个文档。此每个分片的限制适用于索引统计信息 API 报告的 `docs.count` 加上 `docs.deleted` 的总和。超过此限制将导致如下错误：

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

Tip 提示

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.

此计算可能与 Count API 的计算不同，因为 Count API 不包括嵌套文档，也不对已删除的文档进行计数。

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

此限制远高于建议的最大文档计数 （每个分片约 200M 个文档）。

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

如果你遇到这个问题，可以尝试使用 强制合并 API 来合并一些已删除的文档来缓解这个问题。例如：

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

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

这将启动一个异步任务，该任务可以通过 Task Management API 进行监控。

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

删除不需要的文档，或者将索引拆分或重新索引为具有更多分片的索引也可能有所帮助。