

Common Elasticsearch Errors and How to Solve Them - Part 1

Jochen Kressin · 2024-04-24



As a DevOps engineer, Elasticsearch errors can be some of the most frustrating issues you'll face. These errors can slow down the performance of your server, lead to data loss, and negatively impact user experience. To avoid these complications, it's important to know the top ten most common Elasticsearch errors, what they indicate, and how to solve them. In this blog post series, we'll cover each error in great detail and provide a step-by-step guide to troubleshooting it. Let's dive in!

MapperParsingException

The `MapperParsingException` in Elasticsearch typically arises when there's an issue with how data is formatted or structured before it's indexed. This exception is commonly triggered during the indexing process when Elasticsearch encounters document fields that don't conform to the defined mapping.

Source of the Issue:

Inconsistent Data Types: The most common cause is a mismatch in field data types. For example, if your mapping expects a date or integer but receives a string or a different format, Elasticsearch will throw this exception.

Incorrect JSON Format: Another source could be improperly formatted JSON in the indexing request. Elasticsearch expects well-formed JSON, and any deviation can cause parsing issues.

Dynamic Mapping Issues: Elasticsearch automatically creates field mappings in certain cases. If incoming data doesn't align with these automatic mappings, it can result in a `MapperParsingException`.

Nested or Complex Data Structures: Sometimes, the error is due to the way nested or complex data structures are handled, especially if they don't align with the predefined mappings.

Remedies

Review and Align Data Types: Ensure that the data you're trying to index matches the field types defined in your mappings. If you're indexing dynamic data, consider implementing stricter mappings to prevent type conflicts.

Validate JSON Format: Before indexing, validate your JSON payloads to ensure they're correctly structured. Tools like JSONLint can be helpful for this.

Adjust Mapping Settings: If dynamic mappings are causing issues, you might need to explicitly define mappings for your fields. This gives you control over how Elasticsearch interprets each field's data type.

Handle Nested Fields Properly: For nested or complex data, ensure your mappings correctly define these structures. Using the correct types (like nested for nested objects) is crucial.

Limit of Total Fields [1000] in Index

The error message "Limit of total fields [1000] in index" in Elasticsearch indicates that the index has reached the maximum number of fields allowed. Elasticsearch imposes this limit to prevent mappings explosions, which can occur when too many fields are created, potentially leading to performance issues or out-of-memory errors. Here's a detailed explanation and remedies:

Source of the Issue

Too Many Fields in Mapping: This error occurs when an index's mapping exceeds the default limit of 1000 fields. This can happen in dynamically mapped indices where new fields are automatically created based on the indexed data.

Dynamic Mapping with Varied Data: If you're indexing documents with a high variety of fields or with inconsistent structures, Elasticsearch can generate a large number of fields dynamically, leading to this error.

Nested or Complex Data Structures: Indices containing documents with deeply nested structures or a large number of different object fields can easily hit this limit.

Remedies

Increase the Fields Limit: If the large number of fields is justified by your use case, you can increase the limit by updating the `index.mapping.total_fields.limit` setting. However, do this cautiously, as a very high number of fields can lead to performance issues.

Example:

```
PUT /your_index/_settings
{
  "index.mapping.total_fields.limit": 2000
}
```

Review and Optimize Your Mapping: Evaluate your data model and mapping. Consolidate fields where possible, and ensure that you're not unintentionally creating unnecessary fields, especially in cases of dynamic mapping.

Disable or Control Dynamic Mapping: Consider disabling dynamic mapping and explicitly defining all fields in your mappings. This approach requires a better understanding of your data schema but provides more control.

Normalize Your Data: If your documents have a lot of variable fields, it may be worth normalizing your data structure. This can mean splitting the data into multiple indices or reorganizing the data to reduce the variety of fields.

Clean Up Unused Fields: Over time, indices can accumulate fields that are no longer used. Identifying and removing these can help stay within the fields limit.

ClusterBlockException

A "ClusterBlockException" in Elasticsearch is an error that indicates your request has been blocked due to certain conditions on the cluster or the index that prevent the operation from being executed. Understanding the cause of this exception is key to resolving it.

Source of the Issue

Read-Only Blocks: If a cluster or an index is set to read-only, write operations will be blocked. This often happens automatically when the disk space on a node falls below a certain threshold, as a protective measure.

Cluster Health Issues: If the cluster is in a red state, certain operations might be blocked. This can happen if primary shards are not allocated or there is a significant issue with the cluster.

Initialization or Recovery: During initial startup, recovery, or when shards are being relocated, certain operations may be blocked to ensure cluster stability.

Remedies

Check Disk Space: Ensure there is sufficient disk space on all nodes. Elasticsearch automatically sets indices to read-only when disk space is low (usually below 85%). Free up disk space or add more storage.

Reset Read-Only Blocks: To remove the read-only block from an index, use the following API call:

```
PUT /your_index/_settings
{
  "index.blocks.read_only_allow_delete": null
}
```

For the entire cluster:

```
PUT /_all/_settings
{
  "index.blocks.read_only_allow_delete": null
}
```

Check Cluster Health: Use the `_cluster/health` API to check the health of your cluster. If it's red, investigate and resolve the underlying issue (like unassigned shards).

Review Cluster and Index Settings: Look for any manual blocks that might have been set on the cluster or index level. These can be removed via the appropriate settings in the Elasticsearch API.

Manage Shard Allocation: If the issue is due to shard allocation or relocation, you may need to wait for this process to complete, or manually intervene if there's a configuration issue.

UnavailableShardsException

The "UnavailableShardsException" in Elasticsearch is an error that occurs when Elasticsearch cannot access one or more shards needed to complete an operation. This issue can impact both search and indexing operations and is crucial to address for the health and performance of your Elasticsearch cluster.

Source of the Issue

Shards Not Allocated: The primary reason for this exception is that some shards, either primary or replica, are not allocated to any node in the cluster. This can happen due to various reasons like node failures, network issues, or configuration problems.

Cluster Health Problems: A cluster in a yellow or red state often indicates issues with shard allocation. A red state means at least one primary shard (and its replicas) is not allocated.

Resource Constraints: Sometimes, resource constraints like insufficient disk space, memory, or CPU can prevent shards from being allocated or remaining available.

Configuration Issues: Misconfiguration in shard allocation settings or cluster settings can lead to shard allocation failures.

Remedies

Check Cluster Health: Use the `_cluster/health` API to check the health of your cluster. A red or yellow status requires immediate attention. Look for unassigned shards using the `_cat/shards` API and identify why they are unassigned.

Allocate Missing Shards: If shards are unassigned due to node failure, consider adding new nodes or restarting failed nodes. Use the cluster allocation explain API (`_cluster/allocation/explain`) to understand why shards are not allocated and take corrective actions.

Address Resource Issues: Ensure that there is enough disk space on the nodes. Elasticsearch can block writing to indices on nodes with low disk space. Scale your resources (CPU, RAM) if they are insufficient, particularly in clusters with heavy search or indexing loads.

Adjust Index Settings: Modify index settings to reduce the number of shards if over-sharding is an issue. For temporary relief, you can also consider changing the `index.unassigned.nodeleft.delayedtimeout` setting to give more time for a node to rejoin before reallocating its shards.

Where to go next

- Read the [next article in this series](#)
- Ask any question you have on our [Search Guard Forum](#)
- Want to learn more about a specific error message? [Just drop us a note!](#)

This article was published on the Search Guard blog: <https://search-guard.com/blog/common-elasticsearch-errors-and-how-to-solve-them-part-1/>

Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · [Start a free trial](#)