How Elasticsearch Checks for Document Existence with Custom IDs

Jochen Kressin · 2026-07-21

When you index a document in Elasticsearch with a custom `_id`, the system needs to determine whether that document already exists. If it does, Elasticsearch performs an update; if not, it creates a new document. But how exactly does this existence check work under the hood? Is Elasticsearch scanning the entire index every time you send a document with your own ID?

The short answer: no, it's far more efficient than that. Let's dig into the details.

The Concern

If you're running a high-throughput ingestion pipeline with application-generated IDs, you might worry about performance implications. After all, checking whether a document exists before deciding to create or update it sounds expensive. Does Elasticsearch need to search through millions of documents every time?

How the Existence Check Actually Works

Elasticsearch's approach is elegant and efficient. It combines shard routing with a two-tier lookup so that it never scans the whole index.

Step 1: Shard Routing

When you index a document with a custom `_id`, Elasticsearch first determines which shard should hold that document. This is done by hashing either:

- The document's `_id` value, or
- A custom routing key if you've specified one

The hash determines exactly one primary shard (and its replicas) where the document belongs. This means Elasticsearch already knows precisely where to look, it doesn't need to check every shard in the index.

Step 2: A Two-Tier Lookup by ID

Here's where it gets interesting. Even within the target shard, Elasticsearch doesn't perform anything resembling a "full scan." Instead, it resolves the ID in two tiers.

Tier 1: The in-memory version map. Elasticsearch keeps a small in-memory structure—the *LiveVersionMap*—that maps the IDs written since the last refresh to their current version and sequence number. Documents that were just indexed aren't yet searchable in a Lucene segment, so this map is what lets Elasticsearch resolve them in real time. A lookup here is essentially a hash-map hit (constant time), and for high-throughput upserts—where you keep touching recently-written IDs—this is the real fast path. The map is pruned on every refresh, so it only holds recent activity.

Tier 2: A per-segment lookup in Lucene. If the ID isn't in the version map, Elasticsearch looks it up in Lucene. This is where the terms dictionary comes in: Lucene maintains, for each field (including `_id`), a sorted terms structure that is indexed in memory by a finite state transducer (FST) over an on-disk block-tree dictionary. You can think of it loosely like a sorted tree—the important point is that a single lookup is sub-linear, not a scan.

There's a detail the "one lookup" picture misses, though: a shard is made up of many Lucene *segments*, each with its own terms dictionary. So the ID lookup is performed per segment (newest first, accounting for deletes), and the real cost scales with the number of segments as much as with the number of documents. Background merges, which reduce segment count, are what keep this cheap over time.

As a rough intuition for a single segment, the per-lookup cost grows sub-linearly rather than linearly:

- ~1,000 documents → on the order of ~10 comparisons
- ~1,000,000 documents → on the order of ~20 comparisons
- ~1,000,000,000 documents → on the order of ~30 comparisons

Treat those numbers as a mental model for "grows very slowly," not a literal description of how the FST is traversed. The cost grows incredibly slowly as your data grows.

The Performance Trade-off

While the existence check is efficient, using custom IDs isn't entirely "free." There are some performance considerations to keep in mind:

Auto-generated IDs Are Faster

When you let Elasticsearch generate the `_id` automatically, it can skip the version lookup entirely on the normal path. Because it just generated the ID, it knows the document is new and can index it as a plain insert rather than an "update if it exists." This makes auto-generated IDs more efficient for pure append-only workloads.

One caveat: this optimization has to stay safe under retries. If the client loses its connection to the primary and resends the request, the same auto-generated document could arrive twice. Elasticsearch detects that situation and falls back to a checked insert to avoid creating a duplicate—so the "no lookup" fast path applies to the common case, not to literally every operation.

Shard Size Matters

As a shard accumulates more data—and, more to the point, more segments—the Lucene tier of the existence check gradually does more work. You might notice a pattern in high-throughput scenarios:

1. **New shards**: few segments, minimal lookup cost
2. **Growing shards**: more segments and documents, gradually increasing overhead
3. **Full shards**: the most accumulated segments, just before rollover
4. **After rollover**: writes move to a fresh, small shard and the pattern resets

The underlying driver is segment count and merge activity rather than raw document count alone, which is why background merges and sensible shard sizing matter for write-heavy workloads. This is one reason proper index lifecycle management (ILM) and rollover strategies help.

When Custom IDs Make Sense

Despite the overhead, custom IDs are the right choice in many scenarios:

- **Idempotent ingestion**: When you need to safely retry failed indexing operations without creating duplicates
- **Known entity updates**: When your application needs to update specific documents by a business identifier
- **Cross-system consistency**: When the document ID needs to match an identifier from another system
- **Upsert patterns**: When you're frequently updating existing documents rather than just appending new ones

Practical Example

Let's trace through what happens when you index a document:

```
PUT /my-index/_doc/user-12345
{
  "name": "Alice",
  "email": "alice@example.com"
}
```

1. **Hash calculation**: Elasticsearch hashes `user-12345` to determine the target shard (let's say shard 2)
2. **Request routing**: The request is routed to the node holding shard 2's primary
3. **ID resolution**: The primary first checks its in-memory version map for `user-12345`; if it isn't there, it does a per-segment lookup in Lucene's `_id` terms dictionary
4. **Decision**:
   - If found → perform an update (increment version, merge fields)

- If not found → create a new document

5. **Replication**: The operation is replicated to replica shards

The entire existence check happens in step 3 and is extremely fast.

Optimizing Custom ID Performance

If you're using custom IDs and want to maximize performance:

Use Bulk Operations

Batching many documents into a single bulk request amortizes the per-request overhead—network round-trips, request parsing, and coordination—across many documents:

```
POST /_bulk
{"index": {"_index": "my-index", "_id": "user-12345"}}
{"name": "Alice"}
{"index": {"_index": "my-index", "_id": "user-67890"}}
{"name": "Bob"}
```

Note that bulk doesn't make the existence check itself cheaper: each document in the batch is still resolved individually. The win is in everything around the check, which is substantial at high volume.

Consider Your Shard Strategy

Smaller, more numerous shards mean smaller terms dictionaries and fewer documents per segment. However, don't go overboard—too many shards create their own overhead. The sweet spot depends on your specific workload.

Use Routing When Appropriate

If your documents naturally group by some attribute (like tenant ID or user ID), custom routing can improve both write and read performance by co-locating related documents.

Takeaways

1. **No full index scan**: Elasticsearch uses shard routing, an in-memory version map, and per-segment Lucene lookups to check document existence—never a full scan.
2. **Sub-linear complexity**: The Lucene lookup scales incredibly well, and recently-written IDs are resolved from the in-memory version map in near-constant time. Doubling your data barely increases lookup time.
3. **Auto-generated IDs are faster**: If you don't need custom IDs, letting Elasticsearch generate them avoids the version lookup on the normal path (retries are the exception).
4. **Segment count affects performance**: More segments—which come with larger, older shards—mean slightly slower ID lookups. Background merges and sensible shard sizing, via ILM and rollover, keep this in check.

5. **Custom IDs are still efficient**: Don't avoid custom IDs out of performance fear—the overhead is minimal for most workloads. Use them when they make architectural sense.

Understanding these internals helps you make informed decisions about your indexing strategy. Whether you choose auto-generated or custom IDs, Elasticsearch is designed to handle both patterns efficiently at scale.

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-custom-id-existence-check-mechanism/