Elasticsearch 8 to 9 Snapshot Restore Fails? The 7.x-Origin Index Trap and the Fix That Worked

Daniel Gleim · 2026-06-17

Elasticsearch 8 to 9 Snapshot Restore Fails? The 7.x-Origin Index Trap and the Fix That Worked

Upgrading Elasticsearch clusters often involves restoring snapshots from older versions to newer ones. While this process is generally straightforward, version compatibility issues can sometimes throw unexpected curveballs. If you have ever encountered a snapshot restore failure despite following all the documented steps, you are not alone. This scenario highlights a subtle but important aspect of Elasticsearch's cross-version compatibility that caught one user off guard during their upgrade from 8.7.1 to 9.0.1.

Problem Description

A user was attempting to restore a snapshot from Elasticsearch 8.7.1 to a new 9.0.1 cluster. They had followed the standard procedure by setting `index.blocks.write: true` on their source indices in 8.7.1 before taking the snapshot. However, when they tried to restore the snapshot to their 9.0.1 cluster, the operation failed with an error indicating that the index could not be upgraded because of its creation version.

A typical message in such cases looks like this:

```
cannot restore index [my-index] because it cannot be upgraded
```

The reason usually explains that the index was originally created on an older major version and that the newer cluster cannot upgrade it in place. This error was particularly puzzling in the real-world case

because the user had confirmed that no indices with the same names existed in the target 9.0.1 cluster and that the index in question had originally been created on a 7.x cluster and then upgraded into 8.7.1 over time.

Root Cause Analysis

The issue in this case was not that Elasticsearch 9.0.1 cannot restore 8.x snapshots in general, but that some indices inside the snapshot did not meet 9.x's stricter compatibility requirements, especially indices originally created in 7.x.

Elasticsearch snapshot compatibility depends on the creation version of each index stored in the snapshot, not only on the cluster version that created the snapshot. When you restore a snapshot, the target cluster must be able to read the on-disk format of every index it contains.

- Elasticsearch 9.x can read indices created in 8.x and, by design, also indices created in 7.x that have been prepared correctly.
- If a snapshot contains indices created in an older, incompatible major version, for example 6.x when upgrading to 8.x or 9.x, those indices cannot be opened on the newer cluster at all and must be reindexed on a compatible cluster or left on an older deployment.

For indices created in 7.x and later upgraded into 8.x, there currently appears to be a usability issue in how 9.x handles them during restore if they are not marked read-only in the way 9.x expects. In the real-world case this post is based on, simply setting `index.blocks.write: true` via `_settings` on 8.7.1 was not enough; 9.0.1 still refused to restore the index. Using the dedicated add index block API before taking a fresh snapshot turned out to be the crucial step:

```
PUT /my-index/_block/write
```

This API is different from simply updating the setting. It ensures that the write block is applied consistently to all shards and that the cluster properly accounts for in-flight writes and block-related metadata.

In the real-world case, the user:

- Had a 7.x-created index running on 8.7.1.
- Manually set `index.blocks.write: true` via index settings.
- Took a snapshot and tried to restore it on 9.0.1, which failed with a compatibility error.
- Then removed the manual `index.blocks` settings, applied `PUT /my-index/_block/write` on 8.7.1, created a new snapshot, and was able to restore that snapshot successfully into 9.0.1.

An Elastic engineer reviewing this behaviour described it as a usability bug and noted that, for now, you need to use the put index block API on a sufficiently recent 8.x before taking the snapshot so that 9.x accepts the index. This is not currently documented as a general rule for all upgrades, but it has been observed to solve the problem in at least this concrete case.

The practical takeaway is that, with the current 9.x behaviour, using the dedicated index block API on the 8.x side is a useful troubleshooting step for 7.x-origin indices that refuse to restore from snapshots, even

when they were already marked as write-blocked via `_settings`. It looks more like a usability issue or implementation quirk in how 9.x checks legacy indices during restore than a formally documented, permanent requirement.

A note on the “index already exists” error

There is another snapshot-restore error that often confuses people:

```
cannot restore index [my-index] because an open index with same name already exists in the cluster. Either close or delete the existing index or restore the index under a different name by providing a rename pattern and replacement name
```

This message is still literally about name conflicts. Elasticsearch refuses to restore an index if there is already an open index or data stream, including system indices, with that name in the target cluster. Typical examples include internal indices such as `.geoip_databases` or `.ds-*-history-*` that already exist in the new cluster.

In other words, this particular error is resolved by closing, deleting, or renaming the existing index or by restoring the snapshot index under a different name, not by changing compatibility settings. It is a separate class of problem that you might encounter during snapshot restore alongside compatibility issues like the 7.x-origin case described in this post.

Solution

The solution does not necessarily require an entirely separate 8.x hop cluster just to sit between 8.7.1 and 9.0.1. Instead, it relies on two main ideas:

1. Being on a supported 8.x version when taking the snapshot, ideally the latest minor of the previous major before going to 9.x.
2. Using the dedicated add index block API to set `index.blocks.write` correctly before snapshotting when you need to mark 7.x-origin indices as read-only.

Here is the step-by-step approach that worked in the real-world scenario and how it relates to the official guidance.

Step 1: Upgrade the source cluster to a suitable 8.x

Before taking the snapshot you want to restore into 9.0.1, make sure your source cluster is on an 8.x version that is part of the supported upgrade path to 9.x, and that can read all the indices you care about. For in-place 7.17 to 9.x upgrades, Elastic’s official guidance is to go via 8.19.latest, run the Upgrade Assistant there to reindex or mark older indices read-only, and only then move to 9.x.

In the snapshot-based migration described here, the snapshot was taken from 8.7.1 and, after applying the write-block workaround, successfully restored into 9.0.1. This shows that earlier 8.x versions can work in practice, but 8.7.1 is not the officially recommended baseline for upgrading to 9.x.

This still aligns with Elastic’s general guidance to use the latest minor of the previous major when preparing for a major version upgrade, even if you ultimately migrate via snapshot and restore rather than an in-place version bump.

Step 2: Clean up old index block settings (if necessary)

If you previously experimented with manually setting `index.blocks.write` via `_settings`, it can be a good idea to reset those settings so you start from a clean state. In the original case, the user removed the existing `index.blocks` object via the Kibana UI before reapplying the block with the new API.

Step 3: Add a write block using the dedicated API

Now, on the upgraded 8.x cluster, use the add index block API instead of a raw settings update:

```
PUT /my-index/_block/write
```

A successful response looks roughly like this:

```
{
  "acknowledged": true,
  "shards_acknowledged": true,
  "indices": [
    {
      "name": "my-index",
      "blocked": true
    }
  ]
}
```

This API:

- Sets `index.blocks.write` for the index.
- Ensures all shards have finished in-flight writes and are correctly accounting for the block.
- Produces the state that 9.x has been shown to handle more reliably when accepting older (for example 7.x-created) indices from a snapshot in this particular reported case.

Step 4: Create a new snapshot on 8.x

With the write block in place, take a new snapshot on your 8.x cluster:

```
PUT /_snapshot/my_repository/my_migration_snapshot
{
  "indices": "my-index",
  "ignore_unavailable": false,
  "include_global_state": false
}
```

This follows the standard Snapshot API and limits the snapshot to the index you care about. As always, make sure the snapshot completes successfully and is marked as such in the repository.

Step 5: Restore the snapshot into 9.0.1

Finally, restore this new snapshot into your 9.0.1 cluster:

```
POST /_snapshot/my_repository/my_migration_snapshot/_restore
{
  "indices": "my-index",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

In the forum case, this sequence (upgrade to a suitable 8.x, apply `PUT /my-index/_block/write`, snapshot, restore) allowed the 7.x-origin index from the 8.x snapshot to be restored into Elasticsearch 9.0.1 without any reindexing.

The Restore Snapshot API itself ignores index blocks during the actual restore operation. It only requires that the cluster's global metadata be writable and that at least one index is selected for restore. The working theory, supported by 9.x upgrade error messages and the behaviour observed here, is that 9.x uses `index.blocks.write` as a compatibility signal for 7.x-origin indices and that the add index block API produces a state that passes this check more reliably than a plain settings update.

Alternative Approach: Reindexing

If you cannot or do not want to rely on snapshot restore for some indices, especially if they were created in versions that are too old for 9.x to read, you can use the reindex API as an alternative migration strategy.

A typical pattern looks like this:

1. Configure the 9.x cluster to treat the 8.x cluster as a remote cluster.
2. Use `POST /_reindex` with a `remote` source to pull data from the old cluster into new indices created natively on 9.x.
3. Optionally, adjust mappings, analyzers, or templates as you go, and then switch clients over to the new indices when ready.

This approach works well when:

- You need full control over the new index layout.
- Some indices in your old snapshots are fundamentally incompatible, for example created in 6.x or earlier and not readable by 9.x.
- You prefer an incremental migration with the ability to validate and backfill data gradually.

Takeaways

This scenario teaches several important lessons about Elasticsearch upgrades and snapshot management.

Understand index-level compatibility, not just cluster versions: Snapshot restores depend on the creation version of each index inside the snapshot. Elasticsearch 9.x can read indices created in 8.x and, with the right preparation, indices created in 7.x, but much older indices must usually be reindexed or left on older clusters.

Use the right tools to mark legacy indices read-only: When preparing 7.x-origin indices for migration to 9.x, do not rely solely on ad-hoc updates to `index.blocks.write` via `_settings`. Either use the Upgrade Assistant in Kibana, which will highlight old indices and offer to reindex or mark them read-only, or, if you are scripting this yourself, use `PUT /{index}/_block/write` so Elasticsearch properly applies and records the write block across all shards in a way that 9.x has been observed to handle more reliably during snapshot restore.

Follow the official upgrade path: Always consult Elasticsearch's official upgrade documentation before planning major version upgrades. For in-place upgrades from 7.17 to 9.x, this typically means upgrading to at least 8.19.latest, resolving any incompatible or legacy indices with the Upgrade Assistant, and only then moving to 9.x. Even if you ultimately migrate via snapshot restore, testing your indices and snapshots from that recommended 8.x baseline can surface problems earlier.

Test your upgrade path end-to-end: Before performing production upgrades, test the full flow in a non-production environment. Add index blocks, take snapshots, restore into a test 9.x cluster, and verify both user indices and system indices. This helps surface compatibility issues and name-conflict problems before they impact production.

Consider reindexing when compatibility stops you: If your snapshots contain indices whose creation versions are too old, or if you want to clean up mappings and settings, reindexing into new indices on 9.x can be safer and more flexible than trying to force a restore.

The key takeaway is that while Elasticsearch's snapshot and restore functionality is robust, major version upgrades require careful planning around index creation versions and the proper handling of index blocks, especially for indices that originated several major versions ago.

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-snapshot-restore-fails-despite-setting-index-blocks-write-a-version-compatibility-deep-dive/