Upgrading to Elasticsearch 9: Why Your "Read-Only" 7.x Indices Still Block the Boot

Daniel Gleim · 2026-09-03

Upgrading to Elasticsearch 9: Why Your "Read-Only" 7.x Indices Still Block the Boot

Major version upgrades in Elasticsearch come with one strict rule: old indices from before version 8.0 must be dealt with before you cross into 9.x. You can reindex them, delete them, or mark them read-only, and for reindexing or marking them read-only, the Upgrade Assistant walks you through it. It sounds simple, and the docs spell it out clearly.

So it feels frustrating when you follow the steps, set your old indices to read-only, and Elasticsearch still refuses to start. If you have hit this wall during an 8.x-to-9.x upgrade, you are not alone, and the fix is more straightforward than it first appears. Let's walk through what actually blocks the upgrade and how to get moving again.

Problem Description

Imagine you are upgrading a cluster from `8.19.4` to `9.1.4`, following the recommended path.

- You first moved from an older 8.x release up to the latest 8.19.x version.
- You set your indices to read-only, paying special attention to the ones created back in 7.x.
- Your cluster runs in a Docker container, behind an Nginx proxy.

To mark the indices read-only, you ran a write block across everything:

```
PUT /_all/_settings
{
  "settings": {
    "index.blocks.write": true
  }
}
```

You then confirmed the setting stuck:

```
GET /_all/_settings?include_defaults=true
```

The response shows `"index.blocks.write": true` on your indices. Everything looks correct. Yet when the `9.1.4` container starts, it dies on boot with a fatal error:

```
The index [.ds-.logs-deprecation.elasticsearch-default-2022.04.26-000010/CLfSJWgRTzytCO-Dxw-WQQ]
created in version [7.16.2] with current compatibility version [7.16.2]
must be marked as read-only using the setting [index.blocks.write] set to [true]
before upgrading to 9.1.4.
```

This is the confusing part. The error asks for exactly the setting you already applied. At this point it is tempting to blame the environment. Maybe a Docker volume did not persist the change? Maybe the Nginx proxy interfered with the API call? Those are reasonable guesses, but the real cause lies elsewhere.

Root Cause Analysis

The write block you set is not the same thing the upgrade check is looking for.

Elasticsearch tracks two separate things here, and they are easy to confuse because both involve the phrase "read-only":

Concept	What it means
`index.blocks.write`	A normal write block you set through the index settings API. It stops writes to the index, and you can turn it on or off whenever you like. It is fully reversible.
`index.verified_read_only`	A private, internal flag that records an index was deliberately marked read-only in 8.18+ and is allowed into 9.x. The Upgrade Assistant sets this flag for you; it is not a general setting you apply yourself. The boot check looks for this marker, not the plain write block.

Internally, Elasticsearch uses that private `index.verified_read_only` flag to confirm an index was intentionally frozen as read-only before the upgrade. When you set `index.blocks.write: true` by hand through the settings API, you only apply the reversible block. You never create the verified marker that way. The 9.x startup check inspects each legacy index for that marker, does not find it, and stops the node from booting.

Why does Elasticsearch insist on a separate, permanent marker? Because the read-only decision becomes irreversible once you reach 9.x. A plain write block is something you might toggle off later to

© 2026 floragunn GmbH · search-guard.com

write to an index again. If the upgrade accepted that as "good enough," you could carry a 7.x index into 9.x while still intending to make it writable, and that combination is not supported. The verified marker forces a deliberate choice: you confirm up front that the index will stay read-only for good.

One more detail worth clearing up: your Docker and Nginx setup is not the culprit. The setting persisted correctly, and the proxy delivered your API call. The upgrade simply needs a different action than the one you took.

Solution

The simplest and best-supported way to set the verified read-only marker is through the Upgrade Assistant, a feature that ships with Kibana. It inspects your cluster, flags every index that would block the upgrade, and gives you a safe button to resolve each one.

If you run Elasticsearch without Kibana, this is the step that trips people up. Tools like Grafana are excellent for dashboards, but they do not include the Upgrade Assistant. For a major version upgrade, a Kibana instance is worth spinning up, even temporarily, because it is tightly integrated with Elasticsearch and handles this migration for you.

There is also an API route worth knowing about. The dedicated add index block API, `PUT <index>/_block/write`, is not the same as a settings update. As the index blocks documentation puts it, the dedicated API "ensures for write blocks that, once successfully returning to the user, all shards of the index are properly accounting for the block", including that in-flight writes have completed. That guarantee is the difference between the two routes. If you are scripting an upgrade across many indices, this matters, because the Upgrade Assistant resolves indices one at a time. Note it requires Elasticsearch 8.18.0 or later.

Here is the path forward.

1. Get your cluster back to a running state. The failed 9.1.4 node will not start. Bring the cluster back up on `8.19.4`, where your data is still accessible. If you took a snapshot before the upgrade, keep it close as a safety net.

2. Run a Kibana instance on the same 8.19.4 version. Point it at your cluster. It does not need to be a permanent part of your stack; you can use it just for the migration.

3. Open the Upgrade Assistant. Find it under Stack Management → Upgrade Assistant. It lists every critical issue that would prevent a clean upgrade, including your old 7.x indices.

4. Resolve each old index. For every flagged index, the assistant offers two safe choices:

- **Reindex** the data into a new 8.x index. Pick this when you may still need to write to the data later. Reindexing can change the index, so update any alerts, transforms, or code that targets the old name.
- **Mark as read-only.** Pick this for historical data you no longer write to. This is the action that sets `index.verified_read_only` correctly. It is also the faster route for large indices, since you skip the wait for a full reindex.

5. Clear every critical issue. The node will refuse to start while any incompatible index remains, so work through the list until the assistant reports no blockers.

© 2026 floragunn GmbH · search-guard.com

6. Upgrade to 9.1.4. With every legacy index either reindexed or verified read-only, the boot check passes and your 9.1.4 node starts cleanly.

A quick note on the read-only choice: in 9.x, indices admitted as verified read-only are treated as permanently read-only. If there is any chance you will need to write to an index again, reindex it now rather than marking it read-only.

Takeaways

A few lessons make the next major upgrade smoother:

1. **A write block set through the settings API is not the verified read-only state.** `index.blocks.write: true` applied with `PUT _settings` does not satisfy the 9.x boot check. The check needs `index.verified_read_only`, which the Upgrade Assistant sets for you.
2. **You do not need to touch every index.** Only indices created before 8.0 need action. Setting `_all` to read-only is more than the upgrade path asks for and still misses the marker that matters.
3. **Run the Upgrade Assistant before every major upgrade.** It catches deprecated settings and incompatible indices early, and it applies the correct fixes. Skipping it is the most common reason upgrades fail on boot.
4. **Keep a Kibana instance available for upgrades.** Even if your day-to-day dashboards live elsewhere, the Upgrade Assistant is a Kibana feature and the smoothest way through a major version jump.
5. **Choose read-only or reindex on purpose.** Read-only is quick and permanent; reindex keeps the door open for future writes. Decide based on whether the data is truly historical.

For the full checklist, see Elastic's Prepare to upgrade guide, which walks through the Upgrade Assistant and the read-only-versus-reindex decision in detail.

A clean upgrade is the perfect moment to double-check that your cluster's security travels with it. See how Search Guard keeps authentication, authorization, and audit logging solid across every Elasticsearch version.

This article was published on the Search Guard blog: https://search-guard.com/blog/upgrading-to-elasticsearch-9-why-your-read-only-7-x-indices-still-block-the-boot/
Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · Start a free trial