

#Elasticsearch

# Best Practices for Managing Elasticsearch Indices



Daniel Berman  
Sep 10th, 2019



Elasticsearch is a powerful distributed search engine that has, over the years, grown into a more general-purpose NoSQL storage and analytics tool. The recent release of Elasticsearch 7 added many improvements to the way Elasticsearch works. It also formalized support for various applications

deserves special focus is Elasticsearch indexing and managing indices.

The way data is organized across nodes in an Elasticsearch cluster has a huge impact on performance and reliability. For users, this element of operating Elasticsearch is also one of the most challenging elements. An non-optimized or erroneous configuration can make all the difference. While traditional best practices for managing Elasticsearch indices still apply, the recent releases of Elasticsearch have added several new features that further optimize and automate index management. This article will explore several ways to make the most of your indices by combining traditional advice with an examination of the recently released features.

## Understanding indices

Data in Elasticsearch is stored in one or more indices. Because those of us who work with Elasticsearch typically deal with large volumes of data, data in an index is partitioned across

**shards** to make storage more

manageable. An index may be too large to fit on a single disk, but shards are smaller and can be allocated across different nodes as needed. Another benefit of proper sharding is that searches can be run across different shards in parallel, speeding up query processing. The number of shards in an index is decided upon index creation and cannot be changed later.

### More on the subject:

[Key Trends in Logging Workflows](#)

[Battle of the Automation Servers: Jenkins vs. Bamboo vs. TeamCity](#)

[Monitoring Zoom Metrics from Your Machine with Logz.io](#)

against data loss. To deal with this, we can set up **replication**. Each shard may have a number of **replicas**, which are configured upon index creation and may be changed later. The **primary shard** is the main shard that handles the indexing of documents and can also handle processing of queries. The replica shards process queries but do not index documents directly. They are always allocated to a different node from the primary shard, and, in the event of the primary shard failing, a replica shard can be promoted to take its place.

While more replicas provide higher levels of availability in case of failures, it is also important not to have too many replicas. Each shard has a state that needs to be kept in memory for fast access. The more shards you use, the more overhead can build up and affect resource usage and performance.

## Optimizations for time series data

Using Elasticsearch for storage and analytics of time series data, such as application logs or Internet of Things (IoT) events, requires the management of huge amounts of data over long periods of time.

### Elasticsearch Rollover

Time series data is typically spread across many indices. A simple way to do this is to have a different index for arbitrary periods of time, e.g., one index per day. Another approach is to use the [Rollover API](#), which can automatically create a new index when the main one is too old, too big, or has too many documents.

### Elasticsearch Shrink



you can do to make them use fewer resources so that the more active indices have more resources available. One of these is to use the [Shrink API](#) to flatten the index to a single primary shard. Having multiple shards is usually a good thing but can also serve as overhead for older indices that receive only occasional requests. This, of course, greatly depends on the structure of your data.

## Frozen indices

For very old indices that are rarely accessed, it makes sense to completely free up the memory that they use. Elasticsearch 6.6 onwards provides the [Freeze API](#) which allows you to do exactly that. When an index is frozen, it becomes read-only, and its resources are no longer kept active.

The tradeoff is that frozen indices are slower to search, because those resources must now be allocated on demand and destroyed again thereafter. To prevent accidental query slowdowns that may occur as a result, the query parameter `ignore_throttled=false` must be used to explicitly indicate that frozen indices should be included when processing a search query.

## Index lifecycle management

The above two sections have explained how the long-term management of indices can go through a number of phases between the time when they are actively accepting new data to be indexed to the point at which they are no longer needed.

earlier versions of the E Stack, would have to be done manually or by using external processes. ILM, which is available under Elastic's Basic license and not the Apache 2.0 license, allows users to specify policies that define when these transitions take place as well as the **actions** that apply during each phase.

We can use ILM to set up a **hot-warm-cold architecture**, in which the phases as well as the actions are optional and can be configured if and as needed:

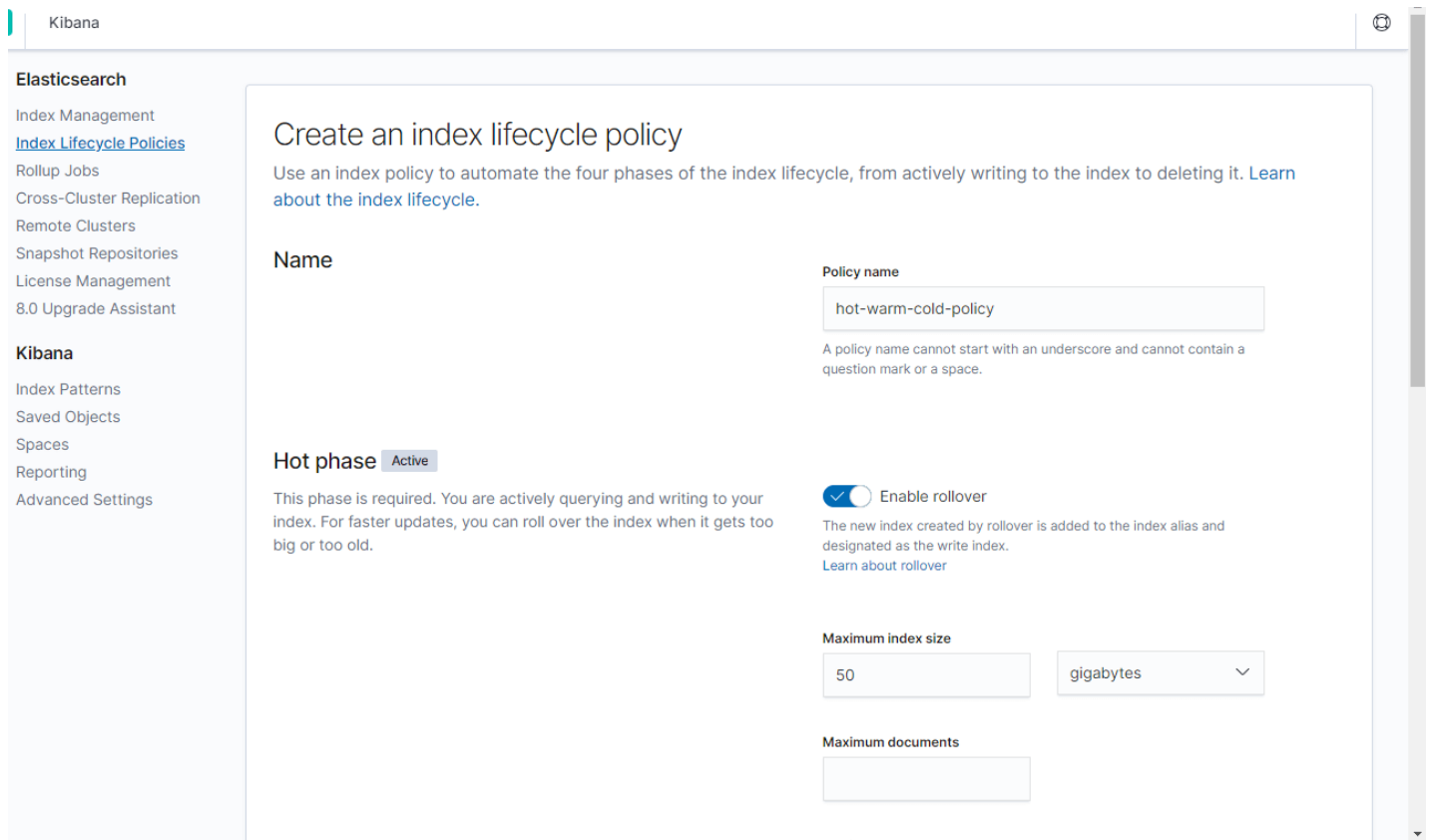
- **Hot** indices are actively receiving data to index and are frequently serving queries. Typical actions for this phase include:
  - Setting high priority for recovery.
  - Specifying rollover policy to create a new index when the current one becomes too large, too old, or has too many documents.
- **Warm** indices are no longer having data indexed in them, but they still process queries. Typical actions for this phase include:
  - Setting medium priority for recovery.
  - Optimizing the indices by shrinking them, force-merging them, or setting them to read-only.
  - Allocating the indices to less performant hardware.
- **Cold** indices are rarely queried at all.

Typical actions for this phase include:

- Setting low priority for recovery.
- Freezing the indices.

- **Delete** indices that are older than an arbitrary retention period.

ILM policies may be set using the Elasticsearch REST API, or even directly in Kibana, as shown in the following screenshot:



The screenshot shows the Kibana interface for creating an index lifecycle policy. The left sidebar contains navigation links for Elasticsearch (Index Management, Index Lifecycle Policies, Rollup Jobs, Cross-Cluster Replication, Remote Clusters, Snapshot Repositories, License Management, 8.0 Upgrade Assistant) and Kibana (Index Patterns, Saved Objects, Spaces, Reporting, Advanced Settings). The main content area is titled 'Create an index lifecycle policy' and includes a description: 'Use an index policy to automate the four phases of the index lifecycle, from actively writing to the index to deleting it. [Learn about the index lifecycle.](#)'

The form includes the following fields and options:

- Name:** A text input field containing 'hot-warm-cold-policy'. Below it, a note states: 'A policy name cannot start with an underscore and cannot contain a question mark or a space.'
- Hot phase:** A toggle switch labeled 'Active'.
- Enable rollover:** A toggle switch that is turned on. Below it, a note states: 'The new index created by rollover is added to the index alias and designated as the write index. [Learn about rollover](#)'
- Maximum index size:** A text input field containing '50' and a dropdown menu set to 'gigabytes'.
- Maximum documents:** An empty text input field.

## Organizing data in Elasticsearch indices

When managing an Elasticsearch index, most of your attention goes towards ensuring stability and performance. However, the structure of the data that actually goes into these indices is also a very important factor in the usefulness of the overall system. This structure impacts the accuracy and flexibility of search queries over data that may potentially come from multiple data sources and as a result also impacts how you analyze and visualize your data.

on the input data it receives, its intuition is based on a small sample of the data set and may not be spot-on. Explicitly creating a mapping can prevent issues with data type conflicts in an index.

Even with mappings, gaining insight from volumes of data stored in an Elasticsearch cluster can still be an arduous task. Data incoming from different sources which may have a similar structure (e.g., an IP address coming from IIS, NGINX, and application logs) may be indexed to fields with completely different names or data types.

The [Elastic Common Schema](#), released with Elasticsearch 7.x, is a new development in this area. By setting a standard to consolidate field names and data types, it suddenly becomes much easier to search and visualize data coming from various data sources. This enables users to leverage Kibana to get a single unified view of various disparate systems they maintain.

## Summary

Properly setting up index sharding and replication directly affects the stability and performance of your Elasticsearch cluster. The aforementioned features are all useful tools that will help you manage your Elasticsearch indices. Still, this task remains one of the most challenging elements for operating Elasticsearch, requiring an understanding of both Elasticsearch's data model and the specific data set being indexed.

For time-series data, the Rollover and Shrink APIs allow you to deal with basic index overflow and optimize indices. The recently added ability to freeze indices allows you to deal with another category of aging indices.



that they take up fewer resources, leaving more resources available for the more active indices.

Finally, creating mappings for indexed data and mapping fields to the Elastic Common Schema can help get the most value out of the data in an Elasticsearch cluster.

**Observability at scale, powered by open source**

See Plans



#### YOU MIGHT ALSO LIKE

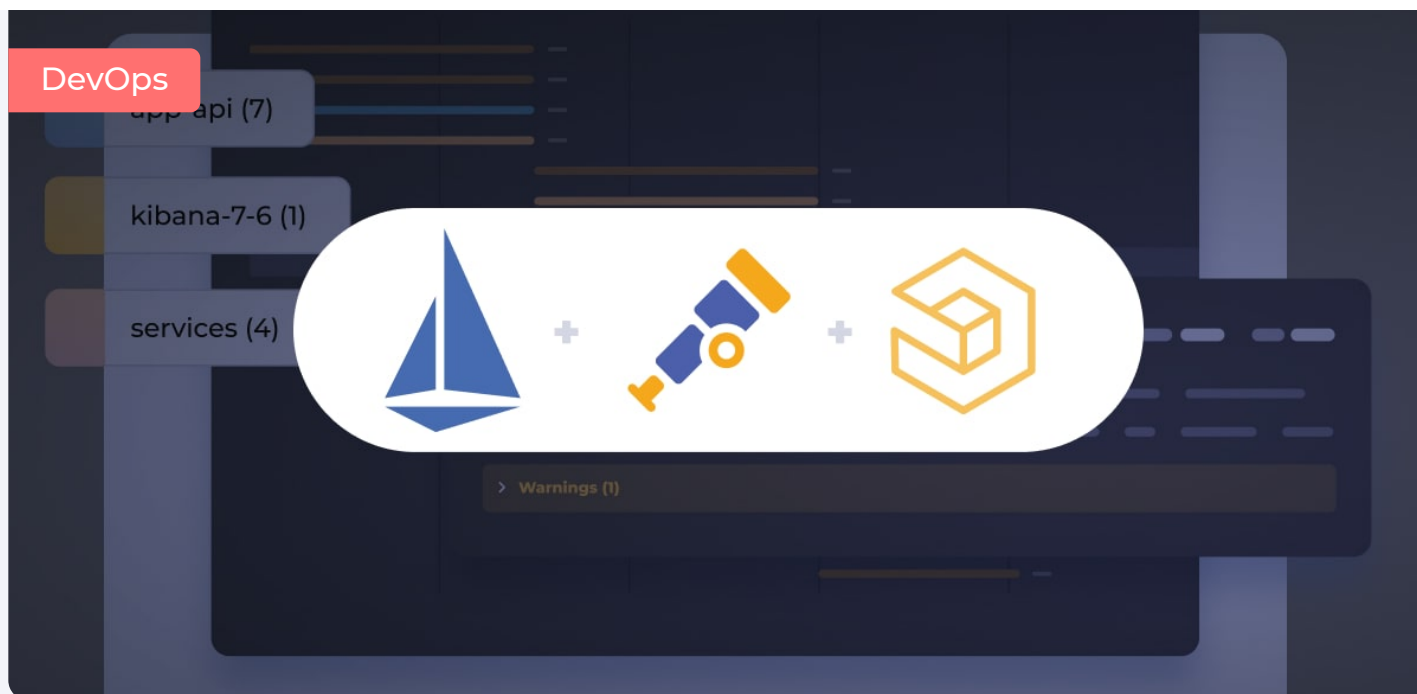
DevOps

Tutorial

## Importing Okta Logs to Logz.io Cloud SIEM

okta





## Instrumenting Microservices with Istio for Distributed Tracing



## How to Give Your eCommerce Monitoring Dashboards More Context

[← Back to Blog](#)



[Privacy Policy](#)

[Terms Of Use](#)

[Trademarks Legal Notice](#)

[Logz.io SLA](#)



All Rights Reserved © 2015-2022, Logshero Ltd.