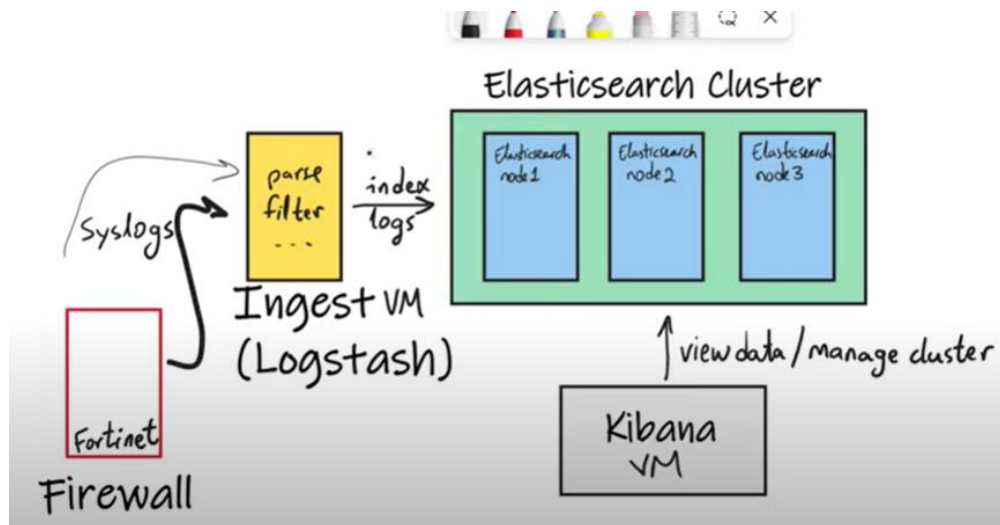


## Elastic Search Index LifeCycle Management ILM:



With data management, we have firewall logs coming into elasticsearch and it can get really big over the period of time. So we need data management and retention period and what happens to indexes in the cluster.

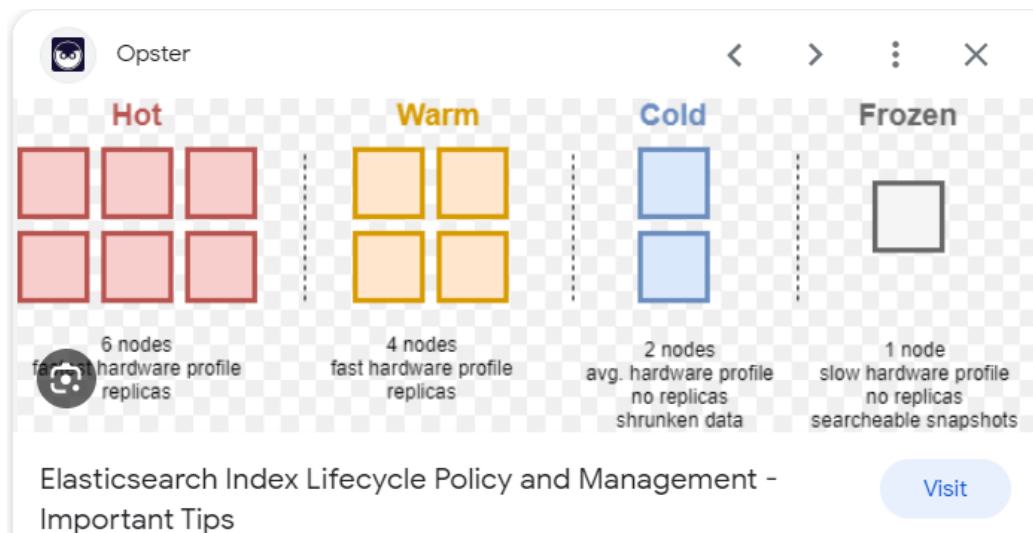
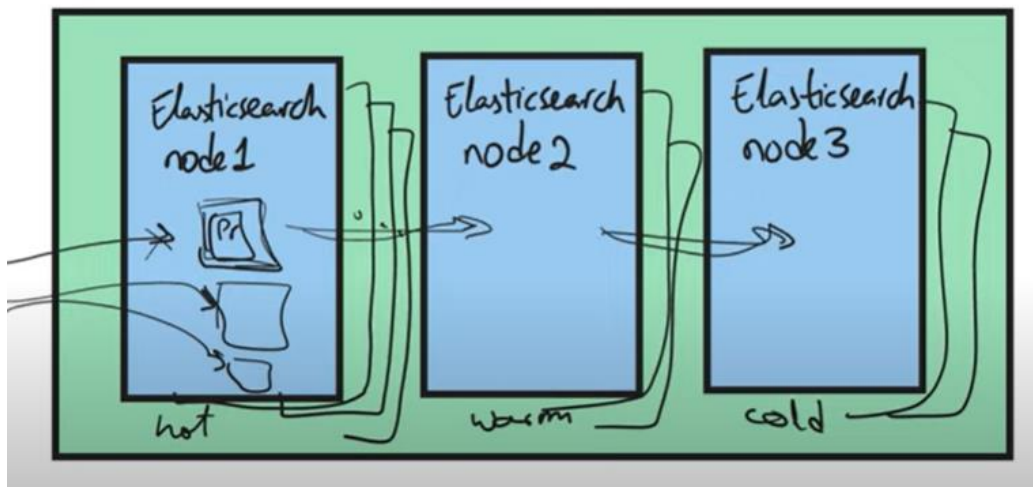
The data you store in elastic search falls into two categories:

1. Content: A collection of items you want to search such as catalog of product.
2. Timeseries data: a stream of continuously generated timestamp data such as log entries.

Storage is expensive you need to determine retention period for the timeseries data.

To help you manage data elastic search enables us to:

1. Determine multiple tiers:
  - Content tier: nodes handle the indexing and query load for content such as product catalog.
  - Hot tier: nodes handle the indexing load for timeseries data such as logs or metrics and hold your most recent and most frequently accessed data.
  - Warm tier: nodes hold the timeseries data that is less frequently accessed data and rarely needs to be updated
  - Cold tier: nodes hold the timeseries data that is accessed infrequently and not normally updated. To save space you can keep fully mounted indices of searchable snapshots on the cold tier. They eliminate the need of replicas.
  - Frozen tier: nodes hold the timeseries data that is accessed rarely and never updated. stores partially mounted indices of searchable snapshots exclusively.



You have to specify in elasticsearch.yml file to determine your node belongs to which tier.

```
GNU nano 2.9.8 /etc/elasticsearch/elasticsearch.yml

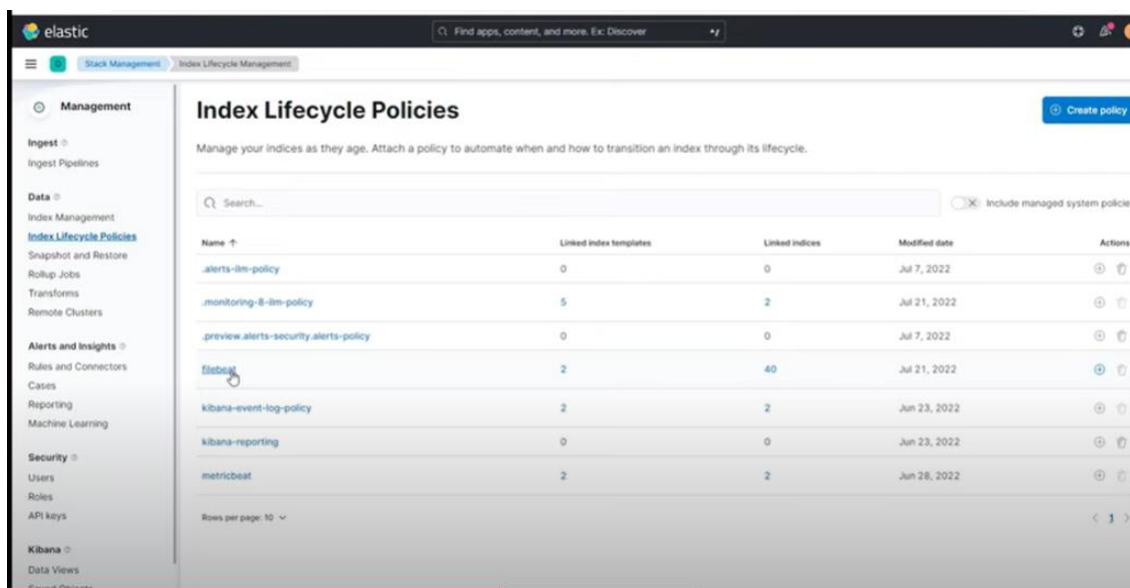
cluster.name: Fortified
node.name: chamber1
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 192.168.25.100
http.port: 9200

node.roles: [ master, data_hot, data_content, ingest, transform, remote_cluster$

# Enable security features
xpack.security.enabled: true
xpack.security.enrollment.enabled: true
# Enable encryption for HTTP API client connections, such as Kibana, Logstash, $
xpack.security.http.ssl:
  enabled: true
  keystore.path: certs/http.p12
# Enable encryption and mutual authentication between cluster nodes
xpack.security.transport.ssl:
  enabled: true
```

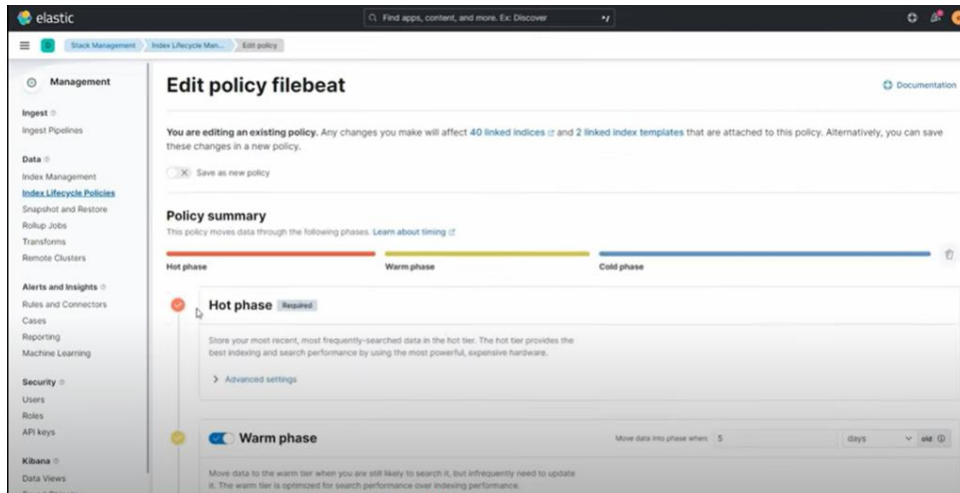
In Kibana go to Stack management -> under data click on Index lifecycle policies

You have filebeat policy created by default because we are using filebeat

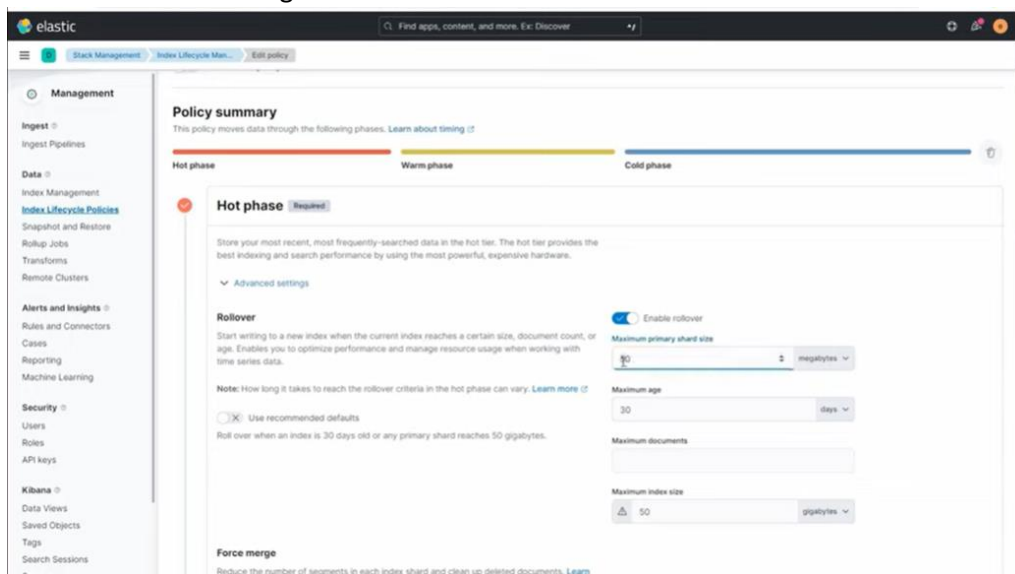


The screenshot shows the Kibana Index Lifecycle Policies page. The left sidebar contains a navigation menu with sections: Management, Ingest, Data, Alerts and Insights, Security, and Kibana. The 'Data' section is expanded, showing 'Index Lifecycle Policies' as the selected option. The main content area is titled 'Index Lifecycle Policies' and includes a search bar and a table of policies. The table has columns for Name, Linked Index templates, Linked indices, Modified date, and Actions. The 'filebeat' policy is highlighted in blue.

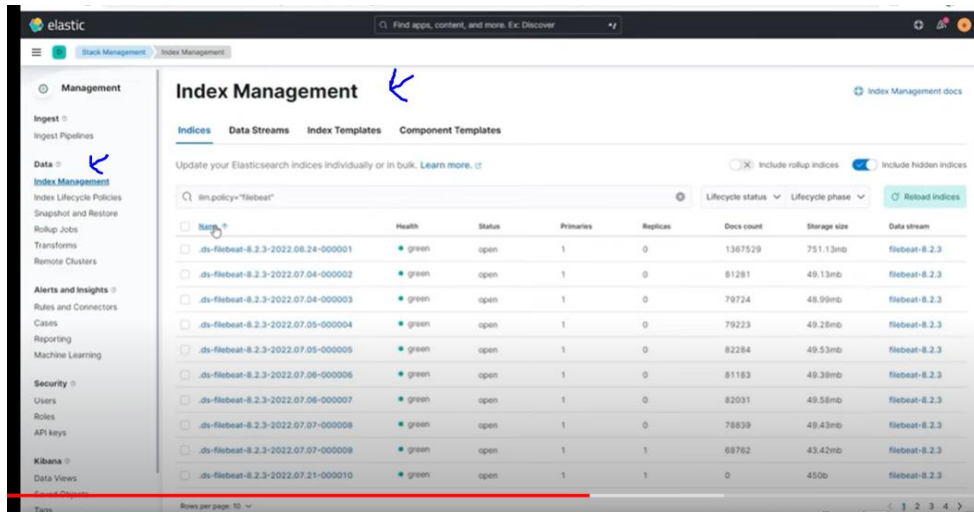
Name	Linked Index templates	Linked indices	Modified date	Actions
.alerts-lm-policy	0	0	Jul 7, 2022	<a href="#">+</a> <a href="#">-</a>
.monitoring-8-lm-policy	5	2	Jul 21, 2022	<a href="#">+</a> <a href="#">-</a>
.preview.alerts-security.alerts-policy	0	0	Jul 7, 2022	<a href="#">+</a> <a href="#">-</a>
filebeat	2	40	Jul 21, 2022	<a href="#">+</a> <a href="#">-</a>
kibana-event-log-policy	2	2	Jun 23, 2022	<a href="#">+</a> <a href="#">-</a>
kibana-reporting	0	0	Jun 23, 2022	<a href="#">+</a> <a href="#">-</a>
metricbeat	2	2	Jun 28, 2022	<a href="#">+</a> <a href="#">-</a>



Click on advance settings:



Index Management:



## Snapshot Lifecycle Management:

<https://coralogix.com/blog/tutorial-elasticsearch-snapshot-lifecycle-management-slm/>

Snapshot Lifecycle Management, or *SLM*, as we'll refer to it in this course, helps us fully automate backups of our Elasticsearch clusters and indices.

We can set up policies to instruct SLM when to backup, how often, and how long to keep each snapshot around before automatically deleting it.

To experiment with Elasticsearch Snapshot Lifecycle Management we'll need to:

1. **Set up a repository** where the snapshots can be stored
2. **Configure repository** in our Elasticsearch cluster
3. **Define the SLM policy** to automate snapshot creation and deletion
4. **Test the policy** to see if we registered it correctly and works as expected

The steps we will take are easy to understand, once we break them down the basic actions

## 1. Set Up a Repository

A repository is simply a place to save files and directories, just as you would on your local hard-drive. Elasticsearch uses repositories to store its snapshots.

The first type we'll explore is the **shared file system repository**. In our exercise, since we're working with a single node, this will be easy. However, when multiple nodes are involved, we would have to configure them to access the same filesystem, possibly located on another server.

The second repository type we will explore relies on **cloud data storage services**, that uses service-specific plugins to connect to services like AWS S3, Microsoft Azure's object storage, or Google Cloud Storage Repository (GCS).

## 2. Configure Repository

After picking the repository that best fits our needs, it's time to let Elasticsearch know about it. If we use a shared file system, we need to add a configuration line to the *elasticsearch.yml* file, as we'll see in the exercises.

For cloud-based storage repositories, we'll need to install the required repository [plugin](#) on each node. Elasticsearch needs to log in to these cloud services, so we will also have to add the required secret keys to its keystore. This will be fully explained in the exercises.

## 3. Define the Elasticsearch Snapshot Policy

At this point, all prerequisites are met and we can define the policy to instruct SLM on how to automate backups, with the following parameters

- 
- ***schedule***: What frequency and time to snapshot our data. You can make this as frequent as you require, without worrying too much about storage constraints. Snapshots are incremental, meaning that only the differences between the last snapshot and current snapshot need to be stored. If almost nothing changed between yesterday's backup and today's data, then the Elasticsearch snapshot will require negligible storage space, meaning that even if you have gigabytes of data, the snapshot might require just a few megabytes or less.
- ***name***: Defines the pattern to use when naming the snapshots
- ***repository***: Specifies where to store our snapshots
- ***config.indices***: List of the indices to include
- ***retention***: Is an optional parameter we can use to define when SLM can delete some of the snapshots. We can specify three options here:
  - ***expire\_after***: This is a time-based expiration. For example, a snapshot created on January 1, with *expire\_after* set to 30 days will be eligible for deletion after January 31.
  - ***min\_count***: Tells Elasticsearch to keep at least this number of snapshots, even if all are expired.
  - ***max\_count***: Tells Elasticsearch to *never* keep more than this number of snapshots. For example, if we have 100 snapshots and only one is expired, but *max\_count* is set to 50, then 50 of the oldest snapshots will be deleted – even the unexpired ones.

## 4. Test the Policy

With our policy finally defined, we can display its status. This will list policy details and settings, show us how many snapshot attempts were successful, how many failed, when SLM is scheduled to run the next time, and other useful info.

### Kibana roles/mapping:

Role mappings

Role mappings are part of single sign-on (SSO), a subscription feature. This feature allows you to describe which roles to assign to your users using a set of rules.

Role mappings are required when authenticating via an external identity provider, such as Active Directory, Kerberos, PKI, OIDC, or SAML. Role mappings have no effect for users inside the native or file realms.

To manage your role mappings, open the main menu, then click **Stack Management > Role Mappings**.

With **Role mappings**, you can:

- View your configured role mappings
- Create/Edit/Delete role mappings

## Role Mappings

Create role mapping

Role mappings define which roles are assigned to users from an external identity provider. [Learn more.](#)

Reload

<input type="checkbox"/> Name ↑	Roles	Enabled	Actions
<input type="checkbox"/> Administrators	superuser	Enabled	
<input type="checkbox"/> Operations	kibana_admin monitoring_user rollup_user	Enabled	
<input type="checkbox"/> Sales	kibana_admin	Enabled	

Rows per page: 20 < 1 >

## Required permissions

The `manage_security` cluster privilege is required to manage Role Mappings.

## Create a role mapping

1. Open the main menu, then click **Stack Management > Role Mappings**.
2. Click **Create role mapping**.
3. Give your role mapping a unique name, and choose which roles you wish to assign to your users.

If you need more flexibility, you can use [role templates](#) instead.

4. Define the rules describing which users should receive the roles you defined. Rules can optionally be grouped and nested, allowing for sophisticated logic to suite complex requirements.
5. View the [role mapping resources for an overview of the allowed rule types](#).

## 14 tips on how to reduce Elasticsearch search latency and optimize search performance:

### 1. Size parameter

Assigning a **huge value to size parameter** causes Elasticsearch to compute vast amounts of hits, which causes severe performance issues. Instead of setting a huge size, you should batch requests in small sizes.

### 2. Shards and replicas

Optimize necessary index settings that play a crucial role in Elasticsearch performance, like the **number of shards and replicas**. In many cases having more replicas helps improve search performance. Please refer to [Opster's guide on shards and replicas](#) to learn more.

### 3. Deleted documents

Having a large number of deleted documents in the Elasticsearch index also causes search performance issues, as explained in this [official document](#). **Force merge** API can be used to remove a large number of deleted documents and optimize the shards.

### 4. Search filters



Effective use of filters in Elasticsearch queries can improve search performance dramatically as the **filter clauses** are 1) cached, and 2) able to reduce the target documents to be searched in the query clause.

#### 5. **Wildcard queries**

Avoid wildcard, especially **leading wildcard** queries, which causes the entire Elasticsearch index to be scanned.

#### 6. **Regex and parent-child**

Note that Regex queries and parent-child can cause search latency.

#### 7. **Implementing features**

There are multiple ways to implement a specific feature in Elasticsearch. For example, Autocomplete can be implemented in various styles. [Opster's blog](#) gives a 360-degree view of both functional and non-functional features (especially performance).

#### 8. **Multitude of small shards**

Having **many small shards** could cause a lot of network calls and threads, which severely impact search performance; please refer to this [real-world case study by Opster's expert](#) on this topic.

#### 9. **Heavy aggregations**

Avoid **heavy aggregations** that involve unique IDs. Refer to Opster's [slow logs](#) guide to identify such search slow logs effectively.

#### 10. **Timeout and terminate**

**Timeout param and terminate** after param can be useful when executing heavy searches, or when result data is vast. This [official guide can help](#).

## 11. Search templates

Use [search templates](#) to achieve better abstraction, meaning without exposing your query syntax to your users. Search templates also help you transfer less data over the network, which is particularly useful when you have large Elasticsearch queries.

## 12. Multi search API

Use [msearch](#) whenever possible. In most of the applications it's required to query multiple Elasticsearch indices for a single transaction, and sometimes users do so in a serial order even when it's not required. In both cases, when you need to query multiple indices for the same transaction and when the result of these queries are independent, you should always use msearch to execute the queries in parallel in Elasticsearch.

## 13. Term queries

Use term query when you need an exact match and on keywords fields. By default, Elasticsearch generates both text and keyword fields for every field that consists of a string value if explicit mapping is not supplied. Users tend to use the match query even on keyword data types like product-ids, which is costly as match query goes through an analysis operation. Read the difference between [Term vs Match query](#) and always use term query on keyword data types and wherever you need exact searches for better performance.

## 14. Source filtering

[\\_source filtering](#) is a great way to improve the performance of Elasticsearch queries when retrieving a large number of documents or documents of large sizes. By default, Elasticsearch returns the complete source of matching documents. If you don't need `_source` at all or need only values of specific fields, you can achieve this with `_source` filtering.