Coralogix

Getting the best performance from Elasticsearch

BY DEKLAN VAN DE LAARSCHOT

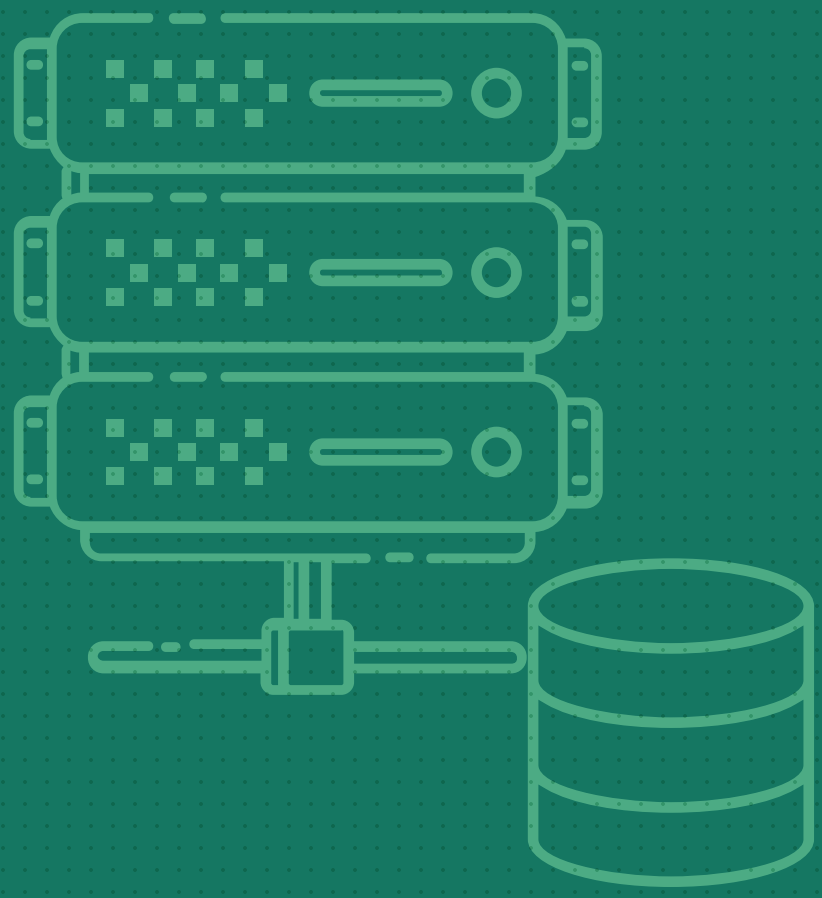
INTRODUCTION

Elasticsearch is a platform that provides a distributed open-source search solution for all types of data. A common use case is ELK (Elastic, Logstash & Kibana). These products combined create a powerful platform for infrastructure and application central logging with the ability to visualize large sums of data. As a result, more and more businesses are implementing the ELK stack.

However, they all quickly run into trouble with scaling and keeping query performance at its optimum. This eBook will uncover the art of keeping the ELK stack running at its best and help you improve existing implementations.

<PART 1>

INFRASTRUCTURE



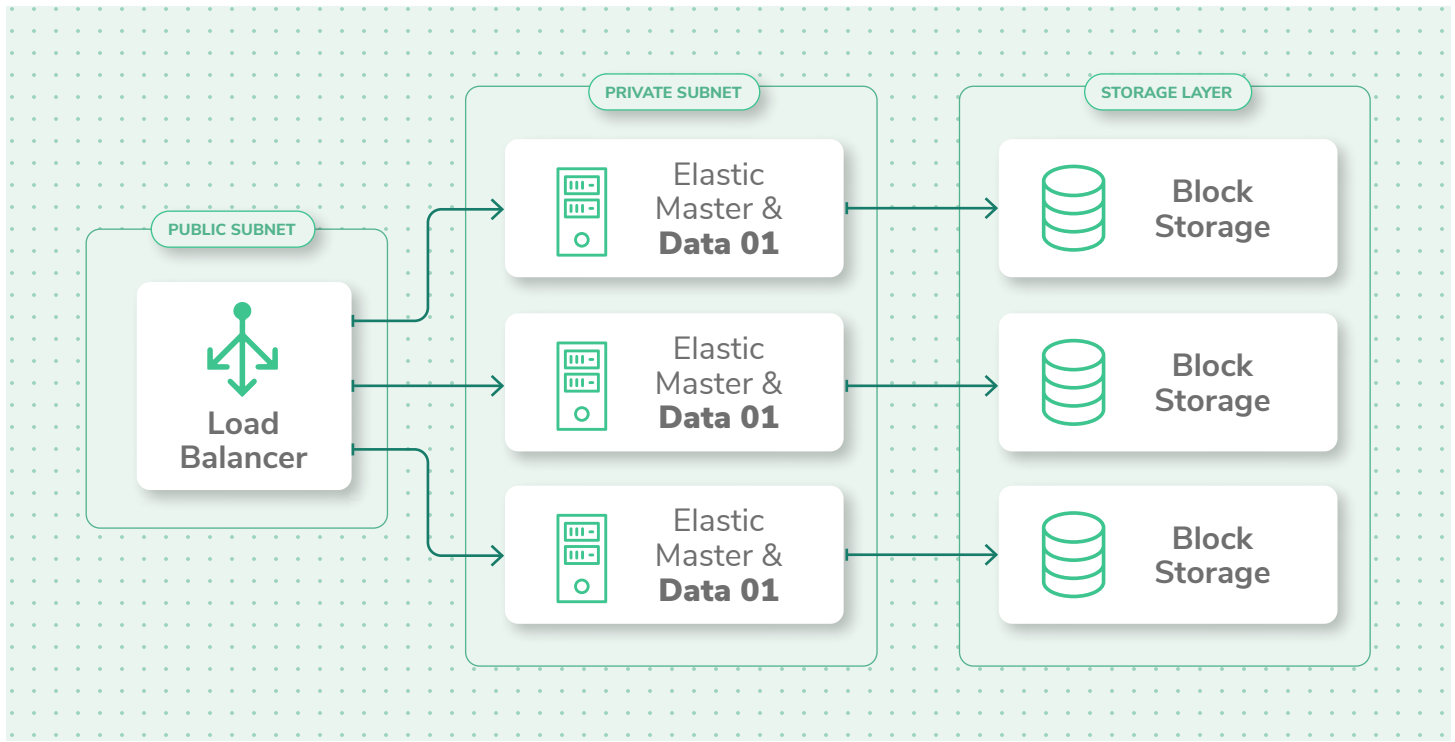
INFRASTRUCTURE

Getting your foundations right is essential for building a performant cluster. The infrastructure (Virtual Machines, Network, Storage, etc) is the most critical part of an elastic deployment. If you get it wrong no matter how much you adjust any performance configurations the platform will always underperform. Resulting in poor query, write, index and update performance!

Let's explore what a good infrastructure base looks like when deploying the Elastic stack. The Elastic stack can be deployed on-premises, in the cloud, in docker, and a number of other ways. The requirements for the base infrastructure remain mostly the same. Firstly and most importantly you are going to want a minimum of three nodes (Virtual Machines Or Containers). You should ideally design this to span multiple zones to build a fully resilient cluster. For example in the Cloud, you want to ensure you have nodes sat in different zones to mitigate risks of outages.

In a physical environment, this might be a separate rack or data centre. Remember latency and bandwidth are important. You want to ensure you have a low latency connection between nodes and ample bandwidth. When designing these capabilities Elasticsearch has a setting for 'Shard Allocation Awareness'. This configuration provides the cluster with information regarding which nodes are on the same physical server, in the same rack, or in the same zone. This allows Elastic to distribute the primary shard and its replica shards to minimize the risk of losing all shard copies in the event of a failure. Check out how to enable Shared Allocation Awareness here.

INFRASTRUCTURE



3 Nodes as above should be ideal for most Elasticsearch deployments but for much larger deployments nodes can be split into different personas. This provides the capability to scale horizontally, which is very suited for Cloud computing. The different personas are as follows:

MASTER NODES	This node's role is to control the cluster.
HTTP NODES	This node's role is to run queries from.
DATA NODES	These nodes contain the data for the cluster.
COORDINATING NODES	This node provides load balancing capabilities.
LOGSTASH NODES	This node's role is to run the Logstash application. To collect logs and parse them into Elasticsearch.
KIBANA NODES	This node's role is to run the Kibana application.

INFRASTRUCTURE

When selecting CPU, RAM and storage as a general rule of thumb start bigger and scale back as required. The minimum number of nodes required for a production environment is 3. You will want to ensure they have SSD storage (write performance is seriously important when building your cluster), a minimum of 16 GB of ram (The sweet spot is actually around 64 GB) and a powerful CPU with multiple cores. Below is the recommended infrastructure for the cloud and on-premises.

PROVIDER	COMPUTE	STORAGE	NOTES
AWS	R Series with a minimum of 2 cores.	EBS - IO2	Avoid EFS
AZURE	E Series with a minimum of 2 cores.	Managed Disks - P series	Avoid Azure Files
ON-PREM	2 cores & a minimum of 8GB of ram.	SSD's	N/A

<PART 2>

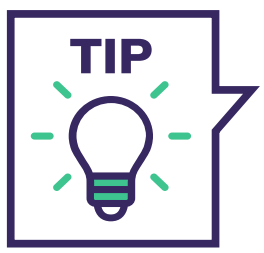
BASIC PERFORMANCE TUNING

BASIC PERFORMANCE TUNING

In this section we are going to run though adjusting your cluster for the best possible performance. It's important to collect metrics to verify that your changes have indeed improved the performance of your cluster. As a result we will also explore how to monitor parts of your Elastic cluster and which metrics provide insight into the granular parts of your Elastic platform.

How to measure performance

Setting up performance benchmarks is essential for monitoring and verifying your elastic cluster is performing at its best. Elasticsearch provides all of the required metrics to tackle all of the problems you might encounter. To collect our monitoring analytics we shall use Metric beat, it's super simple to set up and has all of the monitoring we are going to need built in! The areas that you should be monitoring are broken down below.



In production, it is strongly recommended to use a separate elastic cluster for monitoring. Using a separate cluster will prevent cluster issues impacting your ability to access monitoring data.

Setting up Metric Beat

Setting up Metric beat is very similar to most Elastic products, you will need to install the Metric beat package on each of the nodes and add the correct configuration to the config files to ship the logs to your monitoring cluster. Different operating systems have different packages. The most common way is to use a package manager as below:

INSTALLATION

APT PACKAGE MANAGER

Download and install the Public Signing Key:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch
| sudo apt-key add -
```

On debian you will need to ensure https transport is enabled for APT, the below will fix that:

```
sudo apt-get install apt-transport-https
```

Save the repository definition to /etc/apt/sources.list.d/elastic-7.x.list:

```
echo “deb https://artifacts.elastic.co/packages/7.x/apt
stable main” | sudo tee -a /etc/apt/sources.list.d/elastic-
7.x.list
```

BASIC PERFORMANCE TUNING

You will need to update apt using the below, the && will then run the install command for the metric beat package:

```
sudo apt-get update && sudo apt-get install metricbeat
```

Make sure you enable metric beat to run at boot with the below:

```
sudo systemctl enable metricbeat
```

If your system does not use systemd then run:

```
sudo update-rc.d metricbeat defaults 95 10
```

INSTALLATION

YUM PACKAGE MANAGER

Download and install the public signing key:

```
sudo rpm --import https://packages.elastic.co/GPG-KEY-
elasticsearch
```

Create a file with a .repo extension (for example, elastic.repo) in your /etc/ yum.repos.d/ directory and add the following lines:

```
[elastic-7.x]
name=Elastic repository for 7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
gpgcheck=1
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch
enabled=1
autorefresh=1
type=rpm-md
```

Now you can install Metric beat by running:

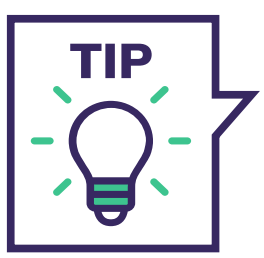
```
sudo yum install metricbeat
```

Make sure you enable Metric beat to run at boot with the below:

```
sudo systemctl enable metricbeat
```

If your system does not use systemd then run:

```
sudo chkconfig --add metricbeat
```



Metric Beat can be installed manually and from source. More information about this method can be found on the ElasticSearch website.

Configuring Metricbeat

Now that we have Metric beat installed on each of the nodes in our cluster, it's time to configure Metric beat to ship monitoring data to our monitoring cluster. Firstly you need to enable monitoring on the cluster. Using curl you can enable this:

To check if monitoring is enabled:

```
curl -X GET "localhost:9200/_cluster/settings?pretty"
```

BASIC PERFORMANCE TUNING

To enable monitoring:

```
curl -X PUT “localhost:9200/_cluster/settings?pretty” -H
‘Content-Type: application/json’ -d’
{
  “persistent”: {
    “xpack.monitoring.collection.enabled”: true
  }
}
’
```

Enable the x-pack monitoring service on each of the nodes:

```
metricbeat modules enable elasticsearch-xpack
```

Configure the Metric beat service. Edit the modules.d/elasticsearch-xpack.yml file to contain the below:

```
- module: elasticsearch
    xpack.enabled: true
    period: 10s
    hosts: [“http://localhost:9200”]
    #scope: node
    #username: “user”
    #password: “secret”
    #ssl.enabled: true
    #ssl.certificate_authorities: [“/etc/pki/root/ca.pem”]
    #ssl.certificate: “/etc/pki/client/cert.pem”
    #ssl.key: “/etc/pki/client/cert.key”
    #ssl.verification_mode: “full”
 xpack.enabled: true
```

BASIC PERFORMANCE TUNING

The localhost configuration is where metricbeat will collect data from. As we are deploying the agent on each of the nodes this will work fine. We now need to configure metricbeat to ship data to our monitoring cluster. Edit the `metricbeat.yml` file to contain the below:

```
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: [“http://es-mon-1:9200”, “http://es-mon2:9200”]
  # Optional protocol and basic auth credentials.
  #protocol: “https”
  #username: “elastic”
  #password: “changeme”
```

The hosts field should contain the address of your monitoring cluster.

Now we can restart Metric beat on each of the nodes.

```
systemctl restart metricbeat
```

Now we can disable the default collection of monitoring metric on our production cluster using curl:

```
curl -X PUT “localhost:9200/_cluster/settings?pretty” -H
‘Content-Type: application/json’ -d’
{
  “persistent”: {
    “xpack.monitoring.elasticsearch.collection.enabled”:
false
  }
}
‘
```

Viewing monitoring metrics in Kibana

Now that we are collecting the data it's time to see what it looks like. Kibana has a number of premade templates to view your Elastic Search metrics with minimal work! In this section we will quickly cover how to view your metrics, before jumping into tuning and explaining what they all mean.

Before we start, verify that the `monitoring.ui.enabled` setting is set to **true** in your kibana.yml configuration file. Whilst this is the default value, if it has been changed you will be unable to see the monitoring tab in Kibana.

➜ **NOTE: If you are using Elastic's security features you will need to provide a user ID and password for Kibana to retrieve data.**

Firstly let's load up all the premade templates shipped with Metric beat. On a node that has Metric beat installed run the below, to import the default templates into your Kibana instance.

```
metricbeat setup -e
```

Next we need to create a user that has the `monitoring_user built-in` role on the monitoring cluster. Add the `monitoring.ui.elasticsearch.` username and `monitoring.ui.elasticsearch.password` settings in the kibana.yml file. The Kibana configuration file will be contained on nodes running Kibana!

➜ **NOTE: That if these settings are omitted, Kibana will use the elasticsearch.username and elasticsearch.password setting values.**

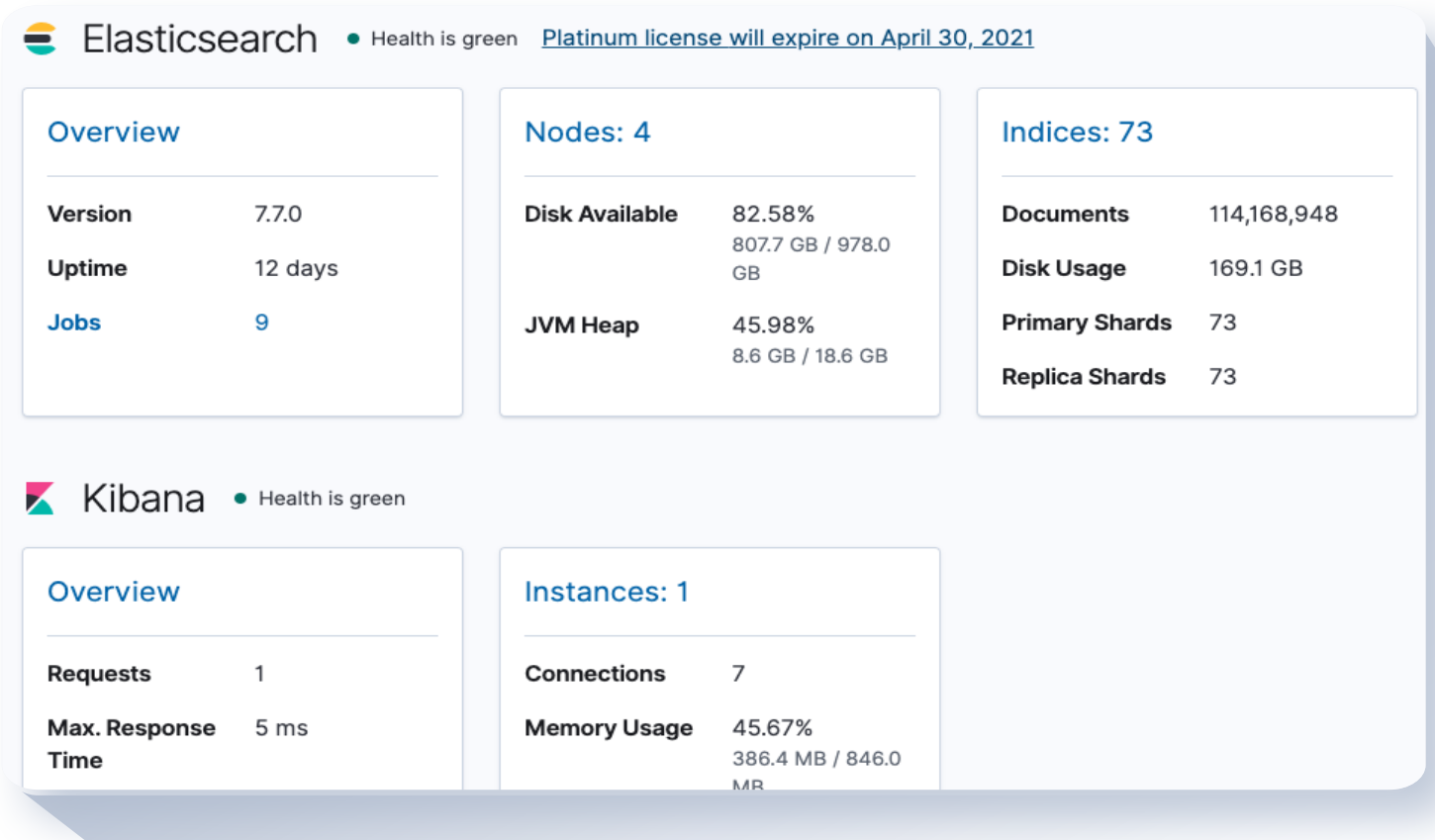
BASIC PERFORMANCE TUNING

If the Elastic security features are enabled on the Kibana server, only users that have the authority to access Kibana indices and to read the monitoring indices can use the monitoring dashboards. These users must exist on the monitoring cluster. If you are accessing a remote monitoring cluster, you must use credentials that are valid on both the Kibana server and the monitoring cluster. The users you create should have the `monitoring_user and kibana_admin built-in roles.`

Open Kibana in your web browser.
If you are running Kibana locally, browse to http://localhost:5601/.

If the Elastic security features are enabled, log in.

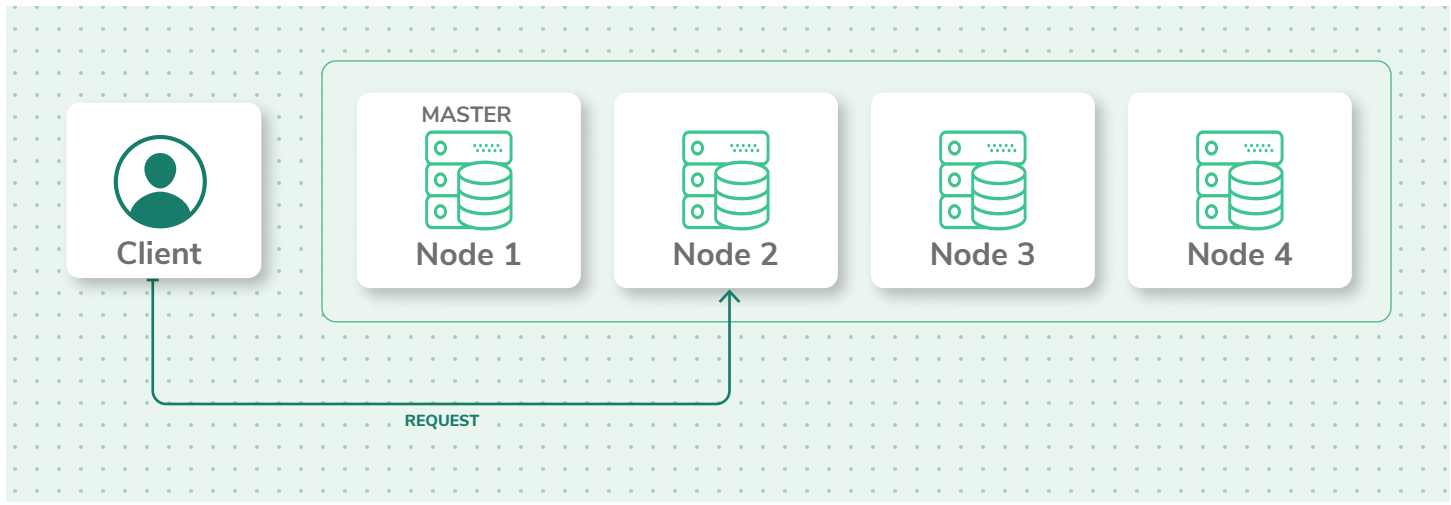
Finally Open ‘Stack Monitoring’. You are now presented with the below and ready to enjoy monitoring using Elasticsearch, Metric beat & Kibana!



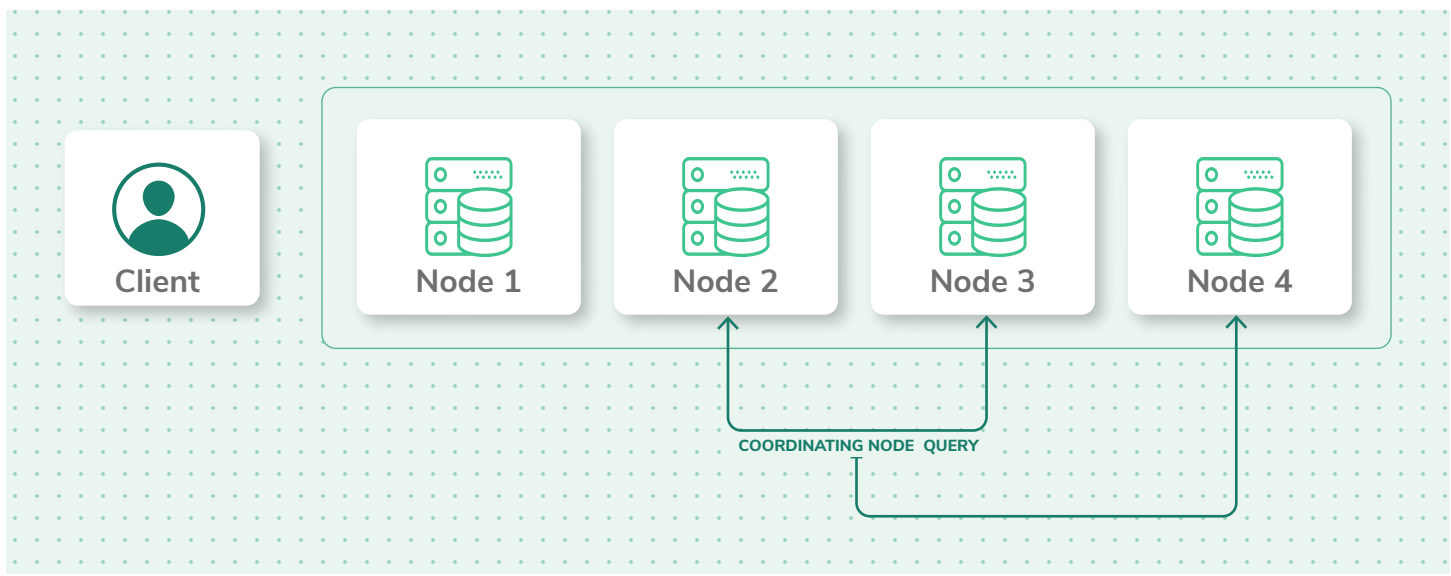
Search performance

In Elasticsearch, search requests are one of two types. They are very similar to traditional read and write requests. Elastic provides metrics that correspond with the two phases of the search process (fetch & query). The below will describe how this process works:

1 The client send a request to node 2

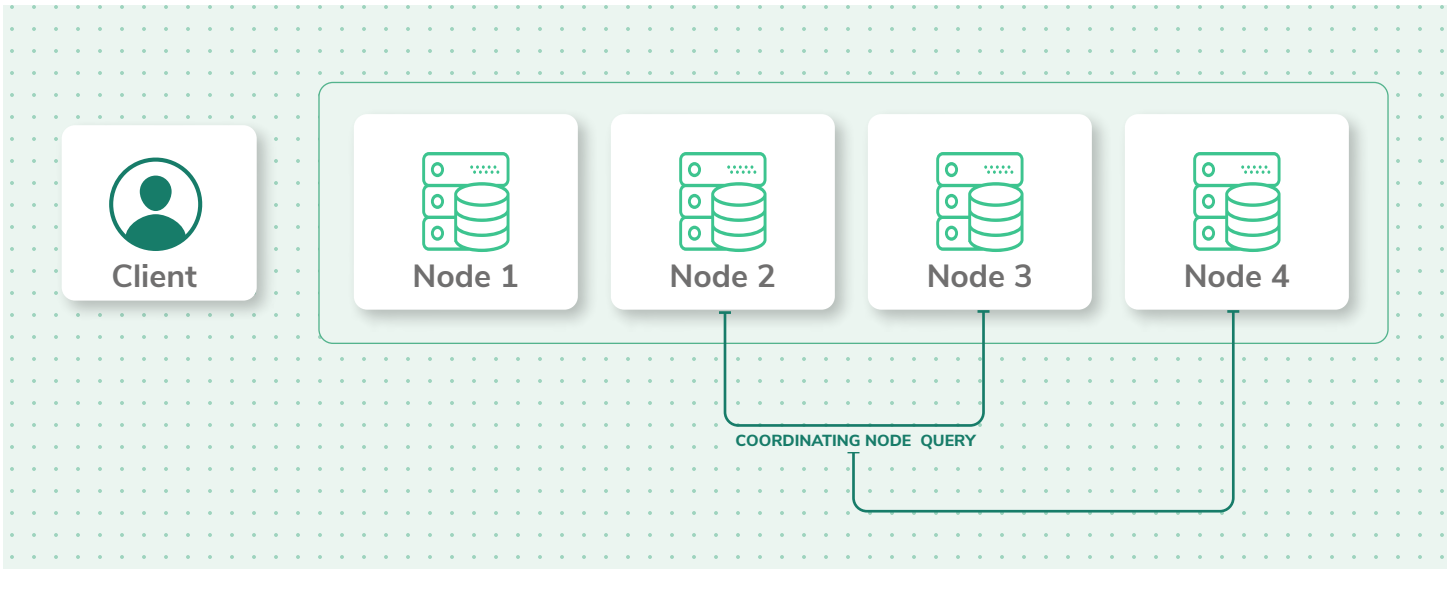


2 Node 2 (which is the coordinating node) sends the query to a copy (either replica or primary) of all of the shards in the index.

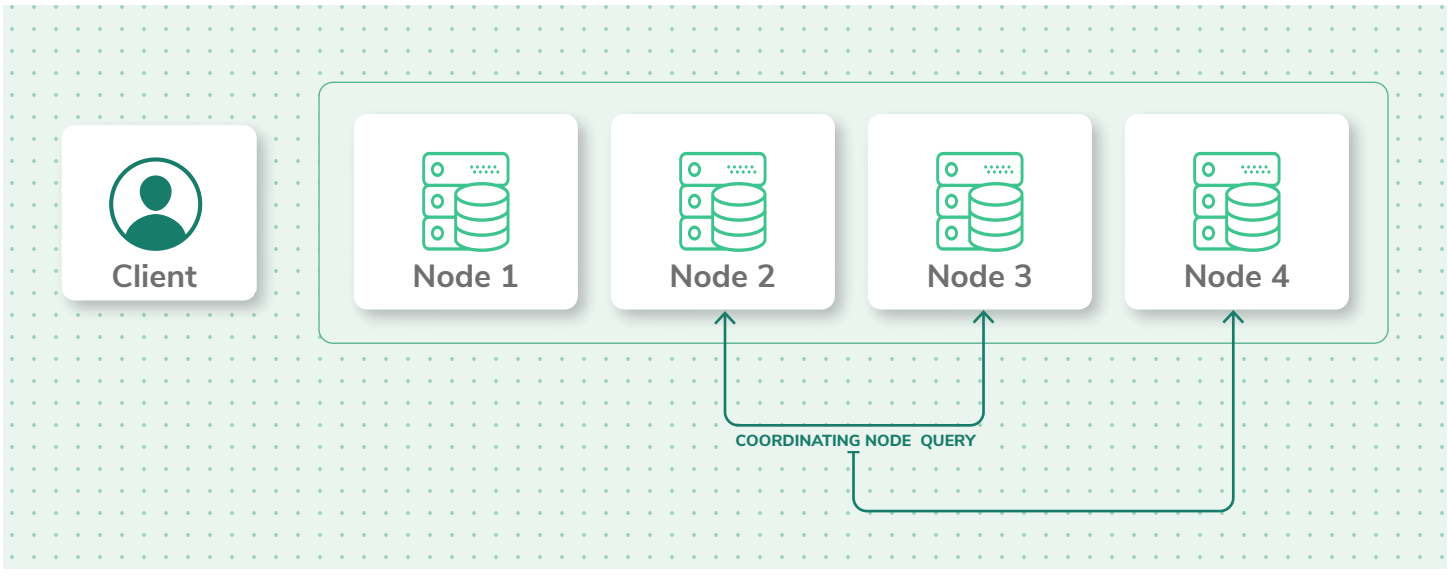


BASIC PERFORMANCE TUNING

3 Each of the shards executes the query locally and delivers results to Node 2. Node 2 is then responsible for sorting and compiles them into a global priority queue.

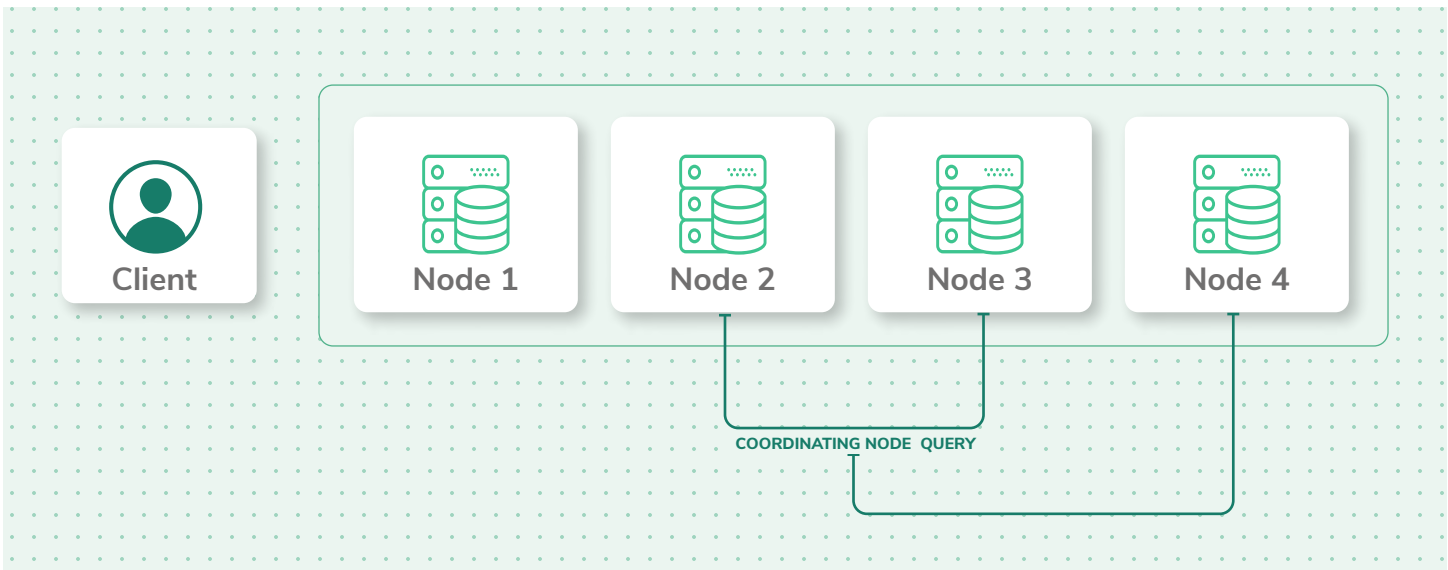


4 Node 2 then finds out which documents need to be fetched and sends a multi GET request to all of the relevant shards.

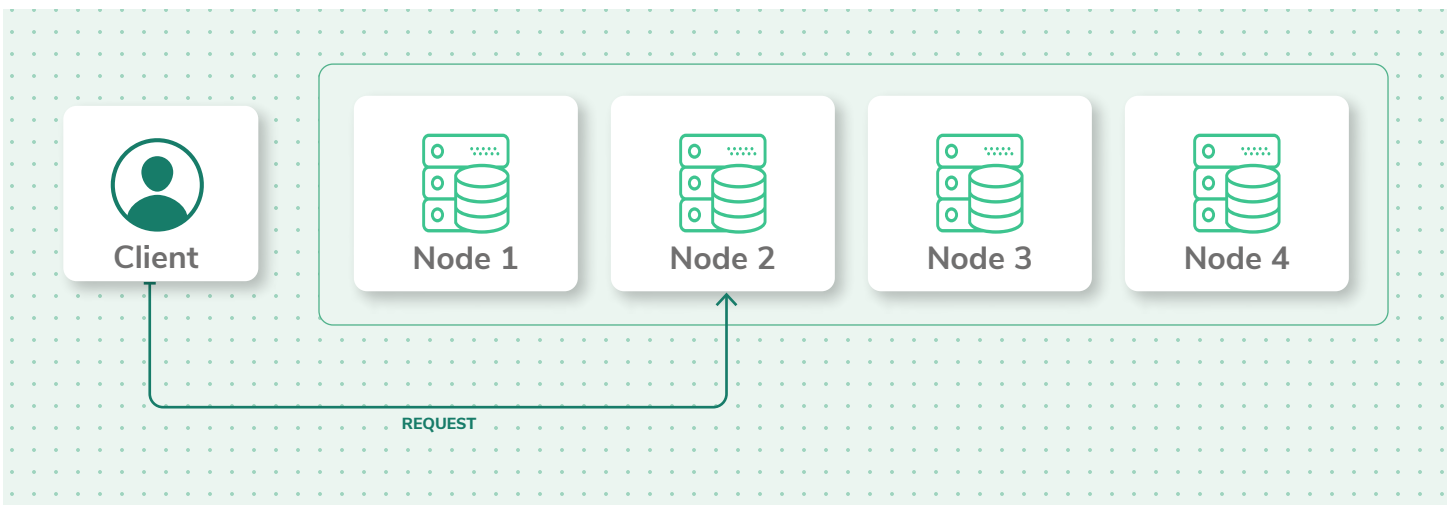


BASIC PERFORMANCE TUNING

5 Each of the shards load the documents and returns them to Node 2.



6 Node 2 then delivers the search results to the client. Providing the queried data.



OPTIMIZING YOUR CLUSTER FOR SEARCH PERFORMANCE

For Elasticsearch clusters that are used mainly for search or if searching is a customer-facing feature, then you should be monitoring query latency and start remediating issues in the event that the cluster surpasses a defined threshold. Collecting metrics over a couple of months will provide great insight into areas of the platform that need to be tuned and this data will be integral to sections later on in this ebook.

METRICS TO MONITOR

NAME	TYPE	DESCRIPTION
`indices.search.query_total`	Throughput	The total number of queries run.
`indices.search.query_time_in_millis`	Performance	The total time spent processing queries.
`indices.search.query_current`	Throughput	The total number of queries currently being processed.
`indices.search.fetch_total`	Throughput	The total number of fetches.
`indices.search.fetch_time_in_millis`	Performance	The total time spent processing fetches.
`indices.search.fetch_current`	Throughput	The total number of fetches currently being processed.

BASIC PERFORMANCE TUNING

QUERY LOAD

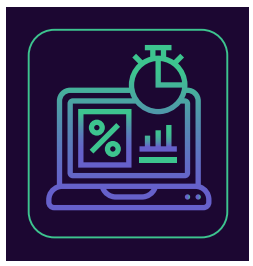
(Total Queries & Current Running Queries)

Collecting these metrics will allow you to generate historical data presenting the normal load for your cluster whilst processing a number of queries. Moreover, the currently running queries can be used to quickly find underlying problems. Consider alerting on unusual spikes or dips, these are normally indications of a problem.

QUERY LATENCY

(Total queries & Time)

Using these metrics combined will allow you to calculate the average latency, this is achieved by sampling the total number of queries and the total elapsed time at regular intervals. It would be prudent to set up an alert if the latency exceeds a threshold. This should provide a good indication of any potential resource bottlenecks or queries that require optimisation.



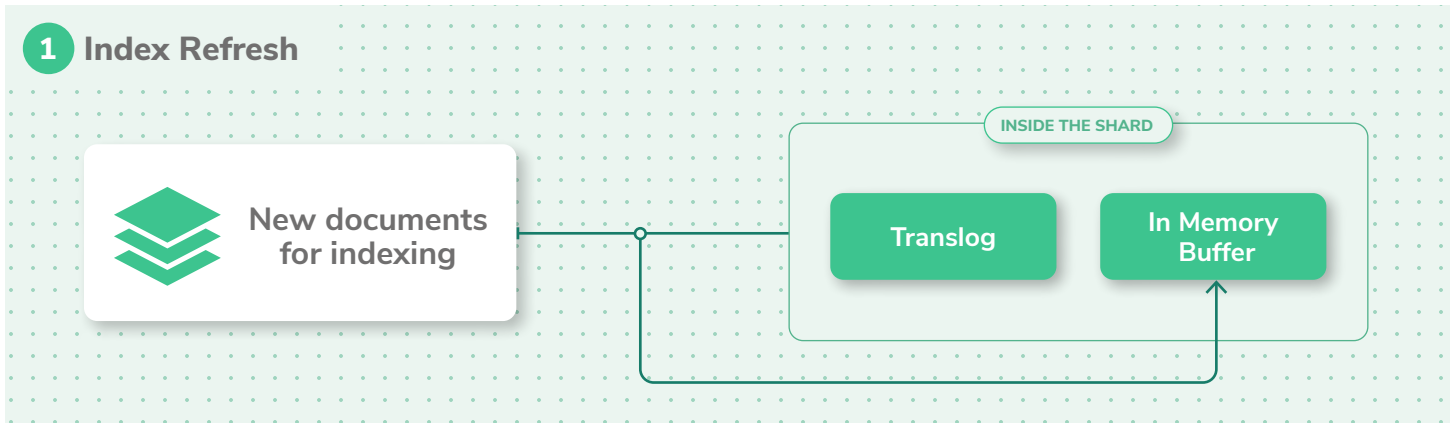
FETCH LATENCY

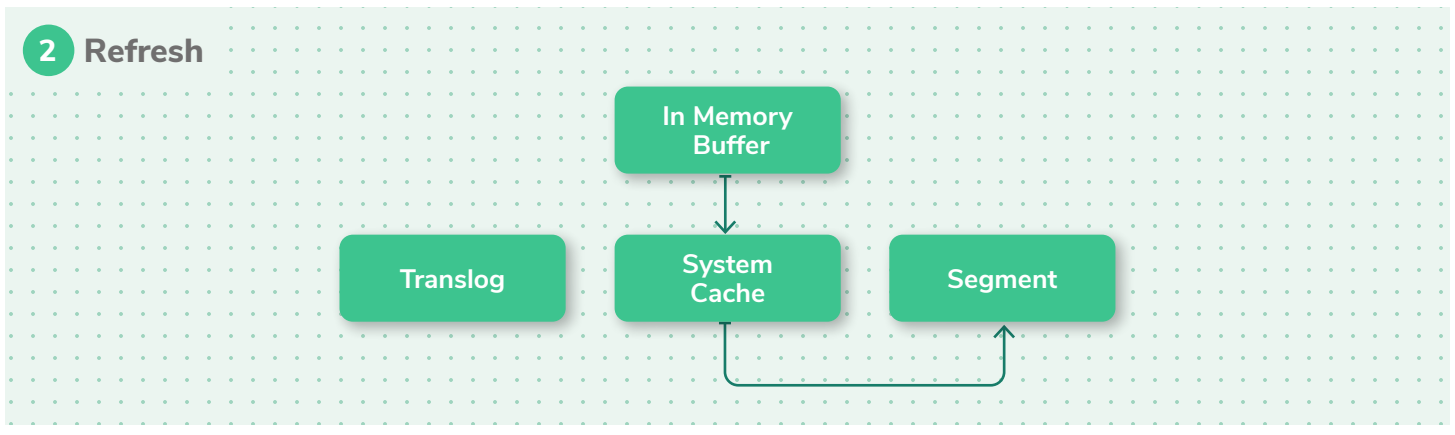
(Total fetch & time)

As explained above this is the second part of the search process and should normally take considerably less time than the query phase. This should provide a good indication on performance issues as a result of slow disks, enriching documents or requesting too many results.

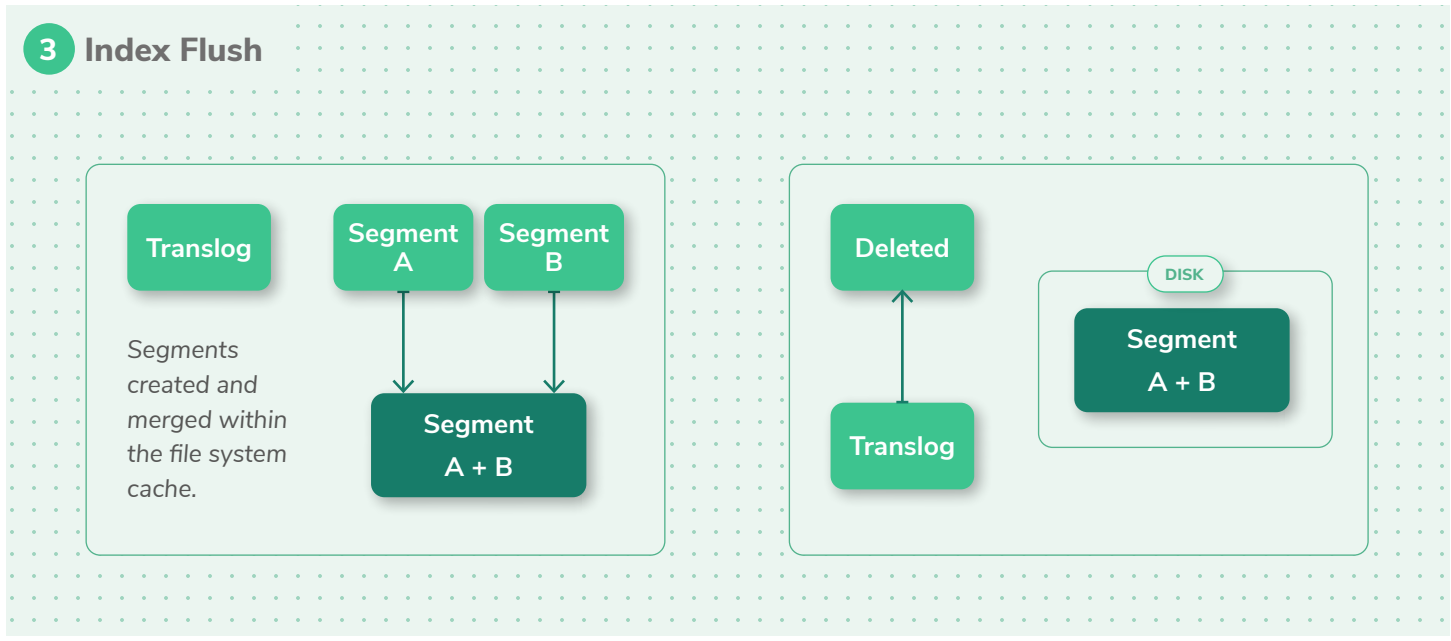
Index Performance

Indexing requests are effectively the same as write requests in traditional database platforms. For Elasticsearch workloads that are write-heavy, it's important to analyze how effectively your platform is able to update indices with new information. When new information is added to an index or existing data is updated or remote, each shared in the index is updated using two processes, refresh & flush. A newly indexed document is not immediately made available for search. They are first written to an in-memory buffer which is then processed as part of the next index refresh.





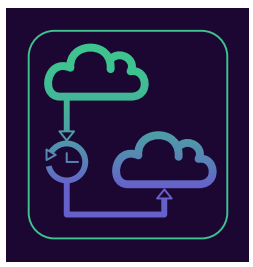
BASIC PERFORMANCE TUNING



METRICS TO MONITOR

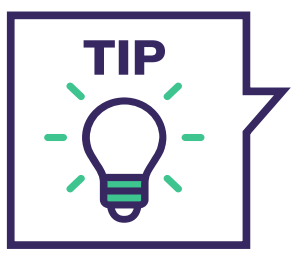
NAME	TYPE	DESCRIPTION
indices.indexing.index_total	Throughput	The total number of documents that have been indexed.
indices.indexing.index_time_in_millis	Performance	The total time spent indexing documents.
indices.indexing.index_current	Throughput	The number of documents that are currently being indexed.
indices.refresh.total	Throughput	The total number of index refreshes that have taken place.
indices.refresh.total_time_in_millis	Performance	The total time that has been spent refreshing indices.
indices.flush.total	Throughput	The total number of index flushes that have taken place to disk.
indices.flush.total_time_in_millis	Performance	The total time spent flushing indices to disk.

BASIC PERFORMANCE TUNING



INDEX LATENCY

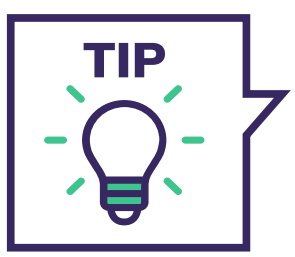
Elasticsearch does not directly expose this metric however it can be calculated using the `index_total` *and the* `index_time_in_millis metrics`.



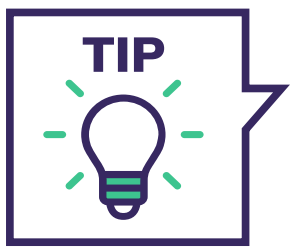
If your latency is increasing, your platform is likely indexing too many documents at one time.

FLUSH LATENCY

Data is not written to disk until a flush has been completed as a result monitoring the flush latency will provide valuable insight into your elastic performance. An increasing metric will likely indicate slow disks. In extreme cases, this could present you from being able to add new information to your index.



Try experimenting with the `index.translog.flush_threshold_size` *in the index's flush setting. The default size is set to 512 MB. When the tranlogs hits 512 MB of usage it is flushed. By increasing the flush threshold size, the Elasticsearch cluster creates a few large segments (instead of multiple small segments). Large segments merge less often, and more threads are used for indexing instead of merging, but as the merge process occurs this can cause a performance degradation if the merge is too large.*



Lowering the threshold size is better for systems with low indexing query volumes. You should look to change this setting inline with your cluster's use case.

➜ **NOTE: An increase in `index.translog.flush_threshold_size` will increase the time that it takes for a translog to complete. If a shard fails, recovery will take longer, because the translog is larger.**

Garbage collection (Memory management)

Elastic loves memory and getting the right amount is going to be key for a performant elastic stack. Elastic and Lucene will consume all available memory on your nodes in the following ways the JVM heap and the file system cache.

JVM HEAP TUNING & MONITORING

You should look to set the JVM heap to less than 50% of the available system memory and never go higher than 32 GB. As your system is restricted by the total memory available adjusting the JVM heap will reduce the available memory for Lucene. As a result, it is important to get this balancing impact correct. Setting too much memory to the heap will result in Lucene responding to requests slowly, however setting the JVM heap too small will result in out of memory errors or reduces overall application performance.

SYSTEM MEMORY	JVM HEAP	LUCENE	OTHER
NORMAL	40%	40%	20%
ADJUSTED	60% →	← 20%	20%

OVERRIDING THE DEFAULT JVM HEAP SIZE

The default configuration whilst installing elastic is 1 GB for the JVM Heap. This is simply too small for most deployments. Exporting the required size as an environmental variable will resolve this! You will need to restart Elastic for this to take effect.

```
$ export ES_HEAP_SIZE=10g
```

➜ **NOTE: You need to specify your memory argument using one of the letters “m” or “M” for MB, or “g” or “G” for GB. Your setting won’t work if you specify “MB” or “GB.”**

The other option is to set the heap size on start as below:

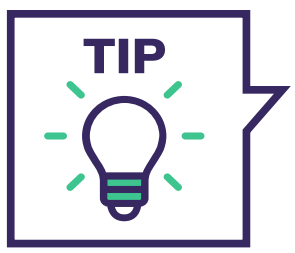
```
$ ES_HEAP_SIZE=”10g” ./bin/elasticsearch
```

You can verify the change has been applied with the below command:

```
$ curl -XGET http://<hostname>:9200/_cat/nodes?h=heap.max
```

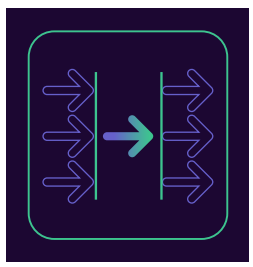
Garbage collection

To free up heap memory Elasticsearch uses a garbage collection process, but this process subsequently requires system resources to run! As a result it’s important to configure a frequency and run time that supports the workload of your cluster. It is highly recommended that Heap size not be more than half of the total memory. So if you have 64 GB of memory, you should not set your Heap Size to 48 GB. It is also not recommended to have a heap size of over 32 GB.



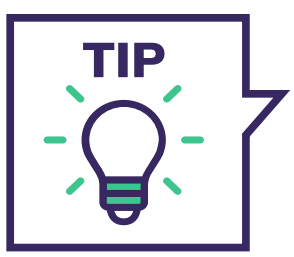
Setting your heap too large can result in excessive garbage collection times which can have a negative impact on your cluster and in some cases cause your cluster to incorrectly register a node as being offline. A node that has continatily higher than 85% JVM heap usage and CPU spikes following this will indicate that garage collection is causing a performance bottleneck. See below for more detail.

The garage collector also has multiple different collection algorithms that can be used. Elasticsearch has a default Garbage Collector of Concurrent-Mark and Sweep (CMS). Elasticsearch comes with very good default garbage collector settings. It is recommended that you do not change the collector, however it's worth understanding the other options. Other options include Serial Collector & Parallel Collector.



SERIAL GARBAGE COLLECTOR

Serial Collector is best-suited to single processor machines as it can't take advantage of multiprocessor hardware, although it can be useful on multiprocessors for applications with small data sets (up to approximately 100 MB).



Serial Collector is suited when only 1 CPU is available and no pause requirements exist, when using very small JVM's and small live data set (less than 100 MB). Using the serial collector with Elasticsearch can be devastating for performance.

BASIC PERFORMANCE TUNING

PARALLEL COLLECTOR

The parallel collector is also known as throughput collector, it's a generational collector similar to the serial collector. The primary difference between the serial and parallel collectors is that the parallel collector has multiple threads that are used to speed up garbage collection. The parallel collector is intended for applications with medium-sized to large-sized data sets that are run on multiprocessor or multi-threaded hardware. If you don't care about pause time and prefer to throughput, this collector would be best.

G1 GARBAGE COLLECTOR (DEFAULT)

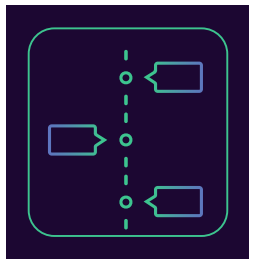
This server-style collector is for multiprocessor machines with a large amount of memory. It meets garbage collection pause-time goals with high probability, while achieving high throughput.

METRICS TO MONITOR

NAME	TYPE	DESCRIPTION
`jvm.gc.collectors.young.collection_count`	Other	The total number of young-generation garbage collections.
`jvm.gc.collectors.young.collection_time_in_millis`	Other	The total time spent processing young-generation garbage collections.
`jvm.gc.collectors.old.collection_count`	Other	The total number of old-generation garbage collections.

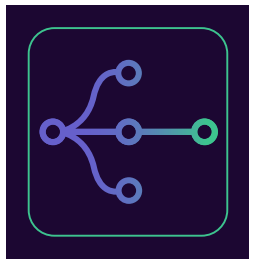
BASIC PERFORMANCE TUNING

NAME	TYPE	DESCRIPTION
jvm.gc.collectors.old.collection_time_in_millis	Other	The total time spent processing old-generation garbage collections.
jvm.mem.heap_used_percent	Utilization	The total percentage of the JVM heap that is currently in use.
jvm.mem.heap_committed_in_bytes	Utilization	The amount of JVM heap that has been committed.
indices.flush.total_time_in_millis	Performance	The total time spent flushing indices to disk.



GARBAGE COLLECTION & FREQUENCY

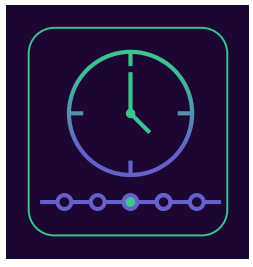
Whilst garbage collection is running there are periods where the node performing the task will halt. This results in the node being unable to perform any tasks. You should look for collections that exceed 30 seconds. After 30 seconds of collection time the master will believe that the node has failed and thus its important to monitor this metric.



JVM HEAP IN USE

At 75% JVM heap usage Elasticsearch will perform a garbage collection. Setting up an alert metric of 85% usage will highlight if nodes having issues with garbage collection. A node that often exceeds this limit is likely unable to keep up with the rate of garbage collection. This problem is addressed by increasing the nodes heap size or adding more nodes to the cluster.

BASIC PERFORMANCE TUNING



JVM HEAP USED COMPARED TO JVM HEAP COMMITTED

In a typical Elasticsearch cluster the amount of JVM heap in use will rise and fall indicating successful garbage collection. If the pattern begins to increase over a period of time this could indicate that garage collection is unable to keep up with the amount of objects being created on the node. The result is likely slow garbage collection times and this could result in nodes running out of memory. This metric will help you identify if you need to increase the amount of nodes or have disk performance issues with a node.

Compute Performance

A key performance indicator is the underlying host operating system. Items like Memory, CPU usage, Disk I/O & Network usage provide critical information into how a node is behaving within a cluster. Our previous metrics are likely to assist you in finding bottlenecks and these metrics will verify your conclusions.

MEMORY USAGE

Elasticsearch is very memory intensive and its ability to consume all available memory makes it complex to diagnose memory usage. Using rich monitoring that can provide details as to memory usage broken down by processes, will provide valuable data allowing you to further tweak your Elastic nodes for better performance.

METRICS TO MONITOR

NAME	METRIC TYPE
MEMORY USAGE	Utilization
CPU USAGE	Utilization
NETWORK USAGE	Utilization
DISK IO	Utilization

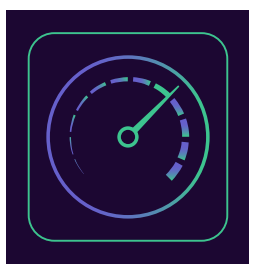
Cluster Health

The health of the cluster is of great importance. Any serious issues with the cluster will be detected with these metrics. Whilst not directly related to performance they provide valuable insight whilst applying tweaks to your Elastic cluster.

METRICS TO MONITOR

NAME	TYPE	DESCRIPTION
`cluster.health.status`	Other	The overall cluster Status
`cluster.health.number_of_nodes`	Availability	The total number of nodes
`cluster.health.initializing_shards`	Availability	The Number of initalizating shards
`cluster.health.unassigned_shards`	Availability	The total number of unassigned shards.

BASIC PERFORMANCE TUNING



CLUSTER STATUS

This metric provides feedback regarding the general health of the cluster. A yellow status indicates that at least one replica shard is unallocated or missing. Whilst searching is still possible another shard going offline may result in data loss.

If the cluster status is red this indicates that at least one of the primary shards is missing. The result of this is that the cluster is missing data. Whilst searches are still possible, it will only include data available to the cluster. The cluster will also be blocked from indexing into the missing shard.

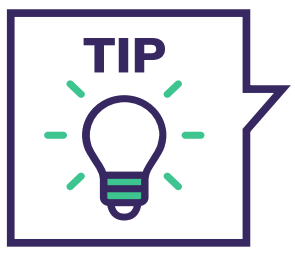
It is recommended to have a warning configured to alert on a yellow status that has been active for longer than 5 minutes. Moreover if the status has been red for more than a minute this should indicate a critical error warning.

INITIALIZING AND UNASSIGNED SHARDS

Whilst a node with shards on is rebooting or whilst first creating an index, the shards will display a status of "initializing". Once this process has completed the shard will transition to a status of "Started" or "Unassigned". The master node will attempt to assign the shard to the cluster during this period. An early warning sign of an unstable cluster is if you see a shard remaining in the initialization or unassigned state for too long.

Storage

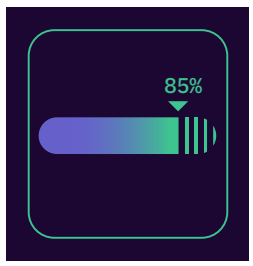
Your elastic cluster is heavily dependent upon the storage on which it resides. Setting your storage up wrong can have huge consequences and result in your cluster performing poorly. It's also important to make sure the storage is balanced amongst nodes to distribute the load across the nodes in the cluster.



Azure, AWS & GCP all offer shared storage solutions, do not use these services with Elastic. Use block level storage and always select the fastest disks your budget allows.

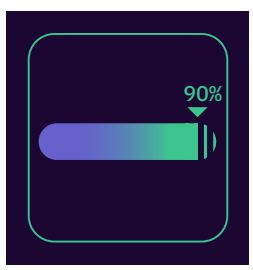
METRICS TO CONSIDER

When it comes to sizing your storage in the cluster it's important to understand the factors that contribute to disk utilization outside of the data you send.



LOW WATER MARK

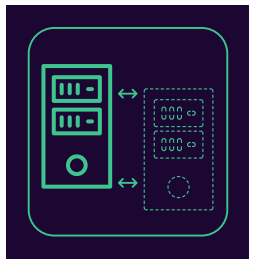
Should a nodes remaining disk space reach less than 15% then elastic will stop sending new shards to the node. It should be noted that whilst new shards are not being allocated, existing shards can still have data added to them.



HIGH WATER MARK

Should a nodes remaining disk space reach less than 10% then elastic will attempt to move shards away from the node. Elastic will move shards to other nodes with more space in an attempt to get the nodes available disk space

back to under 85% utilisation. Moving shards like this will consume much of the nodes resources and that of the nodes the shards are being reallocated to.



REPLICAS

Out of the box elastic is configured to have a single replica. Organizations might look to increase this to improve the failover capabilities of the cluster. Each replica is a full carbon copy of the index and thus it will consume the same amount of space.

SHARDS

The larger the shard the more efficient the shards ability to store indexes to a point. You will need to experiment with the number of shards in your cluster. Remember though when a node fails its shards are moved to another node's if a replica is available.

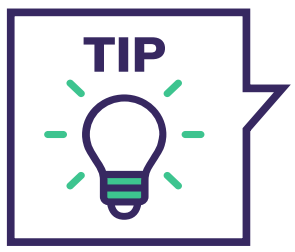
Resource saturation and errors

Elastic automatically configures resource pools which are used to distribute load across the clusters nodes, enabling multi-threaded processes maximumsing nodes CPU and memory. Elastic is pretty good at calculating the correct number of resource pools and thus it doesn't normally make sense to adjust this configuration. It does however make sense to monitor the metrics whilst tweaking your platform in the search of performance. Elastic uses the metrics 'queues' and 'rejections', monitoring these metrics will provide visibility into your nodes ability to

keep up. If you do find that nodes are unable to keep up you might need to add more to distribute the load.

THREAD POOLS & REJECTIONS

Each node in the cluster maintains different types of thread pools. The exact ones that will be relevant to your Elasticsearch cluster will depend on your use case. Generally you will want to watch search, merge and bulk, these correspond to the request type.



Starting from Elasticsearch version 7 the index thread pool will be removed. However you may want to monitor this pool if you're using a previous version of Elastic.

Each thread pool's queue size represents the number of requests that are waiting to be served while the node is at capacity. This queue is used to track and serve these requests on the node. It's used as a buffer instead of just discarding them. Thread pool rejections take place once the thread pool's maximum queue size is hit.

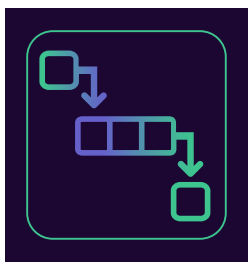
NAME	TYPE	DESCRIPTION

`thread_pool.search.queue`
`thread_pool.merge.queue`
`thread_pool.write.queue`
`(or thread_pool.bulk.queue*)`
`thread_pool.index.queue*`
	Saturation	The total number of threads in the thread queue pool

`thread_pool.search.rejected`
`thread_pool.merge.rejected`
`thread_pool.write.rejected`
`(or thread_pool.bulk.rejected*)`
`thread_pool.index.rejected*`
	Errors	The total number of errors in a thread pool

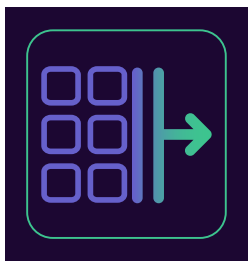
BASIC PERFORMANCE TUNING

METRICS TO MONITOR



THREAD POOL QUEUES

The larger the queues the more resources they use up. Moreover larger queues also introduce the chance of losing a request if a node failure occurs. Should you see the queues increasing steadily you might want to attempt to slow down the rate of requests, increasing the nodes processor count or adding more nodes to the cluster.



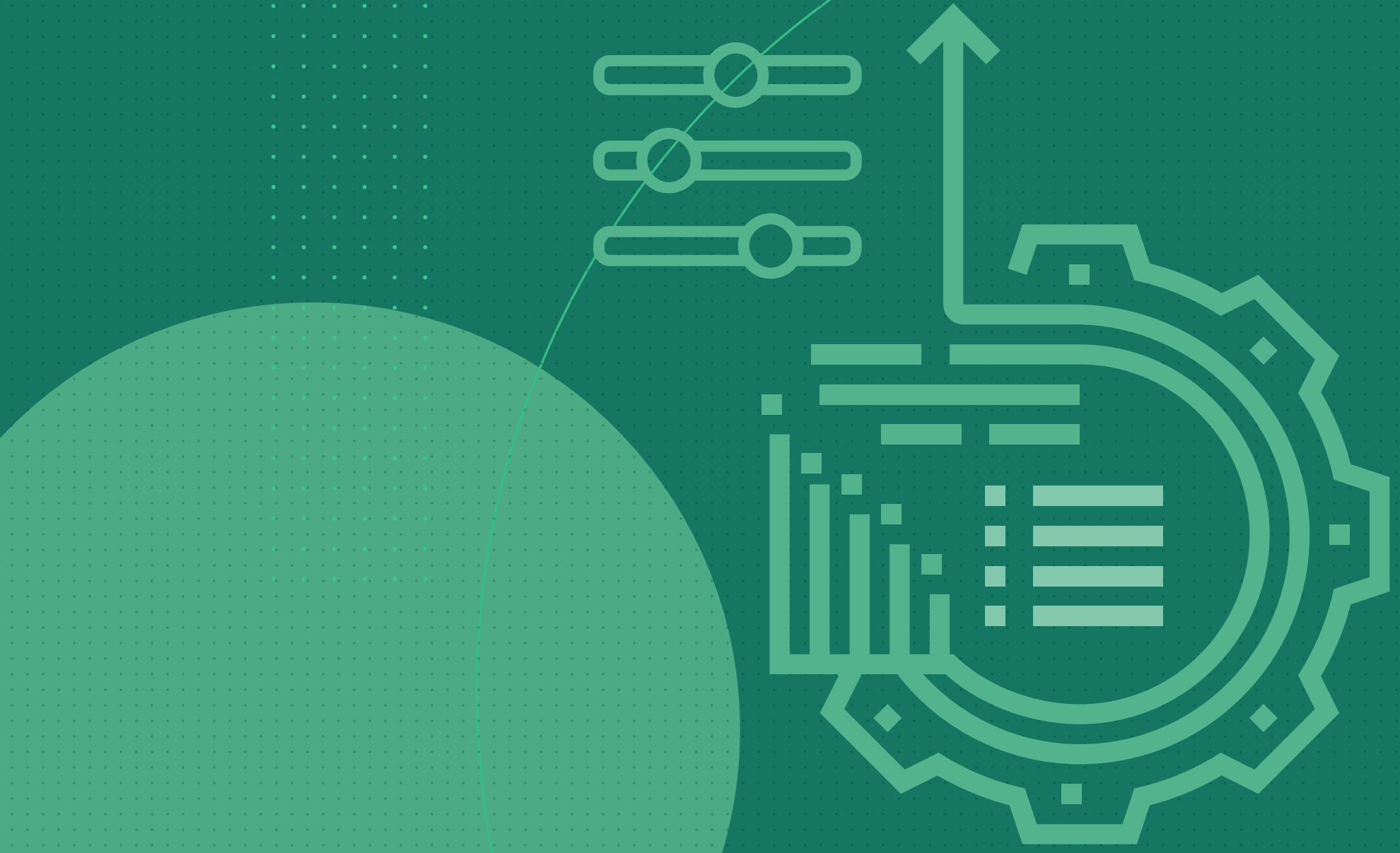
BULK REJECTIONS AND BULK QUEUES

Bulk operations offer a more efficient way to send multiple requests at one time. If you are looking to perform a large amount of actions (create an index, or add, update, or delete documents), then bulk operations are much better than multiple individual requests.

If you are seeing a large number of bulk rejections this is normally related to attempting to index too many documents in one large bulk request. Bulk rejections are not necessarily anything to worry about. You should however, attempt to implement a linear or exponential backoff strategy to deal with bulk rejections

<PART 3>

ADVANCED TUNING



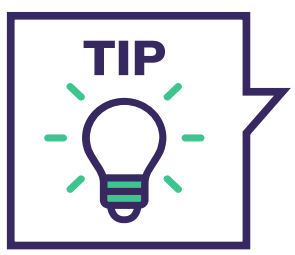
ADVANCED TUNING

Now that we have explored all of the essential elements to elastic, it's time to jump into the more complex areas that can be tuned to increase your elastic stacks performance. It's important to ensure the areas covered previously in this book are applied before using this section.

Catching slow or long running queries

Slow or long running queries eat up resources and if they are not providing meaningful results they can be completely unnecessary. We are going to explore how you can catch these queries as elastic provides no restriction or priority to the user performing the query.

It's important to have the correct logging enabled to catch these queries. You will want to ensure that you collect the heap dump after a 'out of memory' crash or the heap dump from the running JVM.



Elastic 7.0 introduced a new 'Circuit breaker strategy'. This provides the ability to measure real heap memory usage at the time when the memory was being reserved. This new strategy improves a nodes ability to deal with these queries. It's enabled by default and can be adjusted with the `indices.breaker.total.use_real_memory` setting.

Elastic provides some additional protection settings that can be applied to safeguard the cluster against running out of memory. This setting is the max bucket soft limit, it restricts the number of items that can be returned in a search to 10,000 by default in Elastic 7.0 and above. It can be amended using the `search.max_buckets setting.`

Additionally it is possible to narrow down long running queries with the 'Circuit breaker' setting in our tip. Setting `indices.breaker.request.limit` to a low threshold and gradually moving up will help you run though quieres and test their cost to resources of the cluster.

Tracking long running queries with Slow Logs

Long running queries can be reported on using Slowlogs which is a feature that can be enabled in elastic. The feature works only on data nodes as it works at shard level. This feature will help provide more detail around the speed of a query and the request body.

Sample Output from json logs:

```
{
  "type": "index_search_slowlog",
  "timestamp": "2030-08-30T11:59:37,786+02:00",
  "level": "WARN",
  "component": "i.s.s.query",
  "cluster.name": "distribution_run",
  "node.name": "node-0",
  "message": "[index6][0]",
  "took": "78.4micros",
```

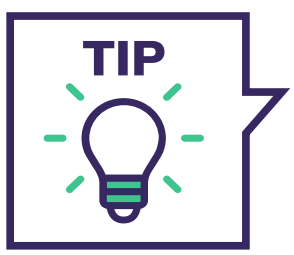
```
  "took_millis": "0",
  "total_hits": "0 hits",
  "stats": "[]",
  "search_type": "QUERY_THEN_FETCH",
  "total_shards": "1",
  "source": "{\"query\":{\"match_all\":{\"boost\":1.0}}}",
  "id": "MY_USER_ID",
  "cluster.uuid": "Aq-c-PAeQiK3tfBYtig9Bw",
  "node.id": "D7fUYfnfTLa2D7y-xw6tZg"
}
```

As you can see the json response provides great detail into the queries that are running and how long they took. The long running queries feature can be adjusted using the below curl request to the cluster:

```
curl -X PUT "localhost:9200/my-index-000001/_settings?pretty"
-H 'Content-Type: application/json' -d'
{
  "index.search.slowlog.threshold.query.warn": "10s",
  "index.search.slowlog.threshold.query.info": "5s",
  "index.search.slowlog.threshold.query.debug": "2s",
  "index.search.slowlog.threshold.query.trace": "500ms",
  "index.search.slowlog.threshold.fetch.warn": "1s",
  "index.search.slowlog.threshold.fetch.info": "800ms",
  "index.search.slowlog.threshold.fetch.debug": "500ms",
  "index.search.slowlog.threshold.fetch.trace": "200ms",
  "index.search.slowlog.level": "info"
}
'
```

Audit Logs

Whilst this feature is only included in clusters with a gold or premium subscription it is still worth exploring or activating if you are lucky to have the correct license! The audit logs feature provides much more context around the security side of the queries running, but this can provide valuable information whilst auditing queries. These logs will provide information around who requested the query, when it was requested and some context around the query itself.



Activating audit logs with the default setting will cause excessive data to be logged. Ensure you modify your setting to match the needs of your organization. It is recommended that these logs should only be enabled whilst diagnosing issues.

ENABLING AUDIT LOGS

The following setting will need to be applied to the elasticsearch.yml configuration file.

To enable the audit logs feature add

```
xpack.security.audit.enabled: true
```

To enable log or security audits in the output add

```
xpack.security.audit.outputs:[logfile, index]
```

ADVANCED TUNING

To enable authentication successes in the output logs add

```
xpack.security.audit.logfile.events.include:
authentication_success
```

NOTES:

- **Xpack.security.audit.outputs only applied to version 6-6.2. Version 7.0 ignores this setting and defaults to json output when xpack.security. audit.enabled is true.**

- **Audit mode can be very noisey resulting in excessive generation of logs. Only use this feature whilst remediating an issue with the cluster. Once you have completed your remediation ensure this feature is disabled.**

- **It is recommended to choose logfile over index due to the verbosity of the audit logs and the stress that it will put on the cluster resulting in performance degradation.**

- **Audit logs will provide a very detailed picture of the queries that are running, including the user account that ran them. This will help you identify your slow running queries and adjust as required.**

CONCLUSION

In this book we have explored how you can monitor the metrics of your Elastic cluster and adjust settings to exploit the very best performance from your elastic cluster.

Tuning Elastic is a blend of settings tweaks, hardware adjustments and software adjustments.. This book has covered the fundamental basics to get your cluster into the best possible state, but expect to continue to evolve your Elasticsearch cluster over time. Elasticsearch requires maintenance regularly to keep it working at its optimum!

Start solving your
production issues faster

Managed, scaled,
and compliant monitoring,
built for CI/CD