elastic

wp2ss2xAxTY5UnRUdgo2tUgW

Why Kibana ?

Easiest way of running queries is using Kibana tool. It also sends requests to Elastic search API.

It formats the response for us and makes it easier to read.

It also sets the correct-content type header.

Auto typing.

Postman / Curl can be used.

1. GET /\_cluster/health

\_cluster is the api

health is the command

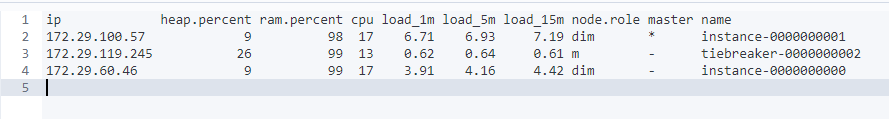
2. \_cat api which outputs data in a human readable format.

\_cat/nodes : all nodes in the cluster

command :

GET /\_cat/nodes?v

v : query parameter : instructs elastic search to include a descriptive header in the output- to identify each peice of info

o/p:  


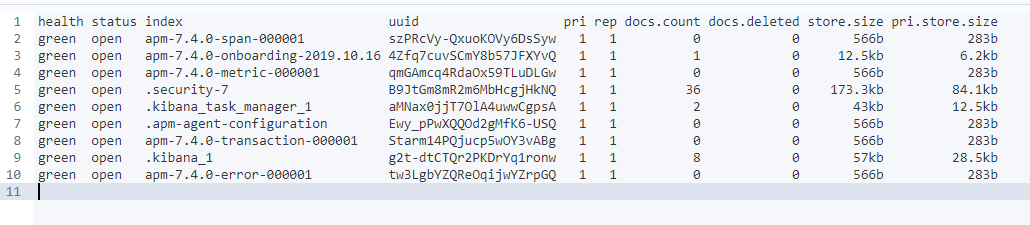
3. GET /\_nodes/stats

API : nodes

To inspect nodes in a lot of detail.  
  


4. GET /\_cat/indices?v

curl -XGET "http://24e374bd44bb4bf884cfd220f7f55cf0.containerhost:9244/\_cat/indices?v"

List all indices  


If its elastic cloud : we can send elastic search requests with the CURL http client

curl –XGET –u elastic: wp2ss2xAxTY5UnRUdgo2tUgW ‘<https://24e374bd44bb4bf884cfd220f7f55cf0.ap-southeast-1.aws.found.io:9243/.kibana/_serch> –H “Content-Type: application/json” –d { “query” : { “match\_all”:{} }}’

5. Sharding and scalability.

Elastic search has a cluster with nodes.

Each node has a capacity and you can store 200gb of data in 2 nodes each having 100gb of data.

Sharding : Makes it possible to scale the amount of documents we can store.

It’s a way to divide indices into small pieces called shard[any number of shards] – done at index level.  
 Indices could contain a few hundred records to a million records.

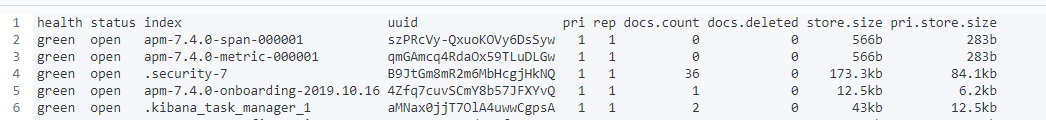
To horizontally scale the data volume indices are divided into shards.

If an index has 5 shards : There is no need that all shards be in different nodes – they could be in the same node as well.

Search query on an index can be run independently on different nodes – increasing the speed of output [as shards could be stored in different nodes.].

Sharding : it is a process to sub divide an index in smaller pieces.

: it increases the number of documents an index can store.

6. Get /cat/indices?v  
  


Pri – primary shard : the number of shards a given index has.

spilt api : to increase the number of shards in an index.

shrik api : to reduce the number of shards for an index.

7. Replication.

- it is configured at the index level

- Copies of shards are created – replica shards.

- When an index is created we can choose how many replicas of each shard we want. 1 being the default.

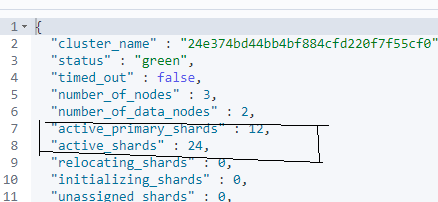
- Replica shard is never stored in the same node.

- Throughput can be increased as the query is distributed.

8. snapshot: to store an index completely.

Used for daily backup, manual backups are taken just before applying changes – just to be sure the changes are good.

9. Create and Index: PUT /indexname

10. Check the cluster Health: GET /\_cluster/health  


The cluster health is **green** as there is replication of shards.

11. check the shards : GET /\_cat/shards?v  


p- primary shard.

r – replica shard.

state = STARTED: both primary and replica shards are available for requests.

12. different roles of a node.

Master node :

Data node :

Ingest Node : This enables a node to run Ingest pipelines.

Node.ml : identifies a node as machine learning

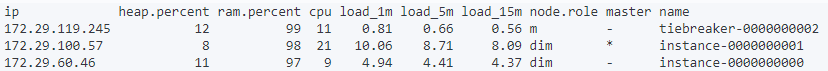
Xpack.ml.enabled : Determines if the node should respond to machine learning requests.

Co-ordination node : how ES process a request and delegates it internally to data nodes[by removing other roles – no role exists., kind of a load balances].

13. Roles of our nodes ?

Command : GET /\_cat/nodes?v

o/p :



dim – data ingest and master

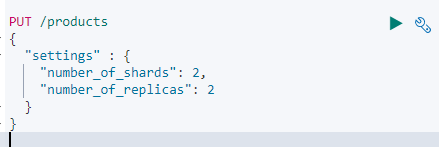
m - master

14. ES exposes a rest API.

- HTTP verb is important to perform a particular action.

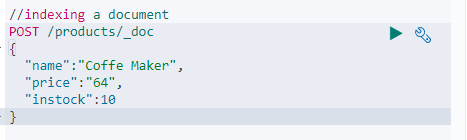
Delete an index :   
  
Create an index by specify the number of shards and replicas. [ For indices that we use to create for production purposes we should stick to the default values]  
first line : http verb + end point

Other lines : json request body AND index settings should be passed as a json object



15. INDEXING A DOCUMENT BY sending a request to an end point.

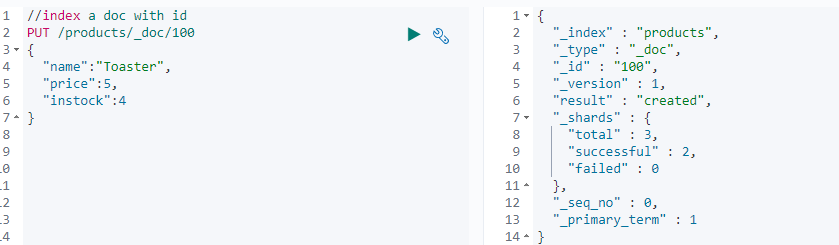
*- we need to define the document within the request body as a json object*

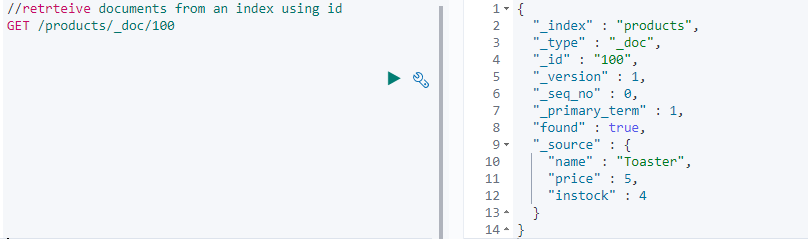


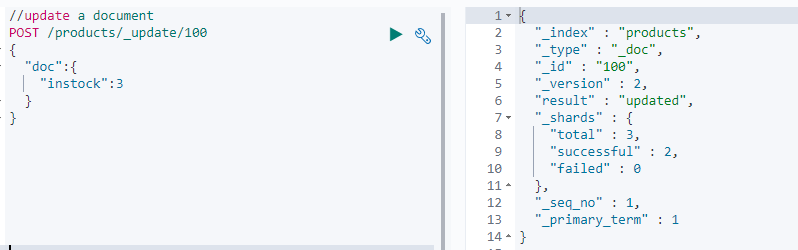
o/p :   
  
  
\_shards : The number of shards in which you successfully stored a document.  
In step 14 : replics are 2, so document was added to primary and also to the 2 replicas so total is 3.

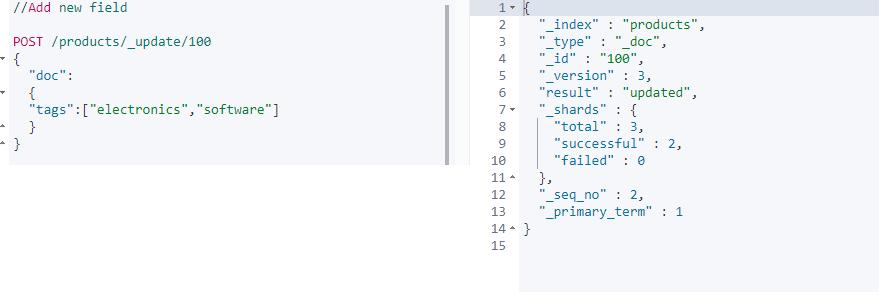
\_id : was created automatically but we can add that too.

16. Add \_id while indexing a document.

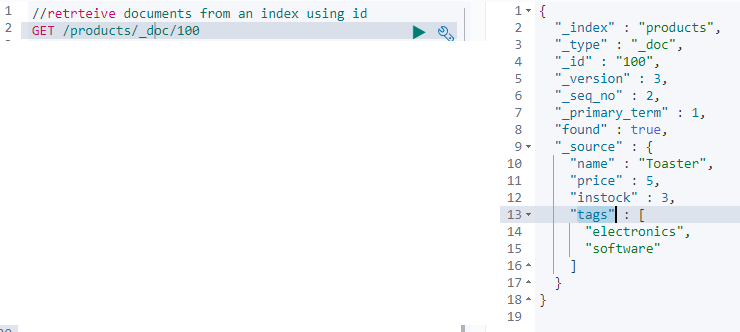
To specify an id : we need to change the http verb to **PUT** as this is a convention for REST api.  
  


17. Retrieving documents from an index.  


18. Updating documents: by sending a update request   


19. Adding new fields to existing documents.  


Now we see the new field with data.

****

How this works internally.

* ES documents are immutable (!).
* The \_update api : retrieved the document/changed its fields/re-indexed the document with the same id.

20. Scripted updates

- There is a use case where you retrieve a document first + update a field value + update the document.

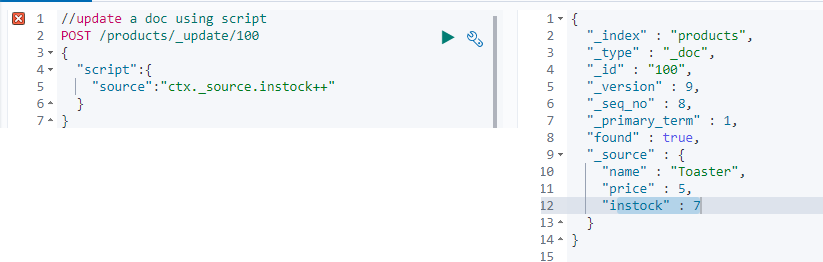
- all of this can be done in one go.

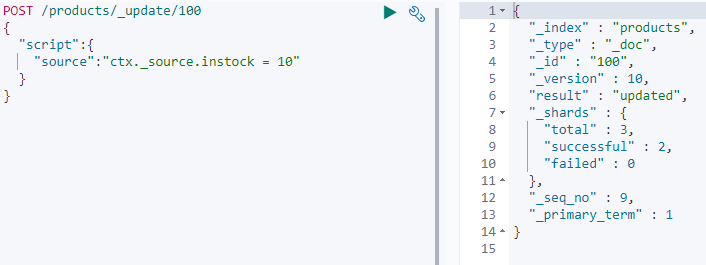
- with scripting : we can use if statements

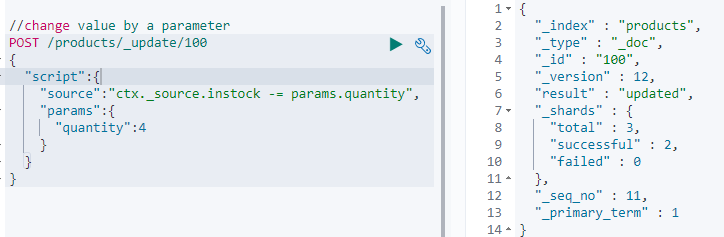
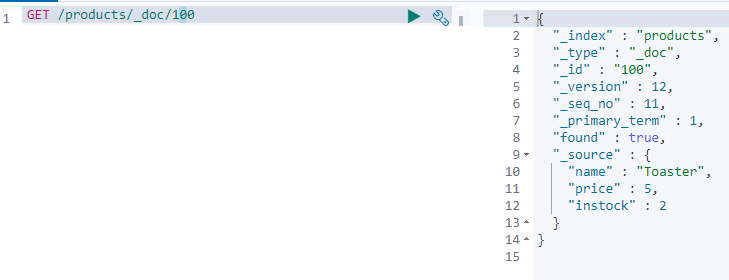
- use the \_update api + script update with the request body

Example :

ctx: it is a ES variable and short for context.

\_source : access the source document using the \_source property, which gives us an object containing the objects fields.  
  


* We can do assignments also apart from updating a field.  
    
  
* Now if some customer purchases 4 products, the application must send a request to reduce the instock count by 4.
* We use this by writing a params object and using its value to reduce the count

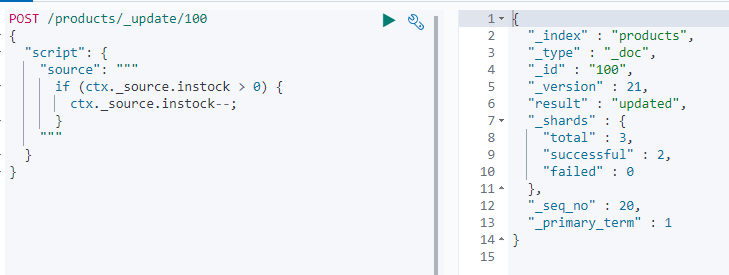
  


* If you try to update a field value with its existing value : result will be “no-op” else updated
* If the field value is set as part of a script the ”result” will always be as “updated”
* There are 2 exceptions to this – both being if we explicitly set the operation within the scripts.

Case 1 : A script to ignore a document based on a condition – by setting the **op** property on the **ctx** variable to “noop”.



What the above script does is reduce the instock field value in the doc by 1 and set the resultkeyto ‘updated’; if it is zero the **result** key will be set to a value of ‘noop’

Case 2 :  


Ir-respective of the execution of the if block : the result will always contain “updated”

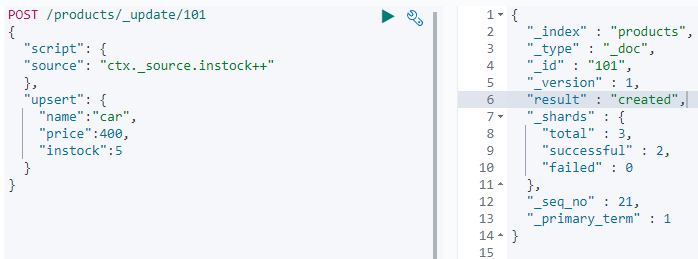
If you want to detect if nothing was changed then : follow case 1.

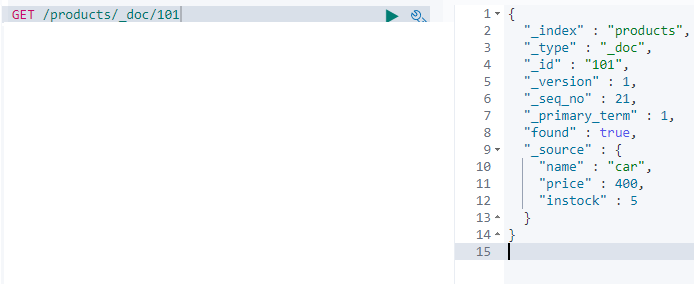
Case 3 : We can set the operation to delete which will cause the document to be deleted.  
This will set the result key to deleted with in the results.

**ctx.op = ’delete’;**

21. Updating documents using upserts

Insert if not present or else update it.

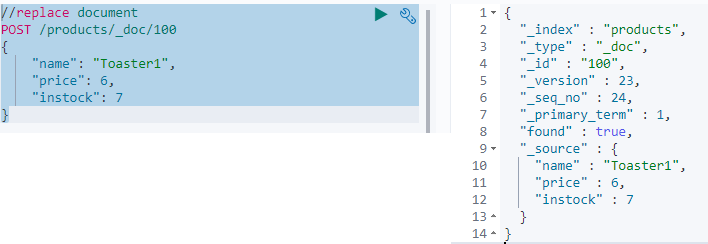


The instock is 5 after creation.  


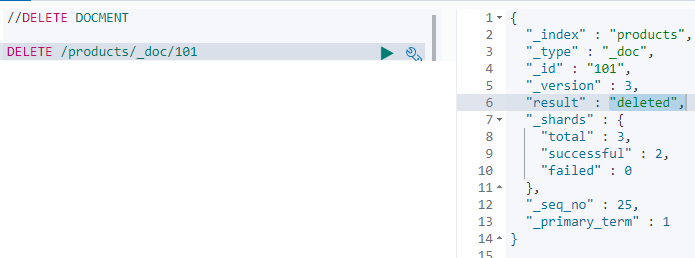
If the POST query is run again : **"result" : "updated".**

So, The script is run if the document exists else the upsert is run.

22. Replacing documents.



As the whole document was replaced if any other field existed in the old document they will not still exist.

23. DELETE documents  


24. How ES reads data ?

- its about reading a single document.

- a given node receives the read request.

- this node is responsible for co-ordinating the request – so its called the co-ordinating node.

- locate where the document is stored -> that’s done with routing.

- with Routing : Routing resolves to a shard that stores a given document.

- i.e : It resolves to a primary shard or a replication group [contains the list of primary shards + its replica groups]

- Note : If elastic search directly retrieved the document from primary shard – all retrievals will end up in the same shard – which does not scale well : INSTEAD a shard is chosen from the replication group.

- So a technique called ARS : Adaptive replica selection is used.

- ES tries to select a shard copy it believes that can yield the best performance.

- then the co-ordinating node send the read request to that shard… and so on.

25. How ES writes data

- the request is resolved to a replication group that stores/SHOULD STORE the document.

- the write is always routed to the primary shard

- It validates the field values and structure of the request.

- The write op is performed in the primary shard before it fwds to the replica shards (this could be in parallell).

- Note that the operation succeeds even if the operation cannot be replicated to the replica shards.

> Since elastic search is distributed and many tasks happen asynchronously many things can go wrong.  
> When a document is indexed – the primary shard validates the operation and indexes the document locally  
> Assume there are 2 replicas in the replication group, then the primary shards send the operation to these two replicas.  
> Let us assume the operation reaches only 1 replica shard (RS1) – as the primary shard (PS) goes down due to a hardware failure.[PS RS1 RS2]  
> When this happens ES goes through a recovery process – In which one of the replica shard will be promoted as the new Primary shard [each RG must have a primary shard.]  
> Only one replica shard indexed the new document and other did not.  
> RS 2 thinks it is up to date – but this is not the case.  
> The new document will be found only half of the time – depending on which shard serves the request.

SO HOW ES solve this : primary terms + sequence numbers.

Primary terms (PT) :   
A way to distinguish between old and new primary shards, when the primary shard of a replication group has changed.  
The primary term for a replication group is essentially just a counter about how many times the primary shard has changed. [1 or 2 or 3]  
In the above case : The Primary Term for the RG will be increased by 1 as the primary shard failed and one of the replication shard was promoted to be the primary shard.  
The Primary terms for all Replication groups are persisted in the cluster’s state.  
When write operations are performed : the current PT is appended to the operations that are sent to the replica shards.  
This enables the RS to tell if the PS was changed since the operation was forwarded.

**Sequence Number**  
Apart from associating each operation with a primary term a **sequence number** is given to each operation.

This is also just a counter that is incremented for each operation – until the PS changes.

The primary shard is responsible for increasing this number when it process a write request.

These sequence numbers help ES to know in which order write operations happened, on a given primary shard.

Thus instead of comparing data on the disk – we can use PT and SN to figure out which operations have already been performed + which are needed to bring a given shard up to date.

[global checkpoints and local checkpoints are also imp]  
  
**Global checkpoints** : exists for each replication group.  
**Local Check Points** : kept for each replication shard.

It is the active sequence number that all of the active shards with in a replication group have been aligned atleast up to.

This means that any operations containing a sequence number lower than the global check point have already been performed on all shards with in the RG.

If a primary shard fails and re-joins the cluster at a later point : ES only need to compare the operations that are above the global check point that it last knew about.

Like wise : if a replica shard fails only the operations that have a sequence number higher than its local checkpoint need to be applied when it comes back.

THIS MEANS TO RECOVER ES JUST NEEDS TO COMPARE THE OPERATIONS WHILE THE SHARD WAS GONE , INSTEAD OF THE ENTIRE HISTORY OF THE REPLICATION GROUP.