Logstash:

input -> reading from multiple sources

filter -> grok

output/storing -> filter data storing

Elastic Search:

Store

retrival

->ElasticHQ plugin to monitoring and troubleshooting

bigdesk -> monitoring

elasticsearch-head plugin for chrome

start elastic:

go to C:\elastic\_stack\elasticsearch-7.5.2\bin

Run the service:

>elasticsearch

From browser:

http//: localhost:9200

From cygwin:

* Curl -XGET 'localhost:9200'
* Curl –XGET ‘localhost:9200/nodes?pretty’
* curl -XPUT 'localhost:9200/test'

Index has multiple documents, document will be divided and stored as in sigments in shards.

* Segments are lucene index.

**Health check**:

$ curl -XGET 'localhost:9200/\_cat/health'

1581313837 05:50:37 elasticsearch yellow 1 1 2 2 0 0 2 0 - 50.0%

Administrator@DESKTOP-9A5G7S5 ~

$ curl -XGET 'localhost:9200/\_cat/health?v'

* Health color :yellow -> some replicas are not yet allocated.

For node details of elastic :

$ curl -XGET 'localhost:9200/\_cat/nodes?v'

To get the indices:

$ curl -XGET 'localhost:9200/\_cat/indices?v'

Ex: bank/chennai/2

**bank- index, chennai – type, id – 1;**

Create index:

$ curl --header 'content-type:application/JSON' -XPUT 'localhost:9200/test/external/1?pretty' -d '{"name":"eshan"}'

{

"\_index" : "test",

"\_type" : "external",

"\_id" : "1",

"\_version" : 1,

"result" : "created",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 0,

"\_primary\_term" : 2

}

To access: localhost:9200/bank/external/1

To access data of index:

$ curl -XGET 'localhost:9200/test/external/1/\_source?pretty'

{

"name" : "eshan"

}

* To use the sql based help
* Go to bin directory and run the below command

$ elasticsearch-sql-cli

sql> desc test;

column | type | mapping

---------------+---------------+---------------

name |VARCHAR |text

name.keyword |VARCHAR |keyword

from cygwin run the below command:

$ curl -H 'content-type:application/JSON' -XPOST 'localhost:9200/\_xpack/sql?format=txt' -d '{"query":"select \* from test"}'

name

---------------

eshan

Converting sql data into elastic based data:

$ curl -H 'content-type:application/JSON' -XPOST 'localhost:9200/\_xpack/sql/translate?format=txt&pretty' -d '{"query":"select \* from test"}'

{

"size" : 1000,

"\_source" : {

"includes" : [

"name"

],

"excludes" : [ ]

},

"sort" : [

{

"\_doc" : {

"order" : "asc"

}

}

]

}

To update the document/record command:

$ curl --header 'content-type:application/JSON' -XPOST 'localhost:9200/test/external/1/\_update?pretty' -d '{"doc":{"name":"Tanish"}}'

{

"\_index" : "test",

"\_type" : "external",

"\_id" : "1",

"\_version" : 2,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 1,

"\_primary\_term" : 2

}

* To delete the document/record:

$ curl --header 'content-type:application/JSON' -XPOST 'localhost:9200/test/external/\_delete\_by\_query?pretty' -d '{"query":{"match":{"name":"Tanish"}}}'

{

"took" : 133,

"timed\_out" : false,

"total" : 1,

"deleted" : 1,

"batches" : 1,

"version\_conflicts" : 0,

"noops" : 0,

"retries" : {

"bulk" : 0,

"search" : 0

},

"throttled\_millis" : 0,

"requests\_per\_second" : -1.0,

"throttled\_until\_millis" : 0,

"failures" : [ ]

}

* Updating the record:

$ curl -H 'content-type:application/JSON' -XPOST 'localhost:9200/test/external/3/\_update?pretty' -d '{"doc": {"name":"jhon", "age":23}}'

{

"\_index" : "test",

"\_type" : "external",

"\_id" : "3",

"\_version" : 1,

"result" : "noop",

"\_shards" : {

"total" : 0,

"successful" : 0,

"failed" : 0

},

"\_seq\_no" : 5,

"\_primary\_term" : 2

}

* To update the particual column records:

$ curl --header 'content-type:application/JSON' -XPOST 'localhost:9200/test/external/1/\_update?pretty' -d '{"script":"ctx.\_source.age+=5"}'

{

"\_index" : "test",

"\_type" : "external",

"\_id" : "1",

"\_version" : 3,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 6,

"\_primary\_term" : 2

}

* Adding bulk record into Elastic search:

$ curl --header 'content-type:application/JSON' -XPUT 'localhost:9200/test/external/\_bulk' -d'

{"index":{"\_id":"2"}}

{"name":"Paresh", "age":34}

{"index":{"\_id":"3"}}

{"name":"Minal","age":35}

'

{"took":61,"errors":false,"items":[{"index":{"\_index":"test","\_type":"external","\_id":"2","\_version":1,"result":"created","\_shards":{"total":2,"successful":1,"failed":0},"\_seq\_no":7,"\_primary\_

term":2,"status":201}},{"index":{"\_index":"test","\_type":"external","\_id":"3","\_version":2,"result":"updated","\_shards":{"total":2,"successful":1,"failed":0},"\_seq\_no":8,"\_primary\_term":2,"sta

tus":200}}]}

* Multiple operation can be done :

$ curl --header 'content-type:application/JSON' -XPUT 'localhost:9200/bank/external/\_bulk' -d'

> {"index":{"\_id":"4"}}

> {"name":"Kunal","age":54}

> {"update":{"\_id":"1"}}

> {"doc": { "name": "John Doe becomes Jane Doe" } }

'> {"delete":{"\_id":"2"}}

> '

{"took":804,"errors":true,"items":[{"index":{"\_index":"bank","\_type":"external","\_id":"4","\_version":1,"result":"created","\_shards":{"total":2,"successful":1,"failed":0},"\_seq\_no":0,"\_primary\_

term":1,"status":201}},{"update":{"\_index":"bank","\_type":"external","\_id":"1","status":404,"error":{"type":"document\_missing\_exception","reason":"[external][1]: document missing","index\_uuid"

:"ttLcWsbZQgOM0ucoXdxGEg","shard":"0","index":"bank"}}},{"delete":{"\_index":"bank","\_type":"external","\_id":"2","\_version":1,"result":"not\_found","\_shards":{"total":2,"successful":1,"failed":0

},"\_seq\_no":1,"\_primary\_term":1,"status":404}}]}

* Bulk insertion using json file:

$ curl --header 'content-type:application/JSON' -XPUT 'localhost:9200/account/\_bulk?pretty' --data-binary @C:\\accounts.json

**Kibana:**

**Run:**

C:\elastic\_stack\kibana-7.5.2-windows-x86\_64\bin>kibana.bat

Access from brower: <http://localhost:5601/app/kibana>

Go to dev tools option : (run the below query to find all the records of account index)

GET /account/\_search

Creating new index from exiting one:

POST \_reindex

{

"source": {

"index": "bank"

},

"dest": {

"index": "new\_bank"

}

}

POST \_reindex

{

"source":{

"index": "account",

"query": {

"term": {

"balance": 120020

}

}

},

"dest": {

"index": "find\_rec"

}

}

Salary ordering:

GET /account/\_search

{

"sort":[

{

"balance": {"order": "desc"}

}]

}

Get only one record:

GET /account/\_search

{

"size": 1,

"sort":[

{

"balance": {"order": "desc"}

}]

}

Get the record by the below source:

GET /account/\_search

{

"size": 1,

"sort":[

{

"balance": {"order": "desc"}

}

],

"\_source": ["firstname", "balance" ]

}

* Create new index with top 10 records from exiting index:

POST \_reindex

{

"size": 10,

"source": {

"index": "account",

"sort": [

{

"balance": {

"order": "desc"

}

}

],

"\_source":["firstname", "balance"]

},

"dest": {

"index": "new\_account4"

}

}

Finding the records between the range:

POST /account/\_search

{

"query": {

"range": {

"balance": {

"gte": 49355,

"lte": 49567

}

}

}

}

* Update the index with particualr constraints:

POST /new\_account4/\_update\_by\_query

{

"script": {

"source": "ctx.\_source.balance=0"

},

"query": {

"range": {

"balance": {

"gte": 40000,

"lte": 50000

}

}

}

}

Curl command from cygwin tool:

curl -XGET 'localhost:9200/account/\_search?pretty'

From browser:

[http://localhost:9200/account/\_search?pretty&q=\*](http://localhost:9200/account/_search?pretty&q=*)

From devtools :

GET account/\_search

{

"query": {"match\_all": {}}

}

From the browser Find the record based on Name:

<http://localhost:9200/account/_search?pretty&q=firstname:Amber>

**Get 2 records:**

GET account/\_search

{

"query": {"match\_all": {}},

"size": 2

}

**Get the records on pagination:**

GET account/\_search

{

"query": {"match\_all": {}},

"from": 10,

"size": 10

}

**Sorting the records:**

GET account/\_search

{

"query": {"match\_all": {}},

"sort": [

{

"firstname.keyword": {

"order": "desc"

}

}

]

}

**Find the records where the bala?**

GET account/\_search

{

"query": {

"match": {

"balance": 22776

}

}

* Match: with single word and muliple words
* Match\_phrase : for muliple words want to combine
* Match\_all: for all columns

GET account/\_search

{

"query": {

"match\_phrase": {

"address.keyword": "984 Woodside Avenue"

}

}

}

GET account/\_search

{

"query": {

"prefix": {

"firstname.keyword": {

"value": "A"

}

}

}

}

GET account/\_search

{

"query": {"match\_all": {}},

"from": 10,

"size": 10

}

GET account/\_search

{

"query": {"match\_all": {}},

"sort": [

{

"firstname.keyword": {

"order": "desc"

}

}

]

}

* Display the detilas of account holder whoes names starts with “Y” and having max balance in “y” starting names

GET account/\_search

{

"query": {

"prefix": {

"firstname.keyword": "Y"

}

},

"sort":[

{

"balance": {"order": "desc"}

}

]

}

Elastic HQ Analyasis plugin for elastic

Installation process of HQ

Download or clone the repository.

Open terminal and point to root of repository. Type: pip install -r requirements.txt

Run server with: python3 application.py. Alternatively: ./manage.py runserver

Access HQ with: <http://localhost:5000>

* Get the segments from dev tools:

GET \_cat/segments

* Creating index

POST employee/hr/2

{

"age": 23

}

* Find by Range:

GET employee/hr/\_search

{

"query": {

"range": {

"age": {

"gte": 30,

"lte": 40

}

}

}

}

* To be able to use those characters as actual parts of the url (without using them for the curl url ‘globbing system’) use –g or –globoff options.

$ curl -XGET --globoff 'localhost:9200/employee/hr/\_search?q=age:["30"+TO+"40"]&pretty=true'

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 1,

"successful" : 1,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : {

"value" : 1,

"relation" : "eq"

},

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "employee",

"\_type" : "hr",

"\_id" : "5",

"\_score" : 1.0,

"\_source" : {

"age" : 34

}

}

]

}

}

<https://github.com/elastic/logstash/blob/v1.4.2/patterns/grok-patterns>

Logfile:

127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] "GET /xampp/status.php HTTP/1.1" 200 3891 "http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"

Grok pattern:

%{COMBINEDAPACHELOG}

Bool filter:

Must: AND – ALL OF THESE clauses must match

Must\_not: NOT - all of these clauses must not match

Should: OR – at least one of these clauses must match

* Select \* from account where state like% ‘MD’ and age >20 & age<30; can be writeen as below in devtools:

GET account/\_search

{

"query": {

"bool": {

"must": [

{

"term": {

"state.keyword": {

"value": "MD"

}

}

},

{

"range": {

"age": {

"gte": 20,

"lte": 30

}

}

}

]

}

}

}

* Using Must\_not query:

GET account/\_search

{

"query": {

"bool": {

"must\_not": [

{

"term": {

"state.keyword": {

"value": "MD"

}

}

},

{

"range": {

"age": {

"gte": 20,

"lte": 30

}

}

}

]

}

}

}

* With sorting and limited fields data:

Select \* from account where state like% ‘MD’ and age >20 & age<30 and order by age descending ,,,,....

GET account/\_search

{

"query": {

"bool": {

"must": [

{

"term": {

"state.keyword": {

"value": "MD"

}

}

},

{

"range": {

"age": {

"gte": 20,

"lte": 30

}

}

}

]

}

},

"sort": [

{

"age": {

"order": "desc"

}

}

]

, "\_source": ["firstname", "age", "state"]

}

* SELECT product FROM products

WHERE (price = 20 OR

productID = "XHDK-A-1293-#fJ3")

AND (price != 30)

"bool" : {

“must”:{

"should" : [

{ "term" : {"price" : 20}},

{ "term" : {"productID" : "XHDK-A-1293-#fJ3"}}

],

"must\_not" : {

"term" : {"price" : 30}

}

}

}

* Aggregation:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword"

}

}

}

}

* Group by gender:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "gender.keyword"

}

}

}

}

* Aggregation by state and innner aggregation by balance:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "gender.keyword"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

->

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword",

"order": {

"\_term": "desc"

}

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

* Label using for ordering in inner aggregation:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword",

"order": {

"average\_balance": "asc"

}

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

* Aggregation with range:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

}

}

}

}

Output:

{

"took" : 1,

"timed\_out" : false,

"\_shards" : {

"total" : 1,

"successful" : 1,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : {

"value" : 1000,

"relation" : "eq"

},

"max\_score" : null,

"hits" : [ ]

},

"aggregations" : {

"group\_by\_age" : {

"buckets" : [

{

"key" : "10.0-20.0",

"from" : 10.0,

"to" : 20.0,

"doc\_count" : 0

},

{

"key" : "20.0-30.0",

"from" : 20.0,

"to" : 30.0,

"doc\_count" : 451

},

{

"key" : "30.0-40.0",

"from" : 30.0,

"to" : 40.0,

"doc\_count" : 504

}

]

}

}

}

* Aggregation by Gender:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword"

}

}

}

}

}

}

* Include another aggregation bala:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

}

}

* Exclude in the aggregation:

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword",

"exclude": "M"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

}

}

* Minimum document count :

GET account/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword",

"exclude": "M",

"min\_doc\_count": 100

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

}

}

* Running the command from the Cygwin tools:

curl -XGET --header 'content-type:application/JSON' 'localhost:9200/account/\_search?pretty' -d'

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 10,

"to": 20

},

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword",

"exclude": "M",

"min\_doc\_count": 200

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

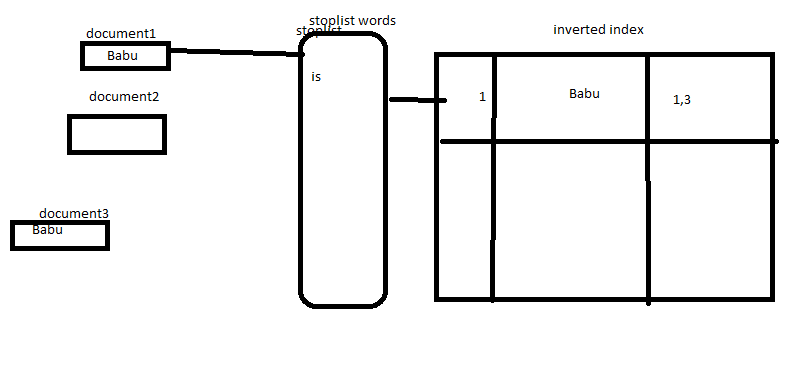
}

}

}

}'

Internal mapping between shards and documents:



* **Mapping**

Mapping is the process of defining how a document, and the fields it contains, are stored and indexed. For instance, use mappings to define:

which string fields should be treated as full text fields.

which fields contain numbers, dates, or geolocations.

whether the values of all fields in the document should be indexed into the catch-all \_all field.

the format of date values.

custom rules to control the mapping for dynamically added fields.`

* To know that datatype of the index run the below query:

GET account/\_mapping

Mapping for all indexes:

<http://localhost:9200/_mapping>

Creating index and checking mappings:

PUT test123

GET test123/\_search

GET test123/\_mapping

POST test123/1

{

"name":"Eshan"

}

GET test123/\_mapping

* Mapping types:

Core Data Types

String, Date, Numeric (long, integer, short, byte, double, and float), Boolean, Binary.

Complex Data Types

Array: Array support does not require a dedicated type.

Object: Object for single JSON objects.

Nested: Nested for arrays of JSON objects.

Geo Data Types

Geo-point: Geo\_point for latitude/longitude points.

Geo-shape: Geo\_shape for complex shapes such as polygons.

Specialized Data Types

IPv4: IP for IPv4 addresses.

Completion: completion to provide autocomplete suggestions.

Token count: token\_count to count the number of tokens in a string.

Attachment: Mapper-attachments plugin which supports indexing attachments in formats such as Microsoft Office, Open Document, ePub, and HTML, into an attachment datatype.

* Creating index with datatype:

PUT new\_index

{

"mappings": {

"properties": {

"user\_name": {

"type": "keyword"

}

}

}

}

* To insert records:

POST new\_index/\_doc/1

{

"user\_name":"test"

}

POST new\_index/\_doc/2

{

"user\_name":"test2"

}

* In Kibana with help on inspect we can find the internal HQL queries.
* Creating index with Date data type and sort the documents:

PUT myindex1

{

"mappings": {

"properties": {

"date": {

"type": "date"

}

}

}

}

PUT myindex1/\_doc/1

{

"date": "2015-12-12T12:23:20Z"

}

PUT myindex1/\_doc/2

{

"date": "2015-12-10"

}

PUT myindex1/\_doc/3

{

"date": "1420070400001"

}

GET myindex1/\_search

{

"sort": [

{

"date": {

"order": "asc"

}

}

]

}

* Creating index with geo\_location data type:

PUT restaurant

{

"mappings": {

"properties": {

"name": {

"type": "keyword"

},

"location": {

"type": "geo\_point"

}

}

}

}

GET restaurant/\_search

PUT restaurant/\_doc/1

{

"name": "Mical",

"location": [40.345, -74.3455]

}

PUT restaurant/\_doc/2

{

"name": "Mical",

"location": [34.345, -74.3455]

}

PUT restaurant/\_doc/3

{

"name": "Mical",

"location": [23.345, 33.3455]

}

PUT restaurant/\_doc/4

{

"name": "Mical",

"location": [27.345, 20.3455]

}

* Creating user info..

PUT userinfo

{

"mappings": {

"properties": {

"handle": {

"type": "text"

},

"age": {

"type": "integer"

},

"hobbies": {

"type": "text"

},

"computer": {

"properties": {

"cpu": {

"type": "text"

},

"speed": {

"type": "integer"

}

}

}

}

}

}

* Creating array of Recrods:

PUT userinfo

{

"mappings": {

"properties": {

"handle": {

"type": "text"

},

"age": {

"type": "integer"

},

"hobbies": {

"type": "text"

},

"computer": {

"properties": {

"cpu": {

"type": "text"

},

"speed": {

"type": "integer"

}

}

}

}

}

}

PUT userinfo/\_doc/1

{

"handle": "ramu",

"age": 43,

"hobbies": ["play","eat"],

"computer": {"cpu":"pentium", "mhz":200}

}

GET userinfo/\_search

Templates:

GET \_template/.logstash-management

**To create tempate:**

PUT \_template/mytemplate

{

"template": "customer\*",

"settings": {

"number\_of\_shards": 1,

"number\_of\_replicas": 1

},

"mappings": {

"properties": {

"cust\_name": {

"type": "keyword"

}

}

}

}

* To check the template details:

GET \_template/mytemplate

PUT customer\_test

* To check the mapping of the index...

GET customer\_bank/\_mapping

* Above statment proves that customer template is applied for customer\_test index.

GET \_template/customer\_bank

GET \_template/.logstash-management

* To check the status of the template:

$ curl -XHEAD -i 'localhost:9200/\_template/mytemplate'

Warning: Setting custom HTTP method to HEAD with -X/--request may not work the

Warning: way you want. Consider using -I/--head instead.

HTTP/1.1 200 OK

content-type: application/json; charset=UTF-8

content-length: 200

$ curl -XHEAD -i 'localhost:9200/\_template/mytemplate2'

Warning: Setting custom HTTP method to HEAD with -X/--request may not work the

Warning: way you want. Consider using -I/--head instead.

HTTP/1.1 404 Not Found

content-type: application/json; charset=UTF-8

content-length: 2

* From the devtools to check the template status:

HEAD \_template/mytemplate

* Create template for DNS:

PUT /\_template/dns\_template

{

"template": "dnslog-\*",

"settings": {

"number\_of\_shards": 3,

"number\_of\_replicas": 1

},

"mappings": {

"properties": {

"Timestamp": {

"type": "date"

},

"URL": {

"type": "keyword"

},

"IP Address": {

"type": "ip"

}

}

}

}

* Inserting document

POST dnslog-1/\_doc/1

{

"Timestamp" : "2015-04-09T14:12:12",

"URL" : "opendns.com/enterprise-security",

"IP Address" : "127.0.0.1"

}

* To verify whether the particual template is applied or not?

GET dnslog-1/\_mapping