Elasticsearch Summer Internship

This page discusses the approach and results achieved for the project assigned during Summer Internship.

# **Project**

* Transfer one and a half day Sandvine Data from HIVE table to Elsticsearch Index.
* Change the type of ‘*framedip*’ field from ‘*string’* to ‘*geo\_point*’ for *‘geohash’* in tile-map kibana.
* Queries, aggregations and analysis of data in Kibana.
* Further analysis of the Sandvine Dataset through visualizations and Dashboard.

## **Sandvine Data from HIVE to Elastisearch**

To create a new index in Elasticsearch, its template was created in Elasticssearch with the format of IP address as *‘IP’*.

Following is the template for index sandvine\_monitoring:

{

"sandvine\_monitoring": {

"mappings": {

"sandvinemonitoring": {

"dynamic": "true",

"\_all": {

"enabled": false

},

"\_timestamp": {

"enabled": true

},

"dynamic\_templates": [

{

"disable\_string\_index": {

"mapping": {

"index": "not\_analyzed",

"type": "string"

},

"match": "\*",

"match\_mapping\_type": "string"

}

}

],

"date\_detection": false,

"properties": {

"acceptlanguage": {

"type": "string"

},

"acctsessionid": {

"type": "string"

},

"clientdevice": {

"type": "string"

},

"contentencoding": {

"type": "string"

},

"contentlength": {

"type": "string"

},

"contenttype": {

"type": "string"

},

"day": {

"type": "string"

},

"filename": {

"type": "string"

},

"flowage": {

"type": "long"

},

"flowid": {

"type": "long"

},

"flowisend": {

"type": "boolean"

},

"framedip": {

"type": "ip"

},

"host": {

"type": "string"

},

"hour": {

"type": "string"

},

"httpcommand": {

"type": "string"

},

"internettxbytes": {

"type": "long"

},

"month": {

"type": "string"

},

"referer": {

"type": "string"

},

"requesttime": {

"type": "long"

},

"resource": {

"type": "string"

},

"responsetime": {

"type": "long"

},

"statuscode": {

"type": "integer"

},

"subscriberid": {

"type": "string"

},

"subscribertxbytes": {

"type": "long"

},

"transactionid": {

"type": "string"

},

"transactionstatus": {

"type": "integer"

},

"year": {

"type": "string"

}

}

}

}

}

}

To create an external table in HIVE to ES node, following code in HIVE was used:

create external table test1\_hive\_es (acctsessionid string, subscriberid string, framedip string,transactionid string,transactionstatus int,requesttime bigint,acceptlanguage string,responsetime bigint,httpcommand string,host string,resource string,referer string,statuscode int,contenttype string,contentlength string,flowage bigint, flowid bigint,contentencoding string,internettxbytes bigint,subscribeertxbytes bigint,flowisend boolean,clientdevice string,filename string,year string,month string,day string,hour string)

STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'

TBLPROPERTIES ('es.resource'= 'sandvine\_monitoring/sandvinemonitoring', 'es.nodes' = '10.177.228.211' , 'es.port' = '9200' ,'es.nodes.wan.only'= 'true' , 'es.batch.size.bytes' ='5mb', 'es.batch.size.enteries'= '5000', 'es.mapping.timestamp' = 'create\_date' ,'es.net.http.auth.user'='es\_username’,'es.net.http.auth.pass'='es\_password');

In the *‘incoming’* database in HIVE, sandvine data exists in the table *‘cvc\_ipnt\_sandvine\_http\_transactions’*. To copy this table to the new external table, following code was used to transfer one-hour data and so on by selecting the partitions:

insert overwrite table test1\_hive\_es

select cvc\_ipnt\_sandvine\_http\_transactions.\*

from cvc\_ipnt\_sandvine\_http\_transactions

WHERE cvc\_ipnt\_sandvine\_http\_transactions.year = '2016' and cvc\_ipnt\_sandvine\_http\_transactions.month = '05' and cvc\_ipnt\_sandvine\_http\_transactions.day = '18' and cvc\_ipnt\_sandvine\_http\_transactions.hour = '18';

## ‘IP’ to ‘geo\_point’ data type transformation for sandvine field ‘framedip’

### **Logstash**

This conversion was first carried out in Logstash. In the Logstash , input and output both was taken from ES and a filter ‘*geoip*’ was used. ‘*geoip*’ filter takes an ‘*ip*’ address from sandvine\_monitoring index and gives back its ‘*longitude’* and ‘*latitude*’ coordinates.

Below mentioned is the code from .conf file in logstash.

input {

elasticsearch {

hosts => ["10.177.228.211"]

index => "sandvine\_monitoring"

size => 100

scroll => "5m"

docinfo => true

scan => true

user => "es\_username"

password => "es\_password"

}

}

filter {

geoip {

source => "framedip"

target => "geoip"

}

}

output{

elasticsearch {

hosts => ["10.177.228.211"]

index => "sandvine\_may"

document\_type => "sandvinemonitoring"

user => "es\_username"

password => "es\_password"

workers => 4

}

}

These new fields were then put into the ES index with new mapping whose template is shown below in the next sectin. Note the type of framedip as ‘*geo\_point*’ defined.

### **HIVE**

As the indexing rate from Logstash to ES is around 4K documents/second as compared to 65K documents/seconds in HIVE to ES. The conversion of IP from string to geo\_point was done in HIVE itself.

Following are the steps, for this conversion:

Step 1: Create a template sandvine-monitoring in elasticsearch:

PUT \_template/sandvinetemplate2

{

"template": "sandvine-monitoring\*",

"mappings": {

"sandvinemonitoring":

{

"dynamic": "true",

"\_all": {

"enabled": false

},

"\_timestamp": {

"enabled": true

},

"dynamic\_templates": [

{

"disable\_string\_index": {

"mapping": {

"index": "not\_analyzed",

"type": "string"

},

"match": "\*",

"match\_mapping\_type": "string"

}

}

],

"date\_detection": false,

"properties": {

"acctsessionid": {

"type":"string"

},

"subscriberid": {

"type":"string"

},

"framedip": {

"type":"ip"

},

"transactionid": {

"type":"string"

},

"transactionstatus": {

"type": "integer"

},

"requesttime": {

"type": "long"

},

"acceptlanguage": {

"type": "string"

},

"responsetime": {

"type": "long"

},

"httpcommand": {

"type": "string"

},

"host": {

"type": "string"

},

"resource": {

"type": "string"

},

"referer": {

"type": "string"

},

"statuscode": {

"type": "integer"

},

"contenttype": {

"type": "string"

},

"contentlength": {

"type": "string"

},

"flowage": {

"type": "long"

},

"flowid": {

"type": "long"

},

"contentencoding": {

"type": "string"

},

"internettxbytes": {

"type": "long"

},

"subscribertxbytes": {

"type": "long"

},

"flowisend": {

"type": "boolean"

},

"clientdevice": {

"type": "string"

},

"filename": {

"type": "string"

},

"year": {

"type": "string"

},

"month": {

"type": "string"

},

"day": {

"type": "string"

},

"hour": {

"type": "string"

},

"country":

{

"type": "string"

},

"country\_code":

{

"type": "string"

},

"area\_code":

{

"type": "integer"

},

"city":

{

"type": "string"

},

"dma\_code":

{

"type": "integer"

},

"latitude":

{

"type": "double"

},

"longitude":

{

"type": "double"

},

"metro\_code":

{

"type": "integer"

},

"postal\_code":

{

"type": "long"

},

"region":

{

"type": "string"

},

"org":

{

"type": "string"

},

"id":

{

"type": "long"

},

"location":

{

"type": "geo\_point"

}

}

}

}

}

Step 2: Download the pom.xml and GenericUDPGeoIP.java from the following link:

<http://www.github.com/edwardcapriolo/hive-geoip>

Place the two documents in same folder and remove the package name from the .java file.

Step 3: Run the command “*mvn install*” in the same folder to create “*target*” folder.

Step 4: Download the geoip-api.jar and Compile the .java file

Download the geoip-api jar from the following link:

<http://www.java2s.com/Code/Jar/g/Downloadgeoipapi1210jar.htm>

Compile the .java file using following command:

javac –cp /usr/hdp/current/hive-client/lib/hive-exec-1.2.1.2.3.4.0-3485.jar:/usr/hdp/current/hive-client/lib/hive-exec.jar:/usr/hdp/current/Hadoop-client/hadoop-common.jar:/home/rkale/geoip-api-1.2.10.jar GenericUDFGeoIP.java

Step 5: Create a .jar file

jar –cf myudfs.jar GenericUDFGeoIP.class

Step 6:

Open the geoip-api.jar file and copy the contents of com.maxmind.geoip folder to the current directory working on to avoid the error : com.maxmind.geoip.LookupService not found.

Note: Cloudera was used to open the jar file to copy contents

Step 7: Download the GeoIp.dat and GeoLiteCity.dat from the following link

<http://dev.maxmind.com/geoip/legacy/geolite/>

Step 8: Login to hive and run following commands

add file GeoIP.dat;

add file GeoLiteCity.dat;

add jar geoip-api-1.2.10.jar;

add jar myudfs.jar;

create temporary function geoip as ‘GenericUDFGeoIP’;

Check whether geoip function working:

select geoip(ip, ‘COUNTRY\_NAME’, ‘./GeoIP.dat’);

select geoip(ip, ‘CITY’, ‘./GeoLiteCity.dat’);

Note: GeoIp.dat and GeoLiteCity.dat have following data:

GeoIP.dat: COUNTRY, COUNTRY\_CODE

GeoLiteCity.dat: CITY, AREA\_CODE, DMA\_CODE, LATITUDE, LONGITUDE, METRO\_CODE, POSTAL\_CODE, REGION, ORG, ID

Step 9: create external table temp:

create external table temp(acctsessionid string, subscriberid string, framedip string, transactionid string, transactionstatus int, requesttime bigint, acceptlanguage string, responsetime bigint, httpcommand string, host string, resource string, referer string, statuscode int, contenttype string, contentlength string, flowage bigint, flowid bigint, contentencoding string, internettxbytes bigint, subscribertxbytes bigint, flowisend boolean, clientdevice string, filename string, year string, month string, day string, hour string, country string, country\_code string, area\_code int, city string, dma\_code int, latitude double, longitude double, metro\_code int, postal\_code bigint, region string, org string, id bigint, location array<double>) stored by 'org.elasticsearch.hadoop.hive.EsStorageHandler' tblproperties('es.resource' = '/sandvine-monitoring/sandvinemonitoring', 'es.nodes' = '10.177.228.211:9200', 'es.net.http.auth.user' = 'es\_username', 'es.net.http.auth.pass' = 'es\_password', 'es.nodes.wan.only' = 'true');

Step 10: insert the values into the table:

insert overwrite table temp select e.acctsessionid, e.subscriberid, e.framedip, e.transactionid, e.transactionstatus, e.requesttime, e.acceptlanguage, e.responsetime, e.httpcommand, e.host, e.resource, e.referer, e.statuscode, e.contenttype, e.contentlength, e.flowage, e.flowid, e.contentencoding, e.internettxbytes, e.subscribertxbytes, e.flowisend, e.clientdevice, e.filename, e.year, e.month, e.day, e.hour, geoip(e.framedip, 'COUNTRY\_NAME', './GeoIP.dat') as country, geoip(e.framedip, 'COUNTRY\_CODE', './GeoIP.dat') as country\_code, geoip(e.framedip, 'AREA\_CODE', './GeoLiteCity.dat') as area\_code, geoip(e.framedip, 'CITY', './GeoLiteCity.dat') as city, geoip(e.framedip, 'DMA\_CODE', './GeoLiteCity.dat') as dma\_code, geoip(e.framedip, 'LATITUDE', './GeoLiteCity.dat') as latitude, geoip(e.framedip, 'LONGITUDE', './GeoLiteCity.dat') as longitude, geoip(e.framedip, 'METRO\_CODE', './GeoLiteCity.dat') as metro\_code, geoip(e.framedip, 'POSTAL\_CODE', './GeoLiteCity.dat') as postal\_code, geoip(e.framedip, 'REGION', './GeoLiteCity.dat') as region, geoip(e.framedip, 'ORG', './GeoLiteCity.dat') as org, geoip(e.framedip,'ID', './GeoLiteCity.dat') as id, array(cast(geoip(e.framedip, 'LONGITUDE', './GeoLiteCity.dat') as double), cast(geoip(e.framedip, 'LATITUDE', './GeoLiteCity.dat') as double)) as location from cvc\_ipnt\_sandvine\_http\_transactions e where e.year='2016' and e.month='05' and e.day='19' and e.hour='23';

## **Example Queries and Aggregations on Sandvine Data**

Full-text search

GET sandvine\_monitoring/\_search

{

"query": {

"match\_all": {

}

}

}

Filter the documents with client device as Iphone3 description

GET sandvine-may/\_search

{

"filter": {

"term": {

"clientdevice": "iphone3"

}

}

}

Minimum request time across all the documents

GET sandvine-may/\_search

{

"size": 0,

"aggs": {

"mintime": {

"min": {

"field": "requesttime"

}

}

}

}

Aggregation to find the total the documents with success status and then the unique IP address for only these documents

GET sandvine-may/\_search

{

"size": 0,

"aggs": {

"Success": {

"filter": {

"range": {

"statuscode": {

"gte": 199,

"lte": 400

}

}

},

"aggs": {

"UniqueIP": {

"cardinality": {

"field": "framedip"

}

}

}

}

}

}

Aggregation with a bool filter to find the total the documents with ‘GET’ as the request, success status and then the unique IP address for only these documents

GET sandvine-may/\_search

{

"size": 0,

"query": {

"bool": {

"must": [

{

"match": {

"httpcommand": "get"

}

}

],

"must\_not": [

{

"match": {

"statuscode": "200"

}

}

]

}

},

"aggs": {

"UniqueIP": {

"cardinality": {

"field": "framedip"

}

}

}

}

## **Link to Dashboard**

This is the link to the dashboard containing visualizations on Sandvine Dataset.

[http://10.177.228.211:5601/app/kibana#/dashboard/Sandvine-dash1?\_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-90d,mode:quick,to:now))&\_a=(filters:!(),options:(darkTheme:!f),panels:!((col:7,id:Unique-URL-count-per-hour-Sandvine-Dataset,panelIndex:25,row:5,size\_x:6,size\_y:4,type:visualization),(col:1,id:Success-and-Failure-of-httpcommands-Sandvine-dataset,panelIndex:27,row:5,size\_x:6,size\_y:4,type:visualization),(col:1,id:Unique-IP-count-per-hour-Sandvine-Dataset,panelIndex:28,row:1,size\_x:6,size\_y:4,type:visualization),(col:7,id:'HTTP-Commands,-Suceess,-Failures,-GET-Failures,-POST-Failures-per-hour-for-sandvine-dataset',panelIndex:29,row:1,size\_x:6,size\_y:4,type:visualization),(col:1,id:'Unique-IP-Addresses-for-failures-18th-May-19:00-sandvine-dataset',panelIndex:30,row:9,size\_x:12,size\_y:7,type:visualization)),query:(query\_string:(analyze\_wildcard:!t,query:'\*')),title:'Sandvine%20dash1',uiState:())](http://10.177.228.211:5601/app/kibana#/dashboard/Sandvine-dash1?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-90d,mode:quick,to:now))&_a=(filters:!(),options:(darkTheme:!f),panels:!((col:7,id:Unique-URL-count-per-hour-Sandvine-Dataset,panelIndex:25,row:5,size_x:6,size_y:4,)

1. Unique IP Count per hour – It gives the unique ip address count for each hour for sandvine customer data set
2. HTTP Commands, Successes, Failures, GET failures, POST failures per hour for sandvine dataset – It is a line graph with 5 lines each representing the total http command count, total successes count, total failures count, total failures for http command GET, total failures for http command POST.
3. Success and Failure of httpcommands – Sandvine dataset: The total number of successes and failures for each of the http commands.
4. Unique URL count per hour: Total number of web addresses visited for each hour.
5. Unique IP Addresses for failures – 18th May 19:00 sandvine dataset: The number of unique failed IP addresses for each latitude and longitude on tile map.