SAS® GLOBAL FORUM 2018

USERS PROGRAM

Integrating SAS® and Elasticsearch: Performing Text Indexing and Search

Presented by

Edmond Cheng

Booz | Allen | Hamilton

April 8 – 11 | Denver, CO

#SASGF

INTEGRATING SAS® AND ELASTICSEARCH: PERFORMING TEXT INDEXING AND SEARCH

A guide on using Elasticsearch to expand the power of performing textual analysis in SAS® products with fast and scalable documents indexing, complex search queries, and rapid visualization

SAS AND ELASTIC TEXT SOLUTIONS

SAS text specialized products enables text analysis, natural language processing, and statistical model application. Elasticsearch is a popular, fast, and scalable search solution, powering many modern webpages.

BASE SAS

FIND, VERIFY, SUBSTR, INDEX, TRANWRD, SCAN, COMPARE, SPESID, COMPGED, CONTAINS, LIKE, and etc.

PRODUCTS

- SAS Text Miner
- SAS Sentiment Analysis
- SAS Content Categorization
- SAS Contextual Analysis
- SAS Visual Text Analytics

elasticsearch

kibana

logstash

beats

READING APACHE WEB LOGS TO SAS

SAS codes performs extract, transform, load (ETL) operation on NASA web log as INFILE, SAS Perl RegEXP parses common log format to individual fields

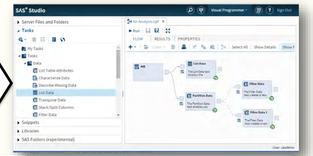
Web Logs

LOG

1.8M HTTP Requests

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/
HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET
/shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET
/shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET
/shuttle/countdown/liftoff.html HTTP/1.0" 304 0
```

SAS Studio



1 SAS INFILE to read the log file

```
infile "&nasafile" lrecl=32767 end=end;

if _N_ = 1 then
   rx = prxParse('/^(\S+) (\S+) (\S+) \[(.+)\] \"(\S+) (\S+)\s?(.*)\" (\S+) (\S+)/') ;
```

2 RegEXP captures fields

```
/* Define pattern matching */
start=1;
stop = length (_infile_);
call prxNext (rx,start,stop,_infile_,matchAt,matchLength);

/* Extract */
if prxMatch (rx, _infile_) then do;

   id       = strip(put(_n_,best12.));
   ipaddr   = prxPosN (rx, 1, _infile_);
   identity = prxPosN (rx, 2, _infile_);
   auth     = prxPosN (rx, 3, _infile_);
   timestamp= prxPosN (rx, 4, _infile_);
   method   = prxPosN (rx, 5, _infile_);
   path     = prxPosN (rx, 6, _infile_);
   http     = prxPosN (rx, 7, _infile_);
   status   = prxPosN (rx, 8, _infile_);
   size     = prxPosN (rx, 9, _infile_);
```

3 DATA STEP writes to SAS dataset

Obs	id	ipaddr	identity	auth	timestamp	method	path	http	status	size
1	1	199.72.81.55	-	-	1995:07:01 00:00:01	GET	/history/apollo/	HTTP/1.0	200	6245
2	2	unicomp6.unicomp.net	-	-	1995:07:01 00:00:06	GET	/shuttle/countdown/	HTTP/1.0	200	3985
3	3	199.120.110.21	-	-	1995:07:01 00:00:09	GET	/shuttle/missions/sts-73/mission-sts-73.html	HTTP/1.0	200	4085
4	4	burger.letters.com	-	-	1995:07:01 00:00:11	GET	/shuttle/countdown/liftoff.html	HTTP/1.0	304	0
5	5	199.120.110.21	-	-	1995:07:01 00:00:11	GET	/shuttle/missions/sts-73/sts-73-patch-small.gif	HTTP/1.0	200	4179

INDEXING DOCUMENTS TO ELASTICSEARCH

Elasticsearch uses standard RESTful APIs and JSON to perform indexing and search operations, mapping, and analytics functions. SAS PROC HTTP and JSON Library handles APIs and contents outputs

Indexing API

REST APIs — elastic — JSONs

PROC HTTP — SAS — JSON LIB

PUT Single Index

1 Defining Index PUT URL and content

```
do i = 1 to dim(charvar) ;
  charvar{i}=cats('"',vname(charvar{i}),'":"',charvar{i},'"');
end ;

* URL and Content ;
url = cats("&es./&index./&type./",id);
content = cats("{",catx(",", of allvar{*}),"}");
```

2 PROC HTTP sends the INDEX PUT request

```
proc sql noprint;
  select url, content into :url1-:url1000000, :content1-:content1000000
  from es_index_put;
quit;
proc http
   url="&&url&i"                          /* SAS generated ES PUT Index URL */
   method="PUT"
   in= %unquote(%str(%'&&content&i%'))  /* SAS generated record json */
   out=respfile
   headerout=hdrsfile;
run;
```

URL

```
https://search-sas-es-f2ecw7w781kvlliroci3fd5lf4.us-east-1.es.amazonaws.com/nasa/doc/1
```

CONTENT

```
{"ipaddr":"199.72.81.55","identity":"-","auth":"-","timestamp":"1995/07/01 00:00:01","method":"GET","path":"/history/apollo/","http":"HTTP/1.0","status":"200","size":6245}
```

POST Bulk Load

1 Creating the JSON metadata and content file

```
data es_index_bulk;
  length bulk_content $500;
  set es_index_put (firstobs=&firstnobs obs=&nobs);

  bulk_content = cats('{"index" : {"_index" : "','&index','", "_type" : "','&type', '", "_id" : "',id,'"}}');
  output;
  bulk_content = content;
  output;
run;
```

Line 1: `{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750007"}}`

Line 2: `{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}`

2 PROC HTTP sends the BULK POST request

```
data _null_;
  set es_index_bulk;
  file ingest;
  put bulk_content;
run;

proc http
   url="&es/_bulk"          /* ES POST Bulk API */
   in=ingest                /* SAS generated bulk records json */
   out=respfile
   headerout=hdrsfile;
   headers "Content-Type"="application/x-ndjson";
run;
```

```
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750006"}}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750007"}}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750008"}}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750009"}}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750010"}}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750011"}}
{"ipaddr":"www-b5.proxy.aol.com","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:50","method":"GET","path":"/shuttle/missions/sts-70/news/","http":"HTTP/1.0","status":"200","size":3306}
```

QUERYING, SEARCHING, AND STORING

SAS can query and search documents by sending HTTP requests via Elasticsearch Search API. Save the JSON-based response in SAS dataset, allowing SAS applications to enrich the results and build analytical models

GET Search API

URI SEARCH

Simple query encoding search parameters in the URL

```
proc http
   url="&es/&index/&type/13250?pretty"
   headerout=hdrsfile
   out=respfile;
run;
```

REQUEST BODY SEARCH

Complex searches supporting JSON Query DSL in message body

```
proc http
   url="&es/&index/&type/_search"
   in='{
      "query": {
         "bool":{
            "must":{"term":{"path":"mission"}},
            "filter":{"term":{"status":"200"}},
            "filter":{"range":{"size":{"gte":1000000}}}
         }
      },
      "sort" : { "timestamp" : {"order" : "desc"}},
      "size":100
   }'
   headerout=hdrsfile
   out=respfile;
```

Storing in SAS

1 PROC HTTP to request a search

```
%let size=100;
proc http
   url="&es/&index/&type/_search?size=&size"        /* ES Query lite API */
   out=respfile                                    /* Output the results to a
   headerout=hdrsfile;
run;
```

2 Assign JSON library to respfile

```
libname es JSON fileref=respfile;
proc datasets lib=es;
quit;
```

3 MERGE by ordinal values

```
data es_results;
  merge es.hits_hits (in=a drop=ordinal_hits rename=(ordinal_hit
  by ordinal_hits;
  if a and b;
run;
```

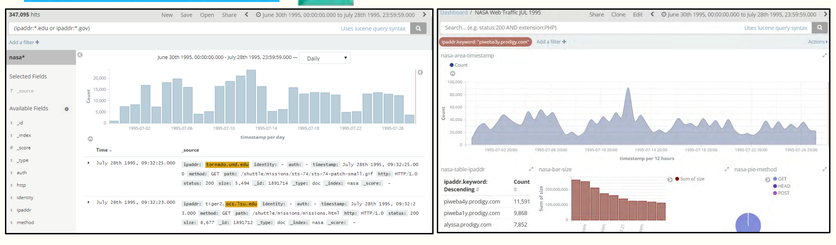
nce	ipaddr	identity	auth	timestamp	method	path	http	status	size
1	idhc027.tes.emc.com	-	-	1995:07:05 00:01:25	GET	/images/MOSAIC-logosmall.gif	HTTP/1.0	200	363
2	idhc027.tes.emc.com	-	-	1995:07:05 00:01:27	GET	/images/WORLD-logosmall.gif	HTTP/1.0	200	669
3	cairns.ucc.hull.ac.uk	-	-	1995:07:05 00:01:33	GET	/shuttle/missions/sts-67/images/KSC-95EC-0397.jpg	HTTP/1.0	200	125249
4	163.205.78.3	-	-	1995:07:05 00:01:34	GET	/images/ksclogo-medium.gif	HTTP/1.0	200	5866
5	163.205.78.3	-	-	1995:07:05 00:01:34	GET	/images/NASA-logosmall.gif	HTTP/1.0	200	786

ANALYSIS AND VISUALIZATION

Kibana is an interactive analysis tool providing powerful search features and interactive analytics dashboard for performing data exploration and text mining, complementing contextual analysis alongside SAS products.

Dashboard

Discover — kibana — Dashboard



cheng_edmond@bah.com www.linkedin.com/in/chengedmond Booz | Allen | Hamilton

SAS GLOBAL FORUM 2018

Paper 2900-2018

Integrating SAS® and Elasticsearch: Performing Text Indexing and Search

Edmond Cheng, Booz Allen Hamilton

ABSTRACT

Integrating Elasticsearch document indexing and text search components expands the power of performing textual analysis with SAS® solution products. Information technology, digitization, social connection, modern data storage, and big data accelerate unstructured text data production. Understanding the advantage in processing textual data and extracting underlying information provides valuable insights, setting an edge in competition, as seen in e-commerce, internet companies, communication media, marketing, health care, and across many industrial sectors.

This paper covers the benefits of architecting and implementing Elasticsearch applications alongside with SAS solutions. The first section presents an overview of Elasticsearch and common use cases. The paper demonstrates indexing SAS datasets into Elasticsearch NoSQL index, writing SAS codes to pass Elasticsearch REST APIs, and storing search query results. The final section demonstrates the use of Elasticsearch Kibana to further complement data visualization and business intelligent reporting capability with SAS analytics.

INTRODUCTION

SAS TEXT ANALYSIS AND SOLUTIONS

SAS as a leading analytics software company, offers a variety of specialized products for performing unstructured textual data processing and analytics. These solutions enable developers and end-users to discover meaningful context, apply statistical model application, and perform text searches to produce relevant information for their use cases.

The **BASE SAS** software includes text manipulation, parsing and, analyze functions for programmers to work with string datatype. BASE SAS functions such as variations of FIND, VERIFY, SUBSTR, INDEX, TRANWRD, SCAN, COMPARE, SPESID, COMPGED, and use of CONTAINS and LIKE with support for wildcards can handle text searches at the record or variable level.

The SAS text product line includes: **SAS Text Miner™**, a component of SAS Enterprise Miner™, provides advanced text extraction and natural language process features, combining quantitative variables with unstructured text to perform data mining techniques. **SAS Sentiment Analysis™** uses a combination of statistical and linguistic rules to identify sentiment in multiple level of the collection – record, category, overall document. **SAS Contextual Analysis™**, often works with SAS Visual Analytics™, uses contextual analysis categorizing textual data and building concept models, furthermore integrating results with other SAS products. **SAS Visual Text Analytics™**, which runs on SAS Viya™, uses comprehensive solution to identity topic, categorizing text data, building models, and run sentiment analysis. All of these running on difference SAS platforms offer a range of text processing capabilities for customer realize the analytical value of text-based data.

ELASTIC STACK

The most popular search solution is **Elasticsearch** developed by the company **Elastic**. Elasticsearch was developed to be the fastest and easiest to use search engine. If you have browsed Wikipedia, Yelp, or Netflix, then you have run a search powered by Elasticsearch. The search engine is based on Lucene, which is a free and open-source library for performing full text indexing and searching capability. Elasticsearch further provides distributed storage, a JSON-based common interface, RESTful APIs, and high accessibility for client applications. The official clients are available in Java, .NET, PHP, Python, Ruby, Groovy, and many other languages. The product opensource development continues to releasing new search attributes, adding new analytical functions, and improving enterprise features supporting customers with wide variety of use cases based on Elastic Stack.

Application Developers and Data Engineers are finding new use of Elasticsearch for storing documents and records with its build-in NoSQL like database. Other use cases leverage Elasticsearch for running advanced analytics on textual data, outperforming enterprise-grade database and Hadoop systems.

The Elastics Stack is a combination of three tools – Elasticsearch, Logstash, and Kibana.

Logstash serves as a data processing pipeline that ingests data from multiple data sources, performs transformations, send the data to Elasticsearch.

Kibana is a visualization tool to work with documents stored in Elasticsearch.

The rest of this paper demonstrates integrating SAS with Elastic Stack using document indexing and search results to perform text analysis.

NASA WEB TRAFFIC LOG

The data is a public release of NASA Kennedy Space Center web server logs collected in 1995 July containing approximately 1.8 million logs. The data is publicly available in http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html. Total size of the uncompressed file is 205.2MB.

The log file is in Apache common log format:

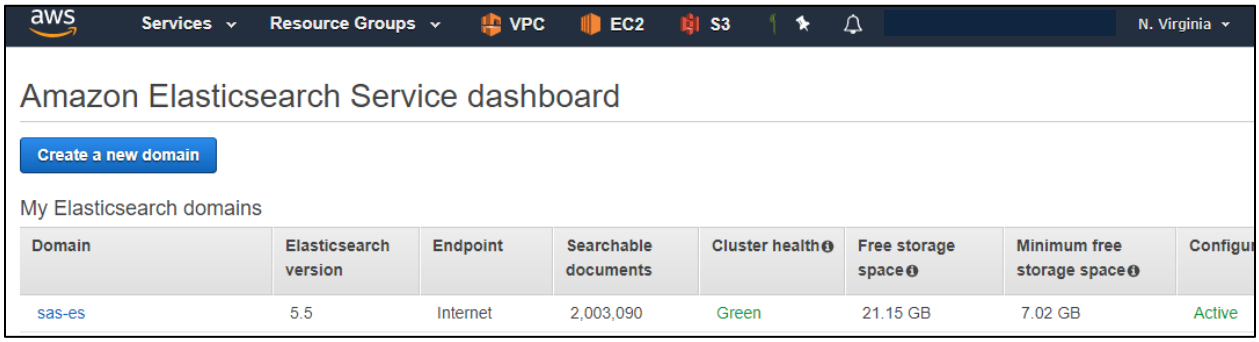
```
{host} {identify} {authentication} [{timestamp}] "{request}" {reply_code} {size}
```

Example:

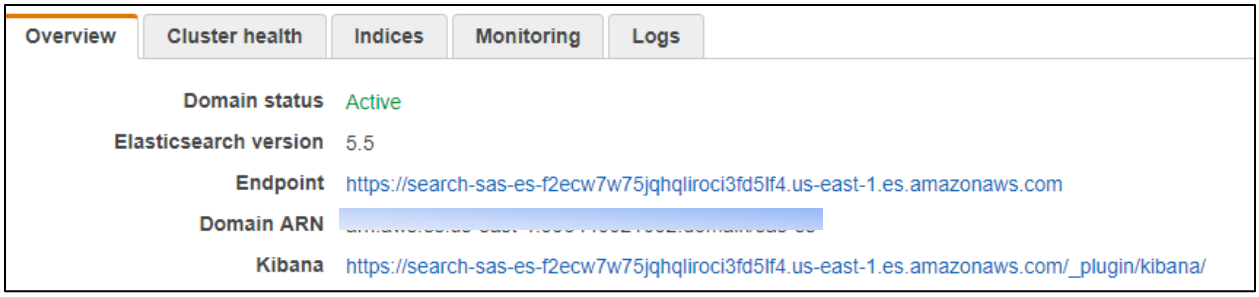
```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/
HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-
sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET
/shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-
patch-small.gif HTTP/1.0" 200 4179
```

SETTING UP THE ELASTICSEARCH CLUSTER

The Elasticsearch cluster is provisioned in **Amazon AWS** using the **AWS Elasticsearch** service. The ease of quickly deploying a healthy 3-node ES cluster with security enabled and high performance save the headaches in manually installing and configuring your own cluster for prototyping use cases. The AWS service offers ES 5.x/6.x versions, various EC2 instance and storage options, and setting access policy.



Once the AWS Elasticsearch cluster is active, the service creates the Elasticsearch and Kibana endpoints. SAS will be to use PROC HTTP to communicate Elasticsearch RESTful API for passing PUT, POST, and GET HTTP requests.



Note: the paper only shows the essential codes illustrating the syntax and technique to accomplish the task. See the Appendix for the complete sample SAS program

READING LOG FILE TO SAS

For data processing, SAS codes are programmed to perform light extract, transform, load (ETL) operation on reading the NASA web log file and writing to a SAS dataset. The NASA web log file can be handled by using **INFILE** function and **SAS Perl Regular Expression**. First, define the regular expression to match the common log format. Within the REQUEST message, the 'rx' expression is extended to further capture the METHOD, PATH, and HTTP fields. The **PRXPARSE** function compiles the RX regex expression.

```
71  infile "&nasafile" lrecl=32767 end=end;
72
73  if _N_ = 1 then
74    rx = prxParse('/^(\S+) (\S+) (\S+) \[(.+)\] \"(\S+) (\S+)\s?(.*)\" (\S+) (\S+)/')  ;
```

Use **PRXNEXT, PRXMATCH, and PRXPOSN** to parses each line of the record to 9 capture groups.

```
81    /* Define pattern matching */
82    start=1;
83    stop = length (_infile_);
84    call prxNext (rx,start,stop,_infile_,matchAt,matchLength);
85
86    /* Extract */
87    if prxMatch (rx, _infile_) then do;
88
89      id        = strip(put(_n_,best12.));
90      ipaddr    = prxPosN (rx, 1, _infile_);
91      identity  = prxPosN (rx, 2, _infile_);
92      auth      = prxPosN (rx, 3, _infile_);
93      timestamp = prxPosN (rx, 4, _infile_);
94      method    = prxPosN (rx, 5, _infile_);
95      path      = prxPosN (rx, 6, _infile_);
96      http      = prxPosN (rx, 7, _infile_);
97      status    = prxPosN (rx, 8, _infile_);
98      size      = prxPosN (rx, 9, _infile_);
```

This example stores all incoming fields as string datatype. A different DATA STEP later has a separate routine to convert all the fields to JSON notation, it is optional or rather simpler storing numeric datatype as string for ease of data manipulation. The 'timestamp' variable needs to be put in a datetime format matching to an Elasticsearch built-in date format. Elasticsearch is preconfigured with a set of recognizable date formats and able to parse date formatted as string into supported datatime values. With properly define date format, Elasticsearch can perform additional time-based analysis and aggregation.

Once the DATA STEP completes reading the log file, run a PROC PRINT of the SAS dataset to verify the capture groups are properly stored and formatted. Also run PROC JSON to print the outputs in JSON format verifying formatting issues before moving to the indexing section.

```sas
/* Verify outputs in 1) proper SAS data format and 2) JSON file */
proc print data=nasa_log (obs=10); run;

proc json out="&nasajson" nosastags pretty;
  export nasa_log (obs=10);
run;
```

Output from PROC PRINT:

Obs	id	ipaddr	identity	auth	timestamp	method	path	http	status	size
1	1	199.72.81.55	-	-	1995/07/01 00:00:01	GET	/history/apollo/	HTTP/1.0	200	6245
2	2	unicomp6.unicomp.net	-	-	1995/07/01 00:00:06	GET	/shuttle/countdown/	HTTP/1.0	200	3985
3	3	199.120.110.21	-	-	1995/07/01 00:00:09	GET	/shuttle/missions/sts-73/mission-sts-73.html	HTTP/1.0	200	4085
4	4	burger.letters.com	-	-	1995/07/01 00:00:11	GET	/shuttle/countdown/liftoff.html	HTTP/1.0	304	0
5	5	199.120.110.21	-	-	1995/07/01 00:00:11	GET	/shuttle/missions/sts-73/sts-73-patch-small.gif	HTTP/1.0	200	4179

Output from PROC JSON:


```
{
  "id": "1",
  "ipaddr": "199.72.81.55",
  "identity": "-",
  "auth": "-",
  "timestamp": "1995/07/01 00:00:01",
  "method": "GET",
  "path": "/history/apollo/",
  "http": "HTTP/1.0",
  "status": "200",
  "size": "6245"
},
{
  "id": "2",
  "ipaddr": "unicomp6.unicomp.net",
  "identity": "-",
  "auth": "-",
  "timestamp": "1995/07/01 00:00:06",
  "method": "GET",
  "path": "/shuttle/countdown/",
  "http": "HTTP/1.0",
  "status": "200",
  "size": "3985"
```


INDEXING DOCUMENTS USING PUT AND BULK LOAD

Elasticsearch uses standard RESTful APIs and JSON based data exchange to perform all indexing and search operations, plus other analytics functions, mapping, and cluster administration. Elasticsearch provides excellent documentation on its API library, including examples which can run directly in client application, command line, and Kibana. You can access the APIs using standard HTTP, or any client technology, such as Java, Python, .NET, C, C#, Perl, Ruby, and other languages. Recent release of SAS 9.4 brought number of improvements to **PROC HTTP** and **JSON Library**, increasing the flexibility in handling custom header and application contents. The learning curve to integrate Elasticsearch to SAS development is mostly translating SAS DATA STEP to prepare the API request URL and content body, using PROC HTTP to call the Elasticsearch API URL endpoint defined for the cluster.

To get familiar with sending messages to the Elasticsearch cluster for verification, define the API endpoint and index information as macro parameters, then use PROD HTTP to send the API call.

```sas
/***** PARAMETERS *****/
%let es          = https://search-sas-es-f2ecw7w75jqhqliroci3fd5lf4.us-east-1.es.amazonaws.com;
%let index       = nasa;
%let type        = doc;


/* Verifying connectivity to Elasticsearch cluster */
proc http
   url="&es"
   out=respfile
   headerout=hdrsfile;
run;
```

If the messages successfully reach the Elasticsearch server, the service will response with a response message showing the Elasticsearch cluster information.

The header message shows HTTP 200 OK status	The response message shows information about the cluster
HTTP/1.1 200 OK Access-Control-Allow-Origin: * Content-Type: application/json; charset=UTF-8 Content-Length: 333 Connection: keep-alive	{ "name" : "PTsEo9t", "cluster_name" : " "cluster_uuid" : "OHxKfk0GS_SP5BrVOcKP6w", "version" : { "number" : "5.5.2", "build_hash" : "Unknown", "build_date" : "2017-10-18T04:35:01.381Z", "build_snapshot" : false, "lucene_version" : "6.6.0" }, "tagline" : "You Know, for Search" }

INDEXING METHOD 1: PUT INDEX API

Elasticsearch provides two API methods for indexing documents.

The first option is the INDEX API that involves sending a **HTTP PUT** request to insert or update a single document. The URL specification includes the cluster endpoint, index, type, and id number. The body of the message is JSON based. One important and popular Elasticsearch feature is dynamic mapping, without having to define an initial mapping nor modifying schema for mapping changes when ingesting documents. Therefore, if an initial mapping is not found for the index, a new mapping will be dynamically generated.

Reading from the nasa_log.sas7bdat dataset, use a DO-LOOP converts the character and number variables to individual key-pair store conforming to JSON format. Notice the code has a slight variation for handling numeric datatype because quoting is not needed around the value. Writing the URL is simply defining macro parameters for the Elasticsearch endpoint &ES, index name &INDEX, and type &TYPE concatenated with the ID value, taken from _n_ reference.

```
36  do i = 1 to dim(charvar) ;
37    charvar{i}=cats('"',vname(charvar{i}),'":"',charvar{i},'"');
38  end ;
39  do i = 1 to dim(numvar) ;
40    if numvar{i} = '-' then numvar{i} = '0';
41    numvar{i}=cats('"',vname(numvar{i}),'":',numvar{i});
42  end ;
43
44  * URL and Content ;
45  url = cats("&es./&index./&type./",id);
46  content = cats("{",catx(",", of allvar{*}),"}");
```

This example produces **URL** and **CONTENT** as follows:

```
URL     = https://search-sas-es-f2ecw7w75jqhqliroci3fd5lf4.us-east-
          1.es.amazonaws.com/nasa/doc/1
CONTENT = {"ipaddr":"199.72.81.55","identity":"-","auth":"-","timestamp":
          "1995/07/01 00:00:01","method":"GET","path":"/history/apollo/",
          "http":"HTTP/1.0","status":"200","size":6245}
```

Next, store the URL and CONTENT as a series of macro variable sequence

```
51 proc sql noprint;
52   select url, content into :url1-:url1000000, :content1-:content1000000
53   from es_index_put;
54 quit;
```

Lastly, using PROC HTTP to pass the INDEX PUT requests with the macroized &&URL&i and &&CONTENT&I to the Elasticsearch cluster. *Macro parameterized DO-LOOP is not shown.*

```
67 proc http
68    url="&&url&i"                          /* SAS generated ES PUT Index URL */
69    method="PUT"
70    in= %unquote(%str(%'&&content&i%'))  /* SAS generated record json */
71    out=respfile
72    headerout=hdrsfile;
73 run;
```

INDEXING METHOD 2: BULK LOAD API

Since the HTTP request can insert only one document at a time, the PUT INDEX API is not suitable for indexing large volume of records. Indexing 1.8 million NASA log messages would take more than 20 hours. The second method is the Bulk Load API, it performs multiple index operations in a single **HTTP POST API** call. The Bulk Load API can insert thousands of records, within the HTTP transfer limit per the Elasticsearch server configuration. For example, AWS Elasticsearch service places a 10MB or 100MB HTTP transfer limit per single request, depending on the instance type. Regardless, **the Bulk Load API is what you want SAS to interface when indexing to Elasticsearch.**

The Bulk data format specification is a JSON-based file containing two lines for each record, a metadata and content entry, separated by a newline. The **first line** contains the metadata information including the index, type, and ID. The **second line** contains the message body in JSON object, which is also identical to the CONTENT format used in PUT INDEX API.

Create a new SAS dataset based on the previous PUT Index dataset. For each record, output a new record for the metadata key-value pairs and output another record for the content.

```
87 data es_index_bulk;
88   length bulk_content $500;
89   set es_index_put (firstobs=&firstnobs obs=&nobs);
90
91   bulk_content = cats('{"index" : {"_index" : "',"&index",'", "_type" : "',"&type", '", "_id"
92   output;
93   bulk_content = content;
94   output;
95
96   keep bulk_content;
97 run;
```

This example produces Line-1 and Line-2 for each record as follow:

```
Line 1: {"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750007"}
Line 2: {"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":
        "1995/07/26 04:58:45","method":"GET","path":"/shuttle/missions/sts-
        70/news/","http":"HTTP/1.0","status":"200","size":3306}
```

Prepare a JSON-based input file containing all the documents to be ingested by Bulk Load API

```
100 data _null_;
101    set es_index_bulk;
102    file ingest;
103    put bulk_content;
104 run;
```

Contents from file ingest:

```
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750007"}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:45","method":"GET","path"
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750008"}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:48","method":"GET","path"
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750009"}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:48","method":"GET","path"
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750010"}
{"ipaddr":"148.197.173.75","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:48","method":"GET","path"
{"index" : {"_index" : "nasa", "_type" : "doc", "_id" : "1750011"}
{"ipaddr":"www-b5.proxy.aol.com","identity":"-","auth":"-","timestamp":"1995/07/26 04:58:50","method":"GET",
```

Lastly, use PROC HTTP to pass the Bulk Load Post request, sending the input file to the Elasticsearch cluster

```
proc http
   url="&es/_bulk"          /* ES POST Bulk API */
   in=ingest                /* SAS generated bulk records json */
   out=respfile
   headerout=hdrsfile;
   headers "Content-Type"="application/x-ndjson";
run;
```

While not shown, the HEADEROUT=HDRSFILE and OUT=RESPFILE captures the Elasticsearch response. If the index is successful, the header file will show STATUS 200, and the response file will show all the indexed documents as JSON objects.

QUERYING, SEARCHING, AND STORING AS SAS DATASET

If you follow the Elasticsearch release cycle, the product opensource development moves at a rapid pace. Every Elasticsearch release offers new features and improvements. This paper demonstrates only few SEARCH API examples in context of integrating with SAS. In the ES documentation, select the version matching the working ES cluster and examine the Search APIs. The first part demonstrates how SAS sends GET HTTP request calling the Search API to query or search document. The second part shows how SAS stores the JSON-based response as SAS dataset. By storing the Elasticsearch results in SAS, SAS can consume the results and further build analysis models for insightful reporting.

SEARCH METHOD 1: URI SEARCH

Elasticsearch provides two Search APIs. The URI Search or 'query lite' API sends a **HTTP GET** request with the required parameters encoded in the URL. URI Search is a simple way to query for results without passing message body. However, the number of search options are limited in this mode.

Example 1: Query document ID 13250 and print pretty

SAS Code	Result

`proc http`
`  url="&es/&index/&type/13250?pretty"`
`  headerout=hdrsfile`
`  out=respfile;`
`run;`
	`{`
	`  "_index" : "nasa",`
	`  "_type" : "doc",`
	`  "_id" : "13250",`
	`  "_version" : 1,`
	`  "found" : true,`
	`  "_source" : {`
	`  "ipaddr" : "140.168.249.1",`
	`  "identity" : "-",`
	`  "auth" : "-",`
	`  "timestamp" : "1995/07/01 05:54:39",`
	`  "method" : "GET",`
	`  "path" : "/images/USA-logosmall.gif",`
	`  "http" : "HTTP/1.0",`
	`  "status" : "304",`
	`  "size" : 0`
	`  }`
	`}`

Example 2: Search thru the web logs for terms 'apollo' or 'discovery' and STATUS equals to '200' - note the URL encoding

SAS Code

```
140 proc http
141    url="&es/&index/&type/_search?q=%2Bpath%3A(
142    headerout=hdrsfile
143    out=respfile;
144 run;


       (apollo+discovery)+%2Bstatus%3A200&pretty"
```

Partial Result


```
{
  "took" : 46,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 223308,
    "max_score" : 6.8866477,
    "hits" : [
      {
        "_index" : "nasa",
        "_type" : "doc",
        "_id" : "161818",
        "_score" : 6.8866477,
        "_source" : {
          "ipaddr" : "slip160.hk.super.net",
          "identity" : "-",
          "auth" : "-",
          "timestamp" : "1995/07/03 12:19:45",
          "method" : "GET",
          "path" : "/htbin/wais.pl?discovery",
          "http" : "HTTP/1.0",
          "status" : "200",
```


SEARCH METHOD 2: REQUEST BODY SEARCH

The Request Body Search method sends a JSON body as part of the request search. The JSON body is a **Query DSL (Domain Specific Language)** based on a structured format to define the query. The query uses reserved words to perform operations, these terms include 'match', 'term', 'range', and 'bool'. The Request Body Search handles the most extensive and complex searches, including aggregation, multi fields, and fuzzy search. See the Elasticsearch documentation for full functionality.

Example 3: Search thru the web logs for the top 100 relevant results with the terms matching 'apollo', status code equals '200', byte size greater than '100kb' and sort in descend order by 'timestamp'.

SAS Code

```
151 proc http
152    url="&es/&index/&type/_search"
153    in='{
154         "query": {
155             "bool":{
156                 "must":{"term":{"path":"mission"}},
157                 "filter":{"term":{"status":"200"}},
158                 "filter":{"range":{"size":{"gte":100000}}}
159                 }
160             },
161         "sort" : {"timestamp" : {"order" : "desc"}},
162         "size":100
163         }
164         '
165    headerout=hdrsfile
166    out=respfile;
167 run;
168 %response_outputs(send_to_log=YES);
```

Partial Result


```
{"took":13,"timed_out":false,"_shards":{"total":5,"successful":5,"failed":0},"hit
:"nasa","_type":"doc","_id":"1890940","_score":null,"_source":{"ipaddr":"mdrint.s
995/07/28 13:19:30","method":"GET","path":"/shuttle/missions/sts-65/mission-sts-6
165},"sort":[806937570000]},{"_index":"nasa","_type":"doc","_id":"1889391","_scor
m","identity":"-","auth":"-","timestamp":"1995/07/28 12:52:38","method":"GET","pa
"http":"HTTP/1.0","status":"200","size":174454},"sort":[806935958000]},{"_index":
"_source":{"ipaddr":"not_registered.dayt.tasc.com","identity":"-","auth":"-","tim
:"/shuttle/missions/sts-65/mission-sts-65.html","http":"HTTP/1.0","status":"200",
sa","_type":"doc","_id":"1889225","_score":null,"_source":{"ipaddr":"www-c2.proxy
95/07/28 12:49:10","method":"GET","path":"/shuttle/missions/sts-65/mission-sts-65
65},"sort":[806935750000]},{"_index":"nasa","_type":"doc","_id":"1886615","_score
"identity":"-","auth":"-","timestamp":"1995/07/28 12:05:33","method":"GET","path"
tp":"HTTP/1.0","status":"200","size":174454},"sort":[806933133000]},{"_index":"na
ource":{"ipaddr":"sreynold.ssc.nasa.gov","identity":"-","auth":"-","timestamp":"1
/missions/sts-65/mission-sts-65.html","http":"HTTP/1.0","status":"200","size":131
":"doc","_id":"1886249","_score":null,"_source":{"ipaddr":"130.237.226.146","iden
08","method":"GET","path":"/shuttle/missions/sts-65/mission-sts-65.html","http":"
932748000]},{"_index":"nasa","_type":"doc","_id":"1886156","_score":null,"_source
"-","timestamp":"1995/07/28 11:57:47","method":"GET","path":"/shuttle/missions/st
":"200","size":131165},"sort":[806932667000]},{"_index":"nasa","_type":"doc","_id
178.17.149","identity":"-","auth":"-","timestamp":"1995/07/28 11:54:48","method":
65.html","http":"HTTP/1.0","status":"200","size":131165},"sort":[806932488000]},{
re":null,"_source":{"ipaddr":"proxy.bellatlantic.com","identity":"-","auth":"-","
```


STORING ELASTICSEARCH RESULTS IN SAS

The Elasticsearch speed and extensive text search algorithm can enrich text analysis capabilities in SAS. Elasticsearch executes complex searches across multiple indices and takes advantage of distributed

shards and replicas, returning results faster than SAS functions can produce. Elasticsearch allows fuzzy searches and relevance scoring return closest matching results taking consideration of language processing engine. Elasticsearch could replace time intensity regex running in SAS DATA STEP code. **SAS Stored Procedure** and **ODS** interfacing with Elasticsearch PUT and GET API can be added to **Visual Analytics** extending customized keyword search capability when viewing and search reports.

SAS 9.4M4 released the **JSON Library** allowing data access to JSON document as a SAS library reference This method allows storing JSON output returned from Elasticsearch as SAS dataset. Assigning the JSON Library reads the JSON file as a SAS dataset. An optional JSON map can be provided to define the JSON output schema. If none is specified, the SAS JSON library will automatically generate a JSON map based on the data.

Use PROC HTTP to call the URI Search API to query for first 100 results

```
174 %let size=100;
175 proc http
176    url="&es/&index/&type/_search?size=&size"        /* ES Query lite API */
177    out=respfile                                     /* Output the results to a
178    headerout=hdrsfile;
179 run;
```

Assign a JSON library referencing the SEARCH response file, the library assignment reads the JSON file and auto-generates a hierarchy of the table structure. To examine the library contents, run PROC DATASET and PROC CONTENTS to explore the table and column references.

```
185 libname es JSON fileref=respfile;
186 proc datasets lib=es;
187 quit;
```

In this example, the HITS_HITS table contains the original document ID. The HITS__SOURCE table contains the message content. Therefore to produce a SAS DATASET with doc id #, the two datasets needs to be merged.

```
193 data es_results;
194   merge es.hits_hits (in=a drop=ordinal_hits rename=(ordinal_hits1=ordinal_hits
195   by ordinal_hits;
196   if a and b;
197 run;
```

Lastly, verify the new SAS dataset is properly storing the outputs

```
199 /* Verify outputs showing the first 10 matched records sorted by relevance scor
200 proc print data=es_results (obs=10); run;
```

Outputs from PROC PRINT:

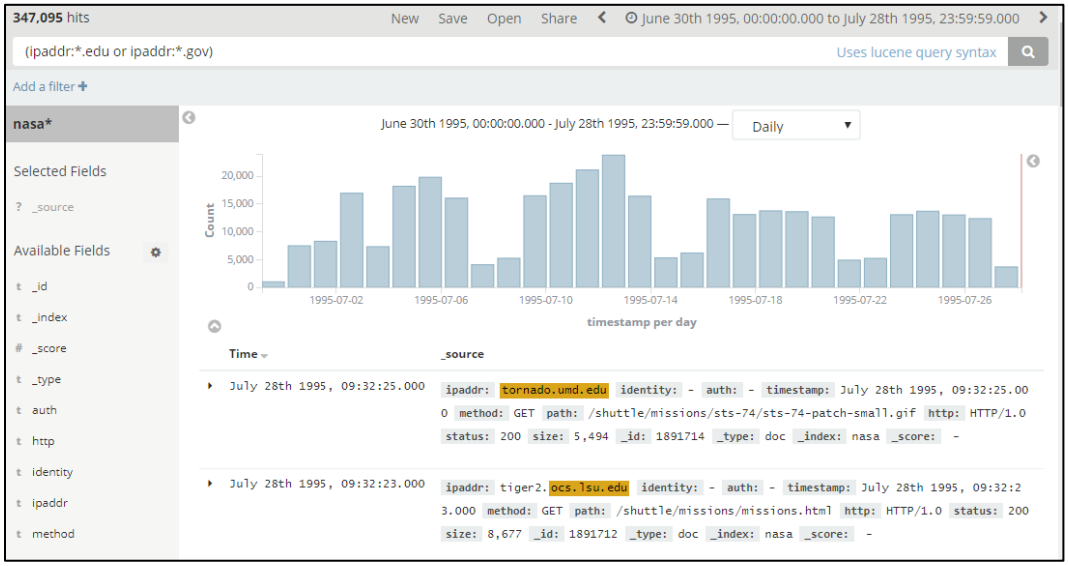
Obs	ordinal_hits	_index	_type	_id	_score	ordinal__source	ipaddr	identity	auth	timestamp	method	path	http	status	size
1	1	nasa	doc	301473	1	1	ldhc027.lss.emc.com	-	-	1995/07/05 08:01:25	GET	/images/MOSAIC-logosmall.gif	HTTP/1.0	200	363
2	2	nasa	doc	301476	1	2	ldhc027.lss.emc.com	-	-	1995/07/05 08:01:27	GET	/images/WORLD-logosmall.gif	HTTP/1.0	200	669
3	3	nasa	doc	301478	1	3	cairns.ucc.hull.ac.uk	-	-	1995/07/05 08:01:33	GET	/shuttle/missions/sts-67/images/KSC-95EC-0397.jpg	HTTP/1.0	200	125249
4	4	nasa	doc	301482	1	4	163.205.78.3	-	-	1995/07/05 08:01:34	GET	/images/ksclogo-medium.gif	HTTP/1.0	200	5866
5	5	nasa	doc	301484	1	5	163.205.78.3	-	-	1995/07/05 08:01:34	GET	/images/NASA-logosmall.gif	HTTP/1.0	200	786

TEXT ANALYSIS IN VISUALIZATION

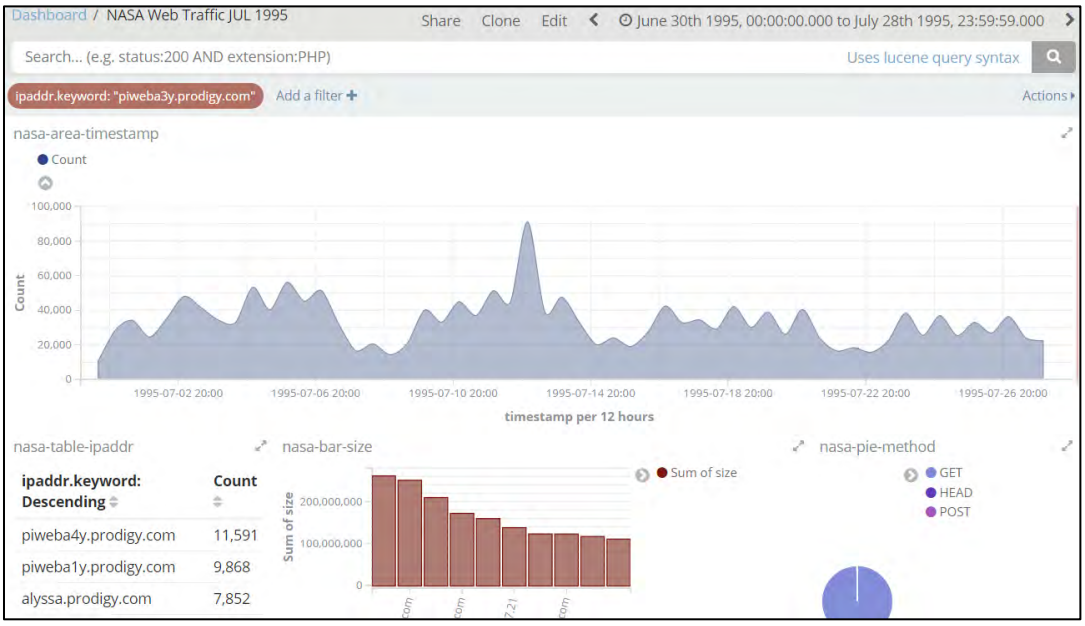
Elastic Stack provides **Kibana** as an interactive development tool and visualization web console operating on top of Elasticsearch. Developers can use its **Dev Tools** to perform operations on the cluster and indexes,

instead of passing JSON messages thru `curl` commands. Users can access **Dashboard** created with pre-defined filters and shared visualizations. The Dashboard supports extensive Elasticsearch search operations and Lucene search syntax, plus interactive filters on all visualizations in a dashboard. Visualizations from multiple indices can be displayed in given dashboard allowing searches across different collections. Kibana is a good complementary tool for performing text mining and terms exploration when building contextual models using SAS products.

The Kibana **Discover** tool allows user entering a search query matching over 1.8 million log entries with IP address ending in '.edu' or '.gov'. The application took less than a second to paginate the first batch of results. In some way, searching in Discover can complement SAS users who work or without SAS text mining solutions to discover keywords when building synonyms or topics for contextual analysis.



Kibana **Dashboard** impressively displays multiple interactive charting on the NASA web logs loaded from SAS. The time series chart plots the daily aggregated server web requests handled by the server. The table shows the 'hostname' of the top ten requestors. The bar charts show the 'total packet size' returned to the top ten 'requestors', and the pie chart indicates the GET method leading all HTTP requests to the server. While functionality and customization are limited comparing to SAS Visual Analytics, Kibana offers powerful search analytics and interactive dashboard for data exploration and reporting.



CONCLUSION

The significant growth and difficulty of processing unstructured textual data analysis demands fast, flexible, and efficient approach in discovering relevant information. SAS products offers number of solutions with unique approaches in supporting text data mining, topic identification, concept building, and combining quantitative variables with textual context for analysis. Elasticsearch is a powerful tool for indexing and searching over large volume of documents returning results in milliseconds, plus a robust RESTful API library and JSON data common interface. Integrating SAS solutions with other search-based applications like Elastic Stack further complements the exploration, enrichment, and reporting key information in textual data.

REFERENCES

DeVenezia, Richard A. “Parsing Apache log files”. “Devenezia”. March 1, 2018. https://www.devenezia.com/downloads/sas/samples/read-web-log.sas.

Elastic. “Elasticsearch Reference 5.5”. “Docs”. March 1, 2018. https://www.elastic.co/guide/en/elasticsearch/reference/5.5/index.html.

SAS Institute, Inc. “HTTP Procedure”. “Base SAS 9.4 Procedure Guide, Seventh Edition”. March 1, 2018. http://documentation.sas.com/?docsetId=proc&docsetTarget=n197g47i7j66x9n15xi0gaha8ov6.htm&docsetVersion=9.4&locale=en.

The Internet Traffic Archive. “NASA-HTTP”. “Contrib.” March 1, 2018. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

Windham, Mathew. 2014. Introduction to Regular Expressions. 1st ed. SAS Institute.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Edmond Cheng
Booz Allen Hamilton
cheng_edmond@bah.com

APPENDIX

```
/*****************************************
* PROGRAM:          SAS_Elasticsearch.sas
* DESCRIPTION:      Demonstration of expanding Base SAS textual analysis with Elasticsearch func:
*                         - Indexing high volume of documents from SAS to ES
*                         - Querying ES indexes and storing results in SAS dataset
*                         - Sending SAS dataset to Kibana for data visualization and text analysis
* INPUTS            NASA http log in Common Log Format from The Internet Traffic Archive,
*                   http://ita.ee.lbl.gov
* OUTPUTS           SAS dataset
*                         Elasticsearch index
* PROGRAMMER: Edmond Cheng
* DATE:                   2018-03-01
* NOTES:
*****************************************/

/***** OPTIONS *****/
options nomprint compress=yes reuse=yes;

proc printto log='/folders/myfolders/saslogs/saslog.log'
             print='/folders/myfolders/saslists/saslist.lst' new;
run;


/***** PARAMETERS *****/
%let nasafile= '/folders/myfolders/rawdata/NASA_access_log_Jul95';
%let nasajson= '/folders/myfolders/outputs/nasa_log.json';

%let es                   = https://search-sas-es-f2ecw7w75jqhqliroci3fd5lf4.us-east-
1.es.amazonaws.com;
%let index          = nasa;
%let type           = doc;


/***** ENVIRONMENT *****/
libname saslib '/folders/myfolders/sasdata';                /* SAS library */
filename hdrsfile '/folders/myfolders/outputs/hdrs.txt';    /* fileref for response headers */
filename respfile '/folders/myfolders/outputs/resp.json';   /* fileref for return output */
filename ingest '/folders/myfolders/outputs/ingest.json';   /* fileref for ingest file */

/***** FORMATS *****/
proc format;
picture dt
other='%0Y/%0m/%0d %0H:%0M:%0S' (datatype=datetime);
run;


/***** MACROS *****/
%macro response_outputs(send_to_log);
%put 'PRINTING Response Header';
data _null_;
   infile hdrsfile;
   input;
   %if &send_to_log=YES %then put _infile_;;
run;
%put 'PRINTING HTTP Output';
data _null_;
   infile respfile;
   input;
   %if &send_to_log=YES %then put _infile_;;
run;
%mend response_outputs;
```

```
/***** READING COMMON LOG FILE *****/
/* Reference code from Richard A. DeVenezia, https://www.devenezia.com/downloads/sas/samples/read-
web-log.sas */
/* Read in the raw nasa http log file and parse out the elements */
data nasa_log;

  retain rx;

  infile "&nasafile" lrecl=32767 end=end;

  if _N_ = 1 then
    rx = prxParse('/^(\S+) (\S+) (\S+) \[(.+)\] \"(\S+) (\S+)\s?(.*)\" (\S+) (\S+)/')  ;

  * process each line of log file;
  do i = 1 until (end);

    input;

    /* Define pattern matching */
    start=1;
    stop = length (_infile_);
    call prxNext (rx,start,stop,_infile_,matchAt,matchLength);

    /* Extract */
    if prxMatch (rx, _infile_) then do;

      id            = strip(put(_n_,best12.));
      ipaddr    = prxPosN (rx, 1, _infile_);
      identity  = prxPosN (rx, 2, _infile_);
      auth      = prxPosN (rx, 3, _infile_);
      timestamp = prxPosN (rx, 4, _infile_);
      method    = prxPosN (rx, 5, _infile_);
      path      = prxPosN (rx, 6, _infile_);
      http      = prxPosN (rx, 7, _infile_);
      status    = prxPosN (rx, 8, _infile_);
      size      = prxPosN (rx, 9, _infile_);

      /* Transform */
      timestamp = strip(put(input(compress(timestamp,'/'),datetime.),dt.));

      /* Load */
      output;
    end;

    else
      put _n_ 'data line in Apache log file is not in common format';

  end;

  drop i rx start stop matchAt matchLength;

run;


/* Verify outputs in 1) proper SAS data format and 2) JSON file */
proc print data=nasa_log (obs=10); run;

proc json out="&nasajson" nosastags pretty;
  export nasa_log (obs=10);
run;

/***** VERIFICATION *****/
/* Verifying connectivity to Elasticsearch cluster */
proc http
   url="&es"
```

```
   out=respfile
   headerout=hdrsfile;
run;
%response_outputs(send_to_log=YES);

/* Print a list of existing indexes */
proc http
   url="&es/_cat/indices?v"
   out=respfile
   headerout=hdrsfile;
run;
%response_outputs(send_to_log=YES);


/***** INDEXING *****/
/***** INDEXING: PUT INDEX API */
/* Using single PUT _index API to index one document *****/
/* Step 1:   Prepare URL and Content to be passed into ES in the SAS dataset
                  URL is the http with the ES address, index name, type, and record ID
                  Content is the message body in JSON object */
%let charvar = ipaddr identity auth timestamp method path http status ;
%let numvar         = size;

data es_index_put;
  length url content $500;
  set nasa_log  (obs=max);

  array charvar{*} &charvar  ;
  array numvar{*} &numvar ;
  array allvar{*} &charvar &numvar;

  do i = 1 to dim(charvar) ;
    charvar{i}=cats('"',vname(charvar{i}),'":"',charvar{i},'"');
  end ;
  do i = 1 to dim(numvar) ;
    if numvar{i} = '-' then numvar{i} = '0';
    numvar{i}=cats('"',vname(numvar{i}),'":',numvar{i});
  end ;

  * URL and Content ;
  url = cats("&es./&index./&type./",id);
  content = cats("{",catx(",", of allvar{*}),"}");

run;

/* Step 2: Generate the sequence of url and content string as macro variables */
proc sql noprint;
  select url, content into :url1-:url1000000, :content1-:content1000000
  from es_index_put;
quit;

%put The first and last definitions are:;
%put #1: url=&url1;
%put #1: content=&content1;
%put #&sqlobs: url=&url1;
%put #&sqlobs: content=&content1;

/* Step 3: Using Proc HTTP to pass the ES _index PUT API with the URL/Content */
%macro put_es;

%do i = 1 %to &sqlobs;

proc http
   url="&&url&i"                                    /* SAS generated ES PUT Index URL */
   method="PUT"
   in= %unquote(%str(%'&&content&i%'))  /* SAS generated record json */
   out=respfile
```

```
   headerout=hdrsfile;
run;
%response_outputs(send_to_log=NO);
%end;
%mend put_es;
%put_es;



/***** INDEXING: BULK LOAD API */
/* Using bulk index API to put batches of docs *****/
/* Step 1:  The bulk load API requires a metadata and body entry for each record, separated be a
newline
                    Bulk Content line 1 contains the index name, type, and id in JSON object
                    Bulk Content line 2 contains the message body in JSON object, also identical
with index API */
%macro es_index_bulk(firstnobs=,nobs=);
data es_index_bulk;
  length bulk_content $500;
  set es_index_put (firstobs=&firstnobs obs=&nobs);

  bulk_content = cats('{"index" : {"_index" : "',"&index",'", "_type" : "',"&type", '", "_id" :
"',id,'"}');
  output;
  bulk_content = content;
  output;

  keep bulk_content;
run;

/* Step 2: Prepare an input file to be ingested by bulk api */
data _null_;
   set es_index_bulk;
   file ingest;
   put bulk_content;
run;

/* Step 3: Using SAS Proc HTTP to define the ingest file in ES _bulk POST API */
proc http
   url="&es/_bulk"              /* ES POST Bulk API */
   in=ingest                    /* SAS generated bulk records json */
   out=respfile
   headerout=hdrsfile;
   headers "Content-Type"="application/x-ndjson";
run;
%response_outputs(send_to_log=NO);
%mend es_index_bulk;

/* Run in batches of 250,000 records conforming to server bandwidth */
%es_index_bulk(firstnobs=1,nobs=250000);
%es_index_bulk(firstnobs=250001,nobs=500000);
%es_index_bulk(firstnobs=500001,nobs=750000);
%es_index_bulk(firstnobs=750001,nobs=1000000);
%es_index_bulk(firstnobs=1000001,nobs=1250000);
%es_index_bulk(firstnobs=1250001,nobs=1500000);
%es_index_bulk(firstnobs=1500001,nobs=1750000);
%es_index_bulk(firstnobs=1750001,nobs=2000000);


/***** QUERY, SEARCH, AND TEXT ANALYSIS *****/
/* Step 1 Using GET query lite API to query docs */
/* Query document ID 13250 */
proc http
   url="&es/&index/&type/13250?pretty"
   headerout=hdrsfile
   out=respfile;
run;
```

```
%response_outputs(send_to_log=YES);

/* Search for http log with term 'apollo or discovery' and status '200' OK - note the URL encoding
*/
proc http
   url="&es/&index/&type/_search?q=%2Bpath%3A(apollo+discovery)+%2Bstatus%3A200&pretty"
   headerout=hdrsfile
   out=respfile;
run;
%response_outputs(send_to_log=YES);


/* Step 2 Using GET request body search allowing more complex and multi fields search
   Search for top 100 best matching http logs with term 'apollo, status 200 OK code,
   size greater than 100kb and sort in descend order by timestamp  */
proc http
   url="&es/&index/&type/_search"
   in='{
           "query": {
              "bool":{
                     "must":{"term":{"path":"mission"}},
                     "filter":{"term":{"status":"200"}},
                     "filter":{"range":{"size":{"gte":100000}}}
                     }
              },
              "sort" : {"timestamp" : {"order" : "desc"}},
           "size":100
           }
           '
   headerout=hdrsfile
   out=respfile;
run;
%response_outputs(send_to_log=YES);


/* Step 3 Storing the results in SAS dataset */
/* Set a pagination size of 1,000 documents */
%let size=100;
proc http
   url="&es/&index/&type/_search?size=&size"         /* ES Query lite API */
   out=respfile                                                    /* Output the
results to a response file */
   headerout=hdrsfile;
run;
%response_outputs(send_to_log=NO);

/* Set up a JSON library referenceing the response file output from ES */
/* The JSON file is represented by auto generated hierarchy of tables, hits
   Use the proc dataset and content to explore the appropiate table and columns needed */
libname es JSON fileref=respfile;
proc datasets lib=es;
quit;

/* In this example,
   The hits_hits table contains the document id #
   The hits__source table contains the message content
   To produce a complete record with the doc id #, the two datasets are merged by the SAS generated
ordinal_hits field */
data es_results;
  merge es.hits_hits (in=a drop=ordinal_hits rename=(ordinal_hits1=ordinal_hits)) es.hits__source
(in=b);
  by ordinal_hits;
  if a and b;
run;

/* Verify outputs showing the first 10 matched records sorted by relevance score */
proc print data=es_results (obs=10); run;
```