BLOG POSTS

	Content
Post Title	Exploring an ELK Stack: Part 4: Pivot Table
Post Date	
Attributed To	Peter Welcher
Written By	Peter Welcher
Reviewed By (Name & Date)	Dave Donati (11/19)
Reviewed By (Name & Date)	

Meta Title (55 characters including spaces)	Exploring an ELK Stack: Part 4: Pivot Table
Meta Description (156 characters including spaces)	
Target Keywords	
Categories	Technology
Tags	N/A
Call to Action	N/A
Image	Put a brief description of what the image should be or note that a file is attached. DO NOT paste the image into this Word doc; send it as a separate file.

Note: Naming convention for files as they go back and forth

- Original writer names file with "_V1" at the end (e.g., blogtitle_V1)
- First reviewer, makes edits and renames with initials at end (e.g., blogtitle_V1_af)
- If another reviewer, again add initials to end to keep the string of reviewers (e.g., blogtitle_V1_af_pw)
- When original writer gets it post back with edits, she makes revisions and saves the file as V2 (e.g. blogtitle_V2) – then reviewers continue as above with initials
- When post is complete, it is saved with "Final" and the post date at the end (e.g. blogtitle_FINAL_022012)

Copy for post:

This is Part 4 of a blog series tutorial about the ELK stack. The prior blogs in the series are:

- Exploring an ELK Stack Part 1: Importing Data and Patterns
- Exploring an ELK Stack: Part 2: Kibana Visualizations
- Exploring an ELK Stack: Part 3: Dashboards

Commented [PW1]: Insert link

Commented [PW2]: Insert link

Commented [PW3]: Ditto: insert link

Part 1 introduced the ELK stack and covered getting some data into Elasticsearch via a new experimental Kibana feature. It also covered Logstash grok patterns lightly, and some web tools for working with them (regular expressions made easier!).

Part 2 covered some basic things you can do with the Kibana visualization tool (the “K” in “ELK).

Part 3 walked through creating a Kibana dashboard.

This blog looks at creating a query to the Elasticsearch search engine, with a particular objective in mind: producing the data to support a pivot table. I set this goal in part with the idea of reproducing something I was getting from my PERL syslog scripts, except doing it with an ELK stack.

Goal: Pivot Tables

My adapted PERL script for syslog processing, which badly needs rewriting in Python, outputs tab-separated fields: Error Code, IP Address / Hostname, Interface, and Count. The count is how many times the specific error code, IP address, and interface occurred in the syslog data. There is (or easily can be) a header line with tab-separated column names.

Tab-separating the output means that I can easily import the data into Excel and in about one minute, create a pivot table. The pivot table provides more or less the same data as the graph at the end of blog #2 in this series.

Commented [PW4]: Insert link to prior blog

You can then easily drill down to explore the worst problems — the ELK counterpart is hovering the cursor over the graph we produced in blog #2.

Web research appears to show that:

- ELK has no current Kibana pivot table functionality
- Third parties can provide a Kibana plug-in that does pivot tables

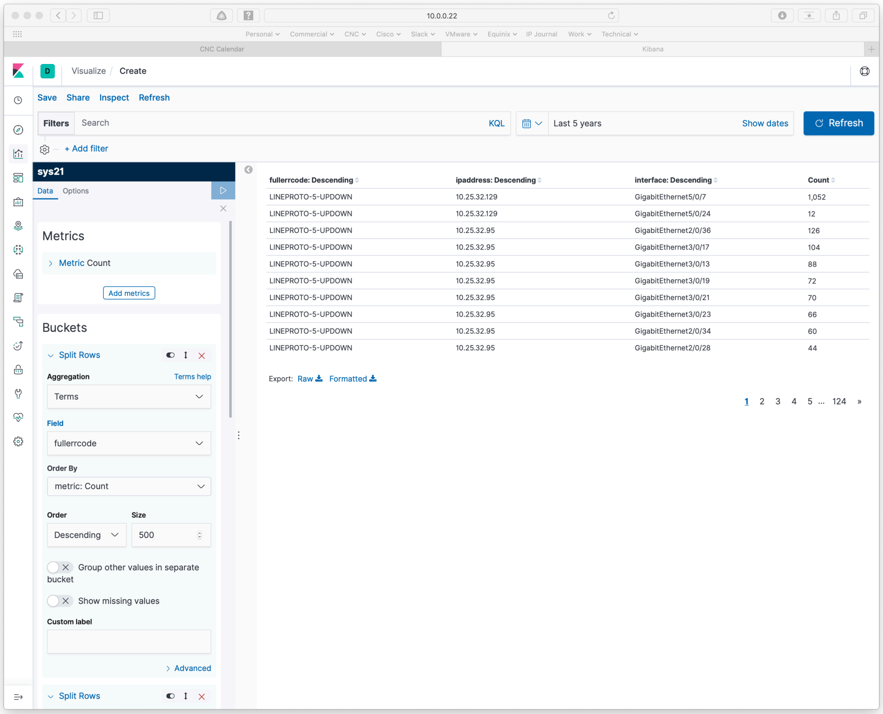
You can, however, approximate what I was looking for within Kibana, albeit not as a pivot table. That is, get the raw data out and use Excel for the pivot table part of things.

Kibana Data Table

I created a Data Table visualization from an Index created using the latest version of my pattern, the one with “fullerrcode”.

I filled in Split-rows consecutively for sub-buckets on the fields “fullerrcode”, “ipaddress”, and “interface”. I set the bucket sizes to 500 to ensure no lost data (for this limited log file, anyway).

The result can be seen in the screen capture below.



Aggregation: Terms

Field: fullerrcode

Order By: metric: Count

Order: Descending

Size: 500

Group other values in separate bucket

Show missing values

Custom label:

fullerrcode: Descending	ipaddress: Descending	interface: Descending	Count
LINEPROTO-5-UPDOWN	10.25.32.129	GigabitEthernet5/0/7	1,052
LINEPROTO-5-UPDOWN	10.25.32.129	GigabitEthernet5/0/24	12
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet2/0/36	126
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet3/0/17	104
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet3/0/13	88
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet3/0/19	72
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet3/0/21	70
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet3/0/23	66
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet2/0/34	60
LINEPROTO-5-UPDOWN	10.25.32.95	GigabitEthernet2/0/28	44

Export: Raw Formatted

1 2 3 4 5 ... 124 »

Then click on the Export link (bottom middle of the above screen capture).

Apparently, Export Raw or Formatted both produce the same CSV. Here is an excerpt from the exported CSV file:

```
"fullerrcode: Descending","ipaddress: Descending","interface:
Descending",Count
"LINEPROTO-5-UPDOWN","10.25.32.129","GigabitEthernet5/0/7",1052
"LINEPROTO-5-UPDOWN","10.25.32.129","GigabitEthernet5/0/24",12
"LINEPROTO-5-UPDOWN","10.25.32.95","GigabitEthernet2/0/36",126
```

```
"LINEPROTO-5-UPDOWN","10.25.32.95","GigabitEthernet3/0/17",104
"LINEPROTO-5-UPDOWN","10.25.32.95","GigabitEthernet3/0/13",88
"LINEPROTO-5-UPDOWN","10.25.32.95","GigabitEthernet3/0/19",72
"LINEPROTO-5-UPDOWN","10.25.32.95","GigabitEthernet3/0/21",70
```

That suffices. Simple GUI way to solve the problem.

For how to take the CSV file and create a pivot table in Excel, see below.

Search Engine or Bust

Since the pivot table task seems like a database report, I thought I'd see if there's a way to do the entire task within Elasticsearch. Please note, I've committed SQL when I had to, but am not fond of working with database queries. But for you, I attempted it!

After some experimentation using Postman and CLI, based on the Elasticsearch search and aggregation documentation, I got what I was looking for.

The following CLI command pulled data from Elasticsearch via the REST API.

```
curl -s \
-H "Content-Type: application/json" \
-X POST http://10.0.0.22:9200/sys20/_search?pretty \
-d ' {
    "size" : 0,
    "aggs": {
        "errcodecounts": {
            "terms": {
                "field": "errcode",
                "size": 500
            },
            "aggs": {
                "ipaddresscounts" : {
                    "terms": {
                        "field": "ipaddress",
                        "size": "500"
                    },
                    "aggs": {
                        "interfacecounts" : {
                            "terms": {
                                "field": "interface",
                                "size": "500"
                            }
                        }
                    }
                }
            }
        }
    }
}'
```

Note that you only get 10 results unless you specify more. The above begs the question of whether 500 of each bucket is enough, and do I need some form of sorting to ensure I get the biggest counts. I have not yet tried hard to answer that.

Here's an excerpt from the output, which I captured to file:

```
{
  "took" : 8,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "gte"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "errcodecounts" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "UPDOWN",
          "doc_count" : 19079,
          "ipaddresscounts" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "10.25.32.129",
                "doc_count" : 1212,
                "interfacecounts" : {
                  "doc_count_error_upper_bound" : 0,
                  "sum_other_doc_count" : 0,
                  "buckets" : [
                    {
                      "key" : "GigabitEthernet5/0/7",
                      "doc_count" : 1188
                    },
                    {
                      "key" : "GigabitEthernet5/0/24",
                      "doc_count" : 24
                    }
                  ]
                }
              },
              {
                "key" : "10.25.32.95",
                "doc_count" : 1160,
                "interfacecounts" : {
                  "doc_count_error_upper_bound" : 0,
                  "sum_other_doc_count" : 0,
                  "buckets" : [
                    {
                      "key" : "GigabitEthernet2/0/36",
                      "doc_count" : 222
                    },
```

Yes, that's long-winded JSON. I wrote some hasty python to re-format it as needed:

```
#! /usr/bin/env python
import sys
import simplejson as json
from pprint import pprint

filename='agg-results2.txt'

with open(filename) as f:
    info=json.load(f)
    # NOTE: errors will occur if you don't clean up junk at the beginning
and end
    # of what was captured.
ilist = info['aggregations']['errcodecounts']['buckets']

# Assumption: every syslog message will have an error code and an IP
address
# but possibly not associated interfaces
print('Errcode', '\t', 'IP address', '\t', 'Interface', '\t', 'Count')

for i in ilist:
    jlist=i['ipaddresscounts']['buckets']
    for j in jlist:
        klist=j['interfacecounts']['buckets']
        if len(klist)==0:
            print(i['key'], '\t', j['key'], '\t', 'N/A', '\t',
j['doc_count'])
        else:
            for k in klist:
                print(i['key'], '\t', j['key'], '\t', k['key'], \
                    '\t', k['doc_count'])
```

The output from that, tab-separated, looks like:

```
Errcode    IP address        Interface   Count
```

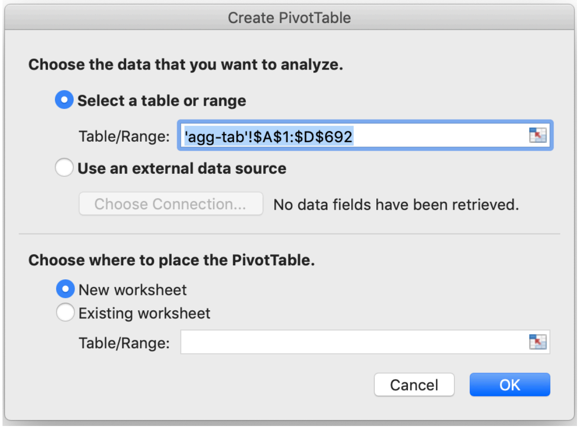
```
UPDOWN     10.25.32.129      GigabitEthernet5/0/7    1188
UPDOWN     10.25.32.129      GigabitEthernet5/0/24   24
UPDOWN     10.25.32.95       GigabitEthernet2/0/36   222
UPDOWN     10.25.32.95       GigabitEthernet3/0/17   178
```

Excelling That

Let’s walk through converting that into an Excel pivot table, as I don’t think most readers will have had that pleasure. It can look a bit obscure the first time you try it.

I opened the above text output file with Excel. Select all the data and go to Data -> Summarize with Pivot Table (on a Mac, that’s in the top menu bar, not the menus in the Excel window).

You should see the following:



Create PivotTable

Choose the data that you want to analyze.

● Select a table or range

Table/Range: 'agg-tab'!$A$1:$D$692

○ Use an external data source

No data fields have been retrieved.

Choose where to place the PivotTable.

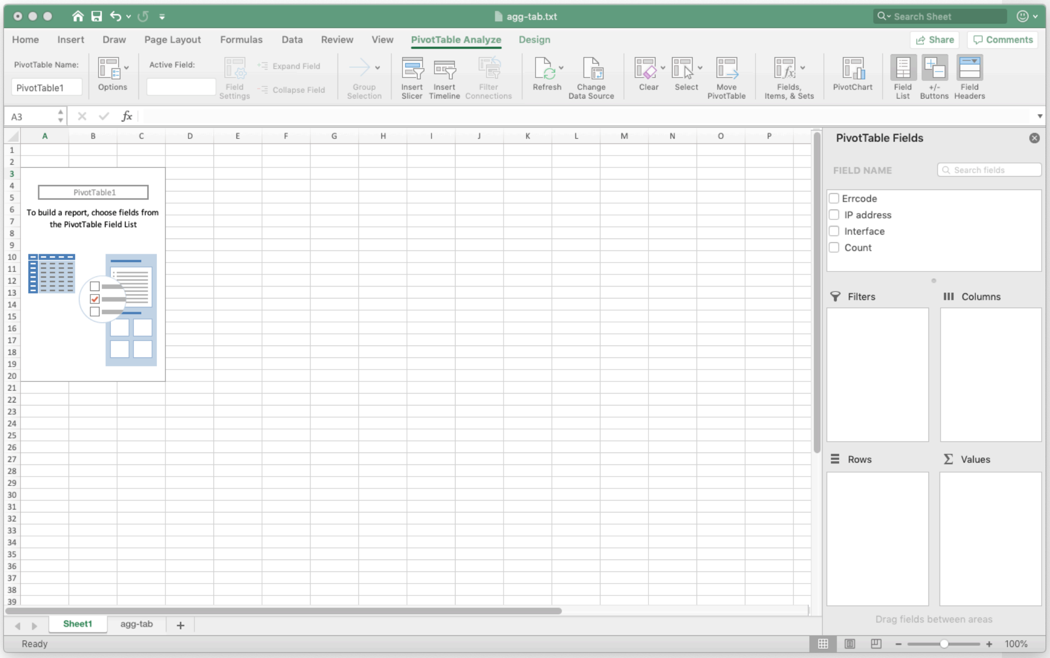
● New worksheet

○ Existing worksheet

Table/Range:

I usually just click OK on this.

You should see something like the following next:



To build a report, choose fields from the PivotTable Field List

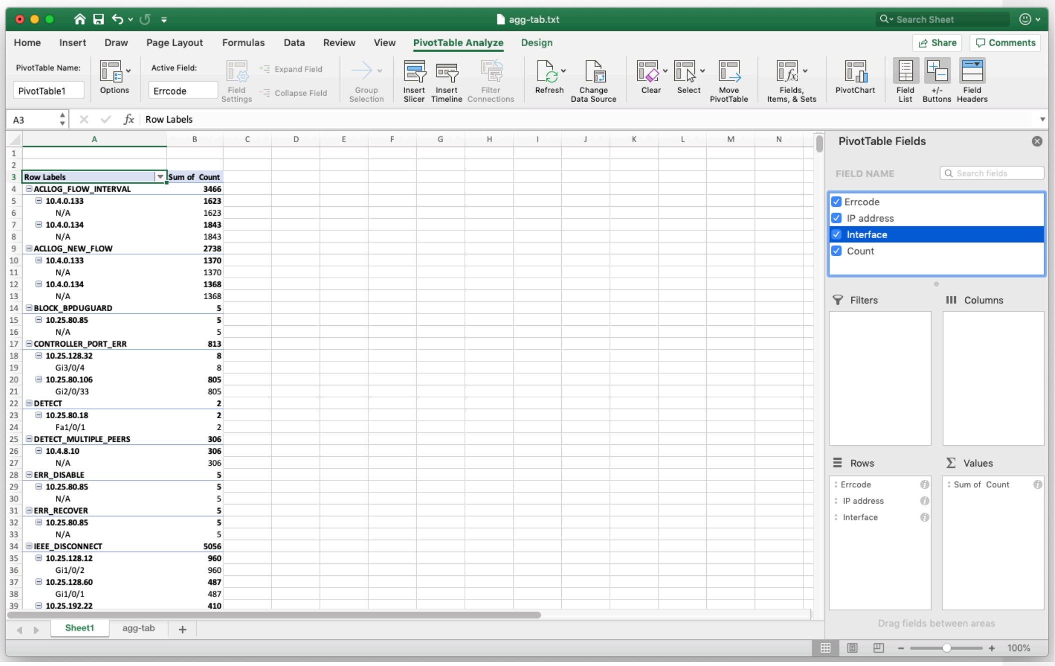
□ Errcode
□ IP address
□ Interface
□ Count

Carefully drag, in order, “errcode”, then “IP Address”, then “interface” from the top right to the Rows box on the bottom right. You can drag them to rearrange them into that order in the box if they didn’t end up in the specified order.

Hopefully you did that without anything ending up in the Values box. If you see Count already there, you’ll need to uncheck the boxes at the top right and try again.

Drag “Count” to the Values (Sum of values) box on the bottom right.

You should now see this:



A3 Row Labels

	Row Labels	Sum of Count
4	⊟ ACLLOG_FLOW_INTERVAL	3466
5	⊟ 10.4.0.133	1623
6	N/A	1623
7	⊟ 10.4.0.134	1843
8	N/A	1843
9	⊟ ACLLOG_NEW_FLOW	2738
10	⊟ 10.4.0.133	1370
11	N/A	1370
12	⊟ 10.4.0.134	1368
13	N/A	1368
14	⊟ BLOCK_BPDUGUARD	5
15	⊟ 10.25.80.85	5
16	N/A	5
17	⊟ CONTROLLER_PORT_ERR	813
18	⊟ 10.25.128.32	8
19	Gi3/0/4	8
20	⊟ 10.25.80.106	805
21	Gi2/0/33	805
22	⊟ DETECT	2
23	⊟ 10.25.80.18	2
24	Fa1/0/1	2
25	⊟ DETECT_MULTIPLE_PEERS	306
26	⊟ 10.4.8.10	306
27	N/A	306
28	⊟ ERR_DISABLE	5
29	⊟ 10.25.80.85	5
30	N/A	5
31	⊟ ERR_RECOVER	5
32	⊟ 10.25.80.85	5
33	N/A	5
34	⊟ IEEE_DISCONNECT	5056
35	⊟ 10.25.128.12	960
36	Gi1/0/2	960
37	⊟ 10.25.128.60	487
38	Gi1/0/1	487
39	⊟ 10.25.192.22	410

PivotTable Fields

FIELD NAME

☑ Errcode
☑ IP address
☑ Interface
☑ Count

Filters

Columns

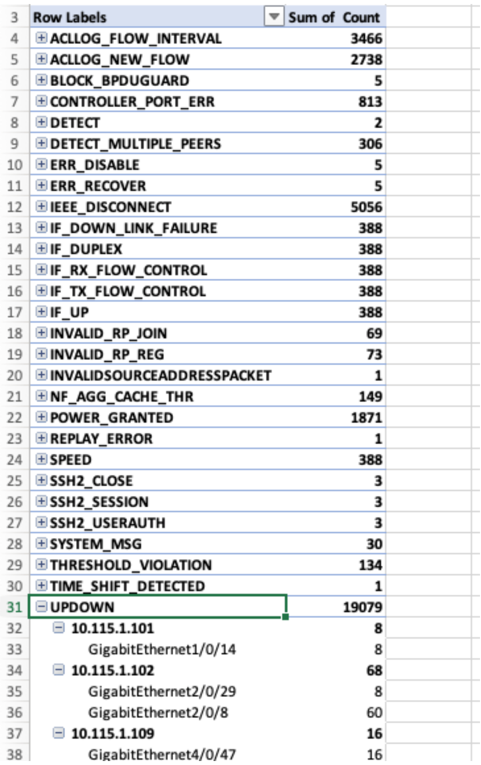
Rows: Errcode, IP address, Interface

Values: Sum of Count

You can click on the minus (–) signs on the left to collapse any of the blocks. Click on a plus (+) sign to expand.

I usually click in the top left title bar box as shown above, then go up to the menu bar at the top and click on Collapse Field, then go back down to the pivot table and click on '+' to expand one field at a time.

Here's how that looks:



Row Labels	Sum of Count
⊞ ACLLOG_FLOW_INTERVAL	3466
⊞ ACLLOG_NEW_FLOW	2738
⊞ BLOCK_BPDUGUARD	5
⊞ CONTROLLER_PORT_ERR	813
⊞ DETECT	2
⊞ DETECT_MULTIPLE_PEERS	306
⊞ ERR_DISABLE	5
⊞ ERR_RECOVER	5
⊞ IEEE_DISCONNECT	5056
⊞ IF_DOWN_LINK_FAILURE	388
⊞ IF_DUPLEX	388
⊞ IF_RX_FLOW_CONTROL	388
⊞ IF_TX_FLOW_CONTROL	388
⊞ IF_UP	388
⊞ INVALID_RP_JOIN	69
⊞ INVALID_RP_REG	73
⊞ INVALIDSOURCEADDRESSPACKET	1
⊞ NF_AGG_CACHE_THR	149
⊞ POWER_GRANTED	1871
⊞ REPLAY_ERROR	1
⊞ SPEED	388
⊞ SSH2_CLOSE	3
⊞ SSH2_SESSION	3
⊞ SSH2_USERAUTH	3
⊞ SYSTEM_MSG	30
⊞ THRESHOLD_VIOLATION	134
⊞ TIME_SHIFT_DETECTED	1
⊟ UPDOWN	19079
⊟ 10.115.1.101	8
GigabitEthernet1/0/14	8
⊟ 10.115.1.102	68
GigabitEthernet2/0/29	8
GigabitEthernet2/0/8	60
⊟ 10.115.1.109	16
GigabitEthernet4/0/47	16

The point of this exercise is that you can easily see which errors have the biggest error counts, which device IP's have the biggest counts, and for each device, which interfaces have the biggest error counts, for each type of error.

Expanding one or a few errors at a time lets you figure out what you need to go fix.

A Better Query

In looking around and thinking about sort and "how can I ensure I have it all" (error results, that is), I came across a better approach, Elasticsearch "composites".

```
curl -s \
-H "Content-Type: application/json" \
-X POST http://10.0.0.22:9200/sys20/_search?pretty \
-d ' {
    "size" : 0,
    "track_total_hits" : false,
    "aggs": {
        "superbuckets": {
            "composite": {
                "size": 10000,
                "sources" : [
                    { "byerrcode": { "terms" : { "field" : "errcode" } }
},
                    { "byipaddress": { "terms" : { "field" : "ipaddress"
} } },
                    { "byinterface": { "terms" : { "field" : "interface"
, "missing_bucket" : "true" } } }
                ]
            }
        }
    }
}'
```

My impression is that this should produce full output.

The “missing_bucket” item is in there because some error messages don’t have interfaces associated with them.

When you run this and capture the output, the format is a bit different than before. Here’s a sample from the beginning of the output:

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "superbuckets" : {
      "after_key" : {
        "byerrcode" : "XPPEDCINTERRUPTPRMERR",
        "byipaddress" : "10.1.13.64",
        "byinterface" : null
      },
      "buckets" : [
        {
          "key" : {
            "byerrcode" : "ACLLOG_FLOW_INTERVAL",
            "byipaddress" : "10.4.0.133",
            "byinterface" : null
          },
          "doc_count" : 1623
        },
```

The Python needed to reformat that was different (simpler!):

```
#! /usr/bin/env python
import sys
import simplejson as json
from pprint import pprint

filename='agg-results3.txt'

with open(filename) as f:
    info=json.load(f)

# NOTE: errors will occur if you don't clean up junk at the beginning and
end
# of what was captured.

ilist = info['aggregations']['superbuckets']['buckets']

# Assumption: every syslog message will have an error code and an IP
address
# but possibly not associated interfaces

print('Errcode', '\t', 'IP address', '\t', 'Interface', '\t', 'Count')

for i in ilist:
    k=i["key"]
    print(k['byerrcode'], '\t', k['byipaddress'], '\t', k['byinterface'],
\
        '\t', i['doc_count'])
```

Other than order, the output is the same as that shown earlier (comparing the two on my first attempt demonstrated the need for the missing_bucket flag).

Opening the result in Excel for the pivot table can be done just as before.

Exercise for the reader: I'd captured most of this before I went back and figured out the pattern for full error codes. Figure out how to update the above to product a pivot table with full error codes of the form SYS-5-UPDOWN (or whatever) in the pivot table.

Hint: it should not be hard! Just replace occurrences of "errcode" with "fullerrcode" and run against an Index where the "fullerrcode" pattern was applied.

Considering Alternatives

As we've seen, I was able to get ELK to more or less reproduce what my syslog PERL script did, in two different ways. It will likely be another tool in my toolbox, one that can be used in certain settings.

There are some trade-offs involved here. The ELK stack approach is fast and powerful, and easier to tune. On the other hand, I've been able to fairly quickly process 100 GB or more of syslog data with my PERL script.

Using the GUI is manual. The API via curl approach coupled with Python could easily be automated. Automating the Excel part is likely possible, but outside my intended scope here.

One key factor with using a PERL script was that I didn't have to load all the data into memory at one time. If you're running ELK in the cloud and the company is paying for it, storage might not be a problem. If you're running on your laptop, space / memory is a likely a constraint (darn, I can't replace my script!).

How that worked:

- I could unzip one unit of data, syslog from one day at a large company.
- Then do a preliminary run on the first part of the data (Linux head command piped to script).
- That would let me write a Linux grep filter to remove noise (like most of the ASA ACL rule logging events, and some other chatty items). Usually eliminating three (3) to ten (10) chatty messages reduced the log file size by 2 to 3 orders of magnitude (100 to 1000-fold!). Bonus: the chatty messages usually are of little value.
- I could pipeline Linux grep commands into the PERL script to avoid having intermediate files touching disk.
- Repeat as needed to get to a workable amount of data on disk. Then concatenate & process fully.

While you could also do something like that with ELK, even in a container, when it comes to aggregating the reduced pieces, the end result might still be too large to process at once (RAM or virtual disk within the container). Processing in pieces with ELK and aggregating the piecemeal results is one alternative.

Conclusion

I hope this blog has been useful, for showing several ways to get at data within Elasticsearch. A secondary goal was to share the PivotTable reporting, which I've found useful.

Thanks to Nikolay, for showing the value and getting me started working with PivotTable in this context.

References

See also the Part 1 to 3 References, which I have elected to not repeat below.

- Elasticsearch Documentation: Bucket Aggregations: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket.html
- Elasticsearch Documentation: Composite Aggregation: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-composite-aggregation.html
- Elasticsearch Blog about Composite Aggregation:
- https://www.elastic.co/blog/composite-aggregations-elasticsearch-pizza-delivery-metrics

Comments

Comments are welcome, both in agreement or constructive disagreement about the above. I enjoy hearing from readers and carrying on deeper discussion via comments. Thanks in advance!

Hashtags: #CiscoChampion #TechFieldDay #TheNetCraftsmenWay

Twitter: @pjwelcher

Disclosure Statement

[INSERT the usual IMAGES HERE: 20 Year CCIE and Cisco Champions **2019** as per recent blogs]

NetCraftsmen Services

Did you know that NetCraftsmen does network /datacenter / security / collaboration design / design review? Or that we have deep UC&C experts on staff, including @ucguerilla? For more information, contact us at <<insert suitable link here>>.

Social Media:

Facebook: Like, comment or share our status using this link.

Twitter: Like and RT our tweet using this link.

LinkedIn: Like, comment or share our status using this link.