**Owen Galvin, CSCI E-88 Homework 7**

NOTE: I'm generally trying to cut down on screenshots in favor of raw text output in order to reduce size of final Word document, but sometimes screenshots tell a better story. The general formatting pattern is to use text like below for text that I enter into a console by itself:

echo $SPARK\_HOME

And then below for lines that includes both commands I entered, and also the output generated by those commands, with the text I entered in bold:

**(v34\_e88) [cloudera@localhost A6]$ spark-submit --master local[2] a6\_p1\_streaming.py**

17/11/04 11:46:33 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.2.12 instead (on interface enp0s3)

17/11/04 11:46:33 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

**Problem 1:**

* **Install ElasticSearch, Kibana and X-Pack - follow instructions from ElasticSeach official site:** [**https://www.elastic.co/guide/en/elasticsearch/reference/5.6/install-elasticsearch.html (Links to an external site.)Links to an external site.**](https://www.elastic.co/guide/en/elasticsearch/reference/5.6/install-elasticsearch.html)**-- either locally or on EC2/VM/Docker**

Create base target directory, wget the tar, and extract to that directory. Change ownership on the extracted directory to current user (cloudera)

**(v34\_e88) [cloudera@localhost A7]$ sudo mkdir /opt/elasticsearch**

**(v34\_e88) [cloudera@localhost A7]$ wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.4.tar.gz**

--2017-11-12 09:49:47-- https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.4.tar.gz

Resolving artifacts.elastic.co (artifacts.elastic.co)... 184.72.218.26, 54.243.108.41, 54.235.82.130, ...

Connecting to artifacts.elastic.co (artifacts.elastic.co)|184.72.218.26|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 33776770 (32M) [application/x-gzip]

Saving to: ‘elasticsearch-5.6.4.tar.gz’

100%[==================================================================>] 33,776,770 1.79MB/s in 28s

2017-11-12 09:50:15 (1.17 MB/s) - ‘elasticsearch-5.6.4.tar.gz’ saved [33776770/33776770]

(v34\_e88) [cloudera@localhost A7]$ sudo tar -zxvf elasticsearch-5.6.4.zip -C /opt/elasticsearch/tar (child): elasticsearch-5.6.4.zip: Cannot open: No such file or directory

tar (child): Error is not recoverable: exiting now

tar: Child returned status 2

tar: Error is not recoverable: exiting now

(v34\_e88) [cloudera@localhost A7]$ sudo tar -zxvf elasticsearch-5.6.4.tar.gz -C /opt/elasticsearch/

elasticsearch-5.6.4/

elasticsearch-5.6.4/lib/

elasticsearch-5.6.4/lib/elasticsearch-5.6.4.jar

elasticsearch-5.6.4/lib/lucene-core-6.6.1.jar

...

elasticsearch-5.6.4/modules/reindex/elasticsearch-rest-client-5.6.4.jar

elasticsearch-5.6.4/modules/reindex/commons-codec-1.10.jar

elasticsearch-5.6.4/modules/ingest-common/

elasticsearch-5.6.4/modules/ingest-common/ingest-common-5.6.4.jar

elasticsearch-5.6.4/modules/ingest-common/joni-2.1.6.jar

elasticsearch-5.6.4/modules/ingest-common/plugin-descriptor.properties

elasticsearch-5.6.4/modules/ingest-common/jcodings-1.0.12.jar

elasticsearch-5.6.4/modules/percolator/

elasticsearch-5.6.4/modules/percolator/percolator-5.6.4.jar

elasticsearch-5.6.4/modules/percolator/plugin-descriptor.properties

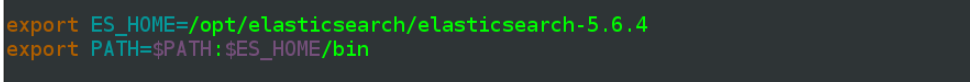
elasticsearch-5.6.4/plugins/

**sudo chown -R cloudera:cloudera /opt/elasticsearch/elasticsearch-5.6.4/**

[sudo] password for cloudera:

(v34\_e88) [cloudera@localhost A7]$

Edit my .bashrc and put the above location on the path



Install matching kibana version (5.6.4) following same pattern as above, wget on the package, create base target directory, and extract the tar.gz. The extraction overwhelmed my terminal's up-scroll, below is the ending part of initial configuration + some output from recent bash history. Finally, chown the directory.

...

kibana-5.6.4-linux-x86\_64/webpackShims/moment-timezone.js

kibana-5.6.4-linux-x86\_64/webpackShims/angular.js

kibana-5.6.4-linux-x86\_64/webpackShims/numeral.js

kibana-5.6.4-linux-x86\_64/webpackShims/moment.js

kibana-5.6.4-linux-x86\_64/webpackShims/sinon.js

kibana-5.6.4-linux-x86\_64/webpackShims/leaflet.js

kibana-5.6.4-linux-x86\_64/webpackShims/ace.js

kibana-5.6.4-linux-x86\_64/webpackShims/lodash.js

kibana-5.6.4-linux-x86\_64/webpackShims/gridster.js

kibana-5.6.4-linux-x86\_64/webpackShims/elasticsearch-browser.js

kibana-5.6.4-linux-x86\_64/webpackShims/jquery.js

kibana-5.6.4-linux-x86\_64/webpackShims/elasticsearch.js

kibana-5.6.4-linux-x86\_64/webpackShims/angular-ui-select.js

**(v34\_e88) [cloudera@localhost A7]$ history | tail -5**

2203 [2017-11-12 10:53:02] rm elasticsearch-5.6.4.tar.gz

2204 [2017-11-12 10:53:16] wget https://artifacts.elastic.co/downloads/kibana/kibana-5.6.4-linux-x86\_64.tar.gz

2205 [2017-11-12 10:54:29] sudo mkdir /opt/kibana

2206 [2017-11-12 10:55:00] sudo tar -zxvf kibana-5.6.4-linux-x86\_64.tar.gz -C /opt/kibana/

2208 [2017-11-12 10:55:49] history | tail -6

**(v34\_e88) [cloudera@localhost A7]$ sudo chown -R cloudera:cloudera /opt/kibana/kibana-5.6.4-linux-x86\_64/**

To get a browser GUI set up, begin by installing x-pack on elasticsearch per instructions on <https://www.elastic.co/guide/en/elasticsearch/reference/5.6/installing-xpack-es.html>,

**[cloudera@localhost elasticsearch-5.6.4]$ ./bin/elasticsearch-plugin install x-pack**

-> Downloading x-pack from elastic

[=================================================] 100%

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ WARNING: plugin requires additional permissions @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

\* java.io.FilePermission \\.\pipe\\* read,write

\* java.lang.RuntimePermission accessClassInPackage.com.sun.activation.registries

\* java.lang.RuntimePermission getClassLoader

\* java.lang.RuntimePermission setContextClassLoader

\* java.lang.RuntimePermission setFactory

\* java.security.SecurityPermission createPolicy.JavaPolicy

\* java.security.SecurityPermission getPolicy

\* java.security.SecurityPermission putProviderProperty.BC

\* java.security.SecurityPermission setPolicy

\* java.util.PropertyPermission \* read,write

\* java.util.PropertyPermission sun.nio.ch.bugLevel write

\* javax.net.ssl.SSLPermission setHostnameVerifier

See http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html

for descriptions of what these permissions allow and the associated risks.

Continue with installation? [y/N]y

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ WARNING: plugin forks a native controller @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

This plugin launches a native controller that is not subject to the Java

security manager nor to system call filters.

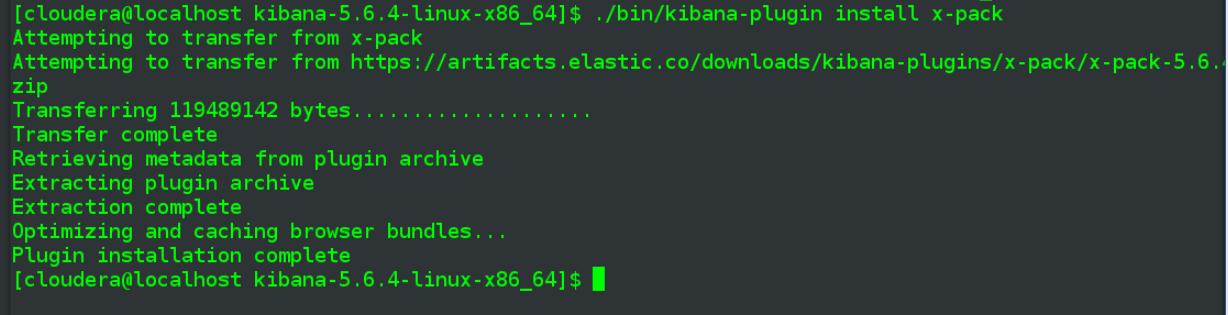
Continue with installation? [y/N]y

-> Installed x-pack

[cloudera@localhost elasticsearch-5.6.4]$

The x-pack instructions also imply it should be installed "on" kibana. From the kibana directory, run below, takes a while to finish up, screenshot for change of pace. Not going to bother with a KIBANA\_HOME in .bashrc

./bin/kibana-plugin install x-pack



Run elastic search with node.name and path.data arguments, as demonstrated in the lecture:

**[cloudera@localhost elasticsearch-5.6.4]$ elasticsearch -Enode.name=BestNodeNumberAwesome -Epath.data=data\_dir1**

[2017-11-12T11:20:06,294][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] initializing ...

[2017-11-12T11:20:06,479][INFO ][o.e.e.NodeEnvironment ] [BestNodeNumberAwesome] using [1] data paths, mounts [[/ (rootfs)]], net usable\_space [3.6gb], net total\_space [17.5gb], spins? [unknown], types [rootfs]

[2017-11-12T11:20:06,480][INFO ][o.e.e.NodeEnvironment ] [BestNodeNumberAwesome] heap size [1.9gb], compressed ordinary object pointers [true]

[2017-11-12T11:20:06,482][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] node name [BestNodeNumberAwesome], node ID [TOJ9ADOvR66BaJ2EuAW-8g]

[2017-11-12T11:20:06,483][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] version[5.6.4], pid[6809], build[8bbedf5/2017-10-31T18:55:38.105Z], OS[Linux/3.10.0-514.26.2.el7.x86\_64/amd64], JVM[Oracle Corporation/OpenJDK 64-Bit Server VM/1.8.0\_131/25.131-b12]

[2017-11-12T11:20:06,483][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupancyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+HeapDumpOnOutOfMemoryError, -Des.path.home=/opt/elasticsearch/elasticsearch-5.6.4]

[2017-11-12T11:20:09,245][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [aggs-matrix-stats]

[2017-11-12T11:20:09,266][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [ingest-common]

[2017-11-12T11:20:09,266][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [lang-expression]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [lang-groovy]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [lang-mustache]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [lang-painless]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [parent-join]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [percolator]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [reindex]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [transport-netty3]

[2017-11-12T11:20:09,267][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded module [transport-netty4]

[2017-11-12T11:20:09,268][INFO ][o.e.p.PluginsService ] [BestNodeNumberAwesome] loaded plugin [x-pack]

[2017-11-12T11:20:12,619][DEBUG][o.e.a.ActionModule ] Using REST wrapper from plugin org.elasticsearch.xpack.XPackPlugin

[2017-11-12T11:20:14,382][INFO ][o.e.x.m.j.p.l.CppLogMessageHandler] [controller/6850] [Main.cc@128] controller (64 bit): Version 5.6.4 (Build 324db309768a2c) Copyright (c) 2017 Elasticsearch BV

[2017-11-12T11:20:14,457][INFO ][o.e.d.DiscoveryModule ] [BestNodeNumberAwesome] using discovery type [zen]

[2017-11-12T11:20:15,782][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] initialized

[2017-11-12T11:20:15,782][INFO ][o.e.n.Node ] [BestNodeNumberAwesome] starting ...

[2017-11-12T11:20:16,485][INFO ][o.e.t.TransportService ] [BestNodeNumberAwesome] publish\_address {127.0.0.1:9300}, bound\_addresses {[::1]:9300}, {127.0.0.1:9300}

...

In separate terminal, run kibana

**[cloudera@localhost kibana-5.6.4-linux-x86\_64]$ ./bin/kibana**

log [16:20:27.820] [info][status][plugin:kibana@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:28.019] [info][status][plugin:elasticsearch@5.6.4] Status changed from uninitialized to yellow - Waiting for Elasticsearch

log [16:20:28.090] [info][status][plugin:xpack\_main@5.6.4] Status changed from uninitialized to yellow - Waiting for Elasticsearch

log [16:20:28.514] [info][status][plugin:graph@5.6.4] Status changed from uninitialized to yellow - Waiting for Elasticsearch

log [16:20:28.528] [info][status][plugin:monitoring@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:29.732] [warning][reporting] Generating a random key for xpack.reporting.encryptionKey. To prevent pending reports from failing on restart, please set xpack.reporting.encryptionKey in kibana.yml

log [16:20:29.739] [info][status][plugin:reporting@5.6.4] Status changed from uninitialized to yellow - Waiting for Elasticsearch

log [16:20:35.320] [info][status][plugin:xpack\_main@5.6.4] Status changed from yellow to yellow - No existing Kibana index found

log [16:20:35.321] [info][status][plugin:graph@5.6.4] Status changed from yellow to yellow - No existing Kibana index found

log [16:20:35.323] [info][status][plugin:reporting@5.6.4] Status changed from yellow to yellow - No existing Kibana index found

log [16:20:35.324] [info][status][plugin:elasticsearch@5.6.4] Status changed from yellow to yellow - No existing Kibana index found

log [16:20:37.861] [info][status][plugin:elasticsearch@5.6.4] Status changed from yellow to green - Kibana index ready

log [16:20:38.121] [info][license][xpack] Imported license information from Elasticsearch for [data] cluster: mode: trial | status: active | expiry date: 2017-12-12T11:20:21-05:00

log [16:20:38.137] [info][status][plugin:xpack\_main@5.6.4] Status changed from yellow to green - Ready

log [16:20:38.139] [info][status][plugin:graph@5.6.4] Status changed from yellow to green - Ready

log [16:20:38.140] [info][status][plugin:reporting@5.6.4] Status changed from yellow to green - Ready

log [16:20:38.146] [info][status][plugin:monitoring@5.6.4] Status changed from green to yellow - Waiting for Monitoring Health Check

log [16:20:42.118] [info][status][plugin:monitoring@5.6.4] Status changed from yellow to green - Ready

log [16:20:50.406] [info][status][plugin:security@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:50.407] [warning][security] Generating a random key for xpack.security.encryptionKey. To prevent sessions from being invalidated on restart, please set xpack.security.encryptionKey in kibana.yml

log [16:20:50.416] [warning][security] Session cookies will be transmitted over insecure connections. This is not recommended.

log [16:20:50.485] [info][status][plugin:searchprofiler@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:50.512] [info][status][plugin:ml@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:50.597] [info][status][plugin:ml@5.6.4] Status changed from green to yellow - Waiting for Elasticsearch

log [16:20:50.606] [info][status][plugin:ml@5.6.4] Status changed from yellow to green - Ready

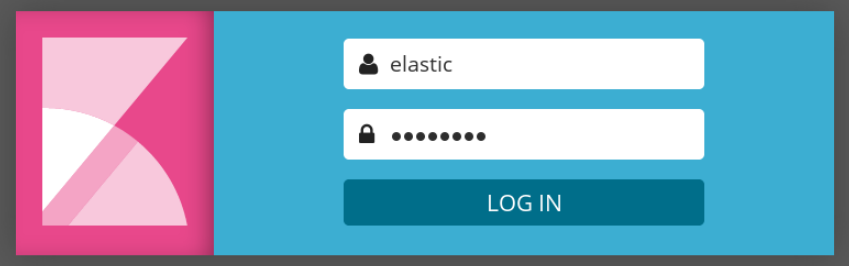
log [16:20:50.617] [info][status][plugin:tilemap@5.6.4] Status changed from uninitialized to green - Ready

log [16:20:50.625] [info][status][plugin:watcher@5.6.4] Status changed from uninitialized to green - Ready

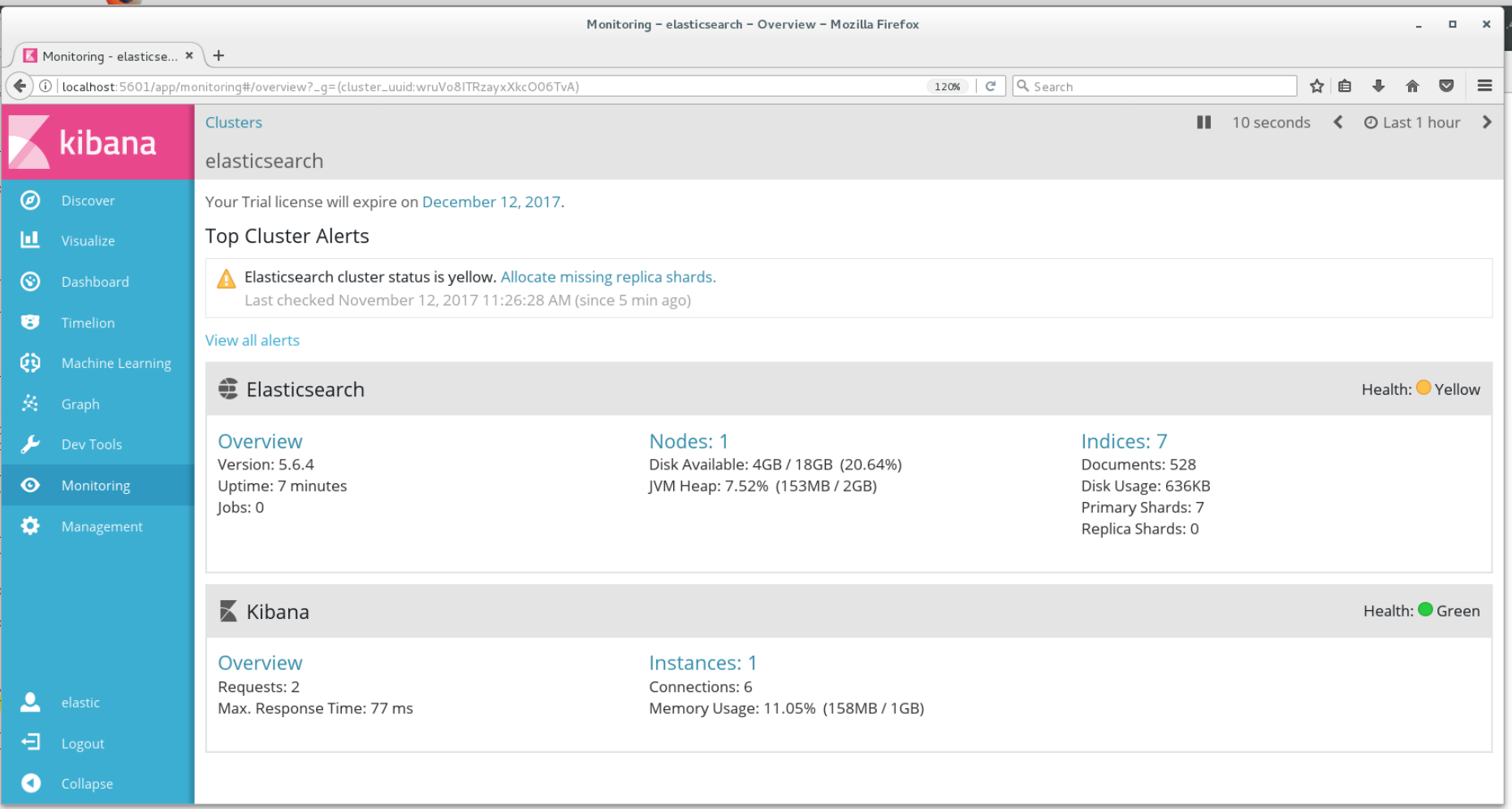
...

* **Demonstrate your cluster state by exploring "Monitoring" tab**

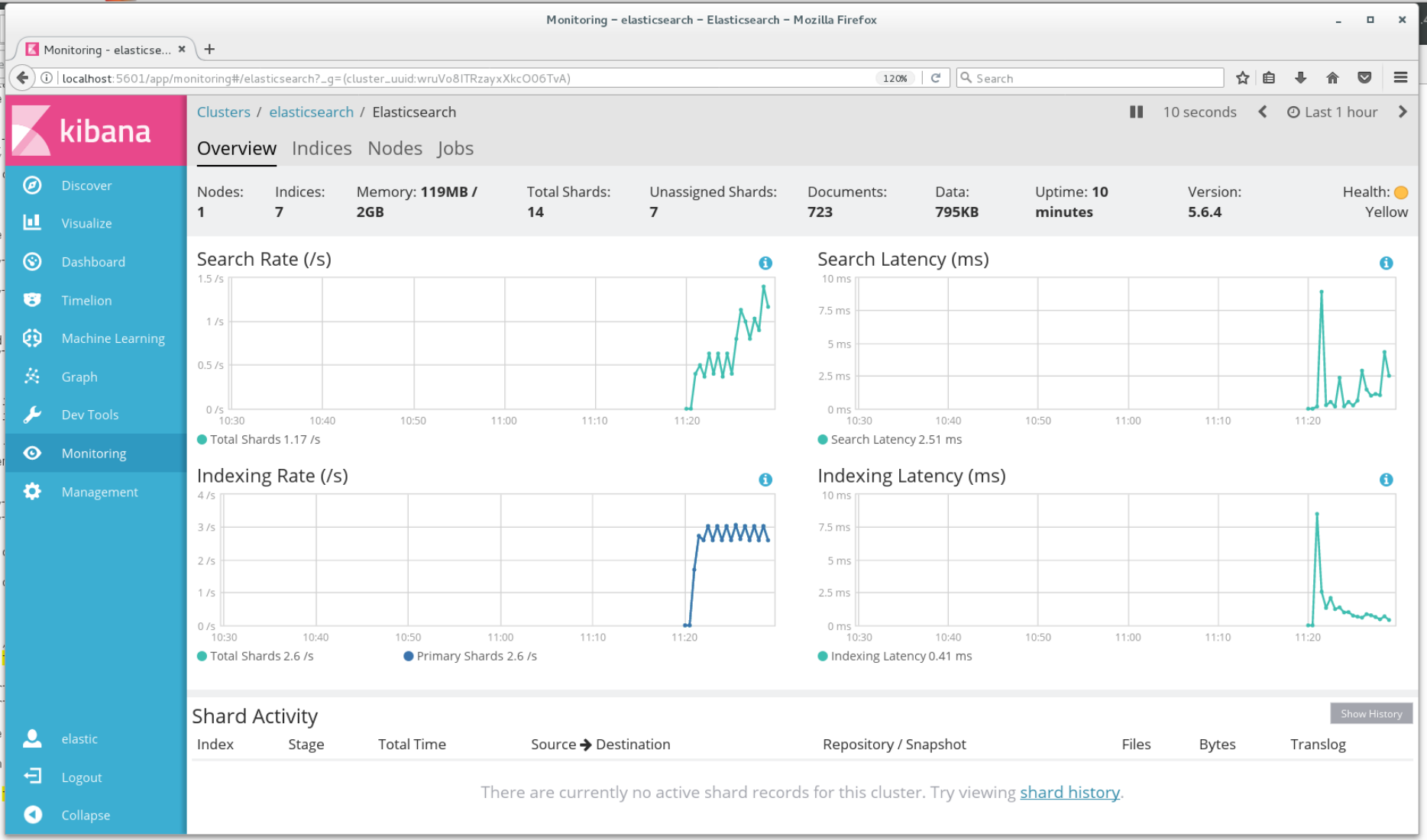
Navigate to <http://localhost:5601/> using Firefox on my VM, am greated by a login prompt, populate with default values per <https://www.elastic.co/guide/en/kibana/5.6/installing-xpack-kb.html>



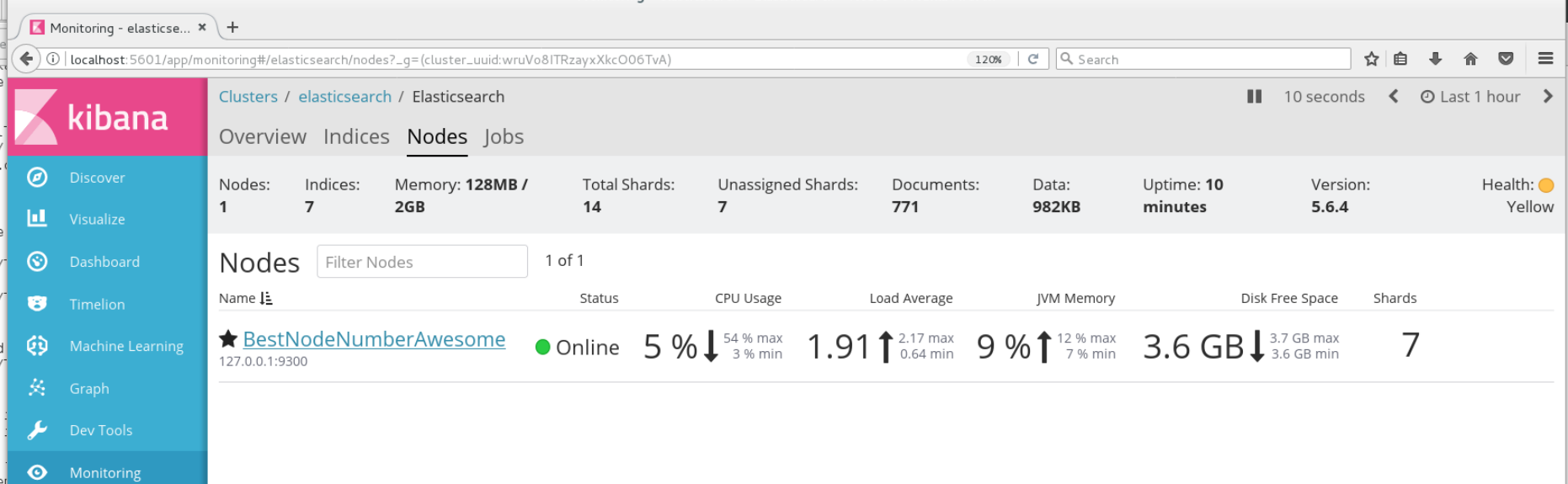
Click on Monitoring link from there, below is a snapshot of resulting overview



Click on the Overview link in Elasticsearch section and get a view that includes graphs of various measurements on the elasticsearch cluster



Click on Nodes link of above, BestNodeNumberAwesome reports as Status = Online.



**Problem 2:**

* **We will be using events in the format:**

**<uuid><timestamp><url><ua\_country><userId><ua\_browser><ua\_os><response\_status><TTFB>:**

**Where new fields are:**

**<ua\_browser>: should have values like: "Firefox", "Chrome", "IE", …**

**<ua\_os>:  should have values of a few operating systems like: "Mac OSX 10.2", "Windows ", "CentOS 5.3" …**

**<reponse\_status>: values of HTTP status like "200", "400", "401", "404", "500", "503", …**

* **design and create a mapping in ES; create an index with this mapping**

Below is the mapping I came up with - I initially created most fields as simple text but that proved to be inflexible when it came time to run searches later in the assignment, so I went back and re-created the index using the text-with-keyword pattern presented at <https://www.elastic.co/guide/en/elasticsearch/reference/current/fielddata.html>. The non-text fields include event\_time as date type with a specific datetime format expected (the same as my generated timestamps), response status of type = short, which will well cover the range of integer-type values that are expected, and finally ttfb\_ms is set as an integer.

|  |
| --- |
| {  **"mappings"** : {  **"doc"**: {  **"properties"**: {  **"uuid"**: {  **"type"**: **"text"** },  **"event\_time"**: {  **"type"**: **"date"**,  **"format"**: **"yyyy-MM-dd HH:mm:ss"** },  **"url"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  }  },  **"country"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  }  },  **"user\_id"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  }  },  **"browser"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  }  },  **"os"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  }  },  **"response\_status"**: {  **"type"**: **"short"** },  **"ttfb"**: {  **"type"**: **"integer"** }  }  }  } } |

Creating the index doesn't consist of more than pasting the above with a "PUT a7\_problem2" prefix into the Dev Tools console, and executing it. Which is what I did, below is a screenshot

|  |
| --- |
|  |

* **create/index a few events using CURL commands (PUT ....) - write a script to generate events with a few different values for each of specified event fields**

Below is the script, the idea is to simply print out several PUT commands to the console.

**p2\_es\_events.py**

|  |
| --- |
| **from** random **import** randint **import** collections **import** uuid   *# return an n count deque of timestamp string values in format of 'YYYY-MM-DD HH:MM:SS'* **def** get\_all\_timestamps(count):  Timepoint = collections.namedtuple(**'Timepoint'**, **'year month day hour minute second'**)  *# generate timepoints on Nov 9th, 2017, >= 6am and < 9am* time\_points = (Timepoint(year=2017, month=11, day=9,  hour=randint(6, 8), minute=randint(0, 59), second=randint(0, 59)) **for** i **in** range(count))   *# below doesn't handle February, but that's ok for this problem* **return** collections.deque(**'{year}-{month:02}-{day:02} {hour:02}:{minute:02}:{second:02}'**.format(  year=t.year, month=t.month, day=t.day **if** t.month **in** [4,6,9,11] **else** randint(1,31), hour=t.hour,  minute=t.minute, second=t.second) **for** t **in** time\_points)  *# generate a list of user-ids in format of u##, zero-filled from the left as appropriate to total count* **def** get\_userids(count):  **return** [**'u'** + str.zfill(**'{}'**.format(i), len(str(count))) **for** i **in** range(1, count+1)]  *# generate the list of events and insert them to hw6\_7 table* **def** generate\_events(num\_events):  *# same urls as previous problems* urls=[**'yahoo.com'**,**'google.com'**,**'harvard.edu'**]  *# total of six countries* countries=[**'Jamaica'**,**'Japan'**,**'Jordan'**,**'Fiji'**,**'Finland'**,**'France'**]  *# 5 possible browsers* browsers = [**'Opera'**,**'Firefox'**,**'Edge'**,**'Chrome'**,**'Vivaldi'**]  *# 6 operating system values* os = [**'Windows 7'**,**'Windows 10'**,**'Mac OS X 10.12'**,**'Mac OS X 10.10'**,**'Windows Me'**,**'CentOS 7.4'**]  *# 8 http response values* responses = [200,302,400,401,403,404,500,502]   user\_id\_count = 20  uids = get\_userids(user\_id\_count)  stamps = get\_all\_timestamps(num\_events)  events = []  **for** stamp **in** stamps:  *# A7, Problem 2, event format: <uuid><timestamp><url><ua\_country><userId><ua\_browser><ua\_os><response\_status><TTFB>* Event = collections.namedtuple(**'Event'**, **'event\_uuid event\_time url country user\_id browser os response ttfb\_ms'**)  *# randomly assign an url, country, user\_id, browser, os, response, and ttfb value to each event* events.append(Event(event\_uuid=uuid.uuid4(),  event\_time=stamp,  url=urls[randint(0, len(urls)-1)],  country=countries[randint(0, len(countries)-1)],  user\_id = uids[randint(0, user\_id\_count)-1],  browser=browsers[randint(0, len(browsers)-1)],  os=os[randint(0, len(os)-1)],  response=responses[randint(0, len(responses)-1)],  ttfb\_ms=randint(200, 400)))   *# iterate through the above events and create PUT strings for inserting the data into elasticsearch* **for** i, event **in** enumerate(events):  put\_command = **""" PUT /a7\_problem2/doc/{id} {{  "uuid": "{uuid}",  "event\_time": "{event\_time}",  "url": "{url}",  "country": "{country}",  "user\_id": "{user\_id}",  "browser": "{browser}",  "os": "{os}",  "response\_status": {response},  "ttfb": {ttfb\_ms} }} """**.format(id = str.zfill(str(i+1), 4), *# considered using uuid value here but figured an arbitrary value was more flexible* uuid=event.event\_uuid,  event\_time=event.event\_time,  url=event.url,  country=event.country,  user\_id=event.user\_id,  browser=event.browser,  os=event.os,  response=event.response,  ttfb\_ms=event.ttfb\_ms  )   print(put\_command)   generate\_events(10) |

Run the script, which is hard-coded for 10 events right now, first four events are below and these are the ones I'll insert first.

**(v34\_e88) [cloudera@localhost A7]$ python p2\_es\_events.py**

PUT /a7\_problem2/doc/0001

{

"uuid": "898f38e2-daa6-4e1c-94cd-87bc7066ac10",

"event\_time": "2017-11-09 08:29:06",

"url": "yahoo.com",

"country": "Jordan",

"user\_id": "u17",

"browser": "Opera",

"os": "Mac OS X 10.10",

"response\_status": 401,

"ttfb": 277

}

PUT /a7\_problem2/doc/0002

{

"uuid": "d2b558da-77cc-4f63-a815-048973b53d70",

"event\_time": "2017-11-09 06:09:58",

"url": "harvard.edu",

"country": "Finland",

"user\_id": "u03",

"browser": "Opera",

"os": "Mac OS X 10.12",

"response\_status": 302,

"ttfb": 242

}

PUT /a7\_problem2/doc/0003

{

"uuid": "5baaeb2b-f095-4d0d-8a16-fc3a7b763876",

"event\_time": "2017-11-09 08:38:39",

"url": "yahoo.com",

"country": "Finland",

"user\_id": "u08",

"browser": "Chrome",

"os": "Mac OS X 10.10",

"response\_status": 302,

"ttfb": 323

}

PUT /a7\_problem2/doc/0004

{

"uuid": "745423da-bf0f-49e2-9737-540ae2d5be70",

"event\_time": "2017-11-09 06:56:45",

"url": "yahoo.com",

"country": "Fiji",

"user\_id": "u12",

"browser": "Chrome",

"os": "Mac OS X 10.12",

"response\_status": 502,

"ttfb": 208

}

Paste into the Dev Tools Console and execute

|  |
| --- |
|  |

In case the idea was more to do a CURL from a regular terminal window I did the next two events as such, passing in creds for the default user.

**(v34\_e88) [cloudera@localhost A7]$ curl --user elastic:changeme -XPUT 'localhost:9200/a7\_problem2/doc/0005?pretty' -H 'Content-Type: application/json' -d'**

**> {**

**> "uuid": "13df90a2-7df0-4207-b543-2f140c319bc7",**

**> "event\_time": "2017-11-09 08:22:05",**

**> "url": "google.com",**

**> "country": "Jordan",**

**> "user\_id": "u01",**

**> "browser": "Chrome",**

**> "os": "CentOS 7.4",**

**> "response\_status": 500,**

**> "ttfb": 328**

**> }**

**> '**

{

"\_index" : "a7\_problem2",

"\_type" : "doc",

"\_id" : "0005",

"\_version" : 1,

"result" : "created",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"created" : true

}

**(v34\_e88) [cloudera@localhost A7]$ curl --user elastic:changeme -XPUT 'localhost:9200/a7\_problem2/doc/0006?pretty' -H 'Content-Type: application/json' -d'**

**> {**

**> "uuid": "f1681553-0839-458c-8010-561fec1b5b4d",**

**> "event\_time": "2017-11-09 08:08:43",**

**> "url": "yahoo.com",**

**> "country": "Jordan",**

**> "user\_id": "u06",**

**> "browser": "Edge",**

**> "os": "Mac OS X 10.12",**

**> "response\_status": 400,**

**> "ttfb": 345**

**> }**

**> '**

{

"\_index" : "a7\_problem2",

"\_type" : "doc",

"\_id" : "0006",

"\_version" : 1,

"result" : "created",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"created" : true

}

* **demo the events by querying the index and showing a few events**

Search for a single id out of the two "batches" I inserted using Console & then regular terminal, two records come back fine.

|  |
| --- |
|  |

**Problem 3:**

* **Write a program to generate and index events specified in Problem 2 - this time using Java API (or Python API) calls to ES to index the data**
* **Make sure you have at least 5-15 different values for each of the event fields (except UUID - UUID has to be, well, a UUID for each event :-) ), but vary it in time - say, 5 variations of URLs for one hour, than 10 for another hour, then back to 5… to simulate more"realistic" data where you do not see the same users/URLs all the time**

Install the python client into my virtual environment

**(v34\_e88) [cloudera@localhost A7]$ pip install elasticsearch**

Collecting elasticsearch

Downloading elasticsearch-5.5.1-py2.py3-none-any.whl (112kB)

100% |████████████████████████████████| 112kB 1.3MB/s

Collecting urllib3<1.22,>=1.21.1 (from elasticsearch)

Downloading urllib3-1.21.1-py2.py3-none-any.whl (131kB)

100% |████████████████████████████████| 133kB 1.9MB/s

Installing collected packages: urllib3, elasticsearch

Found existing installation: urllib3 1.22

Uninstalling urllib3-1.22:

Successfully uninstalled urllib3-1.22

Successfully installed elasticsearch-5.5.1 urllib3-1.21.1

**p3\_es\_api.py**

|  |
| --- |
| **from** random **import** randint **import** collections **import** uuid  **from** elasticsearch **import** Elasticsearch   *#Make sure you have at least 5-15 different values for each of the event fields (except UUID - UUID has to be, well, a UUID for each event :-) ), #but vary it in time - say, 5 variations of URLs for one hour, than 10 for another hour, then back to 5… to simulate more"realistic" data #where you do not see the same users/URLs all the time  # return an n count deque of timestamp string values in format of 'YYYY-MM-DD HH:MM:SS'* **def** get\_all\_timestamps(count):  Timepoint = collections.namedtuple(**'Timepoint'**, **'year month day hour minute second'**)  *# generate timepoints on Nov 9th, 2017, >= 2am and < 11am, increase hour span for problem 3, anticipating p4 queries* time\_points = (Timepoint(year=2017, month=11, day=9,  hour=randint(2, 11), minute=randint(0, 59), second=randint(0, 59)) **for** i **in** range(count))   *# below doesn't handle February, but that's ok for this problem* **return** collections.deque(**'{year}-{month:02}-{day:02} {hour:02}:{minute:02}:{second:02}'**.format(  year=t.year, month=t.month, day=t.day **if** t.month **in** [4,6,9,11] **else** randint(1,31), hour=t.hour,  minute=t.minute, second=t.second) **for** t **in** time\_points)  *# generate a list of user-ids in format of u##, zero-filled from the left as appropriate to total count* **def** get\_userids(count):  **return** [**'u'** + str.zfill(**'{}'**.format(i), len(str(count))) **for** i **in** range(1, count+1)]  *# generate the list of events and insert them to hw6\_7 table* **def** generate\_events(num\_events):  *# increase url count to 5* urls=[**'yahoo.com'**,**'google.com'**,**'harvard.edu'**,**'nytimes.com'**,**'cambridgema.gov'**,**'baidu.com'**,**'amazon.com'**,**'ebay.com'**,**'cnn.com'**]  *# total of six countries* countries=[**'Jamaica'**,**'Japan'**,**'Jordan'**,**'Fiji'**,**'Finland'**,**'France'**]  *# 5 possible browsers* browsers = [**'Opera'**,**'Firefox'**,**'Edge'**,**'Chrome'**,**'Vivaldi'**]  *# 6 operating system values* os = [**'Windows 7'**,**'Windows 10'**,**'Mac OS X 10.12'**,**'Mac OS X 10.10'**,**'Windows Me'**,**'CentOS 7.4'**]  *# 8 http response values* responses = [200,302,400,401,403,404,500,502,503,504]   es = Elasticsearch(http\_auth=(**'elastic'**, **'changeme'**))  user\_id\_count = 20  uids = get\_userids(user\_id\_count)  stamps = get\_all\_timestamps(num\_events)  events = []  **for** stamp **in** stamps:  *# A7, Problem 2, event format: <uuid><timestamp><url><ua\_country><userId><ua\_browser><ua\_os><response\_status><TTFB>* Event = collections.namedtuple(**'Event'**, **'event\_uuid event\_time url country user\_id browser os response ttfb\_ms'**)  *# randomly assign an url, country, user\_id, browser, os, response, and ttfb value to each event  # for Problem 3, don't know that is necessary but add in some logic to weight url/country/user ids by time  # all the timestamps are in hours 2-11 of one day, if the hour = 6 or 7, supply values from beginning of each list* stamp\_hour = stamp[12]  events.append(Event(event\_uuid=uuid.uuid4(),  event\_time=stamp,  url=urls[randint(0, len(urls)-3) **if** stamp\_hour **in** [**'5'**,**'6'**] **else** randint(0, len(urls)-1)],  country=countries[randint(0, len(countries)-4) **if** stamp\_hour **in** [**'5'**,**'6'**] **else** randint(0, len(countries)-1)],  user\_id = uids[randint(0, user\_id\_count)-11 **if** stamp\_hour **in** [**'5'**,**'6'**] **else** randint(0, user\_id\_count)-1],  browser=browsers[randint(0, len(browsers)-1)],  os=os[randint(0, len(os)-1)],  response=responses[randint(0, len(responses)-1)],  ttfb\_ms=randint(200, 400)))   *# iterate through the above events and create PUT strings for inserting the data into elasticsearch* **for** i, event **in** enumerate(events):  doc = **""" {{  "uuid": "{uuid}",  "event\_time": "{event\_time}",  "url": "{url}",  "country": "{country}",  "user\_id": "{user\_id}",  "browser": "{browser}",  "os": "{os}",  "response\_status": {response},  "ttfb": {ttfb\_ms} }} """**.format(uuid=event.event\_uuid,  event\_time=event.event\_time,  url=event.url,  country=event.country,  user\_id=event.user\_id,  browser=event.browser,  os=event.os,  response=event.response,  ttfb\_ms=event.ttfb\_ms  )   id = **'{}'**.format(str.zfill(str(i + 1), 4))  res = es.index(index=**'a7\_problem2'**, doc\_type=**'doc'**, id=id, body=doc)  print(**'id {} "created" status: {}'**.format(id, res[**'created'**]))  generate\_events(5000) |

Each of my event fields, aside from url, already had at least 5 values so I only needed to update the url list to add several more websites. Looking at queries in next problem I decide to add a few more 5xx response codes also.

I'm not sure how much the second bullet applied to my events as all the values, uuid aside, were already being randomly generated from within a certain set of defined values. Either way I weighted the values for url/country/user\_id so that if the timestamp hour = 6 or 7 (the entire range of hour values is 2 to 11) then the range of values would be restricted to the lower part of each source list, e.g. if 6 am all the userids would be from u01 through u10 while for 7 am - 8 am the full range of u01 through u20 would be possible.

Before invoking the new script, I run "DELETE a7\_problem2" in the Console, followed by the code in Problem 2 that re-creates the index with expected mapping.

Run the script, with the hardcoded number of events/creates = 5000

(v34\_e88) [cloudera@localhost A7]$ python p3\_es\_api.py

id 0001 "created" status: True

id 0002 "created" status: True

id 0003 "created" status: True

id 0004 "created" status: True

id 0005 "created" status: True

id 0006 "created" status: True

id 0007 "created" status: True

id 0008 "created" status: True

id 0009 "created" status: True

id 0010 "created" status: True

id 0011 "created" status: True

id 0012 "created" status: True

id 0013 "created" status: True

id 0014 "created" status: True

...

id 4984 "created" status: True

id 4985 "created" status: True

id 4986 "created" status: True

id 4987 "created" status: True

id 4988 "created" status: True

id 4989 "created" status: True

id 4990 "created" status: True

id 4991 "created" status: True

id 4992 "created" status: True

id 4993 "created" status: True

id 4994 "created" status: True

id 4995 "created" status: True

id 4996 "created" status: True

id 4997 "created" status: True

id 4998 "created" status: True

id 4999 "created" status: True

id 5000 "created" status: True

* **demo the events by querying the index and showing a few events**

Run a query in the Console for four of the id's that were inserted

|  |
| --- |
|  |

Full text results from the right pane above:

{

"took": 10,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"skipped": 0,

"failed": 0

},

"hits": {

"total": 4,

"max\_score": 1,

"hits": [

{

"\_index": "a7\_problem2",

"\_type": "doc",

"\_id": "0001",

"\_score": 1,

"\_source": {

"uuid": "8a8086d9-c62c-4ea6-b634-303d8bce34ab",

"event\_time": "2017-11-09 07:13:00",

"url": "amazon.com",

"country": "Jordan",

"user\_id": "u16",

"browser": "Edge",

"os": "CentOS 7.4",

"response\_status": 502,

"ttfb": 316

}

},

{

"\_index": "a7\_problem2",

"\_type": "doc",

"\_id": "0009",

"\_score": 1,

"\_source": {

"uuid": "f4847723-6b5b-4e7f-adff-b6254747d215",

"event\_time": "2017-11-09 07:00:05",

"url": "amazon.com",

"country": "France",

"user\_id": "u15",

"browser": "Opera",

"os": "Mac OS X 10.10",

"response\_status": 403,

"ttfb": 382

}

},

{

"\_index": "a7\_problem2",

"\_type": "doc",

"\_id": "1234",

"\_score": 1,

"\_source": {

"uuid": "d7cd4604-3c25-4c92-aab5-65b16691694b",

"event\_time": "2017-11-09 02:17:17",

"url": "nytimes.com",

"country": "Finland",

"user\_id": "u09",

"browser": "Vivaldi",

"os": "Mac OS X 10.10",

"response\_status": 400,

"ttfb": 266

}

},

{

"\_index": "a7\_problem2",

"\_type": "doc",

"\_id": "3333",

"\_score": 1,

"\_source": {

"uuid": "01ff18ce-8326-4056-8511-be6fc2eaabd8",

"event\_time": "2017-11-09 02:32:50",

"url": "harvard.edu",

"country": "Jamaica",

"user\_id": "u09",

"browser": "Vivaldi",

"os": "Windows 7",

"response\_status": 401,

"ttfb": 256

}

}

]

}

}

**Problem 4:**

* **Using "Dev Tools" tab/utility in Kibana (or using Postman or other HTTP request handling tools), create/execute/demo results of the following queries:**
  + **Q1:  Show popularity of different browsers in different countries (count of events per country per browser)**

Query on the left is executed in Conosle, screenshot first

|  |
| --- |
|  |

Followed by text of query & full text of output showing a grouping for each browser type, along with a sub-list of per-country hit counts, in descending order.

**POST /a7\_problem2/\_search?size=0**

**{**

**"aggs": {**

**"browser\_count": {**

**"terms": { "field": "browser.keyword" },**

**"aggs": {**

**"country\_count": {**

**"terms": { "field": "country.keyword" }**

**}**

**}**

**}**

**}**

**}**

**OUTPUT:**

{

"took": 14,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"skipped": 0,

"failed": 0

},

"hits": {

"total": 5000,

"max\_score": 0,

"hits": []

},

"aggregations": {

"browser\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Opera",

"doc\_count": 1035,

"country\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Japan",

"doc\_count": 215

},

{

"key": "Jordan",

"doc\_count": 215

},

{

"key": "Jamaica",

"doc\_count": 202

},

{

"key": "France",

"doc\_count": 144

},

{

"key": "Fiji",

"doc\_count": 140

},

{

"key": "Finland",

"doc\_count": 119

}

]

}

},

{

"key": "Vivaldi",

"doc\_count": 1001,

"country\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Jamaica",

"doc\_count": 213

},

{

"key": "Japan",

"doc\_count": 213

},

{

"key": "Jordan",

"doc\_count": 177

},

{

"key": "France",

"doc\_count": 140

},

{

"key": "Finland",

"doc\_count": 131

},

{

"key": "Fiji",

"doc\_count": 127

}

]

}

},

{

"key": "Firefox",

"doc\_count": 998,

"country\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Jordan",

"doc\_count": 210

},

{

"key": "Japan",

"doc\_count": 206

},

{

"key": "Jamaica",

"doc\_count": 193

},

{

"key": "Finland",

"doc\_count": 146

},

{

"key": "France",

"doc\_count": 129

},

{

"key": "Fiji",

"doc\_count": 114

}

]

}

},

{

"key": "Edge",

"doc\_count": 987,

"country\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Jordan",

"doc\_count": 205

},

{

"key": "Jamaica",

"doc\_count": 187

},

{

"key": "Japan",

"doc\_count": 182

},

{

"key": "Finland",

"doc\_count": 151

},

{

"key": "France",

"doc\_count": 136

},

{

"key": "Fiji",

"doc\_count": 126

}

]

}

},

{

"key": "Chrome",

"doc\_count": 979,

"country\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Jordan",

"doc\_count": 222

},

{

"key": "Jamaica",

"doc\_count": 207

},

{

"key": "Japan",

"doc\_count": 200

},

{

"key": "France",

"doc\_count": 140

},

{

"key": "Finland",

"doc\_count": 114

},

{

"key": "Fiji",

"doc\_count": 96

}

]

}

}

]

}

}

}

* + **Q2: find top 5 URLs that resulted in most error response codes (response code is in 500-599 range) (top 5 URLs where the max count of events with response code 5xx)**

Similar presentation format as Q1

|  |
| --- |
|  |

Run the query that first filters to only those with a response status >= 500, since there aren't status codes higher than > 599 don't need an upper-range cutoff. Aggregate by url and only display top 5 results.

**POST /a7\_problem2/\_search?size=0**

**{**

**"aggs": {**

**"url.keyword": {**

**"filter": {**

**"range": { "response\_status": {"gte": 500 }}},**

**"aggs": {**

**"url\_count": {"terms": {"field": "url.keyword", "size": 5}}}**

**}**

**}**

**}**

**OUTPUT:**

{

"took": 1,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"skipped": 0,

"failed": 0

},

"hits": {

"total": 5000,

"max\_score": 0,

"hits": []

},

"aggregations": {

"url": {

"doc\_count": 2067,

"url\_count": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 814,

"buckets": [

{

"key": "harvard.edu",

"doc\_count": 259

},

{

"key": "baidu.com",

"doc\_count": 255

},

{

"key": "yahoo.com",

"doc\_count": 251

},

{

"key": "cambridgema.gov",

"doc\_count": 247

},

{

"key": "google.com",

"doc\_count": 241

}

]

}

}

}

}

Quick sanity check on the range filter portion, the number of results from that alone matches the number reported in the full query above.

Query:

**GET /a7\_problem2/\_search**

**{**

**"query": {**

**"range": {**

**"response\_status": {**

**"gte": 500**

**}**

**}**

**}**

**}**

**OUTPUT:**

{

"took": 15,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"skipped": 0,

"failed": 0

},

"hits": {

"total": 2067,

"max\_score": 1,

"hits": [

{

"\_index": "a7\_problem2",

"\_type": "doc",

"\_id": "0028",

"\_score": 1,

"\_source": {

**Problem 5: (Points: 15)**

* using Dev Tools from X-Pack - profile the Problem 4 Q1 and Q2 queries

Move over to the Search Profiler tab section of Dev Tools. Enter the index name at the top left and then the base query text for Q1 into the box below that and click the Profile button.

|  |
| --- |
|  |

Cumulative Time: 0.581ms

* represents the cumulative time of all shards in the index. In a production setup index searches would normally be performed in parallel across multiple nodes, so this does not represent "wall-clock" time. (In a debug single node setup like this the 0.581 ms value is likely close to actual query time.)

[TOJ9ADOvR66BaJ2EuAW-8g][0] etc.

* I've expanded each of these items, which represent the per-shard performance
* to the far left are details for the second shard listed above
* shards are sorted by their execution time in descending order, so the one at the top was the slowest for this particular query, at this point in time

For profiling Q2, swap out the query text and replace with appropriate json lines, click Profile.

|  |
| --- |
|  |

General display details were discussed above, not really anything to add. For this one I do click Profile several more times and see that the time values fall in the same general time range, though that value is now lower than that first displayed, i.e. when I first profiled the query it reported 4.5ms but the below runs average to something around 1.8ms:

Cumulative Time: 1.595ms

Cumulative Time: 1.962ms

Cumulative Time: 1.807ms

Cumulative Time: 2.051ms

I believe I witnessed something similar with Q1, only I didn't take a screenshot right after the first Profile run so I can't be sure. I'm guessing the query is being cached after the initial run; these queries are being executed a couple of days after they were initially developed and run for Problem 4, so any caching that might have taken place back then had likely expired.

**Problem 6: (Points: 15)**

* **you can use "Dev Tools" for this problem, or any other HTTP-request handling tool**
* **create two indices - one per day: for example:** 
  + **date1 = Nov 17, 2017 → index name: problem6\_11172017**
  + **date2 = Nov 18, 2017 → index name: problem6\_11182017**

Since the events will be in same format I use the mapping from Problem2, and the same method for creating the index, below is partial screenshot.

|  |
| --- |
|  |

For the second index, substitute " problem6\_11182017" for that top line and execute again

{

"acknowledged": true,

"shards\_acknowledged": true,

"index": "problem6\_11182017"

}

* **generate a few events in the same format as in Problem2 and with event "timestamp" values falling into either date1 or date2**
* **Index events for date1 into the problem6\_11172017, and with date2 into the problem6\_11182017 indices**

Adjust the script from problem 2 so that it randomly assigns a generated timestamp to either 11/17/2017 or 11/18/2017 instead of a single day and then use that value to create the index name that gets output

|  |
| --- |
| put\_command = **""" PUT /problem6\_11{index\_day}2017/doc/{id} {{  "uuid": "{uuid}",  "event\_time": "{event\_time}",  "url": "{url}",  "country": "{country}",  "user\_id": "{user\_id}",  "browser": "{browser}",  "os": "{os}",  "response\_status": {response},  "ttfb": {ttfb\_ms} }} """**.format(index\_day = event.event\_time[8:10],  id = str.zfill(str( i +1), 4), *# considered using uuid value here but figured an arbitrary value was more flexible* uuid=event.event\_uuid,  event\_time=event.event\_time,  url=event.url,  country=event.country,  user\_id=event.user\_id,  browser=event.browser,  os=event.os,  response=event.response,  ttfb\_ms=event.ttfb\_ms  )  print(put\_command) |

Run updated script

**(v34\_e88) [cloudera@localhost A7]$ python p2\_es\_events.py**

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11172017/doc/0001

{

"uuid": "de75726e-6e4e-46cd-af4e-c52f7faaf050",

"event\_time": "2017-11-17 08:45:51",

"url": "google.com",

"country": "France",

"user\_id": "u02",

"browser": "Vivaldi",

"os": "Windows 10",

"response\_status": 200,

"ttfb": 301

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11172017/doc/0002

{

"uuid": "65a5f2ed-aabb-4b5f-ad31-fd62f35a6bfe",

"event\_time": "2017-11-17 09:59:32",

"url": "yahoo.com",

"country": "France",

"user\_id": "u08",

"browser": "Firefox",

"os": "Windows 10",

"response\_status": 404,

"ttfb": 221

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11182017/doc/0003

{

"uuid": "40096832-6dd6-4571-aefd-4fae09cbb907",

"event\_time": "2017-11-18 05:15:46",

"url": "google.com",

"country": "Jamaica",

"user\_id": "u10",

"browser": "Chrome",

"os": "Windows 10",

"response\_status": 500,

"ttfb": 233

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11182017/doc/0004

{

"uuid": "1c417cf0-43f1-4601-9b7c-13328c4fe6a2",

"event\_time": "2017-11-18 03:24:12",

"url": "harvard.edu",

"country": "Jamaica",

"user\_id": "u16",

"browser": "Chrome",

"os": "Windows Me",

"response\_status": 403,

"ttfb": 295

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11172017/doc/0005

{

"uuid": "10a74115-0aab-4bf5-bef5-4517aea8701f",

"event\_time": "2017-11-17 07:35:14",

"url": "harvard.edu",

"country": "Japan",

"user\_id": "u08",

"browser": "Opera",

"os": "Windows Me",

"response\_status": 400,

"ttfb": 303

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11172017/doc/0006

{

"uuid": "56f008c4-b232-4091-aff1-93c78fd825ad",

"event\_time": "2017-11-17 11:26:43",

"url": "yahoo.com",

"country": "Jordan",

"user\_id": "u20",

"browser": "Opera",

"os": "CentOS 7.4",

"response\_status": 502,

"ttfb": 359

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11182017/doc/0007

{

"uuid": "65de2ec2-267d-4a41-8d99-974aa35b0ebf",

"event\_time": "2017-11-18 04:13:56",

"url": "google.com",

"country": "Jordan",

"user\_id": "u20",

"browser": "Chrome",

"os": "CentOS 7.4",

"response\_status": 400,

"ttfb": 285

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11172017/doc/0008

{

"uuid": "60b796c4-a420-4c2e-815c-e0d575381632",

"event\_time": "2017-11-17 10:13:47",

"url": "harvard.edu",

"country": "France",

"user\_id": "u04",

"browser": "Vivaldi",

"os": "CentOS 7.4",

"response\_status": 500,

"ttfb": 249

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11182017/doc/0009

{

"uuid": "95465ba1-f01b-413e-9088-cd78883340b8",

"event\_time": "2017-11-18 11:25:47",

"url": "harvard.edu",

"country": "Jordan",

"user\_id": "u15",

"browser": "Firefox",

"os": "Windows 7",

"response\_status": 500,

"ttfb": 266

}

#### PROBLEM 6, one two days, new indices ####

PUT /problem6\_11182017/doc/0010

{

"uuid": "c55f43b6-ec2f-4fbe-a011-ee35a1744574",

"event\_time": "2017-11-18 02:19:05",

"url": "harvard.edu",

"country": "Fiji",

"user\_id": "u07",

"browser": "Edge",

"os": "Windows Me",

"response\_status": 502,

"ttfb": 255

}

Paste those into Console and execute, simply by counting the elasticsearch output it looks like each index got 5 out of the 10 events.

|  |
| --- |
|  |

* **Implement and demo results of the following query:**
  + **Q5: find all distinct response codes - across both indices**

Use a wildcard in the index part of the GET query to handle either index: problem6\_111\*2017

GET /problem6\_111\*2017/\_search?size=0

{

"aggs": {

"distinct\_respones": {

"terms": {

"field": "response\_status",

"size": 100

}

}

}

}

Execute that query (first query in below

|  |
| --- |
|  |

See six unique response\_status values, there are another two possible values that didn't happen to be part of the 10 inserted records. The counts 3+2+2+1+1+1 add up to 10 as expected.

Run a couple of sanity check queries - the same distinct query but on each of the separate indices. The per-index counts add up to 5 and there are few enough values that the results from the cross-index query can be manually confirmed.

|  |  |
| --- | --- |
| **# 11/17 index**  **GET /problem6\_11172017/\_search?size=0**  **{**  **"aggs": {**  **"distinct\_respones": {**  **"terms": {**  **"field": "response\_status",**  **"size": 100**  **}**  **}**  **}**  **}** | **# 11/18 index**  **GET /problem6\_11182017/\_search?size=0**  **{**  **"aggs": {**  **"distinct\_respones": {**  **"terms": {**  **"field": "response\_status",**  **"size": 100**  **}**  **}**  **}**  **}** |
| **OUTPUT:**  {  "took": 11,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 5,  "max\_score": 0,  "hits": []  },  "aggregations": {  "distinct\_respones": {  "doc\_count\_error\_upper\_bound": 0,  "sum\_other\_doc\_count": 0,  "buckets": [  {  "key": 200,  "doc\_count": 1  },  {  "key": 400,  "doc\_count": 1  },  {  "key": 404,  "doc\_count": 1  },  {  "key": 500,  "doc\_count": 1  },  {  "key": 502,  "doc\_count": 1  }  ]  }  }  } | **OUTPUT:**  {  "took": 0,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 5,  "max\_score": 0,  "hits": []  },  "aggregations": {  "distinct\_respones": {  "doc\_count\_error\_upper\_bound": 0,  "sum\_other\_doc\_count": 0,  "buckets": [  {  "key": 500,  "doc\_count": 2  },  {  "key": 400,  "doc\_count": 1  },  {  "key": 403,  "doc\_count": 1  },  {  "key": 502,  "doc\_count": 1  }  ]  }  }  } |

**Problem 7: (Points: 30)**

* **add geo-location fields to your events:**

**(useful reference for mappings: https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-point.html)**

**<uuid><timestamp><url><ua\_country><location><userId><ua\_browser><ua\_os><response\_status><TTFB>:**

**where the new field 'location' should contain two coordinates:**

**<location>: [lt, ln]**

* **create two new indices for events in this format - similar to Problem 6:** 
  + **problem7\_11172017 and problem7\_11182017**

The only difference in index creation vs. Problem 2 (and Problem 6) is the introduction of the location field into the mapping, new lines highlighted below.

|  |
| --- |
| ...  **"country"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  } }, **"location"**: {  **"type"**: **"geo\_point"** }, **"user\_id"**: {  **"type"**: **"text"**,  **"fields"**: {  **"keyword"**: {  **"type"**: **"keyword"** }  } },  ... |

Execute index creation using that mapping via PUT on each index name

PUT problem7\_11172017

<mapping>

PUT problem7\_11182017

<mapping>

Below are screenshots of the Console results, showing successful index creation

|  |  |
| --- | --- |
|  |  |

* **generate a few events in the new format and with event "timestamp" values falling into either date1 or date2; make sure you have some good variation of values for TTFB field (say, between 0.04 and 10.6 seconds)**

First install python Faker library into my virtual environment, will use that to create the Latitude/Longitude values

**(v34\_e88) [cloudera@localhost A7]$ pip install Faker**

Collecting Faker

Downloading Faker-0.8.7-py2.py3-none-any.whl (691kB)

100% |████████████████████████████████| 696kB 752kB/s

Requirement already satisfied: six in /home/cloudera/virtualenvs/v34\_e88/lib/python3.4/site-packages (from Faker)

Requirement already satisfied: python-dateutil>=2.4 in /home/cloudera/virtualenvs/v34\_e88/lib/python3.4/site-packages (from Faker)

Collecting text-unidecode (from Faker)

Downloading text\_unidecode-1.1-py2.py3-none-any.whl (77kB)

100% |████████████████████████████████| 81kB 2.9MB/s

Installing collected packages: text-unidecode, Faker

Successfully installed Faker-0.8.7 text-unidecode-1.1

Updated the script from Problem 3 to generate events of the specified type

**p7\_es\_geo\_api.py**

|  |
| --- |
| **from** random **import** randint **import** collections **import** uuid  **from** faker **import** Factory **from** elasticsearch **import** Elasticsearch   *# return an n count deque of timestamp string values in format of 'YYYY-MM-DD HH:MM:SS'* **def** get\_all\_timestamps(count):  Timepoint = collections.namedtuple(**'Timepoint'**, **'year month day hour minute second'**)  *# PROBLEM 6/7, events for either of two days in November, 11/17, 11/18, increase hour range to 1 to 11* time\_points = (Timepoint(year=2017, month=11, day=randint(17,18),  hour=randint(1, 11), minute=randint(0, 59), second=randint(0, 59)) **for** i **in** range(count))   *# below doesn't handle February, but that's ok for this problem* **return** collections.deque(**'{year}-{month:02}-{day:02} {hour:02}:{minute:02}:{second:02}'**.format(  year=t.year, month=t.month, day=t.day **if** t.month **in** [4,6,9,11] **else** randint(1,31), hour=t.hour,  minute=t.minute, second=t.second) **for** t **in** time\_points)  *# generate a list of user-ids in format of u##, zero-filled from the left as appropriate to total count* **def** get\_userids(count):  **return** [**'u'** + str.zfill(**'{}'**.format(i), len(str(count))) **for** i **in** range(1, count+1)]  *# generate the list of events and insert them to hw6\_7 table* **def** generate\_events(num\_events):  *# same urls as previous problems* urls=[**'yahoo.com'**,**'google.com'**,**'harvard.edu'**]  *# total of six countries* countries=[**'Jamaica'**,**'Japan'**,**'Jordan'**,**'Fiji'**,**'Finland'**,**'France'**]  *# 5 possible browsers* browsers = [**'Opera'**,**'Firefox'**,**'Edge'**,**'Chrome'**,**'Vivaldi'**]  *# 6 operating system values* os = [**'Windows 7'**,**'Windows 10'**,**'Mac OS X 10.12'**,**'Mac OS X 10.10'**,**'Windows Me'**,**'CentOS 7.4'**]  *# 8 http response values* responses = [200,302,400,401,403,404,500,502]   faker = Factory.create()  es = Elasticsearch(http\_auth=(**'elastic'**, **'changeme'**))  user\_id\_count = 20  uids = get\_userids(user\_id\_count)  stamps = get\_all\_timestamps(num\_events)  events = []  **for** stamp **in** stamps:  *# A7, Problem 2, event format: <uuid><timestamp><url><ua\_country><userId><ua\_browser><ua\_os><response\_status><TTFB>* Event = collections.namedtuple(**'Event'**, **'event\_uuid event\_time url location country user\_id browser os response ttfb\_ms'**)  *# randomly assign an url, country, user\_id, browser, os, response, and ttfb value to each event* events.append(Event(event\_uuid=uuid.uuid4(),  event\_time=stamp,  url=urls[randint(0, len(urls)-1)],  country=countries[randint(0, len(countries)-1)],  location=**'{lat},{lon}'**.format(lat=faker.latitude(), lon=faker.longitude()),  user\_id = uids[randint(0, user\_id\_count)-1],  browser=browsers[randint(0, len(browsers)-1)],  os=os[randint(0, len(os)-1)],  response=responses[randint(0, len(responses)-1)],  *# p7, increase the ttfb range to 0.04 - 11 seconds* ttfb\_ms=randint(40, 11000)))   *# iterate through the above events and create PUT strings for inserting the data into elasticsearch* **for** i, event **in** enumerate(events):  doc = **"""  {{  "uuid": "{uuid}",  "event\_time": "{event\_time}",  "url": "{url}",  "country": "{country}",  "location": "{location}",  "user\_id": "{user\_id}",  "browser": "{browser}",  "os": "{os}",  "response\_status": {response},  "ttfb": {ttfb\_ms}  }} """**.format(uuid=event.event\_uuid,  event\_time=event.event\_time,  url=event.url,  country=event.country,  location=event.location,  user\_id=event.user\_id,  browser=event.browser,  os=event.os,  response=event.response,  ttfb\_ms=event.ttfb\_ms  )   id = **'{}'**.format(str.zfill(str(i + 1), 4))  res = es.index(index=**'problem7\_11{index\_day}2017'**.format(index\_day = event.event\_time[8:10]),  doc\_type=**'doc'**,  id=id,  body=doc)  print(**'id {} "created" status: {}'**.format(id, res[**'created'**]))    generate\_events(50) |

The differences vs. Problem 3 involve

* restricting the date value to either 11/17 or 11/18
* increase the hour range on those dates to 1am to 11am
* use of Faker library to generate Latitude/Longitude values
* updating of the event/es.index() structure to handle those Lat/Long
* set total event count = 50
* **Index events for date1 into the problem7\_11172017, and with date2 into the problem7\_11182017 indices**

Run the script

**(v34\_e88) [cloudera@localhost A7]$ python p7\_es\_geo\_api.py**

id 0001 "created" status: True

id 0002 "created" status: True

id 0003 "created" status: True

id 0004 "created" status: True

id 0005 "created" status: True

id 0006 "created" status: True

...

Perform a query on each index to make sure no surprises, 24 went into 11/17 and remaining 26 went into 11/18 index.

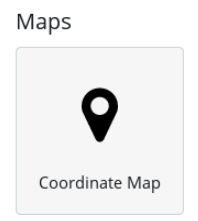
|  |  |
| --- | --- |
| GET problem7\_11172017/\_search  {  "query": {  "match\_all": {}  }  } | GET problem7\_11182017/\_search  {  "query": {  "match\_all": {}  }  } |
|  |  |

* **Using Kibana: create geohash grid visualization which shows locations where TTFB > 5 seconds - for all data from the problem7\_xxx indices**

Clicked on the Visualization link and am prompted to create an index pattern. Through some trial and error it appears I do NOT want to set a Time Filter field name, or at least skipping that step for now will make following steps more straightforward. (Later, in Problem 8 figured out what was going on.)

|  |
| --- |
|  |

Select the viz typ = Maps/Coordinate Map



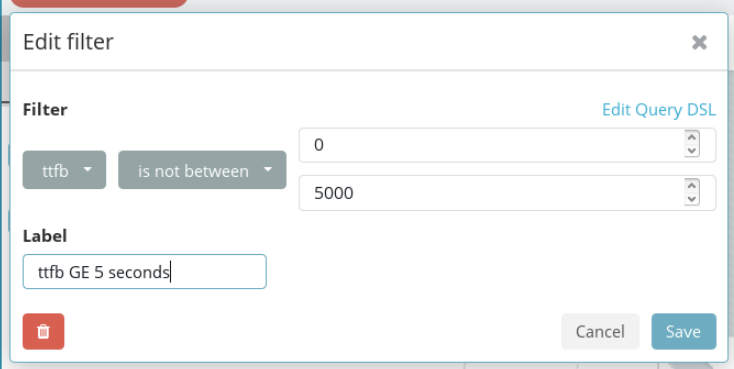
Select **problem7\_11\*2017** index pattern from the next dialog and a world map appears without any markers. Under Geo Coordinates on the left, select the field containing lat/long values, **location**.

|  |
| --- |
|  |

Click "Play" arrow to update and get all 50 points, across both indices, displayed on the world map



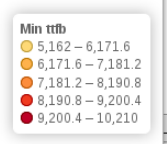
From there I'm going to make a Filter on the results, presumably the change could be made at the base query level but this seems the simplest approach. Select "Add a filter" link and fill out as below, choosing to filter out any data points that are <= 5 seconds.



Adjust the metrics section a little bit to report Min on ttfb and have the final viz, with the events that had longer ttfb being a darker color red, looks like the circle gets a bit larger also as ttfb values increase.

|  |
| --- |
|  |

Legend at the lower right indicates the values were put into 5 separate bins.



**Problem 8: (Points: 40)**

* **we will use the same indices /data created in Problem 7**
* **create Kibana Dashboard with the following visualizations:** 
  + **geohash grid created in Problem 7**

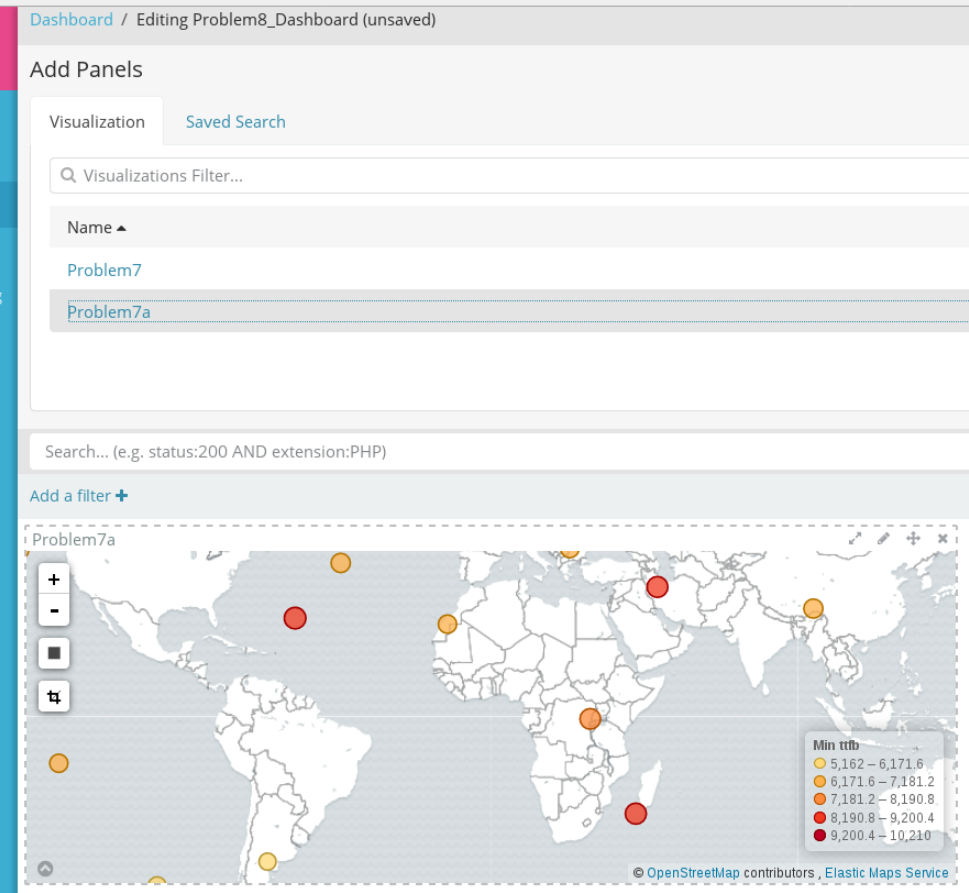
After starting to look at the Dashboard layout it looks like any Filter will affect all Visualizations, so I go into Problem7 viz, remove the filter and add a query in form of

NOT ttfb: [\* TO 5000]

and save that one as Problem7a.

(Then some deeper configuration necessary as I realize the reason my initial Index Pattern for Problem 7 wasn't showing results when I first created with a Time Filter field, was because the default setting of that filter was "w/in last 15 minutes". I go back now and create a new very similar index pattern, problem7\_11\*2017, that effectively spans the two indices and copy down the unique ID value elasticsearch has assigned to it. Via Management tab, edit the 7a visualization object so that it points to the new with-time-filter index pattern, using the unique ID I'd copied out. Now Problem7a visualization can work with the date time picker object.)

On Dashboard tab create a new Problem8\_Dashboard and add the Problem 7a visualization



* + **average TTFB**

Seems to be calling for only an average overall TTFB, so create a new visualization of type = Metric



Base it on problem7\_11\*2017 index pattern. On the ensuing viz configuration, select Average from the Metric -> Aggregation dropdown, followed by seleing Field = ttfb in the dropdown directly below. Give it a label and winds up looking like below.

|  |
| --- |
|  |

Save it as Problem8\_ttfb and add to the Dashboard I had started in earlier step.

* + **pie chart with count of request per response code**

Create new visualization, type = Pie, select problem7\_11\*2017 index pattern.

Under metrics, the default Slice Size = Count should remain. Then the Split Slices section is populated as below.

|  |
| --- |
|  |

This will do the aggregate counting on the response status field, ordering by that Term such that the 12 to 2 "clock" slice in the pie is for status code = 200 and then the numbers increase in clockwise order, with the final 502 slice appearing at 10 to 12. There are only 8 possible values (and only 6 wound up appearing in the two indices), but set a Size = 100 to future proof the viz.

|  |
| --- |
|  |

Add the above to the Dashboard, adjust the panels a bit and we have the final dashboard. It was somewhat unclear in the instructions as to how the > 5 second filter might be involved. Since I had to go through that re-work to get the query embedded in the geohash grid I've decided to leave that as-is but keep the "TTFB > 5s" filter out of the Average and Pie Chart visualizations - which means the two sets of viz elements won't match but also makes clear how this could be an issue.

|  |
| --- |
|  |

* **using Kibana's date/time picker, pick different time ranges within the date1-date2 timeframe; demo the results**

In the date/time picker, selct Absolute as the Time Range and run below queries

1. **2017-11-17 03:00:00 - 2017-11-17 10:00:00**

Enter those values directly into control

|  |
| --- |
|  |

Only two events fit both the > 5 seconds base query and also fall within the above range. The pie chart indicates six different values, so there must be at least 6 events that are within the time range, regardless of TTFB value.

|  |
| --- |
|  |

1. same steps for below range

**2017-11-18 01:00:00 - 2017-11-18 11:00:00**

This wider range returns 7 events that match both time & ttfb filter. Those in the time range have an overall average higher TTFB compared to the unfiltered time range. Also response status = 401 (bottom 5pm to 8pm in purple) is overrepresented as compared to the full dataset.

|  |
| --- |
|  |

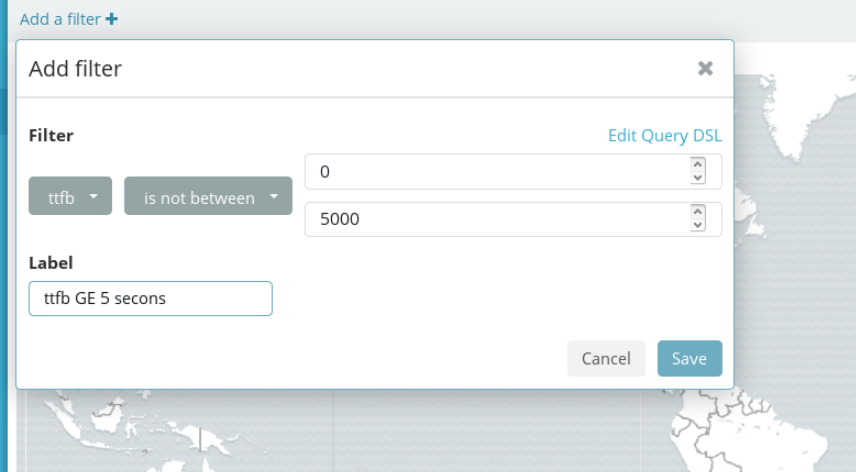
1. A very narrow time filter

**2017-11-18 02:30:00 - 2017-11-18 03:00:00**

Results in one hit that is in both ttfb and time range filters. The Pie Chart indicates there were at least two hits in only-time-range filter.

|  |
| --- |
|  |

At the Dashboard level, Add the same TTFB filter I initially introduced in Problem 7



Save the filter and indeed it is applied to all viz present in the dataset.

|  |
| --- |
|  |

The same data point remains in the map but now the pie is only one color since there is only one data point also in that viz. Addditional confirmation is in the Average TTFB - before it had been close to 5 seconds but now, restricted to only the event with > 5,000 milliseconds it shoots up, presumably = the value from that one event.

**Problem 8\_Extra: Bonus: +20**

* **add one more visualization to your P8 Dashboard:** 
  + **visualization of the Q2 query from Problem 4 ( top 5 URL’s with the most error response codes)**

The code I ran for problem 7 didn't have enough url values for "top 5" to make sense so I insert another batch of events with 6 new url's

|  |
| --- |
| urls = [**'nytimes.com'**, **'cambridgema.gov'**, **'baidu.com'**, **'amazon.com'**, **'ebay.com'**, **'cnn.com'**] |

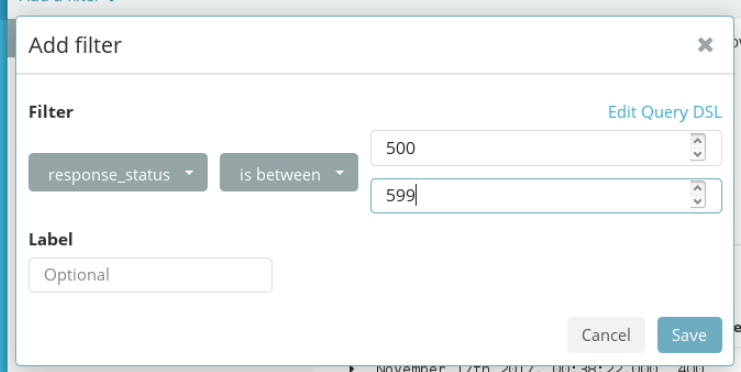
Run script again:

python p7\_es\_geo\_api.py

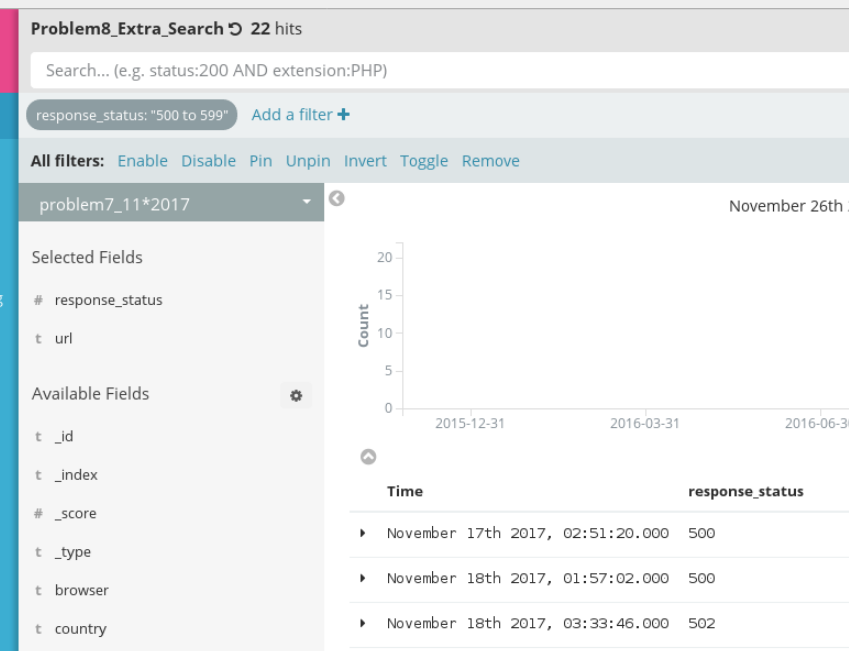
Within ES, results of Q2 look like below

|  |
| --- |
|  |

Tried starting with a Visualization from scratch but then decided to base off a Saved Search, need to create that first by beginning in Discover panel of kibana interface. Choose to base off of problem7\_11\*2017 index pattern, click "add a filter" at top and populate to restrict to 5\*\* response codes, i.e. those that are of some error type.



22 hits in that filter, add both response\_status and url fields



Save the search as Problem8\_Extra\_Search and create a new Visualization of Vertical bar chart type, select the above Saved Search from the right hand side. Under X-Axis section select "Terms" as the Aggregation type and for Field select "url.keyword", leave the rest of the options in default state.

|  |
| --- |
|  |

Note that the results match that of raw JSON query performed earlier in Console. Save visualization as " Problem8\_extra". Open the dashboard from earlier, add the above bar chart, move around the sub-elements a bit and save as Problem8\_Dashboard\_extra

|  |
| --- |
|  |