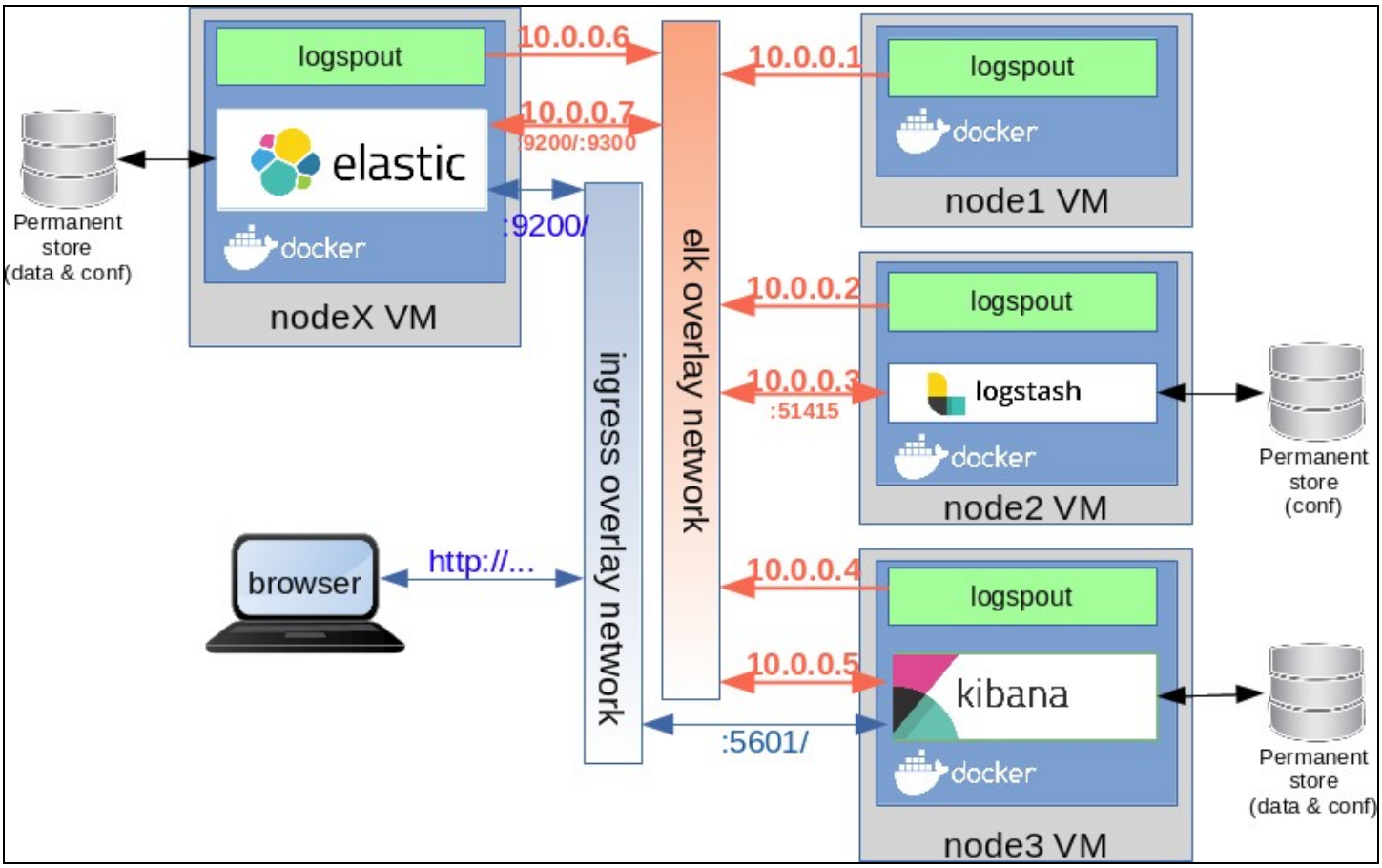
Centralized logging in swarm

<< Back to Docker main

Contents

Bevezet?

A cél az, hogy minden a swarm cluster-ben futó konténer logját összegy?jtsük egy központi, nagyon hatékonyan kereshet? adatbázisban, ami az Elasticsearch lesz. Az Elastichserach-be a szintén Elastic termékkel, a logStash-el fogjuk betölteni az adatokat. LogStash-b?l is csak egy darabra van szükség, nem kell minden node-on fusson egy konténer bel?le. A Logstash képes a sysout-ot átalakítani Elastichsearch dokumentumokra. A logStash-hez a logspout nev? program fogja elküldeni a sysout-ot minden node minden docker image-éb?l. A logspout rákapcsolódik a docker szoket-re, majd log proxiként továbbít minden logot a logStash konténernek. Értelem szer?en a logspout-ot minden egyes konténerre fel kell telepíteni.



A logokat a szintén Elastic termékkel, a Kibana-val fogjuk vizualizálni, elemezni. Az el?bb felsorolt 4 komponens az erre a célra létrehozott elk nev? overlay hálózaton fog egymással kommunikálni, ahol közvetlen el fogják egymást érni az overlay hálózatos IP címükkel. Mind az Elastichsearch, mind a Kibana webkonzol portját publikálni fogjuk az ingress hálózaton, hogy elérjük ?ket a böngész?b?l.

ElasticSearch

https://www.elastic.co/products/elasticsearch

Elasticsearch is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene. Since the first version of Elasticsearch was released in 2010, it has quickly become the most popular search engine, and is commonly used for log analytics, full-text search, and operational intelligence use cases.

When coupled with Kibana, a visualization tool, Elasticsearch can be used to provide near-real time analytics using large volumes of log data. Elasticsearch is also popular because of its easy-to-use search APIs which allow you to easily add powerful search capabilities to your applications.

It scales horizontally to handle kajillions of events per second, while automatically managing how indices and queries are distributed across the cluster for oh-so smooth operations

1. **Log Analytics** - Analyze un-structured and semi-structured logs generated by websites, mobile devices, servers, sensors, and more for a wide variety of applications such as digital marketing, application monitoring, fraud detection, ad tech, gaming, and IoT. Capture, pre-process, and load log data into Elasticsearch using Logstash, Amazon Kinesis Firehose, or Amazon CloudWatch Logs. You can then search, explore, and visualize the data using Kibana and the Elasticsearch query DSL to gain valuable insights about your users and applications.
2. **Full Text Search** - Provide your customers with a rich search and navigation experience. Elasticsearch supports faceting, which allows your customers to narrow their search results by value ranges for fields like price, product characteristics, and brands; ability to create advanced search criteria filters; search-as-you-type suggesters; and near real-time index updates.
3. **Distributed Document Store** - Power your application with an easy to use JSON document-oriented storage platform. Elasticsearch provides a simple REST API, fast performance, powerful search capabilities, so you can build highly performant applications that can store and retrieve billions of documents.
4. **Real-time Application Monitoring** - Capture activity logs across your customer-facing applications and websites. Use Logstash to push these logs to your Elasticsearch cluster. Elasticsearch indexes the data and makes it available for analysis in near real-time (less than one second). You can then use Kibana to visualize the data and perform operational analyses like identifying outages and problems. With Elasticsearch?s geospatial analysis, you can identify the geographical region where the problem is occurring. Troubleshooting teams can then search the index and perform statistical aggregations to identify root cause and fix issues.

Note
Nincs benne valódi tranzakció kezelés, ugyan vannak rá módszerek, hogy valami hasonlót meg lehessen valósítani

Logstash

A **Logstash** nem más mint egy log konverter. A LogStash-t fogjuk a logok betöltésére használni az Elasticsarch-be. Csak egyetlen egy példányt kell futtatni bel?le az egész cluster-ben. A Logsput-tól meg fogja kapni a docker konténerek system logját, amit konvertálni fog Elasticsarch formátumra, aztán el fogja küldeni az Elastaticsearch-nek.

Az Elastichsearch-ben a Minden megkapott log-sort konvertálni fog Elasticsearch dokumentumra, amit be fog szúrni egy logstash-<dátum> nev? index-be. Minden nap az aznapi index-be. Az Elasticsearch-ben ha egy document-et egy olyan indexbe akarunk beszúrni, ami még nem létezi, akkor létre fogja hozni azt az indexet a beszúrás el?tt az Elasticsarch. Így a logstash-nek nem kell külön az index-ekkel bajlódni, minden nap az adott dátummal az alábbi szerkezet? dokumentumokat szúrja be ha log érkezik hozzá a logsput-ol:

Ebb?l az egyszer? logsorból:

```
hello logspout
```

Az alábbi Elasticsearch REST hívást hajtotta végre a dokumentum beszúrására az aznapra létrehozott index-be:

```
PUT 'elasticsearch:9200/logstash_<dátum>/_doc/
{
               "host" => "10.0.0.2",
         "@timestamp" => 2018-09-19T19:15:00.000Z,
                "pid" => "6648",
          "logsource" => "622a2aa183b0",
            "program" => "ubunto",
           "severity" => 6,
            "message" => "hello logspout\n",
           "@version" => "1",
           "priority" => 14,
           "facility" => 1,
          "timestamp" => "2018-09-19T19:15:00Z",
     "severity_label" => "Informational",
      "timestamp8601" => "2018-09-19T19:15:00Z",
     "facility_label" => "user-level"
}
```

Ezt írták magukról:
https://www.elastic.co/products/logstash
LogStash allows us to centralize data processing. It can be easily extended to custom data formats and offers a lot of plugins that can fulfill almost any need. Finally

LogSpout is a log router for Docker containers that runs inside Docker. It attaches to all containers on a host, then routes their logs wherever we want. It also has an extensible module system. It's a mostly stateless log appliance. It's not meant for managing log files or looking at history. It is just a tool to get your logs out to live somewhere else, where they belong.

Logspout

Ez az egyetlen termék az ELK stack-ben ami nem az Elstatic-tól származik. A Logspout nem csinál mást, mint hogy rácsatlakozik a node-on futó docker démonra, és a docker-hez érkez? logokat átirányítja a megadott helyre, esetünkben ez a logstash lesz. Ez az egyetlen komponens amit minden egyes node-ra rá kell telepíteni, node-onként pontosan egyet, tehát majd globális módban kell telepíteni.

Logspout is a log router for Docker containers that runs inside Docker. It attaches to all containers on a host, then routes their logs wherever you want. It also has an extensible module system.

It's a mostly stateless log appliance. It's not meant for managing log files or looking at history. It is just a means to get your logs out to live somewhere else, where they belong.

For now it only captures stdout and stderr, but a module to collect container syslog is planned.

Kibana

Leegyszer?sítve, a Kibana kimondottan az Elasticsearch adatbázis webes nézeget?, lekérdez?, manipuláló konzolja.

Ezt írják magukról:
Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. You use Kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.

Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time.

Setting up Kibana is a snap. You can install Kibana and start exploring your Elasticsearch indices in minutes???no code, no additional infrastructure required.

Elasticsearch bemutatása

Az Elasticsearch adatbáziskezel? Elasticsearch node-okból épül fel. Mikor elindítunk egy node-ot, akkor az csatlakozni fog a konfigurációjában megadott cluster-hez. Ha a cluster még nem létezik, akkor létre fog hozni a node egy egy-node-os cluster-t. Egy környezetben (pl AWS) több Elasticsearch cluster-t is létrehozhatunk, lényeg, hogy a nevük különbözzön. Az alapértelmezett cluster név: elasticsearch

Akkor is Elasticsearch cluster-nek hívják, ha csak 1 node-ból áll a cluster. Tehát egy Elasticsearch adatbáziskezel?n mindig egy Elasticsearch cluster-t értünk.

https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-network.html

- TCP Transport (9300): Used for communication between nodes in the cluster, by the Java Transport client and by the Tribe node. See the Transport module for more information.
- HTTP (9200): Exposes the JSON-over-HTTP interface used by all clients other than the Java clients. See the HTTP module for more information.

Index

https://www.elastic.co/blog/what-is-an-elasticsearch-index
Az ElasticSearch-ben az Index megfelel egy adatbázisnak egy relációs adatbázis kezel?ben, RDBM-nek (Relational Database Management System)

- MySQL => Databases => Tables => Columns/Rows
- Elasticsearch => Indices => Types => Documents with Properties

Egy Elasticsearch cluster-ben tehát tetsz?leges számú **Index**-et hozhatunk létre (adatbázist), amiben tetsz?leges számú **Type** lehet (tábla). A type-okon belül **Document**-ek vannak (ezek a sorok), és a Document-nek vannak tulajdonságai, **Property** (ezek az oszlopok)

- Index -> Adatbázis
- Type -> Tábla
- Document -> Row
- Properties -> Column

Az én értésem szerint a típus bármilyen szó lehet, ez csak particionálja az adott indexbe behelyezett dokumentumokat. Pl: **mycomany/accounting** esetében a **mycompany** az index és az **account**-ing egy típus, amibe olyan dokumentumokat fogok rakni, amik az accounting témakörbe tartoznak. Keresésnél szintén meg lehet adni, hogy melyik típuson belül keresek.

A lekérdezésekre szintén RESTful webservice-t biztosít az Elasticsearch. A lekérdezésnek az alábbi az interfész definíciója:

```
http://localhost:9200/[index]/[type]/[operation]
```

Nézzünk egy olyan példát, ami közel áll az RDBM szemlélethez. Van egy autógyárunk, amiben vannak autók:

- Index name: SubaruFactory
- Types:
    - Cars
    - Workers
- Document in Car type:
    - Imprezza DOC

Ez teljesen egybevág egy RDBM-es megközelítéssel. Az Imprezza dokumentumot az alábbi lekérdezéssel kaphatjuk meg:

```
$ curl -XGET localhost:9200/SubaruFactory/Cars/SubaruImprezza
```

Fontos látni, hogy az Index-ekkel sokkal rugalmasabb struktúrát lehet kialakítani mint az RDBM-ben. Egy Elasticsearch cluster-ben több ezer Index-et is létrehozhatunk tetsz?leges logika mentén. Tipikus felhasználási területe az Elasticsearch-nek a logelemzés. Tipikus index-elési stratégia, hogy minden egyes napra létrehozunk egy külön Index-et a logokoknak (ez elképzelhetetlen lenne RDMS-ben)

- logs-2013-02-22
- logs-2013-02-21
- logs-2013-02-20

Aztán ebben létrehozhatunk Type-okat a logszinteknek megfelel?en, pl Debug, Info, Error. Egy példa lekérdezés:

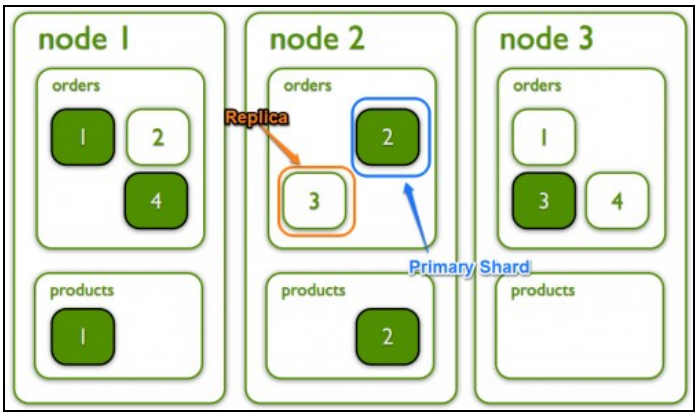
```
$ curl -XGET localhost:9200/logs-2013-02-22,logs-2013-02-21/Errors/_search?query="q:Error Message"
```

Shards & Replicasedit & Routing

Alapok

Az Elasticseach-ben minden index (adatbázis) fel van osztva kisebb tárolási egységekre, shard-okra (szilánkokra). Ez els?sorban arra szolgál, hogy ne ütközzünk bele egy node hardware korlátaiba lemezterület vagy cpu sebesség tekintetében, ezért az indexünket particionáljuk több node között. (Persze ha csak egy node-unk van, ez nem túl érdekes). Ezen felül minden shard-nak van 0 vagy több replikája, ami tökéletes másolata az adott shard-nak (szilánknak). Az írás mindig a primary shard-ra történik, de a keresés már párhuzamosítható a primary shard és a replikái között, ezzel nagyon felgyorsítva a keresést. Ezen felül a replikák failover kezelésre is szolgálnak, ha a primary shard-et tároló node meghal, akkor a replika lép a helyére. Fontos kikötés, hogy a replica shard-ok nem hozhatók létre ugyan azon a "fizikai" node-on ahol a primary van.

A shard-ok és replikák számát az index létrehozásakor kell megadni. Ha err?l külön nem rendelkezünk, akkor alapértelmezetten 5 shard-et fog létrehozni minden indexhez és 1 replikált minden egyes shard-hez tehát összesen 10 shard-et, amib?l 5 primary. Ha csak 1 darab node-unk van, akkor a replica shard-eket nem fogja tudni létrehozni az Elasticsarch, így az index állapota mindig "yellow" lesz. Ha már legalább két node-unk van, akkor akkor az 5 primary shard-et az egyik node-ra fogja rakni, míg az 5 replikát a másikra.



A fenti ábrán egy 3 node-os cluster-t láthatunk, amiben két index van összesen. Az **order** index úgy lett létrehozva, hogy 4 primary (tele zöld) és 4 replica (zöld keret) shard-b?l álljon. Tehát 4 felé van particionálva az order index, alapértelmezetten az Elsaticsearch dönti el, hogy egy új dokumentum melyik shard-re kerüljön, ez a user számára átlátszó. Az order index-ben ezen felül minden pirmary shard-nek egy replikája van mindig másik node-on mint ahol a primary van.

Ezen felül a product nev? index úgy lett létrehozva, hogy csak kett? darab primary shard -et használjon replikák nélkül.

Shard-ek definiálása

Ha nem az alapértelmezett shard számot akarjuk használni (5 primary és 1-1 replica mindegyikhez) akkor a shard és replica számot az index létrehozásánál kell megadni. Az alábbi példában létre fogunk hozni egy **mycompany** nev? indexet 3 primary és 2-2 replica shard-al minden primary-hez így összesen 3 + 3*2 = 9 shard-unk lesz. .

```
curl -XPUT "192.168.123.71:9200/mycompany" -H 'Content-Type: application/json' -d'
{
    "settings" : {
        "index" : {
            "number_of_shards" : 3,
            "number_of_replicas" : 2
        }
    }
}
'
```

Ha most lekérdezzük az indexek listáját, láthatjuk, hogy 3 shard-el létrejött a mycompnay inev? indexünk.

```
$ curl -XGET "192.168.123.71:9200/_cat/indices?v"
health status index               uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   mycompany           XhbjouKWSAeSgcBEkPzlkA   3   2          0            0       690b           690b
```

Note
Nagyon fontos, hogy el?re kitaláljuk, fogalma sincs mi alapján, hogy hány shard-re és replikára lesz szükségünk az optimális teljesítményre való tekintettel. Kés?bb ezen nem ajánlatos változtatni, tehát ha nem megfelel? az alapértelmezett 5 master és 5 replica, akkor az index létrehozásánál meg kell adni ezeket

Routing vs custom Routing

https://www.elastic.co/blog/customizing-your-document-routing
A routing az Elasticsearch világában azt jelenti, hogy melyik (primary) shard-re fog kerülni a dokumentum amit beszúrunk. Ha külön nem adjuk meg, akkor ez kvázi véletlen szer? lesz. A shard ID meghatározásához, vagyis hogy melyik shard-re kerüljön a beszúrandó dokumentum, a következ? formulát használja az Elasticsearch:

```
shard_num = hash(_routing) % num_primary_shards
```

A modulo osztás biztosítja, hogy mindig egy valid shard számot kapjunk.

Ha ezt külön nem adjuk meg, akkor a dokumentum ID -ja lesz a **_routing** változó értéke. A dokumentum ID mez?jét vagy mi adjuk meg a dokumentum beszúrásakor a **_ID** paraméterrel, vagy az Elasticsearch-re bízzuk, hogy vegye el? a következ? szabad ID-t. A document id -> routing id behelyettesítés biztosítja, hogy egyenletesen legyenek elosztva a dokumentumok az összes shard között, a keresés és a beszúrás is teljesen átlátszó a felhasználók számára shard szempontból.

Mikor rákeresünk egy dokumentumra, alapesetben az Elasticsearch-nek fogalma sincs, hogy melyik shard-ban van a dokumentum, így kiküld egy broadcast üzenetet az összes node-nak, akik ezt továbbítják párhuzamosan az összes shard-re, majd minden shard visszaküldi a keresés eredményét a gateway node-nak. Ez gyors, viszont nagyon er?forrás intenzív. A dokumentum mindig csak 1 darab primary shard-en van. Ha 20 primary shard-et használ az index, akkor ez 19 darab "felesleges" keresés.

Lehet?ség van rá, hogy egyrészt beszúráskor megmondjuk az Elasticsearch-nek hogy melyik shard-re tegye a dokumentumot, illetve keresésnél megmondhatjuk, hogy melyiken keresse, ezzel eliminálhatjuk a felesleges kereséseket. Ezt hívják custom routing-nak.

Ha elhatároztuk hogy egy egész indexre, vagy típusra custom routing-ot akarunk használni, akkor best-practice ha beállítjuk hogy csak olyan dokumentumokat fogadjon el az Elasticsearch az adott indexbe/típusba, ahol meg van adva a custom routing.

A az el?bb létrehozott **mycompany** index-be hozzunk létre egy **order** típust, amin beállítjuk, hogy csak custom routing-al megadott dokumentumokat lehet beleszúrni.

```
curl -XPUT '192.168.123.71:9200/mycompany/order/_mapping' -H 'Content-Type: application/json' -d'
{
   "order":{
      "_routing":{
         "required":true
      }
   }
}
'
```

Ez nem kötelez?, de így ki lehet er?szakolni a custom routing megadását, nehogy véletlen lemaradjon.

Most szúrjunk be egy dokumentumot a order típus alá egyedi routing-id-val. Ezt a **routing** paraméterrel kell megadni:

```
$ curl -XPOST '192.168.123.71:9200/mycompany/order?routing=user123' -H 'Content-Type: application/json' -d'
{
   "productName":"sample",
   "customerID":"user123"
}
'
```

Note
Fizikailag nem tudjuk, hogy melyik shard-re fog kerülni a dokumentum. Csak azt mondtuk meg az Elasticsearch-nek, hogy mikor a hash-t képzi, akkor a routing-id helyére a képlet-be azt írja be hogy **user123**. Mikor a keresésénél ugyan ezt a **routing-id**-t adjuk majd meg, akkor a fenti képlet ugyan azt a shrad ID-t fogja visszaadni, tehát ugyan azzal a routing id-val mindig ugyan azt a shard id-t állítjuk el?!

Keressük meg a feint dokumentumot úgy hogy megmondjuk az Elasticsearch-nek hogy pontosan melyik shrad-ban keresse. Ahogy a beszúrásánál is, itt is megadjuk a **routing=user123** értéket, ami ugyan arra a shard ID-ra fog leképez?dni, ahova korábban beszúrta.

```
$ curl -XGET '192.168.123.71:9200/mycompany/order/_search?routing=user123&pretty' -H 'Content-Type: application/json' -d'
{
   "query":{
      "match_all":{}
   }
}
'
```

A match_all-al az összes dokumentumot lekérjük az **order** típusból.

És a válasz:

```
...
      {
        "_index" : "mycompany",
        "_type" : "order",
        "_id" : "kwczBmYB5rEaU4uAZ6UJ",
        "_score" : 1.0,
        "_routing" : "user123",
        "_source" : {
          "productName" : "sample",
          "customerID" : "user123"
        }
```

Láthatjuk benne a routing id-t a _routing mez?ben. Persze a routing paraméter nélkül is kereshettünk volna, de akkor az összes shard-re el lett volna küldve a kérés.

REST API

A REST API-n keresztül a következ? 4 témakörben hajthatunk végre m?veletetek:

- Lekérdezhetjük a cluster és a node-ok állapotát
- Elvégezhetjük a cluster, node-ok és index-ek adminisztrációját (
- CRUD (Create, Read, Update, and Delete)
- Komplex kereséseket állíthatunk össze (paging, rendezés, filterezés, aggregáció)

ELK stack telepítése

```
docker network create --driver overlay elk
```

ElasticSearch

Az Elastichsearch docker image a https://www.docker.elastic.co/ oldaláról tölthet? le

Itt van a telepítési leírás:

https://www.elastic.co/guide/en/elasticsearch/reference/6.4/docker.html

Volume plugin

Ahogy azt már sokszor említettem, a konténerek legfels?, írható rétegét, nem szabad írás intenzíven használni, mivel az ottani overlay2 fájlrendszer nagyon lassan tudja csak kezelni a változásokat, legrosszabb esetben minden image-et végig kell nézzen, ami a konténert alkotja, hogy megtaláljon egy fájlt. Minden írás intenzív m?veletet volume-okon kell elvégezni, azok arra lettek kitalálva.

Az ElsaticSearch konténer két mappáját fogjuk a Netshare volume plugin-al felcsatolni az NFS megosztásra. Az egyik az adatbázis mappa, a másik a config mappa, hogy

Els?ként az ElasticSearch image-et fel fogjuk standalone docker konténerként telepíteni hogy kimásoljuk bel?le a konfigurációs mappájának a tartalmát, amit az NFS elasticsearch/config mappájába fogunk másolni, hogy ezt majd felcsatoljuk a swarm service alá.

```
# docker run -d --name elasticsearch \
-p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" \
docker.elastic.co/elasticsearch/elasticsearch:6.4.0

# docker cp -L elasticsearch:/usr/share/elasticsearch/config/ /home/adam/Projects/DockerCourse/persistentstore/elasticsearch/
# chmod 777 /home/adam/Projects/DockerCourse/persistentstore/elasticsearch/config
# mkdir /home/adam/Projects/DockerCourse/persistentstore/elasticsearch/data

# docker rm -f elasticsearch
```

Note
Az elasticsearch image elég nagy, 1 Giga körül van, így fontos, hogy legyen elég lemezterület az összes node-on. A base image CentOS 7.5

Telepítés

Az Elasticsarch image-nek két fontos portja van. A REST interfészt a **9200**-as porton biztosítja, míg az Elasticsarch node-ok a **9300**-as porton kommunikálnak egymással. A **9300**-as port publikálása az ingress hálózaton semmiképpen nem szükséges, mert ha több node-unk is lenne, azok mind az **elk** nev? overlay hálózatra csatlakoznának, ahol közvetlen elérnék egymást, ráadásul most csak egy node-os az Elasticsearch cluster. A swarm node-okon a logokat a **logspout** fogja összegy?jteni, amit el fognak küldeni az **elk** nev? overlay hálózaton a **logstash**-nek. A **logstash** fogja betölteni a logokat szintén az **elk** overlay hálózaton keresztül az Elasticsarch-be. Tehát itt sem szükséges a 9200-as port publikálása az ingress hálózaton. Azonban mi szeretnénk a swarm cluster-enk kívül is meghívni az Elasticsarch REST API-ját tesztelés céljából, ezért publikálni fogjuk a 9200-as portot.

```
docker service create \
--detach=false \
--name elasticsearch \
--network elk \
-p 9200:9200 \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/config/\
,dst=/usr/share/elasticsearch/config,volume-driver=nfs" \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/data/\
,dst=/usr/share/elasticsearch/data,volume-driver=nfs" \
-e "discovery.type=single-node" \
--reserve-memory 500m \
docker.elastic.co/elasticsearch/elasticsearch:6.4.0

# docker service ps elasticsearch
ID                  NAME                IMAGE                                                 NODE
phgs0as5w612        elasticsearch.1     docker.elastic.co/elasticsearch/elasticsearch:6.4.0   mg2
```

Logstash

A LogStash image-et is az elastic oldaláról lehet letölteni:
https://www.elastic.co/guide/en/logstash/current/docker-config.html

Logstash konfigurációs fájl

https://www.elastic.co/guide/en/logstash/6.4/configuration.html

A logstash alapértelmezett konfigurációja itt van: **/usr/share/logstash/pipeline/logstash.conf**. A pipeline mappában ezen kívül szerencsére nincs semmi, tehát gond nélkül mountolhatunk rá egy NFS meghajtót, ahova berakjuk a saját konfigurációs fájlunkat:
logstash.conf

```
input {
  syslog { port => 51415 }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
  }
  # Remove in production
  stdout {
    codec => rubydebug
  }
}
```

A logstash-t is a **elk** nev? overlay hálózatra fogjuk kapcsolni. Az overlay hálózatokon egyrészt a konténerek közvetlen elérik egymást, így nincs szükség az ingress hálózatra publikált portokra, másrészt a swarm névfeloldást végez. A szolgáltatás nevére indított DNS lekérdezés visszaadja az összes a szolgáltatáshoz tartozó konténer IP címét a közös overlay hálózaton. Fontos, hogy a lekérdezést olyan konténerb?l indítsuk, ami ugyan arra az overlay hálózatra csatlakozik mint a keresett szolgáltatás.

Az ElasticSearch a 9200-es porton hallgatózik, amit nem publikáltunk az ingress hálózatra. Viszont a logstash a közös **elk** overlay hálózaton fel tudja oldani az elasticsearch domain nevet, ami megegyezik a szolgáltatás nevével.

A logstash az **51415**-ös porton fogja várni a beérkez? logokat, amit majd aztán továbbít a megfelel? alakra hozva a elasticsearch:9200-es címre az ElasticSearch-nek.

Telepítés

```
docker service create --name logstash \
--detach=false \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/logstash/config/\
,dst=/usr/share/logstash/pipeline,volume-driver=nfs" \
--network elk \
--reserve-memory 100m \
-e "LOGSPOUT=ignore" \
docker.elastic.co/logstash/logstash:6.4.0
```

A **LOGSPOUT=ignore** környezeti változóval azt mondjuk meg a logspout-nak, hogy err?l a konténerr?l ne gy?jtse össze a logokat.

Viszonylag lassan indul el a telepítés után, kb 30 másodperc. Ha végre elindul, akkor az alábbi log sor jelenik meg. Láthatjuk hogy sikeresen kapcsolódott az Elasicsearch-öz a konfigurációban megadott URL-en (amúgy ez az alapértelmezett, magától is itt keresné).

```
# docker service  logs -f logstash
...
[2018-09-29T10:42:06,449][INFO ] Elasticsearch pool URLs updated {:changes=>{:removed=>[], :added=>[http://elasticsearch:9200/]}}
[2018-09-29T10:42:06,506][INFO ] Running health check to see if an Elasticsearch connection is working {:healthcheck_url=>http://elasticsearc

[2018-09-12T20:01:26,403][INFO ][logstash.inputs.metrics  ] Monitoring License OK
[2018-09-12T20:01:27,999][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600}
```

Testing logstash

A **logger** nev? programmal, mely része a legtöbb Linux disztribúciónak logot írhatunk egy távoli server syslog-jába a lokális socket helyett. A **logger-test** konténert is a **elk** nev? overlay hálózatra fogjuk kötni, így közvetlen el fogja tudni érni a logstash-t a 51415-ös porton. Az overlay hálózaton használhatjuk a szolgáltatás nevét mint domain nevet, a logstash domain nevet a swarm fel fogja oldani a logstash konténer IP címére.

```
docker service create \
--name logger-test \
--network elk \
--restart-condition none \
debian \
logger -n logstash -P 51415 hello world
```

```
# docker service logs logstash
....
{
          "severity" => 0,
          "@version" => "1",
              "tags" => [
        [0] "_grokparsefailure_sysloginput"
    ],
    "severity_label" => "Emergency",
        "@timestamp" => 2018-09-12T20:24:29.085Z,
          "priority" => 0,
    "facility_label" => "kernel",
           "message" => "<13>1 2018-09-12T20:24:29.013823+00:00 a4f8651665ee root - -
                                [timeQuality tzKnown=\"1\" isSynced=\"0\"] hello world",
              "host" => "10.0.0.2",
          "facility" => 0
}
```

```
# eval $(docker-machine env mg0)
# docker service rm logger-test
```

Logspout

https://hub.docker.com/r/gliderlabs/logspout/

ogspout will gather logs from other containers that are started without the -t option and are configured with a logging driver that works with docker logs (journald and json-file).

For now it only captures stdout and stderr, but a module to collect container syslog is planned.

Telepítés

A logspout az egyetlen komponens amit minden egyes node-ra ki kell rakni, hogy el tudja küldeni a logokat a helyi docker démontól a logstash-nek. Ezért bind mount-ot fogunk létrehozni a node-on futó docker soket-re, hogy hozzáférjen a logokhoz. Command line paraméterként adjuk át neki hogy hova kell küldeni az összegy?jtött logokat. Syslogként kell ?ket elküldeni az 51415-ös portra, ahol a logstash fogadja majd azokat és beküldi az Elasticsearch-be.

```
docker service create --name logspout \
--detach=false \
--network elk \
--mode global \
--mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock" \
-e SYSLOG_FORMAT=rfc3164 \
gliderlabs/logspout:v3.2.5 syslog://logstash:51415
```

Fontos, hogy a logspout is az **elk** nev? overlay hálózatra csatlakozik, ezért közvetlen tud kommunikálni a logstash-el. A swarm DNS az overlay hálózaton fel fogja tudni oldani a logstash szolgáltatás nevet az **elk** hálózaton kapott IP címére.

```
# docker service ls
ID                  NAME          MODE       REPLICAS     IMAGE
e0smhp3tn8ox        logspout      global     4/4          gliderlabs/logspout:v3.2.5
```

A log végén meg kell jelenjen az alábbi sor, ami jelzi, hogy a logstash-ez kapcsolódik az 51415-ös porton.

```
# docker service logs logspout
logspout.0.lh7yq14rapf7@mg2    | #   ADAPTER    ADDRESS         CONTAINERS      SOURCES OPTIONS
logspout.0.lh7yq14rapf7@mg2    | #   syslog     logstash:51415                          map[]
```

Note
Ha nincs elég memória szabadon a node-on, akkor a logspout nem fog tudni elindulni.

Tesztelés

A **logspout** minden egyes node-on ott fut, és bármelyik node-on is keletkezzen log az stdout-on, azt el fogja küldeni a logstash-nek az **51415**-ös portra. Mivel mind a ketten az **elk** nev? overlay hálózatra kapcsolódnak, a logspout közvetlen eléri a **logstash**-t az bármelyik node-on is legyen. A közös overlay hálózatokon a service nevet domain névként használhatjuk, a swarm DNS szervere visszaadja az összes konténer IP címét, ami a service-hez tartozik. Mivel a logstash-böl csak egy példány fut, így az összes logspout konténer közvetlen el tudja neki küldeni az összegy?jtött logokat.

Úgy fogjuk tesztelni, hogy els?ként elkezdjük figyelni interaktív módban (-f) a **logstash** log-ját. Majd egy tetsz?leges node-on (mindegy melyiken, a **logspout** minden node-on fut) elindítunk egy ubuntu image-b?l álló konténert, ami egy sort fog írni az **stdout**-ra. Ennek azonnal meg kell jelennie a logstash logjában. Az ubuntu-nak nem is kell swarm service-ként futnia, a logspout mivel a lokális docker démonra csatlakozik, a standalone docker konténerek logját is be tudja gy?jteni.

Indítsuk el a logstash log-néz?t:

```
# docker service logs -f logstash
```

Indítsuk el az ubuntu-t standalone docker konténerként, ami az stdout-ra fog írni a konténeren belül. Ezt az logspout-nak észre kell venni, és el kell küldeni a logstash-nek, aki már be tudja tölteni a megfelel? alakban az ElasticSarch-be. Az ubuntu konténer ahogy kiírta az üzenetet az stdout-ra le fog állni, így kapásból törölhetjük is. Fontos, hogy **-d** kapcsolóval futtassuk az ubuntu konténert, interaktív módban a logspout nem gy?jti be a logokat.

```
# docker run -d --name ubuntu ubuntu echo "hello logspout" > /dev/stdout
```

Note
Amikor a --rm kapcsolóval futtattam, akkor mintha nem lett volna ideje a logspout-nak elküldeni a logokat, a --rm kapcsoló használata mellett nem jelent meg semmi a logstash-ben

Szinte azonnal meg kell jelenjen a "hello logspout" üzenet a logstash logjában, amit interaktív módban figyelünk:

```
logstash.1.ggjdb8navgso@mg1    | {
logstash.1.ggjdb8navgso@mg1    |           "@version" => "1",
logstash.1.ggjdb8navgso@mg1    |         "timestamp" => "2018-09-14T20:31:00Z",
logstash.1.ggjdb8navgso@mg1    |           "program" => "ubunto",
logstash.1.ggjdb8navgso@mg1    |          "severity" => 6,
logstash.1.ggjdb8navgso@mg1    |    "facility_label" => "user-level",
logstash.1.ggjdb8navgso@mg1    |               "pid" => "3291",
logstash.1.ggjdb8navgso@mg1    |          "priority" => 14,
logstash.1.ggjdb8navgso@mg1    |        "@timestamp" => 2018-09-14T20:31:00.000Z,
logstash.1.ggjdb8navgso@mg1    |          "facility" => 1,
logstash.1.ggjdb8navgso@mg1    |     "timestamp8601" => "2018-09-14T20:31:00Z",
logstash.1.ggjdb8navgso@mg1    |    "severity_label" => "Informational",
logstash.1.ggjdb8navgso@mg1    |           "message" => "hello logspout\n",
logstash.1.ggjdb8navgso@mg1    |              "host" => "10.0.0.5",
logstash.1.ggjdb8navgso@mg1    |         "logsource" => "836892e1d7fa"
logstash.1.ggjdb8navgso@mg1    | }
```

Docker logger driver vs logspout

Kibana

https://www.elastic.co/products/kibana
https://www.elastic.co/guide/en/kibana/current/docker.html

Volume plugin

Három fontos mappáját kell kivezessük az NFS meghajtóra a Kibana konténernek:

- /usr/share/kibana/config: konfig mappa helye
- /usr/share/kibana/data : ez a munkaterület
- /usr/share/kibana/plugins : ide írnak a plugin-ek.

Telepítés

A Kibana-t is az **elk** overlay hálózatra fogjuk csatlakoztatni, így közvetlenül el fogja érni az elasticsearch szolgáltatást a 9200 porton. Az elasticsearch domain nevet fel fogja oldani a swarm DNS szerver a konténer IP címére az **elk** hálózaton. Az Elasticsearch szerver elérhet?ségét most nem config fájlal adjuk meg, hanem a **ELASTICSEARCH_URL** környezeti változóval. A Kibana web-es felülete a **5601**-es porton érhet? el, azt publikáljuk az ingress overlay hálózatra, így bármelyik node publikus IP címével el fogjuk érni a Kibana-t.

```
docker service create --name kibana \
--detach=false \
--network elk \
-e ELASTICSEARCH_URL=http://elasticsearch:9200 \
-p 5601:5601 \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/kibana/config/\
,dst=/usr/share/kibana/config,volume-driver=nfs" \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/kibana/data/\
,dst=/usr/share/kibana/data,volume-driver=nfs" \
--mount "type=volume,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/kibana/plugins/\
,dst=/usr/share/kibana/plugins,volume-driver=nfs" \
docker.elastic.co/kibana/kibana:6.4.0
```

Telepítés után itt érhet? el a Web-es konzol:
http://192.168.123.71:5601/app/kibana

Elasticsearch REST API

Alap m?veletek

Cluster információ:

```
# curl -XGET 192.168.123.71:9200/_cat/health?v
epoch      timestamp cluster        status node.total node.data shards pri relo init unassign pending_tasks
1537214568 20:02:48  docker-cluster yellow          1         1     26  26    0    0       25             0
```

Láthatjuk, hogy a cluster neve docker-cluster, amit 1 darab node alkot. Az Elasticsearch állapota sárga:

- **Green** - Az Elasticsarch cluster-el minden rendben van.
- **Yellow** - A cluster-ben minden adat elérhet?, de bizonyos replica-ák még nincsenek létrehozva.
- **Red** - Valami hiba van a cluster-ben, bizonyos adatok biztos hogy nem elérhet?ek.

Node-ok listája:

```
# curl -XGET http://192.168.123.71:9200/_cat/nodes?v
ip        heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
10.0.0.21           10          94   4    0.03    0.07     0.08 mdi       *      SB7y2Lj
```

Láthatjuk, hogy az egy szem node-unknak az **etk** overlay hálózatbéli IP címét adta vissza a lekérdezés.

Összes index listázása:

```
# curl -XGET http://192.168.123.71:9200/_cat/indices?v
health status index               uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   logstash-2018.09.13 1Hg4zKXYRGC2Iw1TBTUrLg   5   1         14            0     53.5kb         53.5kb
yellow open   logstash-2018.09.17 rQy4MF7fTaOfNRivQuDStw   5   1         16            0     66.7kb         66.7kb
yellow open   logstash-2018.09.12 Oi-8kAdfRGCwrAtb2KXVTA   5   1      45507            0      3.9mb          3.9mb
yellow open   logstash-2018.09.16 b2I5CGDCS4ukEAs2hkSEcg   5   1        123            0    102.3kb        102.3kb
yellow open   logstash-2018.09.14 H51us_IvTzygKzTDwBXS6Q   5   1       1138            0    516.6kb        516.6kb
green  open   .kibana             wt_7PYjeRO6BAwivy2xs5g   1   0          2            3      8.6kb          8.6kb
```

Láthatjuk, hogy naponként létrejött egy új index a logstash által begy?jtött logokohoz. Ezen felül a Kibana maga is töltött be adatokat.

- **pir**: Azt mutatja meg, hogy hány darab **Primary Shard** (szilánk) jött létre az indexhez. Az alapértelmezett 5 darab.
- **rep**: Replica szám. Azt mutatja meg, hogy hány replikát akar az Elasticsearch létrehozni az indexhez. Az alapértelmezett 1. Mivel csak 1 darab node-unk fut jelenleg, nem tud replikát létrehozni, ezrét lesz az index egészsége **yellow**, kivéve a .kibana index, ami **green**, mivel az az index replika nélkül lett létrehozva.

Index beszúrása:

```
# curl -XPUT http://192.168.123.71:9200/customer?pretty
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "customer"
}
```

A pretty hatására írja ki ilyen szépen a végeredményt.

```
# curl -XGET http://192.168.123.71:9200/_cat/indices?v
health status index               uuid                   pri rep docs.count docs.deleted store.size pri.store.size
...
yellow open   customer            BgTjrFs1Qh6eZm5B3nPCzA   5   1          0            0      1.1kb          1.1kb
...
```

Dokumentum hozzáadása az index-hez: Az el?bb létrehozott customer index-hez adjunk hozzá egy dokumentumot. Fontos, hogy a **-H** header kapcsolóval megadjuk a tartalom típusát, ami JSON kell legyen. Ezen felül a -d kapcsolóval adhatjuk meg a PUT törzsét, ahol a JSON-t adjuk meg. A /_doc/ után kell megadni a dokumentum ID-t, ami az 1-es lesz. Ha nem adunk meg ID-t, akkor egy random ID lesz generálva a dokumentumhoz.

```
# curl -XPUT '192.168.123.71:9200/customer/_doc/1?pretty' -H 'Content-Type: application/json' -d '
{
  "name": "John Doe"
```

```
}'
```

És most kérjük le az 1-es ID-val rendelkez? dokumentumot a customer index-en belül:

```
# curl -XGET http://192.168.123.71:9200/customer/_doc/1?pretty
{
  "_index" : "customer",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "name" : "John Doe"
  }
}
```

- found: megmutatja, hogy megtalálta e
- _source: visszaadja a dokumentum tartalmát.

Dokumentum update-elése
A meglév? dokumentumot a _doc/<ID>/_update-el lehet frissíteni, új mez?ket hozzáadni.

```
curl -XPOST '192.168.123.71:9200/customer/_doc/1/_update?pretty' -H 'Content-Type: application/json' -d '
{
  "doc": { "name": "Jane Doe", "age": 20 }
}'
```

Join statement

Áttekintés

Lehet?ség van rá hogy egyfajta relációt vezessünk be egy indexben lév? dokumentumok között, egy szül?-gyerek kapcsolatot. A relációt típusát el?re definiálni kell az index létrehozása közben úgy hogy el?re megadjuk hogy milyen típusú dokumentumok állnak itt relációban egymással és meg kell adni egy mez?t is ami mind a két dokumentum típusban szerepelni fog mint a join mez?. A szül? esetében a join mez? csak hivatkozik a típusra, amit az index létrehozása közben definiáltuk, míg a gyerekél egyrészt hivatkozik a típusra, másrészt a szül? ID-jára is.

A joint kapcsolatot nem szabad relációs adatbázis kapcsolatként használni. Csak akkor szabad használni ha a gyerek elemek lényegesen többen vannak mint a szül? elem, különben nagyon rossz lesz a performancia. Fontos megkötések vannak a join-ak kapcsolatban amit?l egy kicsit úgy is t?nik, hogy nem igazán kiforrott ez még:

- Egy indexen belül csak egy darab join mapping definíció lehet
- A szülés és a gyerek muszáj hogy ugyan abban a shard-ban legyen, erre nekünk kell figyelni a létrehozás közben.
- Egy elemnek csak egy szül?je lehet, de bármennyi gyereke.

Használat

Az alábbi példában létrehozzuk a **mycompany** nev? index-et, miben definiálunk egy relációt, ahol azt mondjuk meg, hogy a **question** a szül?je az answer-nek. Ezek it nem szigorú értelembe vett dokumentum típusok, tehát nem kell hogy *mycompany/question* ill *mycomapany/answer* -be hozzuk ?ket létre. Ezek sokkal inkább egyfajta címkék, amiket majd rá kell aggatni a szül?re ill a gyerekre. Lényeg, hogy mikor majd létrehozzuk a szül? dokumentumot, akkor abban lennie kell majd egy **my_join_field** nev? mez?nek, aminek az értéke vagy question vagy és a gyerek dokumentumban pedig szintén lennie kell majd egy **my_join_field** nev? mez?nek aminek az értéke **answer**.

```
curl -XPUT "192.168.123.71:9200/mycompany" -H 'Content-Type: application/json' -d'
{
  "mappings": {
    "_doc": {
      "properties": {
        "my_join_field": {
          "type": "join",
          "relations": {
            "question": "answer"
          }
        }
      }
    }
  }
}
'
```

Adjunk hozzá egy szül? dokumentumot:

```
curl -X PUT "192.168.123.71:9200/mycompany/_doc/2?refresh" -H 'Content-Type: application/json' -d'
{
  "text": "This is another question",
  "my_join_field": "question"
}
'
```

Majd két gyerek objektumot. Mivel megkötés, hogy ugyan abba a shar-ba kell kerüljön mint a szül?, ezért ezt nekünk külön kézzel ki kell kényszeríteni. A szül? létrehozásakor nem adtunk meg routing id-t, így ahogy azt már korábban láthattuk, a shard meghatározásához a dokumentum ID-t használta fel az Elasticsearch, ami az **1** volt. Ha nem adnánk most itt meg routing ID-t, akkor a **3**-as és **4**-es ID-t használná fel az Elasticsearch (ami itt a két dokumentum ID), ami isten tudja melyik shard-ba esne. Ezért itt külön meg kell adni a **routing** paraméterrel az 1-es ID-t.

```
curl -XPUT "192.168.123.71:9200/mycompany/_doc/3?routing=1&refresh&pretty" -H 'Content-Type: application/json' -d'
{
  "text": "This is an answer",
  "my_join_field": {
    "name": "answer",
    "parent": "1"
```

```
  }
}
'

curl -XPUT "192.168.123.71:9200/mycompany/_doc/4?routing=1&refresh&pretty" -H 'Content-Type: application/json' -d'
{
  "text": "This is an other answer",
  "my_join_field": {
    "name": "answer",
    "parent": "1"
  }
}
'
```

A relációk megjelennek a dokumentumokon, ha lekérdezzük ?ket. A lekérdezéshez nagyon hasznos lesz a **children aggregation** (lásd lentebb)

Bulk m?veletek

Egyszerre több m?veletet is végrehajthatunk a _bulk paranccsal. Minden m?velet egy vagy két sorból áll. Az els? sorban vagy megadunk egy új indexet, vagy egy régit törlünk vagy update-elünk. Az alábbi példában két új dokumentumot adunk a customer index-be, majd a 2-est update-eljük, a 3-ast töröljük.

```
$ curl -XPOST '192.168.123.71:9200/customer/_doc/_bulk?pretty' -H 'Content-Type: application/json' -d '
{"index":{"_id":"2"}}
{"name": "John Doe2" }
{"index":{"_id":"3"}}
{"name": "Jane Doe3" }
'

$ curl -XPOST '192.168.123.71:9200/customer/_doc/_bulk?pretty' -H 'Content-Type: application/json' -d '
{"update":{"_id":"2"}}
{"doc": {"name": "John Doe", "age":30 }}
{"delete":{"_id":"3"}}
'
```

Bulk betöltés fájlból
Az alábbi példában egy fiktív banki model 1000 dokumentumát fogjuk betölteni. A fájl itten tölthet? le:
https://github.com/elastic/elasticsearch/blob/master/docs/src/test/resources/accounts.json?raw=true

```
$ curl -H "Content-Type: application/json" -XPOST "192.168.123.71:9200/bank/_doc/_bulk?pretty&refresh" --data-binary "@accounts.json"
```

Most kérdezzük le az indexeket:

```
# curl -XGET "192.168.123.71:9200/_cat/indices?v"
health status index               uuid                   pri rep docs.count docs.deleted store.size pri.store.size
..
yellow open   bank                6eVLH1Z8QsS-cP5k4EwNhA   5   1       1000            0    474.6kb        474.6kb
```

Láthatjuk, hogy a **bank** indexet az Elasticsearch létrehozta, mert még nem létezett, és hogy 1000 dokumentum van benne.

Query DSL (Domain Specific Language)

https://www.elastic.co/guide/en/elasticsearch/reference/current/_introducing_the_query_language.html

A lekérdezést az **<index név>/_search** végponton kell meghívni. Vagy a request URL-ben állítjuk össze a lekérdezést, vagy a request törzsében.

Az alábbi lekérdezés visszaadja az összes bank indexben lév? dokumentumot ascending sorrendben.

```
$ curl -XGET "192.168.123.71:9200/bank/_search?q=*&sort=account_number:asc&pretty"
```

- q=query
- sort=rendezési paraméterek

Ugyan ez a lekérdezés a GET törzsében megadva:

```
$ curl -H "Content-Type: application/json" -XGET '192.168.123.71:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}'
```

A keresés eredmény mindig egy összefoglalóval kezd?dik:

```
{
  "took" : 571,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : null,
    "hits" : [
```

```
      ... találatok....
    ]
```

- took: mennyi ideig tartott
- total: mennyi találat van összesen

Ha külön nem adjuk meg, mindig az els? 10 találatot fogja visszaadni. A **size** és a **from**-al lehet ezt szabályozni:

```
$ curl -X GET "192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "from": 10,
  "size": 2
}
'
```

Sz?kítés mez?kre
Beállíthatjuk, hogy a document source-bol milyen mez?ket adjon csak vissza a **_source** paraméterrel. Az alábbi példa csak az account_number-t és a balance-ot fogja visszaadni.

```
curl -X GET "192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}
'
```

Keresés mez? értékre
A **match** paraméterrel lehet megadni mez? szint? keresési feltételeket. Az alábbi példában csak azt a dokumentumot keressük, ahol az account_number = 20. Egy ilyen lesz. Az eredményt sz?kítjük két mez?re.

```
curl -X GET "192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "account_number": 20 } },
  "_source": ["account_number", "balance"]
}
'
```

Logikai ÉS
Az alábbi példában csak azokat keressük, ahol mind a két match feltétel teljesül.

```
curl -X GET "192.168.123.71:9200/bank/_search" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}
'
```

Logikai VAGY
Az alábbi példában azokat keressük amire vagy az egyik, vagy a másik match teljesül:

```
..
"query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
```

Összetett query-k:
A bool után több logikai kifejezést is felírhatunk vessz?vel elválasztva <must|should|must_not> [..], <must|should|must_not> [..], ...

```
$ curl -X GET "192.168.123.71:9200/bank/_search" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
}
'
```

Aggregáció

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html
Ez egy elég nagy témakör az Elasticserach-ön belül, ezért külön f? fejezetet édemel.

- **Metric**: Aggregations that keep track and compute metrics over a set of documents.
- **Bucketing**: A family of aggregations that build buckets, where each bucket is associated with a key and a document criterion. When the aggregation is executed, all the buckets criteria are evaluated on every document in the context and when a criterion matches, the document is considered to "fall in" the relevant bucket. By the end of the aggregation process, we?ll end up with a list of buckets - each one with a set of documents that "belong" to it.
- **Pipeline**: Aggregations that aggregate the output of other aggregations and their associated metrics
- **Matrix**: A family of aggregations that operate on multiple fields and produce a matrix result based on the values extracted from the requested document fields. Unlike metric and bucket aggregations, this aggregation family does not yet support scripting.

Metrika aggregáció

A metrika aggregációk olyan függvények, amik a dokumentumok bizonyos mez?inek az értékein valamilyen m?velet hajtanak végre, tipikusan ezek numerikus mez?k, a végeredmény egy vagy több szám. Pl a legegyszer?bb metrika aggregáció az átlag kiszámítása. Az aggregációt lehet keresési eredményekre alkalmazni, vagy más aggregációk kimenetére, és a kimenetet is fel lehet használni további aggregációkhoz, lekérdezésekhez.

- single-value numeric metrics aggregation
- multi-value numeric metrics aggregation

A metrikáknak aggregációnak az alábbi a szintaksisa:

```
POST 192.168.123.71:9200/<keres? kifejezés>
{
    "aggs" : {
        "<aggregáció neve>" : { "<aggregáció típusa>" : { <paraméterek> } }
    }
}
```

A végeredmény is egy dokumentum lesz, amiben lesz egy mez? név a metrika nevével, amit itt megadtunk a lekérdezésben. Azt hogy ez a lekérdezés egy aggregáció lesz, azt az **aggs** kulcsszó jelöli.

Ha egy lekérdezésben volt aggregáció, akkor a válasz legvégére az Elasticsearch odarakja az aggregations nev? listát. Minden egyes aggregációnak lesz egy külön listaeleme azzal a névvel, amit megadtunk az aggregáció definiálásakor.

```
"aggregations" : {
    "<aggregáció neve>" : {
      "value(s)" : <végeredmény, ami lehet egy vagy több elem?>
    }
  }
```

single-value numeric metrics aggregation

A single-value aggregációknál a végeredmény mindig 1 elem?, tehát a végeredmény mindig így néz ki, ahol az XXX egy darab szám.

```
"aggregations" : {
    "<aggregáció neve>" : {
      "value" : XXX
    }
  }
```

Példák egy érték? aggregációkra:

- Átlag (average)
- Súlyozott átlag (weighted average)
- Cardinality (különböz? elemek elemszáma, pl set={1,1,2,3,4,4} kardinalitása = 4

Átlag
Számoljuk ki, hogy a **bank** indexben tárolt ügyfeleknek mekkora az átlagéletkora. Ez egy **single-value numeric metrics aggregation**, amiben az **avg** függvényt fogjuk használni az **age** mez?re. A **size=0**-val azt érjük el, hogy a **_search**-el megtalált dokumentumokat ne adja vissza, csak az aggregáció végeredményét. Alapértelmezetten az els? 10 találatot adná vissza, ahogy azt már láttuk, és a végére tenni oda az aggregáció eredményét. Az aggregációnk neve avg_age lesz, ide bármilyen nevet meg lehet adni, ezzel a névvel lesz a végeredmény a válaszban. Tudjuk hogy a **bank** indexben lév? dokumentumoknak van **age** mez?je, erre akarjuk végrehajtani az **avg** függvényt.

```
curl -XPOST "http://192.168.123.71:9200/bank/_search?size=0&pretty" -H 'Content-Type: application/json' -d'
{
    "aggs" : {
        "avg_age" : { "avg" : { "field" : "age", "missing": 20 } }
    }
}
'
```

A **missing** paraméter azt mondjuk meg, hogy ilyen értékkel vegye figyelembe azokat a dokumentumokat, amikben nincs ilyen mez?, jelen esetben a age. Mert hogy egy index-ben belül sem kell hogy egyezzenek a dokumentumok, bármilyen struktúrájú dokumentumot beszúrhatunk. Ha egy dokumentumban nem lenne **age** mez?, akkor azt 20-al venné figyelembe.

Nézzük meg a végeredményt. Az elején van a szokásos statisztika, láthatjuk, hogy mind az 1000 dokumentumot figyelembe vette, majd jönnek az aggregációs eredmények az **aggregations** listában.

```
{
  "took" : 6,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
```

```
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "avg_age" : {
      "value" : 30.171
    }
  }
}
```

Egy szem eredmény van benne, az **avg_age**, aminek az értéke 30.171, vagyis az átlagéletkora az ügyfeleknek 30.171 év.

Súlyozott átlag
A súlyozott átlagnál, minden egyes értékét megszorzunk a súllyal, és az eredmények összegét elosztjuk a súlyok összegével. (A normál átlag felfogható súlyozott átlagnak, ahol minden súly = 1):

```
?(value * weight) / ?(weight)
```

A súlyozott átlag aggregációnak már nem csak egy szimpla mez? név paramétere van:

- **value**: The configuration for the field or script that provides the values (Required)
  - **field**: The field that values should be extracted from
  - **missing**: A value to use if the field is missing entirely
- **weight**: The configuration for the field or script that provides the weights (Required)
  - **field**: The field that values should be extracted from
  - **missing**: A value to use if the field is missing entirely
- **format**: The numeric response formatter (Optional)
- **value_type**: A hint about the values for pure scripts or unmapped fields (Optional)

A következ? példában nézzük meg az életkorok átlagát súlyozva a számlák egyenlegével. Tehát az átlag value mez?je továbbra is az **age**, és a súly a **balance**. A missing paraméter ugyan arra szolgál mint az el?bb.

```
curl -XPOST "http://192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
    "size": 0,
    "aggs" : {
        "waited_avg_age": {
            "weighted_avg": {
                "value": {
                    "field": "age",
                    "missing": 20
                },
                "weight": {
                    "field": "balance"
                }
            }
        }
    }
}
'
```

A válasz nagyon hasonlít az el?z? példához:

```
  "aggregations" : {
    "waited_avg_age" : {
      "value" : 30.060448837377425
    }
  }
```

{{note|A dokumentumokban, amit aggregálunk, több ugyan olyan mez? is lehet, amit a **value**-ban megadunk, pl lehet akár három darab **age** mez? is. Ilyenkor az összeset be fogja számolni a végeredménybe. Viszont a **weight** mez?b?l csak egy lehet egy dokumentumba, ha több van hibát fogunk kapni. Ilyenkor egy **script**-el lehet megmondani, hogy melyiket vegye figyelembe.

multi-value numeric metrics aggregation

A multi-value aggregációknál a végeredmény egy eredmény lista, vagy egy el?re meghatározott objektum, attól függ?en, hogy milyen típusú metrika aggregációt használunk.

```
"aggregations" : {
    "<aggregáció neve>" : {
      "values" : { aaa=xxx, bbb=ccc, ...}
    }
  }
```

- Stats aggregáció (statisztika egy mez?r?l)
- Extended Stats Aggregation
- Percentiles Aggregation
- ...

stats
A legegyszer?bb a **stats** aggregáció, ami nem csinál mást, mint a dokumentum halmazban egy kiválasztott mez?re visszaadja a következ? statisztikát: *darabszám, min, max, átlag, és summa*, tehát ez nem más mint a négy alap single-value aggregáció ötvözése. Maradva a bankos példánál, nézzük

meg a **age** mez?re a statisztikát. Nevezzük el az aggregációnkat **age_stats**-nak:

```
curl -X POST "192.168.123.71:9200/bank/_search?size=0&pretty" -H 'Content-Type: application/json' -d'
{
    "aggs" : {
        "age_stats" : { "stats" : { "field" : "age" } }
    }
}
'
```

A válasz 5 el?re rögzített mez?t tartalmaz:

```
"aggregations" : {
    "age_stats" : {
      "count" : 1000,
      "min" : 20.0,
      "max" : 40.0,
      "avg" : 30.171,
      "sum" : 30171.0
    }
  }
```

Percentiles Aggregation
Egy mintában a kiugró értékek megkeresésére szolgál. A teljes halmazt növekv? sorrendbe rendezi, majd megmutatja, hogy milyen elem van pl a halmaz hosszának a felénél, a halmaz hosszának a 90%-ánál és a 99%-ánál. Mivel a halmaz rendezett, tudhatjuk, hogy a 70%-hoz tartozó értéknél a halmazban az elemek 70%-a kisebb. Ezeket a %-okat mi határozhatjuk meg. Tipikusan a nagyobb százalékok lehetnek fontosabbak. A képlet az alábbi:

```
 my_array[count(my_array) * (százalékos érték, pl 70%)/100]
```

Ez pl válaszid?knél sokkal személetesebb mint a min/max/átlag, amib?l gyakran nem látszik hogy baj van. Pl adott a következ? rendezett halmazunk (amik szimbolizáljanak válaszid?ket másodpercben):

```
{1,2,3,4,4,5,5,6,10,20}
```

Azt szeretnénk tudni, hogy a kérések 70%-át maximum mekkora válaszid?vel tudtuk kiszámolni:

```
 my_array[count(my_array) * 0.7] = 6
```

Tehát itt a 7. elemét kell vegyük ehhez a halmaznak, tehát a kérések 70%-át 6 másodpercen belül ki tudtuk szolgálni.

A percentiles aggregációt a **percentiles** kulcsszóval kell definiálni. A keresett százalékokat a percents tömbbel kell megadni, itt növekv? sorrendben kell felsorolni számokat 0.1 és 99.9 között. Nézzük a szokásos banki életkor példát.

```
curl -X GET "192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
    "size": 0,
    "aggs" : {
        "age_perc" : {
            "percentiles" : {
                "field" : "age",
                "percents" : [1.0, 5.0, 25.0, 50.0, 75.0, 95.0, 99.0]
            }
        }
    }
}
'
```

A válaszban a stat aggregációval ellentétben nem egy fix adatszerkezet van, hanem egy dinamikus value lista, minden egyes a percents tömbben megadott százalékhoz tartozik egy érték. A percents tömb megadás nem kötelez?, ha nem adjuk meg, pont ez az alapértelmezett felosztás.
Azt láthatjuk, hogy 75% -a a banki ügyfeleknek nem id?sebb mint 35 év. És így tovább.

```
 "aggregations" : {
    "age_perc" : {
      "values" : {
        "1.0" : 20.0,
        "5.0" : 21.0,
        "25.0" : 25.0,
        "50.0" : 31.0,
        "75.0" : 35.0,
        "95.0" : 39.0,
        "99.0" : 40.0
      }
    }
  }
```

Bucket aggregation

A vödrös aggregációknál az indexben lév? dokumentumok részhalmazát egy vagy több kategóriába (vödörbe) soroljuk be. Ezekre a vödrökre aztán további aggregációkat alkalmazhatunk, vagy pusztán az érdekel minket, hogy hány dokumentum van egy vödörben (hisztogram típusú aggregációk)

Filter aggregáció

Egy darab vödröt képez egy sz?résІ feltétel alapján a megadott indexben. Igazából ez egy egyszer? keresés, aminek az eredményére aztán könnyen futtathatunk metrika típusú aggregációkat. A következ? példában els?nek összegy?jtjük a 30 éves ügyfeleket a filter bucket aggregációval, majd meghívjuk rá az avg metrika aggregációt.

```
curl -X POST "192.168.123.71:9200/bank/_search?size=0&pretty" -H 'Content-Type: application/json' -d'
{
    "aggs" : {
        "30_year_old_avg" : {
```

```
            "filter" : { "term": { "age" : "30" } },
            "aggs" : {
                "avg_price" : { "avg" : { "field" : "balance" } }
            }
        }
    }
}
'
```

És a végeredményben láthatjuk, hogy összesen 47 darab 30 éves ügyfél volt, és az ? egyenlegüknek az átlaga 22841.

```
   "aggregations" : {
    "30_year_old_avg" : {
      "doc_count" : 47,
      "avg_price" : {
        "value" : 22841.106382978724
      }
    }
```

Filters aggregáció

Ez a több vödrös változata a Filter-nek. Az aggregáció során tetsz?leges számú vödröt képezhetünk, pl logszintek alapján, pl a warnings kap egy külön vödröt, a debug kap egy külön vödröt, és az infó kap egy külön vödröt. Aztán a végeredményt tetsz?legesen tovább processzálhatjuk. A végeredmény az lesz hogy az összes érintett dokumentum az indexen belül bekerül egy vödörbe. Tegyük külön vödrökbe a 30, 31 és 32 éves banki ügyfeleket:

```
curl -X GET "192.168.123.71:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "size": 0,
  "aggs" : {
    "messages" : {
      "filters" : {
        "other_bucket_key": "other_customers",
        "filters" : {
          "30_yers_old" :   { "match" : { "age" : "30"   }},
          "31_yers_old" :   { "match" : { "age" : "31"   }},
          "32_yers_old" :   { "match" : { "age" : "32"   }}
        }
      }
    }
  }
}
'
```

Az összes nemilleszked? dokumentum a bank indexb?laz other_customers vödörbe fog kerülni.

És íme a végeredmény- 47 darab 30 éves ügyfél van, 61 darab 31 éves, 52 darab 32 éves, és ezen kívül még 840-en vannak.

```
 "aggregations" : {
    "messages" : {
      "buckets" : {
        "30_yers_old" : {
          "doc_count" : 47
        },
        "31_yers_old" : {
          "doc_count" : 61
        },
        "32_yers_old" : {
          "doc_count" : 52
        },
        "other_customers" : {
          "doc_count" : 840
        }
      }
    }
  }
```

Hisztogram

Ez egy nem kommulált hisztogramot képez automatikusan a megadott indxeben lév? dokumentumok egy mez?jéb?l. A végeredmény egy több vödrös aggregáció. A vödrök számát, vagyis a hisztogram csoportokat magától számolja ki, nekünk csak a felbontást (lépés méretet) kell megadni. A hisztogramokról b?vebben itt olvashatunk: Metrics and Monitoring in swarm/Histogram

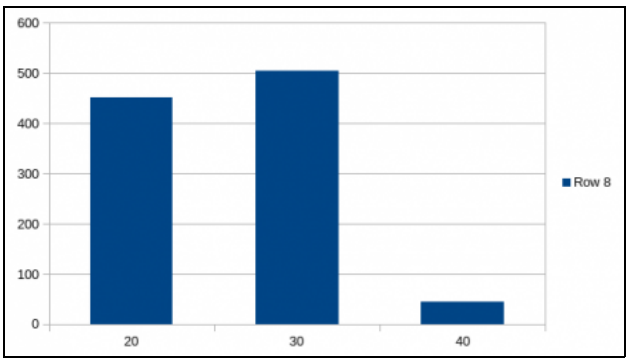
Készítsük el a banki ügyfelek korának hisztogramját 10 éves lépésekben:

```
curl -X POST "192.168.123.71:9200/bank/_search?size=0&pretty" -H 'Content-Type: application/json' -d'
{
    "aggs" : {
        "prices" : {
            "histogram" : {
                "field" : "age",
                "interval" : 10
            }
        }
    }
}
'
```

A végeredményben 3 vödör lesz (három oszlopa lesz a hisztogramnak). Van két bazi magas oszlop, és a végén egy egészen kicsi:

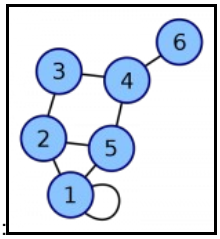
```
"aggregations" : {
    "prices" : {
      "buckets" : [
```

```
        {
          "key" : 20.0,
          "doc_count" : 451
        },
        {
          "key" : 30.0,
          "doc_count" : 504
        },
        {
          "key" : 40.0,
          "doc_count" : 45
        }
      ]
    }
  }
```



Szomszédossági mátrix

A szomszédossági mátrix-al véges gráfokat írhatunk le egy kétdimenziós mátrixban. Egy V csúcsszámú gráfban a mátrix |V| × |V| (téglalap). Ha két végpont között fut él, akkor 1 szerepel a mátrixban, ha nem fut él akkor 0. Ha nincsenek hurkok, akkor a diagonális elemek értelem szer?en 0-ák. Ha a hurok is megengedett, akkor a hurok mindig duplán számít, mint az alábbi példában az 1-es végpont.



:$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$:

....

Pipeline aggregation

A Pipeline aggregációk, ahogy azt a neve is mutatja, mindig egy metrika vagy vödrös aggregáció eredményén dolgozik (mint a Linux pipe). Két f? csoportba lehet ?ket sorolni:

Pipeline aggregations work on the outputs produced from other aggregations rather than from document sets, adding information to the output tree. There are many different types of pipeline aggregation, each computing different information from other aggregations, but these types can be broken down into two families:

- Parent
- Sibling

...

Matrix aggregations

...

Kibana Web interface

A Kibana konzolt bármelyik node "publikus" IP címén elérjük a 5601-es porton, amit publikáltunk az ingress overlay hálózaton. Keressük meg valamelyik node publikus IP címét. Sajnos a docker-machine ip nem a publikus címet adja vissza, ezért ezt csak a node-on belülr?l lehet kinyerni:

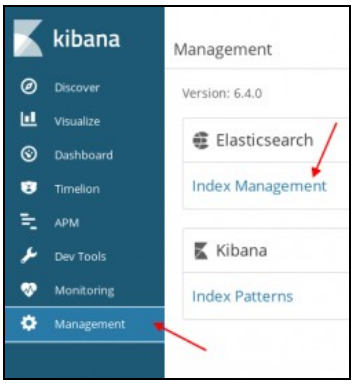
```
# docker-machine ssh mg0 ifconfig | grep -A 1 eth0 | grep "inet addr"
          inet addr:192.168.123.71  Bcast:192.168.123.255  Mask:255.255.255.0
```

Lépjünk be a Kibana konzolra: http://192.168.123.71:5601/app/kibana#/home?_g=()

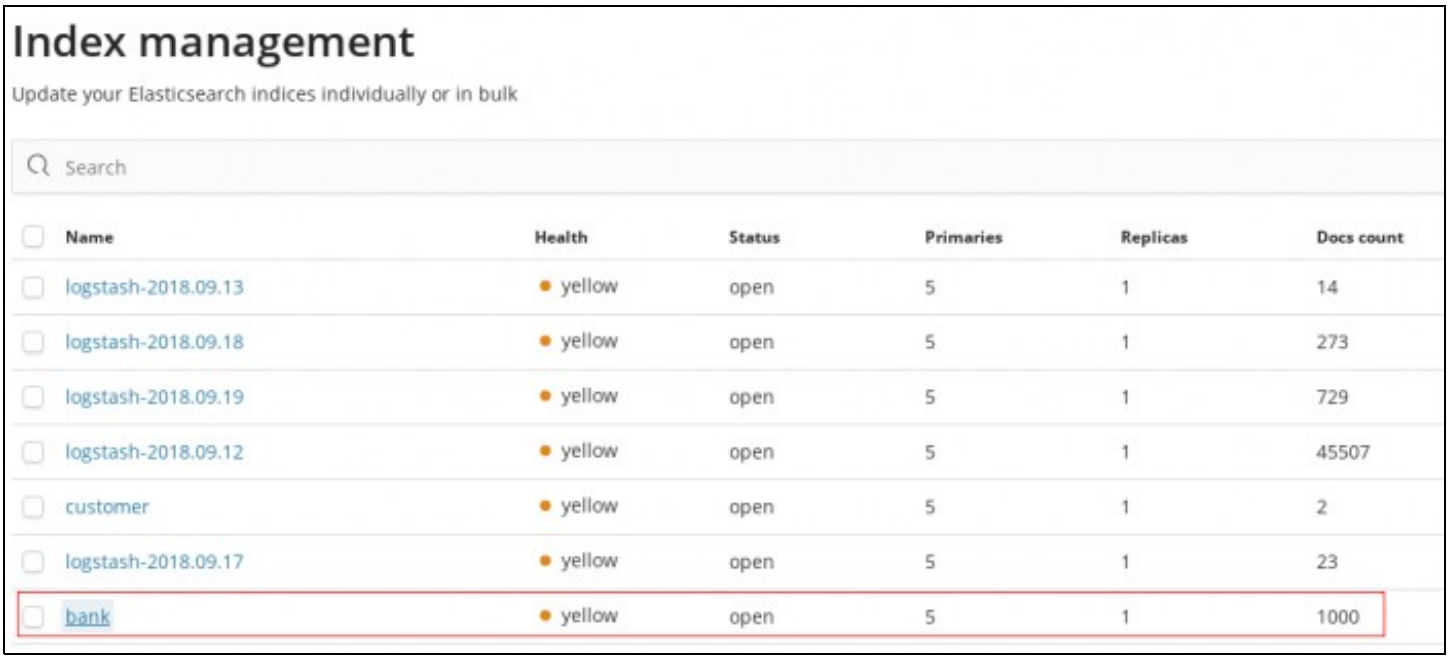
Alapok

Indexek kezelése

Els? lépésben nézzük meg az elérhet? indexeket (adatbázisokat) az Elasticsearch-ben. Kattintsunk baloldalon a **Management**-re, majd az **Index Management**-re.

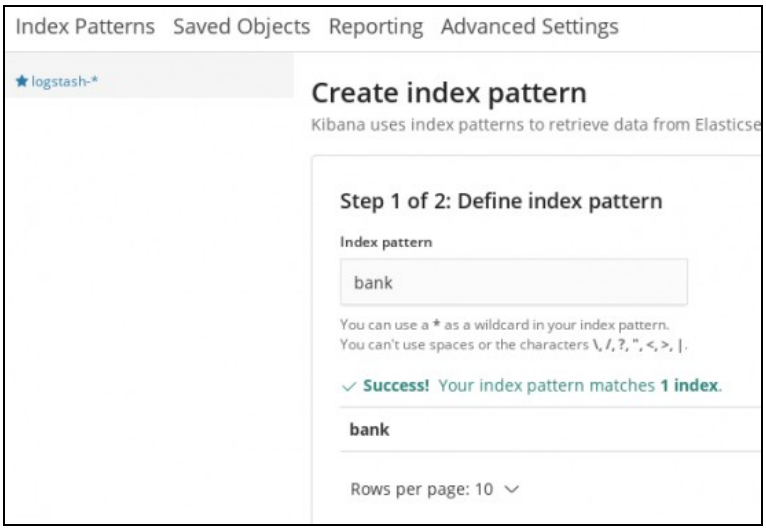


Ekkor megjelenik az index-ek listája. Láthatjuk hogy kétféle index van az adatbázis kezel?ben. Egyrészr?l minden naphoz készített a logstash egy új indexet, másrészt ott van a **bank** index, amivel kísérleteztünk mikor az Elasticsearch -el ismerkedtünk.

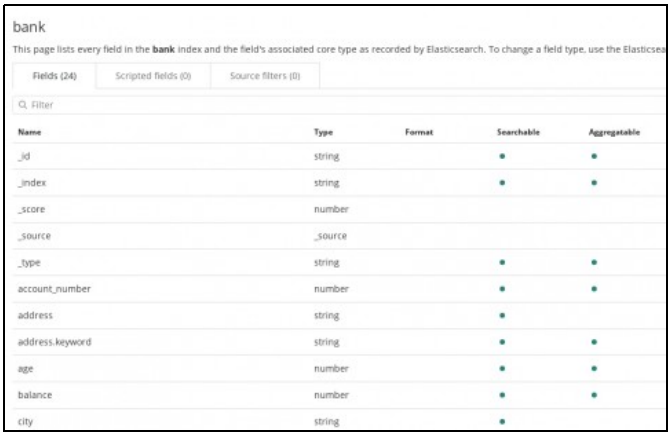


Látatjuk, hogy pontosan 1000 darab dokumentum van benne, amennyit beszúrtunk.

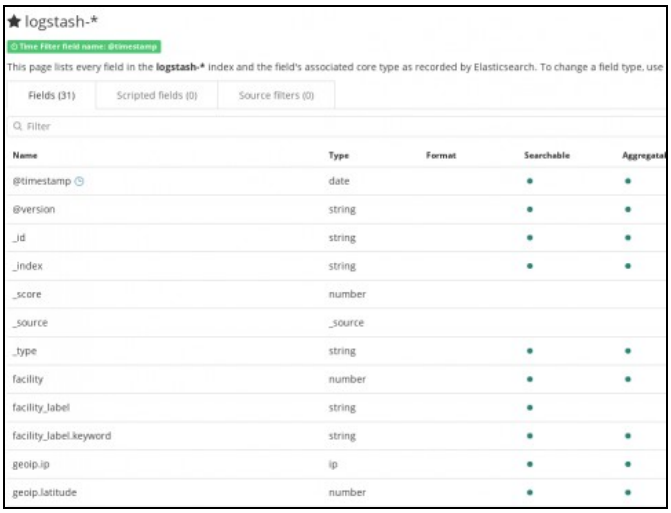
Adjunk hozzá két új index patter-t. Egyet a **bank**-hoz, egyet pedig a **logstash**-ez. Az index pattern-el több index-re is illeszked? index csoportot definiálhatunk. Mivel a logstash-hez kapcsolódó index-ek mindig logstash-<dátum> alakúak, létrehozhatjuk a **logstash-*** pattern, ami az összes logstash index-re illeszkedni fog.



Láthatjuk, hogy a bank index-hez tartozó dokumentumok 24 mez?b?l állnak. Minden mez?nél ki van írva a mez? típusa, hogy kereshet? e, és hogy aggregálható e.

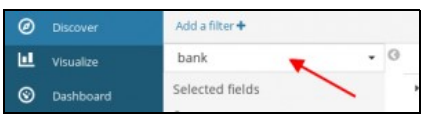


Míg a logstash-* indexekhez tartozó dokumentumok 31 mez?b?l állnak:



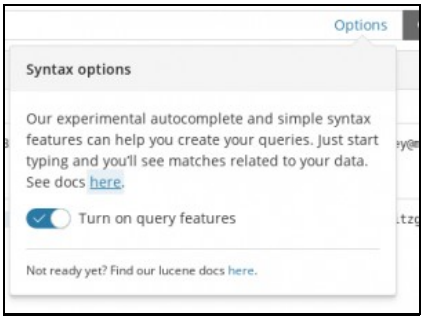
Discover

A Discover fülön egyszer? lekérdezéseket állíthatunk össze a létrehozott index pattern-ekhez. Két index pattern-ünk van, egy logstash-*, ami az összes logstash-es indexet tartalmazza, valamit a bank. A legördül? listából kell kiválasztani, hogy melyikre akarunk lekérdezni.



A lap tetején lév? mez?be írhatunk be egyszer? lekérdezéseket. Vagy szabad szavasan kereshetünk, ekkor az összes dokumentumot ki fogja dobni,

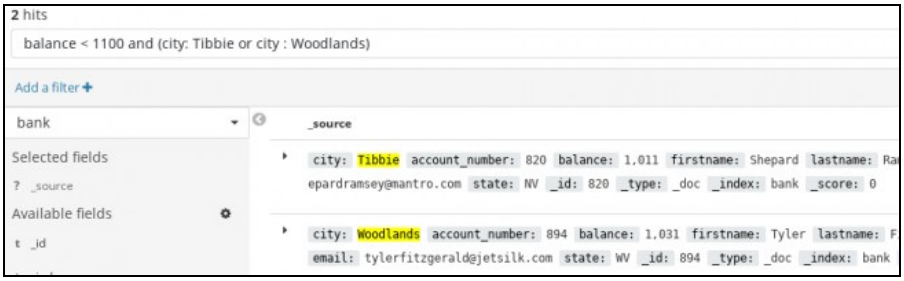
amiben szerepelt a szó bárhol az adott index pattern-ben, vagy kereshetünk a Lucene Query nyelven. Pl Els?nek kapcsoljuk be az automata kiegészítést az **Options**-re kattintva:



Majd kérdezzük le azokat az ügyfeleket, akinek a balanca kisebb mint 1100:

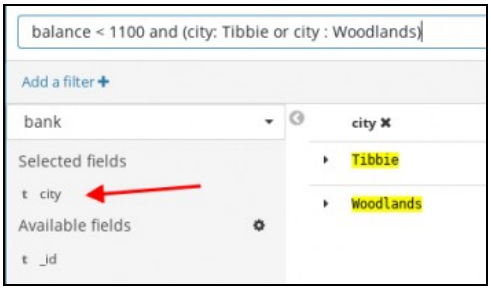
Keressük meg az összes olyan dokumentumot, ahol a balance kisebb mint 1100 és a város vagy Tibbie vagy Woodlands:

```
balance < 1100 and (city: Tibbie or city : Woodlands)
```



A megjelenített dokumentum melletti kis nyílra kattintva megnézhetjük az egyes találatokat tábla és JSON nézetben is:

A baloldali listában filtereket adhatunk meg, hogy a Dokumentumnak csak a kiválasztott részeit mutassa meg, pl sz?kíthetjük a city mez?re a válaszokat:



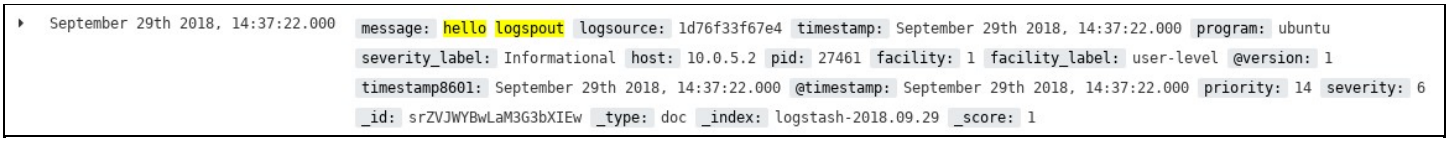
Ha átmegyünk a **logstash-*** index pattern-re akkor megjelenik két új elem a Discover képerny?n. Mivel a logstash-* index tartalmaz egy Timestamp típusú mez?t (kis óra jelzi), ebb?l a Kibana kapásból egy hisztogramot fog kirajzolni, és meg is jelenik egy id?-intervallum választó a jobb fels? sarokban:

Note
A legjobb a **Discover** képerny?ben, hogy szabad szavasan is lehet benne keresni, írjunk be bármilyen szöveget, amelyik logban az szerepelt, azt meg fogja mutatni, pl egy IP cím, vagy bármilyen más id

Most keressünk a logstash-* patternekben.
Listázzuk ki az összes olyan logstash-* indexbe tartozó log bejegyzést, amit nem az elastch termékek generáltak:

```
NOT program: kibana* and NOT program : logmanager_*
```

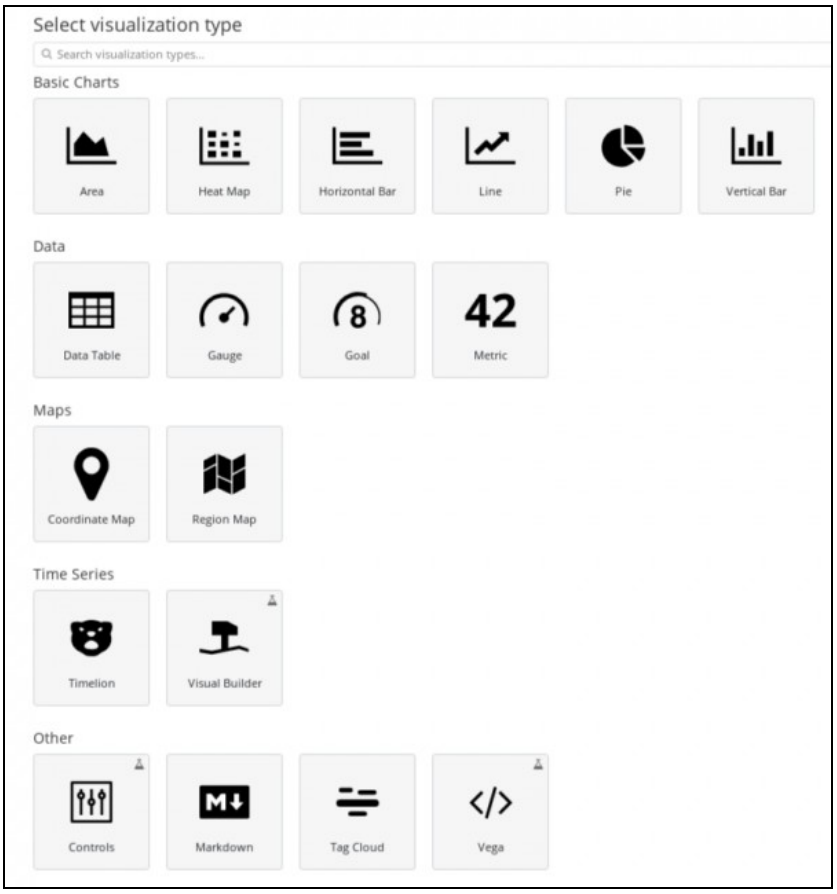
Vagy szimplán keressünk rá a "hello logspout" üzenetre, amit a tesztelés céljából indított ubuntu konténer küldött be. Mivel szabad szavasan is lehet keresni, csak írjuk be felülre hogy "hello logspout".



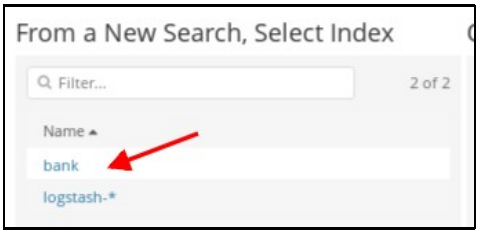
Note
Figyeljünk a jobb fels? sarokban keresési intervallum beállításokra. Alapértelmezetten csak 15 perces adatokban keres

Visualize

A Vizualize felületen 18 féle diagram típusból választhatunk:



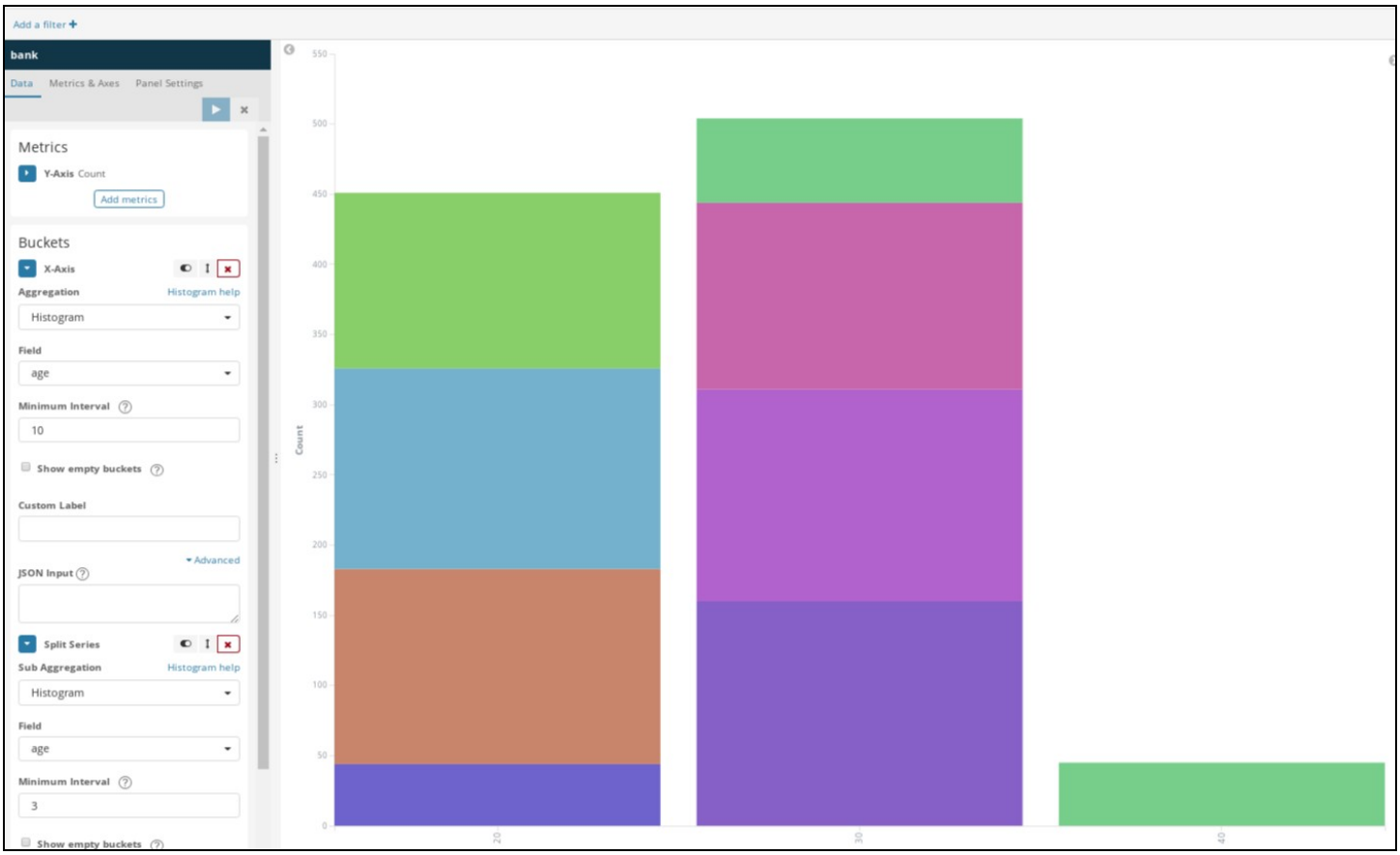
Válasszuk ki az indexet:



Általában az Y tengelyre metrika aggregációkat helyezhetünk el, az X tengelyre meg vödrös aggregációkat.

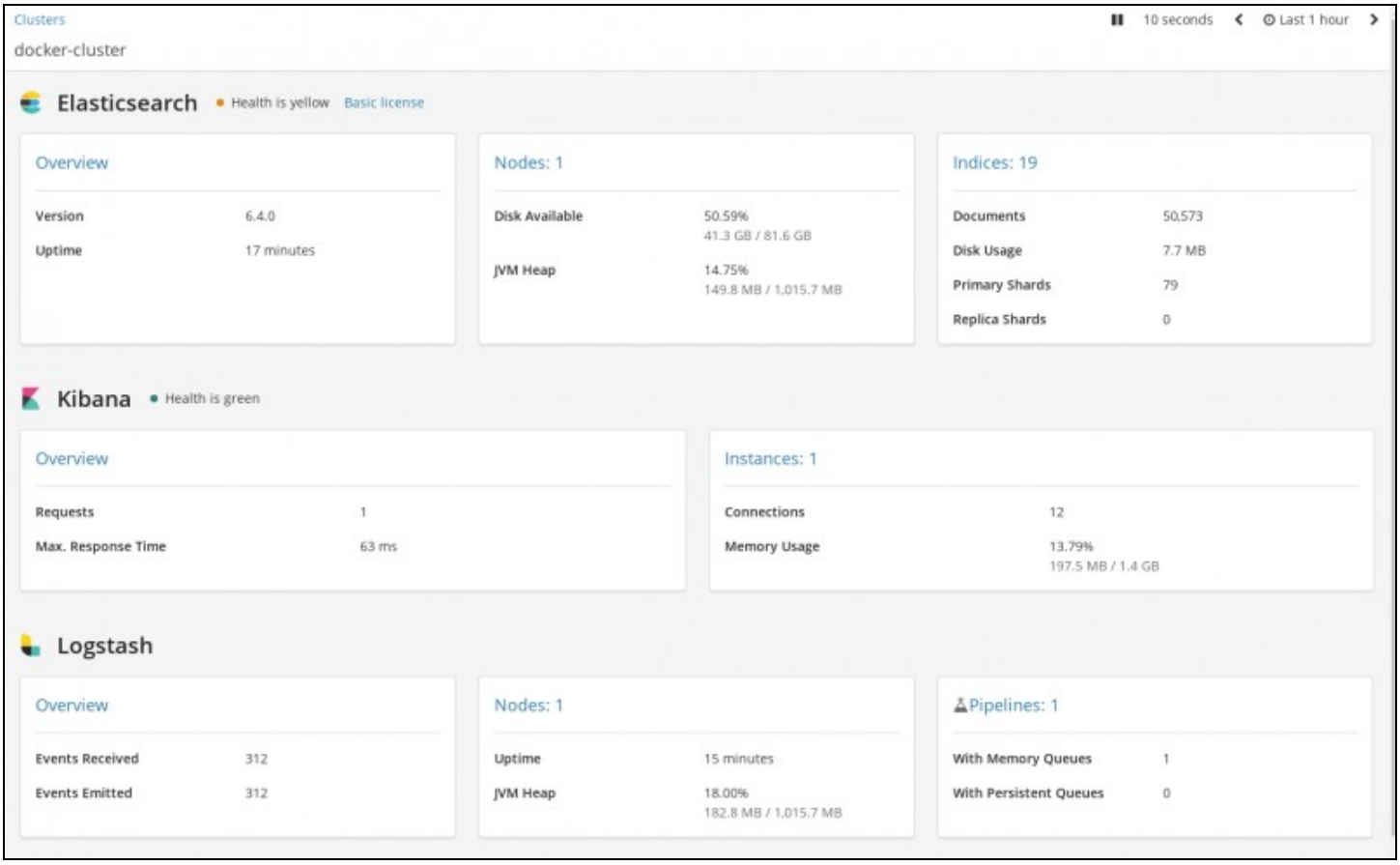
- Az Y tengelyre rakott metrika típusú aggregációk esetén a single-value típus csak egy vonalat fog eredményez (pl count, avg) míg a multi-value több sávot (pl. Pertentiles). Az Y tengelyen mindig a megtalált darabszám lesz (dokumentum darabszám) és ha f?lé visszük az egeret, megmutatja az értkét
- Az X tengelyre rakott aggregációk esetén az X tengelyen lesznek a vödrök, míg az Y tengelyen a vödrök értéke.

Készítsük el újra a banki ügyfelek korának hisztogramját, amit a parancssoros lekérdezésben már megnéztünk. Ehhez válasszuk az X-Axis lehet?séget, majd ott a listából válasszuk ki a Histogram aggregációt 10 éves felbontással az **age** mez?re. Majd adjunk hozzá sub-aggregációt szintén az **age** mez?re de most már csak 3 éves felbontással. Láthatjuk hogy a három nagy oszlopot még felbontotta 3 éves részekre:



Monitoring the ELK stack

A Kibana out of the box képes monitorozni a teljes ELK stack minden porcikáját, fel tudja kutatni mind a három komponens összes node-ját, ehhez nincs más dolgunk, mint hogy a **Monitoring** menüpontra kattintsunk, és elindítsuk a monitorozást:



Swarm stack

elastic-stack.yml

```
version: '3'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.4.0
    ports:
      - "9200:9200"
    networks:
      - elk
    volumes:
      - "elasticsearch-conf:/usr/share/elasticsearch/config"
      - "elasticsearch-data:/usr/share/elasticsearch/data"
    environment:
      - "discovery.type=single-node"
    deploy:
      restart_policy:
        condition: on-failure
      resources:
        reservations:
          memory: 500m
  logstash:
    image: docker.elastic.co/logstash/logstash:6.4.0
    networks:
      - elk
    environment:
      - "LOGSPOUT=ignore"
    volumes:
      - "logstash-conf:/usr/share/logstash/pipeline"
    deploy:
      restart_policy:
        condition: on-failure
      resources:
        reservations:
          memory: 100m
  logspout:
    image: gliderlabs/logspout:v3.2.5
    networks:
      - elk
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    environment:
      - "SYSLOG_FORMAT=rfc3164"
    command:
      - "syslog://logstash:51415"
    deploy:
      mode: global
      restart_policy:
        condition: on-failure
```

```
  kibana:
    image: docker.elastic.co/kibana/kibana:6.4.0
    networks:
      - elk
    volumes:
      - "kibana-conf:/usr/share/kibana/config"
      - "kibana-data:/usr/share/kibana/data"
      - "kibana-plugins:/usr/share/kibana/plugins"
    environment:
      - "LASTICSEARCH_URL=http://elasticsearch:9200"
    ports:
      - 5601:5601
    deploy:
      restart_policy:
        condition: on-failure

networks:
  elk:
    driver: overlay

volumes:
  elasticsearch-conf:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/config
  elasticsearch-data:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/data
  logstash-conf:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/logstash/config
  kibana-conf:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/kibana/config
  kibana-data:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/kibana/data
  kibana-plugins:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/kibana/plugins



# docker stack deploy --compose-file docker-compose.yml logmanager
```

Ugyan a service-ek nevébe a swarm bele fogja tenni a logmanager el?tagot (ami a stack neve), ett?l függetlenül a compose fájlban szerepl? névvel a stack service-ek még mindig tudják használni a DNS névfeloldást egymás IP címeinek a kitalálására az **elk** overlay hálózaton. (A hálózat nevébe is megváltozik: logmanager_elk)

```
# docker stack ls
NAME                SERVICES
logmanager          4

# docker stack ps logmanager ...
```

Elasticsarch cluster

Áttekintés

Alap esetben az ES cluster építése egy automatizált folyamat, a user el?l el van fedve. Kicsit leegyszer?sítve nincs más dolgunk, mint hogy elindítani a kívánt számú ES példányt ugyan azzal a cluster névvel egy közös hálózaton, a cluster felépítése teljesen automatikusan végbe fog menni.

Négyféle alap node típus van:

- **Master-eligible node**: A master node-ok vezérli a cluster infrastruktúrát. Nyilván tartja a cluster tagokat, vezérli az index-ek létrehozását, törlését, valamint dönt róla hogy shard melyik node-ra kerüljön (node.master = true)
- **Data node**: Ezek a node-ok tárolják az adatbázis adatokat és hajtrák végre az adatmanipulációs és keres? m?veleteket, lényegében ?k a munkások. (node.data = true)
- **Ingest node**: Ingest nodes are able to apply an ingest pipeline to a document in order to transform and enrich the document before indexing. With a heavy ingest load, it makes sense to use dedicated ingest nodes and to mark the master and data nodes as node.ingest: false.
- **Tribe node**: több cluster között képes kapcsolatot teremteni, az egyetlen node típus ami több cluster-nek is a tagja lehet.
- **Coordinating node**: A kliens kéréseket a Coordinating node-ok kapják meg és továbbítják a data node-oknak, akik a keresés eredményét visszaküldik a keresést indító Coordinating node-nak, összegzi az eredményeket és visszaküldi a kliensnek. Lényegében minden node egyben Coordinating node is, tehát bárhova beérkezhet a kliens kérés. Azonban nagyon nagy terhelés mellett készíthetünk dedikált Coordinating node-okat, amiken az el?z? négy típust kikapcsoljuk, és a klienseknek csak ezen node-okon keresztül kommunikálhatnak a cluster-el.

Elviekben egy node egyben több szerepben is lehet, s?t, alap beállítások mellett minden egyes létrehozott node egyben master, data és ingest node is egyben. Ez kis cluster méret mellett ideális, nincs más dolgunk mint hogy ugyan azokkal a beállításokkal elindítunk pl 5 node-ot, ezek automatikusan cluster-t fognak formálni és meg fogják választani a vezet?t. Nagyobb terhelés mellett viszont már érdemes specializált node-okat létrehozni, külön master és külön data node-okat. Ezen felül érdemes lehet szintén dedikált coordinating node-okat is bevezetni.

A node-ok létrehozásakor a minimum beállítás:

- Az interfész, ahol eléri a többi node-ot a cluster-ben
- A cluster node-ok listája
- Cluster név, ami azonosítja a cluster-t ahova csatlakoznia kell.

Warning
Saját adat mappa minden data és master node-nak: Fontos hogy a data és master node-oknak saját data mappája legyen, amin nem osztozik más node-okkal, mert akkor összekeveredhetnek.

Két megközelítés közül választhatunk:

- **Automatikus cluster formálás:** egy darab swarm service-t definiálunk, és egyszer?en meghatározzuk a replikák számát, elindul több konténerben az Elasticsearch egy swarm service-ként: http://derpturkey.com/elasticsearch-cluster-with-docker-engine-swarm-mode/
- **Kézi cluster létrehozás:** Minden egyes cluster tagot külön swarm service-ként definiálunk a compose fájlban 1-es replika számmal, tehát el?re pontosan megmondjuk, hogy hány darab fog futni, és hogy melyik node-nak milyen szerepe van: http://blog.ruanbekker.com/blog/2018/04/29/running-a-3-node-elasticsearch-cluster-with-docker-compose-on-your-laptop-for-testing/

Note
A swarm-ra azért van szükség, hogy könnyedén ki tudjuk telepíteni a távoli VM-re az ES konténereket. Swarm nélkül minden egyes VM-re nekünk kéne kézzel kitenni.

Discovery

Az Elasticsearch a "Zen Discovery" szolgáltatást használja a cluster node-ok felkutatására és a master node kiválasztására.

discovery.zen.ping.unicast.hosts
Ebben a paraméterben kell felsorolni a node-ok listáját. Szerencsére itt meg lehet adni olyan host nevet is, ami több IP címére oldódik fel. A swarm cluster-ben minden service névvel indított DNS lekérdezésre a swarm visszaadja az összes konténer IP címét akik a szolgáltatáshoz tartoznak.

discovery.zen.minimum_master_nodesedit
Ebben a paraméterben kell megadni, hogy hány master node-nak kell jelen lennie egyszerre, ahhoz hogy fenntartónak ítéljék meg az egyes nódok a cluster-t. Ezzel el lehet kerülni, hogy hálózati hiba estén, mikor a cluster két fele izolálódik egymástól önálló életre keljen a két oldal, mert mind a kett? azt hiszi, hogy ?k teljes cluster-t alkotnak, és beindul egy párhuzamos m?ködés, ami visszafordíthatatlan károkat okozna a cluser-ben. (split brain)

```
(master_eligible_nodes / 2) + 1
```

To explain, imagine that you have a cluster consisting of two master-eligible nodes. A network failure breaks communication between these two nodes. Each node sees one master-eligible node? itself. With minimum_master_nodes set to the default of 1, this is sufficient to form a cluster. Each node elects itself as the new master (thinking that the other master-eligible node has died) and the result is two clusters, or a split brain. These two nodes will never rejoin until one node is restarted. Any data that has been written to the restarted node will be lost.

Now imagine that you have a cluster with three master-eligible nodes, and minimum_master_nodes set to 2. If a network split separates one node from the other two nodes, the side with one node cannot see enough master-eligible nodes and will realise that it cannot elect itself as master. The side with two nodes will elect a new master (if needed) and continue functioning correctly. As soon as the network split is resolved, the single node will rejoin the cluster and start serving requests again.

Perzisztencia

Ez itt a legnagyobb kérdés. Még akkor is ha nem dinamikusan létrehozott VM-eken futtatjuk az ES cluster-t, a swarm minden egyes újraindításkor más és más node-ra fogja rakni ugyan azt a node-ot.

...

Produkciós beállítások

Ha kivesszük a **discovery.type=single-node** paramétert, és ezen felül még a **network.host** paramétert is beállítjuk, az ES produkciós üzemmódban fog elindulni. Produkciós indulás közben sokkal szigorúbban ellen?rzi a kötelez? beállításokat. Ebb?l a legfontosabb host operációs rendszernek (jelen esetben a boot2docker) a **vm.max_map_count** beállítása, amit fel kell emelni minimum **262144**-ra. Ha ez kevesebb, az adott node nem fog elindulni.

```
docker-machine ssh mg0
...
sudo sysctl -w vm.max_map_count=262144
```

Egy lehetséges megoldás

Mivel minden master és data node-nak saját perzisztencia store-ra van szüksége nem tehetjük meg simán egy darab swarm service-ként elindítjuk a cluster-t és aztán felskálázzuk (docker swarm scale). Tehát az világos, hogy minden data és manager node-ot külön service-ként kell definiálni. Viszont az ingress overlay hálózatra csak egy service-hez tudjuk a 9200-as portot definiálni. (feltéve, ha el akarjuk érni kívülr?l). Szerencsére a koordinációs node-oknak (amik nem végeznek se master, de data se Ingest tevékenységet, kizárólag a kliensek kéréseit rout-olják a megfelel? node-okhoz) nem kell hogy legyen mentett data mappája, így ezeket létre tudjuk hozzon több elem? swarm service-ként, mint Elasticsearch belépési pont, és akkor a swarm ingress hálózat még meg is oldja a load-balancing-ot.

Közös konfigurációs fájl

A közös konfigurációs fájlba felvesszük az összes cluster tag végpontját **discovery.zen.ping.unicast.hosts** paraméterben. A listában minden egyes sor egy swarm service neve, amit a swarm DNS felold konténer IP címére. Egyedül a **elasticsearch_coord** lesz több konténerb?l álló szolgáltatás, amik rá lesznek kötve az ingress hálózatra is, ezek lesznek az ES cluster belépési pontjai. Szerencsére a zen discovery képes olyan DNS válaszokat is kezelni, amik több végpontot adnak vissza.
Az alábbi fájlt fel fogjuk csatolni az összes service-be NFS megosztással.
/usr/share/elasticsearch/config/**elasticsearc.yml**

```
cluster.name: "my-cluster"
network.host: 0.0.0.0

discovery.zen.minimum_master_nodes: 2

discovery.zen.ping.unicast.hosts:
  - elasticsearch_coord
  - elasticsearch1
  - elasticsearch2
  - elasticsearch3
```

Az összes ES node a közös elk nev? overlay hálózaton tud majd közvetlen kommunikálni egymással.

Coordinating node-ok

A Coordinating node-okat több elem? swarm service-ként fogjuk létrehozni. Ezek a node-ok lesznek a ES cluster belépési pontjai. Egyedül ebben a swarm service-ben lesz több mint egy konténer. Data mappát nem is csatolunk fel hozzá. Ahhoz hogy coordinating node-ként viselkedjen egy node be kell állítani, hogy se nem data, se nem master és se nem ingest tevékenységet nem végezhet. Ehhez létrehozhattunk volna egy külön konfigurációs fájlt a coordinating node-oknak, mi most itt beírtuk környezeti változóba. 3 példányt kértünk bel?le. Az ingress hálózaton a 9200 -as porton érhetjük majd el a coordinating node-okat bármelyik swarm node IP címén.

```
elasticsearch_coord:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.4.0
    ports:
      - "9200:9200"
    networks:
      - elk
    volumes:
      - "es-conf:/usr/share/elasticsearch/config"
    environment:
      - node.data=false
      - node.master=false
      - node.ingest=false
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
```

További node-ok definiálása

Mivel most nem akarunk hatalmas cluster-t építeni, három további node-ot fogunk a cluster-hez adni, amik már mind a három szerepkörben benne lesznek (master, data és ingest). Mivel a master és az adat node-oknak már saját data mappára van szüksége, minden node-ot egy külön swarm service-ként fogunk definiálni saját volume plugin megosztással a perzisztens store-ban. Így bárhol is hozza létre ?ket a swarm, mindig ugyan azt a data mappát fogják megkapni.

```
elasticsearch1,2,3:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.4.0
    ports:
      - "9200:9200"
    networks:
      - elk
    volumes:
      - "es-conf:/usr/share/elasticsearch/config"
      - "es-data1,2,3:/usr/share/elasticsearch/data"
    environment:
      - node.name=node1,2,3
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
```

A fenti compose blokkot háromszor kell a compose fájlba rakni a megfelel? sorszámmal a service, node és volume megosztás nevében (1,2,3)

Docker stack fájl

Warning
Nincs még frissítve...

```
version: '3'
services:
 ....

networks:
  elk:
    driver: overlay

volumes:
  elasticsearch-conf:
    driver: nfs
    driver_opts:
```

```
    share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/config
es-data1:
  driver: nfs
  driver_opts:
    share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/data1
es-data2:
  driver: nfs
  driver_opts:
    share: 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/elasticsearch/data2
```