OpenSearch

Posada en marxa d'OpenSearch (i d'OpenSearch-Dashboards)

A https://opensearch.org/downloads.html (i, de forma més "crua", a https://opensearch.org/artifacts) es troben els diversos formats distribuïbles del software OpenSearch (a partir d'ara l'anomenarem "OS") i OpenSearch Dashboards (a partir d'ara l'anomenarem "OSD"). Desgraciadament, a dia d'avui encara no estan disponibles els paquets .deb d'aquests programes (els .rpm sí!), així que no els podrem instal·lar d'aquesta manera. És per això que optarem per implementar un servidor OS (i també OSD) mitjançant contenidors.

NOTA: Una altra opció seria implementar OS/OSD mitjançant màquines virtuals VirtualBox ja preparades a través de la infraestructura Vagrant oficial, com es veu a https://github.com/opensearch-project/OpenSearch/blob/main/Vagrantfile

Concretament, per tenir un servidor OS i OSD funcionant en un sistema Linux només caldrà executar (tenint prèviament instal·lat el paquet "**podman**") el següent script (executat com usuari normal, no cal ser administrador), el qual crea un únic "pod" format per dos contenidors (un implementant el servidor OS i un altre implementant el servidor OSD) que es comuniquen entre sí via la 127.0.0.1

```
#!/bin/bash
podman pod create -n miOS -p 9200:9200 -p 5601:5601
podman container create --pod=miOS -e "discovery.type=single-node"
        -e "OPENSEARCH_INITIAL_ADMIN_PASSWORD=Un4C0ntraSs3ny4C0mplicad4_"
        docker.io/opensearchproject/opensearch:latest
podman container create --pod=miOS -e "opensearch.username=admin"
        -e "opensearch.password=Un4C0ntraSs3ny4C0mplicad4_"
        -e "opensearch.ssl.verificationMode=none"
        docker.io/opensearchproject/opensearch-dashboards:latest
podman pod start miOS
```

NOTA: El "pod" obre tres ports a l'exterior: el port 9200 és per on escolta el servidor OS les consultes HTTPS fetes de l'exterior (com per exemple les fetes manualment amb *curl* o les fetes des del propi servidor OSD), el port 9600 també l'obre el servidor OS però és només per rebre peticions molt específiques relacionades amb l'anàlisi de rendiment del propi servidor, així que no el farem servir (i per això no l'indiquem) i el port 5601 és per on el servidor OSD ofereix a l'usuari el panell web per connectar-s'hi.

NOTA: Els valors indicats als diferents paràmetres *-e* indiquen la configuració concreta que tindran els contenidors dels servidors OS i OSD. Normalment aquests valors hauran de venir indicats en el fitxer de configuració respectiu ("opensearch.yml" i "opensearch-dashboards.yml" respectivament) o bé fent ús de l'API de configuració de clústers (https://opensearch.org/docs/latest/opensearch/rest-api/cluster-settings) però en el cas d'implementar els servidors via contenidors, és més senzill fer-ho directament a través del paràmetre *-e* Concretament, els valors més comuns que se solen indicar per servidors OS són els següents (per saber què signifiquen la resta d'opcions de configuració d'OpenSearch es pot consultar la documentació oficial a https://opensearch.org/docs/latest/opensearch/configuration):

* "**node.name**" : nom del servidor OS. Per defecte no està assignada i agafa el valor "hostname" del sistema.
* "**network.host**" : adreça IP per on serà accessible el servidor OS des de l'exterior. Per defecte no està assignada a cap adreça de cap tarja; això fa que, a la pràctica, el servei OpenSearch només estigui disponible a través de la IP 127.0.0.1 (és a dir, no disponible des de l'exterior del contenidor o, en general, des de l'exterior de qualsevol sistema...a no ser que s'implementi algun tipus de servidor intermediari invers com Apache o HAProxy per redirigir les sol·licituds remotes al servidor OS local, o bé alternativament, un servidor OSD, -aquest sí escoltant a través de la xarxa-). No obstant, gràcies a la redirecció automàtica de ports que fa el "pod" des de l'interior del contenidor a l'exterior, això no és problema i no caldrà establir cap valor concret a aquesta directiva
* "**http.port**" : port d'escolta de les peticions HTTPS. Per defecte és el 9200, com ja hem dit
* "**path.data**" : rutes on es guardaran les dades
* "**path.logs**" : rutes dels fitxers "log" del servidor OS
* "**discovery.type**" : veure nota següent

NOTA: Per defecte, és necessari configurar un "cluster" de nodes abans de poder posar en marxa un servidor OS perquè si no, aquest fallarà. Això vol dir que per iniciar OS caldrà definir explícitament un conjunt de màquines "elegibles com a mestres". La idea és que un servidor OS només pugui atendre peticions remotes si forma part d'un conjunt de servidors prèviament establert, (l'anomenat "clúster"), amb el qual es coordinarà automàticament per tal de poder gestionar i emmagatzemar les dades de forma distribuïda i sincronitzada entre els diversos nodes del clúster. Això és fa així perquè, d'aquesta manera, el comportament d'OS és molt més resistent i tolerant a fallades o atacs i també perquè aquesta

arquitectura permet balancejar dinàmicament tant la càrrega habitual com els pics puntuals de forma molt òptima. No obstant, aquesta manera de funcionar per defecte impedeix, tal com hem dit, que es pugui tenir funcionant un servidor OS d'un sol node. La solució més segura i professional seria definir alguna altre "node mestre" remot per tal de formar un "cluster" de, com a mínim, dos nodes però, afortunadament, si només es volen fer proves amb un sol servidor OS aquesta infraestructura no serà necessària: es pot executar un servidor OS d'un sol node posant-lo en "mode de desenvolupament", que es tradueix a la pràctica simplement en establir la directiva "discovery.type" amb el valor "single-node". En fer-ho, en iniciar-se el servidor OS les comprovacions d'arrencada s'evitaran i el node únic s'escollirà com a node mestre i no s'unirà a un clúster amb cap altre node. En tot cas, per obtenir més informació sobre com configurar un clúster OS i definir els rols dels diferents "nodes" que l'integren (aspecte que en aquest document no veurem perquè és un tema força avançat), podeu llegir https://opensearch.org/docs/latest/opensearch/cluster

NOTA: En el cas del servidor OSD també tenim diverses opcions que podrem especificar amb el paràmetre *-e* de la comanda *podman*. Els més comuns són:

* "**server.host**" : adreça IP per on serà accessible el servidor OSD des de l'exterior. Per defecte no està assignada a cap adreça de cap tarja; això fa que, a la pràctica, el servei OSD només estigui disponible a través de la IP 127.0.0.1 (és a dir, no disponible des de l'exterior del contenidor o, en general, des de l'exterior de qualsevol sistema). Normalment aquest valor caldria canviar-ho per tal de poder accedir via navegador al servidor OSD des de qualsevol màquina client remota (per exemple, indicant el valor "0.0.0.0" per fer que el servidor OSD escolti per totes les IP que tingui establertes el sistema on estigui funcionant) però, no obstant, gràcies a la redirecció automàtica de ports que fa el "pod" des de l'interior del contenidor a l'exterior, això no serà problema i no caldrà establir cap valor concret a aquesta directiva
* "**server.port**" : port on escoltarà OSD les peticions dels clients web -és a dir, dels navegadors-; per defecte el seu valor és 5601 i en principi no el canviarem
* "**opensearch.hosts**" : array de URLs que representen el/s servidor/s OS on el servidor OSD haurà de connectar-se per obtenir les dades a mostrar; per defecte val *["https://127.0.0.1:9200]"* així que, per tant, no caldria canviar-lo (a no ser que haguéssim canviat la IP i/o port d'escolta del nostre servidor OS).
* "**opensearch.ssl.verificationMode**" : si val "none", indica que OSD, actuant com a client HTTPS del servidor OS, no comprovarà la validesa del certificat emès per aquest. Aquest paràmetre és necessari perquè el contenidor oficial d'OS, per poder funcionar sobre HTTPS d'una forma ràpida i còmoda, autogenera un certificat autosignat. Si no indiquéssim "none", OSD es queixaria llavors de què està accedint a un servidor amb un certificat no vàlid i, per tant, no hi acabaria connectant. Òbviament, aquest certificat només és útil per fer proves com les que farem a classe, però en un entorn de producció hauria de substituir-se per un certificat signat per una autoritat de certificació reconeguda i llavors aquesta directiva ja no caldria indicar-la (no obstant, aquest és un tema avançat que no veurem). Per més informació, veieu https://opensearch.org/docs/latest/dashboards/install/tls
* "**opensearch.username**" : usuari amb el qual OSD s'autenticarà contra el servidor OS per poder enviar-li les peticions pertinents (i rebre'n les respostes corresponents, per tal de mostrar-les gràficament a la pantalla). El contenidor d'OS per defecte té configurat l'usuari "admin" com a únic usuari habilitat (i amb permisos totals d'administració; més endavant, si es vol, mitjançant aquest usuari "admin" es podran crear altres usuaris amb permisos més específics). Cal tenir en compte que aquest usuari serà també el que es demani interactivament en voler accedir l'usuari al panell de control web d'OSD; en aquest sentit cal tenir clar que en tot cas qui autentica no és OSD sinó que aquest fa d'intermediari per reenviar les credencials al servidor OS, que és qui realitza l'autenticació (la qual, segons l'usuari i contrasenya que s'hagin indicats, permetrà donar accés a totes les dades indexades pel servidor OS, o a només una certa part o directament no donar accés).
* "**opensearch.password**" : contrasenya de l'usuari anterior. Per defecte és "admin"
* "**server.ssl.enabled**" : per defecte val *false*. Això vol dir que el servidor OSD és accessible només per HTTP (i no HTTPS). Per activar l'accés al panel de control web via HTTPS, a més d'establir a *true* aquesta directiva, llavors també caldria indicar la ruta del certificat pertinent (creat amb anterioritat amb alguna eina especialitzada, com la suite OpenSSL, per exemple) amb la directiva "**server.ssl.certificate**" i la de la clau privada associada amb la directiva "**server.ssl.key**". En tot cas, ho deixem com a exercici

NOTA: En qualsevol moment es pot veure si tenim funcionant o no el "pod" en qüestió amb la comanda *podman pod ls*. Si volguéssim esborrar el "pod",caldrà parar-lo primer amb *podman pod stop miOS* i llavors es pot fer *podman pod rm miOS*

NOTA: Les comandes anteriors instal·len un únic servidor OS/OSD. Aquesta configuració és molt bàsica: qualsevol implementació sèria implica, però, un clúster de diversos servidors OS distribuïts sincronitzats entre sí per emmagatzemar i indexar la informació de forma coherent i íntegra al llarg de tots els nodes. En aquest sentit, a https://opensearch.org/docs/latest/opensearch/install/docker es pot consultar com es podria posar en marxa un servidor OS format per dos nodes de tipus Docker/Podman

NOTA: En realitat, el/s node/s OS per una banda i el servidor OSD per l'altra es poden instal.lar cadascun en màquines diferents. De fet, això és el més habitual: tenir el servidor OSD funcionant en una màquina separada del servidor OS (o millor dit, diferent del clúster de servidors OS, veure nota anterior). En aquest sentit, si volguéssim posar en marxa només un servidor OS (igualment mitjançant contenidors), la comanda a executar seria: *podman container run -d --name=servOS -p 9200:9200 -p 9600:9600 -e "discovery.type=single-node" -e "OPENSEARCH_INITIAL_ADMIN_PASSWORD=Un4C0ntraSs3ny4C0mplicad4_" opensearchproject/opensearch:latest* , on es pot veure com el mapeig de ports es fa a nivell de contenidor individual i, a més, hem afegit el paràmetre **--name** per tal d'indicar el nom del contenidor en qüestió i el paràmetre **-d** per arrencar el contenidor en segon pla (i si volguéssim posar en marxa només un servidor OSD (igualment mitjançant contenidors), la comanda a executar seria: *podman*

container run -d --name=servOSD -p 5601:5601 -e "opensearch.username=admin" -e "opensearch.password=Un4C0ntraSs3ny4C0mplicad4_" -e "opensearch.ssl.verificationMode=none" opensearchproject/opensearch-dashboards:latest). En qualsevol moment es pot veure si tenim funcionant o no el contenidor en qüestió amb la comanda *podman container ls -a*. Si volguéssim esborrar el contenidor,caldrà parar-lo primer amb *podman container stop servOS* i llavors es pot fer *podman container rm servOS*. També podem iniciar un contenidor parat amb *podman container start servOS* En tot cas, però, als exercicis, per simplicitat, optarem per posar en marxa un pod contenint ambdós servidors i ja està.

NOTA: L'arxiu de configuració d'un servidor OpenSearch (si més no, funcionant en forma de contenidor) és "**/usr/share/opensearch/config/opensearch.yml**", el qual és un fitxer de tipus YAML. Allà podríem deixar-hi guardades les diferents directives que a l'script de la primera pàgina hem hagut d'indicar explícitament mitjançant el paràmetre *-e* (com "discovery.type" però també "node.name", "network.host", "http.port", etc). D'aquesta manera, cada cop que arrenqués el contenidor en qüestió ja tindria aquesta configuració establerta per defecte. En principi això no caldrà que ho fem en els exercicis demanats en aquest document, però si volem utilitzar un fitxer "opensearch.yml" personalitzat, es bo saber que tenim dues opcions:

*) Entrar en un terminal de dins del contenidor (així: *podman exec -it nomContenidor /bin/bash* , on el paràmetre *-it* indica que es vol iniciar una sessió interactiva dins el contenidor; és necessari indicar-lo per a poder treballar en un terminal) i llavors editar-hi des d'allà el fitxer "opensearch.yml" directament fent servir la comanda *vi*

*) Sobreescriure el fitxer "opensearch.yml" intern del contenidor amb un altre fitxer, el qual estarà ubicat externament (és a dir, a la màquina amfitriona). Això es fa tenint per exemple un arxiu mínim com el següent (l'anomenarem $HOME/my-os.yml) i llavors executar el contenidor com és habitual però afegin el paràmetre *-v*, que farà el "mapeig" entre l'arxiu exterior i l'interior, així: *podman container run -d --name=servOS -p 9200:9200 -p 9600:9600* ***-v $HOME/my-os.yml:/usr/share/opensearch/config/opensearch.yml*** *opensearchproject/opensearch:latest*

```
cluster.name: "docker-cluster"
discovery.type: single-node
network.host: 0.0.0.0              #Línia opcional
plugins.security.disabled: true  #Línia opcional. Serveix per inhabilitar TLS i l'autenticació
```

NOTA: Una altra possibilitat que dóna el paràmetre *-v* de podman és muntar (i per tant, superposar) una carpeta qualsevol de l'amfitrió (per exemple, "$HOME/data") sobre la carpeta interna "**/usr/share/opensearch/data**". Aquesta carpeta conté físicament tots els índexs gestionats pel servidor OpenSearch (és a dir, les dades). D'aquesta manera, aquesta informació es trobaria més fàcilment accessible (sense ni tan sol haver d'encendre el contenidor). En tot cas, però, per a què aquest muntatge funcioni, per una qüestió de permisos haurem d'executar primer les comandes *chmod a+w -R $HOME/data* i, en el cas de tenir SELinux en mode "enforcing", *sudo setenforce permissive*

NOTA: En tot cas, si volem que aquests canvis siguin permanents per qualsevol contenidor que vulguem posar en marxa en un futur, el més còmode i pràctic és regenerar una nova imatge amb els canvis desitjats incorporats i llavors els contenidors arrencats a partir d'ella ja els tindran incorporats "de "sèrie". Per crear una imatge personalitzada (a partir de la imatge oficial d'OpenSearch) amb els canvis desitjats (per exemple, sense el "plugin" de seguretat que afegeix TLS i autenticació) caldria fer els següents passos:

1) Crear un arxiu anomenat obligatòriament "**Containerfile**" amb un contingut similar a aquest (on la línia FROM indica la imatge base de la qual es parteix, la línia RUN serveix per indicar les comandes internes del contenidor que s'hi vol executar per tal de fer les modificacions desitjades -en aquest exemple, la comanda indicada és la que elimina el "plugin" de seguretat, que bàsicament consisteix en eliminar la carpeta "/usr/share/opensearch/plugins/opensearch-security" de dins del contenidor i també esborrar totes les directives "plugins.security.*" de dins l'arxiu de configuració) i la línia COPY copia els eventuals fitxers de l'exterior que volguem tenir a l'interior -posant en aquest cas com a propietari de la còpia l'usuari i grup indicat en lloc de "root"-):

```
FROM opensearchproject/opensearch:latest
RUN /usr/share/opensearch/bin/opensearch-plugin remove opensearch-security
COPY –chown=opensearch:opensearch my-os.yml /usr/share/opensearch/config
```

2) Executa la comanda següent des de la mateixa carpeta on es troba l'arxiu "Containerfile" (on també s'hi hauria d'ubicar l'arxiu "my-os.yml" allà indicat): *podman build --tag=nomImatgeNova* **.** (atenció al punt; és important). Aquesta comanda generarà una imatge amb el nom indicat que la podrem fer servir llavors en les comandes *podman container run* o *podman container create* com a nova base.

Comprovació del funcionament del servidor OpenSearch

Un cop posat en marxa el servidor OpenSearch (ja sigui mitjançant l'arranc del contenidor/pod pertinent, o bé, si s'ha fet una instal·lació sobre el sistema amfitrió, mitjançant les clàssiques comandes *sudo systemctl start opensearch && sudo systemctl enable opensearch*), aquest ja podrà començar a indexar (és a dir, a emmagatzemar) la informació que rebi de l'exterior (provinent de peticions d'inserció/actualització degudament formatejades) i també a respondre les eventuals consultes que pugui rebre (també de l'exterior) aportant la informació requerida en cadascuna d'elles. Totes aquestes accions, tal com de seguida veurem, es realitzaran mitjançant una API de tipus REST (és a dir, emprant el protocol HTTP/S).

En concret, per comprovar que s´hagi posat en marxa correctament el nostre servidor OpenSearch, podem enviar una simple petició de tipus GET a https://ip.serv.OpenSearch:9200 (bé mitjançant el navegador o bé mitjançant *curl*) per veure que, efectivament, OpenSearch respon. És a dir, per exemple: si executem la següent comanda hauríem de veure metadades informatives sobre el propi servidor OpenSearch en forma d'objecte JSON (com ara el nom del node, el nom del clúster al que pertany o la versió del software):

curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200

NOTA: El paràmetre *-k* serveix per fer que *curl* no refusi el certificat autosignat que per defecte es genera (gràcies a l'execució automàtica de l'script intern "install_demo_configuration.sh") en arrencar per primera vegada el contenidor-servidor d'OpenSearch. Cal saber que, efectivament, en el cas de què una petició HTTPS es realitzi contra un servidor amb certificat autosignat, per defecte *curl* refusa connectar-hi degut a què no hi pot confiar. Aquest paràmetre *-k* és, doncs, per a què *curl* no realitzi aquesta protecció i, per tant, faci la petició (seria equivalent a haver pitjat el botó d'"Establir excepció" quan passa la mateixa circumstància en un navegador). Òbviament, el més adient seria que el nostre servidor OpenSearch tingués un certificat signat per alguna autoritat de certificació que fos validada pels clients (i això no seria difícil de fer), però ho deixarem per més endavant; de moment usarem el certificat autosignat que per conveniència ja se'ns ha generat sol.

NOTA: El paràmetre *-u* serveix per indicar l'usuari i la contrasenya a utilitzar per autenticar-se contra el servidor OpenSearch (fent-se servir per això el propi protocol HTTP internament). Tal com hem dit, per defecte només es troba configurat l'usuari "admin" amb contrasenya "admin", el qual té permisos absoluts d'administració sobre el propi clúster OpenSearch i la informació que hi allotja. Una de les primeres accions que hauríem de fer hauria de ser, doncs, crear altres usuaris amb privilegis restringits per l'ús diari del nostre servidor, però això ja ho veurem més endavant.

Altres consultes que podem fer per obtenir més dades sobre el funcionament intern del nostre servidor OpenSearch (i així, de pas, confirmar que funciona perfectament), són...:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**_cat/nodes***

NOTA: A totes les consultes de tipus "/_cat/elquesigui" al final es pot afegir el paràmetre "?v" (es a dir, així, per exemple: *curl http://127.0.0.1:9200/_cat/nodes?v)* per establir el "mode verbós". A la pràctica establir el mode verbós vol dir simplement que es mostraran els títols de cada columna mostrada.

...la qual ens mostra l'adreça IP de cada node present dins del cluster (si executem OpenSearch dins d'un "pod" apareixerà una IP privada de classe A perquè a tots els contenidors d'un "pod" executat com un usuari estàndard per defecte se'ls assigna una d'aquesta xarxa), a més d'altres dades d'interès com l'ús de CPU i de la RAM, la càrrega de treball (1m, 5m i 15m), així com el rol de cada node dins del cluster i el seu nom. També és interessant la comanda següent...:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**_cat/plugins***

...la qual ens mostra la llista de "plugins" incorporats a una instal·lació del cluster OpenSearch en qüestió (en el cas de posar-lo en marxa via contenidors, ja venen de sèrie tots els "plugins" oficials). Els "plugins" aporten cadascun una funcionalitat "extra" a la instal·lació base de l'OpenSearch; per exemple, el "plugin" "Security", aporta TLS i autenticació d'usuaris, el "plugin" "Alerting" proporciona la capacitat de generar alertes (consistent en l'enviament de missatges) segons si es detecta determinada condició llindar prèviament definida, etc. Respecte els "plugins" més interessants en parlarem més endavant. També és molt útil la comanda següent:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**_cat/indices***

...la qual ens mostra la llista dels índex actualment emmagatzemats dins el servidor OpenSearch en qüestió. És interessant fixar-se, dins d'aquesta llista, en la columna "docs.count", que indica la quantitat de documents que té cada índex, a més d'altres com "docs.deleted", "store.size", etc. D'altra banda, cal saber que els índexs mostrats amb un punt al principi del seu nom es consideren índexs "oculs", els quals contenen metadades que inclouen informació de configuració sobre el propi servidor OpenSearch (o Dashboards) i que, en principi, no manipularem. Finalment, fer notar que el valor "green" o "yellow" simplement indica si aquest índex està recolzat per un "clúster" o no (respectivament).

> NOTA: Una altra consulta que es pot fer és *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/_cat*, la cual ens ofereix una llista d'altres categories d'objectes que podem explorar, com "count", "health", "aliases", etc. Es poden consultar totes les categories d'objectes disponibles a consultar a https://opensearch.org/docs/latest/opensearch/rest-api/cat/index

Creación de índices vacíos y definición de su estructura (via peticiones PUT)

El proceso de agregar (añadir, insertar...) datos a OpenSearch se denomina "indexación" porque cuando se ingresan datos en OpenSearch, estos se incluyen en forma de documentos JSON dentro de una agrupación de documentos llamada "índice". Es decir, un índice se puede definir como un conjunto de objetos JSON (los cuales generalment se denominan, como hemos dicho, "documentos" y son, en cierto sentido, asimilables a las "filas" o "registros" de una tabla en una BD) que se almacenan bajo un nombre común (y que, por tanto, se pueden gestionar de forma global, como las tablas).

> NOTA: Apache Lucene (https://lucene.apache.org/core) es el motor de indexación en el que se base OpenSearch para almacenar y recuperar los datos.

Los documentos pertenecientes a un determinado índice a menudo tienen una estructura definida por un determinado "mapeo" que indica el tipo de datos de cada elemento ("columna") del documento, aunque no tiene por qué: los índices por definición no tienen estructura rígida. Así pues, cada documento podría disponer de diferentes elementos (es decir, parejas clave <->valor) independientemente de los demás documentos almacenados en el índice.

En un índice OpenSearch, cada documento JSON se suele corresponder a un evento recibido del exterior (ya sea desde FluentBit o cualquier otro origen). En este sentido, OpenSearch ofrece una REST API para directamente crear índices y manipular, añadir o borrar documentos en ellos mediante peticiones de tipo PUT, POST o DELETE, respectivamente (y también, obviamente, para consultar datos, en este caso mediante peticiones de tipo GET).

> NOTA: La REST API de OpenSearch es muy extensa y no estudiaremos más que una mínima parte. En https://opensearch.org/docs/latest/opensearch/rest-api/index se encuentra el punto de entrada a su documentación completa. De todas formas, los apartados que más nos interesarán para empezar son:
> * https://opensearch.org/docs/latest/opensearch/rest-api/index-apis/index (para la gestión de índices: creación, borrado, etc)
> * https://opensearch.org/docs/latest/opensearch/rest-api/document-apis/index (" la gestión de documentos: inserción, " , ")
> * https://opensearch.org/docs/latest/opensearch/rest-api/search (para la búsqueda masiva de datos en documentos)
> * https://opensearch.org/docs/latest/security-plugin/access-control/api (propia del "plugin" Security)

Por ejemplo, para crear un índice nuevo bastará con realizar la siguiente orden:

curl ***-X PUT*** *-k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpSeh:9200/****nombre-indice***

> NOTA: Si queremos que la respuesta obtenida por pantalla (en forma de objeto JSON indicando el éxito o error de la operación) se muestre de forma más legible, podemos optar por incluir el parámetro "?pretty" en la "querystring", así: *curl -X PUT http://127.0.0.1:9200/nombre-indice?pretty*

La orden anterior realmente no es necesaria ya que por defecto ElasticSearch crea automáticamente un nuevo índice en el momento de añadir un primer documento a un (en ese momento) índice inexistente. Y no solamente eso, sino que ese primer documento definirá la estructura interna del índice creado (lo que se llama un "mapping"): es decir, qué número, nombre y tipo de campos tendrán a partir de entonces todos los documentos que se vayan indexando allí a partir de entonces. En principio, para establecer el "mapping" del índice por defecto ElasticSearch interpreta todos los campos del primer documento recibido como cadenas excepto los que tengan un valor evidentemente numérico o de fecha.

NOTA: El hecho de que el primer documento indexado defina el "mapping" del índice **no impide** que posteriormente se puedan indexar documentos con diferente estructura (esto es, con más o menos campos, con valores de otros tipos, etc). No obstante, no es recomendable por una simple cuestión de claridad y orden.

La orden anterior se puede aprovechar para indicar también el "mapping" que se desea asociar al índice justo en el momento de crearlo (es decir, de forma predeterminada, antes de que le llegue ni siquiera el primer documento a agregar). Para ello debemos incluir dicho "mapping" como datos a enviar en el cuerpo de la petición PUT en forma de objeto JSON cuyo elemento más relevante es "*properties*" (tal como se puede ver en https://opensearch.org/docs/latest/opensearch/rest-api/index-apis/create-index/#mappings). Así pues, si queremos tener, por ejemplo, un índice lleno de documentos formados por un elemento de tipo entero, otro de tipo cadena, otro booleano y otro de fecha, deberíamos hacer esto para crearlo con la estructura necesaria:

```
curl -X PUT -k  -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/nombre-indice
-H "Content-Type:application/json" -d  '{
        "mappings": {
                "properties" : {
                        "nomCampSencer": { "type": "integer" },
                        "nomCampCadena": { "type": "text" },
                        "nomCampBoolea": { "type": "boolean" },
                        "nomCampData": { "type": "date", "format": "yyyy-MM-dd HH:mm:ss" }
                }
        }
}'
```

NOTA: La comanda anterior es pot fer servir també per actualitzar un "mapping" prèviament aplicat a un índex ja existent. De fet, si es vol actualitzar un "mapping" a tots els índexs existents de cop, es pot indicar la paraula "_all" així: *curl -X PUT https://127.0.0.1:9200/_all*

Los tipos de datos más comunes que se pueden indicar en el elemento "type" son (la lista entera está en https://opensearch.org/docs/latest/opensearch/rest-api/index-apis/create-index/#dynamic-mapping-types):

"**text**": Tipus cadena desestructurada (el missatge d'un email, la descripció d'un producte, etc). El valor dels camps d'aquest tipus es converteixen a una llista de paraules individuals abans de ser indexats: això permet a l'OpenSearch buscar paraules concretes dins de cada cadena completa
"**keyword**": Tipus de cadena estructurada (telèfons, codis postals, direccions d'emails, codis d'estat, hostnames, etc). Els camps d'aquest tipus serveixen habitualment per filtrar, per ordenar o fer agregacions a les consultes i només es poden buscar pel seu valor exacte.
"**date**" : Tipus data. Els formats admesos per aquest tipus de dades (un dels quals caldrà especificar amb l'opció "format"; si no, s'entendrà que el valor està en el format "strict_date_optional_time") es troben a https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-date-format.html
Internament s'emmagatzema com un número sencer representant els milisegons des del 1-1-1970.
"**long**": Tipus sencer amb signe (de 64 bits: valor mínim de -2^{63} i valor màxim de 2^{63} -1)
"**integer**": Tipus sencer amb signe (de 32 bits: valor mínim de -2^{31} i valor màxim de 2^{31} -1)
"**short**": Tipus sencer amb signe (de 16 bits: valor mínim de -2^{15} i valor màxim de 2^{15} -1)
"**byte**": Tipus sencer amb signe (de 8 bits: valor mínim de -2^{7} i valor màxim de 2^{7} -1)
"**double**": Tipus decimal IEEE-754 de 64 bits
"**float**": Tipus decimal IEEE-754 de 32 bits
"**boolean**" : Tipus booleà. Pot tenir els valors *true* o *false* o bé les cadenes "true" o "false"
"**binary**" : Tipus binari que accepta com a valor una cadena codificada en Base64
"**ip**" : Tipus especial que ha de tenir com a valor una cadena amb format IPv4 ("x.x.x.x/mask") o IPv6

Para ver el "mapping" actual de un determinado índice se puede hacer la siguiente consulta GET (veremos más detalladamente la sintaxis de este tipo de consultas en un próximo apartado)...:

```
curl -k  -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpSeh:9200/nom-index?pretty
```

...o más específicamente (porque la consulta anterior en realidad obtiene todos los metadatos del índice indicado, más allá de solamente su "mapping"), esta otra consulta más concreta:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpSeh:9200/**nom-index/_mapping?pretty***

NOTA: En general, si es vol introduir un document en un índex que contingui algun camp amb un valor que no sigui del mateix tipus que l'indicat al "mapping" existent, es produirà un error. No obstant, en principi no hi hauria d'haver problema per introduir documents que tinguin camps no existents prèviament al "mapping" (o pel contrari, documents que no en tinguin tots els camps indicats al "mapping").

Indexación i actualización de documentos indexados (via peticiones PUT/POST)

Para añadir en un determinado índice un nuevo registro/documento con un determinado identificador (a elegir como se desee), se deberá elegir realizar una petición PUT similar a alguna de las siguientes (donde, tal como se ve, el cuerpo del documento a indexar (más allá de los metadatos que OpenSearch pueda incorporar) se corresponde con los datos JSON indicados en el parámetro *-d* del comando *curl*):

NOTA: Recordar que si, al querer indexar un documento no existiera el índice deseado, por defecto ElasticSearch lo creará automáticamente y asumirá como "mapping" la estructura de dicho primer documento.

*Si queremos que, en el caso de que ya exista un registro/documento con el mismo identificador, lo actualice por completo:

*curl **-X PUT** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_doc/idElegido** -H "Content-Type:application/json" -d '{ "campo1":"valor1", "campo2":12345, "campo3":"2022-01-25 12:34:56" }'*

*Si queremos que, en el caso de que ya exista un registro/documento con el mismo identificador, la operación dé un error:

*curl **-X PUT** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_create/idElegido** -H "Content-Type:application/json" -d '{ "campo1":"valor1", "campo2":12345, "campo3":"2022-01-25 12:34:56" }'*

NOTA: El objeto JSON recibido como respuesta contiene información sobre la operación de indexación, como por ejemplo si fue exitosa (campo "result", que puede valer "created" o "updated" o también el campo "successful", cuyo valor indica el número de documentos creados/actualizados), el identificador del documento indexado (campo "_id", que debería de coincidir con el identificador indicado por nosotros), etc.

NOTA: The "_version" field can be used to track how many times a document has been indexed. It's primary purpose however is to allow for optimistic concurrency control as we can supply a version in indexing requests as well and OpenSearch will then only overwrite the document if the supplied version is higher than what's in the index.

Otra manera tanto de crear como de actualizar un registro/documento es mediante peticiones POST. La diferencia más importante entre una petición PUT y una POST es que mediante la segunda no se ha de indicar ningún identificador sino que se delega en OpenSearch para que nos lo asigne él automáticamente. Lo que conlleva, además, que no sea necesario distinguir entre "_doc" y "_create" (se usará siempre el primero) porque siempre se creará un documento nuevo con un identificador aleatorio y diferente. Así pues, la sentencia a usar podría ser similar a:

*curl **-X POST** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_doc** -H "Content-Type:application/json" -d '{ "campo1":"valor1", "campo2":12345, "campo3":"2022-01-25 12:34:56" }'*

Pero donde POST brilla más es a la hora de modificar un registro/documento ya existente, ya que es mucho más fino que PUT al permitir la edición, adición o borrado de campos individuales. Por ejemplo, para modificar el valor del "campo1" del registro/documento generado con la sentencia PUT de los ejemplos anteriores para que ahora valga "nuevovalor1" deberíamos ejecutar:

*curl **-X POST** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_update/idElegido** -H "Content-Type:application/json" -d '{ "**doc**" : { "campo1": "nuevovalor1" } }'*

De la misma manera se podría añadir una nueva pareja clave<->valor no existente previamente...:

curl ***-X POST*** *-k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_update/idElegido** -H "Content-Type:application/json" -d '{ "**doc**" : { "unnuevocampo": "lerele" } }'*

En el caso de querer eliminar una pareja clave<->valor concreta de un determinado documento, hay que recurrir entonces a la posibilidad que ofrece OpenSearch de ejecutar internamente "scripts" escritos en un lenguaje propio llamado "Painless", el cual permite la manipulación al detalle de cualquier aspecto relacionado con la gestión de los índices y los documentos. En particular, de esta manera:

curl ***-X POST*** *-k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_update/idElegido** -H "Content-Type:application/json" -d '{ "**script**" : { "**source**" : "ctx._source.remove(\"nombrecampoaeliminar\")" } }'*

NOTA: Igualment, es podria haver fet servir el llenguatge "Painless" per actualizar el valor d'un camp existent o per afegir-ne un camp nou (així, respectivament: *curl -X POST -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/nombre-indice/_update/idElegido -H "Content-Type:application/json" -d '{ "**script**" : { "**source**": "**ctx._source.campo1 = \"nuevovalor1\"**" } }'* i *curl -X POST -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/nombre-indice/_update/idElegido -H "Content-Type:application/json" -d '{ "**script**" : { "**source**": "**ctx._source.nuevoCampo = \"valorNuevoCampo\"**" } }'*), entre moltes més possibilitats. Consulta https://www.elastic.co/guide/en/elasticsearch/painless/current/index.html per saber més sobre aquest llenguatge (molt semblant a Java, que és el llenguatge amb el qual està desenvolupat el propi OpenSearch)

Consultas de documentos (vía peticiones GET)

Recordemos que para ver la lista de los índices actualmente almacenados en OpenSearch se puede ejecutar el comando ya conocido *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/_cat/indices* . A partir de aquí, para ver todo el contenido almacenado dentro de un determinado índice (junto con ciertos metadatos de la propia consulta) se puede ejecutar el siguiente comando:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpSeh:9200/**nombre-indice/_search?pretty***

NOTA: Se puede usar el comodín * en el nombre del índice para buscar en más de uno o incluso escribir el nombre especial "***_all***" para indicar todos los índices existentes. También se pueden indicar varios nombres de índices concretos, separados por comas, para buscar en ellos en concreto (así: https://ip.serv.OpenSearch:9200/nombre-indice1,nombre-indice2/_search)

De todo el objeto JSON obtenido de la consulta anterior, lo que más nos interesará habitualmente es el elemento "**took**", que indica el tiempo en milisegundos que tardó la búsqueda, el elemento "**timed_out**", que indica si se sobrepasó el timeout o no, el objeto "**hits**", el cual tiene como elementos la propiedad "**total**" indicando el número total de registros/documentos recuperados -no necesariamente en orden- y, sobre todo, el array "**hits**", que contiene cada uno de dichos registros/documentos, los cuales son asimismo objetos que contienen, entre otros, los siguientes elementos importantes:

*El campo "**_id**" : Representa el identificador único de cada documento.
*El campo "**_index**" : Representa el nombre del índice al que pertenece el documento en cuestión.
*El campo "**_type**" : Mantenido por razones históricas. Siempre valdrá "_doc"
*El subobjeto "**_source**": Contiene el documento original (proveniente, por ejemplo, de FluentBit)

Si sólo quisiéramos obtener (dentro del objeto "hits") un determinado registro/documento, se puede realizar una búsqueda por su identificador, así:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpSeh:9200/**nombre-indice/_search?pretty&q=_id:idElegido**"*

NOTA: Fijarse en el uso de las comillas para no confundir al terminal pot la introducción en la URL del símbolo &

Siguiendo esta sintaxis, de hecho, se pueden realizar búsquedas por cualquier campo que no sea necesariamente el campo identificador (pudiéndose obtener así más de un registro/documento dentro del objeto "hits"), asÍ:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSeah:9200/**nombre-indice/_search?q=nomCamp:valor**"*

NOTA: En verdad, el valor del parámetro *q* puede ser tan solo una palabra (***q=palabra***), en cuyo caso la búsqueda se realizará por todos los campos existentes. También se admiten comodines (* y ?), rangos y, en general, todos los elementos reconocidos para consultas en índices **Lucene** ofrece (ver http://www.lucenetutorial.com/lucene-query-syntax.html para más ejemplos). De todas formas, OpenSearch proporciona una manera de realizar búsquedas más allá de la sintaxis basada en Lucene (incluyendo operadores lógicos, expresiones regulares, expresiones "fuzziness"...) en forma de idioma propio ("**Query DSL**"), el cual tiene sus propias reglas sintácticas que permiten realizar al detalle cualquier tipo de búsqueda avanzada (documentado aquí: https://opensearch.org/docs/latest/opensearch/query-dsl/index) . Pero estudiar esto más en profundidad nos llevaría demasiado fuera de nuestros objetivos.

Hay más opciones además de "*q*" que podemos incluir en la URL (combinándolos entre sí si hiciera falta) para modificar el comportamiento de la búsqueda, como por ejemplo (la lista completa se encuentra en https://opensearch.org/docs/latest/opensearch/rest-api/search):

* El parámetro "size" para limitar la cantidad de documentos obtenidos. Por defecto es 10.
* El parámetro "sort" para indicar el nombre del campo a usar para ordenar el resultado
* El parámetro "from" para definir un offset
* El parámetro "_source_includes" para indicar la lista de nombres (separados por comas y con posibilidad de usar comodines) de los campos pertenecientes al subobjeto "_source" que se quieren obtener en la consulta y ninguno más -esto es útil para ahorrar ancho de banda durante su obtención-
* El parámetro "_source_excludes" para indicar la lista de nombres de los campos pertenecientes al subobjeto "_source" que NO se quieren obtener en la consulta, etc).

Así pues, un ejemplo de consulta sería, por ejemplo, este:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSearch:9200/**nombre-indice/_search?size=20&_source_includes=message**"*

NOTA: De forma alternativa, es posible adjuntar algunas opciones (no todas!) no como parámetros del "querystring" de la petición GET sino como datos adjuntos en una petición POST, así: *curl **-X POST** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSearch:9200/nombre-indice/_search?pretty -H "Content-Type:application/json" -d '{ "size" : 20 , "_source": ["message"] }'* (la opción *_source* indicada en el comando anterior es similar en significado a *_source_includes* pero esta última no funciona en dicho comando)

Si solo nos interesa obtener información sobre un determinado registro/documento, en vez de realizar una búsqueda por todo el índice podemos realizar una consulta alternativa indicando directamente de forma explícita la URL completa del documento deseado, así...:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_doc/idElegido**?pretty*

...aunque si sólo deseamos ver el contenido de su subobjeto "_source" lo podemos indicar así:

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_source/idElegido**?pretty*

NOTA: Se puede saber de una manera muy rápida si un determinado registro/documento existe observando el código de respuesta HTTP (200 o 404, según si es sí o si es no) a una consulta realizada mediante el método HEAD, así:
curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" -I https://ip.serv.OpenSearch:9200/nombre-indice/_doc/idElegido

Si solamente quisiéramos obtener algún campo concreto de dicho documento concreto, podemos hacer (a la inversa también podríamos haber utilizado el parámetro "_source_excludes"):

*curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSearch:9200/**nombre-indice/_source/idElegido?_source_includes=message**"*

NOTA: El comando anterior se podría haber escrito usando "_doc" en vez de "_source": el resultado hubiera sido el mismo pero en el 1r caso además se añadirían los campos de metadatos de la consulta ("found", "_index", "_id", "_type", etc).

Eliminación de documentos e índices (vía peticiones DELETE)

Para eliminar un determinado registro/documento tan solo hace falta lanzar una consulta DELETE indicando el identificador deseado, tal como se muestra a continuación (l'objecte de resposta contindrà alguns dels "sospitosos habituals" en termes de metadades però si ens volem assegurar, després d'executar la petició DELETE només cal tornar a fer un GET i així podrem verificar que el document s'ha suprimit):

*curl **-X DELETE** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice/_doc/idElegido***

Igualmente, se puede eliminar un índice entero lanzando la consulta...:

*curl **-X DELETE** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**nombre-indice***

...y todos los índices mediante:

*curl **-X DELETE** -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/**_all***

Operaciones masivas de inserción, actualización y borrado de documentos (via la API "Bulk")

OpenSearch ofereix la possibilitat d'indexar, actualitzar o esborrar molts documents en una sola petició tot de cop, mitjançant l'anomenada "Bulk" API. Aquesta manera de procedir, la qual a nivell de rendiment és més òptima i més pràctica, consistiria en el següent:

1) Tenir disponibles (per exemple, en un fitxer) la llista de documents que es voldran indexar/actualitzar/esborrar dels índexs desitjats. Aquesta llista (que representa el contingut del cos de la petició de tipus POST que s'enviarà fent servir l'API "Bulk") ha d'estar formada per un conjunt tan llarg com es necessiti de parells de línies amb el següent format...

Acció i metadades de l'acció
Document afectat per l'acció anterior (línia opcional)
...

Per exemple, el següent fitxer conté cadascuna de les quatre (i úniques) accions permeses en una petició de tipus "bulk" (que són "delete", "index", "create" i "update"), juntament amb les seves metadades associades més habituals i, a la línia següent (sempre que no sigui l'acció "delete", on no apareix cap "línia següent") s'indica el document sobre el qual s'aplica l'acció en qüestió:

```
{ "delete": { "_index": "movies", "_id": "tt2229499" } }
{ "index": { "_index": "movies", "_id": "tt1979320" } }
{ "title": "Rush", "year": 2013 }
{ "create": { "_index": "movies", "_id": "tt1392214" } }
{ "title": "Prisoners", "year": 2013 }
{ "update": { "_index": "movies", "_id": "tt0816711" } }
{ "doc" : { "title": "World War Z" } }
```

*La primera línia elimina un document. Les metadades indicades són l'identificador del document a esborrar (obligatori) i el nom de l'índex on es troba (opcional si s'indica a la URL de la petició POST (tal com de seguida veurem). Si el document indicat no existís, no es retorna cap error sinó el resultat "not_found"

*La segona línia indexa un document. Concretament, crea un nou document si no existeix i si sí, el sobreescriu. Les metadades indicades són l'identificador desitjat del document i el nom de l'índex on es voldrà afegir. Cap d'aquestes metadades és obligatòria, però: si no s'indica cap identificador, OpenSearch en generarà un aleatori de forma automàtica i el nom de l'índex no caldria indicar-lo si s'especifica a la URI de la petició POST (tal com de seguida veurem). A la línia següent (en aquest cas, la tercera) és on es defineix el document concret a indexar.

*La quarta línia indexa un document, també, però en el cas d'existir prèviament, retorna un error (a diferència de l'acció "index" explicada al paràgraf anterior). Les metadades són indicades són les mateixes que al paràgraf anterior i tampoc són obligatòries (en els mateixos termes). Igualment, a la línia següent (en aquest cas, la cinquena) és on es defineix el document concret a indexar.

*La sisena línia actualitza part o tot un document ja existent (retornant error si no existeix). Les metadades són l'identificador del document a modificar (obligatori) i el nom de l'índex on es troba (opcional si s'indica a la URL de la petició POST, tal com de seguida veurem). La línia següent (en aquest cas, la setena), es correspon a la modificació de part o de tot el document que es desitja establir, seguint la mateixa sintaxi que vam emprar en veure les peticions POST de tipus "/_update" (és a dir, que aquí podríem fins i tot utilitzar scripts Painless, si fos necessari).

2) Realitza la petició POST corresponent fent servir l'API "Bulk". Si fem servir *curl* i tenim un arxiu anomenat "dades.json" amb un contingut com el descrit al punt anterior, això a la pràctica es traduïria en executar una comanda com la següent:

curl ***-XPOST*** *-k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/****_bulk*** *-H "Content-Type:application/json" --data-binary @dades.json*

NOTA: Es fa servir el paràmetre *--data-binary* en lloc de l'habitual *-d* perquè el primer no elimina els salts de línia ("\n") en una entrada multidocument mentre que el segon sí, i els salts de línia és important mantenir-los per respectar l'estructura de les línies de l'arxiu.

NOTA: Tal com ja s'ha comentat, és possible indicar el nom de l'index sobre el qual s'aplicaran les accions "delete"/"index"/"create"/"update" a la pròpia URL de la petició (si totes aquestes accions afectaran a un únic índex). D'aquesta manera, no serà necessari indicar la metadada "_index" dins del cos de la petició. En aquest cas, doncs, la petició quedaria de la següent manera: *curl -XPOST -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/****nombre-indice/_bulk*** *-H "Content-Type:application/json" --data-binary @dades.json*

EXERCICIS:

1.-a) Arrenca la mateixa màquina virtual utilitzada als exercicis anteriors sobre FluentBit i instal·la-hi (i posa en marxa) un servidor OpenSearch implementat com a contenidor. Comprova, amb la comanda *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200* (executada des de la màquina real) que el servidor OpenSearch respon i, concretament, esbrina la seva versió.

NOTA: En realitat, un exercici més realista seria instal.lar Journalctl/Falco... i FluentBit en una màquina (la "client", d'on es recol.lecten les dades) i OpenSearch i OpenSearch Dashboards en una altra (la "servidora", que fa de magatzem) però degut a la gran quantitat de memòria RAM que necessita OpenSearch, per conveniència i comoditat ho farem tot en una sola màquina virtual

b) Sempre des de la màquina real, crea amb *curl* (concretament, amb una petició PUT) un nou registre/document amb identificador "Manolo" dins d'un índex anomenat "persones" (en no existir aquest índex, es crearà automàticament amb un "mapping" associat "ad-hoc") que estigui format per dos camps JSON: un anomenat "cognom" amb el valor de cadena "Pérez" i un altre anomenat "edat" amb el valor numèric 27

> NOTA: Els noms dels índex no poden incloure lletres majúscules. Més en general, les restriccions es troben llistades a https://opensearch.org/docs/latest/opensearch/index-data/#naming-restrictions-for-indices

c) Comprova amb *curl* la llista d'índexs existents per tal de comprovar que l'índex anterior s'ha creat bé i que conté un document.

d) Realitza una consulta amb *curl* que mostri només el contingut del subobjecte "_source" corresponent al registre/document acabat de crear. ¿Com faries per només mostrar només l'element "edat"?

e) ¿Què mostra la comanda *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/persones/_mapping* ?

f) Afegeix amb *curl* un nou registre/document a l'index "persones" (anomenat "Pepe", per exemple) però ara amb els camps "cognom", "edat" (i un nou camp anomenat "tlf"!) amb els valors que tu vulguis. Observa com, tot i no tenir la mateixa estructura els documents "Manolo" i "Pepe", poden coexistir dins el mateix índex. ¿Per què creus que passa això?

g) Intenta ara afegir un altre document a l'índex "persones" (anomenat "Maria", per exemple) també amb els tres camps "cognom", "edat" i "tlf" però ara assigna al camp "edat" un valor de cadena en lloc de numèric. ¿Què passa en intentar fer l'indexació? ¿Per què creus que passa això?

h) Fes una recerca amb *curl* de tots els documents dins de l'índex "persones" amb el valor 27 al camp "edat".

hII) Modifica la recerca anterior per visualitzar, de tots el camps existents als documents trobats ("cognom", "edat", "tlf"), només el camp "cognom" de cadascun.

i) A nivell conceptual, ¿per què creus que la sortida de les dues comandes següents és diferent?:

curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" -s https://ip.serv.OpenSearch:9200/persones/_doc/Manolo | jq "."

curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" -s https://ip.serv.OpenSearch:9200/persones/_search?q=_id:Manolo | jq "."

j) Escriu un arxiu anomenat "persones.json" amb el següent contingut...:

```
{ "index": { } }
{ "cognom": "García" , "edat" : 56 , "tlf" : "934565434" }
{ "update": { "_id": "Manolo" } }
{ "doc" : { "cognom": "Gómez" , "tlf" : "934565434" } }
```

...i executa la consulta POST adient (amb *curl*) per tal d'aplicar l'API "Bulk" contra el servidor OpenSearch. ¿Què haurà passat? Fes (i digues quines són) les consultes adients per comprovar-ho.

k) Elimina (amb *curl* de nou) tot l'índex "persones"

En el següent exercici s'introdueix l'ús de FluentBit com recopilador de dades (a partir d'orígens diversos) que s'enviaran, mitjançant l'ús del seu "output" anomenat "opensearch", a un servidor OpenSearch, on s'hi emmagatzemaran en forma d'índex. Això és el que més sovint farem a partir d'ara

2.-a) Crea un fitxer (anomenat "ej2.conf") a la teva carpeta personal de la màquina virtual de treball, amb el següent contingut...:

```
[SERVICE]
  Flush  5
  Daemon off

[INPUT]
  Name tail
  Path trash.log

[OUTPUT]
  Name opensearch
  Host 127.0.0.1
  Port 9200
  Index fluentbit
  HTTP_User admin
  HTTP_Passwd Un4C0ntraSs3ny4C0mplicad4_
  Tls on
  Tls.verify off
  Suppress_Type_Name On  <-Aquesta línia cal afegir-la per a què FluentBit no retorni errors de parsejament
```

... i executa la comanda *fluent-bit -c ej2.conf* (confirma que no tinguis el servei Fluentbit en marxa)

b) Obre un altre terminal de la màquina virtual (el tercer) i executa-hi la comanda *echo $RANDOM >> trash.log* varis cops. Això provocarà que FluentBit detecti entrades que envïarà al servidor OpenSearch

NOTA: Hem fet servir el redireccionador ">>" i no ">" per tal de simular l'addició de logs al final del fitxer monitoritzat

c) Atura la comanda FluentBit posada en marxa al primer apartat i tot seguit observa amb la comanda *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/_cat/indices* (la pots executar des de la màquina real) la llista d'índexs existents per tal de comprovar que s'ha creat un nou índex anomenat "*fluentbit*" amb tants registres/documents com cops hagis executat la comanda anterior.

d) Executa la consulta següent i raona què veus:

```
curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSearch:9200/fluentbit*/_search?pretty&q=log:*"
```

dII) Executa la consulta següent i raona què veus:

```
curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/fluentbit*/_search?pretty -H "Content-Type:application/json" -d '{ "_source" : ["log"] }'
```

3.-a) Crea ara un fitxer (anomenat "ej3.conf") a la teva carpeta personal de la màquina virtual de treball, amb el següent contingut i dedueix-ne el seu significat:

```
[SERVICE]
  Flush  5
  Daemon off

[INPUT]
  Name mem
  Interval_Sec 2

[FILTER]
  Name   record_modifier
  Match   *
  Remove_key Swap.*
```

[OUTPUT]
Name opensearch
Host 127.0.0.1
Port 9200
Index fluentbit2
HTTP_User admin
*HTTP_Passwd Un4C0ntraSs3ny4C0mplicad4_*
Tls on
Tls.verify off
Suppress_Type_Name On

b) Executa la comanda *fluent-bit -c ej3.conf* i espera uns segons per a què s'envïin al servidor OpenSearch uns quants documents autogenerats pel propi FluentBit i tot seguit atura la comanda FluentBit

c) Observa amb la comanda *curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" https://ip.serv.OpenSearch:9200/_cat/indices* (la pots executar des de la màquina real) la llista d'índexs existents per tal de comprovar que s'ha creat un nou índex anomenat "*fluentbit2*"

d) Executa la consulta següent i raona què veus:

curl -k -u "admin:Un4C0ntraSs3ny4C0mplicad4_" "https://ip.serv.OpenSearch:9200/fluentbit2/_search?pretty&size=3&_source_includes=Mem.*"*