Proyecto de Dockers ELK: Elasticsearch, Logstash y Kibana

Autor: Alberto Bozal Chaves

Aprender de Dockers:

->Documentación Oficial de Dockers

https://docs.docker.com/engine/userguide/intro/

Siguiendo la documentación oficial podes tener una idea general de cómo funcionan los Dockers como crearte tus propias imágenes y como crearte un repositorio de Dockers.

-> Videos Dockers YouTube

https://www.youtube.com/channel/UC0870Uo4lVU-_JEEY5LWu-A/playlists

Este canal de YouTube ha subido varias guías de cómo funcionan los Dockers, desde lo más básico hasta crear tu propias imágenes, volcara imágenes o cualquier tipo de posibilidad de uso de Dockers.

-> Creación de Dockerfiles

La creación de Dockers se basa en poder utilizar otras imágenes de Dockers para completarlas con funciones que tu quieras darles, se usa el comando FROM, al inicio de cada Dockers tienes el comando FROM que será de la imagen que extiende.

Ej.

```
FROM ubuntu:latest
```

Otros comandos que hay:

-RUN

-VOLUME

-ADD/COPY

- ENV

Si se quiere usar Dockers de manera local dentro del proxy se ha de usar estas variables de entorno:

ENV http_proxy 'http://10.110.8.42:8080'

ENV https_proxy 'http://10.110.8.42:8080'

ENV HTTP_PROXY 'http://10.110.8.42:8080'

ENV HTTPS_PROXY 'http://10.110.8.42:8080'

Instalación de Dockers en una EC2:

-> Instalación de Dockers en AWS (documentación oficial)

http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

```
#Instalar Dockers
sudo yum update -y
sudo yum install -y docker
sudo service docker start
```

SERVICIO DOCKER ELK

-> Uso de Dockers oficiales:

Elasticsearch: (https://hub.docker.com/_/elasticsearch/)

El Dockers oficial de Elasticsearch es muy completo, para la mayoría de funciones servirá perfectamente. Si se quiere añadir algún tipo de parámetro de configuración se puede enviar por línea de comandos al ejecutarlo :

Ej.

```
docker run elasticsearh -Des.node.name="Master" -Des.node.master="true"
```

Nosotros lo vamos a configurar dandole acceso por el puerto 9200(http) y 9300(socket), también le vamos a dar un nombre al container para que sea fácil de identificar y usaremos un volumen para poderlo ver de manera externa i poder reutilizar el indexado si hace falta.

Comando para ejecutarlo:

```
docker run --name some-elasticsearch -p 9200:9200 -p 9300:9300 -v /home/ec2-user/AWS/Elasticsearch/ElasticsearchData:/usr/share/elasticsearch/data Elasticsearch -Des.insecure.allow.root=true
```

Únicamente se configure la ruta de Elasticsearch que quieres que se compartan los volúmenes, sería la ruta que te permite accede a la información de los nodos de Elasticsearch.

También se permite cambiar la configuración de Elasticsearch pero a notros nos va bien la predeterminada: (`-v "$PWD/config":/usr/share/elasticsearch/config).`

Kibana: (https://hub.docker.com/_/kibana/)

Docker official de Kibana es perfecto tal cual esta compilado. Lo único que tenemos que hacer es configurar los puertos de http del kibana para que se pueda ver la web y se ha de conectar

con el Elasticsearch. Para conectarlo con Elasticsearch docker tiene un modo que es usando el comando link:

```
docker run --link <nombre del container Elasticsearch>:elasticsearch  kibana
```

Comandos para ejecutarlo:

```
docker run --name some-kibana --link some-elasticsearch:elasticsearch -p 5601:5601 -d kibana
```

El link une el Elasticsearch que necesita kibana con “elasticsearch” que tiene definida la imagen en el interior (some-elasticsearch es el nombre del Docker y “elasticsearch” ese el nombre/puerto que tiene definido Kibana para recibir información por allí). No sé exactamente que como funciona en el interior pero la función de link funciona así.

Y “–p” es el puerto para ver la web; la “-d” únicamente es para que no se quede el terminal con la salida de logs del Docker.

Logstash: (https://hub.docker.com/_/logstash/)

Comando para ejecutarlo:

```
docker run -it --name logstash --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash logstash -f /config-dir/logstash.conf
```

Logstash requiere de mucha más configuración para usarlo. Usaremos la imagen oficial por comodidad pero habremos de configurarlo desde el conf. Primero tendremos que volcar toda la información que quiere recibir el Logstash en una carpeta en este caso: “/home/ec2-user/AWS/logstash” dentro de la carpeta incluiremos la configuración del Logstash “logstash.conf” , también los logs en “/home/ec2-user/AWS/logstash/logs” y la carpeta de Patterns para poder leer lo, al tener una sola entrada al Docker si queremos recibir desde un fichero de logs es la manera más fácil.

La configuración de Logstash(logstash.conf):

-Input

```
input {
stdin {
            type => "stdin-type"
            }
file {
   type => "syslog-ng"
   # Wildcards work, here :)
   path => [ "/config-dir/logs/*.log"]
   start_position => "beginning"
  }
}
```

El input configura la entrada de Logstash. Stdin es la entrada por terminal que se configura con el comando “-it” en la ejecución del Docker. File es para coger desde un fichero de logs que escoges el directorio "/config-dir/logs/*.log" -> “/home/ec2-user/AWS/logstash/logs” que está cargado desde el volumen montado en la carpeta “/home/ec2-user/AWS/logstash/” a "/config-dir/” . La position te permite escoger desde donde leer el archivo, si desde el principio, por tanto leer todos los logs que hay o cuando se añadan nueva líneas “end”.

***Mirar problemas Logstash leer ficheros en Docker.**

-Filters

```
filter {
      grok {

          #Importacion de los patrones REGEX.
           patterns_dir =>"/config-dir/Patterns"

           #Linea de [FIN] con estado OK
           match =>["message", "%{TIMESTAMP:timestamp} %{TIME_ZONE:time_zone}
\[%{GENERIC_USER:Id_Usuario}\] \[%{GENERIC_WORD:Id_Terminal}\]
\[%{GENERIC_WORD:Id_Peticion}\] \[%{MODE:Modo}\] \[%{EVENT_LINE:Tipo_evento}\]
\[%{STATUS:Resultado}\] \[Servicio\: %{GENERIC_WORD:Servicio}\] \[Metodo\:
%{GENERIC_WORD:Metodo}\] \[Tiempo_peticion\:
%{GENERIC_NUMBER:Tiempo_peticion:float} ms\]"]

           #Linea de [INICIO] generica
           match =>["message", "%{TIMESTAMP:timestamp} %{TIME_ZONE:time_zone}
\[%{GENERIC_USER:Id_Usuario}\] \[%{GENERIC_WORD:Id_Terminal}\]
\[%{GENERIC_WORD:Id_Peticion}\] \[%{MODE:Modo}\] \[%{EVENT_LINE:Tipo_evento}\]
\[Servicio\: %{GENERIC_WORD:Servicio}\] \[Metodo:\ %{GENERIC_WORD:Metodo}\]"]

}
      if([Tipo_evento] =~"(FIN)") {
          mutate{
                add_field =>["Parametros", "Sin parametros o pendiente de actualizarlos"]
                add_field =>["Inicio_peticion", "Pendiente de actualizacion"]
          }
      }
      else {
          mutate{
                  add_field =>["Veces_procesada", 0]} }
}
```

Filters es la parte de procesado de logs, donde encuentra cada patrón y le asigna su variable. Cada variable/patrón que detecta es un field, por ejemplo timestamp es una variable de tipo TIMESTAMP con el valor detectado en el mensaje para saber que es una variable de tipo TIMESTAMP se usan los Patterns.

·Patterns

Se te configura una carpeta donde se tendrán todos los Patterns, los Patterns se puede separar en ficheros o se pueden guardar en un solo fichero. La parte importante es ver como se configura cada Pattern un ejemplo seria TIMESTAMP:

```
TIMESTAMP ([0-9]{4}-[0-1][0-9]-[0-5][0-9] [0-2][0-9]:[0-5][0-9]:[0-5][0-9])
```

·Plugins

Otra parte muy importante de Logstash son los Plugins, hay tanto externos como ya internos de Logstash. Como internos tenemos grok y ruby que es para ejecutar código en dichos lenguajes. Como externo un plugin interesante es Aggregate que te permite enviar información entre diferentes trazas. Se puede saber más de Aggregate en optimización de este documento pero dicho plugin no funciona ni en múltiples worker ni varios Logstash leyendo a la vez el mismo archivo.

-Output

```
output {
elasticsearch { hosts => ["172.17.0.2:9200"] }
}
```

El output se introduce la ip en la cual está la entrada de elasticsearch 172.17.0.2:9200 usando el plugin que ya tiene logstash por defecto poniendo:
elasticsearch { hosts =>.

(EXTERNO)Logs

Para probar el Logstash hemos de generar los logs, para ello usamos los archivos jars.

Los comandos son los siguientes:

```
chmod -x /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar

java -jar /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar
peticiones /home/ec2-user/AWS/logs/log_peticiones_exactas/services.properties 25000
```

Se pueden hacer dos tipos de Logs: Jobs y Servicios. Nosotros usamos servicios y usaremos un generador que genera el fichero entero con un número concreto de logs, por cada vez que lazas se generar 40 logs por tanto 25.000->1.000.000 de logs. Si se quiere usar un generador continuo de logs que trabaje en tiempo real se usan los siguientes comandos:

```
nohup java -jar log.jar peticiones services.properties true &
```

Para decir a dónde va el generador de logs se ha de modificar el archivo de log4j "log4j.properties". Para ello cambiamos el lugar donde va dirigido y pones el directorio y el archivo concreto:

```
log4j.appender.rastreator=org.apache.log4j.FileAppender

log4j.appender.rastreator.File=/home/ec2-user/AWS/logstash/logs/rastreator-delta.log

log4j.appender.rastreator.layout=org.apache.log4j.PatternLayout

log4j.appender.rastreator.layout.conversionPattern=%d{yyyy-MM-dd HH:mm:ss Z} %X{username} %X{idTerminal} %X{idPeticion} [%p] %m%n
```

EC2 Servidores Docker

->ELK (Elasticsearch- Logstash-Kibana) misma maquina EC2

```
#Instalar Dockers

sudo yum update -y
sudo yum install -y docker
sudo service docker start

##########
#Elasticsearch-> configurar la ruta de la data

docker run -d --ulimit nofile=98304:98304 --name elasticsearch -v /home/ec2-user/AWS/Elasticsearch/ElasticsearchData:/usr/share/elasticsearch/data elasticsearch -Des.insecure.allow.root=true

#se añade el comando --ulimit nofile=98304:98304 para que sea más eficiente

##########
#KIBANA:

docker run -d --name kibana --link elasticsearch:elasticsearch -p 5601:5601 kibana

##########
#Logstash:-> configurar ruta

#se configura el volumen de la ruta donde tendrás la configuración logstash.conf, los paterns y logs

docker run -it --name logstash --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash logstash -f /config-dir/logstash.conf

##########
#Ya generados o tiempo real: SERVICIOS DOCKER ELK->Uso de Dockers oficiales ->(EXTERNO)Logs, de este documento.
```

->ELASTICSEARCH+KIBANA en una EC2

```
#Instalar Dockers

sudo yum update -y
sudo yum install -y docker
sudo service docker start

docker run -d --ulimit nofile=98304:98304  --name some-elasticsearch -p 9200:9200 -p
9300:9300 -v /home/ec2-
user/AWS/Elasticsearch/ElasticsearchData:/usr/share/elasticsearch/data elasticsearch -
Des.insecure.allow.root=true

docker run -d --name some-kibana --link some-elasticsearch:elasticsearch -p 5601:5601
kibana
```

Es muy importante recordar que hay que abrir los puertos en el inbound de AWS.

->LOGSTASH+Aggregate plugin en una EC2 con (l2.micro)

#Instalar Dockers

sudo yum update -y
sudo yum install -y docker
sudo service docker start

Instalar Github y descargar de nuestra repo

sudo yum install -y git
git clone https://github.com/jcanopui/awsELK
mv /home/ec2-user/awsELK/AWS /home/ec2-user/AWS/
sudo rm -r /home/ec2-user/awsELK/

#SI SE QUIERE USAR AGGREGATE
#crear un build de logstash con el agreggate

docker build -t logstash_agreggate /home/ec2-user/AWS/docker_aggregate/

######
#Generar los ficheros de logs (500k=> son un millón pero el logstash solo coge las trazas de fin procesado)

chmod -x /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar
java -jar /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar peticiones /home/ec2-user/AWS/logs/log_peticiones_exactas/services.properties 25000
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.1 /home/ec2-user/AWS/logstash/logs/rastreator-delta1.log
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.2 /home/ec2-user/AWS/logstash/logs/rastreator-delta2.log
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.3 /home/ec2-user/AWS/logstash/logs/rastreator-delta3.log

#ejecutar el logsatsh con configuración del git con el agreggate, **modificar para cambiar la IP de destino**

docker run -it --name logstash --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash_agreggate logstash -f /config-dir/logstash_aggregate_micro.conf

->LOGSTASH en una EC2 pruebas de carga

#Instalar Dockers

sudo yum update -y
sudo yum install -y docker
sudo service docker start

Instalar Github y descargar de nuestra repo

sudo yum install -y git
git clone https://github.com/jcanopui/awsELK
mv /home/ec2-user/awsELK/AWS /home/ec2-user/AWS/
sudo rm -r /home/ec2-user/awsELK/

######
#Generar los ficheros de logs (500k=> son un millón pero el logstash solo coge las trazas de fin procesado)

chmod -x /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar
java -jar /home/ec2-user/AWS/logs/log_peticiones_exactas/log_peticiones_exactas.jar peticiones /home/ec2-user/AWS/logs/log_peticiones_exactas/services.properties 25000
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.1 /home/ec2-user/AWS/logstash/logs/rastreator-delta1.log
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.2 /home/ec2-user/AWS/logstash/logs/rastreator-delta2.log
mv /home/ec2-user/AWS/logstash/logs/rastreator-delta.log.3 /home/ec2-user/AWS/logstash/logs/rastreator-delta3.log

#ejecutar el logsatsh con configuracio del git, **modificar para cambiar la IP de destino**

docker run -it --name logstash --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash logstash -f /config-dir/logstash.conf

Problemas y optimizaciones

->Elasticsearch

OPTIMIZACIÓN: Quitar limite OPENMAX FILES

Elasticsearch por la manera de trabajar con la indexación mantiene muchos ficheros abiertos, Linux tiene un límite de máximos ficheros que puede abrir un programa a la vez y para optimizarlo únicamente hay que cambiarle el máximo de ficheros al programa. En el uso de Docker se hace añadiendo "--ulimit nofile=98304:98304" por tanto el comando quedaría así:

```
docker run -d --ulimit nofile=98304:98304 --name some-elasticsearch -p 9200:9200 -p 9300:9300 -v /home/ec2-user/AWS/Elasticsearch/ElasticsearchData:/usr/share/elasticsearch/data elasticsearch -Des.insecure.allow.root=true
```

->Logstash

PLUGIN Aggregate

OPTIMIZACIÓN

La idea de usar Aggregate es poder enviar información entre diferentes procesos de trazas y así mantener la información que quizá otra traza necesite. Su funcionamiento es el siguiente:

task_id => "%{Id_Peticion}" -> aquí se introduce la identificación para que puedas separar variables, si un conjunto de trazas tienen una id_peticion determinada se puede usar ese id de petición. Por tanto cada task_id es como si fuera un entorno de trabajo diferente que cada uno guarda sus variables.

code => es donde se pone la parte de código, lo más común es usar map y event que son dos arraymaps para guardar la información, un ejemplo seria esto:

```
code => "
                event['Inicio_peticion'] = map['TIMESTAMP_ENVIADA'];
                map['parseTime3'] ||=0;
                map['parseTime3']+=event['parseTime'];
                map['parseTime3']-=map['parseTime'];
                event['Tiempo_peticion_real']=map['parseTime3'];
                "
```

Event es el map que recibe Aggregate desde fuera, el map del procesado de logs, map en cambio son las variables que se quedaran dentro de la task que serán eliminados al final.

Se recomienda al final de la task que ya se ha procesado todo usar un endtask:

end_of_task => true

También se puede añadir un timeout y códigos más complejos. Hay un par de links en las referencias.

PROBLEMAS

El problema es la no linealidad/causalidad de procesado de trazas de Logstash, cuando hay **varios workers** o cuando se usa el método **multiline** para procesar más rápido, trazas que están antes que otras en el archivo log pueden ser procesadas posteriormente a otra que eran posteriores, hablamos de pocos milisegundos pero en caso de ser dos trazas concatenadas podemos tener problemas. Si las Trazas FIN se procesan antes que las INICIO entonces el plugin Aggregate lanza una excepción porque en el código se llama a un map que no existe-> null.

Multiline

OPTIMIZACIÓN

El uso de multiline te da un procesado más rápido de los logs, entre un 10-20% de aumento de velocidad, su funcionamiento es simple, hay algo que se encarga de separar las trazas sabiendo en con que empezará cada traza.

```
codec => multiline {
    # Grok pattern names are valid! :)
    patterns_dir =>"/config-dir/Patterns"
    pattern => "^%{TIMESTAMP} "
    negate => true
    what => previous
  }
```

PROBLEMAS

El problema que genera el uso de Multiline es la perdida de linealidad/causalidad de procesado de trazas. Al usar multiline las trazas no se procesan por orden i por ejemplo al plugin Aggregate le dan problemas. También para el uso de multiline solo se puede tener un tipo de inicio de trazan si hay dos modelos de trazas que empiezan diferentes ya no se puede usar.

PROBLEMAS: Docker ENV

Logstash guarda en un fichero que ficheros a procesado en que offset se quedó procesando y donde están localizados los archivos. El problema que conlleva es que el fichero se guarda dentro del Docker por tanto no se puede eliminar en caso de que se quieran volver a procesar los archivos, si se intenta cambiar la configuración a través del logsatsh.conf no funcionan los comandos. Presupongo que debe ser que al usar Docker Logstash funciona algo diferente pero no he logrado encontrar la respuesta a este problema.

Crear Dockers propios

Supervisor, múltiples servicios en un mismo Docker

Supervisor es un programa/servicio que se encarga de ejecutar varios comandos al ejecutar el Docker. Es muy configurable todas las posibles configuraciones se pueden encontrar en la configuración oficial: http://supervisord.org/configuration.html

En nuestro caso lo usaremos para crear un Docker con Elasticsearch + SSH para tener una puerta trasera en caso de tener cualquier problema y poder revisar los logs desde fuera con facilidad.

Se podría usar Docker Compose para ejecutar varios servicios, pero es una herramienta preparada para arrancar varios Docker ya preconfigurados y no creo que fuses la opción más viable en este caso.

DOCKER Elasticsearch + SSH

Tener una puerta trasera en un servicio es algo realmente útil, en nuestro caso puede servirnos para cambiar la configuración de Elasticsearch una vez ejecutado el Docker. Para ello usaremos supervisor como hemos presentado anteriormente.

La creación de este Docker personalizado tiene muchas partes diferenciadas:

-Dockerfile

-Configuración del supervisor

-La configuración de Elasticsearch y el uso del docker-entrypoint.sh de Elasticsearch

Dockerfile

En este Dockerfile vamos a tener que configurar varias cosas y todas desde 0 prácticamente, por ello extenderemos del Docker de java: “FROM java:8-jre”. Vamos a instalar el SSH y la última versión de Elasticsearch. Primeramente instalaremos la SSH y supervisor en el Dockerfile:

```
FROM java:8-jre

RUN apt-get update && apt-get install -y openssh-server
RUN apt-get install -y supervisor
RUN mkdir /var/run/sshd
RUN echo 'root:screencast' | chpasswd
RUN sed -i 's/PermitRootLogin without-password/PermitRootLogin yes/'
/etc/ssh/sshd_config
RUN mkdir -p /var/log/supervisor
ADD supervisord.conf /etc/supervisor/conf.d/supervisord.conf

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g'
-i /etc/pam.d/sshd
ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile
```

```
# Install curl
RUN apt-get install -y curl

ENV GOSU_VERSION 1.7
RUN set -x \
	&& wget -O /usr/local/bin/gosu "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-architecture)" \
	&& wget -O /usr/local/bin/gosu.asc "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-architecture).asc" \
	&& export GNUPGHOME="$(mktemp -d)" \
	&& gpg --keyserver ha.pool.sks-keyservers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \
	&& gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu \
	&& rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc \
	&& chmod +x /usr/local/bin/gosu \
	&& gosu nobody true

RUN wget -O - http://packages.elasticsearch.org/GPG-KEY-elasticsearch | apt-key add -
RUN echo "deb http://packages.elasticsearch.org/elasticsearch/2.x/debian stable main" >> /etc/apt/sources.list
RUN apt-get update
RUN apt-get install -y elasticsearch

ENV PATH /usr/share/elasticsearch/bin:$PATH

WORKDIR /usr/share/elasticsearch

RUN set -ex \
	&& for path in \
		./data \
		./logs \
		./config \
		./config/scripts \
	; do \
		mkdir -p "$path"; \
		chown -R elasticsearch:elasticsearch "$path"; \
	done

COPY config ./config

VOLUME /usr/share/elasticsearch/data

COPY docker-entrypoint.sh /

RUN chmod +x /docker-entrypoint.sh

# Prevent elasticsearch calling `ulimit`.
RUN sed -i 's/MAX_OPEN_FILES=/# MAX_OPEN_FILES=/g' /etc/init.d/elasticsearch
```

Ahora solo falta configurar las entradas y salidas del docker:

```
###PORTS###
#SSH
EXPOSE 22
#ELASTICSEARCH
EXPOSE 9200
EXPOSE 9300
#Supervisor
EXPOSE 8888

#ENTRYPOINT ["/docker-entrypoint.sh"]
CMD ["/usr/bin/supervisord"]
#ENTRYPOINT ["/docker-entrypoint.sh"]
#CMD ["elasticsearch"]
```

Tema configuración de Elasticsearch se encuentra en github, sin ella no se podrá compilar el Docker.

Configuración de supervisor

Lo siguiente que hay que configurar es el archivo "supervisord.conf" que es la configuración de supervisor, en nuestro caso solo ejecutamos dos servicios:

```
[supervisord]
nodaemon=true

[inet_http_server]
port = 8888
username = Alberto
password = Nada

[program:sshd]
command=/usr/sbin/sshd -D
stdout_logfile =/var/run/sshd/ssh.log
redirect_stderr=true

[program:elasticsearch]
command=/docker-entrypoint.sh elasticsearch
stdout_logfile=/dev/fd/1
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Para configurar los servicios se usa "[program:nombredelprograma]" después el comando que ejecutarías en el terminal y finalmente la configuración en este caso se diferencian dos tipos de configuración:
·Guardar los logs en un archivo y solo mostrar por termial el stderror: stdout_logfile =/var/run/sshd/ssh.log

·No guardar los logs en un archivo y mostrarlo por termial -> que el comando logs de docker mostrará los logs pero no se pdoran ver los últimos logs desde la web de supervisor.

Inet_http_server es para poder mostrar que tal están los programas que ejecuta el supervisor saber si está funcionando si se ha parado, todo ello usando un panel de control por web.

Configurar Elasticsearch en el Docker

Para configurar Elasticsearch en el Docker se han de configurar varios archivos. Empezaremos con el docker-entrypoint.sh que el script que se encarga de ejecutar Elasticsearch. Este archivo se copia de la web oficial.(También está en el repositorio)

Después quedará la configuración predeterminada de elasticsearch: config/elasticsearch.yml y config/logging.yml.

Elasticsearch.yml:

```
network.host: 0.0.0.0
```

Logging.yml

```
# you can override this using by setting a system property, for example -
Des.logger.level=DEBUG
es.logger.level: INFO
rootLogger: ${es.logger.level}, console
logger:
  # log action execution errors for easier debugging
  action: DEBUG
  # reduce the logging for aws, too much is logged under the default INFO
  com.amazonaws: WARN

appender:
  console:
    type: console
    layout:
      type: consolePattern
      conversionPattern: "[%d{ISO8601}][%-5p][%-25c] %m%n"
```

Una vez hecho todo únicamente habría que hacer la compilación, hay que ir al directorio y ejecutar el comando “docker build -t elasticsearch_ssh .” (el nombre de la imagen le ponemos Elasticsearch_ssh pero se le puede poner cualquier otro nombre).

DOCKER Elasticsearch + Marvel

Para un control más exhaustivo de Elasticsearch instalaremos el plugin marvel. La instalación del plugin se hace sobre Elasticsearch pero para poder interactuar con el usaremos Kibana que también tiene un plugin para eso.

Elasticsearch Marvel plugin

En este caso se instalara sobre una imagen personalizada de Docker la creada anteriormente, que contenía el servicio ssh.

Para ello escribiremos en el Dockerfile de este nuevo Docker lo siguiente:

```
FROM elasticsearch_ssh

RUN /usr/share/elasticsearch/bin/plugin install license
RUN /usr/share/elasticsearch/bin/plugin install marvel-agent
```

Únicamente se ha de ejecutar el instalador de Plugins e instalar una licencia y el marvel-agent. Marvel-agent es un servicio de pago pero se puede usar la versión básica instalándola.

Y solo quedará ejecutar el Docker build: “docker build -t elasticsearch_ssh_marvel .”.

Kibana Marvel plugin

Ahora para añadir el plugin a Kibana solo hay que modifcar la imagen oficial añadiéndole un comando RUN:

```
FROM kibana:latest

RUN kibana plugin --install elasticsearch/marvel/latest \
   && chown kibana:kibana /opt/kibana/optimize/.babelcache.json
```

Ejecutamos Docker build “docker build -t kibana_marvel .” y ya tenemos la imagen generada.

Pruebas de Carga ELK

Preparación EC2- Logstash 2 Instancias, múltiples pruebas

Para las pruebas de carga se necesitan instancias de Logstash. Para ello primeramente se ha de seguir la guía de “EC2 servidores Dockers->LOGSTASH en una EC2” del documento. Una vez se tiene configurada la maquina se tendría que duplicar los archivos de “/home/ec2-user/AWS/logstash/logs/” a dos capetas de ese directorio “logstash1_1kk_n” y “logstash2_1kk_n” donde n final será el número de pruebas de carga, así que recomiendo empezar por 1 “logstash1_1kk_1”, esto es por el problema de ENV de Logstash que al usar dos en una misma maquina no se pisarían i por tanto procesaría las tramas exactas del documento.

También se ha de configurar la configuración de Logstash en el cual crearía dos ficheros de configuración: “logstash1_1kk.conf” y “logstash2_1kk.conf”.

Una vez creados se abrían de editar cada uno para poner que se dirijan a un directorio cada uno:

```
input {
  stdin {
    type => "stdin-type"
  }
file {
    type => "syslog-ng-3"

    # Wildcards work, here :)
    path => [ "/config-dir/logs/logstash1_1kk_1/*.log"]
    start_position => "beginning"
    # sincedb_path => "/dev/null"
  }
}
```

También es importante recordar poner la IP a donde dirigiremos los logs.

```
output {
elasticsearch { hosts => ["ec2-52-203-178-151.compute-1.amazonaws.com:9200"] }
}
```

Preparación EC2- Elasticsearch master y nodos

También es necesario preparar Instancias de Elasticsearch y hay dos tipos: El master que se encargara de gestionar al resto de nodos y los clientes que procesaran la información que el master les diga.

Para la EC2, crear el Docker Elasticsearch con marvel y ssh. Siga la guía de Crear Docker Elasticsearch con marvel y ssh de este documento pero usando una configuración diferente de yml para master y para clientes.

Para configurar el master es necesario usar esta configuración en “Elasticsearch.yml”:

```
network.host: 0.0.0.0
network.publish_host: IP_de_la_EC2
cluster.name: mycluster
node.name: node_master
node.master: true
node.data: true
http.port : 9200
tcp.port : 9300
discovery.zen.ping.multicast.enabled: false

marvel.agent.exporters:

 id1:
  type: http
  host: ["IP_de_la_EC2:9200"]
```

Es muy importante poner la IP del servidor de la ec2 correctamente sino cuando el cliente se quiera conectar al master el master le devolverá una IP errónea i no se podrá conectar.

Para configurar los clientes es necesario usar esta configuración en “Elasticsearch.yml”:

```
network.host: 0.0.0.0
network.publish_host: IP_de_la_EC2
cluster.name: mycluster
node.name: node_${HOSTNAME}
node.master: false
node.data: true
http.port : 9200
tcp.port : 9300
discovery.zen.ping.multicast.enabled: false

discovery.zen.ping.unicast.hosts: ["IP_del_master:9300"]
```

Proceso de ejecución pruebas de carga

Para el proceso de ejecución primeramente se han de arrancar las maquinas, nuestra configuración de pruebas consiste en las EC2 de Elasticsearch en AWS t2.Large y las EC2 de Logstash en t2.medium. Una vez arrancadas, se configura cada uno de los servers.

Recordad que se ha de abrir los inbounds de los servers de Elasticsearch:

-9200-> Elasticsearch http

-9300-> Elasticsearch socket

-2222-> SSH del docker

-8888: configuración de los servicios arrancados en el Docker(te permitirá parar nodos de Elasticsearch para ver el comportamiento de reconstrucción de “shards”.

Una vez este todo abierto y los servidores Elasticsearch ejecutados:

```
docker run -d --ulimit nofile=98304:98304  --name some-elasticsearch_ssh_maven -p 9200:9200 -p 9300:9300 -p 8888:8888 -p 2222:22 -v /home/ec2-user/AWS/Elasticsearch/ElasticsearchData:/usr/share/elasticsearch/data elasticsearch_ssh_marvel
```

*Recordad que si se quiere ver el proceso se ejecuta correctamente se ha de eliminar “-d” de la ejecución. También se puede usar el comando “docker logs elasticsearch_ssh_marvel”.

Ahora para la primera prueba de carga ya debe estar todo configurado en los servidores de Logstash y solo sería necesario ejecutarlos. Para ello recomiendo hacer dos ventanas de terminal por cada servidor, en cada una arrancaremos un Logstash:

```
docker run -it --name logstash1 --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash logstash -f /config-dir/logstash1_1kk.conf
```

```
docker run -it --name logstash2 --rm -v /home/ec2-user/AWS/logstash:/config-dir logstash logstash -f /config-dir/logstash2_1kk.conf
```

Una vez acabe de enviar todas las trazas se han de cerrar los Dockers de Logstash ejecutados: Control+C y esperamos hasta que nos diga que esperemos un poco más y forzamos cierre otra vez con Control+C.

Para volver a ejecutar los Logstash se ha de hacer dos cosas cambiar los nombres de las carpetas de logs y reflejar eso en el archivo de configuración.

“/home/ec2-user/AWS/logstash/logs/logstash1_1kk**_1**” -> “/home/ec2-user/AWS/logstash/logs/logstash1_1kk**_2**”

“/home/ec2-user/AWS/logstash/logs/logstash2_1kk**_1**”-> “/home/ec2-user/AWS/logstash/logs/logstash2_1kk**_2**”

Una vez cambiados los nombres cambiamos la configuración:

logstash2_1kk.conf y logstash1_1kk.conf donde substituimos: “path => ["/config-dir/logs/logstash1_1kk_**1**/*.log"]” por “path => ["/config-dir/logs/logstash1_1kk_**2**/*.log"]” y “path => ["/config-dir/logs/logstash2_1kk_**1**/*.log"]” por “path => ["/config-dir/logs/logstash2_1kk_**2**/*.log"]” y ya se podrían volver a ejecutar los Logstash.

TIPS y posibles problemas

Primeramente me gustaría recordar que los EC2 tiene un límite de tamaño de memoria por tanto se ha de tener cuidado con cuantas trazas se envían al Elasticsearch, un Elasticsearch t2.large tiene un límite superior a 6.000.000 de trazas pero inferior a 8.000.000 por tanto no se

podrán enviar más de 3 servidores de Logstash seguidos(con cada uno 2 instancias de Logstash). Si se quiere hacer una prueba de carga más exigente se puede abrir un solo Logstash por servidor por tanto 6 Instancias de EC2.

Una vez hecha una prueba de carga se recomienda borrar los datos para ello usará el curl XDELETE: “curl -XDELETE 'http://52.203.178.151:9200/_all'”. Se pueden exportar los datos generados con Kibana, una vez estas en Discover, debajo de la gráfica se clica a la flecha que señala hacia arriba, bajamos y seleccionamos “Page size=all” y después lo descargamos en formatted o raw.

En caso de que los Logstash dejen de funcionar porque se ha detenido mal o simplemente ha dejado de funcionar se debería borrar los Docker y todos los ficheros que tenemos en la EC2 relacionado con Dockers:

```
#uninstall Docker

sudo service docker stop
sudo yum erase -y docker
sudo rm -r /var/lib/docker/
```

Resultados pruebas de carga

Antes de empezar con los resultados me gustaría aclarar que la mayoría de resultados son matemáticos debido a que no he podido concluir resultados a partir de los datos aportados por el plugin marvel. Por tanto los resultados son de logs procesados y aproximaciones de dichos resultados.

Todos los datos están reflejados en el Excel.:

Primeramente se puede observar que Elasticsearch tarda un tiempo en ser lo más eficiente posible recibiendo trazas teniendo el primer minuto una velocidad e procesado algo inferior a su máximo. Por ello los primeros test con pocas tramas no tiene una velocidad demasiado alta de procesado de logs.

También podemos ver que la mayor eficiencia enviando log relacionado con Logstash es cuando hay un servidor de Elasticsearch que recibe 2 instancias de Logstash de la misma maquina donde llega a 257.500 logs/min por una sola máquina.

La mayor eficiencia para un solo servidor de Elasticsearch es 2 maquina EC2 Logstash enviando al servidor de Elasticsearch que recibe 303.000 logs/min.

Si sabemos que el máximo por máquina de Logstash es de 275.500 logs/min y usamos dos máquinas el máximo recibido debería ser 515.000 logs/min pero podemos ver que la máquina de Elasticsearch solo puede llegar a procesar 303.000logs/min, por tanto hay alguna parte de la máquina de Elasticsearch que está saturada, comprobado en otros test que se explicarían posteriormente sabemos que la NAT/tarjeta de red no se satura debido a que superará esa cantidad de trazas, pero no se dan evidencias de saturación de memoria ni de CPU, puede que se deba a la configuración del propio Elasticsearch o que sea por la escritura en disco.

Probando Elasticsearch en multi-nodo se puede ver que la eficiencia por maquina se ve reducida muchísimo.

Primeramente comentar que lo más recomendable es tener dos máquinas de la misma potencia de procesado debido a que sino la más lenta hace a la más rápida menos eficiente, como en nuestro caso Master=t2.large y nodo=t2.micro conseguimos una velocidad de procesado entre las dos máquinas de 172.500 logs/min casi la mitad que si solo tuviéramos la maquina Master activa.

Una vez las maquinas son similares cabe recalcar que la eficiencia de procesado baja por cada máquina que añadas. Todo este proceso se hará con dos máquinas EC2 con 2 Dockers de Logstash cada máquina. Con dos large: Master=t2.large y nodo=t2.large conseguimos una velocidad de procesado entre las dos máquinas de 266.500 logs/min que cada máquina procesa aproximadamente 133.250 logs/min. Con 3 large: Master=t2.large , nodo1=t2.large nodo2=t2.large obtenemos entre todas 342.000 logs/min, ya superior a una sola máquina de Elasticsearch, con un procesado aproximado por máquina de 114.000 logs/min. Con 4 large el resultado es de 392.000 logs/min muy superior a una sola EC2 de Elasticsearch.

Si escalamos la cantidad de máquinas de Logstash a 3 EC2 y dos Logstash con cada máquina, podemos observar que seguimos aumentando la eficiencia de procesado. Con 3 EC2 large de Elasticsearch tenemos 374.000 logs/min que es aproximadamente 124.000 logs/min por máquina, si subimos a 4 EC2 de Elasticsearch obtenemos 461.500 logs/min que cada máquina procesa aproximadamente a una velocidad de 115.000 logs/min.

Por tanto como resumen de resultados cuanto mayor es el número de instancias de Elasticsearch menor es la eficiencia de procesado de cada una pero sumando la cantidad de logs procesados entre todas sí que aumenta considerablemente.

COMANDOS UTILES

Dockers

Instalación

```
#Instalar Dockers

sudo yum update -y
sudo yum install -y docker
sudo service docker start
```

Desinstalación

```
#Desinstalar Docker

sudo service docker stop
sudo yum erase -y docker
sudo rm -r /var/lib/docker/
```

Comandos más frecuentes

```
#Mirar los Dockers activos
docker ps -a

#Inspecionar los puertos abierto, variables de ENV etc..

docker  inspect <container or image>

#Mirar logs de un docker

docker  logs <container>

#Crear un docker/Dockerfile

docker build -t <nombre de la imagen a crear> .

#Eliminar container de docker

docker rm -f <container>

#Eliminar imagen de docker

docker rmi -f <image>
```

Elasticsearch

```
#Para poder ver las trazas sin kibana
curl -XGET 'http://172.17.0.2:9200/_search?size=50&pretty=true'

#Para eliminar todas las trazas de elasticsearch
curl -XDELETE 'http://172.17.0.2:9200/_all'

#Para comprobar cuantos nodos hay en un cluster y si están activos, también puede
mirarse con marvel
curl -XGET 'http://172.17.0.2:9200/_cluster/health?pretty=true’
```

REFERENCIAS:

DOCKERFILES:

http://stackoverflow.com/questions/27767264/how-to-dockerize-maven-project-and-how-many-ways-to-accomplish-it

https://kuldeeparya.wordpress.com/page/2/

https://github.com/kuldeeparyadotcom?tab=repositories

PROXYS DOCKER ON WINDOWS:

http://www.netinstructions.com/how-to-install-docker-on-windows-behind-a-proxy/

ELASTICSEARCH DOCUMENTATION:

https://www.elastic.co/guide/en/elasticsearch/guide/current/getting-started.html

EBL elasticsearch:

http://www.ybrikman.com/writing/2015/11/11/running-docker-aws-ground-up/

Examples config multinode elasticsearch:

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-production-elasticsearch-cluster-on-ubuntu-14-04

Configuración ejemplo elasticsearch.yml:

https://gist.github.com/zsprackett/8546403

https://joinup.ec.europa.eu/svn/opencities/tags/opencities-core/release-3.4.3/core-module/src/main/resources/elasticsearch.yml

INFO SOBRE BUGS elasticsearch multinode:

https://discuss.elastic.co/t/node-discovery-in-elasticsearch-on-amazon-ec2/36490/11

Elasticsearch eficiécia no maxfiles open (limit off):

https://github.com/docker-library/elasticsearch/issues/102

Calcular tiempo entre trazas o altres operaciones (Aggregate):

https://github.com/logstash-plugins/logstash-filter-aggregate/issues/15

https://www.elastic.co/guide/en/logstash/current/plugins-filters-aggregate.html

Supervisor en Docker:

https://docs.docker.com/engine/admin/using_supervisord/

Marvel license:

https://www.elastic.co/guide/en/marvel/current/license-management.html