README.md

Parcial 2

Universidad ICESI
Curso: Sistemas Distribuidos
Docente: Daniel Barragán
Tema: Automatización de infraestructura (Docker)
Estudiante: Paula Andrea Bolaños Arias

Objetivos

- Realizar de forma autónoma el aprovisionamiento automático de infraestructura
- Diagnosticar y ejecutar de forma autónoma las acciones necesarias para lograr infraestructuras estables
- Integrar servicios ejecutándose en nodos distintos

Prerrequisitos

- Docker
- docker compose
- docker swam

Descripción

Desplegar un ambiente compuesto por elasticsearch, kibana, fluentd y un servidor web apache o un servicio web de su elección. El servidor ó servicio web debe desplegar al menos 4 replicas y debe ser accedido a través de un punto único de balanceo de carga (consulte el manual de referencia de docker-compose 3 ó superior). Cada una de las replicas del servicio web debe estar limitada a un uso del 10% de cpu y 20Mb de memoria RAM. En el caso de usar el contenedor de servidor web deberá almacenar en elasticsearch los campos del log de apache en forma independiente (consulte el manual de fluentd, parser-plugin).

Actividades

1. Consigne los comandos de linux necesarios para el aprovisionamiento de los servicios solicitados. En este punto no debe incluir archivos tipo Dockerfile solo se requiere que usted identifique los comandos o acciones que debe automatizar

Descarga de la imagen para el contenedor elasticsearch:

```
docker run –d elasticsearch
```

Descarga de la imagen para el contenedor kibana:

```
docker run –d ––link elasticsearch –p 5601:5601 kibana
```

Se debe configurar la imagen del fluentd, agregando un plugin para que se comunique con elasticsearch. Por ello es necesario crear una nueva imagen con un Dockerfile, que debe contener lo siguiente:

```
FROM fluent/fluentd:v0.12–debian
RUN ["gem", "install", "fluent–plugin–elasticsearch", "––no–rdoc", "––no–ri", "––version", "1.9.2"]
```

La imagen se crea así:

```
docker build -t fluentd
docker tag fluentd pauandre27/myfluentd:sd-parcial2
docker push pauandre27/myfluentd:sd-parcial2
```

Posteriormente, se sube el contenedor con la nueva imagen, configurándole el volumen donde se encuentra el fluentd.conf:

```
docker run -p 24224:24224 -link elasticsearch -v /fluentd/conf:/fluentd/etc -e FLUENTD_CONF=fluent.conf
```

Para el servicio web, se va a utilizer una imagen, ya creada, llamada hello-world del repositorio tutum:

```
docker run -d -p 80:80 --log-driver=fluentd -log-opt fluentd-address=localhost:24224 -log-opt tag=fluent
```

Estos servicios se crean dentro de un swarm. Para ello, se deben crear las máquinas que se van a utilizar como manager y worker:

```
docker-machine create --driver virtualbox manager
docker-machine create --driver virtualbox worker
```

Después de haber creado las máquinas, se accede al manager por ssh y se le ingresa el comando con la ip de la máquina:

```
docker swarm init --advertise-addr 192.168.99.100
```

Esto arroja el comando con el token, que se debe ingresar en el worker por ssh, el cual empieza por "docker swarm join ...".

Para tener mejor acceso al manager, se recomienda ingresar el comando:

```
docker-machine env manager
eval $(docker-machine env manager)
```

2. Escriba los archivos Dockerfile necesarios junto con los archivos fuente necesarios. Tenga en cuenta consultar buenas prácticas para la elaboración de archivos Dockerfile.

El Dockerfile se necesita sólo para la creación de la imagen del fluentd con el plugin para la comunicación con elasticsearch y quedaría así:

```
# fluentd/Dockerfile
FROM fluent/fluentd:v0.12-debian
RUN echo "fluentd"
RUN ["gem", "install", "fluent-plugin-elasticsearch", "--no-rdoc", "--no-ri", "--version", "1.9.2"]
RUN rm /fluentd/etc/fluent.conf
COPY ./conf/fluent.conf /fluentd/etc
```

3. Escriba el archivo docker-compose.yml necesario para el despliegue de la infraestructura. Adicione comentarios en el archivo describiendo las configuraciones empleadas. No emplee configuraciones deprecated.

```
version: "3"

services:
```

```
web:
  image: tutum/hello-world
  networks:
    - webnet
  ports: # Puertos a exponer
    - "80:80"
  logging:
    driver: "fluentd" # Logging Driver
    options:
      tag: tutum
  deploy:
    restart_policy: # condiciones para reiniciar
         condition: on-failure
         delay: 20s
         max_attempts: 3
         window: 120s
    mode: replicated # el servicio está replicado por defecto
    replicas: 4 # especifica la cantidad de replicas
    resources:
      limits:
        cpus: "0.1" # % de cpu asignado a cada replica
        memory: 20M # memoria RAM para cada replica
    placement:
      constraints:
        - node.role == worker # nodo en el que se va a desplegar
    update_config:
      delay: 2s

visualizer:
  image: dockersamples/visualizer:stable
  ports:
    - "8080:8080"
  volumes: # ruta del volumen_actual:volumen_final donde debe quedar
    - "/var/run/docker.sock:/var/run/docker.sock"
  deploy:
    restart_policy:
         condition: on-failure
         delay: 20s
         max_attempts: 3
         window: 120s
    mode: replicated
    replicas: 1
    update_config:
      delay: 2s
    placement:
      constraints: [node.role == manager]
  logging: # configuración para el manejo de logs utilizando el fluentd
      driver: "fluentd"
      options:
       tag: visualizer
  networks:
    - webnet

fluentd:
  image: pauandre27/myfluentd:sd-exam2 # imagen creada con plugin
  volumes:
    - "./Logs:/fluentd/log"
  ports:
    - "24224:24224"
    - "24224:24224/udp"
  networks:
    - webnet
  deploy:
```

```
      restart_policy:
           condition: on-failure
           delay: 20s
           max_attempts: 3
           window: 120s
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == manager]
      update_config:
        delay: 2s

  elasticsearch:
    image: elasticsearch
    command: [ elasticsearch, -E, "network.host=_eth1:ipv4_", -E,
discovery.zen.ping.unicast.hosts=tasks.sss_elasticsearch, -E, discovery.zen.minimum_master_nodes=2, -
E, cluster.name=myclustersss ] # especificación de la interfaz eth1 que es la que está en la red
    ports:
      - "9200:9200"
      - "9300:9300"
    networks:
      - webnet
    environment:
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    healthcheck: # para probar funcionamiento del contenedor
        test: ping -c1 localhost >/dev/null 2>&1 || exit 1
        interval: 1m
        timeout: 10s
        retries: 3
    logging:
        driver: "json-file"
        options:
          max-size: 10M
          max-file: 1
    deploy:
      restart_policy:
           condition: on-failure
           delay: 20s
           max_attempts: 3
           window: 120s
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == manager]
      update_config:
        delay: 2s
      resources:
        limits:
          memory: 1000M
    volumes:
      - "./esdata:/usr/share/elasticsearch/data"

  kibana:
    image: kibana
    ports:
      - "5601:5601"
    networks:
      - webnet
    logging:
        driver: "json-file"
        options:
           max-size: 10M
           max-file: 1
```

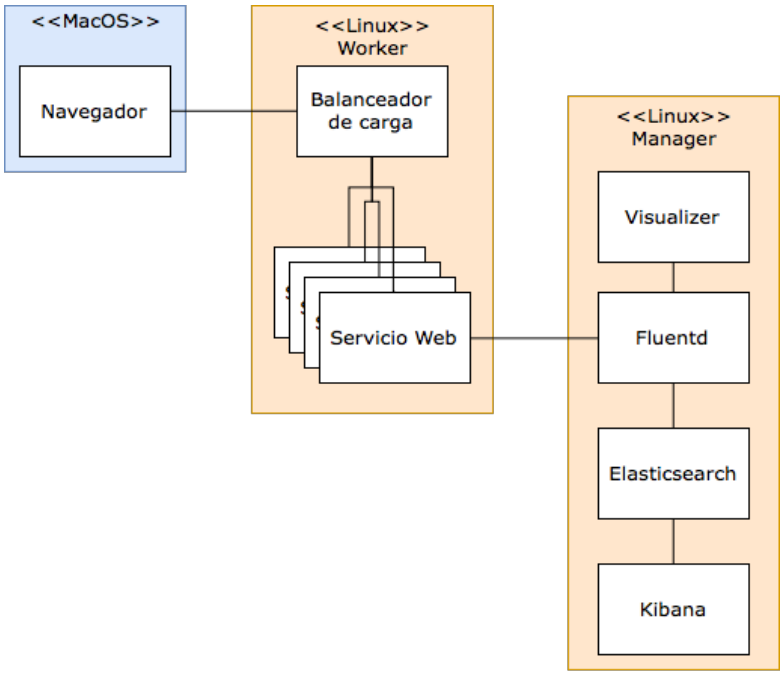
```
    deploy:
      restart_policy:
           condition: on-failure
           delay: 20s
           max_attempts: 3
           window: 120s
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == manager]
      update_config:
        delay: 2s

networks: # red virtual que se debe crear y se ha especificado en cada servicio
  webnet:
```

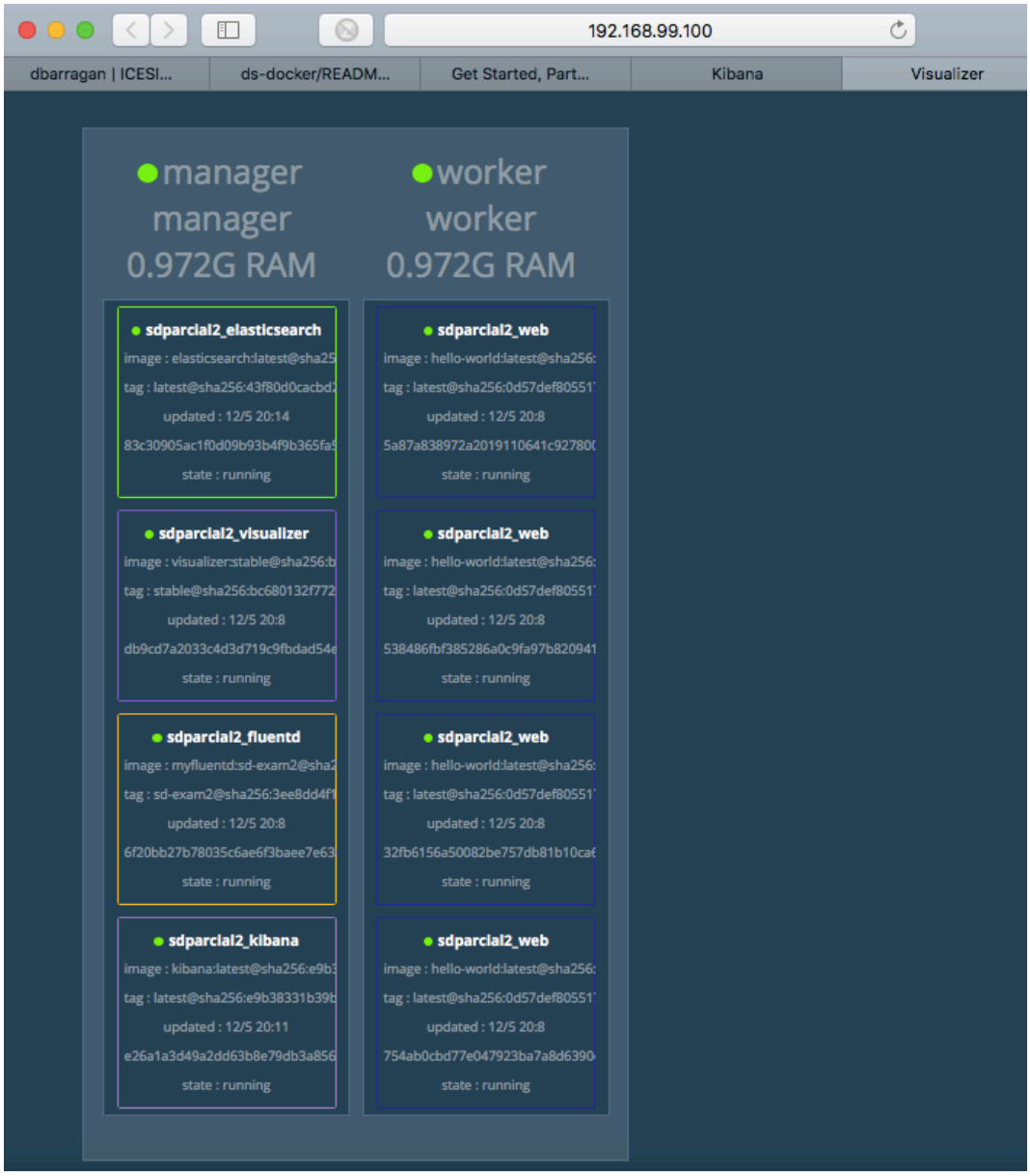
El comando para desplegar la infraestructura es:

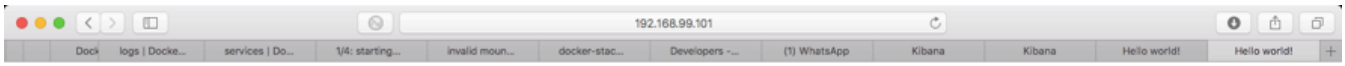
```
docker stack deploy -c docker-compose.yml sdparcial2
```

4. Diagrama de despliegue:



5. Evidencias funcionando

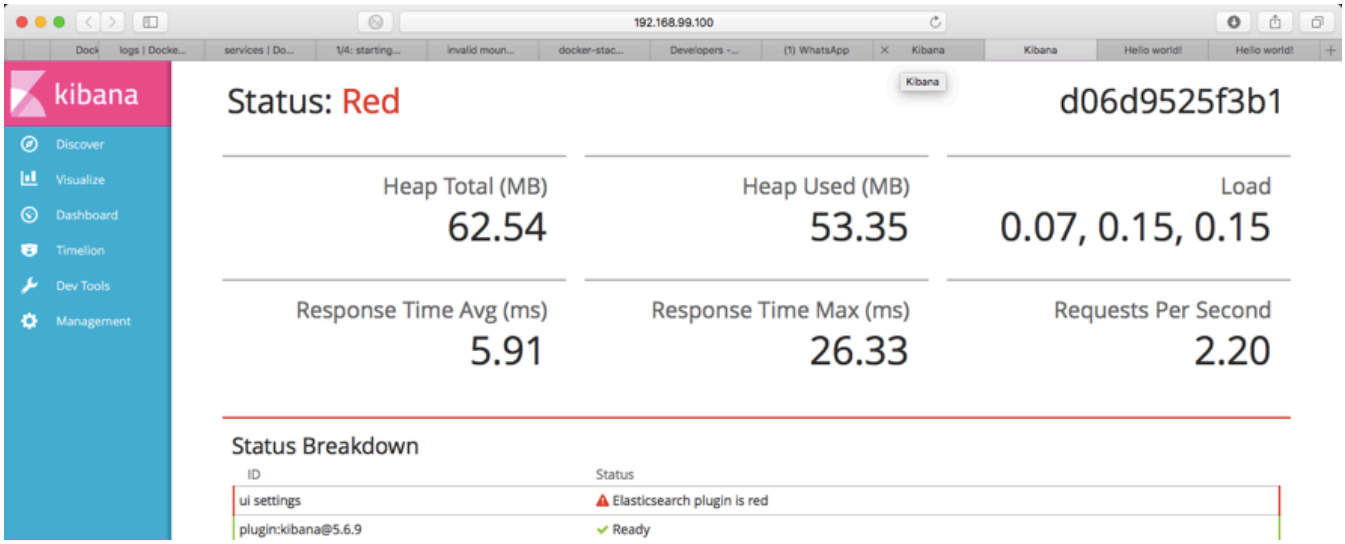




tutum

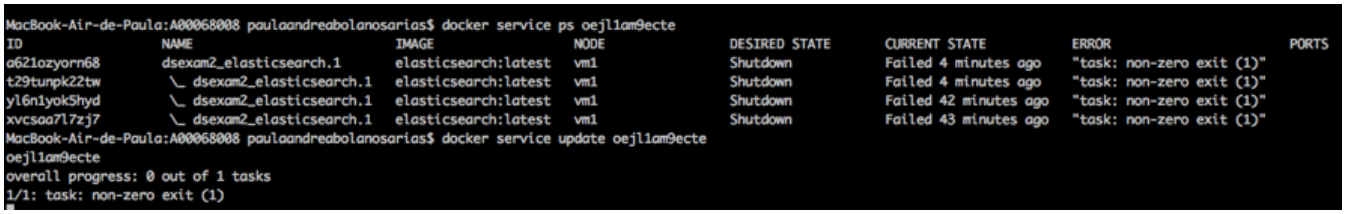
Hello world!

My hostname is c480959fddf1



6. Dificultades encontradas:

- Problemas con el formato yml en el docker-compose, ya que había utilizado tabs para la alineación y el formato sólo admite espacios.
- Problema al crear el contenedor kibana, que inicialmente se creía que faltaba la instalación de algún complemento, pero era un espacio que faltaba en la alineación del docker-compose. Sin embargo, el error que arrojaba no era en el docker-compose, pero se arregló con eso.
- Problemas al subir los contenedores, sólo subía el contenedor kibana. Se revisó con "docker stack ps" y "docker service update" cada uno de los otros servicios, hasta que se encontró que había problema en los volúmenes porque no estaban creados. Se crearon dentro del mismo repositorio y cuando se volvió a desplegar, subieron los contenedores.
- Problemas de reinicio en el contenedor del elasticsearch, ya que inicialmente sube pero se apaga y hace lo mismo varias veces hasta que queda apagado.



Esto afecta la visualización en kibana, por lo que la pantalla que se ve muestra estadísticas mas no los archivos en formato json ni las gráficas. Se agregó al docker-compose un healthcheck para verificar qué estaba pasando al inicializar el contenedor, de esta forma se obtuvieron estos logs: {"log":"Caused by: java.nio.file.AccessDeniedException: /usr/share/elasticsearch/data/nodes/0/node.lock\n"},{"log":"Caused by: java.io.IOException: failed to obtain lock on /usr/share/elasticsearch/data/nodes/0\n"},{"log":"Caused by: java.lang.IllegalStateException: failed to obtain node locks, tried [[/usr/share/elasticsearch/data/myclustersss]] with lock id [0]; maybe these locations are not writable or multiple n}.

Referencias

- https://docs.docker.com/compose/install/
- https://docs.docker.com/compose/compose-file/
- https://docs.docker.com/engine/swarm/stack-deploy/
- http://www.littlebigextra.com/using-kibana-and-elasticsearch-for-log-analysis-with-fluentd-on-docker-swarm/

URL Repositorio: www.github.com/paulaandrea27/sd-exam2