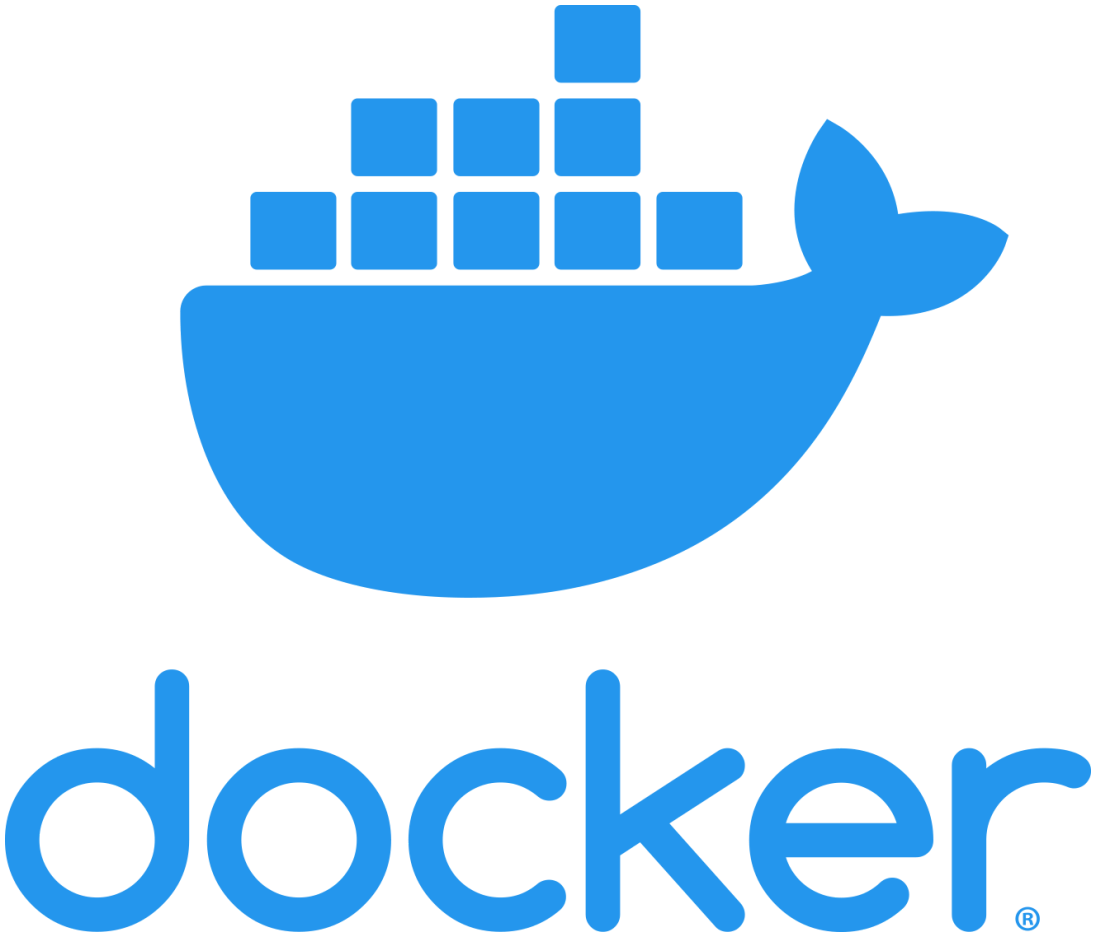
DESARROLLO DE SOFTWARE CRÍTICO

MEDICIONES EN DOCKER

JUAN JOSÉ TIRADO ARREGUI



Introducción

En esta práctica vamos a simular un sistema de monitorización de mediciones aplicando persistencia de datos y replicación en el lado del servidor mediante contenedores. Vamos a crear un stack Docker con un contenedor que proporcionará una API tipo REST, y a la que podremos proporcionar las mediciones realizadas por nuestro sistema de monitorización. La API REST proporcionará, entre otros, servicios para recibir nuevas mediciones o listar las mediciones existentes.

Desarrollo de la práctica

get("/nuevo/:dato", (req, res): Recibe una nueva medición en :dato. Esta medición debe almacenarse por si la réplica falla. Con jedis.rpush(lista, valor) guardamos el valor en la lista y con req.params(":dato") obtenemos el valor que contiene :dato.

get("/listar", (req, res): Muestra en un listado de texto las mediciones almacenadas. Con jedis.llen(lista) obtenemos la longitud de la lista y con jedis.lindex(lista, pos) obtenemos el valor de la lista en la posición pos.

Desarrollo de anomalías y gráfica

get("/detectaAnomalia", (req, res): utilizando lo ya trabajado en la primera práctica, adaptándola para la introducción de las mediciones en la ventana, devuelve el String del JSON junto con toda las mediciones y si esa ventana contiene anomalía. Si el tamaño de los datos introducidos es mayor o igual que el tamaño de ventana, entonces se procederá a detectar las anomalías de todas las listas de mediciones que permita realizar la ventana deslizante.

Al usar la clase Training necesito poner en el DockerFile que añada el fichero de entrenamiento:

```
FROM openjdk:8
WORKDIR /
ADD Ej1.jar Ej1.jar
ADD SortedTraining.txt SortedTraining.txt
CMD java -jar Ej1.jar
```

get("/grafica", (req, res): muestra una gráfica con las últimas 10 mediciones. Para ello utilizo la clase GraficaChart en la cuál añado dos métodos para obtener las mediciones desde Redis, además de incluir el nombre de mi contenedor:

```
try (Jedis jedis = new Jedis(Ej1.REDIS_HOST)) {
    long len = jedis.llen("queue#mediciones");
    for(long i = len - TAM; i<len; i++) {
        lds.addData(Integer.parseInt(jedis.lindex("queue#mediciones", i)));
    }
}
```

```
try (Jedis jedis = new Jedis(Ej1.REDIS_HOST)) {
    long len = jedis.llen("queue#timestamp");
    for(long i = len - TAM - 1; i<len; i++) {
        ld.addLabel(jedis.lindex("queue#timestamp", i));
    }
}
```

```
.setLabel("jjtirado/practica2:ej2")
```

get("/listajson", (req, res): devuelve un listado en formato JSON con las últimas 10 mediciones. El método CreatewebPage de GraficaChart lo llamará para obtener este json.

Desarrollo de docker swarm

Para el desarrollo del docker compose, utilizo el habilitado en el campus y añado lo marcado en negrita.

version: "3"

services:

web:

replace username/repo:tag with your name and image details

image: jjtirado/practica2:ej2 **# incluyo mi contenedor**

deploy:

replicas: 5 **# incluyo el número de réplicas**

restart_policy:

condition: on-failure

environment:

- REDIS_HOST=redis **# incluyo la variable global REDIS_HOST**

ports:

- "4000:4567" **# incluyo el puerto 4567(usado por java_spark)**

networks:

```
    - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
  redis:
    image: redis
    ports:
      - "6378:6378"  # incluyo puertos de redis
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Creación de contenedor y subida docker hub

Comandos utilizados para ello:

1. docker login // log de docker
2. docker build -t jjtirado/practica2:ej2 // para crear el contenedor
3. docker tag jjtirado/practica2:ej2 jjtirado/practica2:ej2 // para etiquetar el contenedor
4. docker push jjtirado/practica2:ej2 // para subir a docker hub

Comandos para ejecutar el docker swarm

Utilizados:

1. docker swarm init
2. docker stack deploy -c docker-compose.yml practica2 // despliega los servicios
3. Diferentes comandos para comprobar, borrar,etc
    a. docker stack ls // ver los stacks en ejecución
    b. docker service ls // ver los servicios en ejecución
    c. docker stack rm <stack> // borrar el stack que queramos

Kubernetes

Para el desarrollo de kubernetes es necesario modificar el web deployment y el web service dados de ayuda en el campus aunque primero debemos activar kubernetes en la configuración de docker desktop.

web- service:

```
apiVersion: v1

kind: Service

metadata:

  name: web
```

```
  labels:
    app: web
spec:
  selector:
    app: web
  type: LoadBalancer
  ports:
  - protocol: "TCP"
    port: 4000
    targetPort: 4567 //cambio puerto a 4567 (usado para java spark)
```

web deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
  labels:
    app: web
spec:
  replicas: 5
  selector:
    matchLabels:
      app: web
  template:
    metadata:
```

```
  labels:
    app: web
spec:
  containers:
  - name: web
    env:
    - name: REDIS_HOST //incluyo la variable global de REDIS HOST con valor redis
      value: redis
    image: jjtirado/practica2:ej2 //incluyo mi imagen
    ports:
    - containerPort: 4567 //cambio puerto a 4567 (usado para java spark)
```

Comandos Kubernetes

1. kubectl apply -f web-deployment.yaml //despliega web deployment
2. kubectl apply -f redis-deployment.yaml //despliega redis deployment
3. kubectl apply -f web-service.yaml //despliega web service
4. kubectl apply -f redis-service.yaml //despliega redis service
5. Comandos de comprobación:
    a. kubectl get pods //para ver pods en ejecución
    b. kubectl get deployments //para ver deployments en ejecución
    c. kubectl get services //para ver servicios en ejecución