

CONTENEDORES CON DOCKER

Desarrollo de Software Crítico



CRISTINA BARÓN SUÁREZ

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Universidad de Málaga

Índice

Creación del código Java.....	3
Creación del contenedor	7
Ejecución.....	10
Kubernetes	11

Imágenes

Ilustración 1. Clase Medición	3
Ilustración 2. Clase Respuesta	3
Ilustración 3. main()	3
Ilustración 4. leerDatos()	4
Ilustración 5. escribirDatos()	4
Ilustración 6. mostrarDatos()	5
Ilustración 7. obtenerDatos()	5
Ilustración 8. detectarAnomalia()	6
Ilustración 9. docker-compose.yml	9
Ilustración 10. Dockerfile	10
Ilustración 11. redis-service.yaml	12
Ilustración 12. redis-deployment.yaml	12
Ilustración 13. web-service.yaml	12
Ilustración 14. web-deployment.yaml	12

Creación del código Java

Clase Medición

```
public class Medicion {  
    public Date fecha;  
    public Integer valor;  
}
```

Ilustración 1. Clase Medición

Se ha creado la clase **Medición** para poder darle forma a los datos que se insertan en la base de datos de jedis.

Clase Respuesta

```
public class Respuesta {  
    public int mediciones;  
    public String datos;  
    public String anomalia;  
}
```

Ilustración 2. Clase Respuesta

Se ha creado también la clase **Respuesta** para darle forma a los datos a la hora de pasarlos a JSON.

Clase Java_Spark: Clase principal del programa. En ella se encuentran los métodos que se muestran a continuación.

```
// Intenta conectarse con REDIS_HOST y si no puede lo hará con localhost  
static final String REDIS_HOST = System.getenv().getOrDefault("REDIS_HOST", "localhost");  
private static int tamVentana = 4; // Igual que en el proyecto de Cadenas de Markov  
private static Gson gson = new Gson();
```

- **Método Main**

En el caso de que no se quisiese borrar el contenido de la base de datos cada vez que se realizase una nueva conexión, se comentaría la línea en la que se indica tal acción.

```
public static void main(String[] args) {  
    try {  
        System.out.println("Conectando con " + REDIS_HOST);  
        Jedis jedis = new Jedis(REDIS_HOST);  
        jedis.flushAll(); // Borra el contenido previo  
        jedis.close();  
    } catch (Exception ex) { }  
    leerDatos();  
    mostrarDatos();  
}
```

Ilustración 3.main()

- **Leer nuevos datos del buscador**

Este método insertará el valor leído y la hora a la que este se creó en la base de datos. Dicha medición es la que se pasa por parámetro.

```

private static void leerDatos() {
    // Detectar anomalías en los últimos 4 introducidos
    get("/nuevo/:dato", (req, res)->{
        Medicion nueva = new Medicion();
        nueva.fecha = new Date(); // Instante actual
        nueva.valor = Integer.parseInt(req.params(":dato"));
        return escribirDatos(nueva);
    });
}

```

Ilustración 4. leerDatos()

Además, se mostrará un mensaje en formato JSON para indicar si se ha podido realizar o no la acción con éxito. En el caso de que se haya podido llevar a cabo, se enseñará el nuevo valor que se ha almacenado en la base de datos y, si es posible, si la última ventana de 4 valores ha originado una anomalía.

```

public static String escribirDatos(Medicion dato) {
    try {
        Jedis jedis = new Jedis(REDIS_HOST); // Conexión
        DateFormat df = new SimpleDateFormat("M dd yyyy HH:mm:ss");
        jedis.rpush("queue#fechas", df.format(dato.fecha));
        jedis.rpush("queue#datos", (dato.valor).toString());

        String s = "";
        // Obtener la cantidad de datos que hay:
        long tam = jedis.llen("queue#fechas");
        // Se puede estudiar si hay alguna anomalía?
        if(tam >= tamVentana) { // Se puede. Hay al menos 4 valores en la BD
            String val = "";
            /* Se guardarán en una lista las últimas 4 mediciones
               que se han insertado en la base de datos */
            List<Integer> mediciones = new ArrayList<>();
            for(long i = tam-tamVentana; i < tam; i++) {
                val = jedis.lindex("queue#datos", i);
                mediciones.add(Integer.parseInt(val));
            }
            // Se estudia si la última ventana introducida origina una anomalía
            s = DetectorAnomalias.detectarAnomaliaPr2(mediciones);
        }
        // Muestra el valor (medida + hora) introducido en la base de datos
        String res = "Se ha introducido: " + gson.toJson(dato);
        jedis.close();
        if(tam < tamVentana) { // Hacen falta más medidas para poder tectar anomalías
            long faltan = tamVentana-tam;
            return gson.toJson(res + ". Hacen falta al menos " + faltan +
                " mediciones más para detectar si se ha producido una anomalía");
        } else {
            // Muestra el valor introducido y si se ha originado una anomalía
            return gson.toJson(res + s);
        }
    } catch (Exception e) {
        e.printStackTrace();
        return gson.toJson("NO se ha podido escribir el dato " + gson.toJson(dato));
    }
}

```

Ilustración 5. escribirDatos()

- **Mostrar el contenido de la base de datos**

Este método mostrará en formato JSON todas las mediciones que hay en la base de datos: sus valores con sus respectivos timestamps, y si alguna de las ventanas

ha sido origen de una anomalía o no. En el caso de no ser posible estimar si ha habido una anomalía por insuficiencia de datos, se notificaría.

```
private static void mostrarDatos() {
    get("/listar", (req, res)->{
        return obtenerDatos();
    });
}
```

Ilustración 6. mostrarDatos()

```
public synchronized static String obtenerDatos() {
    String s;
    try {
        Jedis jedis = new Jedis(REDIS_HOST); // Conexión
        String fecha = "", dato = "";
        List<Integer> mediciones = new ArrayList<>();
        List<Medicion> fecha_valor = new ArrayList<>();
        DateFormat df = new SimpleDateFormat("M dd yyyy HH:mm:ss");
        // Obtiene la cantidad de mediciones que hay en la BD:
        long tam = jedis.llen("queue#fechas");
        // Comprueba que haya un numero mayor o igual de datos que la ventana
        if(tam < tamVentana) { // No se puede detectar anomalía
            // Obtiene las mediciones de la BD:
            for(int i = 0; i < tam; i++) {
                fecha = jedis.lindex("queue#fechas", i);
                dato = jedis.lindex("queue#datos", i);

                Medicion med = new Medicion();
                med.fecha = df.parse(fecha);
                med.valor = Integer.parseInt(dato);
                // Para mostrar las mediciones (valor + hora) como lista:
                fecha_valor.add(med);
            }
            long faltan = tamVentana-tam;
            Respuesta res = new Respuesta();
            res.mediciones = (int)tam;
            res.datos = gson.toJson(fecha_valor);
            // Faltan datos para poder detectar anomalías
            res.anomalia = "Hay " + tam + " mediciones, deben introducirse al menos "
                + faltan + " datos más para poder detectar posibles anomalías";
            s = gson.toJson(res);
        } else { // Se puede detectar anomalía
            for(int i = 0; i < tam; i++) {
                fecha = jedis.lindex("queue#fechas", i);
                dato = jedis.lindex("queue#datos", i);

                Medicion med = new Medicion();
                med.fecha = df.parse(fecha);
                med.valor = Integer.parseInt(dato);
                /* Mete en una lista de mediciones todos los valores para estudiar
                posteriormente si alguna de las ventanas originadas provoca una
                anomalía: DetectorAnomalías.detectarAnomaliaPr2(mediciones) */
                mediciones.add(med.valor);
                // Para mostrar las mediciones (valor + hora) como lista:
                fecha_valor.add(med);
            }
            // Formato JSON:
            Respuesta res = new Respuesta();
            res.mediciones = mediciones.size();
            res.datos = gson.toJson(fecha_valor);
            res.anomalia = DetectorAnomalías.detectarAnomaliaPr2(mediciones);
            s = gson.toJson(res);
        }
        jedis.close();
        return s;
    } catch (Exception ex) {
        return gson.toJson(ex);
    }
}
```

Ilustración 7. obtenerDatos()

Para poder llevar a cabo la detección de anomalías empleadas en los métodos anteriores, se ha reutilizado parte del código de la práctica de las Cadenas de Markov.

- **Programa para detectar anomalías**

Este método se encuentra en una clase llamada **DetectorAnomalias**, dentro del paquete **prMarkovChains** y leerá de fichero datos recopilados en la primera práctica: el mayor y menor estado estudiados y el punto de corte (menor probabilidad de una ventana de 4 que se dio). Estos datos son necesarios para la correcta ejecución del programa. En la anterior práctica, ya había creado un método que introdujese dichos valores en tres distintos ficheros de texto por lo que simplemente los copié y pegué en el nuevo proyecto con el fin de no necesitar el anterior programa entero y bastarme de los métodos justos y necesarios.

La única modificación que representa con respecto al código de la práctica anterior es que esta vez, en vez de no devolver nada, retorna un String que indica si se ha producido una anomalía, en cuyo caso devolverá la ventana causante, y lo recuperará el método que lo llame. Además, se le pasará por parámetro la lista de mediciones que se desea estudiar.

```
private static int rangoVentana = 4;
private static int numIntervalo = 10;

public synchronized static String detectarAnomaliaPr2(List<Integer> nums)
    throws FileNotFoundException {

    double pc = leerPuntoCorte();
    int menor = leerMenor();
    int mayor = leerMayor();
    double[][] matrizProb = leerMatrizP();
    List<Integer> estados = new ArrayList<>();
    boolean anomalia = false;
    String s = "";
    for(int i = 0; i < nums.size(); i++) {
        estados.add(calcEstado(menor, mayor, nums.get(i)));
        if(estados.size() == rangoVentana) {
            double p3t = 1.0;
            for(int j = 1; j < estados.size(); j++) {
                p3t *= matrizProb[estados.get(j-1)][estados.get(j)];
            }
            //tolerancia
            if(p3t <= pc + 0.0000001) {
                anomalia = true;
                s += "[" + nums.get(i-3) + " - " + nums.get(i-2) + " - "
                    + nums.get(i-1) + " - " + nums.get(i) + "]";
            }
            estados.remove(0);
        }
    }
    if(anomalia) {
        return "Se ha producido una anomalía: " + s;
    }else{
        return "No se han detectado anomalías";
    }
}
```

Ilustración 8. detectarAnomalia()

```
static synchronized int calcEstado(int menor, int mayor, int n) {
    float div = (float) (n-menor)/(float) (mayor-menor);
    int aux = Math.round((numIntervalo-1)*div);
    if(aux < 0) { // Por si llegan valores menores que el menor
        aux = 0;
    } else if(aux > (numIntervalo-1)) { // Por si llegan valores mayores que el mayor
        aux = numIntervalo-1;
    }
    return aux;
}
```

El método “calcEstado()” no se ha modificado con respecto a la práctica anterior. Recordemos que dicho método servía para calcular el estado al que pertenece la medición que se esté estudiando. La matriz será de tamaño 10x10 en cualquier caso.

Creación del contenedor

En primer lugar, se ha creado una carpeta en la que se ha introducido el docker-compose.yml, el Dockerfile y el ejecutable creado en Eclipse, llamado, en mi caso, practica2.jar. Se explicará cada uno de los elementos a continuación.

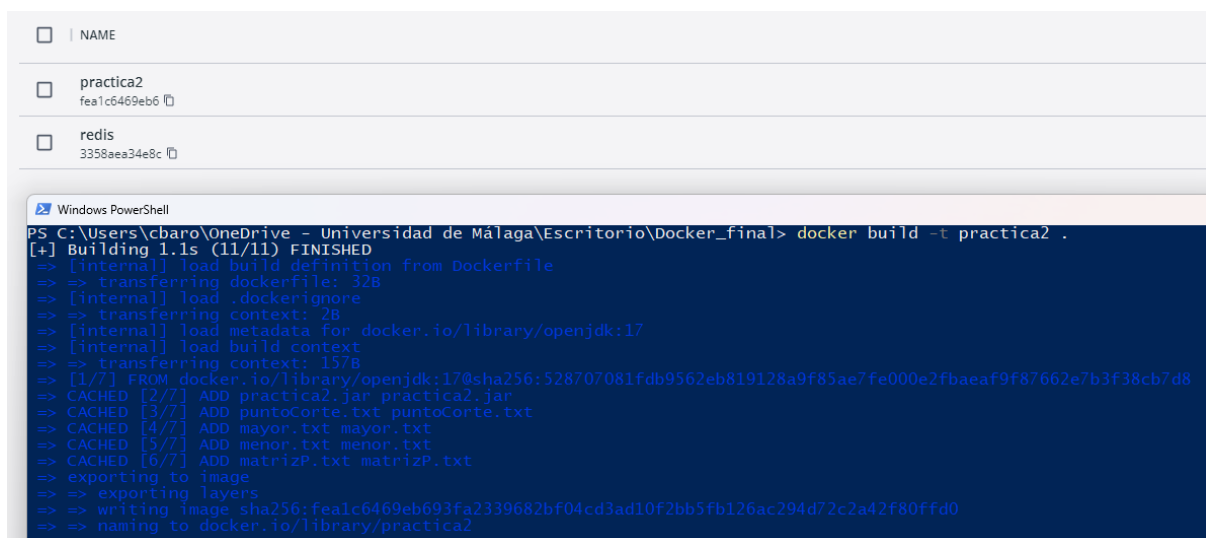
El docker-compose.yml necesita de mi imagen para poder ejecutarse por lo que habrá que subirla al repositorio. Por lo tanto, dentro del directorio mencionado anteriormente, se abre la terminal.

Inicio sesion:

docker login

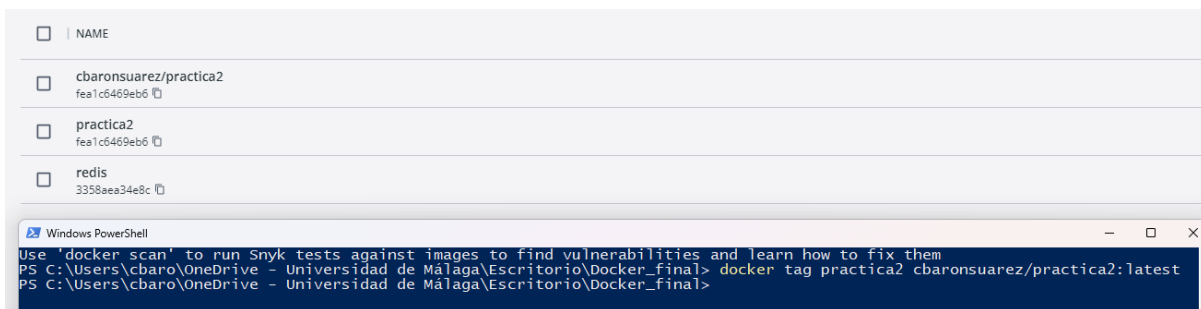
Creo la imagen:

docker build -t practica2 .



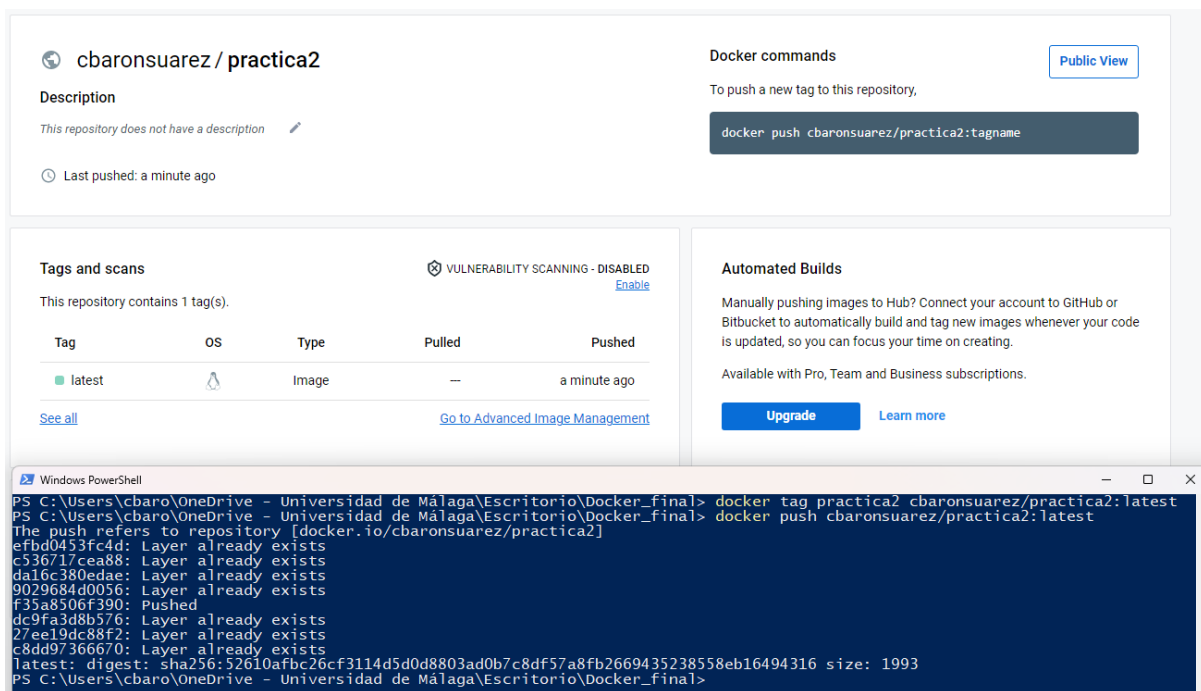
Etiqueto mi imagen como “practica2”.

docker tag practica2 cbaronsuarez/practica2:latest



Subo la imagen “cbaronsuarez/practica2” a hub.docker.com:

`docker push cbaronsuarez/practica2:latest`

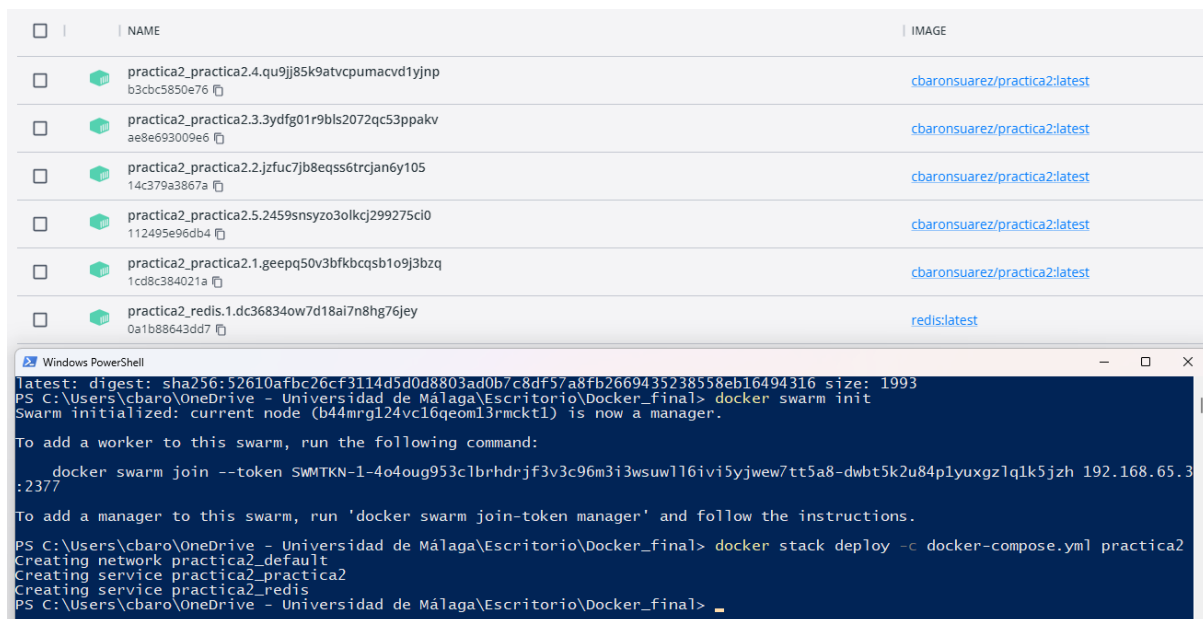


Con el objetivo de asignar distintos puertos a cada réplica y mantener los 5 contenedores activos constantemente, he iniciado mi máquina como un Docker Swarm Manager:

`docker swarm init`

Despliego mis servicios:

`docker stack deploy -c docker-compose.yml practica2`



Tras ejecutar los comandos previos, se puede comprobar que se han creado las 5 replicas y están todas corriendo, junto con redis.

Para llevar a cabo la replicación de los contenedores, he creado un Docker compose al que llamo docker-compose.yml y cuyo contenido es el que sigue:

```

version: "3"
services:
  practica2:
    image: cbaronsuarez/practica2:latest
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
    ports:
      - "80:4567"
    environment:
      - REDIS_HOST=redis

  redis:
    image: redis
    ports:
      - "6379:6379"
    deploy:
      placement:
        constraints: [node.role == manager]

```

Ilustración 9. docker-compose.yml

Se puede apreciar cómo el contenedor “practica2” obtiene la imagen del repositorio que previamente he subido a hub.docker.com y crea 5 réplicas de este sin posibilidad de que dicho número disminuya en el caso de que alguna sea parada. Se puede distinguir también cómo hace uso del puerto indicado.

Por otro lado, debe poder conectarse a la base de datos para poder leer las mediciones que han sido introducidas y ver si en alguna ventana se ha producido una anomalía por lo que se debe indicar también el “environment”.

Por otra parte, el contenedor “redis” hace uso del puerto “6379:6379” y se le asigna cómo orquestador a la hora de controlar las réplicas: *node.role == manager*

```
FROM openjdk:17
WORKDIR /
ADD practica2.jar practica2.jar
ADD puntoCorte.txt puntoCorte.txt
ADD mayor.txt mayor.txt
ADD menor.txt menor.txt
ADD matrizP.txt matrizP.txt
EXPOSE 4567
ENV NAME REDIS_HOST
CMD java -jar practica2.jar
```

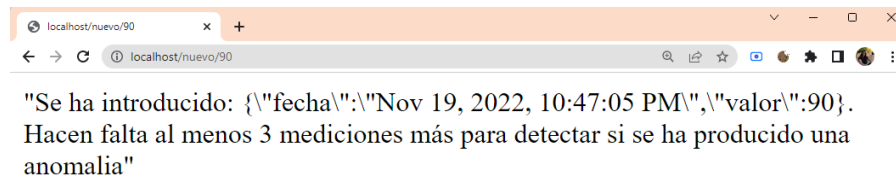
Se necesitan importar los ficheros obtenidos de las ejecuciones de la práctica 1 para que pueda ser posible la ejecución del programa. Dichos ficheros son: "puntoCorte.txt", "mayor.txt", "menor.txt" y "matrizP.txt".

Ilustración 10. Dockerfile

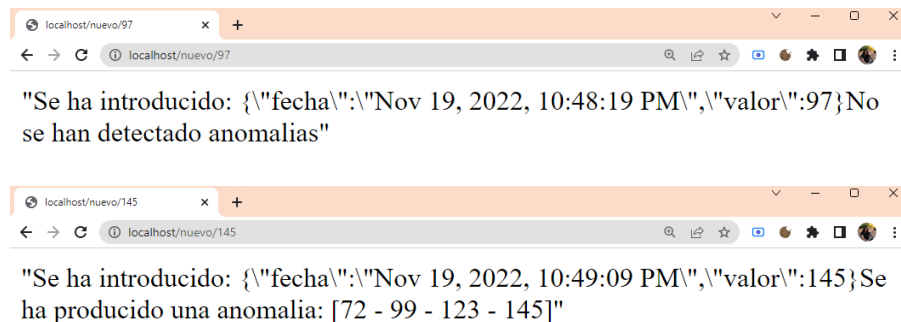
Ejecución

Para probar que el sistema detecta las anomalías, he probado introducir parámetros con los que ya entrenó y no se produce ninguna irregularidad hasta que detecta la cadena que fue la causante del punto de corte:

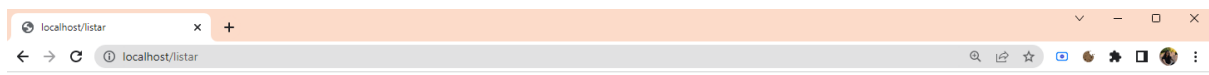
Mientras no haya más de 3 datos en la base de datos, no se le podrá mostrar al usuario si se ha detectado una anomalía. Por ejemplo:



Una vez se haya superado dicho número de mediciones, se comenzarán a enseñar mensajes por cada dato introducido que muestre si se ha causado una anomalía en dicha ventana o no. Por ejemplo:



El resultado de listar todos los datos de la base de datos es el siguiente, donde al final se muestran las ventanas con anomalías. La ventana de ejemplo que uso para detectar la anomalía es el mismo que originó el punto de corte en la práctica 1:




```

{"mediciones":9,"datos":[{"fecha":"Nov 19, 2022, 10:47:05 PM","valor":90}, {"fecha":"Nov 19, 2022, 10:48:06 PM","valor":99}, {"fecha":"Nov 19, 2022, 10:48:13 PM","valor":120}, {"fecha":"Nov 19, 2022, 10:48:19 PM","valor":97}, {"fecha":"Nov 19, 2022, 10:48:59 PM","valor":72}, {"fecha":"Nov 19, 2022, 10:49:04 PM","valor":99}, {"fecha":"Nov 19, 2022, 10:49:07 PM","valor":123}, {"fecha":"Nov 19, 2022, 10:49:09 PM","valor":145}, {"fecha":"Nov 19, 2022, 10:49:33 PM","valor":80}], "anomalia":"Se ha producido una anomalia: [72 - 99 - 123 - 145]"}

```

En el caso de que no hubiese ninguna anomalía, se indicaría como sigue en el ejemplo siguiente:




```

{"mediciones":4,"datos":[{"fecha":"Nov 19, 2022, 11:05:37 PM","valor":90}, {"fecha":"Nov 19, 2022, 11:19:02 PM","valor":99}, {"fecha":"Nov 19, 2022, 11:19:09 PM","valor":70}, {"fecha":"Nov 19, 2022, 11:19:13 PM","valor":80}], "anomalia":"No se han detectado anomalias"}

```

En caso de solicitar listar la base de dato y en esta se encuentren entre 0 y 3 mediciones, se mostrarán dichos valores y además se enseñará un mensaje diciendo que no hay suficientes mediciones para comprobar si se ha dado una anomalía. Por ejemplo:



```

{"mediciones":1,"datos":[{"fecha":"Nov 19, 2022, 11:05:37 PM","valor":90}], "anomalia":"Hay 1 mediciones, deben introducirse al menos 3 datos más para poder detectar posibles anomalías"}

```

Kubernetes

Dentro de la misma carpeta en la que se encontraban los archivos anteriores, se crean los siguientes cuatro ficheros: “redis-deployment.yaml”, “redis-service.yaml”, “web-deployment.yaml” y “web-service.yaml”. Entre ellos se conectan para poder ofrecer un servicio como el de Swarm.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: "redis"
          ports:
            - containerPort: 6379

```

Ilustración 12. redis-deployment.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    app: redis
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    app: redis

```

Ilustración 11. redis-service.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-practica2-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: app-practica2
  template:
    metadata:
      labels:
        app: app-practica2
    spec:
      containers:
        - name: app-practica2
          image: cbaronsuarez/practica2:latest
          env:
            - name: REDIS_HOST
              value: "redis"
          ports:
            - containerPort: 4567

```

Ilustración 14. web-deployment.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: app-practica2-service
  labels:
    app: app-practica2
spec:
  type: NodePort
  ports:
    - protocol: TCP
      port: 4567
      targetPort: 4567
      nodePort: 30101
  selector:
    app: app-practica2

```

Ilustración 13. web-service.yaml

Para poder probar el programa en localhost, se ejecutan los siguientes comandos:

```

kubectl apply -f web-deployment.yaml
kubectl apply -f redis-deployment.yaml
kubectl apply -f redis-service.yaml
kubectl apply -f web-service.yaml

```

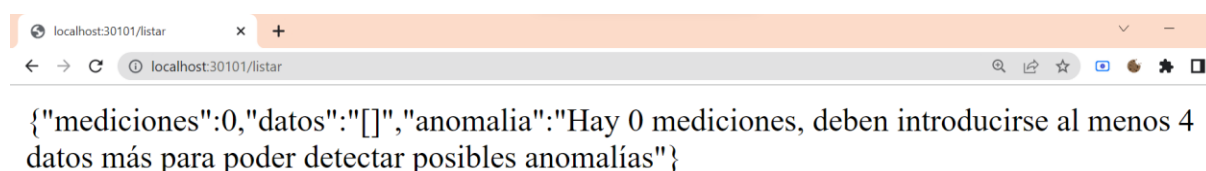
```
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke> kubectl apply -f web-deployment.yaml
deployment.apps/web-deployment created
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke> kubectl apply -f redis-deployment.yaml
deployment.apps/redis-deployment created
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke> kubectl apply -f redis-service.yaml
service/redis created
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke> kubectl apply -f web-service.yaml
service/practica2 created
```

Vemos los servicios que acabamos de crear:

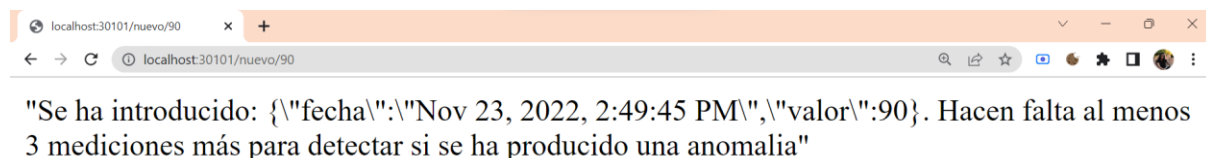
```
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke> kubectl get services
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
app-practica2-service              NodePort    10.99.44.3       <none>            4567:30101/TCP   7m53s
kubernetes                         ClusterIP   10.96.0.1        <none>            443/TCP          86m
practica2                          NodePort    10.109.127.138   <none>            4567:30100/TCP   84m
redis                              ClusterIP   10.101.57.124    <none>            6379/TCP         84m
PS C:\Users\cbaro\OneDrive - Universidad de Málaga\Escritorio\Docke>
```

La única diferencia que mostrará con respecto a cómo debía ejecutarse cuando había un swarm es que en esta ocasión el puerto no será el 80, sino el 30101, por lo que, en vez de poner en el buscador, por ejemplo, <http://localhost:80/nuevo/90> y <http://localhost:80/listar>, habrá que poner: <http://localhost:30101/nuevo/90> para insertar datos y <http://localhost:30101/listar> para listarlos y ver, si es posible, si hay anomalía.

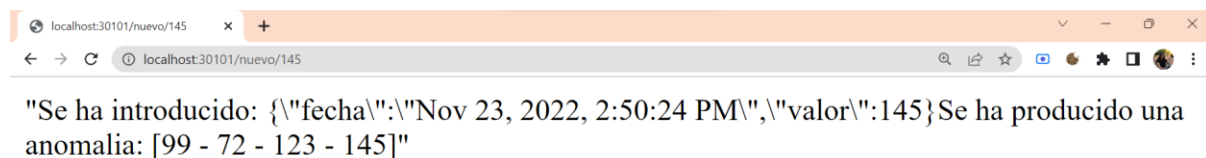
A continuación, muestro algunos ejemplos de ejecución, aunque el resultado sea exactamente el mismo que el de las capturas mostradas con anterioridad:



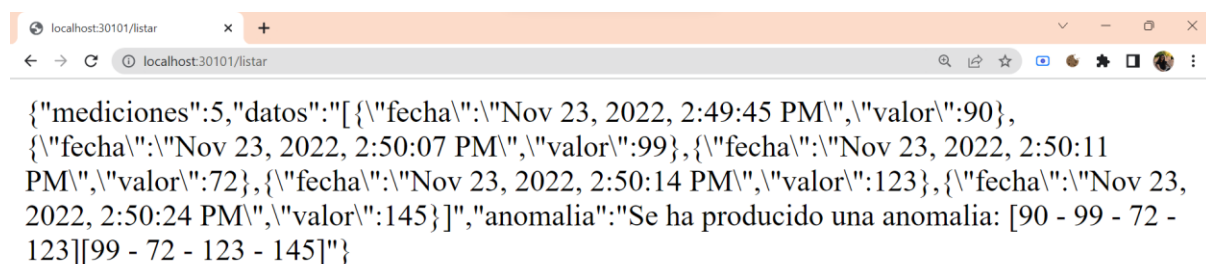
```
{\"mediciones\":0,\"datos\":\"[]\", \"anomalia\":\"Hay 0 mediciones, deben introducirse al menos 4 datos más para poder detectar posibles anomalías\"}
```



```
"Se ha introducido: {\\\"fecha\\\":\\\"Nov 23, 2022, 2:49:45 PM\\\",\\\"valor\\\":90}. Hacen falta al menos 3 mediciones más para detectar si se ha producido una anomalía"
```



```
"Se ha introducido: {\\\"fecha\\\":\\\"Nov 23, 2022, 2:50:24 PM\\\",\\\"valor\\\":145} Se ha producido una anomalía: [99 - 72 - 123 - 145]"
```



```
{\"mediciones\":5,\"datos\":\"[{\\\"fecha\\\":\\\"Nov 23, 2022, 2:49:45 PM\\\",\\\"valor\\\":90}, {\\\"fecha\\\":\\\"Nov 23, 2022, 2:50:07 PM\\\",\\\"valor\\\":99}, {\\\"fecha\\\":\\\"Nov 23, 2022, 2:50:11 PM\\\",\\\"valor\\\":72}, {\\\"fecha\\\":\\\"Nov 23, 2022, 2:50:14 PM\\\",\\\"valor\\\":123}, {\\\"fecha\\\":\\\"Nov 23, 2022, 2:50:24 PM\\\",\\\"valor\\\":145}]\", \"anomalia\":\"Se ha producido una anomalía: [90 - 99 - 72 - 123][99 - 72 - 123 - 145]\"}
```