eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

TRABAJANDO CON DOCKER Y RETHINKDB

TRABAJO INDIVIDUAL

Autor:
Gorka Ortiz

Asignatura:
Administración de Sistemas
4º Curso
Curso 2021/2022

Índice

1. Introducción

El objetivo de este trabajo es poner en práctica los conocimientos sobre técnicas de virtualización, contenedores y orquestación aprendidos durante el curso. Para esto, se ha asignado una imagen de Docker conocida como 'RethinkDB'y tendremos varias tareas a realizar usando esta.

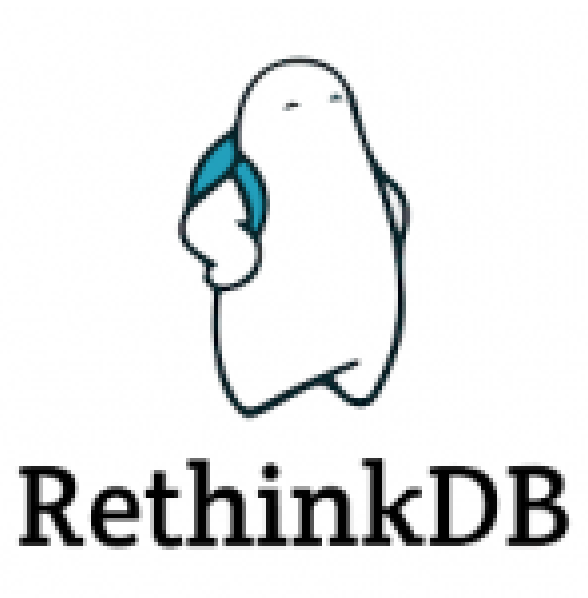

Figura 1: *RethinkDB*

Repositorio GitHub:
`https://github.com/ortiwii/Trabajo_Individual_Sistemas.git`

2. Documentación RethinkDB

2.1. Que es RethinkDB?

RethinkDB es la primera base de datos JSON escalable y de código abierto creada desde cero para la web en tiempo real. Invierte la arquitectura de base de datos tradicional al exponer un nuevo y emocionante modelo de acceso: en lugar de preguntar constantemente sobre los cambios, el desarrollador puede decirle a RethinkDB que envíe continuamente resultados de consultas actualizados a las aplicaciones en tiempo real. La arquitectura push en tiempo real de RethinkDB reduce drásticamente el tiempo y el esfuerzo necesarios para crear aplicaciones escalables en tiempo real.

2.2. Cuando es una buena opción?

RethinkDB es una buena opción cuando tus aplicaciones podrían verse beficiadas por feeds de datos en tiempo real. El modelo de acceso a la base de datos de consulta-respuesta funciona bien en la web porque se asigna directamente a la solicitud-respuesta de peticiones HTTP.
Entra las aplicaciones en las que se recomendaría el uso de esta base de datos No-SQL se destacan:

- Aplicaciones web colaborativas o móviles.
- Streaming de aplicaciones de análisis
- Aplicaciones multijugador
- Mercados en tiempo real

2.3. Como se usa?

2.3.1. Inicializar la base de datos

Para esto, tendremos que utilizar la imagen de docker que contiene el servicio de base de datos RethinkDB que se nos ha asignado, la podremos encontrar en el siguiente link de dockerhub: `https://hub.docker.com/_/rethinkdb`

Para descargar la imagen, usaremos el siguiente comando:

```
docker pull rethinkdb
```

Una vez descargada la imagen, pondremos a correr un contenedor docker con ella, usando este comando:

```
docker run --name some-rethink -v "$PWD:/data" -d rethinkdb
```

2.3.2. Acceso a la Base de Datos

Una vez que el servicio este en marcha, la imagen estará escuchando en diferentes puertos con diferentes fines. Mas concretamente en los siguientes:

- **8080**: El servicio, esta diseñado para que pueda ser administrado con una interfaz web, con lo cual este puerto sera el encargado de responder a las peticiones de la interfaz de administración del sistema.

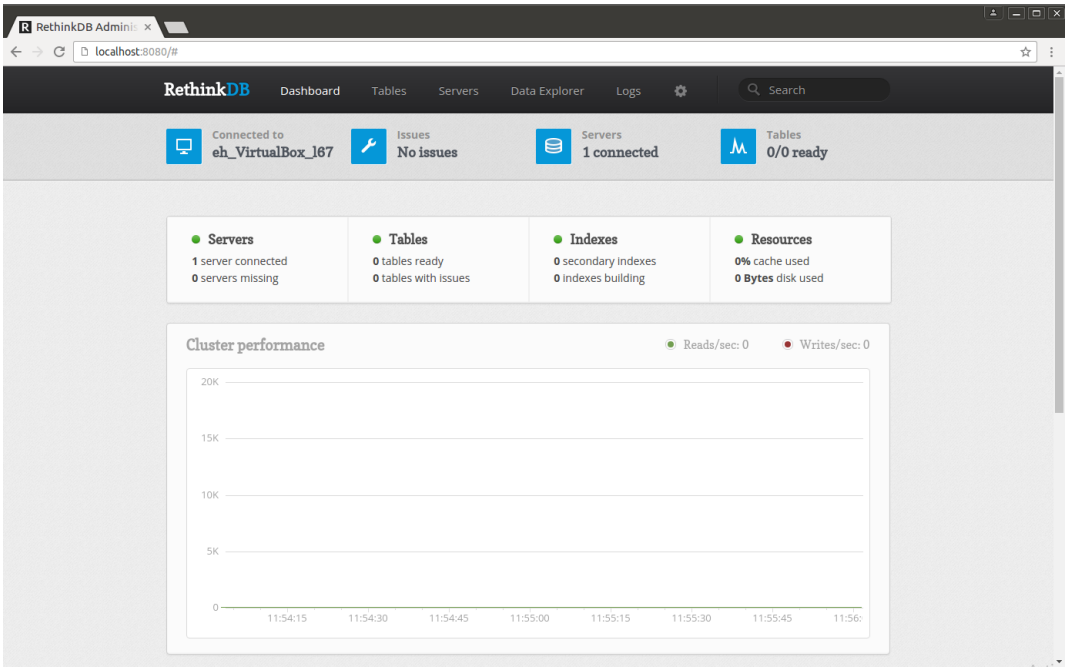

Figura 2: *http://localhost:8080/*

- **28015**: Este puerto es el encargado de atender, procesar y responder a las peticiones provenientes del driver de control de el cliente. Este driver, estará disponible para varios entornos de ejecución como JavaScript, Python, Ruby y Java.
  En este link, estaran disponibles los diferentes clientes ofrecidos: `https://rethinkdb.com/docs/install-drivers/`.

- **29015**: Este ultimo puerto se utilizara para las conexiones intracluster.

2.4. Operaciones básicas

Para la descripción de las operaciones básicas que tiene el cliente de RethinkDB, usaremos el diseñado para el entorno de Python, ya que es el que se ha usado en la implementación.

Abrir conexión

```
from rethinkdb import RethinkDB
r = RethinkDB()
r.connect( "localhost", 28015).repl()
```

Crear una tabla

```
r.db("test").table_create("authors").run()
```

Insertar datos a una tabla

```
r.table("authors").insert([
    { "name": "William Adama",
      "tv_show": "Battlestar Galactica",
      "posts": [
        {"title": "Decommissioning speech",
   "content": "The Cylon War is long over..."
   },
        {"title": "We are at war",
   "content": "Moments ago, this ship received..."
   },
        {"title": "The new Earth",
   "content": "The discoveries of the past few days..."
   }
      ]
    },
    { "name": "Laura Roslin",
      "tv_show": "Battlestar Galactica",
      "posts": [
        {"title": "The oath of office", "content": "I, Laura Roslin,
   ..."},
        {"title": "They look like us", "content": "The Cylons have the
    ability..."}
      ]
    },
    { "name": "Jean-Luc Picard", "tv_show": "Star Trek TNG",
      "posts": [
        {"title": "Civil rights", "content": "There are some words I'
   ve known since..."}
      ]
    }
]).run()
```

Ver todos los documentos de una tabla

```
cursor = r.table("authors").run()
for document in cursor:
    print(document)
```

Filtrar documentos por condición

```
cursor = r.table("authors").filter(r.row["name"] == "William Adama").
   run()
```

```
cursor = r.table("authors").filter(r.row["posts"].count() > 2).run()
```

Obtener documento por llave primaria

```
r.db('test').table('authors').get('7644aaf2-9928-4231-aa68-4
    e65e31bf219').run()
```

Actualizar documentos

```
# Actualizar todos:
r.table("authors").update({"type": "fictional"}).run()

# Actualizar uno:
r.table("authors").
    filter(r.row['name'] == "William Adama").
    update({"rank": "Admiral"}).run()

# Actualizar uno a adiendo contenido:
r.table('authors').filter(r.row["name"] == "Jean-Luc Picard").
    update({"posts": r.row["posts"].append({
        "title": "Shakespeare",
        "content": "What a piece of work is man..."})
    }).run()
```

Borrar documento

```
r.table("authors").
    filter( r.row["posts"].count() < 3 ).
    delete().run()
```

Feed en tiempo real

```
cursor = r.table("authors").changes().run()
for document in cursor:
    print(document)
```

3. Aplicación cliente

Como previamente hemos mencionado, el entorno de ejecución de la aplicación cliente será Python, mas concretamente la versión 3.8.12. La aplicación cliente, nos ofrecerá 4 diferentes opciones, dependiendo de cual sea nuestro objetivo en el momento de la ejecución.

Figura 3: *Aplicación Cliente*

La aplicación cliente la podremos encontrar en el archivo *aplicacion_cliente.py* que se encuentra en el repositorio GitHub mencionado en la introducción.

3.1. Opciones

3.1.1. Crear una base de datos

Esta opción, sera la encargada de crear una base de datos. Primero, se abrirá conexión con Rethinkdb, después se pedirá que introduzcas por consola el nombre para la base de datos que se creara y en caso de que ese nombre ya exista, se mostraran las bases de datos existentes.

3.1.2. Borrar una base de datos

Este apartado, sera el encargado de borrar una base de datos existente en Rethinkdb. Para empezar, se introduce el nombre de la tabla a eliminar por consola, y a no ser que esa BBDD no exista, se eliminara del sistema. En ese caso, se mostraran las BBDD actuales por pantalla.

3.1.3. Crear una tabla

Esta opción, sera la encargada de crear una tabla con un nombre en concreto. Se te pedirá el nombre de la base de datos en la que se quiere crear esa tabla y el nombre de la tabla. En caso de que la base de datos no exista o la tabla ya exista en esa BBDD, se notificara por consola, en caso contrario, se creará.

3.1.4. Ver el estado de una tabla

Este caso, esta mas orientado para la administración del sistema, para hacer consultas de estado. Al igual que cuando creamos una tabla, se nos pedirá el nombre de la BBDD y de la tabla, y en caso de que los datos sean correctos, se enseñara el estado de esta.

3.2. Docker

3.2.1. Encapsulación en imagen de Docker

Para poder conseguir esto, usaremos un archivo Dockerfile. Estos archivos, son utilizados para crear imágenes de docker y contienen toda la información necesaria para su creación. Partiendo de la imagen de python mas básica *python:3.8-slim*, tendremos que copiar el archivo de la aplicación cliente y instalar todos los paquetes necesarios para su ejecución. Asi sera nuestro archivo DockerFile:

```
FROM python:3.8-slim
RUN useradd --create-home --shell /bin/bash app_user
WORKDIR /home/app_user
COPY aplicacion_cliente.py ./
RUN pip install rethinkdb
RUN pip install readchar
USER app_user
CMD ["bash"]
```

3.2.2. Publicación DockerHub

DockerHub, es un repositorio publico de imagenes de Docker, con lo cual, si nosotros publicamos algo ahi, estara al acceso de cualquier otro usuario. Para, subir una imagen, primero tendremos que iniciar sesión en dockerhub mediante el comando:

```
sudo docker login
```

Despues, crearemos un tag para la imagen:

```
sudo docker tag aplicacion_cliente:v9 ortiwii/aplicacion_cliente:v9
```

Para acabar, haremos push del tag creado a DocerHub:

```
docker push ortiwii/aplicacion_cliente:v9
```

Después de esto, si todo ha ido correctamente si visitas tu perfil de DockerHub, podras ver que la imagen que acabas de subir estará disponible para todo el mundo. La imagen que yo he creado se encuentra en el siguiente link: `https://hub.docker.com/repository/docker/ortiwii/aplicacion_cliente`

4. Docker-Compose

Docker-compose, es el entorno que se utiliza para ejecutar varios contenedores a la vez y orquestar el funcionamiento de estos. En el archivo docker-compose.yml que se creara, se diferenciaran 2 contenedores:

1. **RethinkDB**: Este sera el contenedor utilizado para desplegar la imagen que se nos ha asignado. Esta, tendra que tener accesibles los puertos 8080, 28015 y 29015 y utilizara un volumen albergado en el directorio /var/opt/rethinkdb de el anfitrión. No tendremos que acceder a este contenedor en ningún momento, lo haremos mediante los puertos mencionados.

2. **Cliente**: Este contenedor, sera el que se utilizara para albergar la aplicación cliente que hemos creado previamente con sus respectivas dependencias ya instaladas. Una vez ejecutado el entorno de Docker, accederemos a el contenedor, para poder ejecutar el .py con python.

Esta pinta tiene el archivo docker-compose.yml:

```
version: "3.9"
services:
  servidor_rethinkdb:
    image: rethinkdb
    container_name: servidor_rethinkdb
    ports:
      - "8080:8080"
      - "28015:28015"
      - "29015:29015"
    volumes:
      - /var/opt/rethinkdb:/home/app_user
  servidor_cliente:
    build: .
    image: servidor_cliente
    container_name: servidor_cliente
    depends_on:
      - servidor_rethinkdb
    stdin_open: true # docker run -i
    tty: true        # docker run -t
```

Para ejecutar el entorno docker, tendremos que utilizar el siguiente comando, siempre y cuando estemos en la carpeta de el archivo .yml:

```
sudo docker-compose up --build
```

Una vez tengamos el entorno en ejecución, tendremos que acceder al contenedor que contiene la aplicación cliente, para despues poder ejecutar esa aplicacion python. Mas concretamente usaremos estos comandos:

```
user@user:~$ sudo docker exec -it servidor_cliente /bin/bash
app_user@2f5303048adb:~$ python aplicacion_cliente.py
```