Juan Felipe Martínez Bedoya, Juan Camilo Iguarán, Rafael Gomez

**Evidencias del Avance Proyecto 1.**

Hace falta aclarar que la realización de este taller fue enteramente en EC2, nunca se deseó instalar redis en la máquina local.

1. **Instalación de redis en una instancia EC2**

Después de crear una instancia común de EC2 en AWS se ingresaron los siguientes comandos para la correcta instalación de redis:

Texto, Carta

Descripción generada automáticamente

Seguido de los pasos se ingresa “make test” para probar que todos los servicios estén corriendo correctamente:



Como se puede ver, redis si instaló correctamente por lo que se puede seguir con el siguiente paso de crear un nodo sencillo de la base de datos.

1. **Uso de la base de datos en modo single node.**

Se inicializa el servidor para probar unos comandos básicos:

Texto

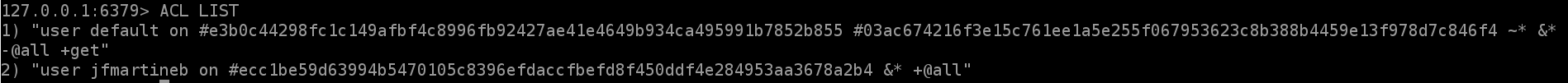
Descripción generada automáticamente

Para realizar la prueba más sencilla se realiza se inicia una nueva terminal, se crea el cliente y se realizan las operaciones más sencillas de escribir y leer, los resultados son los siguientes:

Texto

Descripción generada automáticamente

Además de poder hacer operaciones básicas como set o get también se aumentó la seguridad del servidor, creando un usuario “jfmartineb” que tiene todos los permisos disponibles, y al predeterminado también le debe pedir una contraseña, además de solo tener permisos de lectura:



La siguiente captura muestra el intento que se corrió con el usuario predeterminado,le cual solo puede realizar lecturas:

Texto

Descripción generada automáticamente

Y el siguiente es ingresando con el usuario jfmartineb que ya tiene permisos de realizar todos los cambios necesarios, como se puede ver en la imagen se puede leer, escribir y modificar las configuraciones del servidor tales como que tipo de consultas puede realizar el usuario:

Texto

Descripción generada automáticamente

1. **Operaciones CRUD básicas desde Python**

Antes de crear un archivo, se comprueba que la librería de redis hayan quedado bien instaladas, para eso se utiliza desde la terminal y se trata de leer el valor almacenado con la key ‘session\_id\_1’

Texto

Descripción generada automáticamente

Luego, se prueba con el programa sacado de <https://github.com/aploetz/packt/blob/master/redis/redisHelloWorld.py>, donde se leerá una información en la base de datos, se sobre escribirá y luego se leerá nuevamente. En este caso como los permisos de escritura solo están en usuario, se debe utilizar un método diferente para acceder a redis, para eso se debe utilizar es una url, con el formato de redis://user:password@host:port, eso se puede evidenciar en la siguiente imagen las modificaciones realizadas en el programa:

Texto

Descripción generada automáticamente

Lo que al correr el programa nos da los mismos resultados deseados:

Texto

Descripción generada automáticamente

Ya como última verificación podemos ingresar desde redis-cli para verificar que se hicieron correctamente los cambios a la variable deseada:

Texto

Descripción generada automáticamente

1. **Redis en modo cluster**

Para la creación del cluster se siguieron los pasos de la página oficial de redis, en la cual empieza por la creación de un directorio llamado cluster-test, en el cual hay 6 subdirectorios enumerados desde el 7000 hasta el 7005, donde en cada uno de ellos está el archivo de redis.conf, el cual contiene la siguiente información:

Texto

Descripción generada automáticamente

El puerto se debe cambiar en cada archivo para que corresponda con el nombre del directorio. Luego, se abren 6 instancias diferentes de EC2, en donde en cada una se irá a un diferente directorio y se correrá el comando “redis-client ./redis.conf, el cual corre el archivo ejecutable del cluster:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Luego de tener los 6 servidores corriendo, se crea el cluster con el comando:

Texto

Descripción generada automáticamente con confianza baja

Con el cual se crearan 3 particiones diferentes, con una réplica cada uno.

Texto

Descripción generada automáticamente

En este caso los maestros son los nodos ubicados en los puertos 7000, 7001, y 7002 y el resto son replicas de estos. (7004 del 7000, 7005 del 7001 y 7003 del 7002)

Ya al tener el cluster configurado, se procede a probar las operaciones sencillas CRUD get y set en él.

* 1. **Pruebas por medio de redis-cli**

Para empezar a utilizar el cluster se debe ingresar a uno de los nodos con el comando “redis-cli -c -p #puerto#”, en el puerto especificado se corre la función que se desee, por ejemplo, almacenar un valor. Luego se ingresa a otra partición y se busca ese dato, en ese caso se puede evidenciar que se reenvía a ese nodo donde se almacenó la información. El siguiente ejemplo que se corrió fue almacenar otro valor, el cual la clave hash estaba en otra partición diferente a la que se encontraba el cliente, por lo que se es redirigido a ese nodo. Después se almacena otro valor en el mismo nodo en lo cual sucede lo mismo. Ya para intentar la búsqueda se ingresa la clave a buscar y se direcciona automáticamente a donde este valor se encuentre:

Texto

Descripción generada automáticamente

* 1. **Pruebas por medio de Python**

La mejor manera de trabajar clusters en Python es utilizando la librería que se encuentra en https://github.com/Grokzen/redis-py-cluster.git

Para iniciar se trabajará desde una clase llamada nodo la cual tiene dos funciones, una para leer datos del nodo y otro para escribir nueva información:

Texto

Descripción generada automáticamente

Luego, se crean los 6 nodos y se agregan en una lista. La prueba realizada es muy sencilla, simplemente se escribe en el nodo 7000 y se hace una lectura del 7002 para comprobar que la red está conectada.

Texto

Descripción generada automáticamente

Además, para demostrar que sí funciona se escribe desde la línea de comandos para comprobar que el proceso de escritura se realiza correctamente:

Texto

Descripción generada automáticamente

Esta prueba nos muestra mucha más información, ya que en esta se puede ver que se redirecciona a un nodo diferente al que se utilizó para escribir la información (se pretendía en el 7000, se utilizó el 7001 porque a ese corresponde el rango de clave hash) y también, haciendo la consulta desde el nodo 7003 se puede acceder a la información.

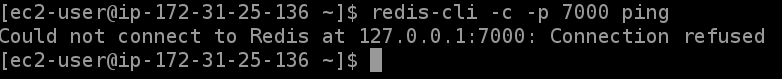
* 1. **Tolerancia a fallos**

Para comprobar la tolerancia a fallos, se almacenará un dato y luego se procederá a apagar el nodo en el cual ese haya sido almacenado, se comprobará haciendo una búsqueda del dato. Luego se apagará la réplica del nodo y se buscará nuevamente el dato para ver que sucede.

Texto

Descripción generada automáticamente

Luego, nos salimos de ese nodo, lo apagamos (nodo en el puerto 7000) y nos metemos a otro diferente y buscamos el mismo con la clave “tolerancia”



Texto

Descripción generada automáticamente

Como se puede ver ya después de apagar el servidor del puerto 7000, la redirección está orientada al puerto 7005

Texto

Descripción generada automáticamente

Al eliminar la réplica del servidor se puede ver que el dato sigue siendo accesible, por lo que como última prueba se apagará el nodo 7005 donde mostró que era la última ubicación del archivo

Texto

Descripción generada automáticamente

Al eliminar este nodo el sistema falla. Para llegar a este punto tocó eliminar un total de 3 nodos (la mitad) y al menos un grupo de partición y réplica (maestro y réplica).

1. **Códigos**

Código en Python de las operaciones básicas del CRUD en redis modo single-node:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Implementando colas con producto-consumidor en java. A continuación se presentan los pasos empleados para implementar el patrón de colas:

1. En consola escribir el siguiete commando para corer el contenedor: docker run --name redis -d redis:5.0.4
2. Creamos un proyecto en eclpise IDE con Gradle y Java
3. Dentro del proyecto, en la carpeta src se encuentra el archivo build.graddle el cual tendrá las siguientes dependencias para poder generar un jar ejecutable:

plugins {

id 'java'

}

dependencies {

implementation group: 'redis.clients', name: 'jedis', version: '3.0.1'

}

repositories {

jcenter()

}

jar {

manifest {

attributes 'Main-Class': 'com.lostsys.redis.Main'

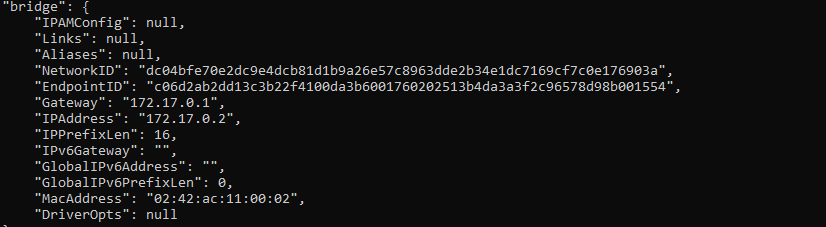
}

from {

configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }

}

}

1. Ejecutar el comando “docker inspect redis” para saber en qué dirección ip se ejecuta el servidor de redis: 
2. Para generar el jar, desde la carpeta del proyecto escribir: gradlew jar
3. Implementar un archivo main.java con el siguiente script:

package com.lostsys.redis;

import java.util.List;

import java.util.UUID;

import redis.clients.jedis.Jedis;

public class Main {

private static String REDIS\_HOST="172.17.0.2";

private static String REDIS\_LIST="listayoutube";

public static void main(String[] args) {

if (args.length!=1) {

printUsage();

return;

}

if ( args[0].equals("producer")) {

producer();

}

else if (args[0].equals("consumer")) {

consumer();

}

else printUsage();

}

public static void producer() {

@SuppressWarnings("resource")

Jedis jedis = new Jedis( REDIS\_HOST );

while(true) {

String newElement=UUID.randomUUID().toString();

jedis.rpush( REDIS\_LIST, newElement );

/\* limpiamos pantalla \*/

System.out.print("\033[H\033[2J");

System.out.flush();

System.out.println("REDIS Productor");

System.out.println("===============");

System.out.println("");

System.out.println("Total de elementos: "+jedis.llen( REDIS\_LIST ));

System.out.println("Elemento: "+jedis.lrange( REDIS\_LIST , 0, -1));

try {

Thread.sleep(1000);

} catch (Exception ex) { ex.printStackTrace(); }

}

}

public static void consumer() {

Jedis jedis = new Jedis( REDIS\_HOST );

/\* limpiamos pantalla \*/

System.out.print("\033[H\033[2J");

System.out.flush();

System.out.println("REDIS Consumidor");

System.out.println("================");

while(true) {

List<String> element=jedis.blpop( 0, REDIS\_LIST );

System.out.print("Elemento leí­do: "+element);

System.out.println(" ("+jedis.llen( REDIS\_LIST )+" Pendientes)");

try {

Thread.sleep(1500);

} catch (Exception ex) { ex.printStackTrace(); }

}

}

public static void printUsage() {

System.out.println("");

System.out.println("USAGE: java -jar Redis.jar <producer/consumer>");

System.out.println("");

}

}

1. Para arrancar la solución, una vez compilada se ejecuta:

# Para arrancar la aplicación como productor  
java -jar lib/build/libs/lib.jar producer  
  
#Para arrancar la aplicación como consumidor  
java -jar lib/build/libs/lib.jar consumer

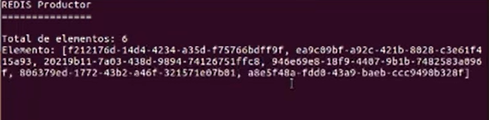


Imagen Productor\*

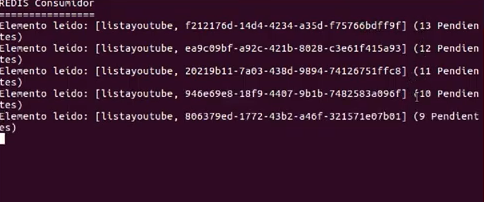


Imagen Consumior\*

Nota: Imagen Consumidor e Imagen Productor tomadas de <https://www.youtube.com/watch?v=mM_SROJZGwQ>

**Referencias:**

https://www.albertcoronado.com/2019/05/07/escalando-servicios-con-colas-redis/

1. **Extras**

Como extra en esta actividad se tuvo en cuenta el proceso de autorización con usuarios, el cual tuvo retos nuevos en la parte de identificación, manejo del ACL y modificación de lo programas en Python.

1. **Conclusiones**

Como objetivo pasado el equipo se había propuesto a cumplir las siguientes características:

Para cumplir con el propósito del proyecto se va a crear una base de datos escrita en el lenguaje de python, se tratará de implementar una base de datos no relacional basada distribuida en tabla de hash, para el enrutamiento de una petición se utilizará un routing tier con un zookeeper que ayude con el trabajo. La replicación será dada por un sistema de líder único y la comunicación entre ellos será asincrónica.

El front end de la aplicación será definido después. La comunicación entre la api del cliente y el servidor sería por protocolo http.

Sin embargo, luego de la experiencia de la práctica se identificaron algunas dificultades que evitar como la reinserción de un nuevo nodo, además el redireccionamiento de los datos al apagarse un nodo. Además se desea también trabajar con las transacciones simples definidas también en este trabajo de get y set.

**REFERENCIAS**

* <https://redis.io/topics/quickstart>
* <https://redis.io/commands/auth>
* <https://redis.io/topics/acl>
* <http://barbra-coco.dyndns.org/yuri/seven/seven2.pdf>
* <https://shawn-shi.medium.com/how-to-install-redis-on-ec2-server-for-fast-in-memory-database-f30c3ef8c35e>
* <https://redis-py.readthedocs.io/en/latest/>
* <https://blog.appliedinformaticsinc.com/redis-authentication-using-scrapy-and-python/>
* <https://redis.io/topics/cluster-spec>
* https://github.com/Grokzen/redis-py-cluster/blob/master/examples/incr-test-writer.py