LABORATORIO 2

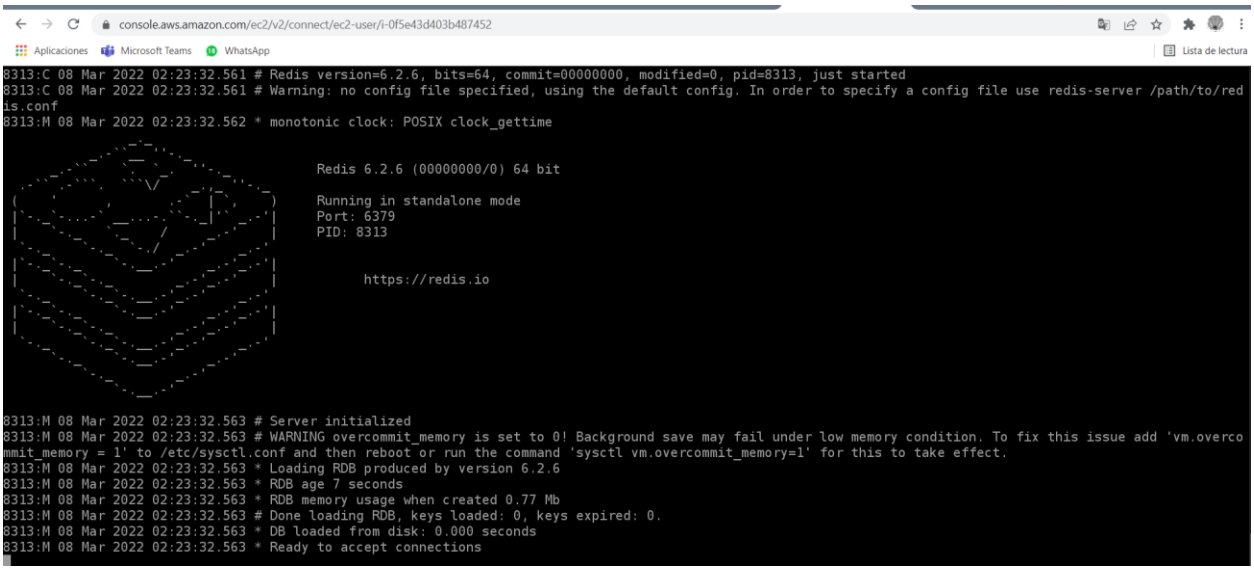
ADRIÁN ALBERTO GUTIÉRREZ LEAL

TRABAJO DE CLASE – LABORATORIO 2

EDWIN NELSON MONTOYA MUNERA

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA
TÓPICOS ESPECIALES EN TELEMÁTICA
MEDELLÍN
2022

1.En principio, se crea una instancia EC2 en AWS, en la que, mediante los comandos presentes en el tutorial, se instala Redis en la máquina.

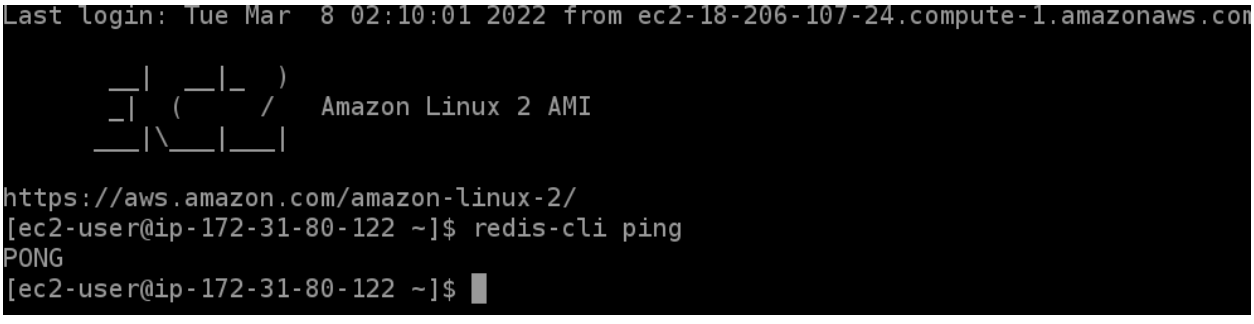


NOTA: Es necesario tener instalado el compilador de C para instalar las dependencias de Redis. Puede instalarse con el siguiente comando:

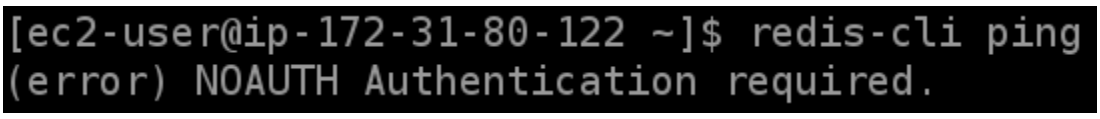
$sudo yum install gcc

Luego, podemos comprobar que esté bien instalada al hacer ping en el cliente de Redis, mediante el comando

$redis-cli ping



Al descomentar la directiva ‘requirepass’ en el archivo redis.conf, podemos asignar una contraseña que nos permita autenticarnos antes de realizar tareas en la base de datos.



Al autenticarnos, es possible utilizar el cliente.

```
[ec2-user@ip-172-31-80-122 ~]$ redis-cli
127.0.0.1:6379> AUTH foobared
OK
127.0.0.1:6379> ping
PONG
```

NOTA: Para probar que todo funciona correctamente, debe iniciarse el servidor Redis desde otra terminal para poder iniciar el cliente. El comando es: $redis-server

2. Desde el cliente, podemos realizar operaciones CRUD, tal y como las expuestas en el capítulo 8 del libro “Siete bases de datos en siete semanas”.

Las más básicas, serían SET y GET.

```
127.0.0.1:6379> SET 7wks http://www.sevenweeks.org/
OK
127.0.0.1:6379> GET 7wks
"http://www.sevenweeks.org/"
```

También, pueden realizarse múltiples tareas en un mismo comando.

```
127.0.0.1:6379> MSET gog http://www.google.com.ezproxy.eafit.edu.co yah http://www.yahoo.com
OK
127.0.0.1:6379> MGET gog yah
1) "http://www.google.com.ezproxy.eafit.edu.co"
2) "http://www.yahoo.com"
```

Asimismo, es posible alterar valores almacenados, como incrementar un valor numérico.

```
127.0.0.1:6379> SET count 2
OK
127.0.0.1:6379> INCR count
(integer) 3
127.0.0.1:6379> GET count
"3"
```

Otros comandos, como es el caso de MULTI, permiten realizar varias acciones distintas, que serán encoladas y ejecutadas en orden.

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> SET prag http://pragprog.com
QUEUED
127.0.0.1:6379(TX)> INCR count
QUEUED
127.0.0.1:6379(TX)> EXEC
1) OK
2) (integer) 4
127.0.0.1:6379>
```

Es posible almacenar un conjunto hasheado:

```
127.0.0.1:6379> HMSET user:luc name "Luc" password s3cret
OK
127.0.0.1:6379> HVALS user:luc
1) "Luc"
2) "s3cret"
```

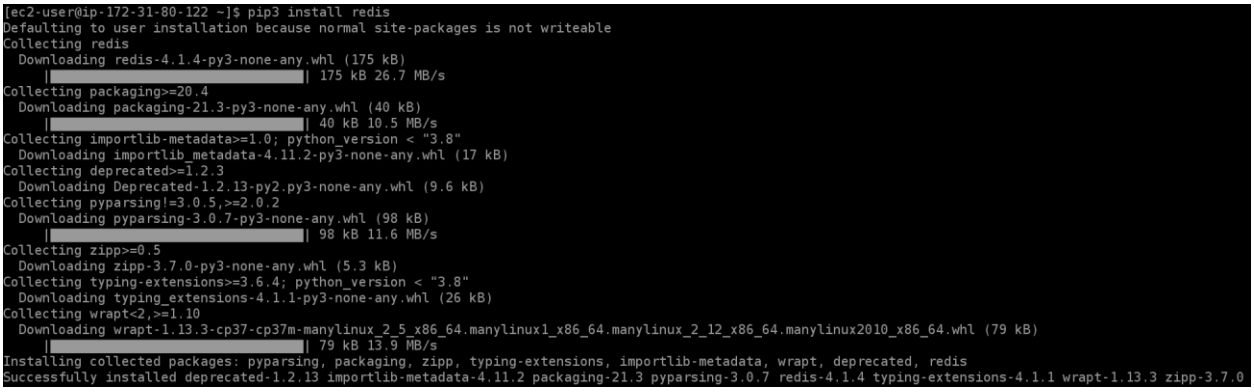
O cambiar de bases de datos mediante su ID

```
127.0.0.1:6379> SET greeting hello
OK
127.0.0.1:6379> GET greeting
"hello"
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]>  GET greeting
(nil)
```

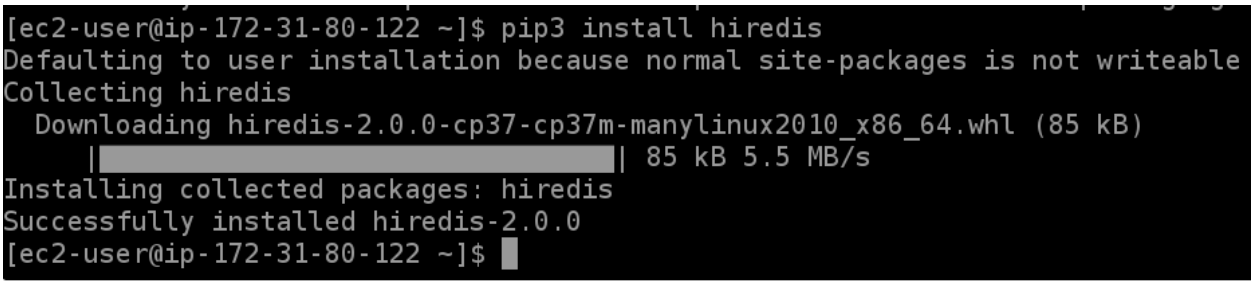
```
127.0.0.1:6379[2]> SELECT 0
OK
127.0.0.1:6379> GET greeting
"hello"
127.0.0.1:6379> MOVE greeting 2
(integer) 1
127.0.0.1:6379> GET greeting
(nil)
127.0.0.1:6379> SELECT 2
OK
127.0.0.1:6379[2]> GET greeting
"hello"
```

3. Una vez instalado Python 3 (incluido con Amazon Linux AMI), se procede a instalar dependencias para el correcto funcionamiento de Redis.

$pip3 install redis



$pip3 install hiredis



Una vez hecho esto, es posible acceder a la base de datos desde programas escritos en Python. Para efectos de este laboratorio, se probó con los scripts propuestos en el repositorio de la materia (https://github.com/aploetz/packt)

```
[ec2-user@ip-172-31-80-122 redis]$ python3 redisHelloWorld.py
#!/usr/bin/python
from redis import StrictRedis
import sys

hostname="127.0.0.1"
password="foobared"

r = StrictRedis(host=hostname,port=6379,password=password,db=0)

def getPacktWelcome():
    #GET value stored in packt:welcome
    print("Displaying current welcome message...")
    value = r.get('packt:welcome')
    print("message = " + str(value))

def setPacktWelcome():
    #SET new value packt:welcome
    print("Writing \"Hello world from Python!\" to Redis...")
    r.set('packt:welcome','Hello world from Python!')

getPacktWelcome()
setPacktWelcome()
getPacktWelcome()
```

Nota: Los argumentos fueron quemados para facilitar la ejecución de pruebas, pues estos siempre serán los mismos durante este punto.

Al experimentar con operaciones CRUD (en un archivo Python adjunto, de autoría propia), podemos evidenciar que ambas operaciones, tanto desde Python como desde el cliente, afectan a la base de datos.

```
[root@ip-172-31-90-20 ec2-user]# python3 redisCRUD.py
password:
->foobared
Connect to a database
SET
key:
->my_key
value:
->something
Writing something to Redis...
```

De esta manera, podemos luego usar GET desde el cliente para observar lo que se guardó (y viceversa).

```
[root@ip-172-31-90-20 ec2-user]# redis-cli
127.0.0.1:6379> AUTH foobared
OK
127.0.0.1:6379> GET my_key
"something"
```

4. Para el último punto, se utiliza el comando $redis-cli -a foobared --cluster create <ip>:<port>, donde <ip> es la ip pública de cada uno de los 4 nodos y port será el mismo (6379) para cada una de las 4 máquinas:

```
[ec2-user@ip-172-31-80-122 ~]$ redis-cli -a foobared --cluster create 35.171.24.81:6379 52.54.116.39:6379 34.236.155.69:6379 107.20.96.66:6379
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
>>> Performing hash slots allocation on 4 nodes...
Master[0] -> Slots 0 - 4095
Master[1] -> Slots 4096 - 8191
Master[2] -> Slots 8192 - 12287
Master[3] -> Slots 12288 - 16383
M: ad1a5d3d698bf5ebec6d1cac45e94a6b726a48fb 35.171.24.81:6379
   slots:[0-4095] (4096 slots) master
M: f7e88b0c0c16cc40f49b4c648ca8c250373fdc4c 52.54.116.39:6379
   slots:[4096-8191] (4096 slots) master
M: 5324b4c38c915bc39e87ddf17ec8681ad2c36434 34.236.155.69:6379
   slots:[8192-12287] (4096 slots) master
M: cee2b57efc27bb0a2ede72a16d904c738f401219 107.20.96.66:6379
   slots:[12288-16383] (4096 slots) master
Can I set the above configuration? (type 'yes' to accept): yes
```

Al escribir ‘yes’ en la terminal, se unen los nodos en un cluster.

```
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
...........
```

Ahora, deberían poder realizarse operaciones CRUD con un balanceo entre los nodos, según una clave hasheada.

5. Conclusiones

Para el primer proyecto queremos crear una API (cuya especificación puede encontrarse en uno de los repositorios del equipo) y un cliente Python similar al implementado durante este laboratorio con StrictRedis, pensamos que al implementar, del CRUD, únicamente créate, read y update debería ser bastante sencillo, siendo la más retadora delete, dependiendo de los métodos de almacenamiento en disco escogidos.

Desde las especificaciones no funcionales, ciertamente podemos evidenciar que resulta imperativo el uso de particionamiento (aunque estático), mediante el uso de un algoritmo DHT simple. La replicación no es una prioridad, aunque se entiende su utilidad, no debería ser el centro de atención con respecto a otras funciones. Nuestro proyecto debe tomar especial énfasis en la propiedad de consistencia de datos, pues solo habrán 3 nodos y es muy probable que no sean replicados. La tolerancia a fallos, por su parte, no será totalmente garantizada (dada la asincronía, pocos nodos e interdependencia entre ellos por motivos de licencia académica), pero no será ignorada del todo.