PRÁCTICA REDIS

Sistemas Distribuidos

Práctica Redis

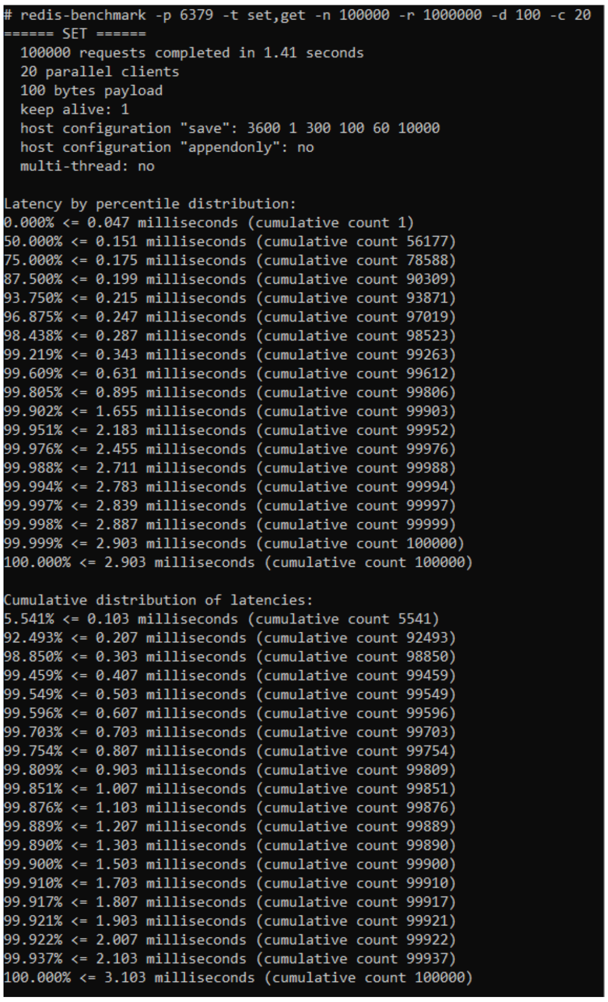
Implementar diferentes versiones de una base de datos Redis y realiza benchmarks

Jaime Arana, Javier Álvarez

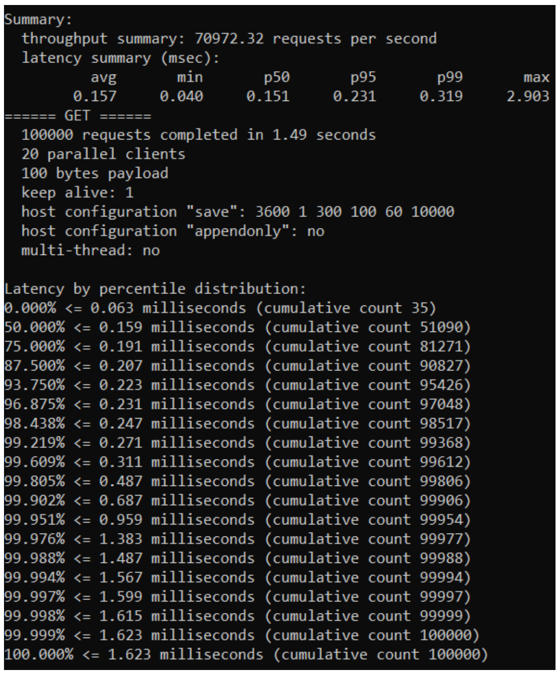
Práctica Redis 1.1 – Instalar REDIS stand-alone sobre Docker

A lo largo de la práctica se han ido haciendo capturas de los apartados donde hay resultados que analizar y no de los procesos de instalación o configuración ya que los comandos están dados en la práctica.

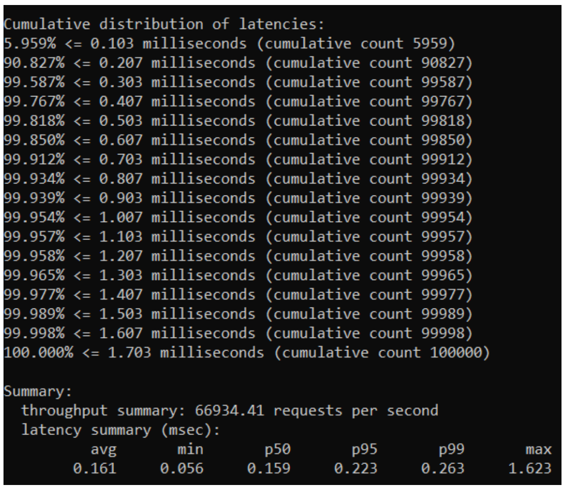
Benchmark desde contenedor redis (redis-cli)



```
# redis-benchmark -p 6379 -t set,get -n 100000 -r 1000000 -d 100 -c 20
====== SET ======
  100000 requests completed in 1.41 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1
  host configuration "save": 3600 1 300 100 60 10000
  host configuration "appendonly": no
  multi-thread: no

Latency by percentile distribution:
0.000% <= 0.047 milliseconds (cumulative count 1)
50.000% <= 0.151 milliseconds (cumulative count 56177)
75.000% <= 0.175 milliseconds (cumulative count 78588)
87.500% <= 0.199 milliseconds (cumulative count 90309)
93.750% <= 0.215 milliseconds (cumulative count 93871)
96.875% <= 0.247 milliseconds (cumulative count 97019)
98.438% <= 0.287 milliseconds (cumulative count 98523)
99.219% <= 0.343 milliseconds (cumulative count 99263)
99.609% <= 0.631 milliseconds (cumulative count 99612)
99.805% <= 0.895 milliseconds (cumulative count 99806)
99.902% <= 1.655 milliseconds (cumulative count 99903)
99.951% <= 2.183 milliseconds (cumulative count 99952)
99.976% <= 2.455 milliseconds (cumulative count 99976)
99.988% <= 2.711 milliseconds (cumulative count 99988)
99.994% <= 2.783 milliseconds (cumulative count 99994)
99.997% <= 2.839 milliseconds (cumulative count 99997)
99.998% <= 2.887 milliseconds (cumulative count 99999)
99.999% <= 2.903 milliseconds (cumulative count 100000)
100.000% <= 2.903 milliseconds (cumulative count 100000)

Cumulative distribution of latencies:
5.541% <= 0.103 milliseconds (cumulative count 5541)
92.493% <= 0.207 milliseconds (cumulative count 92493)
98.850% <= 0.303 milliseconds (cumulative count 98850)
99.459% <= 0.407 milliseconds (cumulative count 99459)
99.549% <= 0.503 milliseconds (cumulative count 99549)
99.596% <= 0.607 milliseconds (cumulative count 99596)
99.703% <= 0.703 milliseconds (cumulative count 99703)
99.754% <= 0.807 milliseconds (cumulative count 99754)
99.809% <= 0.903 milliseconds (cumulative count 99809)
99.851% <= 1.007 milliseconds (cumulative count 99851)
99.876% <= 1.103 milliseconds (cumulative count 99876)
99.889% <= 1.207 milliseconds (cumulative count 99889)
99.890% <= 1.303 milliseconds (cumulative count 99890)
99.900% <= 1.503 milliseconds (cumulative count 99900)
99.910% <= 1.703 milliseconds (cumulative count 99910)
99.917% <= 1.807 milliseconds (cumulative count 99917)
99.921% <= 1.903 milliseconds (cumulative count 99921)
99.922% <= 2.007 milliseconds (cumulative count 99922)
99.937% <= 2.103 milliseconds (cumulative count 99937)
100.000% <= 3.103 milliseconds (cumulative count 100000)
```

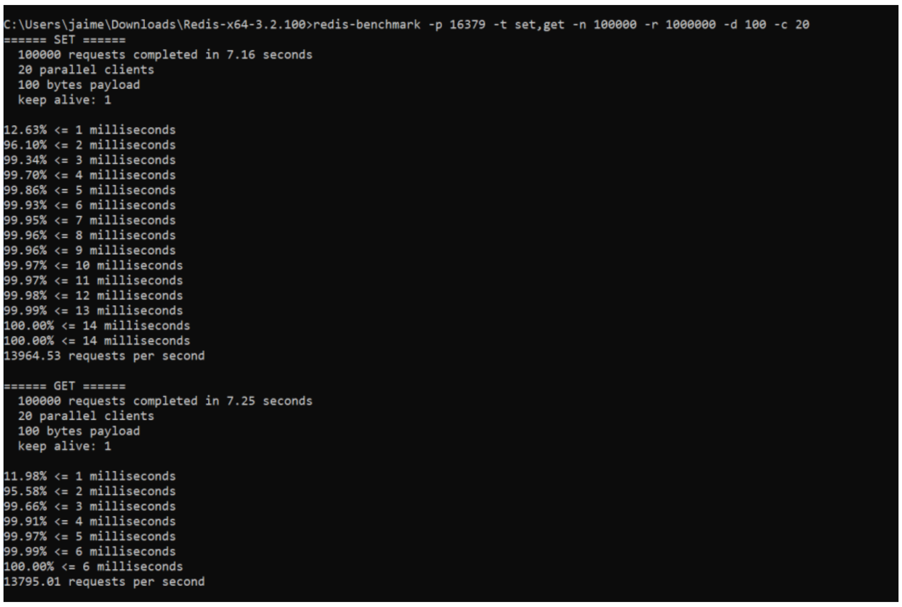


```
Summary:
  throughput summary: 70972.32 requests per second
  latency summary (msec):
          avg       min       p50       p95       p99       max
        0.157     0.040     0.151     0.231     0.319     2.903
====== GET ======
  100000 requests completed in 1.49 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1
  host configuration "save": 3600 1 300 100 60 10000
  host configuration "appendonly": no
  multi-thread: no

Latency by percentile distribution:
0.000% <= 0.063 milliseconds (cumulative count 35)
50.000% <= 0.159 milliseconds (cumulative count 51090)
75.000% <= 0.191 milliseconds (cumulative count 81271)
87.500% <= 0.207 milliseconds (cumulative count 90827)
93.750% <= 0.223 milliseconds (cumulative count 95426)
96.875% <= 0.231 milliseconds (cumulative count 97048)
98.438% <= 0.247 milliseconds (cumulative count 98517)
99.219% <= 0.271 milliseconds (cumulative count 99368)
99.609% <= 0.311 milliseconds (cumulative count 99612)
99.805% <= 0.487 milliseconds (cumulative count 99806)
99.902% <= 0.687 milliseconds (cumulative count 99906)
99.951% <= 0.959 milliseconds (cumulative count 99954)
99.976% <= 1.383 milliseconds (cumulative count 99977)
99.988% <= 1.487 milliseconds (cumulative count 99988)
99.994% <= 1.567 milliseconds (cumulative count 99994)
99.997% <= 1.599 milliseconds (cumulative count 99997)
99.998% <= 1.615 milliseconds (cumulative count 99999)
99.999% <= 1.623 milliseconds (cumulative count 100000)
100.000% <= 1.623 milliseconds (cumulative count 100000)
```



```
Cumulative distribution of latencies:
5.959% <= 0.103 milliseconds (cumulative count 5959)
90.827% <= 0.207 milliseconds (cumulative count 90827)
99.587% <= 0.303 milliseconds (cumulative count 99587)
99.767% <= 0.407 milliseconds (cumulative count 99767)
99.818% <= 0.503 milliseconds (cumulative count 99818)
99.850% <= 0.607 milliseconds (cumulative count 99850)
99.912% <= 0.703 milliseconds (cumulative count 99912)
99.934% <= 0.807 milliseconds (cumulative count 99934)
99.939% <= 0.903 milliseconds (cumulative count 99939)
99.954% <= 1.007 milliseconds (cumulative count 99954)
99.957% <= 1.103 milliseconds (cumulative count 99957)
99.958% <= 1.207 milliseconds (cumulative count 99958)
99.965% <= 1.303 milliseconds (cumulative count 99965)
99.977% <= 1.407 milliseconds (cumulative count 99977)
99.989% <= 1.503 milliseconds (cumulative count 99989)
99.998% <= 1.607 milliseconds (cumulative count 99998)
100.000% <= 1.703 milliseconds (cumulative count 100000)

Summary:
  throughput summary: 66934.41 requests per second
  latency summary (msec):
          avg       min       p50       p95       p99       max
        0.161     0.056     0.159     0.223     0.263     1.623
```

<u>Benchmark desde el terminal (host)</u>



```
C:\Users\jaime\Downloads\Redis-x64-3.2.100>redis-benchmark -p 16379 -t set,get -n 100000 -r 1000000 -d 100 -c 20
====== SET ======
  100000 requests completed in 7.16 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

12.63% <= 1 milliseconds
96.10% <= 2 milliseconds
99.34% <= 3 milliseconds
99.70% <= 4 milliseconds
99.86% <= 5 milliseconds
99.93% <= 6 milliseconds
99.95% <= 7 milliseconds
99.96% <= 8 milliseconds
99.96% <= 9 milliseconds
99.97% <= 10 milliseconds
99.97% <= 11 milliseconds
99.98% <= 12 milliseconds
99.99% <= 13 milliseconds
100.00% <= 14 milliseconds
100.00% <= 14 milliseconds
13964.53 requests per second

====== GET ======
  100000 requests completed in 7.25 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

11.98% <= 1 milliseconds
95.58% <= 2 milliseconds
99.66% <= 3 milliseconds
99.91% <= 4 milliseconds
99.97% <= 5 milliseconds
99.99% <= 6 milliseconds
100.00% <= 6 milliseconds
13795.01 requests per second
```

Como se puede ver en los resultados al hacer el benchmark directamente desde el contenedor de Redis es capaz de hacer el mismo número de peticiones en menor tiempo. Este resultado tiene sentido ya que al estar directamente dentro del contenedor no tiene que hacer el proceso de entrada en el contenedor para cada petición.

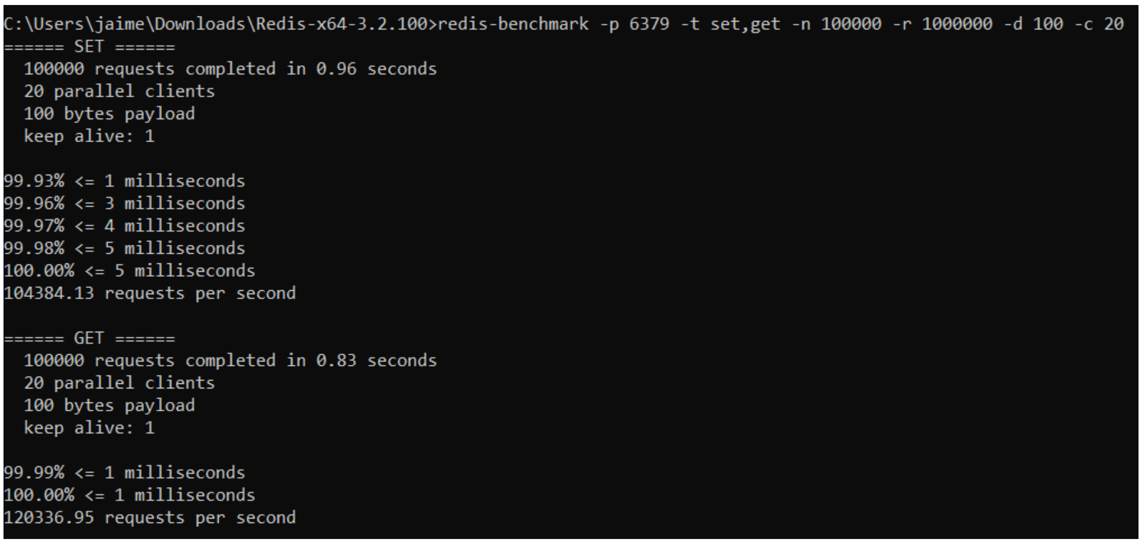
nº peticiones	**Tiempo** dentro del contenedor	**Tiempo** desde el host
100000	1.49 seg	7.16 seg

Práctica Redis 1.2 – Instalar REDIS stand-alone sobre kubernetes

<u>¿Qué tipo de despliegue se está haciendo?</u>

En este caso se trata de un tipo de despliegue de clúster de solo una máquina, tratándose de una instalación en minikube.

<u>Benchmark kubernetes</u>

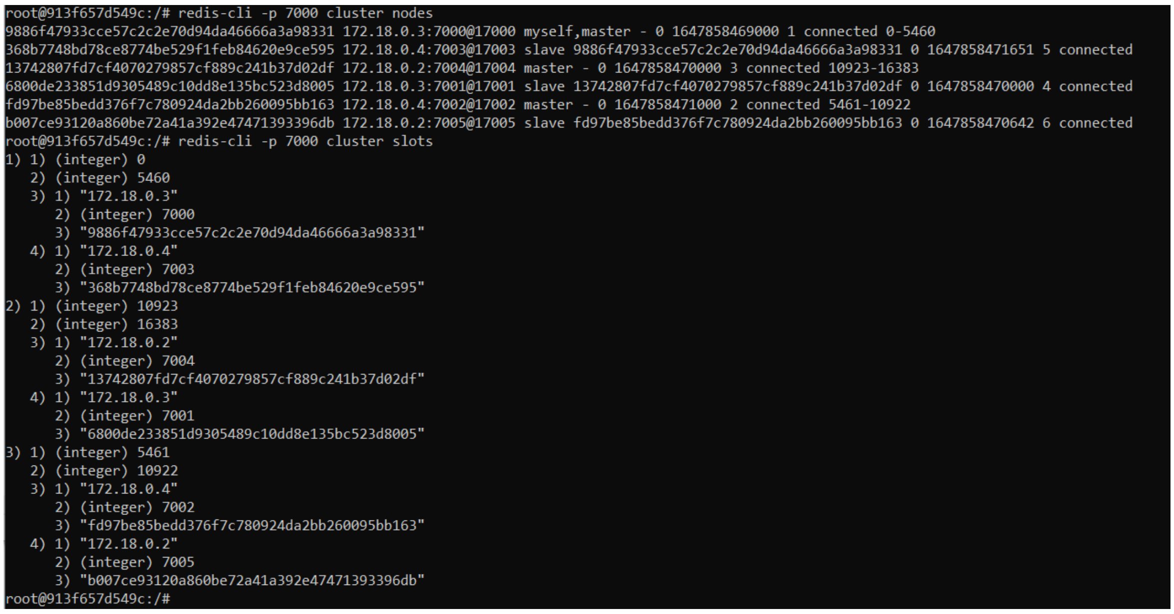


```
C:\Users\jaime\Downloads\Redis-x64-3.2.100>redis-benchmark -p 6379 -t set,get -n 100000 -r 1000000 -d 100 -c 20
====== SET ======
  100000 requests completed in 0.96 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.93% <= 1 milliseconds
99.96% <= 3 milliseconds
99.97% <= 4 milliseconds
99.98% <= 5 milliseconds
100.00% <= 5 milliseconds
104384.13 requests per second

====== GET ======
  100000 requests completed in 0.83 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.99% <= 1 milliseconds
100.00% <= 1 milliseconds
120336.95 requests per second
```

En este caso el resultado no concuerda con lo esperado, ya que el tiempo empleado para hacer las 100000 requests es demasiado bajo. No obstante, tras haber comprobado todos los parámetros y haber repetido el benchmark varias veces, obtenemos el mismo resultado. Así que el resultado, aunque no esperado, si parece correcto.

Práctica Redis 1.4 – Instalar clúster dockerizados de REDIS

En las gráficas inferiores se presentan los resultados tras haber ejecutado los comandos para ver los nodos del clúster y los slots.

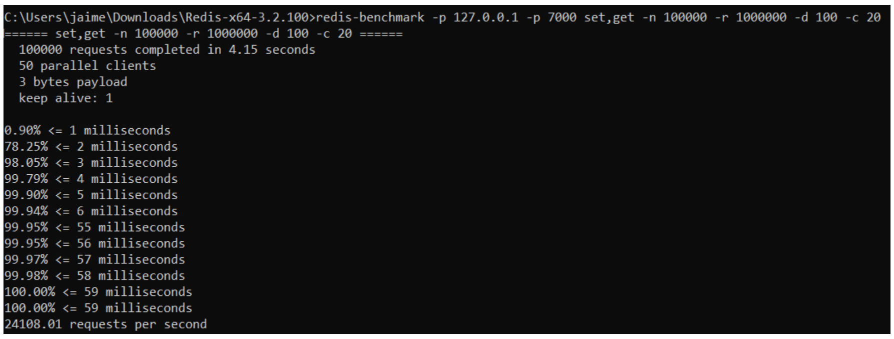


```
root@913f657d549c:/# redis-cli -p 7000 cluster nodes
9886f47933cce57c2c2e70d94da46666a3a98331 172.18.0.3:7000@17000 myself,master - 0 1647858469000 1 connected 0-5460
368b7748bd78ce8774be529f1feb84620e9ce595 172.18.0.4:7003@17003 slave 9886f47933cce57c2c2e70d94da46666a3a98331 0 1647858471651 5 connected
13742807fd7cf4070279857cf889c241b37d02df 172.18.0.2:7004@17004 master - 0 1647858470000 3 connected 10923-16383
6800de233851d9305489c10dd8e135bc523d8005 172.18.0.3:7001@17001 slave 13742807fd7cf4070279857cf889c241b37d02df 0 1647858470000 4 connected
fd97be85bedd376f7c780924da2bb260095bb163 172.18.0.4:7002@17002 master - 0 1647858471000 2 connected 5461-10922
b007ce93120a860be72a41a392e47471393396db 172.18.0.2:7005@17005 slave fd97be85bedd376f7c780924da2bb260095bb163 0 1647858470642 6 connected
root@913f657d549c:/# redis-cli -p 7000 cluster slots
1) 1) (integer) 0
   2) (integer) 5460
   3) 1) "172.18.0.3"
      2) (integer) 7000
      3) "9886f47933cce57c2c2e70d94da46666a3a98331"
   4) 1) "172.18.0.4"
      2) (integer) 7003
      3) "368b7748bd78ce8774be529f1feb84620e9ce595"
2) 1) (integer) 10923
   2) (integer) 16383
   3) 1) "172.18.0.2"
      2) (integer) 7004
      3) "13742807fd7cf4070279857cf889c241b37d02df"
   4) 1) "172.18.0.3"
      2) (integer) 7001
      3) "6800de233851d9305489c10dd8e135bc523d8005"
3) 1) (integer) 5461
   2) (integer) 10922
   3) 1) "172.18.0.4"
      2) (integer) 7002
      3) "fd97be85bedd376f7c780924da2bb260095bb163"
   4) 1) "172.18.0.2"
      2) (integer) 7005
      3) "b007ce93120a860be72a41a392e47471393396db"
root@913f657d549c:/#
```

Es importante ver o comprender que para que el clúster sea útil y sirva, los nodos esclavos y maestros no pueden pertenecer a la misma máquina.

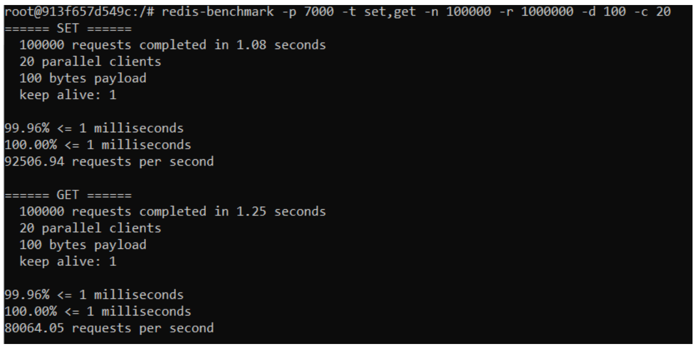
En este caso la configuración es correcta ya que, por ejemplo, el nodo maestro de la máquina Ubun1 está conectado al esclavo de la máquina Ubun2.

<u>Benchmark desde el host a Ubun1</u>



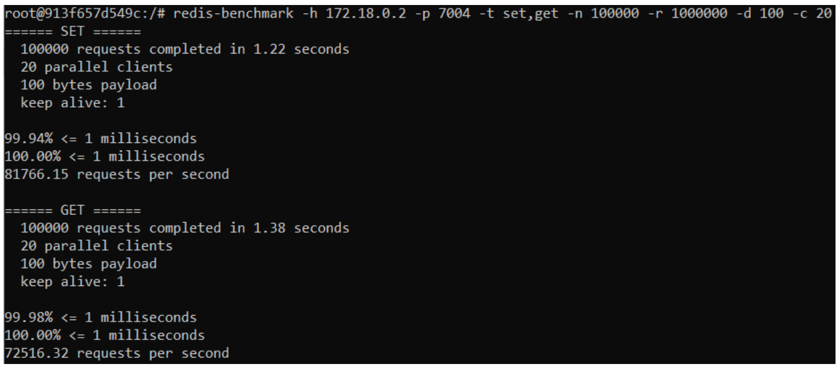
```
C:\Users\jaime\Downloads\Redis-x64-3.2.100>redis-benchmark -p 127.0.0.1 -p 7000 set,get -n 100000 -r 1000000 -d 100 -c 20
====== set,get -n 100000 -r 1000000 -d 100 -c 20 ======
  100000 requests completed in 4.15 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

0.90% <= 1 milliseconds
78.25% <= 2 milliseconds
98.05% <= 3 milliseconds
99.79% <= 4 milliseconds
99.90% <= 5 milliseconds
99.94% <= 6 milliseconds
99.95% <= 55 milliseconds
99.95% <= 56 milliseconds
99.97% <= 57 milliseconds
99.98% <= 58 milliseconds
100.00% <= 59 milliseconds
100.00% <= 59 milliseconds
24108.01 requests per second
```

<u>Benchmark desde Ubun1 a propia instancia</u>



```
root@913f657d549c:/# redis-benchmark -p 7000 -t set,get -n 100000 -r 1000000 -d 100 -c 20
====== SET ======
  100000 requests completed in 1.08 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.96% <= 1 milliseconds
100.00% <= 1 milliseconds
92506.94 requests per second

====== GET ======
  100000 requests completed in 1.25 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.96% <= 1 milliseconds
100.00% <= 1 milliseconds
80064.05 requests per second
```

<u>Benchmark desde Ubun1 a Ubun3</u>



```
root@913f657d549c:/# redis-benchmark -h 172.18.0.2 -p 7004 -t set,get -n 100000 -r 1000000 -d 100 -c 20
====== SET ======
  100000 requests completed in 1.22 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.94% <= 1 milliseconds
100.00% <= 1 milliseconds
81766.15 requests per second

====== GET ======
  100000 requests completed in 1.38 seconds
  20 parallel clients
  100 bytes payload
  keep alive: 1

99.98% <= 1 milliseconds
100.00% <= 1 milliseconds
72516.32 requests per second
```

Práctica Redis 1.5 – Tipos de datos en REDIS

Este último apartado se ha solucionado en el fichero adjunto **redis.ipynb** y para la sección de Publish/Subscribe hay, además, otro archivo .pynb que hace de consumidor de los channels creados.