ROW SOCKET 기반 로드밸런서 가이드 라인(CPP) : Ubuntu 환경

server

```
kjs@kjs:~/iot/test/shared$ ./chat_serv 10000
```

```
kjs@kjs:~/iot/test/shared$ ./chat_serv 10001
```

```
kjs@kjs:~/iot/test/shared$ ./chat_serv 10002
```

Loadbalancer (입력한 서버들 상태 주기적으로 체크 -> 헬스 체크)

```
kjs@kjs-desktop:~/iot/lasttest/cpp$ sudo ./loadbalancerLobin 172.18.136.132 172.18.136.132 10000 172.18.136.132 10001 172.18.136.132 10002
srcIP : 172.18.136.132
IP : <172.18.136.132>, Port : <10001> Server is alive
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
IP : <172.18.136.132>, Port : <10001> Server is alive
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
```

Client(nickname : test1 ->test6 순서대로 접속하고 대화를 입력)

->사진처럼 나오려면 각 client 연결 순서를 지켜야 하지만 client들 끼리는 연결 순서 상관 없이 모두 정상 작동함

->라운드 로빈 방식에 의해 server1 : (test1, test4), server2: (test2, test5), server3: (test3, test6) 이런식으로 각 서버에 할당되어서 같은 서버에 할당된 클라이언트 들은 서로 대화가 공유됨

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test1
hello
[test1] hello
```

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test2
hi
[test2] hi
```

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test3
hi
[test3] hi
```

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test1
hello
[test1] hello
[test4] hi test4
hello test4
[test1] hello test4
```

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test2
hi
[test2] hi
[test5] hi test2
oh test5 good to see you
[test2] oh test5 good to see you
```

```
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test3
hi
[test3] hi
[test6] hello tes3
hi test6
[test3] hi test6
```

```
./chat_clnt
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test4
hi test4
[test4] hi test4
[test1] hello test4
```

```
./chat_clnt
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test5
hi test2
[test5] hi test2
[test2] oh test5 good to see you
```

```
./chat_clnt
kjs@kjs-desktop:~/iot/lasttest/shared$ ./chat_clnt
 172.18.136.132 20000 test6
hello tes3
[test6] hello tes3
[test3] hi test6
```

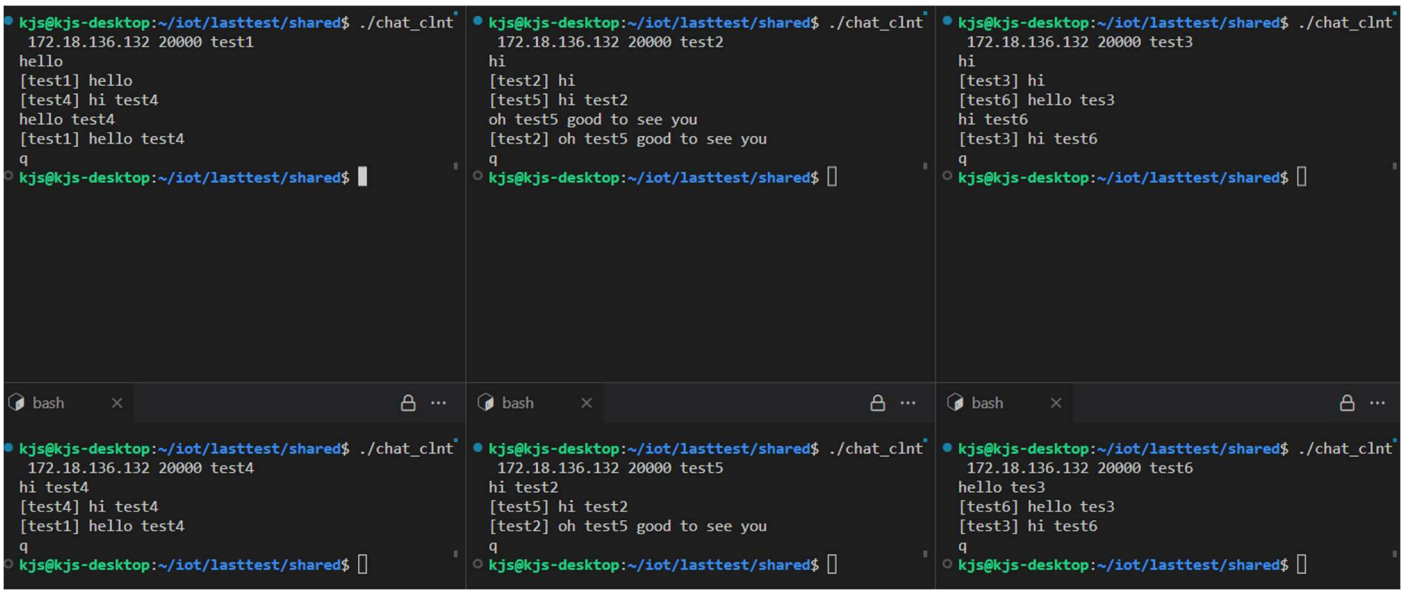
Server(client 커넥션과 주기적인 서버 헬스체크를 위한 로드밸런서의 커넥션 후 바로 컨넥션 종료 의해 서버 터미널창이 지저분해지는 부분은 개선이 필요)

```
client 4 terminated
Connected client IP: 172.18.136.132
client 5 terminated
Connected client IP: 172.18.136.132
client 4 terminated
Connected client IP: 172.18.136.132
client 5 terminated
Connected client IP: 172.18.136.132
```

```
Connected client IP: 172.18.136.132
client 5 terminated
Connected client IP: 172.18.136.132
client 4 terminated
Connected client IP: 172.18.136.132
client 5 terminated
Connected client IP: 172.18.136.132
client 4 terminated
```

```
Connected client IP: 172.18.136.132
client 4 terminated
Connected client IP: 172.18.136.132
client 5 terminated
Connected client IP: 172.18.136.132
client 4 terminated
Connected client IP: 172.18.136.132
client 5 terminated
```

Client(nickname : q를 누르고 모든 client 종료)



위의 순서대로 종료 할 때 마다 loadbalancer는 client가 종료됨을 인식하고 종료된 client에게 할당 됐던 NAT(Network address translation) table의 row를 삭제함

->터미널에 현재 남아있는 NAT table row 개수를 보여줌

row ex: (client Ip, port : client와 연결된 server Ip, port)

```
client <172.18.136.132,42046> term
inated
 Current NAT table length : 5
```

```
 client <172.18.136.132,42048> term
 inated
  Current NAT table length : 4
```

```
client <172.18.136.132,42064> term
inated
 Current NAT table length : 3
```

```
 client <172.18.136.132,42080> term
 inated
  Current NAT table length : 2
```

```
client <172.18.136.132,42096> term
inated
 Current NAT table length : 1
```

```
client <172.18.136.132,49268> term
inated
 Current NAT table length : 0
```

로드밸런서는 클라이언트 부하 할당을 하는 동시에 주기적으로(5초) 서버들에게 커넥션을 시도해 서버 상태를 체크함(헬스체크)

-> 만약 PORT 10001 서버가 shutdown되면 최대 3번까지 헬스체크를 시도합니다.

```
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is bad
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is bad
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
```

-> 만약 3번 시도 안에 커넥션이 연결되지 않는다면 서버가 죽은 것으로 판단 하고 (bad -> dead) 죽은 서버로는 부하를 할당하지 않고 살아있는 서버로만 부하를 할당합니다.

```
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is bad
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is bad
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is bad
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is dead
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
```

-> 만약 죽은 서버가 다시 살아나서 헬스체크를 위한 커넥션에 성공하면 다시 살아난 서버에 부하 분산을 합니다.

```
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is dead
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
connection timeout: Operation now in progress
IP: <172.18.136.132>, Port : <10001> Server is dead
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
IP : <172.18.136.132>, Port : <10001> Server is alive
IP : <172.18.136.132>, Port : <10002> Server is alive
IP : <172.18.136.132>, Port : <10000> Server is alive
IP : <172.18.136.132>, Port : <10001> Server is alive
```

Docker Compose 환경

```
명령 프롬프트 - docker-compose  -f docker-compose.cpp.yml up
4lb-cpp_1   | IP : <172.20.0.5>, Port : <8090> Server is alive
4lb-cpp_1   | IP : <172.20.0.3>, Port : <8091> Server is alive
4lb-cpp_1   | IP : <172.20.0.2>, Port : <8092> Server is alive
4lb-cpp_1   | IP : <172.20.0.5>, Port : <8090> Server is alive
4lb-cpp_1   | IP : <172.20.0.3>, Port : <8091> Server is alive
4lb-cpp_1   | IP : <172.20.0.2>, Port : <8092> Server is alive
4lb-cpp_1   | IP : <172.20.0.5>, Port : <8090> Server is alive
4lb-cpp_1   | IP : <172.20.0.3>, Port : <8091> Server is alive
4lb-cpp_1   | IP : <172.20.0.2>, Port : <8092> Server is alive
4lb-cpp_1   | IP : <172.20.0.3>, Port : <8091> Server is alive
4lb-cpp_1   | IP : <172.20.0.5>, Port : <8090> Server is alive
4lb-cpp_1   | IP : <172.20.0.2>, Port : <8092> Server is alive
4lb-cpp_1   | IP : <172.20.0.5>, Port : <8090> Server is alive
4lb-cpp_1   | IP : <172.20.0.3>, Port : <8091> Server is alive
4lb-cpp_1   | IP : <172.20.0.2>, Port : <8092> Server is alive
```