Server Deployment

3.4.5 Defining Load Balancer Timeout

Purpose

You can learn how to define timeout in the Load Balancer to avoid disconnection to your Liquid UI App.

Disconnection due to Load balancer timeout

When you use the F5 LTM for load balancing TCP connections, the default timeout is only 5 minutes – i.e. a TCP connection which does not send a packet for 301 seconds gets dropped. That’s not that long, unlike the 60 minutes (3600 seconds) . So the whole ‘using a TCP keepalive’ becomes even more important.

This problem persists when the Liquid UI App remain unused for longer than the TCP session timeout value of the load balancer. The load balancer silently drops the connections as being dead, but the client and server think they’re still up. Next time one side or the other decides to send a little traffic on one, you get a ‘broken socket’ error.

This is normal behaviour, and needs to be taken into consideration whenever you are building systems/applications which connect through load balancers.

Setting Load Balancer timeout to avoid disconnection.

This issue can be resolved by setting a TCP keepalive by sending a packet every minute, and you will never have a problem. In fact, you cannot implement such a sensible bunch to work on.

F5 LTM for Load balancing

There are two ways to know you have this issue and you will see this as RST packets being sent back to the client from the F5 (which do not come from the ‘real’ server) when they send traffic on a timed out connection, and also you can see the current connection timers using the ‘b conn client‘ command as shown below:

```
[dan@ltm01:Active] ~ # b conn client | grep tcp

CONN client 10.1.4.90:1873 server 10.31.8.10:https any protocol tcp ag
e 216 – client: 10.1.4.90:1873

CONN client 10.1.4.90:1876 server 10.31.8.10:https any protocol tcp ag
e 217 – client: 10.1.4.90:1876

CONN client 10.1.4.90:1877 server 10.31.8.10:https any protocol tcp ag
e 238 – client: 10.1.4.90:1877
```

You can see even more detail using ‘b conn client show all‘:

```
[dan@ltm01:Active] ~ # b conn client 10.1.4.90 show all

VIRTUAL 10.31.8.10:https NODE any6:any TYPE any

CLIENTSIDE 10.1.4.90:1927 10.31.8.10:https

(pkts,bits) in = (71, 16867) out = (122, 139083)

SERVERSIDE any6:any any6:any

(pkts,bits) in = (0, 0) out = (0, 0)

PROTOCOL tcp UNIT 1 IDLE 83 (300) LASTHOP 8 00:19:a9:f7:c0:00
```

So you can now see what the actual timeout value is for this connection (83 seconds used from a 300 second timer in this case). This is particularly hand as it shows you if your ‘fix’ has actually taken.

If you are a network expert, you can fix the F5 with an irule applied to the virtual server as shown below:

when SERVER_CONNECTED { IP::idle_timeout 3600 }

This simply increases the timeout to 1 hr and you can adjust the timeout as per your requirement. You can actually be quite granular, and set different values for different protocols. You can check the always useful http://devcentral.f5.com site for more detail, or see this excellent post.

To see the change :

```
[dan@ltm01:Active] ~ # b conn client 10.1.4.90 show all

VIRTUAL 10.31.8.10:https NODE any6:any TYPE any

CLIENTSIDE 10.1.4.90:1943 10.31.8.10:https

(pkts,bits) in = (83, 18157) out = (165, 188758)

SERVERSIDE any6:any any6:any

(pkts,bits) in = (0, 0) out = (0, 0)

PROTOCOL tcp UNIT 1 IDLE 4 (3600)

LASTHOP 8 00:19:a9:f7:c0:00
```

Server Deployment

The irules applied to virtual server doesnot immediately effect the current connections. You can see the Server Connection is invoked when a new TCP connection is set up and the F5 makes backend connection to the server.

Unique solution ID: #2156
Author: sarvani.kusuri@guixt.com
Last update: 2019-08-02 09:19