CodeShare Refresh: HTTP Session Limit

Colin Walker, 2011-27-12

The iRules CodeShare on DevCentral is an amazingly powerful, diverse collection of iRules that perform a myriad of tasks ranging from credit card scrubbing to form based authentication to, as in today's example, limiting the number of HTTP sessions allowed. While the codeshare is outstanding, it is a collection of code that has been contributed over the last several years. As such, some of it is written for older versions, like 9.x, where we didn't have some of the powerful, efficient commands and tools that we do currently within iRules. That is where the idea for a CodeShare Refresh series came from...getting those older v9-esque rules moved into modern times with table commands, static namespace variables, out of band connections and all of the other added benefits that come along with the more modern codebase. We'll be digging through the CodeShare fishing out old rules and reviving them, then posting them back for future generations of DevCentral users to leverage. We'll also try to comment on why we're making the changes that we are, so you can see what has changed between versions in the process of us updating the code. With that, let's get started.

First I'll post the older snippet, then the updated version, ready to go into the wild in v11.x. The new rule in its entirety and the link to the older version can be found below.

Previously, in pre CMP compliant v9 iRules, it was relatively common place to set global variables. This is a big no-no now, as it demotes connections out of CMP, which is a large performance hit. So while the old iRule's RULE_INIT section looked like:

```
1: when RULE_INIT {
2:  set ::total_active_clients 0
3:  set ::max_active_clients 100
4:  log local0. "rule session_limit initialized: total/max: $::total_active_clients/$::max_active_clients"
5: }
```

The newer version updated for v11 looks like:

```
1: when RULE_INIT {
2:   set static::max_active_clients 100
3: }
```

Note the use of the static:: namespace. This is a place to safely store static information globally available form that will not interfere with CMP. These values are, as the namespace indicates, static, but it's extremely valuable in many cases like this one where we're setting a cap for the number of clients that we want to allow. Also note that there is no active clients counter at all, due to the fact that we've changed things later on in the iRule. As a result of this it made no sense to log the initialization line from the older iRule either, so we've trimmed the RULE_INIT event down a bit.

Next up, the first half of the HTTP_REQUEST event, in which the max_active_clients is compared to the current number of active clients.

First the v9 code from the CodeShare:

```
1: when HTTP_REQUEST {
2:  ;# test cookie presence
3:  if {[HTTP::cookie exists "ClientID"]} {
4:    set need_cookie 0
5:    set client_id [HTTP::cookie "ClientID"]
6:    ;# if cookie not present & connection limit not reached, set up client_id
7:  } else {
8:    if {$::total_active_clients < $::max_active_clients} {
9:      set need_cookie 1
10:     set client_id [format "%08d" [expr { int(100000000 * rand()) }]]
```

Now the v11 code:

```
1: when HTTP_REQUEST {
2:   # test cookie presence
3:   if {[HTTP::cookie exists "ClientID"]} {
```

```
    set need_cookie 0
    set client_id [HTTP::cookie "ClientID"]
    # if cookie not present & connection limit not reached, set up client_id
  } else {
    if {[table keys -subtable httplimit] < $static::max_active_clients} {
      set need_cookie 1
      set client_id [format "%08d" [expr { int(100000000 * rand()) }]]
```

The only change here is a pretty notable one: Out with global variables, in with session tables! Here we introduce the table command, which was released with v10, that gives us extremely efficient access to the session table. In this iRule all we need is a counter, so we're using a subtable called httplimit and adding a new record to that subtable for each session coming in. Then, with the table keys command we can quickly and efficiently count the number of rows in that table, which gives us the number of HTTP sessions currently active for this VIP. Note that the rest of the code stayed the same. There are many ways to do things in iRules, but I'm trying not to fiddle with the logic or execution of the rules in this series more than necessary to update them for the newer versions.

So now that we're using the table command to do the lookups, we should likely use it to set and increment the counter as well. That occurs in the other half of the HTTP_REQUEST event. First v9:

```
# Only count this request if it's the first on the TCP connection
if {[HTTP::request_num] == 1}{
  incr ::total_active_clients
}
;# otherwise redirect
else {
HTTP::redirect "http://sorry.domain.com/"
return

  }
}
```

Again you can see the use of the global variable, and the incr command. Next is the v11 update:

```
# Only count this request if it's the first on the TCP connection
if {[HTTP::request_num] == 1}{
  table set -subtable httplimit [IP::client_addr]:[TCP::client_port] "blocked"
  set timer [after 60000 -periodic { table lookup -subtable httplmit [IP::client_addr]:[TCP::client_port]
} else {
  HTTP::redirect "http://sorry.domain.com/"
  event CLIENT_CLOSED disable
  return
}
}
```

As you can see things here have changed quite a bit. First of all, here is the way we're using the table command to increment our counter. Rather than keeping a single counter, we are adding rows to the session table in a particular subtable, as I mentioned before. We're using the client's IP & port to give us a unique identifier for that client. This allows us to do the table keys lookup to count the number of active clients. We're also instantiating a timer here. Using the after -periodic command we are setting up a loop (non blocking) that will touch the entry we've just created every 60 seconds. This is because the entry in the session table has a timeout of 180 seconds, which is the default. Now...we could have made that entry permanent, but that's not what we want. When counting things using an event based structure it's important to take into account cases where a particular event might not fire. While it's rare, there are technically cases where the CLIENT_CLOSED event may not fire if circumstances are just right. In that case, using the old structure with just a simple counter, the count would be off and could drift. This timer, which you'll see in the last section is terminated in CLIENT_CLOSED along with removing the entry for this session in the table (effectively decrementing the counter), ensure that even if something wonky happens, the count will resolve and remain accurate. A bit of a concept to wrap around, but a solid one, and this introduces far less overhead than you'd gain back by moving this rule to CMP.

Also note that we're disabling the CLIENT_CLOSED event if the user is over their limit. This is to ensure that the counter for their IP/port combo isn't decremented.

Next is the HTTP_RESPONSE event, which remains entirely unchanged, so the v9 & v11 versions are the same:

```
when HTTP_RESPONSE {
 # insert cookie if needed
 if {$need_cookie == 1} {
   HTTP::cookie insert name "ClientID" value $client_id path "/"
 }
}
```

And last, but not least, the CLIENT_CLOSED event. First v9, with our simple counter and the nearly infamous incr -1:

```
when CLIENT_CLOSED {
 ;# decrement current connection counter for this client_id
 if {$::total_active_clients > 0} {
   incr ::total_active_clients -1
 }
}
```

And now the updated version for v11:

```
when CLIENT_CLOSED {
  # decrement current connection counter for this client_id
  after cancel $timer
  table delete -subtable httplimit [IP::client_addr]:[TCP::client_port]
}
```

The two main things to note here are that we're not doing an if, since we have confidence that we're not going to drop the counter below 0, since that's not possible with this method, and the way we're decrementing things. Note that we're not decrementing at all. We're deleting the row out of the subtable that represents the current HTTP Session. As such, it won't be counted on the next iteration, and poof...decremented counter. We're also dumping the periodic after that we spun up in the HTTP_REQUEST section to keep our entry pinned up in the session table until the session actually terminated.

So there you have it, a freshly updated version of the HTTP Session Limiting iRule out of the CodeShare. Hopefully this is helpful and we'll continue refreshing this valuable content for the ever expanding DevCentral community. Here is the complete v11 iRule, which can also be found in the CodeShare:

```
when RULE_INIT {
  set static::max_active_clients 100
}

when HTTP_REQUEST {
  # test cookie presence
  if {[HTTP::cookie exists "ClientID"]} {
    set need_cookie 0
    set client_id [HTTP::cookie "ClientID"]
    # if cookie not present & connection limit not reached, set up client_id
  } else {
    if {not ([table keys -subtable httplimit] > $static::max_active_clients)} {
      set need_cookie 1
      set client_id [format "%08d" [expr { int(100000000 * rand()) }]]

      # Only count this request if it's the first on the TCP connection
      if {[HTTP::request_num] == 1}{
        table set -subtable httplimit [IP::client_addr]:[TCP::client_port] "blocked"
        set timer [after 60000 -periodic { table lookup -subtable httplmit [IP::client_addr]:[TCP::client_port] }
      }
    } else {
      HTTP::redirect "http://sorry.domain.com/"
      event CLIENT_CLOSED disable
      return
    }
  }
}

when HTTP_RESPONSE {
  # insert cookie if needed
  if {$need_cookie == 1} {
    HTTP::cookie insert name "ClientID" value $client_id path "/"
  }
}

when CLIENT_CLOSED {
  # decrement current connection counter for this client_id
  after cancel $timer
  table delete -subtable httplimit [IP::client_addr]:[TCP::client_port]
}
```

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com