v.10 - iRules and the after command

Colin Walker, 2009-14-04

One of the requests that we've had for an addition to the functionality of iRules is a way to delay the execution of code. Basically, a timer. People want to be able to count the number of new connections with a given time period, they want to know if an application hasn't responded within x seconds, etc. In BIG-IP v10 these requests did not go unheard, and the new after command was added.

The after command already existed in the core TCL code, so we decided not write a completely custom command. Rather, we brought the after command into the iRules world, with the needed tweaks, testing and expansion to make it work the way we wanted it to. This type of functionality was seen as valuable enough to warrant the change, and I'm glad to see it make it into the product. People, myself included, were working around the lack of this functionality with loops and a repeated diffing of the start time of the loop and the current time. While functionally similar, this had many drawbacks. The largest of which is that it holds up processing and traffic passing until the loop completes. This is very, very bad. Thankfully the after command doesn't have the same problem. Let's look at the new command and what it can do.

First, a little bit about the command and how it was implemented. As I already said, we're using the "after" command straight out of TCL, so no new command name to learn for you TCLers. The after command runs in the context of the event that it was executed in, and maintains the variable scope of that context. This means that if you execute a script via after from with HTTP_REQUEST, you can use things HTTP::uri inside it safely. The context will remain the same. It also means that variables will carry over, which is a good thing.

Another good thing to note, which was an initial concern of mine when hearing that this was in the works, is that scripts scheduled to evaluate after a connection is aborted/closed are automatically cancelled. I was very glad to hear we wouldn't have to worry about managing and cancelling these scheduled scripts ourselves. The last thing I want is to have an after spin up on every new request only to leave a trail of things to be executed filling up the memory and processing once that connection is closed.

The other change worth noting is the -periodic option that's been added. This is to fulfill the: "I want to update a counter every x seconds" kind of case. We've seen these requests come in, and there are some very valid reasons for them in some cases. This new flag allows you to do just that. It can also do some pretty outstanding things if you bend it to your will just right. It certainly makes for some new posibilities, and in a much more performant manner than the previous loop and diff method.

Here's a look at the syntax for the command, which you can also find in the iRules wiki, here:

> after <ms>
> Delays evaluation of the current script for milliseconds. Known as an inline after. Think of this like a sleep statement or similar.
>
> after <ms> [-periodic] <script>
> Schedules <script> for evaluation after <ms> milliseconds. Optionally continuously repeats the evaluation
> every <ms> milliseconds if –periodic is specified. Returns an identifier that can be used with cancel or info.
>
> after cancel <id> …
> Cancels the scheduled evaluation of a script identified by <id>.
>
> after info [<id> …]
> Returns information about currently scheduled scripts, or about a specific <id>.

As you can see, the command is pretty straight-forward. It's about what you'd expect given the functionality, which is just as it should be. It's easy to use and extremely handy. That's a great thing to be able to say about a new command. There are plenty of ways in which you can make use of after in your iRules.

- Detecting the lack of an application response.
- Conducting cleanup on global variables or arrays.
- Timing out a “client” or “user”.
- Calculating any kind of rate.
- Aggregating or averaging based on time.
- Scheduling events or changed behavior on a daily or other recurring basis.

This command can make new code possible, and old code better, in some cases at least. Here's an updated version of an old-but-great iRule for connection limiting. Before it was written to disallow more than 1000 connections to be open from a given IP address. Now, with the after command, I can make an easy change to make it so that I'm now limiting a given IP to 120 connections per minute, rather than just 1000 total concurrent. This is not only a better limiting solution in most cases, but it helps do away with concerns of stale connections counting against new users, too. A very handy update thanks to the shiny new after command.

```
when RULE_INIT {
  array set connections { }
  after 60000 -periodic {
    array set connections { }
  }
}
when CLIENT_ACCEPTED {
  if { [info exists ::connections([IP::client_addr])] } {
    if { [incr ::connections([IP::client_addr])] > 120 } {
      reject
    }
  } else {
    set ::connections([IP::client_addr]) 1
  }
}
when CLIENT_CLOSED {
  if { [incr ::connections([IP::client_addr]) -1] <= 0 } {
    unset ::connections([IP::client_addr])
  }
}
```

Get the Flash Player to see this player.

20090414-iRulesAfter.mp3

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