Self-Contained BIG-IP Provisioning with iRules and pyControl – Part 2

Jason Rahm, 2010-14-10

As I stated last week in Part 1, iRules work on the live data between client and server, and iControl works on the management plane out of band to collect information or to modify or create configuration objects. However, what if you could combine forces, wholly contained on your BIG-IP LTM? That's the scenario I started tackling last week. In this second part, I'll address the iRule and iControl components necessary to complete a provisioning task.

- Self-Contained BIG-IP Provisioning with iRules and pyControl - Part 1

The iRule code

I'll get the solution working prior to making it extensible and flashy. First thing I'll do to enhance the test iRule from last week is to utilize the restful interface approach I used to access table information in this tech tip. I want to pass an action, a pool name, and a pool member through the URL to the system. In this case, the action will be enable or disable, and the pool and pool member should be valid entries. In the event a pool or pool member is passed that is not valid, there is simple error checking in place, which will alert the client. Note also in the code below that I incorporated code from George's basic auth tech tip.

```
when HTTP_REQUEST {
    binary scan [md5 [HTTP::password]] H* password

    if { ([HTTP::path] starts_with "/provision") } {
        if { [class lookup [HTTP::username] provisioners] equals $password } {
            scan [lrange [split [URI::path [HTTP::uri]] "/"] 2 end-1] %s%s%s action pname pmem
            if { [catch {[members $pname]} errmsg] && !($errmsg equals "")} {
                if { [members -list $pname] contains "[getfield $pmem ":" 1] [getfield $pmem ":" 2]" } {
                    set pmstat1 [LB::status pool $pname member [getfield $pmem ":" 1] [getfield $pmem ":" 2]]
                    log local0. "#prov=$action,$pname,$pmem"
                    after 250
                    set pmstat2 [LB::status pool $pname member [getfield $pmem ":" 1] [getfield $pmem ":" 2]]
                    HTTP::respond 200 content "<html><body>Original Status<br />$pmem-$pmstat1<p>New Status<br />$pmem-$pmstat2</body></html>"
                } else { HTTP::respond 200 content "$pmem not a valid pool member." }
            } else { HTTP::respond 200 content "$pname not a valid pool." }
        } else { HTTP::respond 401 WWW-Authenticate "Basic realm=\"Secured Area\"" }

    }
}
```

I added the "**after 250**" command to give the pyControl script some time to receive the syslog event and process before checking the status again. This is strictly for display purposes back to the client.

The pyControl code

I already have the syslog listener in place, and now with the iRule passing real data, this is what arrives via syslog:

> <134>Oct 13 11:04:55 tmm tmm[4848]: Rule ltm_provisioning <HTTP_REQUEST>: #prov=enable,cacti-pool,10.10.20.200:80

Not really ready for processing, is it? So I need to manipulate the data a little to get it into manageable objects for pyControl.

```
# Receive messages
while True:
    data,addr = syslog_socket.recvfrom(1024)
    rawdata = data.split(' ')[-1]
    provdata = rawdata.split('=')[-1]
    dataset = provdata.split(',')
    dataset = map(string.strip, dataset)
```

Splitting on whitespace removes all the syslog overhead, the I split again on the "=" sign to get the actual objects I need. Next, I split the data to eliminate the commons and create a list,and the finally, I strip any newline charcters (which happens to exist on the last element, the IP:port object). Now that I have the objects I need, I can go to work setting the enable/disable state on the pool members. Thankfully, the disable pool member pyControl code already exists in the iControl codeshare, I just need to do a little modification. None of the def's need to change, so I added them as is. Because the arguments are coming from syslog and not from a command line, I have no need for the sys module, so I won't import that. I do need to import string, though to strip the newline character in the map(string.strip, dataset) command above. The dataset object is a list and the elements are 0:action, 1:pool, 2:pool_member. pyControl expects the members to be a list, even if it's only one member, which is true of this scenario.

```
### Process Pool, Pool Members, & Desired State ###
POOL = dataset[1]
members = [dataset[2]]
sstate_seq = b.LocalLB.PoolMember.typefactory.create('LocalLB.PoolMember.MemberSessionStateSequence')
sstate_seq.item = session_state_factory(b, members)

session_objects = session_state_factory(b, members)

if dataset[0] == 'enable':
    enable_member(b, session_objects)
elif dataset[0] == 'disable':
    disable_member(b, session_objects)
else:
    print "\nNo such action: \"dataset[0]\" "
```

More error checking here, if the action passed was enable/disable, I call the appropriate def, otherwise, I log to console. OK. Time to test it out!

Results

In this video, I walk through an example of enabling/disabling a pool member

Provision Through iRules

The Fine Print

Obviously, there is risk with such a system. Allowing provisioning of your application delivery through the data plane can be a dangerous thing, so tread carefully. Also, customizing syslog and installing and running pyControl on box will not survive hotfixes and upgrades, so processes would need to be in place to ensure functionality going forward. Finally, I didn't cover ensuring the pyControl script is running in the background and is started on system boot, but that is a step that would be required as well. For test purposes, I just ran the script on the console.

Conclusion

There is an abundance of valuable and helpful information on how to utilize the BIG-IP system, iRules, and iControl. Without too much work on my own, I was able to gather snippets of code here and there to deliver a self-contained provisioning system. The system is short on functionality (toggling pool members), but could be extended with a little elbow grease and MUCH error checking. For the full setup, check out the LTM Provisioning via iRules wiki entry in Advanced Design and Configuration wiki.

Related Articles

- HTTP Basic Access Authentication iRule Style > DevCentral > F5 ...
- F5 DevCentral > Community > Group Details - Python iControl Library
- Getting Started with pyControl v2: Installing on Windows ...
- Getting Started with pyControl v2: Installing on Ubuntu Desktop ...
- Experimenting with pyControl on LTM VE > DevCentral > F5 ...
- Getting Started with pyControl v2: Understanding the TypeFactory ...
- Getting Started with pyControl v2: Constructor Changes ...
- LTM 9.4.2+: Custom Syslog Configuration > DevCentral > F5 ...
- LTM 9.4.2+ Custom Syslog errors when bpsh used - DevCentral - F5 ...
- Custom syslog-ng facility - DevCentral - F5 DevCentral > Community ...
- Customizing syslog-ng f_local0 filter - DevCentral - F5 DevCentral ...
- How to write iRule log statements to a custom log file, and rotate ...
- Syslog locally and remote with specific facility level ...
- DevCentral Wiki: Syslog NG Email Configuration

Technorati Tags: F5 DevCentral,pyControl,iControl,iRules,syslog-ng,provisioning

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

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113