F5 Networks Training

Getting Started with BIG-IP Programmability

Try It Yourself Lab Guide

November 2017

Getting Started with BIG-IP Programmability

Try It Yourself Lab Guide

First Printing; November, 2017

Support and Contact Information

Obtaining Technical Support

Web	tech.f5.com (Ask F5)
Phone	(206) 272-6888
Email (support issues)	support@f5.com
Email (suggestions)	feedback@f5.com

Contacting F5 Networks

Web	www.f5.com
Email	sales@f5.com & info@f5.com

F5 Networks, Inc.	**F5 Networks, Ltd.**	**F5 Networks, Inc.**	**F5 Networks, Inc.**
Corporate Office	**United Kingdom**	**Asia Pacific**	**Japan**
401 Elliott Avenue West	Chertsey Gate West	5 Temasek Boulevard	Akasaka Garden City 19F
Seattle, Washington 98119	Chertsey Surrey KT16 8AP	#08-01/02 Suntec Tower 5	4-15-1 Akasaka, Minato-ku
T (888) 88BIG-IP	United Kingdom	Singapore, 038985	Tokyo 107-0052 Japan
T (206) 272-5555	T (44) 0 1932 582-000	T (65) 6533-6103	T (81) 3 5114-3200
F (206) 272-5557	F (44) 0 1932 582-001	F (65) 6533-6106	F (81) 3 5114-3201
Training@f5.com	EMEATraining@f5.com	APACTraining@f5.com	JapanTraining@f5.com

Legal Notices

Lab 1 – Try It Yourself

Introduction: Escaping New Lines, cURL, jq and Pipes

Escaping New Lines

Almost every command shell available on Linux and macOS uses the same syntax to split long command lines into multiple shorter ones. This is sometimes called line escaping or line continuation. For example, the following command

```
echo Hello World
```

could be made shorter by inserting a backslash at the split point, such as

```
echo Hello \
World
```

or even

```
echo \
Hello \
World
```

All three examples print "Hello World". Notice there is a space in front of the backslash and there is no backslash on the final line of the split command.

Windows shells allow the same technique, but use different characters, depending on the shell. When using the COMMAND (CMD) shell, replace the backslash with a caret or circumflex character (^). When using PowerShell, replace the backslash with a backtick or grave accent character (`).

macOS and Linux (most shells)	Windows (COMMAND shell)	Windows (PowerShell)
echo \ Hello \ World	echo ^ Hello ^ World	echo `   Hello ` World

cURL

cURL is a command line tool for getting or sending data or files using URL syntax. The simplest form of the command is

```
curl www.example.com
```

which performs an HTTP GET to http://www.example.com. Our examples are typically more complex, such as

```
curl -s -k -u admin:admin1 -X POST \
  -H 'Content-Type: application/json' \
  -d '{"send": "GET /\r\n", "recv": "F5 Training Services"}' \
  https://192.168.1.31/mgmt/tm/ltm/monitor/http/wiki.mon | jq .
```

Notice this command has been written for Linux or macOS because it uses a backslash for line escaping.

Dissecting the above command we inspect every component, one at a time. Notice the “-sku” flags above are three flags concatenated for brevity. Below, we’ll examine each flag individually.

curl	Command name
-s	Silent mode, do not show cURL progress meter
-k	For HTTPS, cURL expects a Secure SSL (TLS) connection. Use this flag if the server doesn’t have the proper cert (typical for lab testing)
-u admin:admin1	Specify credentials as username:password
-X POST	Use POST method (GET is default)
-H 'Content-Type: application/json'	Specify header (required to send REST data)
-d '{"send": "GET /\r\n", "recv": "..."}'	JSON data to send (note the outer single-quotes are required)
https://192.168.1.31/mgmt/...	URL

Pipe

After the *curl* command notice the upright bar (|) character which is called a pipe and is available in Linux, macOS, Windows COMMAND and Windows PowerShell. It takes the output from the command on the left (*curl*) and sends it to the input of the command on the right (*jq*).

jq

jq or JSON Query is a JSON processor and it runs on Linux, macOS and Windows. It is designed to query specific parts of a JSON output and then format the output for easier reading. However, we will only be using formatting part of jq. The syntax “jq .” tells jq to neatly format all of the JSON output.

Exercise 1: Create a Monitor, Pool and Virtual Server

In this exercise, three rest commands are required to create a pool monitor, a pool and a virtual server. You are shown the tmsh command used to execute each step. You are also shown the abbreviated REST syntax. Finally, you are given the exact *curl* command needed to perform this task. Use this command in your lab environment.

Note the *curl* command syntax is for Linux or macOS. If you are using Windows, change the backslash to the appropriate character for COMMAND or PowerShell (as described above).

1. Create the monitor.

TMSH	create ltm monitor http wiki.mon send "GET /\r\n" recv "F5 Training Services"
REST	POST /ltm/monitor/http {"name": "wiki.mon", "send": "GET /\r\n", "recv": "F5 Training Services"}
CURL	curl -sku admin:admin1 -X POST \ -H 'Content-Type: application/json' \ -d '{"name": "wiki.mon","send": "GET /\r\n", "recv": "F5 Training Services"}' \ https://192.168.1.31/mgmt/tm/ltm/monitor/http| jq .

Using the Config Utility or tmsh, confirm the monitor was properly created.

2. Create the pool.

TMSH	create ltm pool wiki.pool members add { 192.168.2.1:80 192.168.2.2:80 } monitor wiki.mon
REST	POST ltm/pool {"name": "wiki.pool", "members": ["192.168.2.1:80", "192.168.2.2:80"], "monitor": "wiki.mon"}
CURL	curl -sku admin:admin1 -X POST \\ -H 'Content-Type: application/json' \\ -d '{"name": "wiki.pool","member": ["192.168.2.1:80","192.168.2.2:80"], "monitor": "wiki.mon"}' \\ https://192.168.1.31/mgmt/tm/ltm/pool| jq .

Using the Config Utility or tmsh, confirm the pool was properly created.

3. Create the virtual server.

TMSH	create ltm virtual wiki.vs destination 10.10.1.101:443 profiles add { clientssl } source-address-translation { type automap } pool wiki.pool
REST	POST ltm/virtual {"name": "wiki.vs", "destination": "10.10.1.101:443", "profiles": "clientssl", "sourceAddressTranslation": {"type": "automap"}, "pool": "wiki.pool"}
CURL	curl -sku admin:admin1 -X POST \\ -H 'Content-Type: application/json' \\ -d '{"name": "wiki.vs", "destination": "10.10.1.101:443", \\ "profiles": "clientssl", "sourceAddressTranslation": {"type": "automap"}, \\ "pool": "wiki.pool"}' \\ https://192.168.1.31/mgmt/tm/ltm/pool| jq .

Using the Config Utility or tmsh, confirm the pool was properly created.
Can you connect to the newly created virtual server?

Exercise 2: Perform the tasks necessary to set up a BIG-IP System

These tasks are normally performed by the BIG-IP Setup Wizard. In the previous exercise, you were shown the tmsh syntax as well as the actual *curl* command syntax. In this example, you are only given the abbreviated REST syntax. Use this information to craft your own *curl* commands.

4. Set the time zone.

REST	PUT cm/device/bigip1 {"time-zone": "America/New_York"}

The time zone is set to “America/Los_Angeles” by default. Using the Config Utility or tmsh, confirm the time zone changed.

> Note, it is recommended that you use tmsh, because the Config Utility will be in Setup Mode. However, if you still prefer to use Config Utility, skip ahead to Step 14 and then return here to confirm the Time Zone has been set correctly.

5. Set the host name.

REST	PATCH sys/global-settings {"hostname": "bigip1.f5trn.com"}

The time zone is set to "America/Los_Angeles" by default. Using the Config Utility or tmsh, confirm the host name changed.

6. Set the DNS server.

REST	PUT sys/dns {"nameServers": ["172.16.20.20"]}

The time zone is set to "America/Los_Angeles" by default. Using the Config Utility or tmsh, confirm the DNS server changed.

7. Set the NTP server.

REST	PUT sys/ntp {"servers": ["172.16.20.20"]}

8. Create external VLAN, attach interface, and create self and floating IP Addresses.

REST	POST net/vlan {"name": "external", "tag": 4093}
REST	PUT net/vlan/external {"interfaces": [{"name": "1.1"}], "tag": 4093}
REST	POST net/self {"name": "10.10.1.31", "address": "10.10.1.31/16", "vlan": "external", "allowService": []}
REST	POST net/self {"name": "10.10.1.33", "address": "10.10.1.33/16", "vlan": "external", "allowService": [], "traffic-group": "traffic-group-1"}

Using the Config Utility or tmsh, confirm the external self and floating IP addresses exist.

9. Create internal VLAN, attach interface and create self and floating IP Addresses.

REST	POST net/vlan {"name": "external", "tag": 4093}
REST	PUT net/vlan/external {"interfaces": [{"name": "1.1"}], "tag": 4093}
REST	POST net/self {"name": "10.10.1.31", "address": "10.10.1.31/16", "vlan": "external", "allowService": []}
REST	POST net/self {"name": "10.10.1.33", "address": "10.10.1.33/16", "vlan": "external", "allowService": [], "traffic-group": "traffic-group-1"}

Using the Config Utility or tmsh, confirm the internal self and floating IP addresses exist.

10. Create IP addresses for BIG-IP system clustering.

REST	PATCH sys/state-mirroring {"addr": "172.16.1.31"}
REST	PUT cm/device/bigip1 {"unicastAddress": [{"effectiveIp": "172.16.1.31", "effectivePort": 1026, "ip": "172.16.1.31"}, {"effectiveIp": "management-ip", "effectivePort": 1026, "ip": "management-ip"}], "configsyncIp": "172.16.1.31", "mirrorIp": "172.16.1.31", "multicastIp": "any"}

Using the Config Utility or tmsh, confirm the State Mirroring address, Unicast address, Config Sync IP address and the Mirror IP address are correct.

11. Change the root password.

REST	POST /mgmt/shared/authn/root {"oldPassword": "default", "newPassword": "root1"}

Using ssh, log into BIG-IP and confirm the root password is correct.

12. Change the admin password, and add a description.

REST	PUT auth/user/admin {"password": "admin1"}
REST	PATCH auth/user/admin {"description": "Admin User"}

Using a web browser, log into the BIG-IP Config Utility and confirm the admin password is correct.

13. Save the system configuration (save sys config).

REST	POST sys/config {"command": "save"}

At this point, the configuration is complete and it is an appropriate time to save the most recent configuration. Using ssh, log into BIG-IP and inspect the /config/bigip_base.conf configuration file and confirm the changes you made in steps 4 through 12.

14. Change the resource provisioning.

REST	PATCH sys/provision/apm {"level": "nominal"}

In this example, you will provision BIG-IP APM, but this is the method used to provision or deprovision any other module. Other levels of provisioning are available, just as there are in the Config Utility and tmsh. BIG-IP LTM is provisioned by default. To deprovision an already provisioned module, set the “level” to “none”.

15. Check if BIG-IP System needs to be rebooted, and reboot if needed.

REST	POST util/bash {"command": "run", "utilCmdArgs": "-c \"cat /var/prompt/ps1\""}
REST	POST sys {"command": "reboot"}

Sometimes after the provisioning step, BIG-IP needs to be rebooted. This typically occurs when an already provisioned module is deprovisioned. When you are logged on to BIG-IP using a web browser, you receive a warning in the top-left corner of the Config Utility alerting you to reboot. When you are logged into BIG-IP using ssh and tmsh, the command prompt changes to alert you that a reboot is required.

If you are performing this action programmatically, the automation tool needs to know if a reboot is required. The first command, above, obtains that information. If you look closely at the syntax, you will discover that this command is using iControl REST to run a bash shell that in turn runs a command to list out the contents of a file. The contents of that file are returned as JSON to the calling program which can then determine if a BIG-IP reboot is required. If it is, the second command will perform this task.

16. Take the Config Utility out of setup mode.

REST	PATCH sys/global-settings {"guiSetup": "disabled"}

When BIG-IP is delivered new, or when it is reset to its default settings, the Config Utility is in Setup Mode. When you complete the Setup Wizard, it takes the Config Utility out of Setup Mode. The above command performs that task.

Using the Config Utility, confirm the BIG-IP system is no longer in the Setup Wizard and that you can see all the features and modules expected with your current level of licensing and provisioning.