F5 DNS Automation Demo 12.1.x Documentation

Release

Eric Chen, Vladimir Bojkovic

May 31, 2017

Lab 1: Connecting to UDF 2.0

1 Connecting to UDF 2.0 **3**

2 Launching Deployment **5**

3 About the Demo Environment **7**

4 Servers in the Demo **9**

5 Connecting with Windows RDP **11**

6 Licensing/Resetting BIG-IP **13**

7 Appendix **15**
- 7.1 HiDPI Displays . . . 15
- 7.2 Change keyboard input . . . 16

8 Creating BIG-IP DNS Sync Group **17**

9 Configuring BIG-IP DNS **19**
- 9.1 Enabling DNS Sync . . . 19
- 9.2 Add Data Center . . . 19
- 9.3 Add Server . . . 19
- 9.4 Syncing BIG-IP DNS . . . 20

10 Optional Exercises **23**
- 10.1 DNS Cache . . . 23
- 10.2 DNS Profiles . . . 23
- 10.3 DNS Listeners . . . 24
- 10.4 LTM Configuration . . . 26
- 10.5 DNS Topology . . . 29
- 10.6 Verifying configuration . . . 33
- 10.7 Testing Internal Connections . . . 33
- 10.8 Testing External Connections . . . 34

11 F5 Python SDK **35**

12 Restoring the BIG-IP Configuration **37**

13 Run Demo **39**

14 Exploring the Demo **41**
14.1 Application Services Integration iApp . . . 41
14.2 Testing Connections . . . 42
14.3 Testing External Connections . . . 43
14.4 Testing Internal Connections . . . 43
14.5 Disabling a pool member . . . 44

15 Optional Exercises **45**
15.1 Changing the requirements . . . 45
15.2 Automating the change . . . 45

16 Lab Appendix **47**
16.1 The Script . . . 47

17 Welcome to the Lab 4 Lab Guide **51**

18 Step 0 - Restoring the BIG-IP Configuration **53**

19 Step 1 - login into Jenkins **55**

20 Step 2 - deploying F5 config via automation scripts using Jenkins **57**

21 Jenkins Pipeline **61**

22 Restoring the BIG-IP Configuration **63**

23 Creating a pipeline **65**

24 Launching the pipeline **69**

25 Failing Tests **73**

26 Indices and tables **79**

This is a Demo of using the F5 Python SDK to automate the process of deploying BIG-IP DNS.

CHAPTER 1

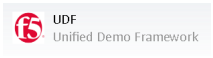
Connecting to UDF 2.0

This lab goes through the process of launching the demo environment, connecting via Windows RDP, and licensing/resetting the BIG-IP devices.

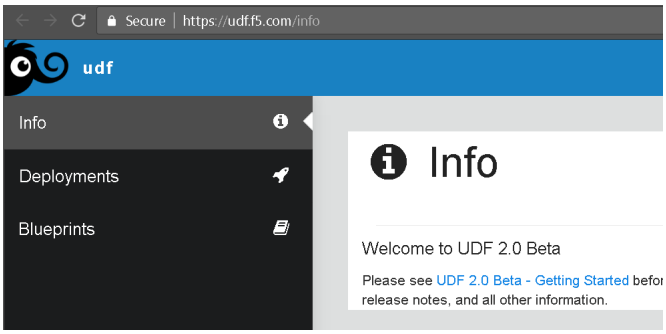
CHAPTER 2

Launching Deployment

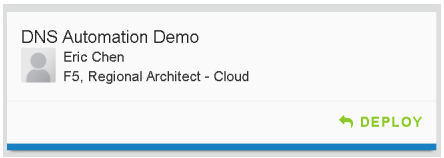
From https://federate.f5.com find the “UDF” link and click on it.



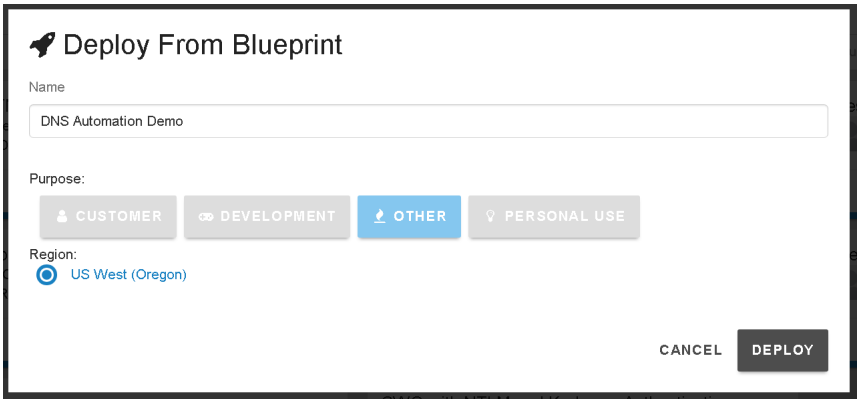
You should see the main UDF page. Click on the “Blueprints” link.



Find the “DNS Automation Demo” Blueprint and click on “Deploy”

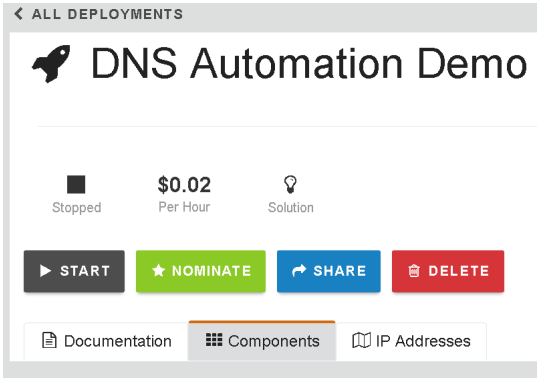


You will see “Deploy From Blueprint” and click “Deploy” again.

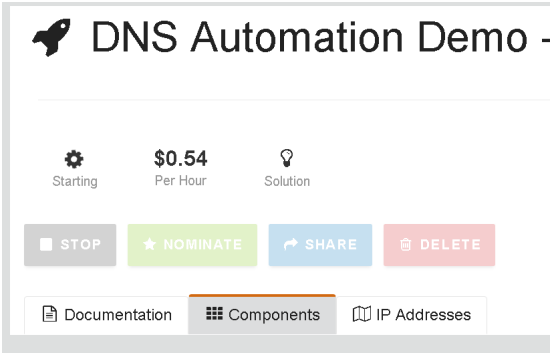


You should now see the "DNS Automation Demo" screen that has the "Start" button listed.

Click on "Start".



The Deployment is now starting.

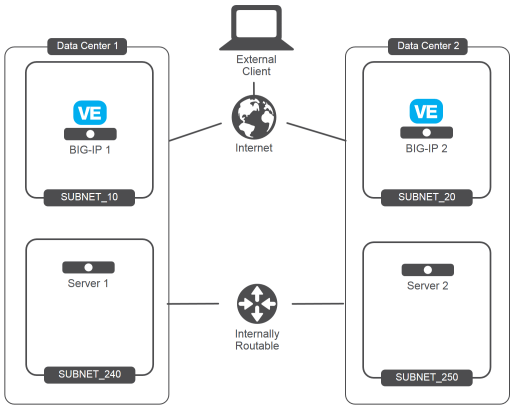


CHAPTER 3

About the Demo Environment

This demo is designed to provide a solution with the following attributes.

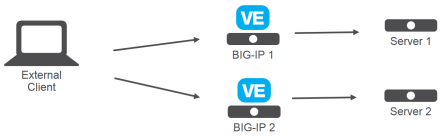
- Two BIG-IP devices in separate Data Centers (Regions, Availability Zone, etc...)
- Two backend servers in separate DC
- The two DC are routable to each other via L3
- Provide recursive DNS for internal clients



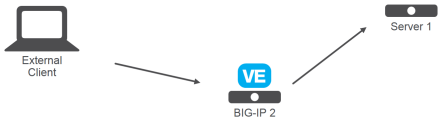
The desired behavior for requests

- External clients round-robin between backend servers
- Persist External client requests to original DC server if requests move between DC
- Internal client requests will have affinity to local DC server

DNS Automation Demo: External Clients

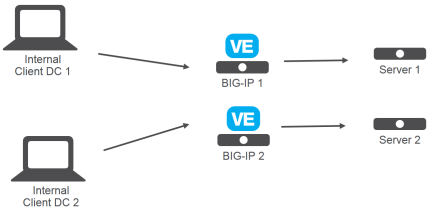


External Requests round-robin BIG-IP 1 or 2



Persist Connection to original server regardless of DC

DNS Automation Demo: Internal Clients



Internal Client Affinity to local DC

CHAPTER 4

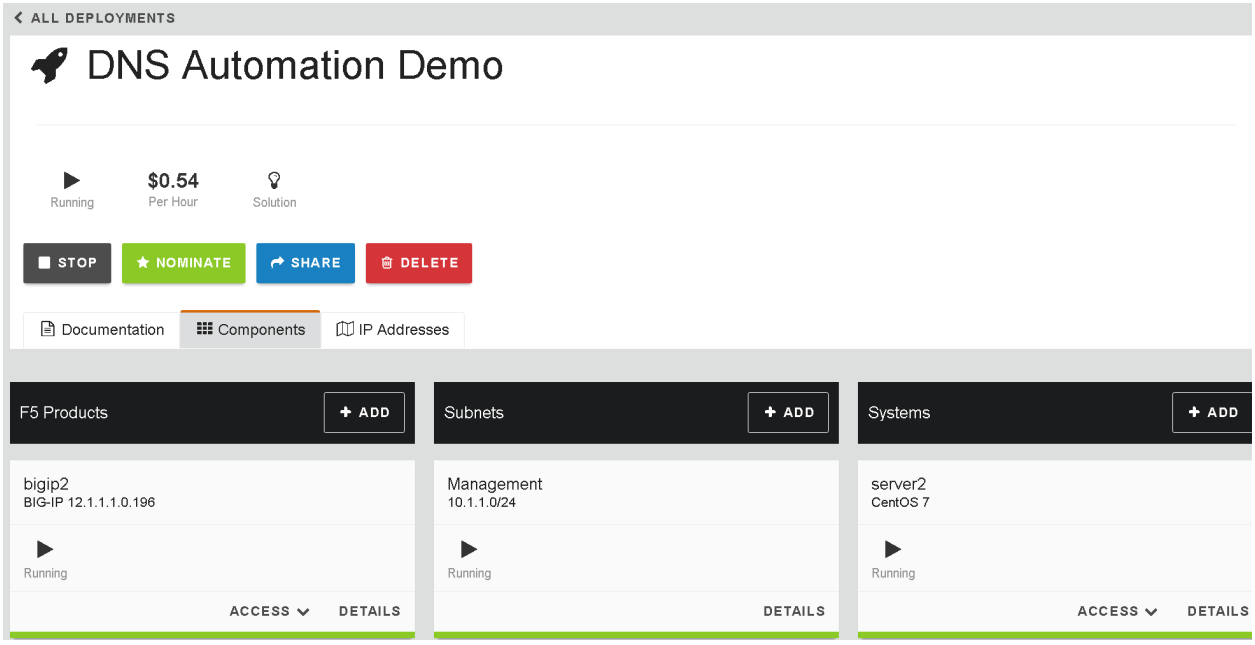
Servers in the Demo

Hostname	MGMT IP	Network IP	Login / Password
win2008-rdp	10.1.1.10	10.1.20.5	administrator / [see details page]
bigip1	10.1.1.7	10.1.10.240	admin / admin
bigip2	10.1.1.8	10.1.10.240	admin / admin
server1	10.1.1.4	10.1.240.10	centos or user / [ssh key]
server2	10.1.1.5	10.1.250.10	centos or user / [ssh key]
automation	10.1.1.6	10.1.20.8	centos or user / [ssh key]
jenkins	10.1.1.11		ubuntu / [ssh key]

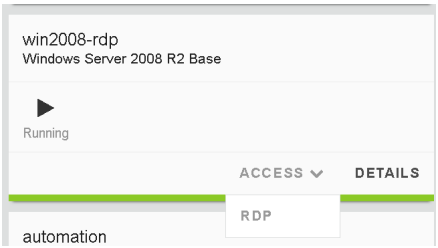
CHAPTER 5

Connecting with Windows RDP

Once the Deployment is finished starting click on the “Components” tab to get a view like the following.



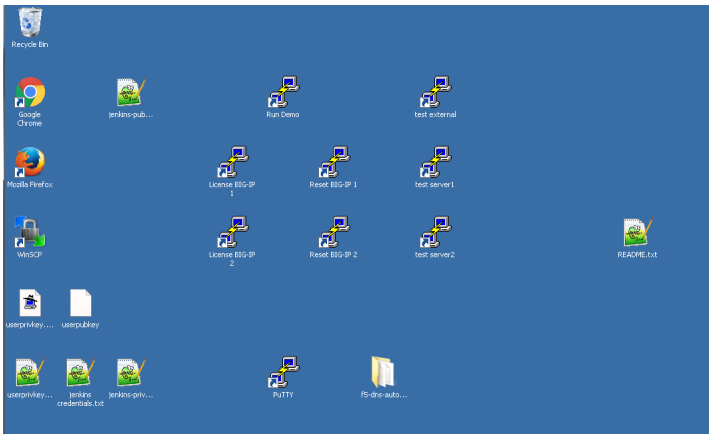
Find the win2008-rdp component and click on the “Access” button to display the “RDP” link.



Download/launch the RDP link.

> **Warning:** For MAC user, it is recommended to use Microsoft Remote Desktop. You may not be able to access your jumpbox otherwise. It is available in the App store (FREE).

The username is "Administrator" and the password can be found under the "Details" page of the win2008-rdp component. Note: copy/paste of the password into the RDP session does not work. You can copy the password, open the RDP session settings in your RPD clinet and paste the password there. Save it and open the RDP session. You should see a desktop that looks like the following.



> **Warning:** Help for small font/text. The lab is optimized for a 1080p display. For HiDPI displays you may want to re-size your screen first. See *HiDPI Displays* for some suggested tips.

For help on changing the Keyboard Language please see *Change keyboard input*

CHAPTER 6

Licensing/Resetting BIG-IP

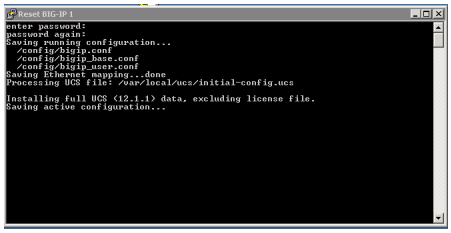
When a UDF Deployment is started you will need to re-license the device. There are links on the Desktop to expedite this process.

First find the “License” links.

Double-click on both of these and you should see a window appear briefly like the following.

Next find the “Resetting” links.

Double-click on both of these and you should see a window appear briefly like the following.



The previous steps go through a scripted process of resetting the license and restoring the system to a known state via a UCS restore.

You are now ready to start the next Lab: *Creating BIG-IP DNS Sync Group*

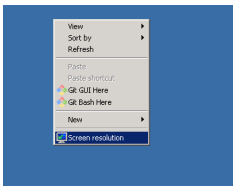
CHAPTER 7

Appendix

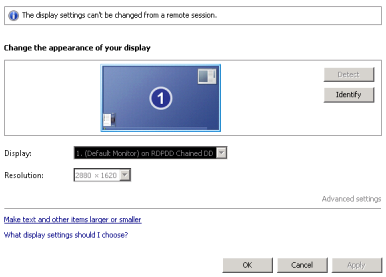
HiDPI Displays

The default resolution is intended for low DPI monitors (i.e. 1080p).

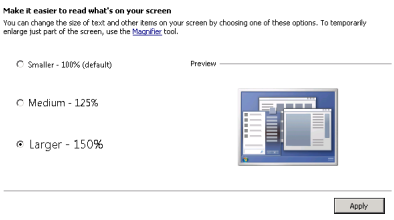
If you need to change the DPI (make the font/text large) right click on the desktop.



Click on the “Make text and other items larger...”



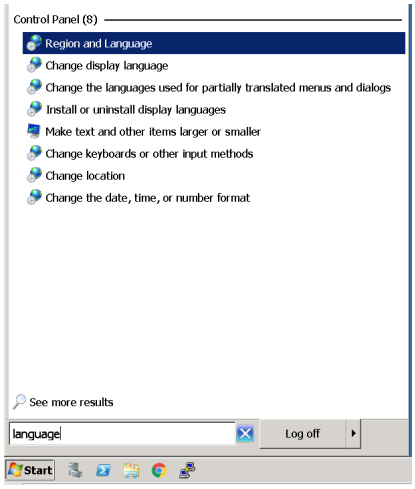
Select “150%”. Logout/login.



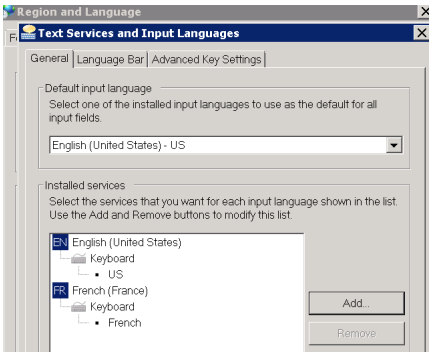
Change keyboard input

The default keyboard mapping is set to english. If you need to change it, here is the method

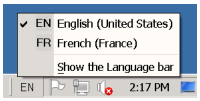
- Click on the start menu button and type ‘Language’ in the search field.
- Click on ‘change keyboards’ option in the search list



- Click on ‘Add a language’



- Add the language you want to have for your keyboard mapping.



- Select the language (bottom right of screen).

Lab 1 will cover the process of connecting to UDF 2.0.

CHAPTER 8

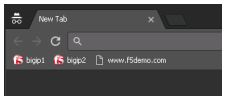
Creating BIG-IP DNS Sync Group

The following is adapted from the official F5 documentation:

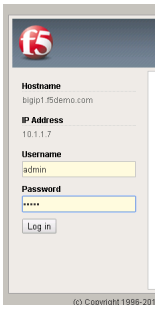
https://support.f5.com/kb/en-us/products/big-ip-dns/manuals/product/bigip-dns-implementations-12-1-0/3.html#conceptid

First start by launching Google Chrome.

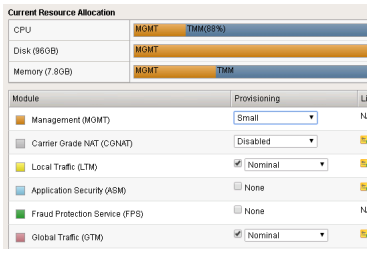
In Chrome you should see links to BIG-IP 1 and 2.



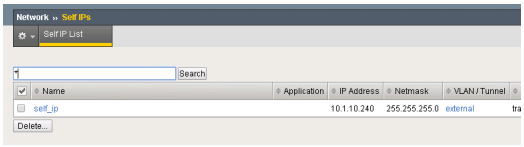
Login to BIG-IP 1 *username: admin, password: admin*



Take a look at the current config. You should see BIG-IP DNS is provisioned.



There is an existing self IP.



CHAPTER 9

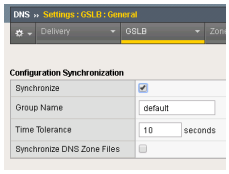
Configuring BIG-IP DNS

The following will go through the basic steps of setting up BIG-IP DNS.

Enabling DNS Sync

First go to DNS -> Settings -> GSLB -> General

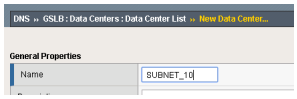
Find the “Synchronize” checkbox and click on it.



Add Data Center

Next go back to DNS -> GSLB -> Data Centers

Create a Data Center (DC) named SUBNET_10 and SUBNET_30.

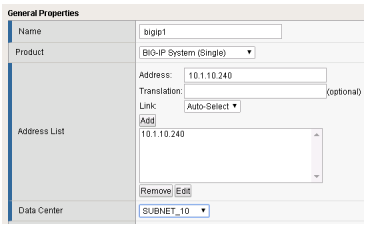


Add Server

Next go to DNS->GSLB->Servers

Create a server bigip1 associated with SUBNET_10 and bigip2 associated with SUBNET_30.

Name	Address	Data Center
bigip1	10.1.10.240	SUBNET_10
bigip2	10.1.30.240	SUBNET_30

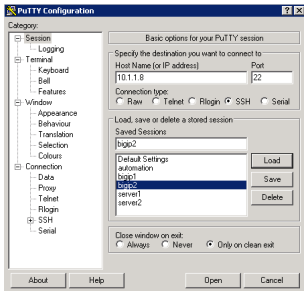


Syncing BIG-IP DNS

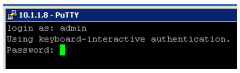
At this point BIG-IP 1 has the desired BIG-IP DNS configuration, but it needs to be synced with BIG-IP 2.

From the Desktop launch "Putty".

Find the BIG-IP 2 login.

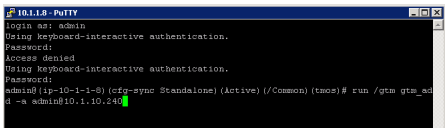


Login. *username: admin, password: admin*



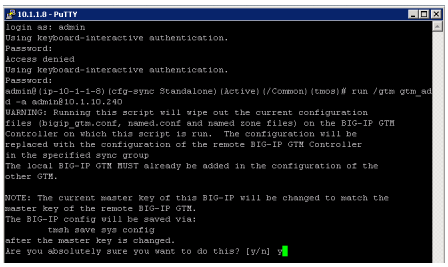
Run the command:

```
run /gtm gtm_add -a admin@10.1.10.240
```

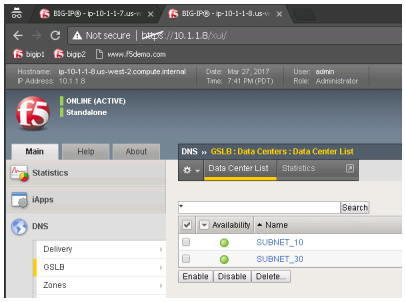


Warning: In the lab the self-ip is configured to allow ALL services. This is not a safe production configuration.

When prompted confirm/enter password.



In Chrome login to BIG-IP 2 and you should see that it is now synced.



You should have a pair of BIG-IP devices that are in a DNS Sync Group. The next lab will go through the process of scripting these actions.

You are now ready to start the next Lab: *F5 Python SDK*

CHAPTER 10

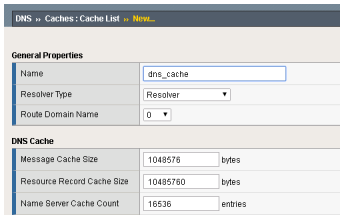
Optional Exercises

The following next lab, *F5 Python SDK*, will perform all of the following sections via an automation script. You can choose to perform all of these tasks to understand what the automation will be performing, but at the end of this lab you will delete the work that you have done.

DNS Cache

In the Demo environment we will use BIG-IP DNS as a DNS resolver. Create a DNS cache named “dns_cache”. This needs to be done separately on each BIG-IP device.

Under DNS -> Caches -> Cache List: Create a DNS cache profile “dns_cache” and accept default values.

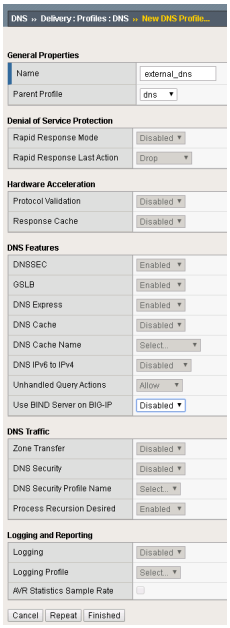


DNS Profiles

Two DNS profiles are required. One for providing a resolving DNS server and one for external DNS requests (bad idea to have an open resolver on the internet). Now create them on both BIG-IP’s.

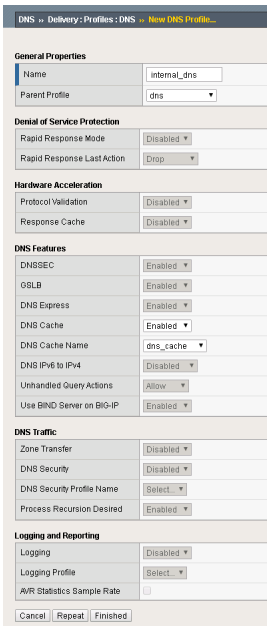
External DNS Profile

Under DNS -> Delivery -> Profiles -> DNS: Create a profile named “external_dns” that only provides GSLB and disables fallback to BIND.



Internal DNS Profile

Under DNS -> Delivery -> Profiles -> DNS: Create a profile named “internal_dns” that enables a DNS cache for resolving names.



DNS Listeners

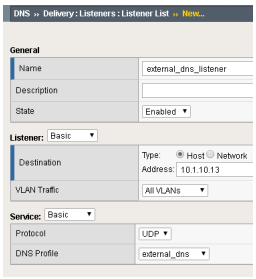
For external DNS we have two listeners for each BIG-IP. One external and one internal. First create on both BIG-IP’s the external Listeners. Apply the external_dns profile to each. Use these IP addresses:

Device	Address	Port
bigip1	10.1.10.13	53
bigip2	10.1.30.13	53

External DNS Listener

DNS -> Delivery -> Listeners

Here is the external DNS listener



Internal DNS Listener

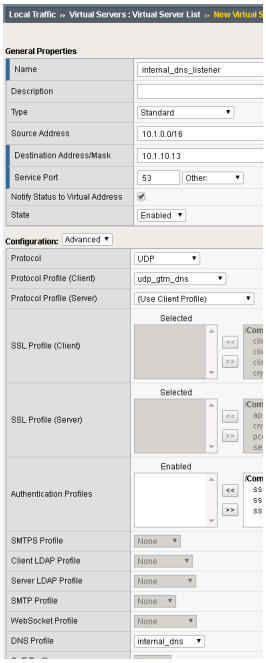
Next create on each BIG-IP internal listeners via the LTM menu. The listener is a virtual server. Specify following source address range on each internal listener: 10.1.0.0/16, UDP profile is set to udp_gtm_dns, and apply the "internal_dns" DNS profile. Keep all other settings as default.

Use these IP addresses:

Device	Address	Port
bigip1	10.1.10.13	53
bigip2	10.1.30.13	53

Create listeners for external and internal DNS. Name the external listener "external_dns_listener" and the internal listener "internal_dns_listener".

Here is an example of the internal Listener:



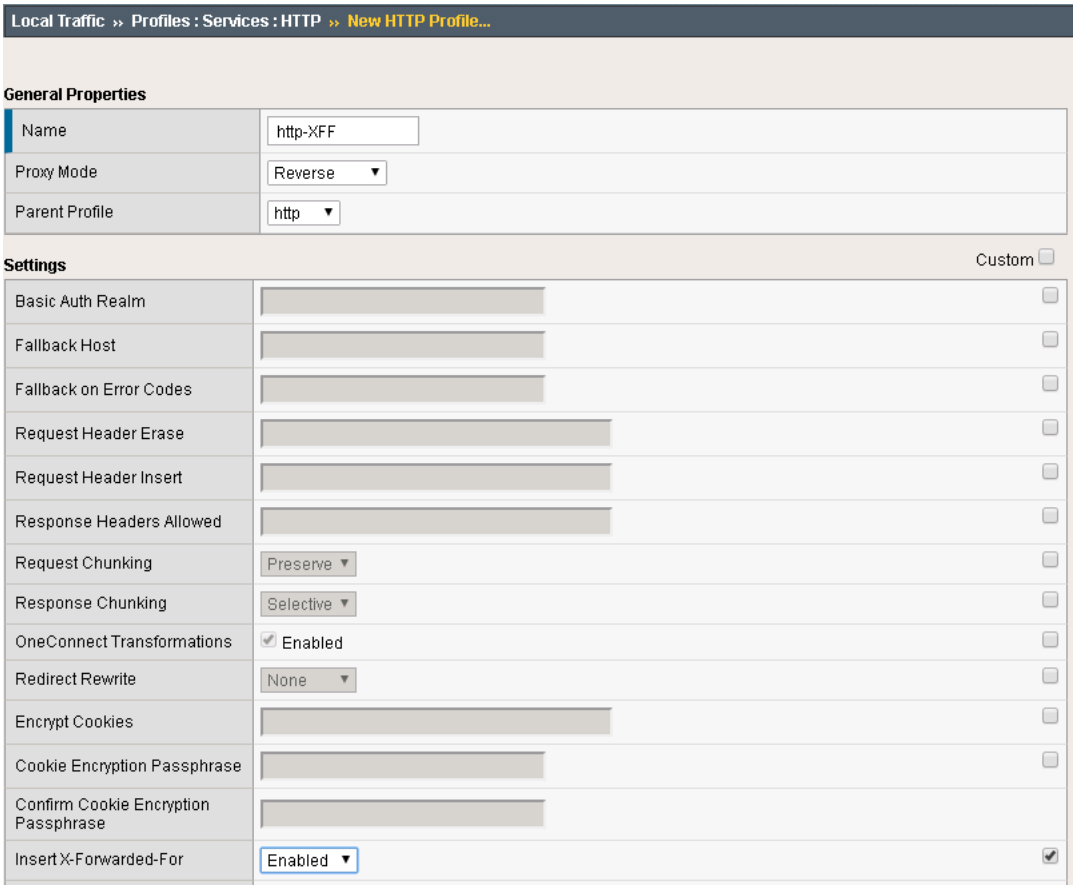
Note that the source address is set to 10.1.0.0/16, , and the DNS profile is set to “internal_dns”.

> For the demo all requests go through the internal listener, but in another environment you could split this out.

LTM Configuration

Now we have to configure the LTM section of both BIG-IP’s. Since both BIG-IP’s are standalone the configuration steps has to be applied to both BIG-IP’s.

First create an http profile named “http-XFF” that inserts X-Forwarded-For headers Local Traffic -> Profiles -> Services -> HTTP



In the Demo LTM is configured to use cookie persistence, insert X-Forwarded-For headers, and use Priority Groups for delivering traffic.

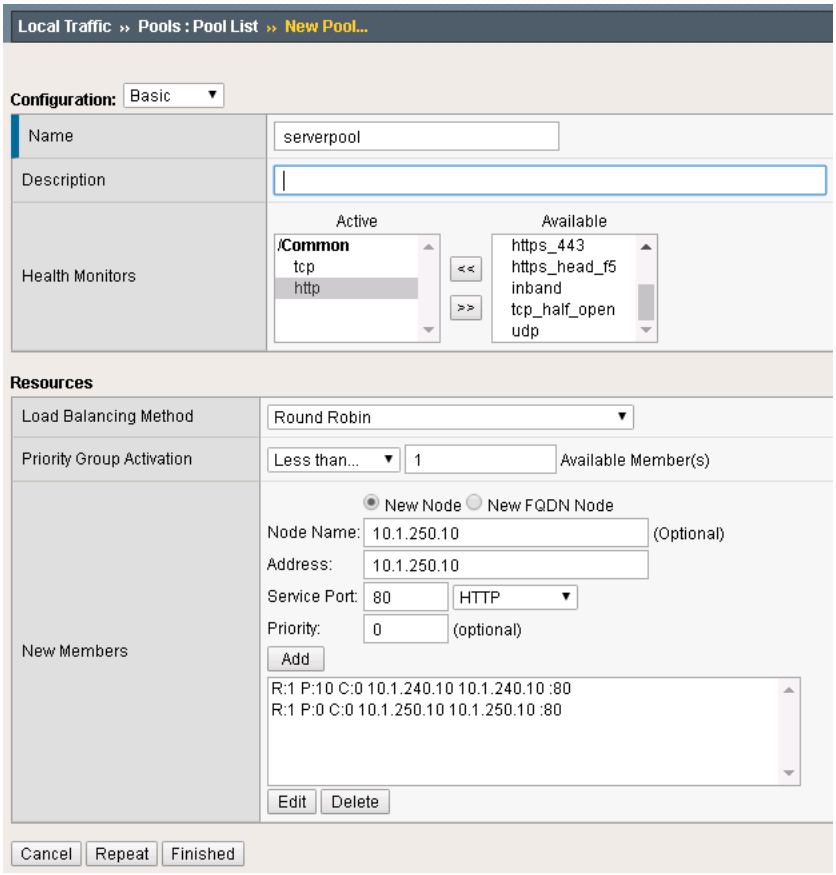
Pools

Create a pool "serverpool" on each BIG-IP. Local Traffic -> Pools

Assign HTTP and TCP monitors Enable Priority Group Activation with "1 Available Member"

Device	Pool Member #1	Port	Priority Group	Pool Member #2	Port	Priority Group
bigip1	10.1.240.10	80	10	10.1.250.10	80	0
bigip2	10.1.250.10	80	10	10.1.240.10	80	0

Pool config example:



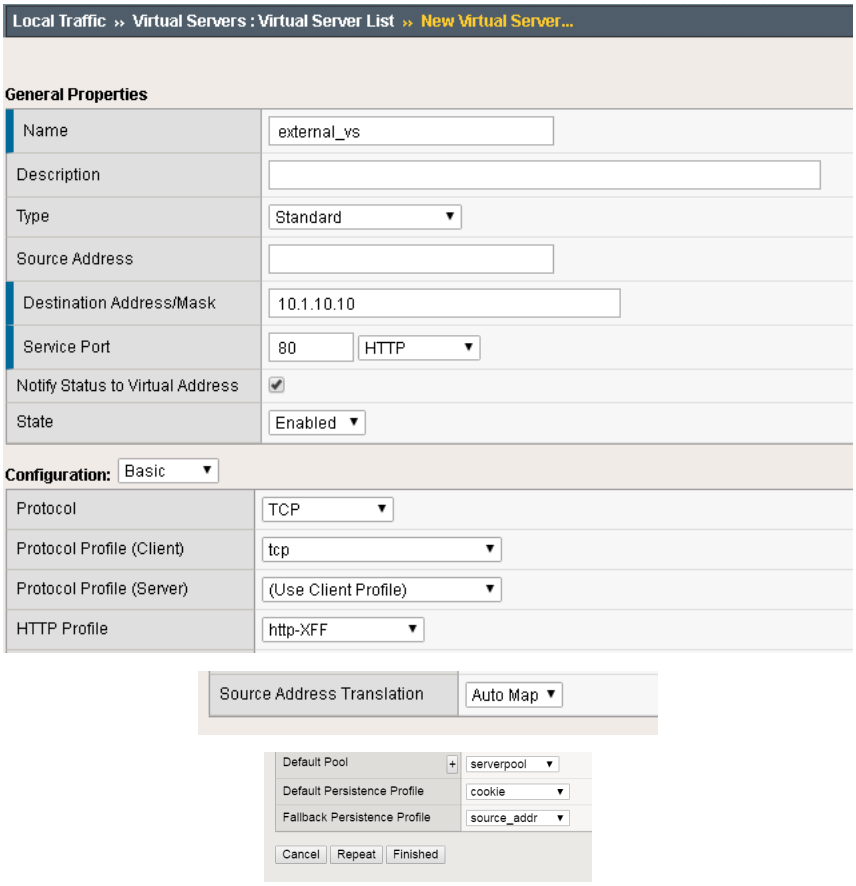
Virtual Servers

In the next step create two standard TCP virtual servers per BIG-IP. One external and one internal. Apply the http-XFF profile, SNAT Automap and the pool "serverpool"

Use following IP addresses

Device	Name	Address	Port
bigip1	external_vs	10.1.10.10	80
bigip1	internal_vs	10.1.10.100	80
bigip2	external_vs	10.1.30.10	80
bigip2	internal_vs	10.1.30.100	80

Here a configuration example:



DNS Topology

After BIG-IP DNS and BIG-IP LTM are configured and ready to run, it is time to create the logical geographical load balancing. BIG-IP DNS will receive DNS requests and respond based on the location of the requesting IP.

The demo will show two sections:

1. Split DNS: BIG-IP DNS will respond to internal clients with private IP addresses differently than for external clients with IP addresses that are not internal.
2. **regional Loadbalancing - based on the region of the client IP the response will be different.** Note: we will simulate the request with a private IP address range, because this lab is not exposed to the Internet.

The steps to create geolocation based load balancing on BIG-IP DNS are:

1. add the virtual servers
2. create the pools
3. create wide-IPs
4. create regions
5. create records

All configurations have to be applied one BIG-IP DNS only. The config changes will be synced to the other BIG-IP DNS via a sync group that was created before.

Step 1: Virtual Servers

BIG-IP DNS has to be aware of the services that are provided by BIG-IP LTM. BIG-IP DNS sees BIG-IP LTM as a Server that is bound to a datacenter. For BIG-IP DNS the virtual servers on the BIG-IP LTM are virtual servers on the server in the datacenter.

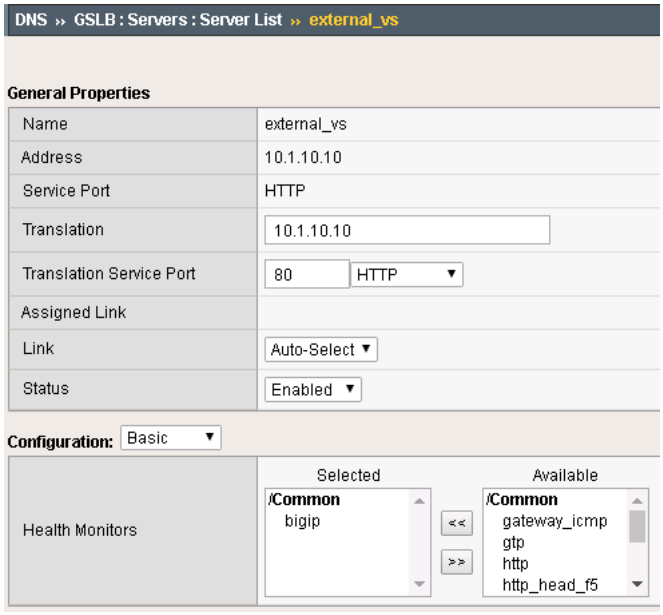
Create the virtual servers on BIG-IP DNS under:

DNS -> GSLB -> Servers

Use following IP addresses for the virtual servers:

Device	Name	IP:PORT	Health Monitor
bigip1	external_vs	10.1.10.10:80	bigip
bigip1	internal_vs	10.1.10.100:80	bigip
bigip2	external_vs	10.1.30.10:80	bigip
bigip2	internal_vs	10.1.30.100:80	bigip

Here an example:



Step 2: Pools

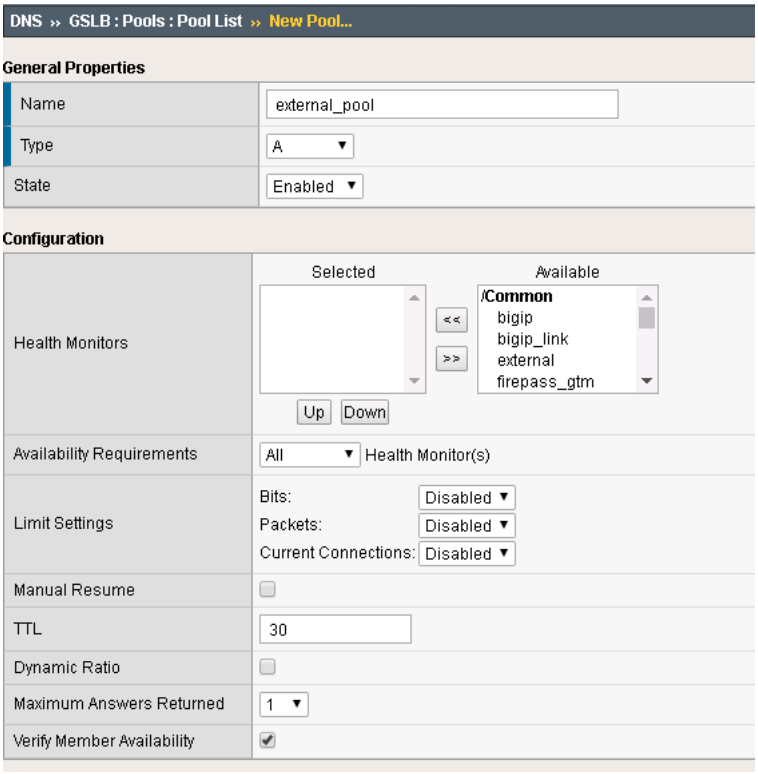
The next step is to configure the pool. under: DNS -> GSLB -> Pools

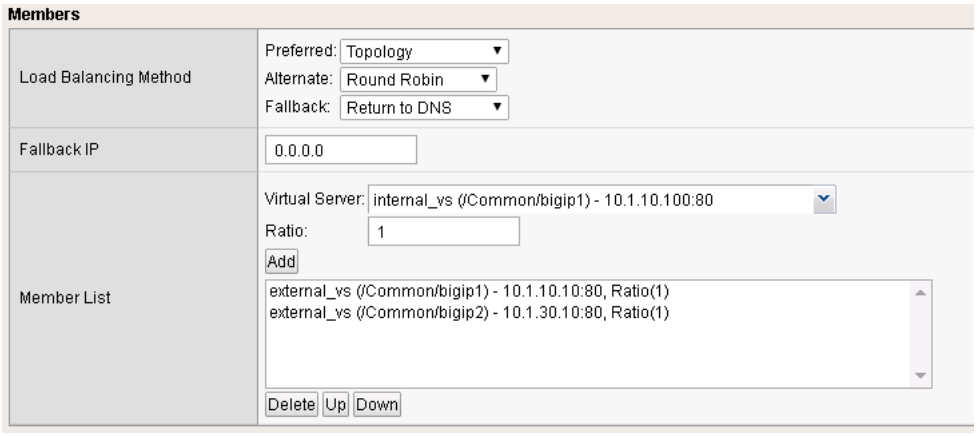
Configure two pools (internal_pool and external_pool) with following load balancing methods: * Preferred: Topology * Alternate: Round-Robin * Fallback: none

use following settings:

Name	Type	Pool Members
external_pool	A	bigip1:external_vs, bigip2:external_vs
internal_pool	A	bigip1:internal_vs, bigip2:internal_vs

Here an example:





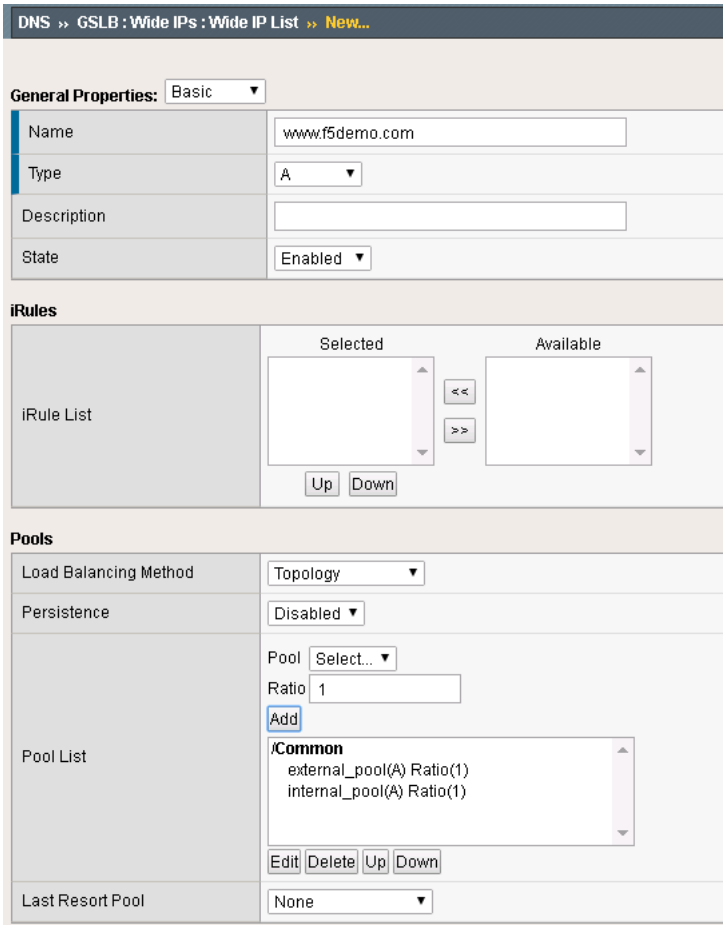
Step 3: Wide IPs

The next step is to configure the Wide IP. under: DNS -> GSLB -> Wide IPs

use following settings and Topology as load balancing method

Name	Type	Pools	Last Resort pool
www.f5demo.com	A	external_pool, internal_pool	external_pool

Here an example:



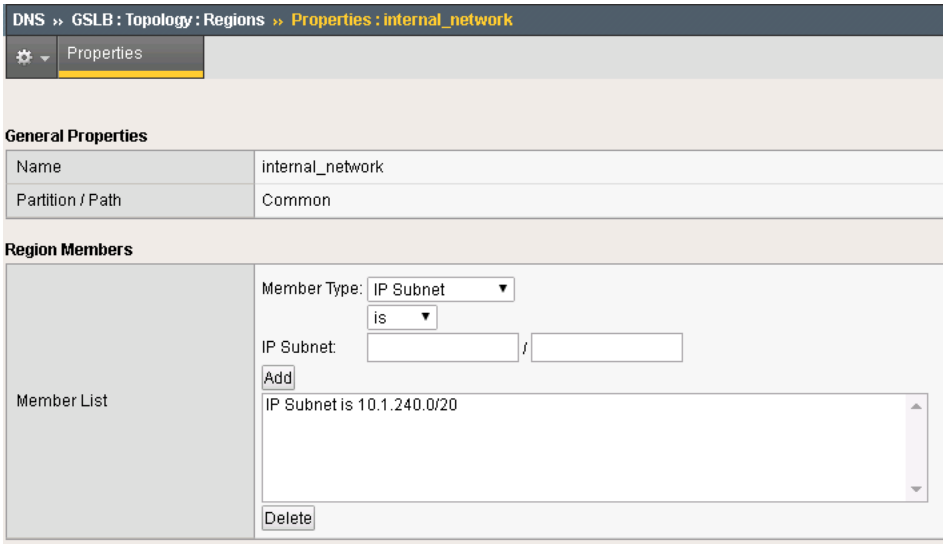
Step 4: Topology Regions

The next step is to define regions that will be used by topology records. under DNS -> GSLB -> Topology -> Regions

Use following IP addresses:

Name	Subnets
internal_network	10.1.240.0/20
region_1	10.1.10.0/24,10.1.240.0/24
region_2	10.1.30.0/24,10.1.250.0/24

Here an example:



Step 5: Topology Records

The last step is to define the topology records, that BIG-IP DNS will use for load balancing decisions under DNS -> GSLB -> Topology -> Records

use following settings

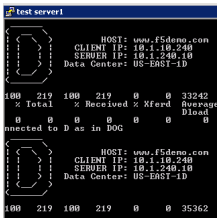
Source	is/is not	Destination
region /Common/internal_network	is not	pool /Common/external_pool
region /Common/internal_network	is	pool /Common/internal_pool
region /Common/region_1	is	region /Common/region_1
region /Common/region_2	is	region /Common/region_2

Verifying configuration

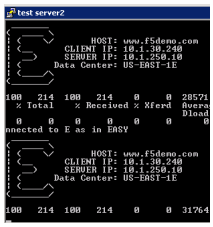
Testing Internal Connections

Now it is time to test if your configuration works.

Now run the “Test Server1” link.



and run the “Test server2” link.



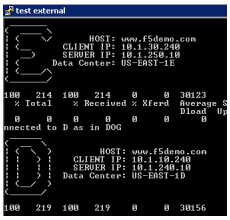
The "test-server[1-2]" links are simulating requests from internal clients. Note that BIG-IP DNS is configured to prefer requests to the same Data Center.

Question Can you explain how this is being done?

Testing External Connections

Find the "Test External" link.

Double-click on it and you should see:



The "Test External" link is simulating requests from an external client. BIG-IP DNS is configured to use round-robin load balancing between the two backend servers.

From Google Chrome find the link for "www.f5demo.com". The Windows Desktop client is configured to act like an external client.

Question Using Google Chrome the requests will always go back to the same server, why?

Deploying without Automation. This will cover the basic steps of:

1. Adding BIG-IP servers to BIG-IP DNS
2. Creating BIG-IP DNS Cluster

This provides some context of what the automation will be performing.

Additional optional exercises.

1. Creating BIG-IP LTM Virtual Server and Pools
2. Creating BIG-IP DNS Virtual Server and Pools
3. Creating BIG-IP DNS Wide-IP

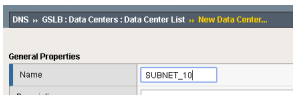
CHAPTER 11

F5 Python SDK

The F5 Python SDK provides an interface to the iControl REST interface.

This provides the ability to translate actions that you would have normally done via the GUI to actions that can be performed from Python.

Via GUI



Via Code

```
def add_datacenter(self,datacenter):
      "add datacenter in BIG-IP DNS"
      self.mgmt.tm.gtm.datacenters.datacenter.create(name=datacenter,partition=self.
↪partition)
```

This Lab will combine the process of creating a BIG-IP DNS deployment in Lab 2 and automate the process using the F5 Python SDK. The Python code in this lab is run as a script to deploy BIG-IP configurations. Think of this script like a remote tmsh Command Line Interface (CLI) to the BIG-IP. Here's an example of adding a server to BIG-IP DNS using the script.

CLI example

```
python bigip_dns_helper.py --host=10.1.1.7 \
--action add_datacenter --datacenter SUBNET_10
```

In this example we have created a Data Center (DC) named "SUBNET_10" using the F5 Python SDK example from before.

A full description of sample inputs (taken from this lab) can be found at the end of this lab in the Lab Appendix. *The Script*

For Python nerds; the pseudo code to invoke this from the CLI:

Python Pseudo Code

```
parser = OptionParser()
parser.add_option('--host')
parser.add_option('--datacenter')
parser.add_option('--action')

(options,args) = parser.parse_args()

dns_helper = DnsHelper(options.host)

if options.action == 'add_datacenter':
  dns_helper.add_datacenter(options.datacenter)
```

CHAPTER 12

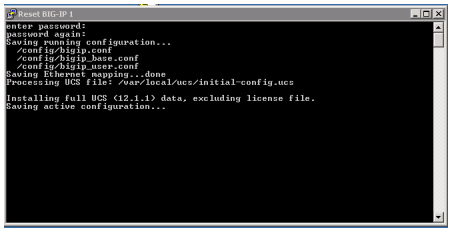
Restoring the BIG-IP Configuration

This step is to cleanup the BIG-IP config that was created in Lab 2. RDP into the Windows jump host.

Reset both BIG-IP to be the same state as after Lab 1.

Find the “Resetting” links on the Desktop.

Double-click on both of these and you should see a window appear briefly like the following.



Verify that you no longer see the changes that were previously deployed.

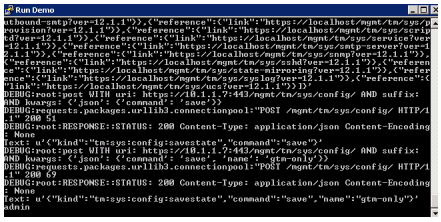
CHAPTER 13

Run Demo

This Lab is not intended to teach you how to write Python code, but instead demo how it can be leveraged to help automate a solution.

On the Desktop you will find the "Run Demo" link. Double-click the link.

The script is currently configured to output all the REST calls making for a verbose output.



An excerpt of the script that performs Lab 2 (create a DNS Sync Group):

```
python bigip_dns_helper.py --host=10.1.1.7 \
--action enable_sync
python bigip_dns_helper.py --host=10.1.1.7 \
--action add_datacenter --datacenter SUBNET_10
python bigip_dns_helper.py --host=10.1.1.7 \
--action add_datacenter --datacenter SUBNET_30

python bigip_dns_helper.py --host=10.1.1.7 \
--action add_server  --datacenter SUBNET_10 --server_name bigip1 --server_ip=10.1.10.
↪240
python bigip_dns_helper.py --host=10.1.1.7 \
--action add_server  --datacenter SUBNET_30 --server_name bigip2 --server_ip=10.1.30.
↪240
```

```
python bigip_dns_helper.py --host=10.1.1.7 \
--action save_config
sleep 3
python bigip_dns_helper.py --host=10.1.1.8 \
--action gtm_add --peer_host=10.1.1.7 --peer_selfip 10.1.10.240
```

There is the same number of steps involved, but one-click!

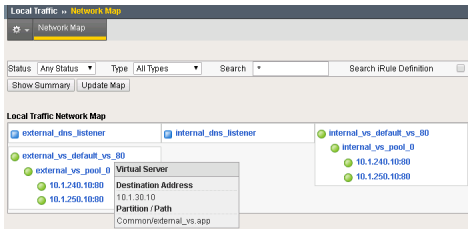
The full script can be found on GitHub.

CHAPTER 14

Exploring the Demo

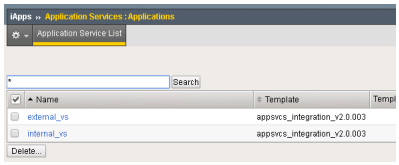
Take a look at what has been deployed. (Hint: Look at the Optional Exercise from Lab 2).

From the Network Map you can see that we have a set of external/internal DNS listeners and external/internal LTM Virtual Servers.



Application Services Integration iApp

The demo script utilizes the Application Services Integration iApp to deploy the LTM L4-L7 services.



iApp Scripts

The iApp is deployed using modified scripts from: https://github.com/F5Networks/f5-application-services-integration-iApp/tree/master/scripts

```
# import Application Services Integration iApp onto BIG-IP
python iapps/import_template_bigip.py  --impl iapps/iapp.tcl --apl iapps/iapp.apl 10.
↪1.1.7 appsvcs_integration_v2.0.003
python iapps/import_template_bigip.py  --impl iapps/iapp.tcl --apl iapps/iapp.apl 10.
↪1.1.8 appsvcs_integration_v2.0.003
```

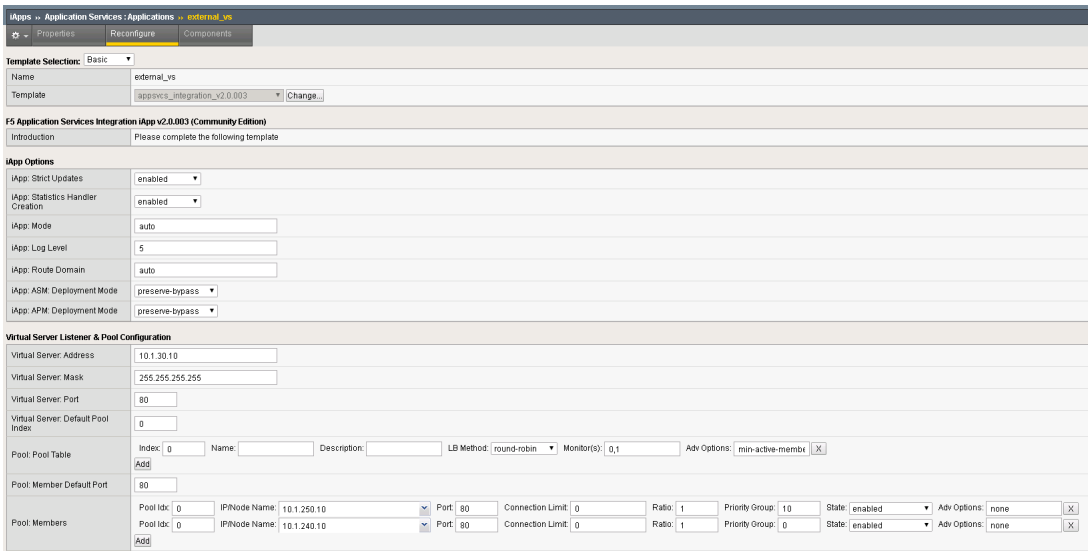
```
# Create L4-L7 services
python iapps/deploy_iapp_bigip.py -r 10.1.1.7 iapps/sample_http.json --strings pool__
↪addr=10.1.10.10 \
    --pool_members=0:10.1.240.10:80:0:1:10:enabled:none,0:10.1.250.
↪10:80:0:1:0:enabled:none --iapp_name external_vs

python iapps/deploy_iapp_bigip.py -r 10.1.1.8 iapps/sample_http.json --strings pool__
↪addr=10.1.30.10 \
    --pool_members=0:10.1.250.10:80:0:1:10:enabled:none,0:10.1.240.
↪10:80:0:1:0:enabled:none --iapp_name external_vs

python iapps/deploy_iapp_bigip.py -r 10.1.1.7 iapps/sample_http.json --strings pool__
↪addr=10.1.10.100 \
    --pool_members=0:10.1.240.10:80:0:1:10:enabled:none,0:10.1.250.
↪10:80:0:1:0:enabled:none --iapp_name internal_vs

python iapps/deploy_iapp_bigip.py -r 10.1.1.8 iapps/sample_http.json --strings pool__
↪addr=10.1.30.100 \
    --pool_members=0:10.1.250.10:80:0:1:10:enabled:none,0:10.1.240.
↪10:80:0:1:0:enabled:none --iapp_name internal_vs
```

You can view the parameters used to configure the iApp under iApps -> Application Services.



Testing Connections

This demo is designed to provide a solution with the following attributes.

- Two BIG-IP devices in separate Data Centers (Regions, Availability Zone, etc...)
- Two backend servers in separate DC
- The two DC are routable to each other via L3
- Provide recursive DNS for internal clients

The desired behavior for requests

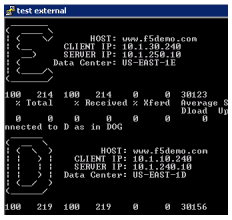
- External clients round-robin between backend servers

- Persist External client requests to original DC server if requests move between DC
- Internal client requests will have affinity to local DC server

Testing External Connections

Find the "Test External" link.

Double-click on it and you should see:



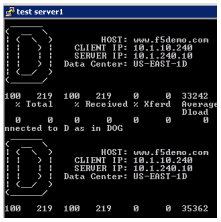
The "Test External" link is simulating requests from an external client. BIG-IP DNS is configured to use round-robin load balancing between the two backend servers.

From Google Chrome find the link for "www.f5demo.com". The Windows Desktop client is configured to act like an external client.

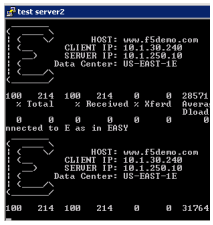
Question Using Google Chrome the requests will always go back to the same server, why? (Hint: Look at the Optional Exercise from Lab 2).

Testing Internal Connections

Now run the "Test Server1" link.



and run the "Test server2" link.



The "test-server[1-2]" links are simulating requests from internal clients. Note that BIG-IP DNS is configured to prefer requests to the same Data Center.

Question Can you explain how this is being done? (Hint: Look at the Optional Exercise from Lab 2).

Disabling a pool member

On BIG-IP 1 go to iApps -> Application Services

Click on “external_vs” and click reconfigure.

Disable the external_vs pool member 10.1.240.10.

Re-run the external tests.

Please proceed to Lab 4. *Welcome to the Lab 4 Lab Guide*

CHAPTER 15

Optional Exercises

Changing the requirements

Can you change the behavior to the following:

1. External requests will not persist to the same backend server (still round-robin, Hint: one change to each external LTM Virtual Server)
2. Flip the affinity of the internal requests (could be done via either LTM/DNS)

Automating the change

The second change "flipping the affinity" can be done via changing the automation script to change how the LTM Virtual Servers are deployed. Reset the deployment and deploy with an updated deployment that implements that change.

CHAPTER 16

Lab Appendix

The Python Script that is used in this lab was created by Eric Chen. This is considered F5 contributed software. K80012344 covers this.

From K80012344.

> **F5 contributed software**
>
> *F5 employees create and test software contributed to GitHub as part of supporting the development community related to F5 products. However, if the software is not a specific tagged release listed as F5 Supported in this article, and if the software is not also available on the F5 Downloads site, F5 does not provide technical support for them, and may refer you back to GitHub for community support for the project in question. For more information and for community discussions, refer to the information available for that software in the GitHub repository.*

The Script

Enable DNS Sync

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action enable_sync
```

This performs the GUI steps of *Enabling DNS Sync*

Add Data Center

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action add_datacenter --datacenter [Data Center Name]
```

This performs the GUI steps of *Add Data Center*

Save Config

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action save_config
```

This performs the TMSH equivalent of:

```
save /sys config
save /sys config gtm-only
```

This ensures that the running configuration is saved to disk.

Syncing BIG-IP DNS

```
python bigip_dns_helper.py --host=[MGMT IP of BIG-IP that you will WIPE OUT DNS␣
↪config] \
                           --action gtm_add
                           --peer_host=[MGMT IP of BIG-IP you want to copy from]
                           --peer_selfip [SELF IP of BIG-IP you want to copy from]
```

This performs the steps of *Syncing BIG-IP DNS*

Create DNS Cache

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action create_dns_cache
```

This performs the steps of *DNS Cache*

Create External DNS Profile

```
python bigip_dns_helper.py --host=[MGMT IP]  --action create_external_dns_profile
```

This performs the steps of *External DNS Profile*

Create Internal DNS Profile

```
python bigip_dns_helper.py --host=[MGMT IP]  --action create_internal_dns_profile
```

This performs the steps of *Internal DNS Profile*

Create External DNS Listener

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action create_external_dns_listener
                           --listener_ip [Listener IP]
```

This performs the steps of *External DNS Listener*

Create Internal DNS Listener

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action create_internal_dns_listener
                           --listener_ip [Listener IP]
                           --internal_network [Network/CIDR]
```

This performs the steps of *Internal DNS Listener*

Import iApp Template

```
python iapps/import_template_bigip.py  --impl [TCL file] \
                                       --apl  [APL File] \
                                              [MGMT IP] \
                                              [Name of Template]
```

This will import the Application Services Integration iApp.

Deploy iApp

```
python iapps/deploy_iapp_bigip.py -r          [MGMT IP] \
                                              [JSON Input] \
                                  --strings  pool__addr=[Virtual Server IP] \
                                  --pool_members=0:[Member IP]:[Member␣
↪Port]:0:1:[Priority Group]:enabled:none,\
                                                 0:[Member IP]:[Member␣
↪Port]:0:1:[Priority Group]:enabled:none
```

This will deploy an iApp. This is modified from the Application Services Integration iApp GitHub scripts to allow the specification of "strings" and "pool_members" from the CLI.

This performs the steps from *LTM Configuration*

Create DNS Virtual Server

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action create_vs \
                           --vip [LTM VS IP]:[LTM VS Port] \
                           --vip_translate [LTM VS External IP]:[LTM VS External Port]
                           --vs_name [DNS VS Name]
                           --server_name [Server Name (BIG-IP Device)]
```

This performs the steps from *Step 1: Virtual Servers*

Create DNS Pools

```
python bigip_dns_helper.py --host=[MGMT IP] \
                           --action create_pool \
                           --name [Pool Name]
```

This performs the steps from *Step 2: Pools*. The pool is configured with a Topology LB method.

Create DNS Wide IP

```
python bigip_dns_helper.py --host=[MGMT IP]  \
                           --action create_wideip \
                           --name [DNS Name] --pool [DNS Pool #1],[DNS Pool #2]
```

This performs the steps from *Step 3: Wide IPs*. The Wide IP is configured with a Topology LB method.

Create Topology Regions

```
python bigip_dns_helper.py --host [MGMT IP] \
                           --action create_region \
                           --name [Region Name] --internal_network [Subnet #1],
↪[Subnet #2]
```

This performs the steps from *Step 4: Topology Regions*.

Create Topology Records

```
python  bigip_dns_helper.py --host [MGMT IP]
                            --action create_topology_record
                            --name [Topology Record]
```

This performs the steps from *Step 5: Topology Records*. An example of a topology record: "ldns: region /Common/internal_network server: pool /Common/internal_pool"

During Lab 3 we will utilize the F5 Python SDK to script the steps that were previously performed manually. The Application Services iApp will also be leveraged to provide Service Catalog of L4-L7 services.

CHAPTER 17

Welcome to the Lab 4 Lab Guide

This Lab will show how Jenkins as a build engine can help automate BIG-IP deployments. The automation will be done in UDF 2.0

This Lab will perform following tasks:

1. License two BIG-IP
2. Reset the config
3. deploy the udf.sh script from lab3 to provision BIG-IP DNS and LTM, create DNS clusters, join LTM and DNS, deploy iapps and deploy BIG-IP DNS configuration

This will be done without touching the BIG-IPs after the default config is loaded.

The goal of this lab is to demonstrate how complex automation tasks can be hidden behind a generic automation engine. In this case the deployment is in UDF 2.0. Running through the Lab demonstrates how easy it is to change the destination to AWS or Azure or private cloud environments with keeping the shim layer unchanged.

CHAPTER 18

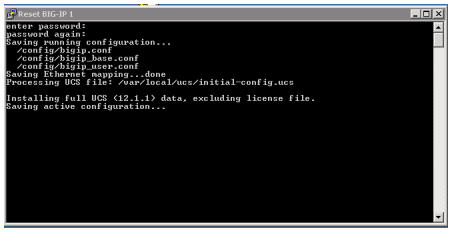
Step 0 - Restoring the BIG-IP Configuration

This step is to cleanup the BIG-IP config that was created in Lab 2 and 3. RDP into the Windows jump host.

Reset both BIG-IP to be the same state as after Lab 1.

Find the “Resetting” links on the Desktop.

Double-click on both of these and you should see a window appear briefly like the following.



Verify that you no longer see the changes that were previously deployed.

CHAPTER 19

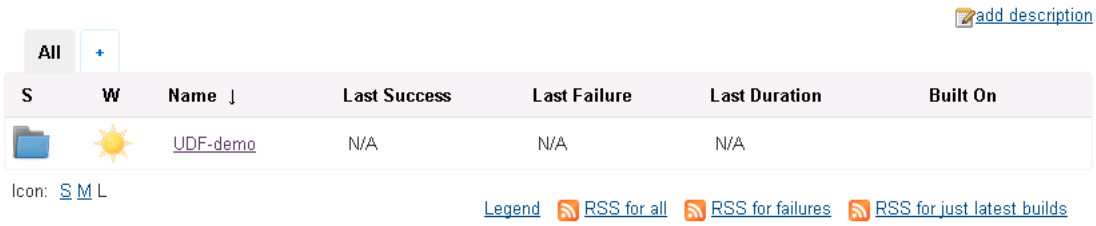
Step 1 - login into Jenkins

This step demonstrates how to login into Jenkins and find the deployment folder.

After both BIG-IP are active again open Chrome in the RDP Session and click on the Jenkins link.

Login to the jenkins server. The credentials are on the RDP Desktop in the “Jenkins credentials.txt” file.

After login to the Jenkins Web interface, please note the UDF-demo folder.



The UDF-folder contains a collection of Projects/Jobs that can be started or build by Jenkins. Click the UDF-demo folder. In the UDF demo folder there are several Projects from job 0 to job 3.

In the next steps these Jobs wil be build and the result verified.

CHAPTER 20

Step 2 - deploying F5 config via automation scripts using Jenkins

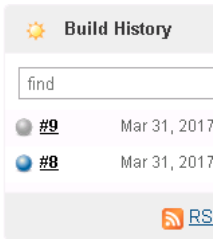
The config deployment is split into multiple steps.

Job 0

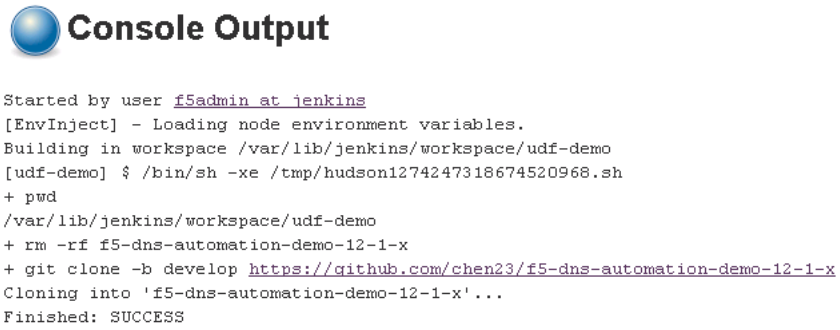
The task of Job 0 is to download the repository for this lab from github to the Jenkins Servers local HDD. This enables customer to test and deploy services always from the latest stable or develop cycle.

In the Jenkins GUI click on “Job 0 - clone git repository”. In order to run the project click on the left side the “Build Now” link.

Note that at the left side the “Building History” list adds a new build.



Click at the new build number. In the next screen there is a link called “Console Output”



Click on “Console Output”

In the middle of the screen the console output is displayed.

Question How is the deployment status?

In the task list click on UDF-demo to return to the folder

Job 1a and 1b

The task of Job 1a and Job 1b is to license the BIG=IP instances.

there are multiple ways to build a project. In this case click on the “Built On” button at the right side of the screen for Job 1a and Job 1b.

Question What is the console status after the Job completed?

Job 2a and 2b

The task of Job 2a and 2b is to send the reset scripts from lab 2 and lab3 to the BIG-IP’s.

Build the jobs 2a and 2b. Check the console output for both Projects and their status responses for success.

Question What is the console status after the Job completed?

Job 3

In Job 1 and 2 the BIG-IP were prepared to receive the configuration. Job 3 deploys the udf.sh script from lab 3 to both BIG-IP’s. After this deployment the BIG-IP’s are ready to serve the service.

Click “Built On” and monitor the console output.

Question What is the console status after the Job completed?

In Lab 3 we launched the automation scripts via ssh/shell scripts. During this lab we will utilize Jenkins to perform the automation steps.

Jenkins can provide a more standard way of deploying and monitoring the health of automated workflows.

The goal of this lab is to demonstrate how complex automation tasks can be hidden behind a generic automation engine. In this case the deployment is in UDF 2.0. Running through the Lab demonstrates how easy it is to change the destination to AWS or Azure or private cloud environments with keeping the shim layer unchanged.

CHAPTER 21

Jenkins Pipeline

During the prior lab *Welcome to the Lab 4 Lab Guide* Jenkins was used to provide a frontend tool to launch the automation scripts from *F5 Python SDK*.

This lab will leverage Jenkins Pipeline to create a workflow that will validate the individual steps that are executing in a more declarative manner.

CHAPTER 22

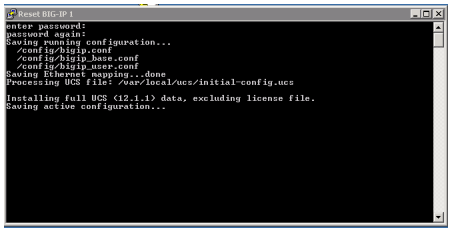
Restoring the BIG-IP Configuration

This step is to cleanup the BIG-IP config that was created in prior Labs. RDP into the Windows jump host.

Reset both BIG-IP to be the same state as after Lab 1.

Find the “Resetting” links on the Desktop.

Double-click on both of these and you should see a window appear briefly like the following.



Verify that you no longer see the changes that were previously deployed.

CHAPTER 23

Creating a pipeline

Jenkins allows you to put together a collection of actions, branch on conditions, and measure the results.

This can be put together to put together a series of stages.

- Stage 1: Enable DNS Sync
- Stage 2: Cluster DNS Devices
- Stage 3: Add additional DNS Configuration
- Stage 4: Import App Svcs iApp Template
- Stage 5: Deploy App Svcs iApp Template
- Stage 6: Add DNS Configuration
- Stage 7: Verify Configuration

To describe these stages we can create a "Jenkinsfile" that will instruct Jenkins to execute these stages.

The following will pull down a copy of the scripts used in this lab and execute the steps to create a DNS cluster.

```
stage('clone git repo') {
   node {
     git url: 'https://github.com/f5devcentral/f5-dns-automation-demo-12-1-x.git',␣
↪branch:'master'
   }
}

stage('enable dns sync') {
  node {
    dir ('lib') {
                  sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action enable_sync'
                  sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action add_datacenter --datacenter ' + params.dc1 + ''
                  sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action add_datacenter --datacenter ' + params.dc2 + ''
                  sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action add_server  --datacenter ' + params.dc1 + ' --server_name bigip1 --server_ip=
↪' + params.bigip1_selfip + ''
```

```
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action add_server  --datacenter ' + params.dc2 + ' --server_name bigip2 --server_ip=
↪' + params.bigip2_selfip + ''
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action save_config'
    }
  }
}
stage('gtm add') {
  node {
      dir ('lib') {

                    sh 'sleep 3'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + ' --
↪action gtm_add --peer_host=' + params.bigip1 + ' --peer_selfip ' + params.bigip1_
↪selfip + ''
                    sh 'sleep 3'
      }
  }
}
stage('additional dns setup') {
  node {
      dir ('lib') {

                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + '  --
↪action create_dns_cache'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + '  --
↪action create_dns_cache'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + '  --
↪action create_external_dns_profile'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + '  --
↪action create_external_dns_profile'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + '  --
↪action create_internal_dns_profile'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + '  --
↪action create_internal_dns_profile'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + '  --
↪action create_external_dns_listener --listener_ip ' + params.bigip1_dns_listener + '
↪'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + '  --
↪action create_external_dns_listener --listener_ip ' + params.bigip2_dns_listener + '
↪'
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + '  --
↪action create_internal_dns_listener --listener_ip ' + params.bigip1_dns_listener +
↪' --internal_network ' + params.internal_network + ''
                    sh 'python bigip_dns_helper.py --host=' + params.bigip2 + '  --
↪action create_internal_dns_listener --listener_ip ' + params.bigip2_dns_listener +
↪' --internal_network ' + params.internal_network + ''
                    sh 'python bigip_dns_helper.py --host=' + params.bigip1 + ' --
↪action save_config'
                    sh 'sleep 3'
      }
  }
}
```

Note that in this code example that we are no longer using hardcoded IP addresses. These are now parameters that can be input when the pipeline is executed.

The full code can be found on GitHub.

CHAPTER 24

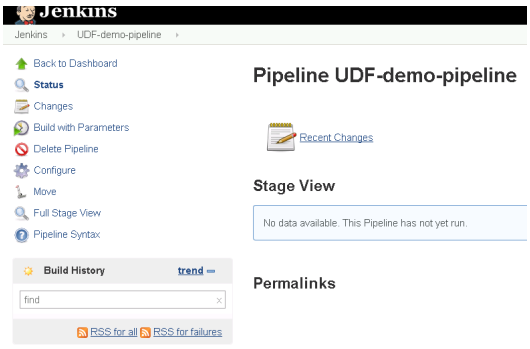
Launching the pipeline

After both BIG-IP are active again open Chrome in the RDP Session and click on the Jenkins link.

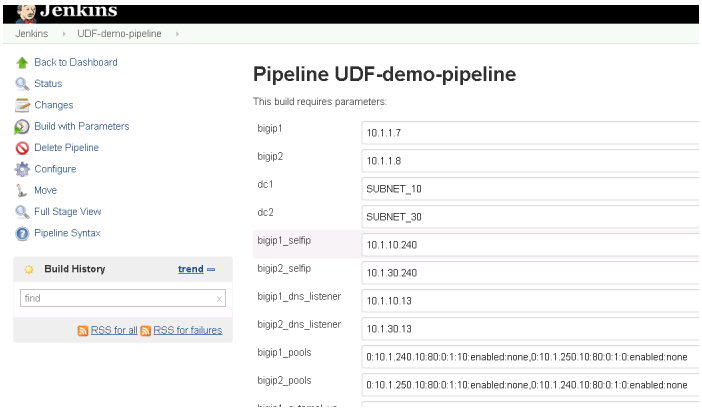
Login to the jenkins server. The credentials are on the RDP Desktop in the “Jenkins credentials.txt” file.

After login to the Jenkins Web interface, please note the UDF-demo-pipeline folder.

Click on “UDF-demo-pipeline” and you should see.

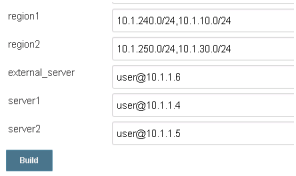


In order to run the project click on the left side the “Build with Parameters” link.

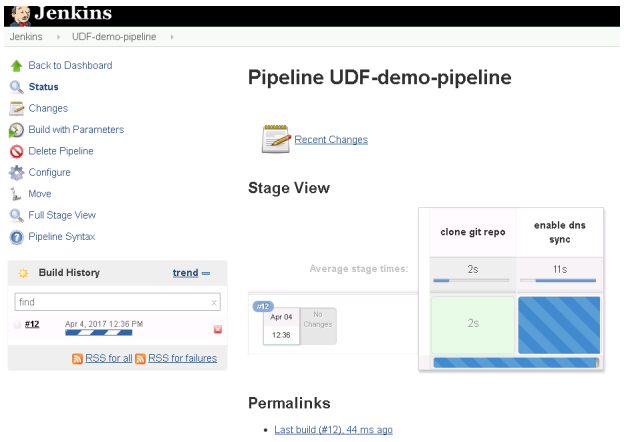


In previous labs all the input parameters were statically defined making the solution only usable with a specific network. By utilizing parameters in Jenkins we are creating a solution that can be deployed more dynamically.

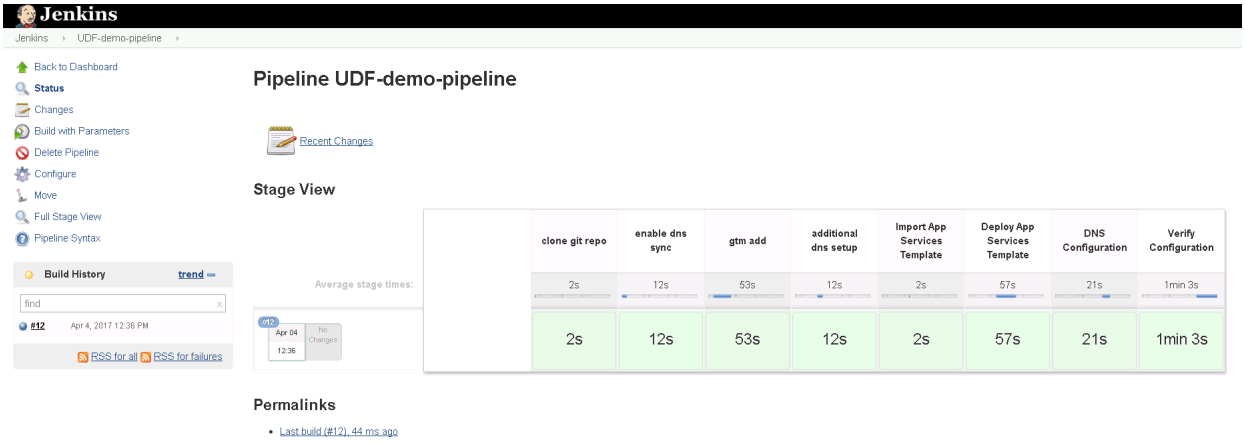
Scroll to the bottom of the page and find the "Build" button and click on it.



While the build is running you will see.



When the pipeline is complete you will see:


Jenkins
Jenkins › UDF-demo-pipeline ›
Back to Dashboard
Status
Changes
Build with Parameters
Delete Pipeline
Configure
Move
Full Stage View
Pipeline Syntax
Build History
trend
find
#12
Apr 4, 2017 12:36 PM
RSS for all
RSS for failures
Pipeline UDF-demo-pipeline
Recent Changes
Stage View
Average stage times:
clone git repo
enable dns sync
gtm add
additional dns setup
Import App Services Template
Deploy App Services Template
DNS Configuration
Verify Configuration
2s
12s
53s
12s
2s
57s
21s
1min 3s
#12
Apr 04
12:36
No Changes
Permalinks
Last build (#12), 44 ms ago

CHAPTER 25

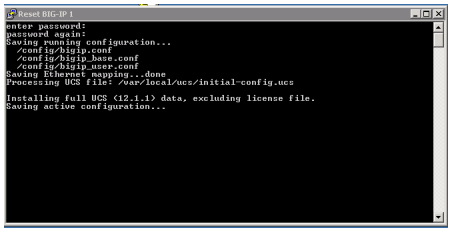
Failing Tests

In the previous exercise we deployed a successful deployment. In this exercise we will purposely break the pipeline.

Reset both BIG-IP to be the same state as after Lab 1.

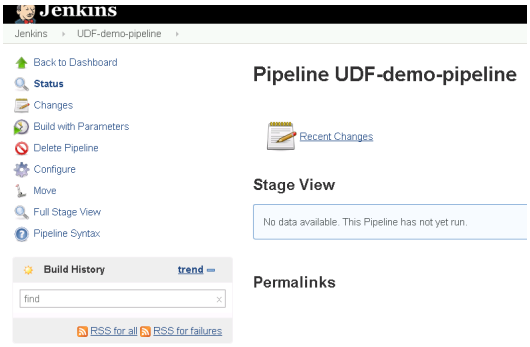
Find the “Resetting” links on the Desktop.

Double-click on both of these and you should see a window appear briefly like the following.



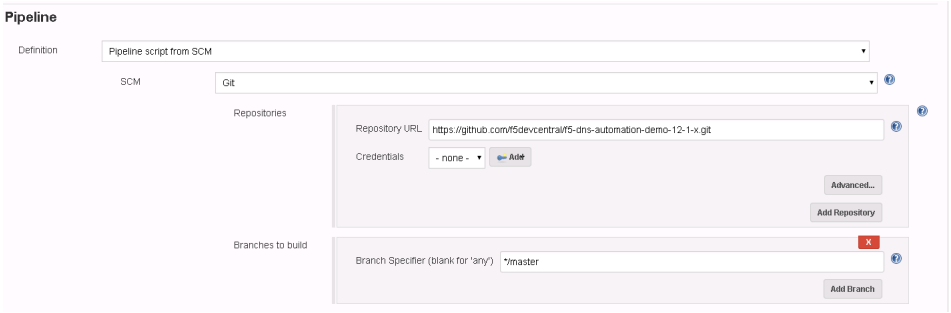
Verify that you no longer see the changes that were previously deployed.

Go back to the ‘UDF-demo-pipeline’ page.



This time click on the “Configure” link.

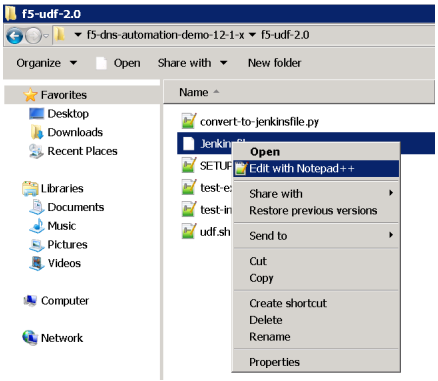
Find the “Pipeline” section.

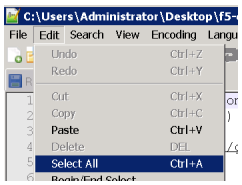


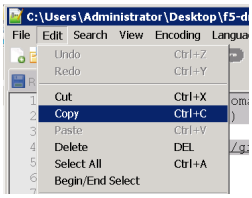
The current pipeline is configured to pull down a copy of the file “f5-udf-2.0/Jenkinsfile” from the f5devcentral GitHub repository. We are going to download a local copy of this file and modify it to simulate a failure.

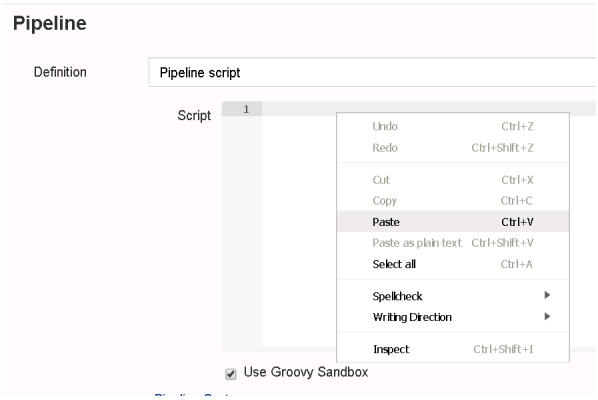
Select the pulldown for “Pipeline script from SCM” and change to “Pipeline Script”.

Go to GitHub and copy the text into your clipboard. If you are using a Mac or your RDP client is not supporting copy and paste you can also find a copy of the file on your Desktop. Browse to “Desktop\f5-dns-automation-demo-12-1-x\f5-udf-2.0”. Right click on the Jenkinsfile and “Edit with Notepad++”.







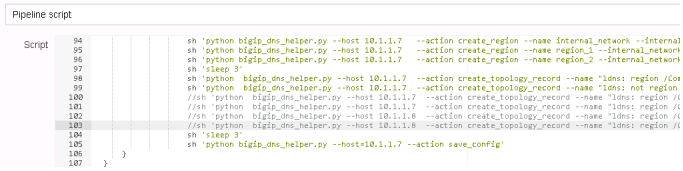


Paste the contents into the script text area.

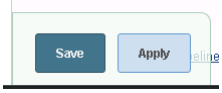
Comment out the following lines (around line 100).

```
//sh 'python  bigip_dns_helper.py --host ' + params.bigip1 + '  --action create_
↪topology_record --name "ldns: region /Common/region_1 server: region /Common/region_
↪1"'
//sh 'python  bigip_dns_helper.py --host ' + params.bigip1 + '  --action create_
↪topology_record --name "ldns: region /Common/region_2 server: region /Common/region_
↪2"'
//sh 'python  bigip_dns_helper.py --host ' + params.bigip2 + '  --action create_
↪topology_record --name "ldns: region /Common/region_1 server: region /Common/region_
↪1"'
//sh 'python  bigip_dns_helper.py --host ' + params.bigip2 + '  --action create_
↪topology_record --name "ldns: region /Common/region_2 server: region /Common/region_
↪2"'
```

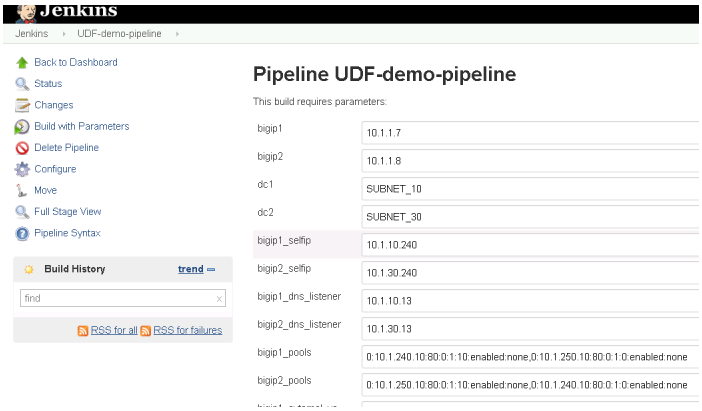
The result should look something like:



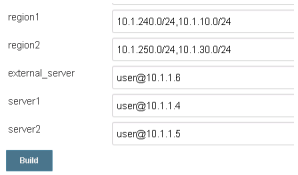
Now click on the Save Button.



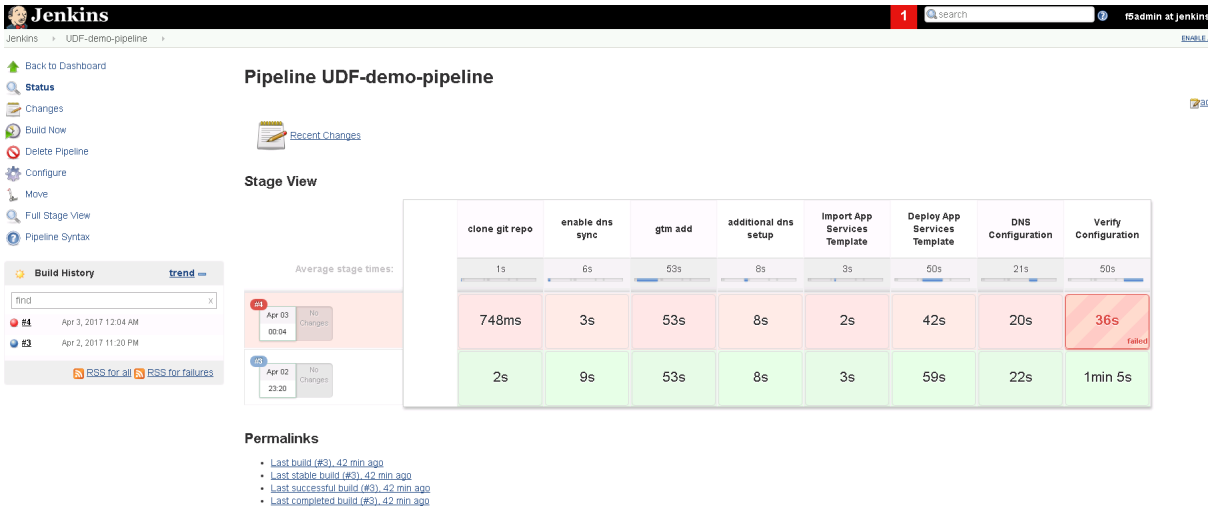
Back on the pipeline page find the “Build with Parameters” link.



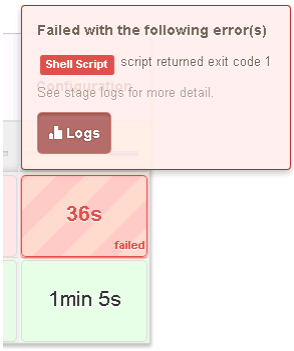
Scroll to the bottom of the page and find the “Build” button and click on it.



Once the build completes you should see a failure.

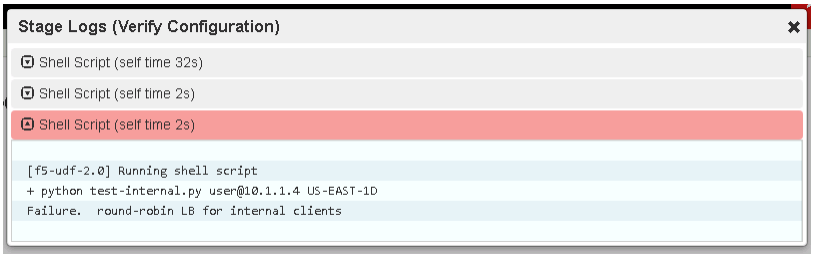


Hover your mouse over the failure.



Click on “Logs” to see the detail and expand the failing task.

Hover your mouse over the failure.



In this case the script “test-internal.py” exited with a non-zero exit code. This causes Jenkins to treat this as a failure. In this case the script was only expecting to see responses from a single Data Center and instead received responses from both Data Centers due to the lack of topology records.

CHAPTER 26

Indices and tables

- genindex
- search