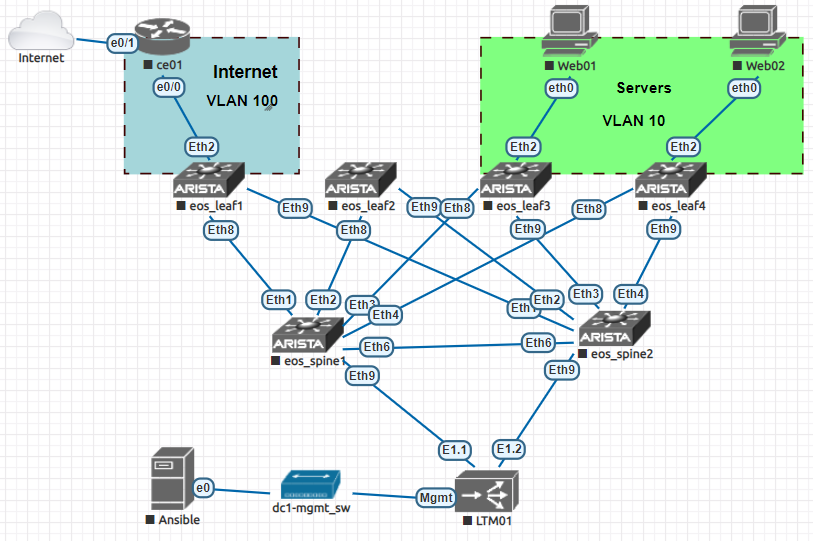
5. Automating Application Delivery with F5 LTM and Ansible

In this chapter, we will outline how to automate F5 BIG-IP platforms running as load balancers or LTM appliances. We will explore how to interact with F5 LTM nodes using Ansible and how to onboard these devices and accelerate application deployment hosed by these devices using various Ansible modules. We will base our illustration based on the below sample network diagram of a single F5 LTM nodes connected to DC Switches.



The main recipes covered in this chapter is shown below

* Building Ansible Network Inventory.
* Connecting and User Authentication to BIG-IP.
* Configuring Basic System settings on BIG-IP.
* Configuring Interfaces on BIG-IP.
* Configuring VLANs and Self-IPs on BIG-IP.
* Configuring Static Routes on BIG-IP.
* Configuring Nodes on BIG-IP.
* Configuring Load Balancing Pool on BIG-IP.
* Configuring Virtual Servers on BIG-IP.
* Retrieving Operational Data from BIG-IP.

**Technical requirements**

All the code used in the recipes in this chapter can be found on the below GitHub Repo

<https://github.com/PacktPublishing/Network-Automation-Cookbook/tree/master/ch5_f5>

Below are the Software releases that this chapter is based on

* Ansible Machine Running Ubuntu 16.04
* Ansible 2.8
* F5 BIG-IP running BIG-IP 13.1.1 Build 0.0.4 Final

**Building Network Inventory**

In this recipe, we will outline how to build and structure our Ansible Inventory to describe our simple F5 BIG-IP nodes. Building an Ansible inventory is a mandatory step in order to tell Ansible how to connect to the managed devices.

**Getting Ready**

We create a new folder that will host all the files that we will create in this chapter. The new folder is named ***ch5\_f5\_netops***.

**How to do it..**

1. Inside the new folder (ch5\_f5\_netops) we create ***hosts*** file with the below content

$ cat hosts  
  
[ltm]

ltm01 ansible\_host=172.20.1.34

1. Create ansible.cfg file as shown below

$ cat ansible.cfg  
[defaults]  
inventory=./hosts  
retry\_files\_enabled=False  
gathering=explicit  
host\_key\_checking=False

**How it works..**

Since we have a single LTM nodes our hosts file is very simple we create a single group (called **ltm**) and we specify a single node in it which is ltm01 and we specify the management IP addresses for the nodes as the ansible\_host.

The management port on the BIG-IP must have this IP address configured and IP connectivity between the Ansible control machine and the BIG-IP node on this management port is established.

Finally, we create the ***ansible.cfg*** file and configure it to point to our ***hosts*** file to be used as ansible inventory file and we disable the setup module which is not needed when running ansible against network nodes.

**Connecting and Authenticating to BIG-IP Devices**

In this recipe, we will outline how to connect to BIG-IP Nodes from Ansible via the REST API exposed by BIG-IP in order to start managing the devices from Ansible. We are going to use Username and passwords to authenticate to the BIG-IP Node in our topology.

**Getting Ready**

In order to follow along with this recipe, an ansible inventory file should be constructed as per the previous recipe, also IP reachability between the Ansible Control machine and all the devices in the network must be established.

**How to do it..**

1. Inside the folder ch5\_f5\_netops create the folder group\_vars.
2. Inside the group\_vars folder create the YAML file **all.yml** with the below contents

conn\_parameters:

user: admin

password: admin

server: "{{ ansible\_host }}"

server\_port: 443

validate\_certs: no  
  
admin\_passwd: NewP@sswd

users:

- name: ansible

passwd: ansible123

role: all:admin

state: present

1. Create a new playbook with the name **pb\_f5\_onboard.yml** with the below contents

- name: Onboarding a New LTM

hosts: ltm01

connection: local

tasks:

- name: "P1T1: Create new Users"

bigip\_user:

username\_credential: "{{ item.name }}"

password\_credential: "{{ item.passwd }}"

partition\_access: "{{ item.role }}"

state: "{{ item.state | default('present')}}"

provider: "{{ conn\_parameters }}"

loop: "{{ users }}"  
  
 - name: "P1T1: Update admin Password"

bigip\_user:

username\_credential: admin

password\_credential: "{{ admin\_passwd }}"

state: present

provider: "{{ conn\_parameters }}"

**How it works..**

In order to establish a REST API connection with the BIG-IP system we need to provide some parameters in order for Ansible to initiate and establish the REST connection with the BIG-IP node. These parameters include the following

* Username/password to authenticate with the BIG-IP REST API
* IP address and port over which the REST API is deployed on BIG-IP node.
* Whether we validate the certificate for the BIG-IP node negotiated over the HTTPs session.

All these parameters we include them in a dictionary called **conn\_parameters** which we include it in the all.yml file under the group\_vars directory so as to be applied on any BIG-IP node.

By default, a new LTM device comes with the **admin/admin** default username and password for GUI and REST API access. We use these credentials as the user and password variables inside the conn\_parameters dictionary and we specify the ansible\_host variable as the IP address over which the REST API can be established over port 443 and we disable certificate validation since certificate on the BIG-IP node is self-signed.

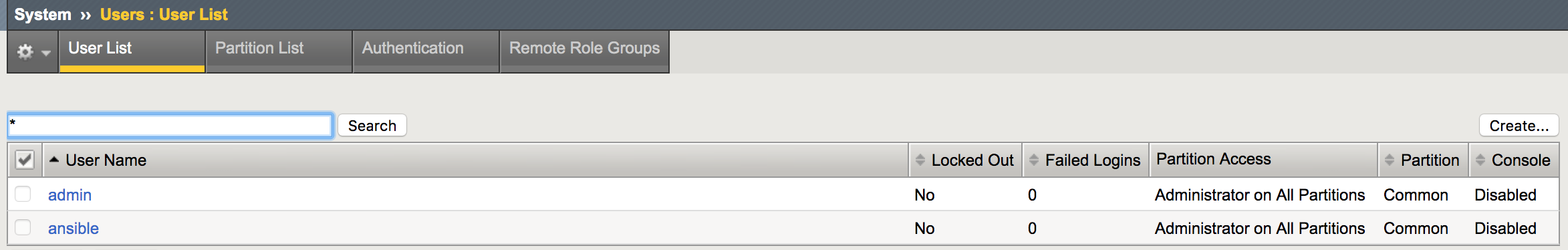
We create a new variable called **users** which holds all the new users that we want to configure on our LTM along with their role/privilege and in this case we want to provide administrative privilege for the user ansible across all the partitions on the LTM node.

We create a new playbook for onboarding a new LTM nodes and we specify in the first task we create the new users using **bigip\_user** module and we provide the parameters to establish the REST API to the provider attribute and we loop over all the users in our users variable to provision them.

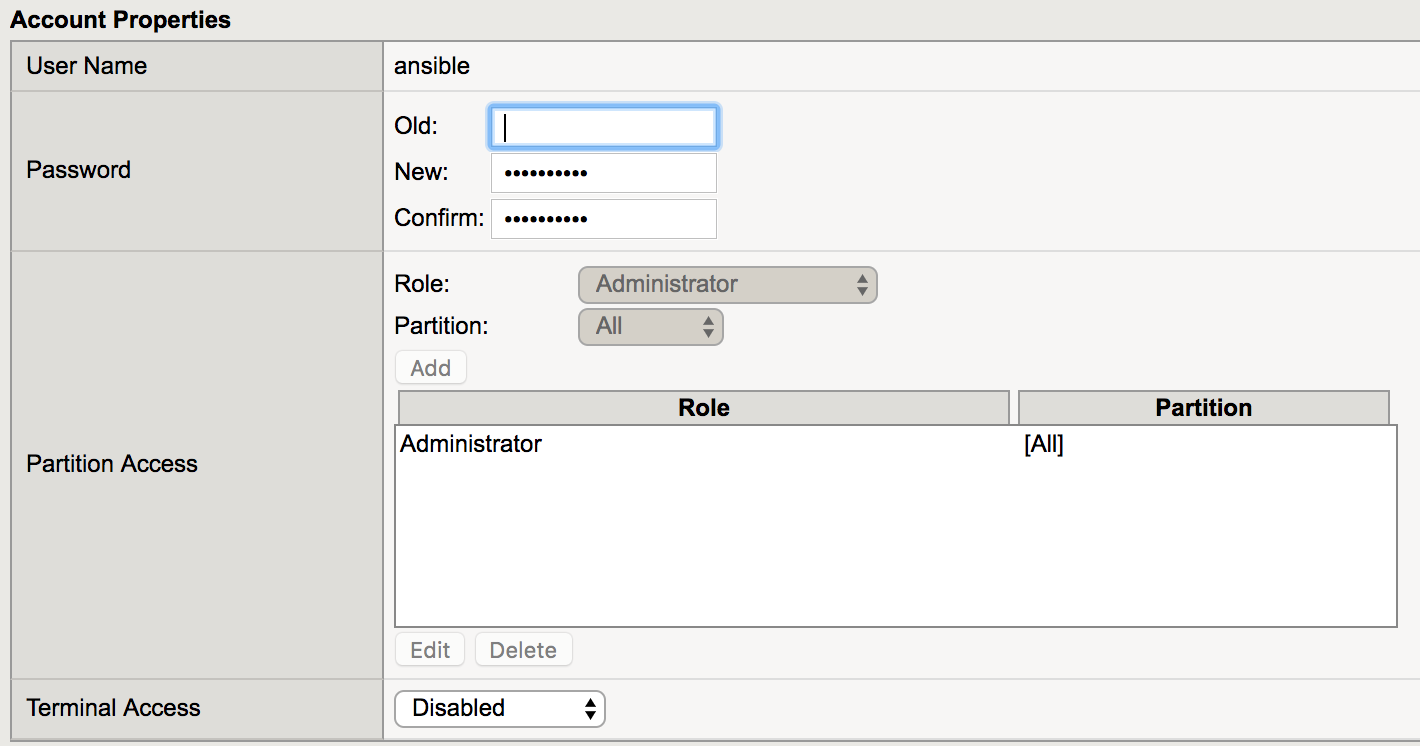
The second task is also using the bigip\_user in order to update the default admin profile on the LTM and change this default password to a new password specified in the admin\_passwd variable.

On the playbook level we are setting the connection to ***local*** since we are going to establish REST API connection from the ansible control machine as we want to prevent ansible from SSH to the LTM node.

The below screenshots outline the new Ansible user created on the BIG-IP node



The below screenshot outline the details of the ansible user created using the playbook.



Note:  
We are using plain text password for simplicity however plain text password shouldn’t never be used and nsible vault should be used to secure the password

**There is More..**

After adding the new ansible user we update the **conn\_parameters** dictionary with the new user that we have created in order to start manage the LTM nodes using this user as shown below

$ cat group\_vars/all.yml  
conn\_parameters:

user: **ansible**

password: **ansible123**

server: "{{ ansible\_host }}"

server\_port: 443

validate\_certs: no

< -- Output Omitted for brevity -->

**Configuring Generic System Options on BIG-IP Devices**

In this recipe, we will outline how to configure some basic system options like hostname, DNS and NTP on BIG-IP nodes. . We will understand how to setup all these system level parameters using the various Ansible modules available.

**Getting Ready**

To follow along with this recipe, an ansible inventory is assumed to be already setup and IP connectivity between Ansible and the BIG-IP nodes is already established with the correct user credentials.

**How to do it..**

1. Update the **all.yml** file inside the group\_vars folder as shown below

$ cat group\_vars/all.yml  
  
< -- Output Omitted for brevity -->  
  
domain: lab.net

nms\_servers:

- 172.20.1.250

1. Create a new folder called **tasks** and create the file f5\_system.yml with the below contents

$ cat tasks/f5\_system.yml  
  
---  
- name: "Setup BIG-IP Hostname"

bigip\_hostname:

hostname: "{{ inventory\_hostname }}.{{ domain }}"

provider: "{{ conn\_parameters }}"

- name: "Setup BIG-IP DNS Servers"

bigip\_device\_dns:

ip\_version: '4'

name\_servers: "{{ nms\_servers }}"

provider: "{{ conn\_parameters }}"

- name: "Setup BIG-IP NTP Servers"

bigip\_device\_ntp:

ntp\_servers: "{{ nms\_servers }}"

provider: "{{ conn\_parameters }}"

1. In the ***pb\_f5\_onboard.yml*** file, add the below highlighted tasks

$ cat pb\_f5\_onboard.yml  
  
< -- Output Omitted for brevity -->

- name: "P1T3: Configure System Parameters"

import\_tasks: "tasks/f5\_system.yml"

tags: system

**How it works..**

In order to configure the various system parameters on BIG-IP nodes we use a separate module for each task. We group all these tasks in a single file called **f5\_system.yml** under the **tasks** folder and inside this file we use three separate tasks/modules as shown below

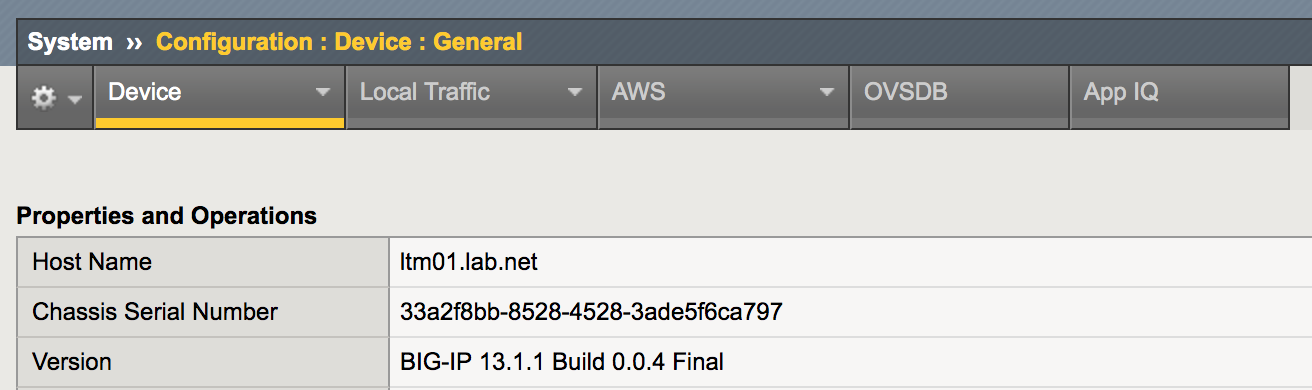
* **bigip\_hostname** to setup the hostname
* **bigip\_device\_dns** to setup the DNS server that BIG-IP node will use
* **bigip\_device\_ntp** to setup the NTP servers on the BIG-IP node.

All these modules take the **conn\_parameters** dictionary to correctly setup how to communicate to the REST API of the BIG-IP node.

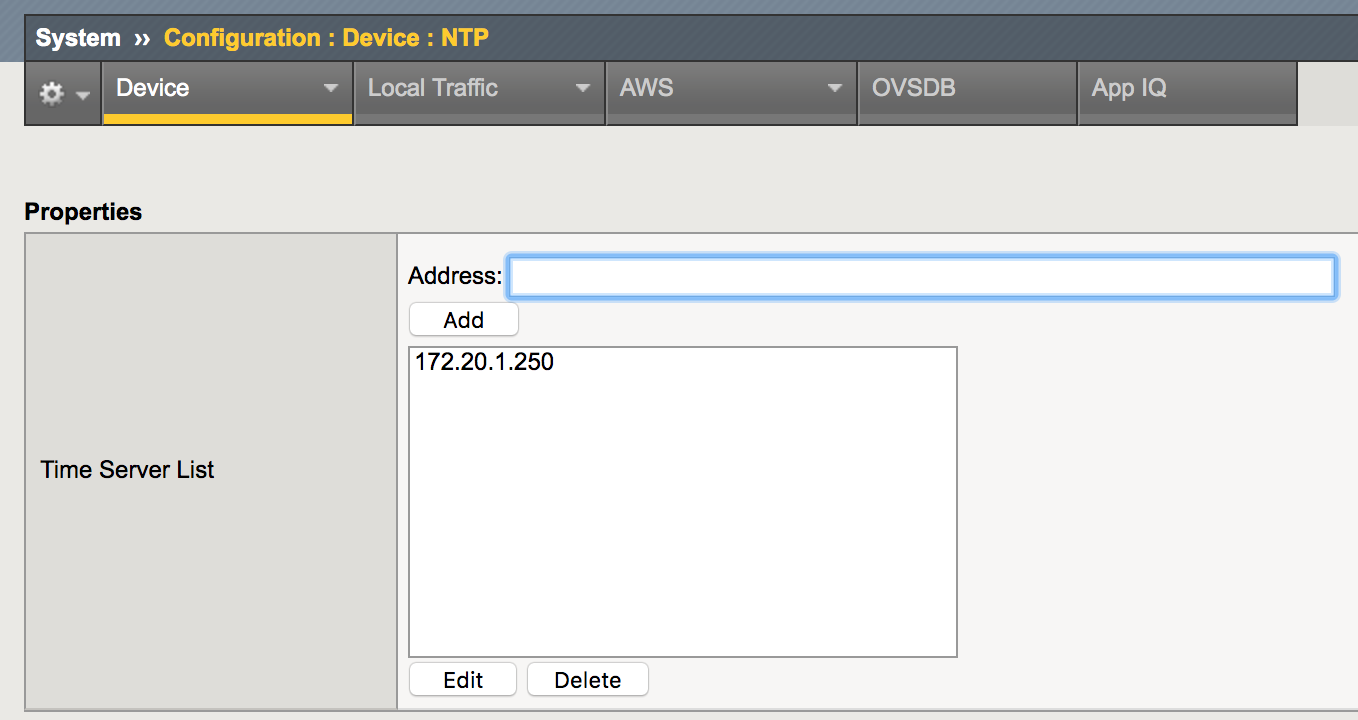
In this sample topology we use a single server as the DNS and NTP and we describe it using the **nms\_servers** variable in the all.yml file to apply to all our nodes in our ansible inventory.

In order to configure the hostname we need to supply a FQDN for the device so we configure our **domain** again under the **all.yml** file and use it in conjunction with the device name to setup its hostname.

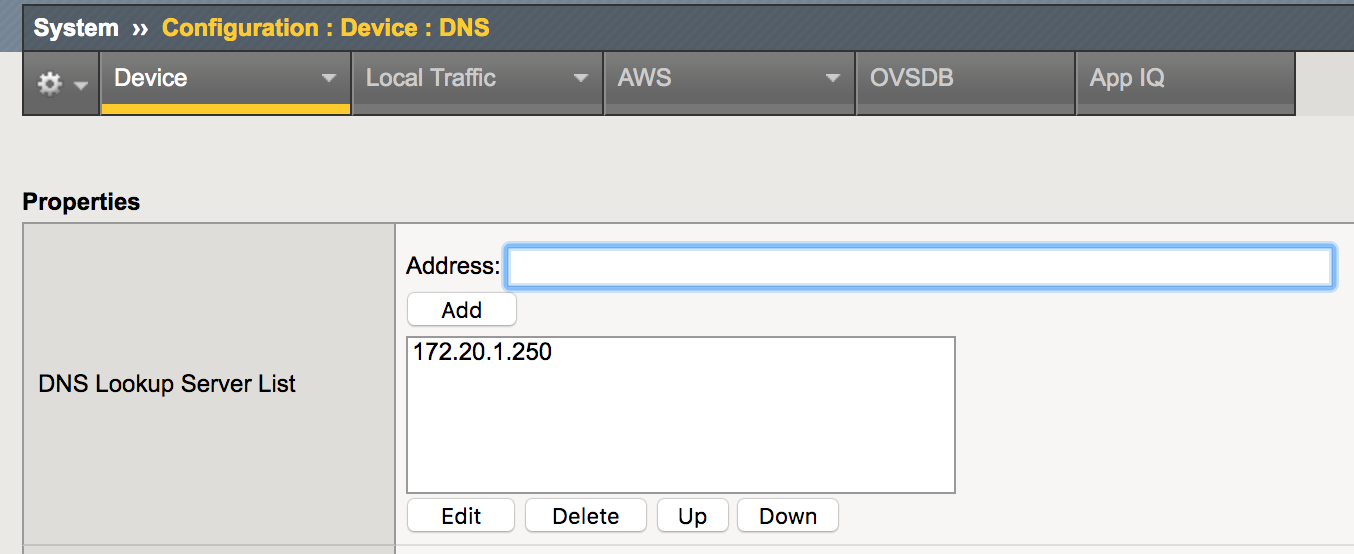
After running this playbook we can see that all the configuration is applied to the BIG-IP node, the below screenshot outline the hostname is correctly provisioned.



The NTP configuration is deployed correctly as per the below screenshot



The DNS is configured correctly as per the below screenshot



**Configuring Interface and Trunks on BIG-IP Devices**

In this recipe we will outline how to setup trunks on BIG-IP Devices. Trunk ports on BIG-IP nodes are to provide increased redundancy for the device by combining multiple interface into a single logical interface and it is very similar to port-channels in traditional network vendors.

**Getting Ready**

To follow along with this recipe, an ansible inventory is assumed to be already setup and IP connectivity between Ansible and the BIG-IP nodes is already established with the correct user credentials.

**How to do it..**

1. Create the host\_vars folder and create a ltm01.yml file with the below contents

$ host\_vars/ltm01.yml  
----  
phy\_interfaces:

- 1.1

- 1.2

trunks:

- name: po1

members: "{{ phy\_interfaces }}"

1. Under the **tasks** folder add a new file called **f5\_interfaces.yml** with the below contents

$ cat tasks/f5\_interfaces.yml  
  
---

- name: Create a Port channel on BIG-IP

bigip\_trunk:

name: "{{ item.name}}"

interfaces: "{{ item.members }}"

link\_selection\_policy: maximum-bandwidth

frame\_distribution\_hash: destination-mac

lacp\_enabled: no

provider: "{{ conn\_parameters }}"

state: present

loop: "{{ trunks }}"

1. Update the playbook **pb\_f5\_onboard.yml** with the below new task

$ cat pb\_f5\_onboard.yml

< -- Output Omitted for bevirty -->

- name: "P1T4: Configure Interfaces"

import\_tasks: "tasks/f5\_interfaces.yml"

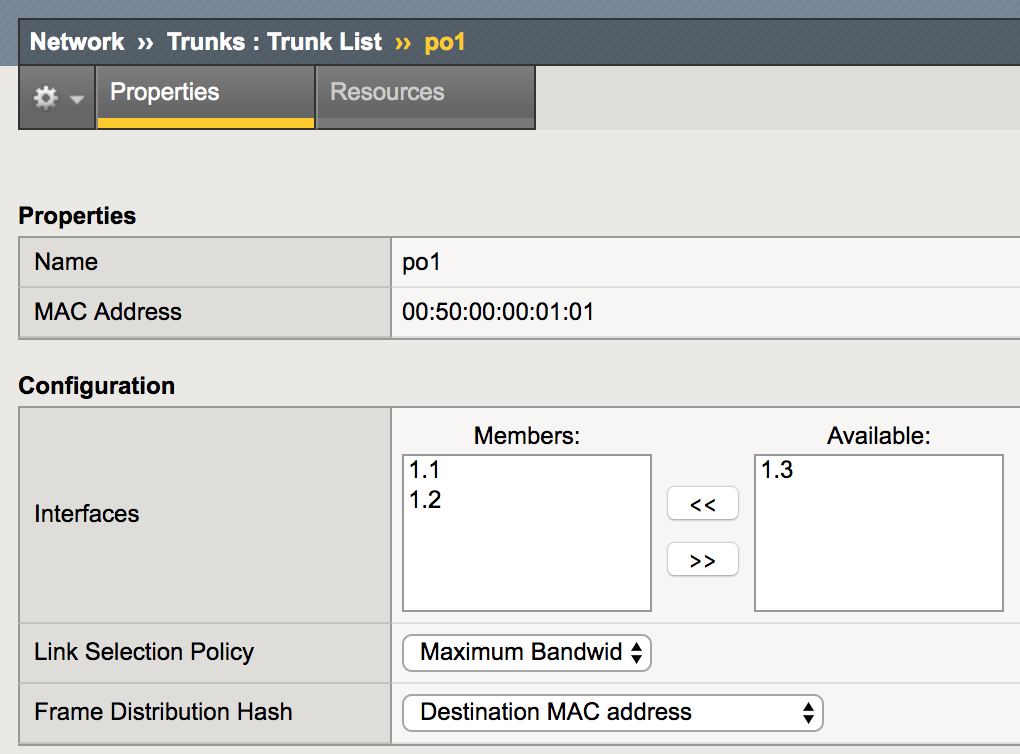
tags: intfs

**How it works..**

We define the host specific data for the LTM device under the host\_vars in a file called ltm01.yml. In this file we define the physical interface on the LTM node under the variable **phy\_interfaces** and we define another variable which is **trunks** in order to define the trunks available on the define. In the trunk variable we reference the **phy\_interfaces** in order to limit data duplication.

On the f5\_interfaces task file we add a new task using the **bigip\_trunk** module to provision the required trunks on the BIG-IP node and we loop over the **trunks** data structure to provision all the required trunk ports.

After running the playbook we can see that the required trunk interfaces are provisioned as shown below



**See Also..**

For more options regarding how to deploy trunk ports on the BIG-IP nodes like enabling LACP and how to adjust multiple LACP parameters on the trunk ports please refer to the blow URL.

[https://docs.ansible.com/ansible/latest/modules/bigip\_device\_ntp\_module.html#bigip-trunk](https://docs.ansible.com/ansible/latest/modules/bigip_device_ntp_module.html#bigip-trunke)-module

**Configuring VLANs and IPs on BIG-IP Devices**

In this recipe, we will outline how to configure VLANs on BIG-IP nodes, VLANs on the BIG-IP nodes are fundamental for traffic separation for the different applications hosted by the BIG-IP LTM nodes and they are fundamental to designate external (Internet facing) and internal (server facing) domains. We also outline how to assign IP address on the VLAN interfaces are we provision.

**Getting Ready**

To follow along with this recipe, an ansible inventory is assumed to be already setup and IP connectivity between Ansible and the BIG-IP nodes is already established with the correct user credentials. Also since all the VLANs in this setup will be deployed on trunk ports we need to have the trunk ports already provisioned as per the previous recipe.

**How to do it..**

1. Update the **ltm01.yml** file under the host\_vars with the below contents

$ cat host\_vars/ltm01.yml  
  
< -- Output Omitted for bevirty -->  
  
vlans:

- vlan: 100

description: Extrnal VLAN (Internet)

ip: 10.1.100.254/24

tagged\_intf: po1

- vlan: 10

description: Server VLAN10 (Internal)

ip: 10.1.10.254/24

tagged\_intf: po1

1. Update the **f5\_interfaces.yml** file under the tasks folder with the task to provision VLANs.

$ cat tasks/f5\_interfaces.yml  
  
< -- Output Omitted for bevirty -->

- name: Create VLANs on BIG-IP

bigip\_vlan:

tagged\_interfaces: "{{ item.tagged\_intf }}"

name: "VL{{item.vlan}}"

description: "{{ item.description }}"

tag: "{{item.vlan}}"

provider: "{{ conn\_parameters }}"

state: present

loop: "{{ vlans }}"

1. Update the **f5\_interfaces.yml** file under the tasks folder with the task to provision Self-IP on the respective VLANs.

$ cat tasks/f5\_interfaces.yml  
  
< -- Output Omitted for bevirty -->

- name: Provision IP addresses on BIG-IP

bigip\_selfip:

address: "{{ item.ip | ipv4('address') }}"

name: "VL{{ item.vlan }}"

netmask: "{{ item.ip | ipv4('netmask') }}"

vlan: "VL{{ item.vlan }}"

provider: "{{ conn\_parameters }}"

state: present

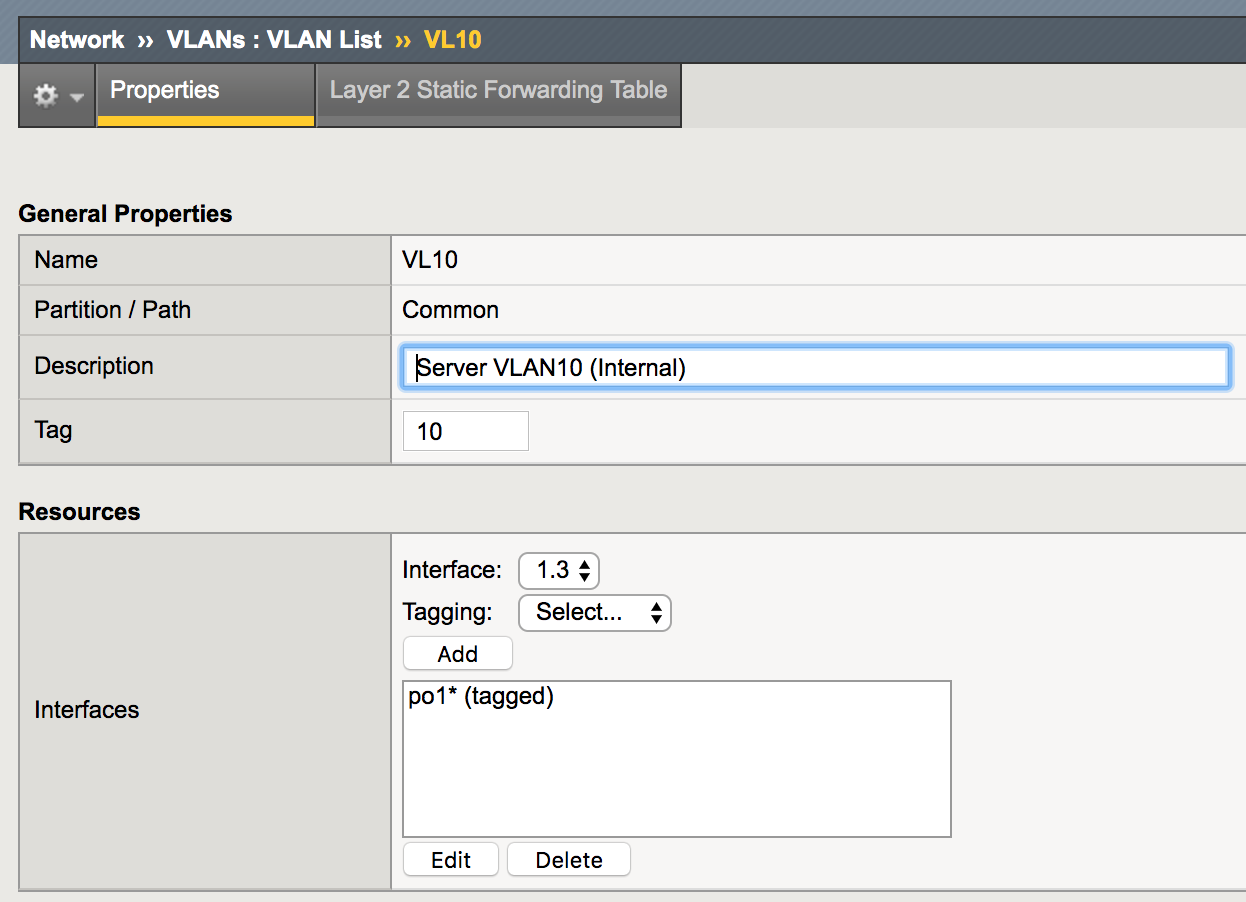
loop: "{{ vlans }}"

**How it works..**

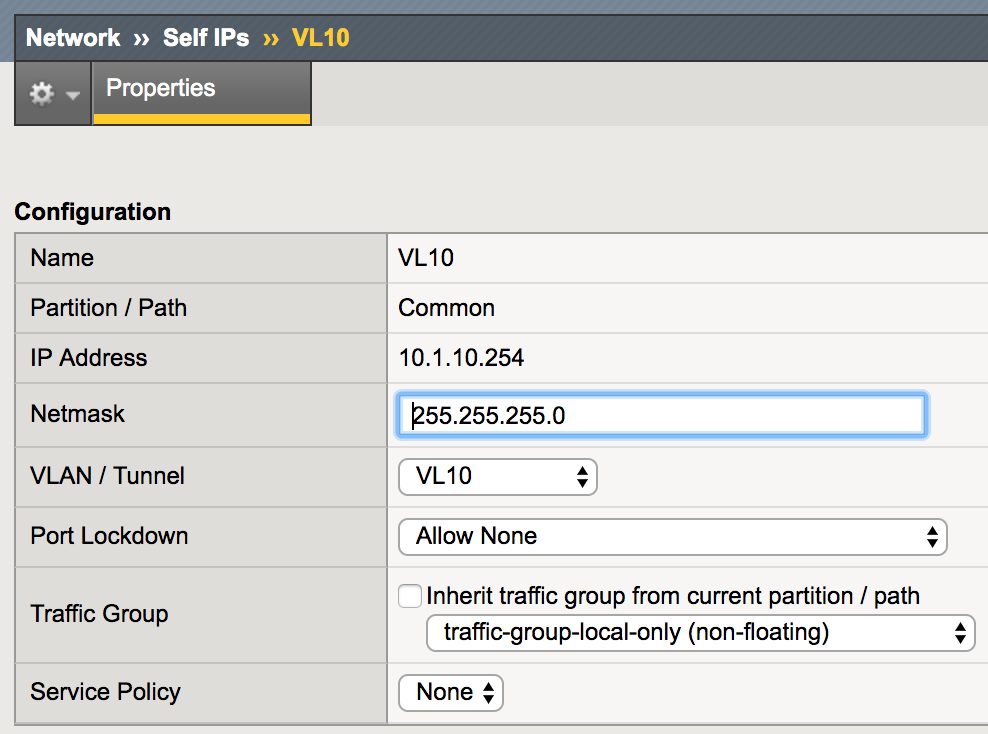
We add the **vlans** data structure in ltm01.yml under the host\_vars to declare all the VLANs that we need to provision on the LTM node along with the IP addresses associated with this VLAN.

We update the f5\_interfaces.yml file with a task using **bigip\_vlan** module to provision the vlans on the BIG-IP node and we loop over the **vlans** data structure to extract all the required parameters to setup the needed VLANs. Next, we add another task using **bigip\_selfip** ansible module to deploy the IP addresses on the VLANs.

After running the playbook again, we can see the VLANs and Self-IPs on the BIG-IP node as shown below

****

The Correct IP address is configured correctly on the VLAN interface as per the below screenshot

****

**See Also..**

For more options regarding how to deploy VLANs and self-IPs on the BIG-IP nodes like please refer to the blow URL.

**bigip-vlan**

<https://docs.ansible.com/ansible/latest/modules/bigip_selfip_module.html#bigip-vlan-module>

**bigip-selfip**

<https://docs.ansible.com/ansible/latest/modules/bigip_selfip_module.html#bigip-selfip-module>

**Configuring Static Routes on BIG-IP Devices**

After deploying the VLANs and IP addresses on the BIG-IP, we need to configure routing on the BIG-IP nodes in order to reach external destination. We use static routes in our topololgy in order to provision the required routing on the LTM node. In this recipe we will outline how to configure Static routes on BIG-IP.

**Getting Ready**

To follow along with this recipe, an ansible inventory is assumed to be already setup and IP connectivity between Ansible and the BIG-IP nodes is already established with the correct user credentials. Further, we need to deploy the VLANs and IP addresses in the BIG-IP node as per the previous recipe.

**How to do it..**

1. Update the **ltm01.yml** file under the host\_vars with the below contents

$ cat host\_vars/ltm01.yml  
  
< -- Output Omitted for bevirty -->  
  
routes:

- dst: 0.0.0.0/0

gw: 10.1.100.1

name: default\_route

1. Update the **pb\_f5\_onboard.yml**  with the below task

$ cat pb\_f5\_onboard.yml  
  
< -- Output Omitted for bevirty -->

- name: "P1T5: Setup External Routing"

bigip\_static\_route:

destination: "{{ item.dst.split('/')[0] }}"

netmask: "{{item.dst | ipv4('prefix')}}"

gateway\_address: "{{ item.gw }}"

name: "{{ item.name }}"

provider: "{{ conn\_parameters }}"

loop: "{{ routes }}"

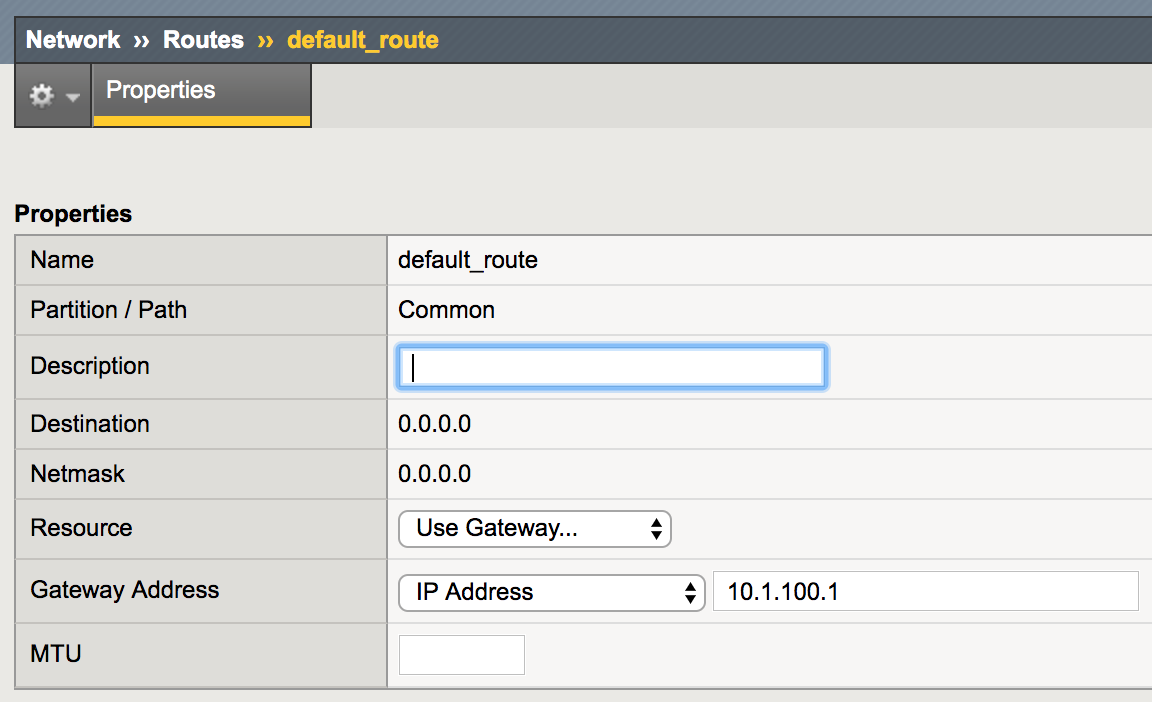
tags: routing

**How it works..**

We add the **routes** data structure in ltm01.yml under the host\_vars to declare all the static routes that needs to be provisioned on the LTM node.

We update the playbook pb\_f5\_onboard-.yml with a task to provision the static routes using the bigip\_static\_route module and we loop over the routes data structure to provision all the needed routes on the device.

After running the playbook again, we can see the correct static routes as shown below



**Deploying Nodes on BIG-IP Devices**

In order to start deploying Application on the BIG-IP LTM nodes in order to start load balancing the application traffic across multiple application instances, we need to define the instances that will be carrying the application in order to add them to the correct load balancing pool. In BIG-IP these instances are called nodes and it identify each server with a unique IP address. In this recipe we are going to start deploying a new Application (Web Server) on the BIG-IP and we will provision the nodes which are carrying this service using Ansible.

**Getting Ready**

Basic Setup for the BIG-IP should be already completed as per the previous recipes and the correct VLANs to reach these nodes (physical servers) must be deployed.

**How to do it..**

1. Create a new YAML file called web\_app.yml with the below content

---

vip: 10.1.100.10

vip\_port: 443

endpoint: dev.internet.net

pool\_name: dev\_web\_app

pool\_members:

- ip: 10.1.10.10

name: "dev01.internal.net"

port: 443

- ip: 10.1.10.11

name: "dev01.internal.net"

port: 443

1. Create a new Ansible playbook called **pb\_f5\_deploy\_app.yml** with the below contents

---

- name: Deploying a New App on BIG-IP

hosts: ltm01

connection: local

vars\_file: web\_app.yml

tasks:

- name: "Create Nodes on BIG-IP"

bigip\_node:

address: "{{ item.ip }}"

name: "{{ item.name }}"

provider: "{{ conn\_parameters }}"

state: present

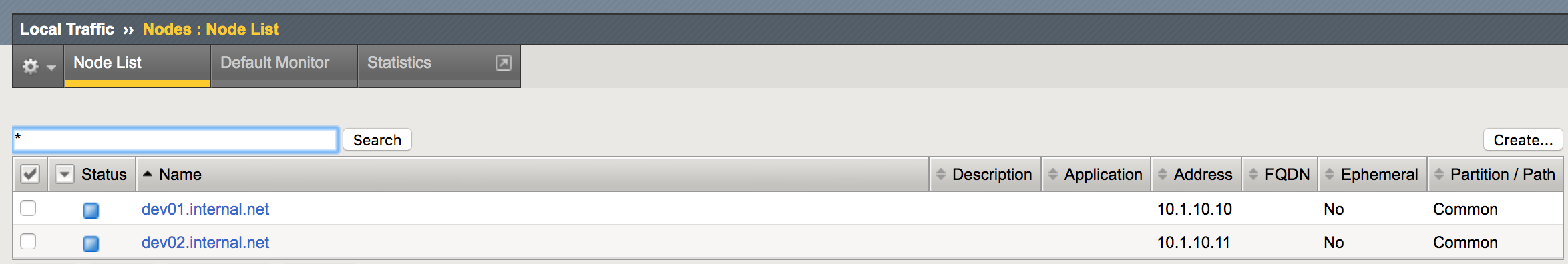
loop: "{{ pool\_members }}"

**How it works..**

We define all the parameters for our new Web application that should be hosted on the BIG-IP LTM device in a YAML file called **web\_app.yaml**. In this file we include a parameter which is the **pool\_members** to outline the web servers which will house the application. We use this parameter to create the nodes on the BIG-IP LTM.

We create a new playbook for application deployment called pb\_f5\_deploy\_app.yml and we include the **web\_app.yml** file so as to have access to all the parameters defined for this app. We create a new task using the **bigip\_node** module to provision a new node on the BIG-IP Node and we loop through the **pool\_members** parameter derived from the **web\_app.yml** file to provision all the required nodes on the BIG-IP node. In order to connect to the BIG-IP node we use the same previous provider attribute with the **conn\_parameters** parameter defined in the all.yml in order to establish the REST connection with the BIG-IP.

Running this playbook, we create the all the needed nodes as shown below



**Configuring LB Pools on BIG-IP Devices**

After creating a Node on the BIG-IP, we need to create a load balancing pool for the application that we are deploying and assign pool members from the nodes that we have created into this pool. In this recipe we will outline how to provision Load balancing pools on the BIG-IP nodes and how to assign members to the load balancing pool.

**Getting Ready**

This recipe assumes that all the previous recipes has been implemented and the nodes on the BIG-IP is already provisioned as per the previous receipe.

**How to do it..**

1. Update the playbook **pb\_f5\_deploy\_app.yml** with the below task to create a new pool

---

- name: Deploying a New App on BIG-IP

hosts: ltm01

connection: local

vars\_file: web\_app.yml

tasks:

< -- Output Omitted for brevity -->  
  
 - name: Create New LB Pool

bigip\_pool:

name: "POOL\_{{ website }}\_{{ vip\_port }}"

lb\_method: round-robin

state: present

provider: "{{ conn\_parameters }}"

1. Update the playbook **pb\_f5\_deploy\_app.yml** with the below task to assign pool members to the new created pool

---

- name: Deploying a New App on BIG-IP

hosts: ltm01

connection: local

vars\_file: web\_app.yml

tasks:

< -- Output Omitted for brevity -->

- name: Add Members to the Pool

bigip\_pool\_member:

pool: "POOL\_{{ website }}\_{{ vip\_port }}"

host: "{{ item.ip }}"

name: "{{ item.name }}"

port: "{{ item.port }}"

description: "Web Server for {{ website }}"

provider: "{{ conn\_parameters }}"

loop: "{{ pool\_members }}"

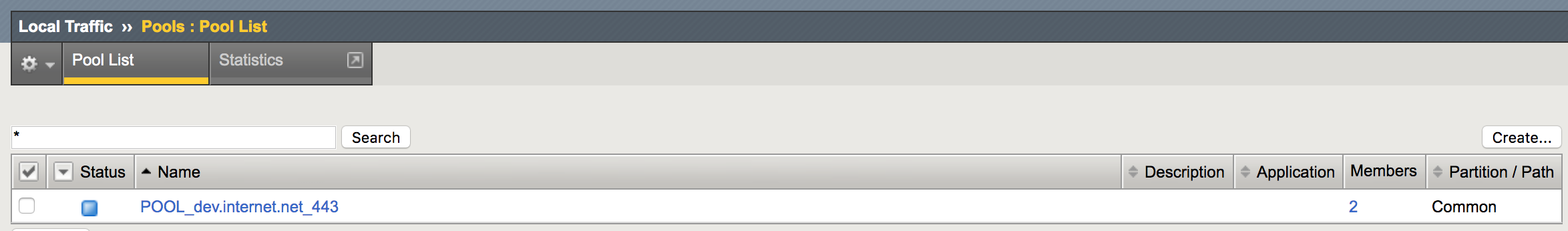
**How it works..**

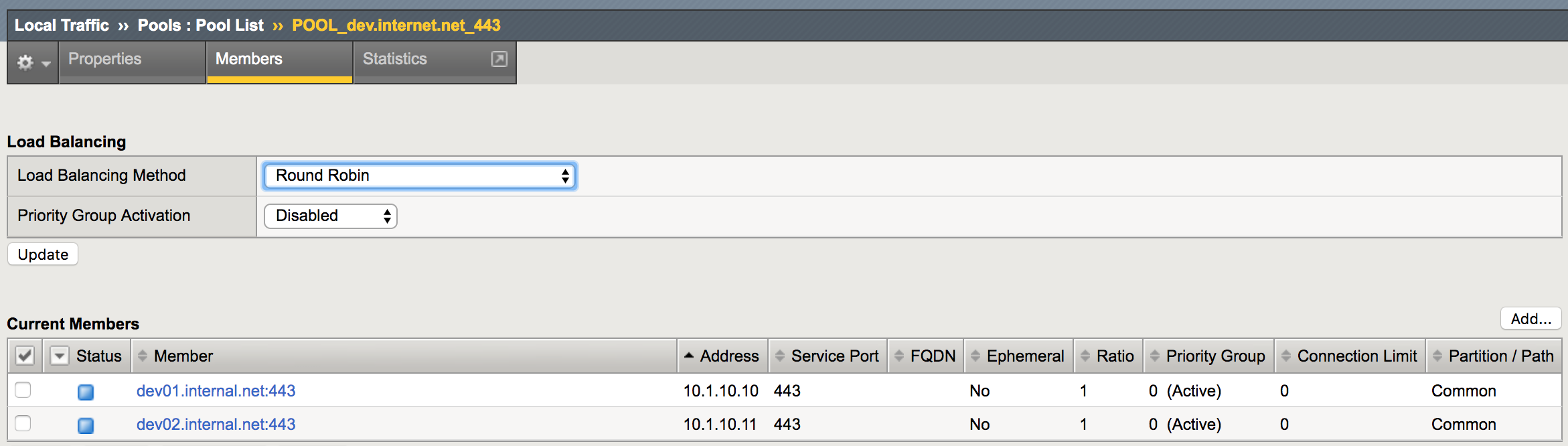
In this recipe we create a load balancing pool on the BIG-IP system using the **bigip\_pool** module and we specify the load balancing technique that should be used on this pool and in this example, we are using the round-robin technique. We create the pool name using the different parameters extracted from the **web\_app.yml** file (mainly the **website** and **vip\_port**) .

Next, we assign the pool members to this newly created pool using the **bigip\_pool\_member** module and looping through all the **pool\_memebers** defined again in the web\_app.yml.

We can see that all these procedures create a consistant method for defining the pool names as well as assigning the required pool members to the correct pool member. All the information is retrieved from a single definition file that describe and outline how the service should be deployed.

Running these two tasks we will see that the pool is correctly created as shown below with the correct pool members





**See Also..**

In this recipe we outlined the basic use for the ansible modules to provision LB pools on the BIG-IP nodes however there is more options available for these modules like the specifying the load balancing ratio for each member as well as attaching monitors for the overall pool. Please consult the below URLs for more options

Bigip\_pool  
<https://docs.ansible.com/ansible/latest/modules/bigip_pool_module.html#bigip-pool-module>

Bigip\_pool\_member  
<https://docs.ansible.com/ansible/latest/modules/bigip_pool_member_module.html#bigip-pool-member-module>

**Creating Virtual Servers on BIG-IP**

The last part in deploying an application on the BIG-IP LTM for load balancing is configuring the virtual server on the BIG-IP LTM node and creating a VIP on the BIG-IP node for this virtual server. In this recipe we outline how to deploy the virtual server using Ansible

**Getting Ready**

This recipe assumes that all the previous recipes are completed and a load balancing pool and pool members are already configured.

**How to do it..**

1. Update the playbook **pb\_f5\_deploy\_app.yml** with the below task

---

- name: Deploying a New App on BIG-IP

hosts: ltm01

connection: local

vars\_file: web\_app.yml

tasks:

< -- Output Omitted for brevity -->  
  
 - name: Create Virtual Server

bigip\_virtual\_server:

name: "{{ website }}\_{{ vip\_port }}\_VS"

destination: "{{ vip }}"

port: "{{ vip\_port}}"

pool: "POOL\_{{ website }}\_{{ vip\_port }}"

description: "VIP for {{ website }}"

profiles:

- http

- name: clientssl

context: client-side

- name: serverssl

context: server-side

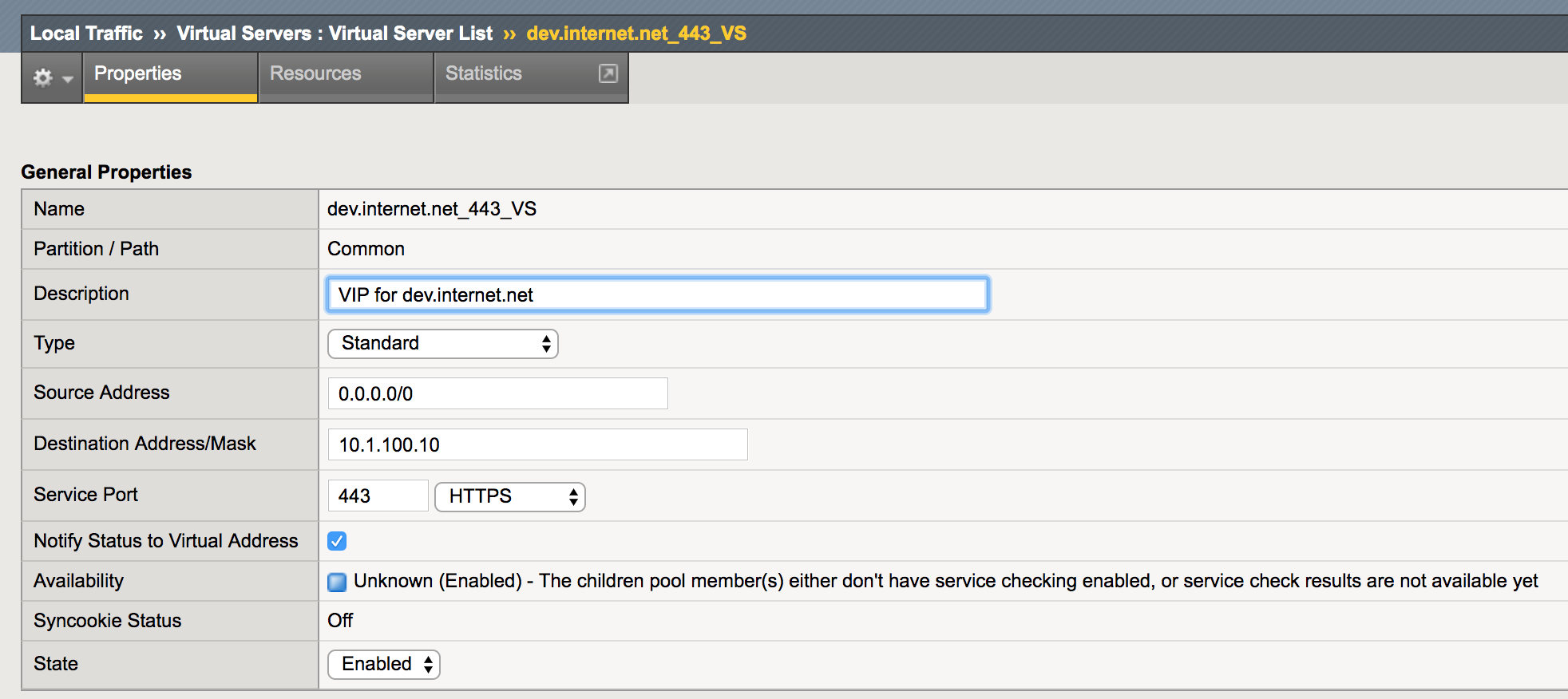
state: present

provider: "{{ conn\_parameters }}"

**How it works..**

We use the **bigip\_virtual\_server** module to provision the required virtual server on the BIG-IP node specifying the parameters defined from the **web\_app.yml** file. We also provision define the profiles that need to be applied to the new created virtual server and which like HTTP and SSL profiles. These profiles are already created by default on the BIG-IP node and in case if we need to create custom profiles we need to create it in a separate task using the appropriate ansible module.

Running this last task we can see that the Virtual Server is created as shown below



With this last task we have created a functional service VIP on the LTM node in order to start processing HTTP requested for our new website and to load balance the traffic across all the instances in the load balancing group.

**See Also..**

In this recipe we discussed the basic use for the ansible module to provision virtual servers on the BIG-IP nodes , however there is more options available in order to tweak the configuration for the virtual server that need to be deployed. Also there are more Ansible modules that let’s you create profiles that you can use to attach to the virtual server . below are some links for this modules from more information.

**Ansible bigip\_virtual\_server Module**  
<https://docs.ansible.com/ansible/latest/modules/bigip_virtual_server_module.html#bigip-virtual-server-module>

**Ansible bigip\_profile\_http Module**<https://docs.ansible.com/ansible/latest/modules/bigip_profile_http_module.html#bigip-profile-http-module>

**Ansible bigip\_profile\_client\_ssl Module**<https://docs.ansible.com/ansible/latest/modules/bigip_profile_client_ssl_module.html#bigip-profile-client-ssl-module>

**Ansible bigip\_profile\_server\_ssl Module**<https://docs.ansible.com/ansible/latest/modules/bigip_profile_server_ssl_module.html#bigip-profile-server-ssl-module>

**Retrieving Operational data from BIG-IP Nodes**

In this recipe we outline how to retrieve operational data for different components on the BIG-IP node in terms of the network state of the BIG-IP nodes like interfaces and vlans as well as relating to the components responsible for application delivery like virtual servers and pools.

**Getting Ready**

To follow along with this recipe, an ansible inventory is assumed to be already setup and IP connectivity between Ansible and the BIG-IP nodes is already established with the correct user credentials.

**How to do it..**

1. Create a new ansible playbook **pb\_f5\_validate.yml** with the below contents.

---

- name: Validating BIG-IP Health

hosts: ltm01

connection: local

tasks:

- name: Collect Device Facts from BIG-IP

bigip\_device\_facts:

gather\_subset:

- interfaces

provider: "{{ conn\_parameters }}"

register: bigip\_facts

1. Update the playbook **pb\_f5\_validate.yml** with a new task to validate interface status

- name: Validate All Interface are operational

assert:

that:

- item.enabled == 'yes'

fail\_msg: " Interface {{ item.name }} is Down"

loop: "{{bigip\_facts.interfaces}}"

**How it works..**

The REST API supported on the BIG-IP Node is also providing interface to retrieve operational data from the device and it output all this data in JSON format. The below snippet outline the Interface status gathered from the BIG-IP nodes using the **bigip\_device\_facts**.

"bigip\_facts": {  
< -- Output Omitted for brevity -->

"interfaces": [

{

"active\_media\_type": "10000T-FD",

"bundle": "not-supported",

"bundle\_speed": "not-supported",

"enabled": "yes",

"flow\_control": "tx-rx",

"full\_path": "1.1",

"if\_index": 48,

"lldp\_admin": "txonly",

"mac\_address": "00:50:00:00:01:01",

"media\_sfp": "auto",

"mtu": 1500,

"name": "1.1",  
< -- Output Omitted for brevity -->

}

We retrieve the operational facts from the BIG-IP nodes using **bigip\_device\_facts** and we restrict only the data retrieved from the node using **gather\_subset** and we only include the interfaces option to get only interface data. We save all the retrieved output to the variable bigip\_facts.

We use the assert module to validate that all the interface are enabled and operational from the retrieved data and we loop over the **bigip\_facts** variable (which is a list) to loop over all the interface and confirm that they are enabled.

**There is More..**

In case we need to get the operational data for the application that we have deployed on the LTM node, we create a new playbook to validate the application deployment as shown below using the bigip\_device\_facts and in this time we limit the data retrieved for only the virtual-servers and we validate the data using the assert statement as done in the previous playbook. Below is the playbook contents for application deployment.

---

- name: Validating BIG-IP Health

hosts: ltm01

connection: local

vars\_files: web\_app.yml

tasks:

- name: Collect Device Facts from BIG-IP

bigip\_device\_facts:

gather\_subset:

- virtual-servers

provider: "{{ conn\_parameters }}"

register: bigip\_app\_facts

- name: Validate Virtual Address Status

assert:

that:

- item.enabled == 'yes'

- item.destination\_address == vip

- item.destination\_port == vip\_port

fail\_msg: " {{ item.name }} is No Setup Correctly"

loop: "{{bigip\_app\_facts.virtual\_servers}}"

For sure we can enhance the validation to validate on multiple items on the virtual-address data as well as validating multiple different facts like ltm-pools to build a more comprehensive validation for the application deployed.

**See Also..**

For more information regarding the Ansible **bigip\_device\_facts** module and all the information that we can retrieve from the BIG-IP node please check the below URL

<https://docs.ansible.com/ansible/latest/modules/bigip_device_facts_module.html#bigip-device-facts-module>