Software Defined Networking in Azure Stack HCI training: Lab for Module 5: Software Load Balancer

Microsoft Corporation
Published: June 04, 2024

Applies to

Software Defined Networking (SDN) training: Module 5: Software Load Balancer

Copyright

Revision History

Release Date	Changes
June 04, 2024	Initial release.

Contents

SDN LAB: M5 SOFTWARE LOAD BALANCER

Overview

Software Load Balancer (SLB) is a NAT-based, layer-4 load balancer that enables load balancing for north-south and east-west TCP/UDP traffic. SLB allows load balancing of public and internal network traffic, and health probes monitor the state of resources. SLB is managed via SLB Manager Service within Network Controller.

M5.2 SLB architecture

Lab 1: Examine SLB Manager service

1. Get the service health state by running:

```
Get-SdnServiceFabricService -ServiceTypeName 'SlbManagerService'
```

```
PS C:\Users\Administrator> Get-SdnServiceFabricService -ServiceTypeName 'SlbManagerService'


HasPersistedState      : True
ServiceKind            : Stateful
ServiceName            : fabric:/NetworkController/SlbManagerService
ServiceTypeName        : SlbManagerService
ServiceManifestVersion : 12.0.10
HealthState            : Ok
ServiceStatus          : Active
IsServiceGroup         : False
```

2. Get the process related to the SlbManagerService

```
Get-Process -Name SDNSLBM
```

```
PS C:\Users\Administrator> Get-Process -Name SDNSLBM

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   2387      92   552192     417824      77.42   4204   0 SDNSLBM
```

3. Identify the primary replica for SDN SLB Manager service

```
Get-SdnServiceFabricReplica -ServiceTypeName 'SlbManagerService' -Primary
```

```
PS C:\Users\Administrator> Get-SdnServiceFabricReplica -ServiceTypeName 'SlbManagerService' -Primary

ReplicaId            : 132333625491286378
ReplicaOrInstanceId  : 132333625491286378
PartitionId          : 9f6d2c9d-02e9-40c0-9388-e79aab483a98
ReplicaRole          : Primary
ServiceKind          : Stateful
Id                   : 132333625491286378
ReplicaStatus        : Ready
HealthState          : Ok
ReplicaAddress       : SDN-NC01.SDN.LAB:0
NodeName             : SDN-NC01
LastInBuildDuration  : 00:00:01
```

Lab 2: Examine load balancer MUXs

SLB MUXs when added into the SDN Fabric are represented under /networking/<api version>/loadBalancerMuxes within the NC NB API. Network Controller then leverages this information to program policies via its SB API to ensure that the appropriate VIP:DIP mappings are populated within the dataplane for the MUXs. For MUXs, the SDN SLB Manager service connects via TCP 8560 to the MUXs, and policies are programmed via WCF using x509 authentication.

1. Get the Load Balancer MUX nodes within the environment:

   ```
   Get-SdnLoadBalancerMux -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl
   ```

2. Connect into one of the Load Balancer MUX VMs and get the status of the SlbMux service and MuxSvcHost process. The SlbMux service is expected to be running.

   ```
   Get-Service -Name SlbMux
   Get-Process -Name MuxSvcHost
   ```

   ```
   PS C:\Users\Administrator> Get-Service -Name SlbMux

   Status   Name               DisplayName
   ------   ----               -----------
   Running  SlbMux             Software Load Balancer Multiplexer


   PS C:\Users\Administrator> Get-Process -Name MuxSvcHost

   Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
   -------  ------    -----      -----     ------     --  -- -----------
       664      31    50368      65972      11.77   2632   0 MuxSvcHost
   ```

3. Identify the TCP connection between MuxSvcHost and the SDN SLB Manager primary service replica.

   ```
   Get-NetTCPConnection -OwningProcess 2632
   ```

```
PS C:\Users\Administrator> Get-NetTCPConnection -OwningProcess 2632

LocalAddress                        LocalPort RemoteAddress                       RemotePort State       AppliedSetting
------------                        --------- -------------                       ---------- -----       --------------
10.10.56.7                          49753     0.0.0.0                             0          Bound
10.10.56.7                          49753     10.10.56.1                          179        Established Internet
10.184.108.18                       8560      10.184.108.14                       50101      Established Datacenter
0.0.0.0                             8560      0.0.0.0                             0          Listen
```

Lab 3: Examine SLBHostAgent

SLBHostAgent is responsible for connecting over TLS to Network Controller to the SDN SLB Manager Service to receive policy updates from SLB Manager. Without this TCP connection, SLBHostAgent is unable to program the VFP policies related to NAT. SLBHostAgent connects to SDN API on port 8570 on startup.

SDN API redirects to the SDN SLB Manager Service primary replica endpoint. SLBHostAgent then tears down the TCP connection to 8570 and creates a new TCP connection to the IP address of NC node via TCP 8571 that is the primary replica for SDN SLB Manager service.

1. Get the Server nodes within the environment:

   ```
   Get-SdnServer -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl
   ```

2. Connect into SDN-HOST01 or SDN-HOST02 and get the status of the SlbHostAgent service and SLBHostPluginService. The SLBHostAgent service is expected to be running. If the service is not running, you will not have the SLBHostPluginService process available.

   ```
   Get-Service -Name SlbHostAgent
   Get-Process -Name SlbHostPluginService
   ```

   ```
   PS C:\Users\administrator.SDN> Get-Service -Name SlbHostAgent

   Status   Name               DisplayName
   ------   ----               -----------
   Running  SlbHostAgent       Software Load Balancer Host Agent


   PS C:\Users\administrator.SDN> Get-Process -Name SlbHostPluginService

   Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
   -------  ------    -----      -----     ------     --  -- -----------
       352      16     8644      18136       0.47   3048   0 SlbHostPluginService
   ```

3. Identify the TCP connection between SlbHostPluginService and SDN SLB Manager primary service replica.

   ```
   Get-NetTCPConnection -OwningProcess 3048
   ```

```
PS C:\Users\administrator.SDN> Get-NetTCPConnection -OwningProcess 3048

LocalAddress                        LocalPort RemoteAddress                        RemotePort State       AppliedSetting OwningProcess
------------                        --------- -------------                        ---------- -----       -------------- -------------
0.0.0.0                             50876     0.0.0.0                              0          Bound                      3048
10.184.108.2                        50876     10.184.108.14                        8571       Established Datacenter     3048
10.184.108.2                        18181     0.0.0.0                              0          Listen                     3048
```

Lab 4: Examine the Load Balancer Manager config

The Load Balancer Manager configuration, which can be exposed via NB API as /networking/<api version>/loadBalancerManager/config contains important configuration related to the Load Balancer Manager IP, Outbound NAT IP exceptions and VIP IP Pools.

1. On any of the nodes within the environment, get the Load Balancer Manager configuration and assign to a variable.

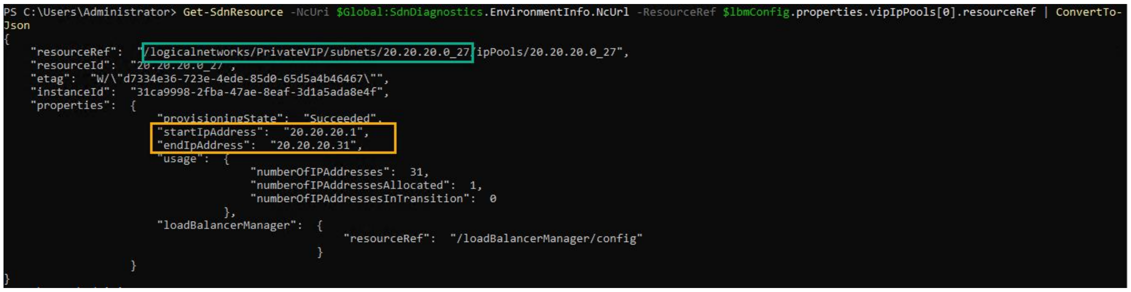
```
$lbmConfig = Get-SdnResource -NcUri
$Global:SdnDiagnostics.EnvironmentInfo.NcUrl -Resource
LoadBalancerManagerConfig
$lbmConfig
```

```
PS C:\Users\Administrator> $lbmConfig = Get-SdnResource -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl -ResourceType LoadBalancerManagerConfig
PS C:\Users\Administrator> $lbmConfig |  ConvertTo-Json
{
    "resourceRef":  "/loadBalancerManager/config",
    "resourceId":  "config",
    "etag":  "W/\"94e9f78f-6e4f-455d-954d-6b51c6835eb5\"",
    "instanceId":  "eab5d336-3bbe-49fc-8d45-1b7618e3ac5f",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "loadBalancerManagerIPAddress":  "20.20.20.1",
                       "outboundNatIPExemptions":  [
                                                       "20.20.20.1/32"
                                                   ],
                       "vipIpPools":  [
                                          "@{resourceRef=/logicalnetworks/PrivateVIP/subnets/20.20.20.0_27/ipPools/20.20.20.0_27}",
                                          "@{resourceRef=/logicalnetworks/PublicVIP/subnets/41.40.40.0_27/ipPools/41.40.40.0_27}"
                                      ]
                   }
}
```

2. The loadBalancerManagerIPAddress is used by the MUX to encap incoming packets to a VIP and route the packet to the appropriate PA IP where the CA IP resides. When following the packet after it has been processed by the MUX, you see that the outer IP header source address matches this IP address.

3. Within the vipIpPools, this can be a combination of private and public VIPs that are used by the SDN environment. When creating a dynamic or static IP allocation, the VIP address is derived from within these VIP pools. Typically, private VIPs are used for scenarios such as internal load balancing. Public VIPs are advertised via the MUXs to the TORs and are routable from external locations. Examine the IP Pools:

```
Get-SdnResource -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl -
ResourceRef $lbmConfig.properties.vipIpPools[0].resourceRef | ConvertTo-Json
```



4. This shows the IP addresses within the IP pool, in addition to other information such as usage. To determine more information regarding the subnet the IP pool is associated with, you need to move up a resource layer. The resourceRef we want to examine is highlighted above in green.

```
Get-SdnResource -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl -
ResourceRef '/logicalnetworks/PrivateVIP/subnets/20.20.20.0_27' | ConvertTo-
Json
```

Examine the information that is returned and make note of some properties such as isPublic to determine the intended usage of the subnet.

M5.4 Load balancer

Lab 1: Create public IP address

1. Navigate to WAC > SDN Fabric Cluster > Public IP addresses (under Extensions)
2. Select Inventory > +New.
    a. Name: VIP02.
    b. IP Address Version: IPV4.
    c. IP Address Allocation Method: Dynamic.
    d. Idle Timeout In Minutes: 4.

Lab 2: Create load balancer with health probes

1. Navigate to Windows Admin Center > SDN Fabric > Load Balancers.
2. Create a Load Balancer object by selecting +NEW.
    a. Name: SLB01.
    b. Type: Public IP.
    c. Public IP Address: VIP02.
3. Click on SLB01.
4. Create a backend pool for Load Balancer by navigating to Backend Pools > +NEW.
    a. Name: BackendPool1.
    b. Associated IP Configurations:
        i. Network Interface: Contoso-VM2_Net_Adapter_#.

    ii. Target Network IP Configuration: Contoso-VM2_Net_Adapter_# - (Private IP).

5. Create a health probe for Load Balancer by navigating to Health Probes > +NEW.
    a. Name: HealthProbe1.
    b. Protocol: TCP.
    c. Port: 80.
    d. Interval: 5.
    e. Unhealthy: 3.
6. Create load balancing rule by navigating to Load Balancing Rules > +NEW.
    a. Name: LoadBalancingRule1.
    b. Frontend IP Configuration: frontip-SLB01 (PublicIP).
    c. Protocol: TCP.
    d. Frontend Port: 80.
    e. Backend Port: 80.
    f. Backend Pool: BackendPool1.
    g. Health Probe: HealthProbe1.
    h. Session Persistence: Default.
    i. Idle Timeout: 4.
    j. Floating IP (direct server return): OFF.
7. Connect into Contoso-VM2 and enable TCP Listener and open appropriate firewalls.

```
# ensure ports allowed in firewall
netsh advfirewall firewall add rule name="TCP Port 80" dir=in action=allow
protocol=TCP localport=80
netsh advfirewall firewall add rule name="TCP Port 80" dir=out action=allow
protocol=TCP localport=80

# install IIS web role
Install-WindowsFeature -Name 'Web-Server' -IncludeManagementTools
```

8. If you have a NSG or ACL attached to Contoso-VM2 NIC, you will also need to add appropriate rules to allow TCP_80 traffic inbound.

Name	Priority	Types	Protocol	Source Address Prefix	Source Port Range	Destination Address Prefix	Destination Port Range	Actions	Logging
TCP_80	101	Inbound	TCP	*	*	*	80	Allow	Enabled

Lab 3: Examine routes on TOR

In normal scenarios, the customer has a Top of Rack (TOR) switch that the servers are physically cabled into. Work with the customer and appropriate hardware vendors to perform switch management tasks. In this lab scenario, we have the BGP Router role installed on DC01 to replicate a TOR switch in this environment.

1. Connect to SDN-DC01 and open PowerShell.
2. Run Get-BgpPeer to see a list of BGP Peers that have established a BGP session with DC01.

```
PS C:\Windows\system32> Get-BgpPeer

PeerName LocalIPAddress PeerIPAddress PeerASN OperationMode ConnectivityStatus
-------- -------------- ------------- ------- ------------- ------------------
SDN-GW01 10.10.56.1     10.10.56.254  64628   Mixed         Connecting
SDN-GW02 10.10.56.1     10.10.56.253  64628   Mixed         Connected
SDN-MUX01 10.10.56.1    10.10.56.17   64628   Mixed         Connected
SDN-MUX02 10.10.56.1    10.10.56.18   64628   Mixed         Connected
```

3. Examine the routes advertised by the MUXs using Get-BgpRouteInformation.

```
PS C:\Windows\system32> Get-BgpRouteInformation

DestinationNetwork NextHop     LearnedFromPeer State LocalPref MED
------------------ -------     --------------- ----- --------- ---
41.40.40.2/32      10.10.56.17 SDN-MUX01       Best
41.40.40.2/32      10.10.56.18 SDN-MUX02       Best
41.40.40.3/32      10.10.56.17 SDN-MUX01       Best
41.40.40.3/32      10.10.56.18 SDN-MUX02       Best
```

4. Examine the route table.

```
Get-NetRoute -NextHop 10.10.56.17
Get-NetRoute -NextHop 10.10.56.18
```

```
PS C:\Windows\system32> Get-NetRoute -NextHop 10.10.56.17
Get-NetRoute -NextHop 10.10.56.18

ifIndex DestinationPrefix          NextHop          RouteMetric ifMetric PolicyStore
------- -----------------          -------          ----------- -------- -----------
17      41.40.40.3/32              10.10.56.17                0 15       ActiveStore
17      41.40.40.2/32              10.10.56.17                0 15       ActiveStore
17      41.40.40.3/32              10.10.56.18                0 15       ActiveStore
17      41.40.40.2/32              10.10.56.18                0 15       ActiveStore
```

   a. Examine the RouteMetric. Both MUXs are advertising the VIP/32 address to the TOR, with equal RouteMetric for each, enabling Equal Cost Multi-Path (ECMP) Routing.

M5.5 L3 forwarding

Lab 1: Create public IP address and enable TNC to public internet

1. Navigate to WAC > SDN Fabric Cluster > Public IP addresses (under Extensions).

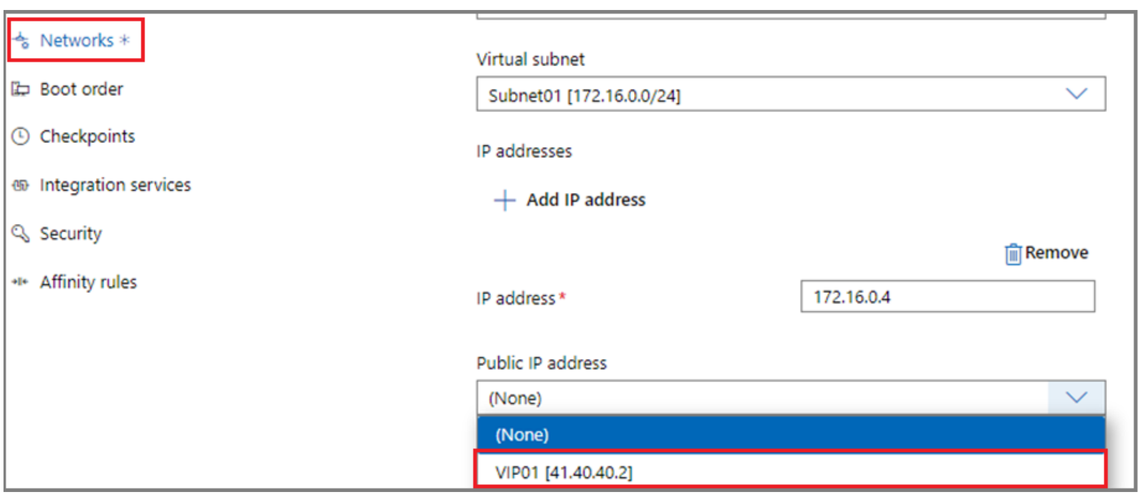
2. Select **Inventory** > **+New.**
    a. Name: VIP01.
    b. IP Address Version: IPV4.
    c. IP Address Allocation Method: Dynamic.
    d. Idle Timeout In Minutes: 4.
3. RDP into Contoso-VM1 and enable Test-NetConnection to public endpoint. What do you currently observe for pattern?

```
while($true){Test-NetConnection -ComputerName login.microsoftonline.com -Port
443}
```

Lab 2: Create L3 forwarding for NIC

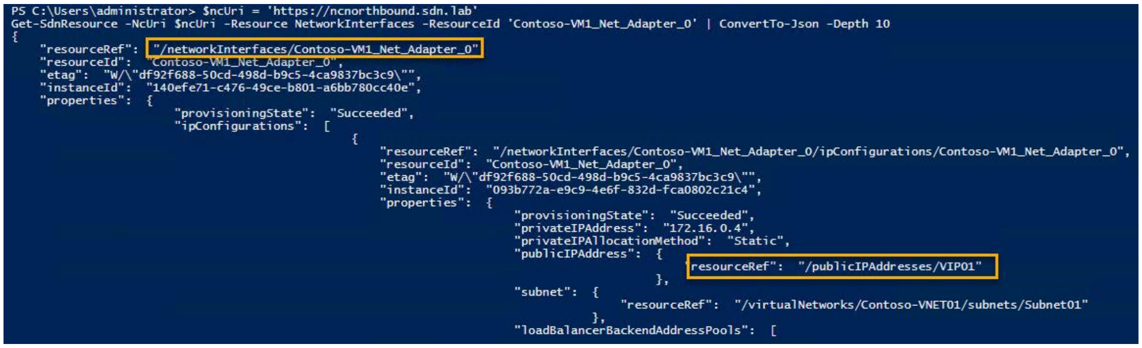
In this lab, we associate a public IP address directly to the network interface. This is referred to as an instance-level public IP address that enables inbound/outbound access to external resources. This also by default enables access for all the ports as well, such as commonly used ports like RDP (3389) and SSH (22). To restrict default access to well-known ports, implement NSGs to block the traffic.

1. Stop Contoso-VM1.
2. Navigate to **Contoso-VM1 Settings** > **Networks.**
3. Under **Public IP address**, select **VIP01** from the drop-down menu.



4. Select **Save network settings**.
5. Start the VM.
6. Now that the public IP is associated with the VM network interface, we can examine this via the Rest API directly.

```
$ncUri = 'https://ncnorthbound.sdn.lab'
Get-SdnResource -NcUri $ncUri -Resource NetworkInterfaces -ResourceId
'Contoso-VM1_Net_Adapter_0' | ConvertTo-Json -Depth 10
```



```
PS C:\Users\administrator> $ncUri = 'https://ncnorthbound.sdn.lab'
Get-SdnResource -NcUri $ncUri -Resource NetworkInterfaces -ResourceId 'Contoso-VM1_Net_Adapter_0' | ConvertTo-Json -Depth 10
{
    "resourceRef":  "/networkInterfaces/Contoso-VM1_Net_Adapter_0",
    "resourceId":  "Contoso-VM1_Net_Adapter_0",
    "etag":  "W/\"df92f688-50cd-498d-b9c5-4ca9837bc3c9\"",
    "instanceId":  "140efe71-c476-49ce-b801-a6bb780cc40e",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "ipConfigurations":  [
                                                {
                                                    "resourceRef":  "/networkInterfaces/Contoso-VM1_Net_Adapter_0/ipConfigurations/Contoso-VM1_Net_Adapter_0",
                                                    "resourceId":  "Contoso-VM1_Net_Adapter_0",
                                                    "etag":  "W/\"df92f688-50cd-498d-b9c5-4ca9837bc3c9\"",
                                                    "instanceId":  "093b772a-e9c9-4e6f-832d-fca0802c21c4",
                                                    "properties":  {
                                                                       "provisioningState":  "Succeeded",
                                                                       "privateIPAddress":  "172.16.0.4",
                                                                       "privateIPAllocationMethod":  "Static",
                                                                       "publicIPAddress":  {
                                                                                               "resourceRef":  "/publicIPAddresses/VIP01"
                                                                                           },
                                                                       "subnet":  {
                                                                                      "resourceRef":  "/virtualNetworks/Contoso-VNET01/subnets/Subnet01"
                                                                                  },
                                                                       "loadBalancerBackendAddressPools":  [
```

7. RDP back into Contoso-VM1 and check the Test-NetConnection that you configured previously in Lab 1: Create Public IP Address and Enable TNC to Public Internet.
    a. What do you observe for a pattern now?

M5.7 Component tracing

Lab 1: Capture network trace for outbound L3 traffic

In this scenario, focus on capturing traces for outbound datapath that the packet will take. As you recall from the training, Direct Server Return (DSR) is utilized and the packet when leaving the VMSwitch where the tenant VM resides SNATs the packet and routes the packet directly to the TOR switches that the hosts are physically connected to. There is no need to analyze logs or data related to the MUXs in this datapath.

1. Connect into Contoso-VM1 and start a Test-NetConnection to login.microsoft.online.com.

```
while ($true) {Test-NetConnection login.microsoft.online.com -Port
443}
```

2. Enable tracing on the host where Contoso-VM1 resides using Start-SdnNetshTrace -Role Server.
3. Disable tracing after 15-20 seconds using Stop-SdnNetshTrace.
4. Copy the trace file over to lab host, and open with Network Monitor and apply the filter description.contains(“externalIpAddress”).
5. Select one of the packets to remove filtering (don’t click anywhere as should jump you to the frame you have selected).
6. Examine the output captured. You should see the following pattern with SNAT rule being applied and re-writing the src IP address from private IP to public VIP.
    a. VMSwitch – NBL received from NIC {GUID-GUID} into SDNSwitch.
    b. VFPEXT – LAYER: FW_ADMIN_LAYER.
    c. VPTEXT – LAYER: FW_CONTROLLER_LAYER.
    d. VFPEXT – LAYER: VNET_METER_LAYER.

e. VFPEXT – LAYER: VNET_DR_REDIRECTION_LAYER (no match).

f. VFPEXT – LAYER: VNET_MAC_REWRITE_LAYER (no match).

g. VFPEXT – LAYER: VNET_ENCAP_LAYER (VNET_MAC_REWRITE).

h. VFPEXT – LAYER: SLB_NAT_LAYER (SNAT).

> 20230 11:48:29 PM 11/29/2022 24.8884510 (0)
> VFPEXT_MicrosoftWindowsHyperVVfpExt
> VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) matched outboundpackets with flow id {src ip = 172.16.0.4, dst ip = 8.8.8.8, protocol = 1 (0x1), icmp type = V4EchoRequest} to flow {layer = SLB_NAT_LAYER, flow type = Snat}

i. VFPEXT – LAYER: SLB_DECAP_LAYER_STATEFUL.

> 20231 11:48:29 PM 11/29/2022 24.8884527 (0)
> VFPEXT_MicrosoftWindowsHyperVVfpExt
> VFPEXT_MicrosoftWindowsHyperVVfpExt:Allowrule with ID N/A processed outboundpackets on port 11 (0xB) with status = Success: flow id {src ip = 41.40.40.2, dst ip = 8.8.8.8, protocol = 1 (0x1), icmp type = V4EchoRequest}, rule {layer = SLB_DECAP_LAYER_STATEFUL,

j. VFPEXT – LAYER: VNET_PA_ROUTE_LAYER (VNET_PA_ROUTE).

k. VFPEXT – Successfully forwarded packet.

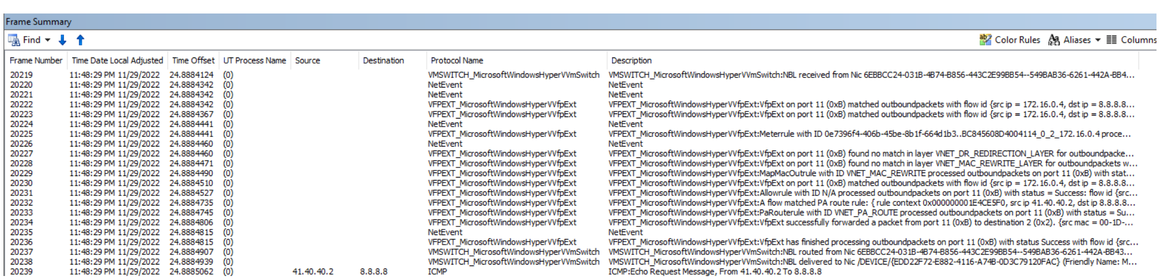
l. VFPEXT – Finished processing outbound packets on port.

m. VMSwitch – NBL routed from NIC.

n. VMSwitch – NBL delivered to NIC.

o. NDIS Packet Capture.

> 20239 11:48:29 PM 11/29/2022 24.8885062 (0) 41.40.40.2 8.8.8.8 ICMP
> ICMP:Echo Request Message, From 41.40.40.2 To 8.8.8.8



Frame Number	Time Date Local Adjusted	Time Offset	UT Process Name	Source	Destination	Protocol Name	Description
20219	11:48:29 PM 11/29/2022	24.8884124	(0)			VMSWITCH_MicrosoftWindowsHyperVVmSwitch	VMSWITCH_MicrosoftWindowsHyperVVmSwitch:NBL received from Nic 6EBBCC24-031B-4B74-B856-443C2E99BB54--549BAB36-6261-442A-BB4...
20220	11:48:29 PM 11/29/2022	24.8884342	(0)			NetEvent	NetEvent
20221	11:48:29 PM 11/29/2022	24.8884342	(0)			NetEvent	NetEvent
20222	11:48:29 PM 11/29/2022	24.8884342	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) matched outboundpackets with flow id {src ip = 172.16.0.4, dst ip = 8.8.8.8...
20223	11:48:29 PM 11/29/2022	24.8884367	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) matched outboundpackets with flow id {src ip = 172.16.0.4, dst ip = 8.8.8.8...
20224	11:48:29 PM 11/29/2022	24.8884441	(0)			NetEvent	NetEvent
20225	11:48:29 PM 11/29/2022	24.8884441	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:Meterrule with ID 0e7396f4-406b-45be-8b1f-664d1b3..BC845608D4004114_0_2_172.16.0.4 proce...
20226	11:48:29 PM 11/29/2022	24.8884460	(0)			NetEvent	NetEvent
20227	11:48:29 PM 11/29/2022	24.8884460	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) found no match in layer VNET_DR_REDIRECTION_LAYER for outboundpacke...
20228	11:48:29 PM 11/29/2022	24.8884471	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) found no match in layer VNET_MAC_REWRITE_LAYER for outboundpackets w...
20229	11:48:29 PM 11/29/2022	24.8884490	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:MapMacOutrule with ID VNET_MAC_REWRITE processed outboundpackets on port 11 (0xB) with stat...
20230	11:48:29 PM 11/29/2022	24.8884510	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt on port 11 (0xB) matched outboundpackets with flow id {src ip = 172.16.0.4, dst ip = 8.8.8.8...
20231	11:48:29 PM 11/29/2022	24.8884527	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:Allowrule with ID N/A processed outboundpackets on port 11 (0xB) with status = Success: flow id {src...
20232	11:48:29 PM 11/29/2022	24.8884735	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:A flow matched PA route rule: { rule context 0x000000001E4CE5F0, src ip 41.40.40.2, dst ip 8.8.8.8...
20233	11:48:29 PM 11/29/2022	24.8884745	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:PaRouterule with ID VNET_PA_ROUTE processed outboundpackets on port 11 (0xB) with status = Su...
20234	11:48:29 PM 11/29/2022	24.8884806	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt successfully forwarded a packet from port 11 (0xB) to destination 2 (0x2). {src mac = 00-1D-...
20235	11:48:29 PM 11/29/2022	24.8884815	(0)			NetEvent	NetEvent
20236	11:48:29 PM 11/29/2022	24.8884815	(0)			VFPEXT_MicrosoftWindowsHyperVVfpExt	VFPEXT_MicrosoftWindowsHyperVVfpExt:VfpExt has finished processing outboundpackets on port 11 (0xB) with status Success with flow id {src...
20237	11:48:29 PM 11/29/2022	24.8884907	(0)			VMSWITCH_MicrosoftWindowsHyperVVmSwitch	VMSWITCH_MicrosoftWindowsHyperVVmSwitch:NBL routed from Nic 6EBBCC24-031B-4B74-B856-443C2E99BB54--549BAB36-6261-442A-BB43...
20238	11:48:29 PM 11/29/2022	24.8884939	(0)			VMSWITCH_MicrosoftWindowsHyperVVmSwitch	VMSWITCH_MicrosoftWindowsHyperVVmSwitch:NBL delivered to Nic /DEVICE/{EDD22F72-E882-4116-A74B-0D3C79120FAC} (Friendly Name: M...
20239	11:48:29 PM 11/29/2022	24.8885062	(0)	41.40.40.2	8.8.8.8	ICMP	ICMP:Echo Request Message, From 41.40.40.2 To 8.8.8.8

Lab 2: Capture network trace for inbound balancer traffic

In scenarios that capture both inbound and outbound datapath for packet flow, you need to also enable tracing on the MUXs. It's also beneficial to enable the tracing on the Servers where the MUXs reside, as everything must flow in/out of the host vmSwitch. In this example, we capture traces when trying to perform a Test-NetConnection from Contoso-VM1 to Contoso-VM2 over the load balancer IP:Port.

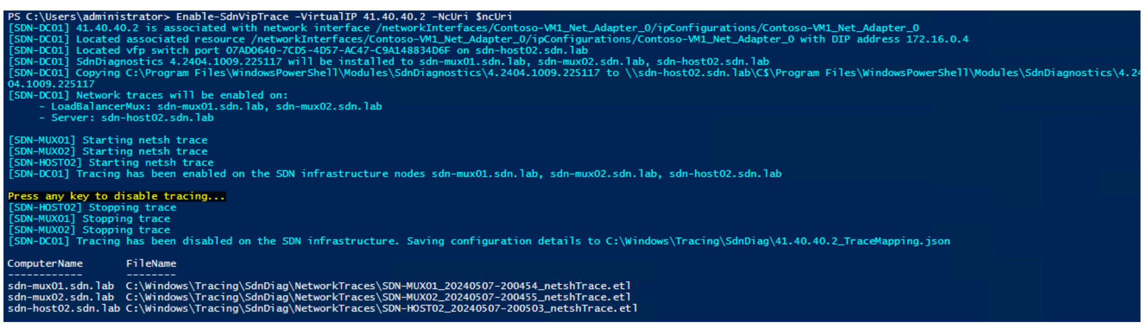
1. Connect to Contoso-VM1 and start a Test-NetConnection loop to Contoso-VM2. This will repeat both the TCP and ICMP requests. When troubleshooting connectivity, always leverage TCP protocols when traversing the SLB layers, as ICMP can be unreliable due to NSGs or OS firewall rules.

```
while ($true) {Test-NetConnection -ComputerName 41.40.40.3 -Port 80 -
InformationLevel Detailed}
```

2. Once the script is running, Connect to DC01 where we enable the tracing and data collection from. The **Enable-SdnVipTrace** function automates isolating the appropriate node(s) to enable tracing on making it simpler in larger scale deployments to enable tracing on the correct nodes.

```
$ncUri = 'https://ncnorthbound.sdn.lab'
Enable-SdnVipTrace -VirtualIP 41.40.40.2 -NcUri $ncUri

# wait for ~ 15-20 seconds before stopping the traces
```



```
PS C:\Users\administrator> Enable-SdnVipTrace -VirtualIP 41.40.40.2 -NcUri $ncUri
[SDN-DC01] 41.40.40.2 is associated with network interface /networkInterfaces/Contoso-VM1_Net_Adapter_0/ipConfigurations/Contoso-VM1_Net_Adapter_0
[SDN-DC01] Located associated resource /networkInterfaces/Contoso-VM1_Net_Adapter_0/ipConfigurations/Contoso-VM1_Net_Adapter_0 with DIP address 172.16.0.4
[SDN-DC01] Located vfp switch port 07AD0640-7CD5-4D57-AC47-C9A148834D6F on sdn-host02.sdn.lab
[SDN-DC01] SdnDiagnostics 4.2404.1009.225117 will be installed to sdn-mux01.sdn.lab, sdn-mux02.sdn.lab, sdn-host02.sdn.lab
[SDN-DC01] Copying C:\Program Files\WindowsPowerShell\Modules\SdnDiagnostics\4.2404.1009.225117 to \\sdn-host02.sdn.lab\C$\Program Files\WindowsPowerShell\Modules\SdnDiagnostics\4.24
04.1009.225117
[SDN-DC01] Network traces will be enabled on:
     - LoadBalancerMux: sdn-mux01.sdn.lab, sdn-mux02.sdn.lab
     - Server: sdn-host02.sdn.lab

[SDN-MUX01] Starting netsh trace
[SDN-MUX02] Starting netsh trace
[SDN-HOST02] Starting netsh trace
[SDN-DC01] Tracing has been enabled on the SDN infrastructure nodes sdn-mux01.sdn.lab, sdn-mux02.sdn.lab, sdn-host02.sdn.lab

Press any key to disable tracing...
[SDN-HOST02] Stopping trace
[SDN-MUX01] Stopping trace
[SDN-MUX02] Stopping trace
[SDN-DC01] Tracing has been disabled on the SDN infrastructure. Saving configuration details to C:\Windows\Tracing\SdnDiag\41.40.40.2_TraceMapping.json

ComputerName       FileName
------------       --------
sdn-mux01.sdn.lab  C:\Windows\Tracing\SdnDiag\NetworkTraces\SDN-MUX01_20240507-200454_netshTrace.etl
sdn-mux02.sdn.lab  C:\Windows\Tracing\SdnDiag\NetworkTraces\SDN-MUX02_20240507-200455_netshTrace.etl
sdn-host02.sdn.lab C:\Windows\Tracing\SdnDiag\NetworkTraces\SDN-HOST02_20240507-200503_netshTrace.etl
```

3. Initiate **Start-SdnDataCollection** to pick up the required data points in addition to the network traces that were just captured.

```
Start-SdnDataCollection -NetworkController 'SDN-NC01' -Role
Server,SoftwareLoadBalancer
```

4. Once data collection is completed, move the folder to lab host and open with Visual Studio Code and examine data:
   - **.\SlbState.JSON**: Contains an aggregated state of all the public and private VIPs in the environment. Search for the VIP under array {datagroups}.{tenant}.{VipConsolidatedState}
   - **.\{MUX##}\NetworkTraces**: Contains the network traces captured from the MUXs. There should be a minimum of one .etl file and one converted trace in .txt format.

5. Examine the netsh.txt trace from the MUXs and search for the VIP to determine which MUX is performing the packet processing for that flow. In scenarios where packet is routed, should see a pattern that resembles:

```
[2]0000.0000::2022/11/30-22:38:59.121488700 [Microsoft-Windows-SlbMuxDriver]SlbMux processing {src ip = 41.40.40.2, dst ip = 41.40.40.3} {protocol = 6, src port = 50031, dst port = 80} NetBufferList = 0xFFFFC10D51348510, ParentNetBufferList = 0x0.

[2]0000.0000::2022/11/30-22:38:59.121503400 [Microsoft-Windows-SlbMuxDriver]SlbMux redirect packet {Outer: src ip = 20.20.20.1, dst ip = 41.40.40.2} {NVGRE: protocol = 0x6558, key = 0xFF00} {Inner: src mac = 00-15-5D-8B-04-05, dst mac = 00-00-00-00-00-00} {Inner: type = 0x800, src ip = 20.20.20.1, dst ip = 41.40.40.2} {ICMP: type = 5, ip address = 10.10.56.10, {protocol = 6, src ip = 41.40.40.2, dst ip = 41.40.40.3} src port = 50031, dst port = 80, sequence number = 0 {AddressFamily = 2, EncapType = 2, Vsid = 4132, DIP PA = 10.10.56.10, VM Mac = 00-1D-D8-B7-1C-18}} NetBufferList = 0xFFFFC10D500BE750, ParentNetBufferList = 0x0.

[2]0000.0000::2022/11/30-22:38:59.121549700 [Microsoft-Windows-SlbMuxDriver]SlbMux encapped packet {Outer: src ip = 20.20.20.1, dst ip = 10.10.56.10} {VXLAN: src port = 65024, dst port = 4789, TenantId = 4132} {Inner: src mac = 00-15-5D-8B-04-05, dst mac = 00-1D-D8-B7-1C-18} {Inner: type = 0x800, src ip = 41.40.40.2, dst ip = 41.40.40.3} {protocol = 6, src port = 50031, dst port = 80} NetBufferList = 0xFFFFC10D53BF3720, ParentNetBufferList = 0xFFFFC10D51348510.
```

- What patterns can you observe with traces when it receives the packet?
- What IP addresses are of note? If you search for the IP addresses or MAC addresses within the data collection, what are they associated with?
    i. What is the SLB Manager VIP address?
    ii. What is the Provider address(es)?
    iii. What is the Public IP address(es)?

6. You may also see patterns where the ICMP packet itself is dropped (by design).

```
[0]0000.0000::2022/11/30-22:39:46.576347300 [Microsoft-Windows-SlbMuxDriver]SlbMux processing {src ip = 41.40.40.2, dst ip = 41.40.40.3} {protocol = 1, src port = 1, dst port = 1} NetBufferList = 0xFFFFC10D51348510, ParentNetBufferList = 0x0.
```

```
[0]0000.0000::2022/11/30-22:39:46.576357300 [Microsoft-Windows-
SlbMuxDriver]SLBMUX dropped IPv4 1(ICMPv4) packet which arrived over
1(INTERNAL) interface 5 in compartment 1 with reason: 23(??).
```

7. In this analysis, we can see that the SLB MUX encapsulated the packet and redirected to 10.10.56.10 which is the provider address (PA) where 00-1D-D8-B7-1C-18 (associated with Contoso-VM2) resides. To investigate further, open the netsh trace for 10.10.56.10, which correlates to SDN-HOST01.SDN.LAB in this example.
8. Open the appropriate host trace and examine the data.
    - Can you find the incoming packets being received into vmSwitch and delivered into the appropriate vPort of the vNIC?