# Software Defined Networking (SDN) in Azure Stack HCI training: Lab for Module 3: Virtual networks

Microsoft Corporation
Published: April 23, 2024

## Applies to

SDN training: Module 3: Virtual networks

## Copyright

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. Some examples depicted herein are provided for illustration only and are fictitious.  No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy, use, and modify this document for your internal reference purposes.

© 2024 Microsoft Corporation. All rights reserved.

Microsoft, Azure, Hyper-V, Internet Explorer, Silverlight, SQL Server, Windows, Windows PowerShell, and Windows Server are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

## Revision History

| Release Date | Changes |
|---|---|
| April 23, 2024 | Initial release. |
| | |
| | |
| | |
| | |
| | |

# Contents

# SDN LAB: M3 VIRTUAL NETWORKS

## Overview

This lab module will help guide you through:

- Examine the vSwitch Service on Network Controller.
- Examine the NCHostAgent that resides on the Hyper-V Hosts.
- Create Virtual Networks and deploy Virtual Machines.
    - Validate Control and Dataplane configuration.
- Deploy iDNS into the SDN infrastructure.
- Create Virtual Networking peering between Virtual Networks
- Collect network traces.

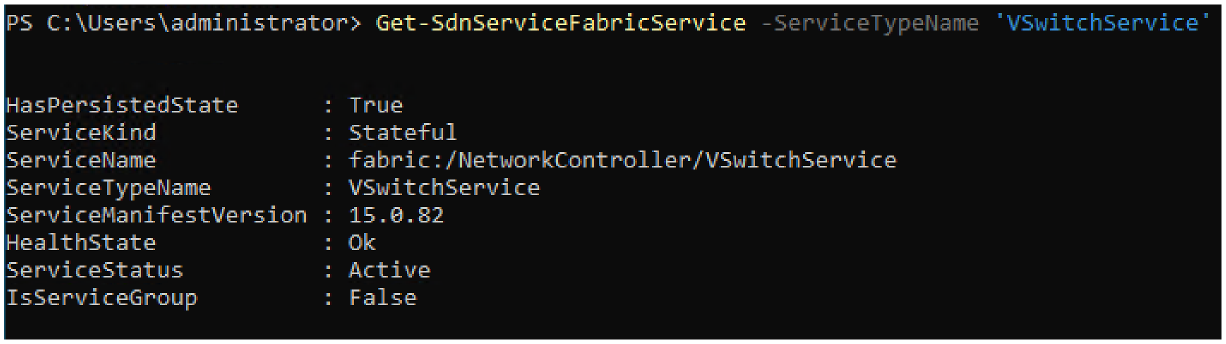
## M3.1 Virtual network architecture

Examine the Virtual Switch service within Network Controller that is responsible for vSwitch and vNet management of resources on southbound devices.

### Lab 1: Examine the vSwitch service

1. Connect to a Network Controller node and get information related to the VswitchService.

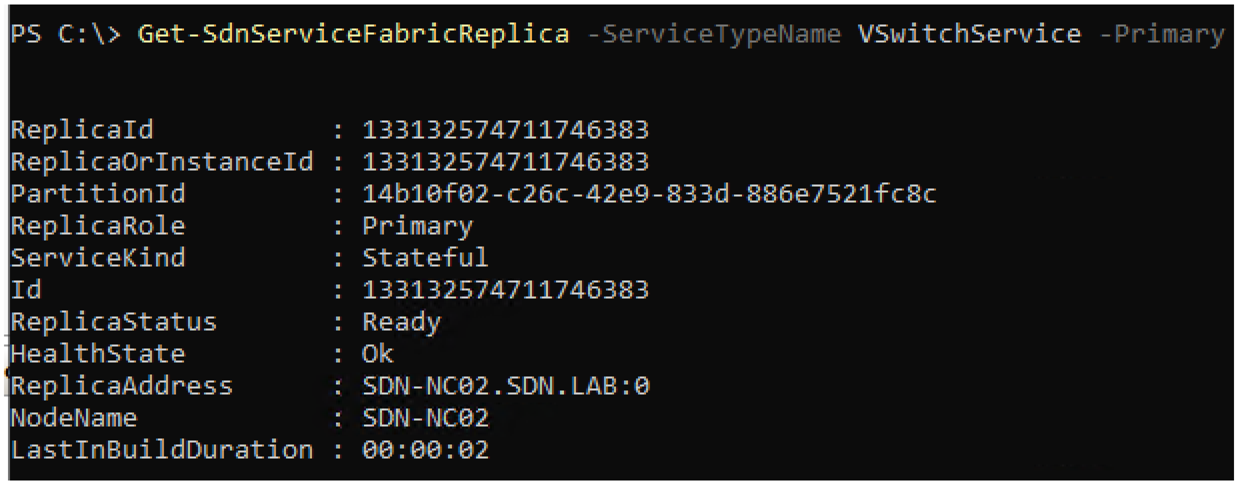
```
Get-SdnServiceFabricService -ServiceTypeName 'VSwitchService'
```



```
PS C:\Users\administrator> Get-SdnServiceFabricService -ServiceTypeName 'VSwitchService'


HasPersistedState      : True
ServiceKind            : Stateful
ServiceName            : fabric:/NetworkController/VSwitchService
ServiceTypeName        : VSwitchService
ServiceManifestVersion : 15.0.82
HealthState            : Ok
ServiceStatus          : Active
IsServiceGroup         : False
```

2. Identify which Network Controller is the primary replica for the VswitchService.

```
Get-SdnServiceFabricReplica -ServiceTypeName VSwitchService -Primary
```



```
PS C:\> Get-SdnServiceFabricReplica -ServiceTypeName VSwitchService -Primary


ReplicaId            : 133132574711746383
ReplicaOrInstanceId  : 133132574711746383
PartitionId          : 14b10f02-c26c-42e9-833d-886e7521fc8c
ReplicaRole          : Primary
ServiceKind          : Stateful
Id                   : 133132574711746383
ReplicaStatus        : Ready
HealthState          : Ok
ReplicaAddress       : SDN-NC02.SDN.LAB:0
NodeName             : SDN-NC02
LastInBuildDuration  : 00:00:02
```

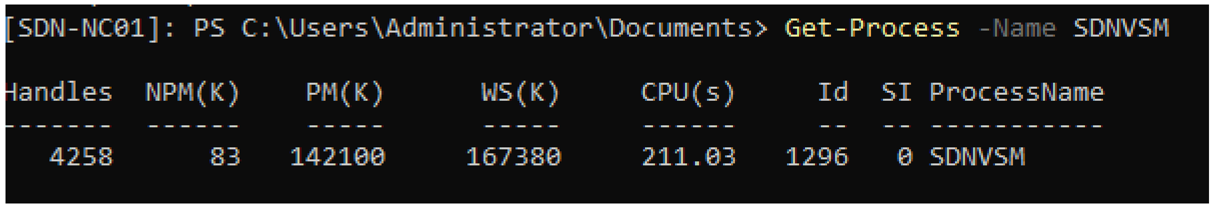
## Lab 1.1 Examine the Southbound connections to NCHostAgent

In this sub-lab, you will connect into the Network Controller node that is hosting the primary replica based on the information you identified above.

**Note:** If you do not have any tenant workloads deployed, there will not be a southbound connection established and are not expected to return any TCP connections.

1. Connect into the Network Controller node that is the primary replica for the VswitchService and get the process related to the vSwitchService

```
Get-Process -Name SDNVSM
```



```
[SDN-NC01]: PS C:\Users\Administrator\Documents> Get-Process -Name SDNVSM

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   4258      83   142100     167380     211.03   1296   0 SDNVSM
```

2. Identify the TCP connections southbound that VSwitchService has established.

```
Get-NetTCPConnection -OwningProcess 1296
```

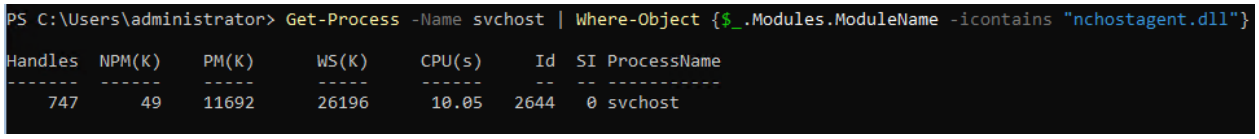
# Lab 2: Examine NCHostAgent

In this lab, you will examine from the NCHostAgent perspective any connections incoming from the VSwitchManager primary replica.

Note: If you do not have any tenant workloads deployed, there will not be a southbound connection established and are not expected to return any TCP connections.

1. Connect to HOST01 or HOST02 and get the svchost process that NCHostAgent is running under.

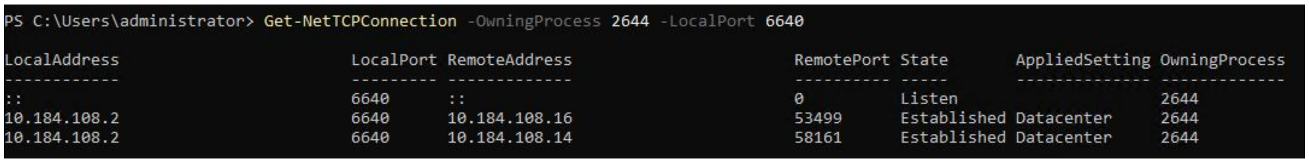
   ```
   Get-Process -Name svchost | Where-Object {$_.Modules.ModuleName -icontains
   "nchostagent.dll"}
   ```



```
PS C:\Users\administrator> Get-Process -Name svchost | Where-Object {$_.Modules.ModuleName -icontains "nchostagent.dll"}

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    747      49    11692      26196      10.05   2644   0 svchost
```

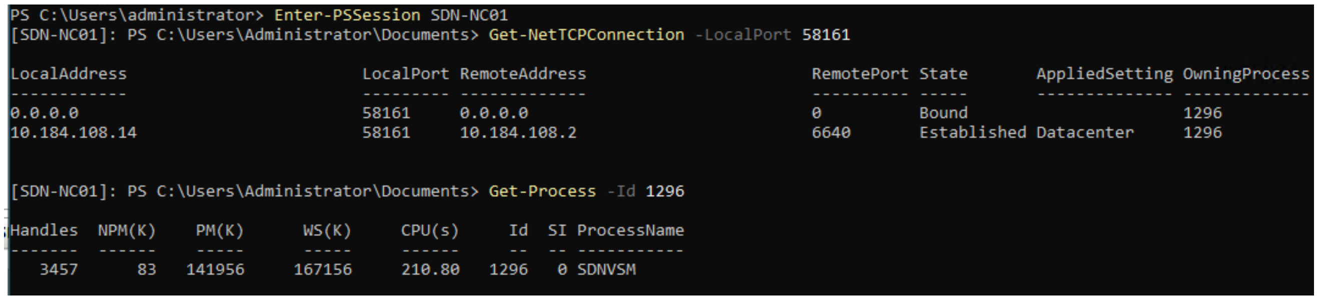
2. Get the TCP connections used by VSwitchService to connect to OVSDB server.

   ```
   Get-NetTCPConnection -OwningProcess 2644
   ```



```
PS C:\Users\administrator> Get-NetTCPConnection -OwningProcess 2644 -LocalPort 6640

LocalAddress                        LocalPort RemoteAddress                        RemotePort State       AppliedSetting OwningProcess
------------                        --------- -------------                        ---------- -----       -------------- -------------
::                                  6640      ::                                   0          Listen                     2644
10.184.108.2                        6640      10.184.108.16                        53499      Established Datacenter     2644
10.184.108.2                        6640      10.184.108.14                        58161      Established Datacenter     2644
```

3. Cross-compare the RemoteAddress and RemotePort properties to the primary replicas for vSwitchServic to determine the correct TCP thread. Both the FirewallService and VswitchService will communicate southbound to NCHostAgent to port 6640, so you may see multiple TCP connections coming from multiple Network Controller nodes, depending on which Network Controller node is the primary replica for the services.



```
PS C:\Users\administrator> Enter-PSSession SDN-NC01
[SDN-NC01]: PS C:\Users\Administrator\Documents> Get-NetTCPConnection -LocalPort 58161

LocalAddress                        LocalPort RemoteAddress                        RemotePort State       AppliedSetting OwningProcess
------------                        --------- -------------                        ---------- -----       -------------- -------------
0.0.0.0                             58161     0.0.0.0                              0          Bound                      1296
10.184.108.14                       58161     10.184.108.2                         6640       Established Datacenter     1296

[SDN-NC01]: PS C:\Users\Administrator\Documents> Get-Process -Id 1296

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   3457      83   141956     167156     210.80   1296   0 SDNVSM
```

## M3.2 Virtual networks

Virtual Network (VNet) is the fundamental building block for your private network in Azure and Azure Stack. VNet enables many types of resources, such as Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is like a traditional network that you'd operate in your own data center but brings with it additional benefits of infrastructure such as scale, availability, and isolation.

## Lab 1: Create virtual network and subnets

1. Open Edge browser on navigate to Windows Admin Center
2. Click on sdnfabric.sdn.lab to connect to SDN infrastructure. If prompted, provide credentials.
3. Navigate to Virtual networks → Inventory and click on +New and provide following details:
    - Name: Contoso-VNET01
    - Address Prefixes: 172.16.0.0/16
    - Subnets:
        i. Name: Subnet01
        ii. Address Prefix: 172.16.0.0/24
4. Click submit

## Lab 2: Create a virtual machine

1. Copy the *-Core.vhdx file from E:\VMs\Template to \\sdn-host01\d$\VMs or \\sdn-host02\d$\VMs under a new folder called Contoso-VM1.
2. Navigate to WAC and connect to the sdnfabric.sdn.lab cluster.
3. Navigate to Virtual Machines -> Add -> +New
4. Specify the following properties:
    - Name: Contoso-VM1
    - Generation: Generation 2
    - Host: (specify the host you copied the .vhdx file to)
    - Path: D:\VMs\Contoso-VM1
    - Virtual processors
        i. Virtual process count: 2
        ii. Enable nested virtualization: $false
        iii. Process compatibility: $false
    - Memory:
        i. Startup memory: 2 GB
        ii. Use dynamic memory: $true
            1. Minimum: 0.5 GB
            2. Maximum: 2GB
    - Network
        i. Virtual switch: SDNSwitch
        ii. Isolation mode: Virtual network (SDN)
        iii. Virtual network: VNET01
        iv. IP Address: 172.16.0.4
    - Storage:

      i. Use an existing virtual hard disk:
         1. Path: (Navigate to D:\VMs\Contoso-VM01 and select vhdx file)

5. Select Create and wait for VM deployment to complete
6. Start the VM

## Lab 3: Examine the VSwitch service and NCHostAgent

Now that workloads have been deployed, perform the steps in Lab 1.1 Examine the Southbound Connections to NCHostAgent and Lab 2: Examine NCHostAgent to see the Southbound connections from Network Controller to NCHostAgent for the Virtual Switch and VNET Manager Service.

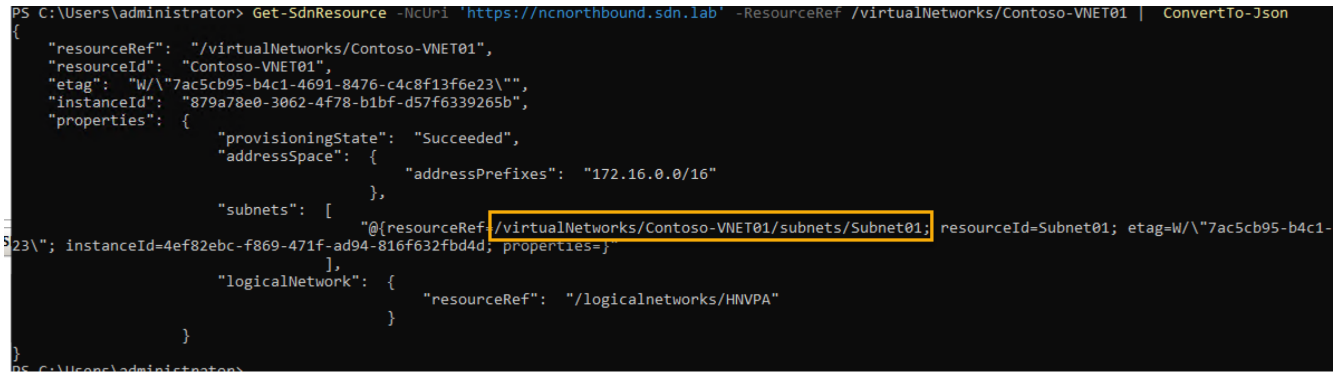
## Lab 4: View virtual network, subnet and network interfaces in Network Controller

1. Connect to SDN-HOST## where you deployed the VM and launch PowerShell
2. Return a list of the Virtual Networks within the environment

```
Get-SdnResource -NcUri "https://NCNORTHBOUND.SDN.LAB" -Resource VirtualNetworks
```

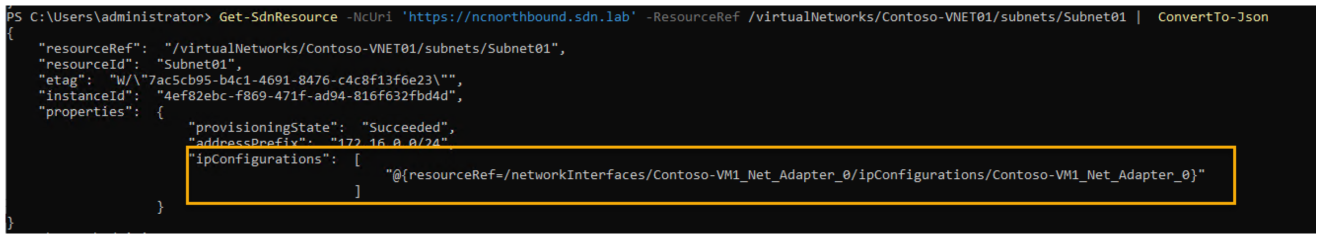
3. Query for the specific URI and convert the output to JSON for viewing within PowerShell window.

```
Get-SdnResource -NcUri "https://NCNORTHBOUND.SDN.LAB" -ResourceRef
/virtualNetworks/Contoso-VNET01 | ConvertTo-Json
```



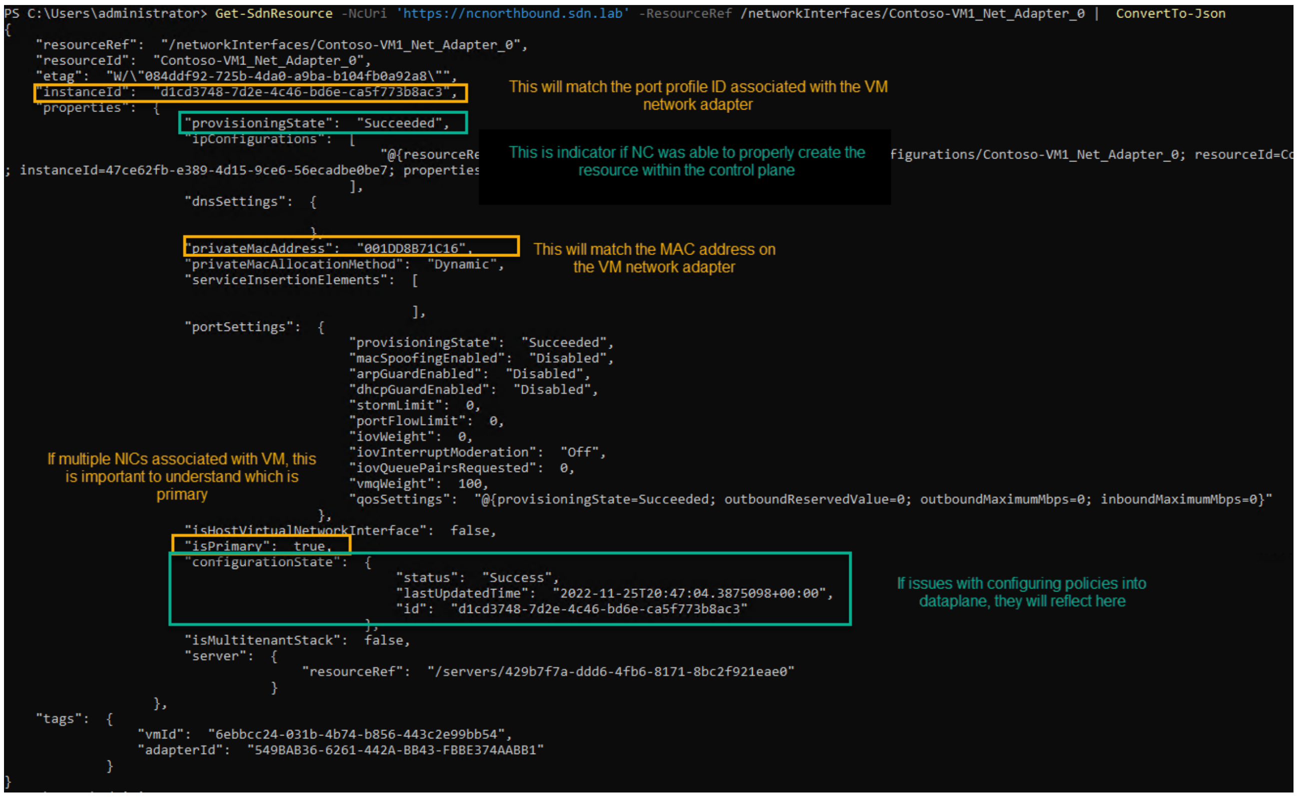
```
PS C:\Users\administrator> Get-SdnResource -NcUri 'https://ncnorthbound.sdn.lab' -ResourceRef /virtualNetworks/Contoso-VNET01 |  ConvertTo-Json
{
    "resourceRef":  "/virtualNetworks/Contoso-VNET01",
    "resourceId":  "Contoso-VNET01",
    "etag":  "W/\"7ac5cb95-b4c1-4691-8476-c4c8f13f6e23\"",
    "instanceId":  "879a78e0-3062-4f78-b1bf-d57f6339265b",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "addressSpace":  {
                                            "addressPrefixes":  "172.16.0.0/16"
                                        },
                       "subnets":  [
                                       "@{resourceRef=/virtualNetworks/Contoso-VNET01/subnets/Subnet01; resourceId=Subnet01; etag=W/\"7ac5cb95-b4c1-
23\"; instanceId=4ef82ebc-f869-471f-ad94-816f632fbd4d; properties=}"
                                   ],
                       "logicalNetwork":  {
                                              "resourceRef":  "/logicalnetworks/HNVPA"
                                          }
                   }
}
```

4. You can modify the command above to leverage **ConvertTo-Json -Depth 10** to expand the properties further, or you can grab the resourceRef in the results and query the data.



```
PS C:\Users\administrator> Get-SdnResource -NcUri 'https://ncnorthbound.sdn.lab' -ResourceRef /virtualNetworks/Contoso-VNET01/subnets/Subnet01 |  ConvertTo-Json
{
    "resourceRef":  "/virtualNetworks/Contoso-VNET01/subnets/Subnet01",
    "resourceId":  "Subnet01",
    "etag":  "W/\"7ac5cb95-b4c1-4691-8476-c4c8f13f6e23\"",
    "instanceId":  "4ef82ebc-f869-471f-ad94-816f632fbd4d",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "addressPrefix":  "172.16.0.0/24",
                       "ipConfigurations":  [
                                                "@{resourceRef=/networkInterfaces/Contoso-VM1_Net_Adapter_0/ipConfigurations/Contoso-VM1_Net_Adapter_0}"
                                            ]
                   }
}
```

5. Examine the ipConfigurations resource that was returned in the previous command. This resource will include properties related to your IP configuration that is applied to the Network Interface that was created and associated to the Contoso-VM1 resource that was created previously. As indicated in the training, there can be multiple ipConfigurations associated to a Network Interface. Take the resourceRef string, and now query the parent object of **/networkInterfaces/{NIC NAME}**

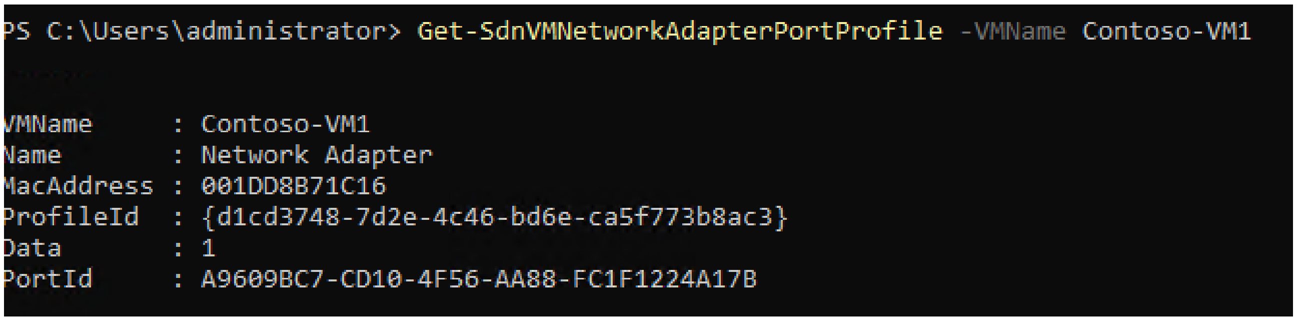


```
PS C:\Users\administrator> Get-SdnResource -NcUri 'https://ncnorthbound.sdn.lab' -ResourceRef /networkInterfaces/Contoso-VM1_Net_Adapter_0 |  ConvertTo-Json
{
    "resourceRef":  "/networkInterfaces/Contoso-VM1_Net_Adapter_0",
    "resourceId":  "Contoso-VM1_Net_Adapter_0",
    "etag":  "W/\"084ddf92-725b-4da0-a9ba-b104fb0a92a8\"",
    "instanceId":  "d1cd3748-7d2e-4c46-bd6e-ca5f773b8ac3",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "ipConfigurations":  [
                                                "@{resourceRe                    figurations/Contoso-VM1_Net_Adapter_0; resourceId=Co
; instanceId=47ce62fb-e389-4d15-9ce6-56ecadbe0be7; properties
                                            ],
                       "dnsSettings":  {

                                       },
                       "privateMacAddress":  "001DD8B71C16",
                       "privateMacAllocationMethod":  "Dynamic",
                       "serviceInsertionElements":  [

                                                    ],
                       "portSettings":  {
                                            "provisioningState":  "Succeeded",
                                            "macSpoofingEnabled":  "Disabled",
                                            "arpGuardEnabled":  "Disabled",
                                            "dhcpGuardEnabled":  "Disabled",
                                            "stormLimit":  0,
                                            "portFlowLimit":  0,
                                            "iovWeight":  0,
                                            "iovInterruptModeration":  "Off",
                                            "iovQueuePairsRequested":  0,
                                            "vmqWeight":  100,
                                            "qosSettings":  "@{provisioningState=Succeeded; outboundReservedValue=0; outboundMaximumMbps=0; inboundMaximumMbps=0}"
                                        },
                       "isHostVirtualNetworkInterface":  false,
                       "isPrimary":  true,
                       "configurationState":  {
                                                  "status":  "Success",
                                                  "lastUpdatedTime":  "2022-11-25T20:47:04.3875098+00:00",
                                                  "id":  "d1cd3748-7d2e-4c46-bd6e-ca5f773b8ac3"
                                              },
                       "isMultitenantStack":  false,
                       "server":  {
                                      "resourceRef":  "/servers/429b7f7a-ddd6-4fb6-8171-8bc2f921eae0"
                                  }
                   },
    "tags":  {
                 "vmId":  "6ebbcc24-031b-4b74-b856-443c2e99bb54",
                 "adapterId":  "549BAB36-6261-442A-BB43-FBBE374AABB1"
             }
}
```

This will match the port profile ID associated with the VM network adapter
This is indicator if NC was able to properly create the resource within the control plane
This will match the MAC address on the VM network adapter
If multiple NICs associated with VM, this is important to understand which is primary
If issues with configuring policies into dataplane, they will reflect here

6. Verify that the Port Profile for the VM adapter is configured properly. This can be done directly on the host using **Get-SdnVMNetworkAdapterPortProfile** cmdlet.

```
Get-SdnVMNetworkAdapterPortProfile -VMName Contoso-VM1
```



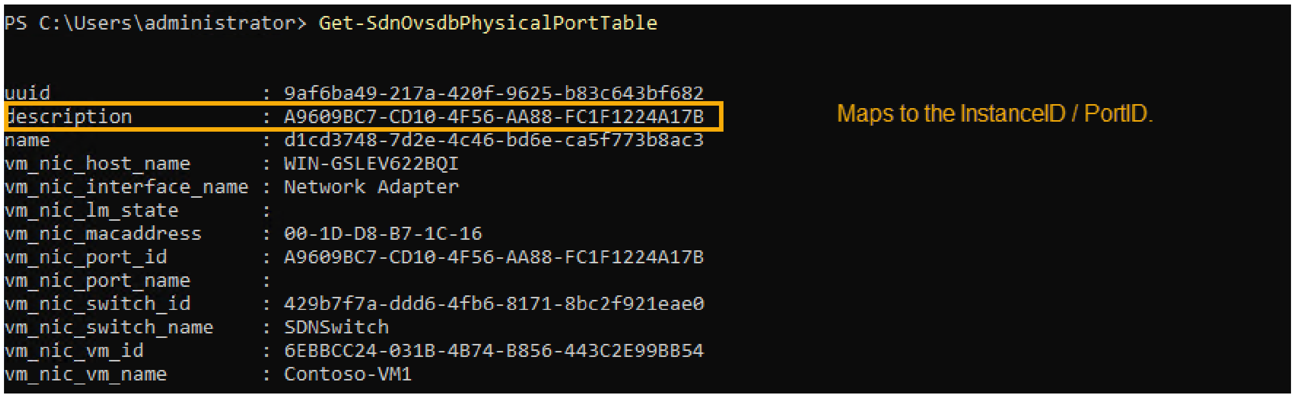
```
PS C:\Users\administrator> Get-SdnVMNetworkAdapterPortProfile -VMName Contoso-VM1

VMName     : Contoso-VM1
Name       : Network Adapter
MacAddress : 001DD8B71C16
ProfileId  : {d1cd3748-7d2e-4c46-bd6e-ca5f773b8ac3}
Data       : 1
PortId     : A9609BC7-CD10-4F56-AA88-FC1F1224A17B
```

   a. There are a couple key attributes that need to be returned in this result. If a warning is returned, or values are not correct, may result in policies not being applied. In the results, the MacAddress should match between NC Network Interface object, as well as VM Network adapter. In addition, the ProfileID returned on the ProfileID should match the InstanceID of the object within NC. Finally, Data property should show 1 meaning VFP enabled. If set to 2, indication that VFP is disabled, and policies will not be managed by NC.

## Lab 5: View dataplane configuration

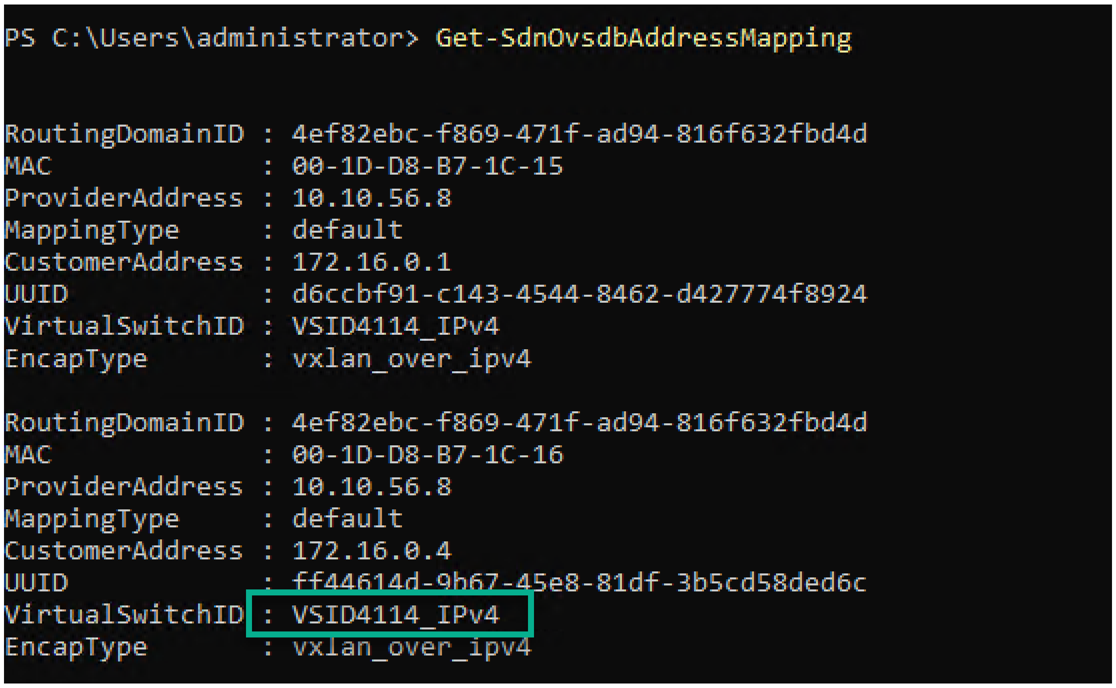
There are several things that get configured into the dataplane for the routing.

1. Examine the OVSDB MS_VTEP database. You can use ovsdb-client.exe dump tcp:127.0.0.1:6641 ms_vtep on the server to output the database results to get an idea of how this data is normally formatted. However, to make things simpler, leverage Get-SdnOvsdbPhysicalPortTable which will output the physical port table of the ms_vtep database into .NET object. This will output the current nodes inventory of the VM network adapters that it is aware of.

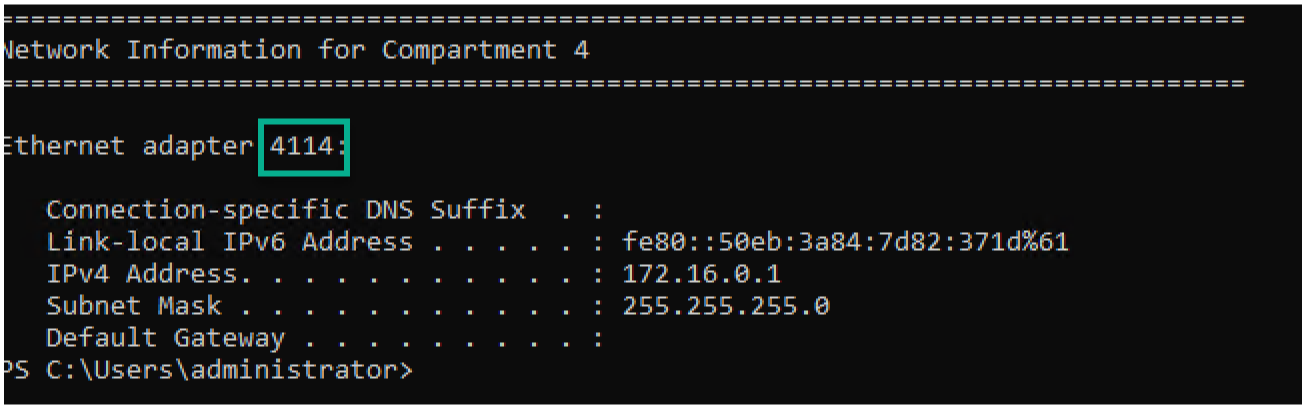


```
PS C:\Users\administrator> Get-SdnOvsdbPhysicalPortTable


uuid                  : 9af6ba49-217a-420f-9625-b83c643bf682
description           : A9609BC7-CD10-4F56-AA88-FC1F1224A17B
name                  : d1cd3748-7d2e-4c46-bd6e-ca5f773b8ac3
vm_nic_host_name      : WIN-GSLEV622BQI
vm_nic_interface_name : Network Adapter
vm_nic_lm_state       :
vm_nic_macaddress     : 00-1D-D8-B7-1C-16
vm_nic_port_id        : A9609BC7-CD10-4F56-AA88-FC1F1224A17B
vm_nic_port_name      :
vm_nic_switch_id      : 429b7f7a-ddd6-4fb6-8171-8bc2f921eae0
vm_nic_switch_name    : SDNSwitch
vm_nic_vm_id          : 6EBBCC24-031B-4B74-B856-443C2E99BB54
vm_nic_vm_name        : Contoso-VM1
```

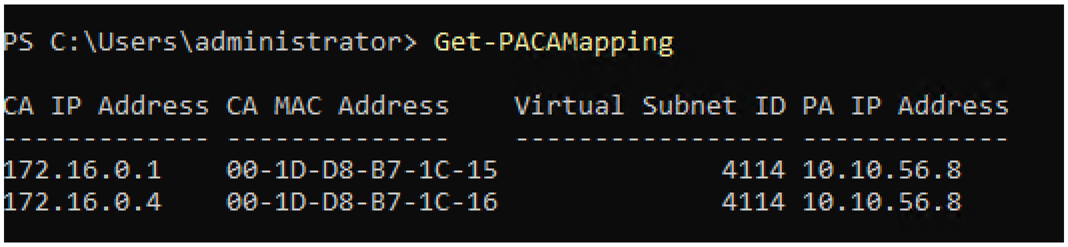
2. To determine the address mappings on the host, use Get-SdnOvsdbAddressMapping to output information related to the VXLAN.


```
PS C:\Users\administrator> Get-SdnOvsdbAddressMapping


RoutingDomainID : 4ef82ebc-f869-471f-ad94-816f632fbd4d
MAC             : 00-1D-D8-B7-1C-15
ProviderAddress : 10.10.56.8
MappingType     : default
CustomerAddress : 172.16.0.1
UUID            : d6ccbf91-c143-4544-8462-d427774f8924
VirtualSwitchID : VSID4114_IPv4
EncapType       : vxlan_over_ipv4

RoutingDomainID : 4ef82ebc-f869-471f-ad94-816f632fbd4d
MAC             : 00-1D-D8-B7-1C-16
ProviderAddress : 10.10.56.8
MappingType     : default
CustomerAddress : 172.16.0.4
UUID            : ff44614d-9b67-45e8-81df-3b5cd58ded6c
VirtualSwitchID : VSID4114_IPv4
EncapType       : vxlan_over_ipv4
```

3. There should be a unique compartment created for each RoutingDomain for workloads that are deployed on the host, which will contain the Distributed Router IP address which is the default gateway for the Virtual Network for the workloads on each of the hypervisor nodes. To view the compartments on the host, use ipconfig /allcompartments.


```
==============================================================================
Network Information for Compartment 4
==============================================================================

Ethernet adapter 4114:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::50eb:3a84:7d82:371d%61
   IPv4 Address. . . . . . . . . . . : 172.16.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
PS C:\Users\administrator>
```

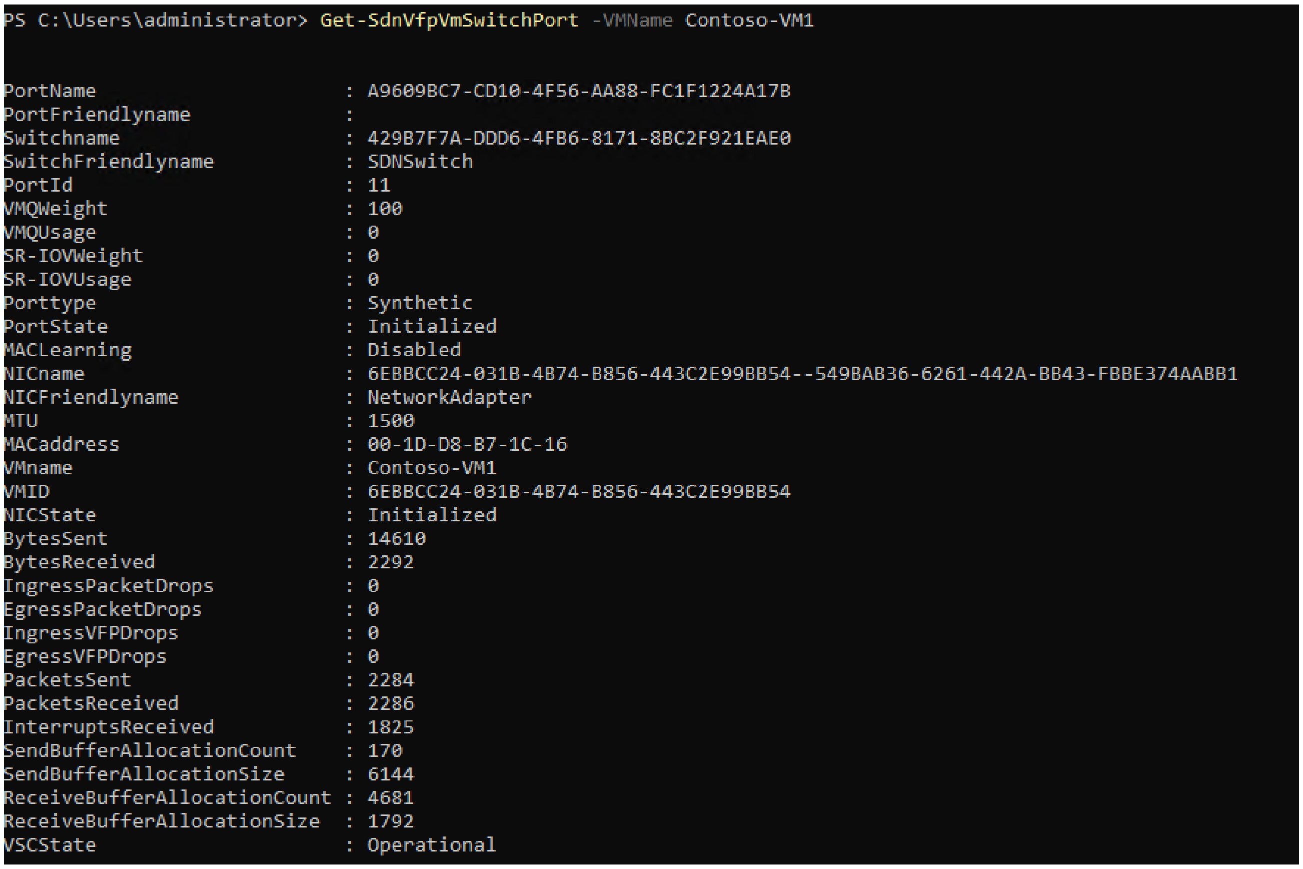
4. There should also be a Provider Address ←→ Customer Address (PACA) mapping that the dataplane will leverage when routing traffic east-west on the Virtual Network between the physical nodes. Run Get-PACAMapping to see the current PACA mappings on the host.



```
PS C:\Users\administrator> Get-PACAMapping

CA IP Address CA MAC Address    Virtual Subnet ID PA IP Address
------------- --------------    ----------------- -------------
172.16.0.1    00-1D-D8-B7-1C-15              4114 10.10.56.8
172.16.0.4    00-1D-D8-B7-1C-16              4114 10.10.56.8
```

5. If you cross reference the PA IP address in your results, you should see they are defined within Compartment 3 when you ran ipconfig /allcompartments in the previous step. All traffic routing east/west will traverse with one of these PA Host Virtual NIC addresses and should see two per host. If you do not see a PA address on the host, or has an APIPA address, then there have been no workloads associated with a Virtual Network deployed yet to the host.
6. Examine the VFP ports associated with the VM switch. This contains some useful information and properties related to the port, as well as some port statistics on packet flows.

```
Get-SdnVfpVmSwitchPort -VMName Contoso-VM1
```

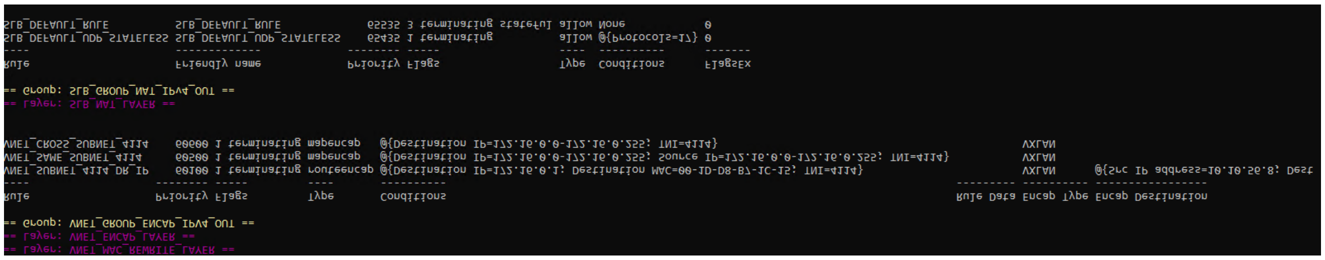


```
PS C:\Users\administrator> Get-SdnVfpVmSwitchPort -VMName Contoso-VM1


PortName                     : A9609BC7-CD10-4F56-AA88-FC1F1224A17B
PortFriendlyname             :
Switchname                   : 429B7F7A-DDD6-4FB6-8171-8BC2F921EAE0
SwitchFriendlyname           : SDNSwitch
PortId                       : 11
VMQWeight                    : 100
VMQUsage                     : 0
SR-IOVWeight                 : 0
SR-IOVUsage                  : 0
Porttype                     : Synthetic
PortState                    : Initialized
MACLearning                  : Disabled
NICname                      : 6EBBCC24-031B-4B74-B856-443C2E99BB54--549BAB36-6261-442A-BB43-FBBE374AABB1
NICFriendlyname              : NetworkAdapter
MTU                          : 1500
MACaddress                   : 00-1D-D8-B7-1C-16
VMname                       : Contoso-VM1
VMID                         : 6EBBCC24-031B-4B74-B856-443C2E99BB54
NICState                     : Initialized
BytesSent                    : 14610
BytesReceived                : 2292
IngressPacketDrops           : 0
EgressPacketDrops            : 0
IngressVFPDrops              : 0
EgressVFPDrops               : 0
PacketsSent                  : 2284
PacketsReceived              : 2286
InterruptsReceived           : 1825
SendBufferAllocationCount    : 170
SendBufferAllocationSize     : 6144
ReceiveBufferAllocationCount : 4681
ReceiveBufferAllocationSize  : 1792
VSCState                     : Operational
```

7. Finally, the last thing we want to do is examine the VFP layers, groups and rules associated with this port. The rules against the port can be extensive, so it is always ideal to understand if you are troubleshooting incoming vs outgoing traffic and whether using IPv4 or IPv6. The PortID in the command will map to the PortName in previous step output. The cmdlet below is doing a more

complex variation of vfpctrl /list-rule /port A9609BC7-CD10-4F56-AA88-FC1F1224A17B in the backend and converting output into .NET objects.

```
# Type [IPv4 | IPv6]
# Direction [IN | OUT]
Show-SdnVfpPortConfig -PortId A9609BC7-CD10-4F56-AA88-FC1F1224A17B -Type IPv4 -
Direction OUT
```



```
== Layer: VNET_MAC_REWRITE_LAYER ==
== Layer: VNET_ENCAP_LAYER ==
== Group: VNET_GROUP_ENCAP_IPv4_OUT ==

Rule                    Priority Flags         Type       Conditions                                                                                      Rule Data Encap Type Encap Destination
----                    -------- -----         ----       ----------                                                                                      --------- ---------- -----------------
VNET_SUBNET_4114_DR_IP    60100 1 terminating routeencap @{Destination IP=172.16.0.1; Destination MAC=00-1D-D8-B7-1C-15; TNI=4114}                                VXLAN      @{Src IP address=10.10.56.8; Dest
VNET_SAME_SUBNET_4114     60200 1 terminating mapencap   @{Destination IP=172.16.0.0-172.16.0.255; Source IP=172.16.0.0-172.16.0.255; TNI=4114}                VXLAN
VNET_CROSS_SUBNET_4114    60600 1 terminating mapencap   @{Destination IP=172.16.0.0-172.16.0.255; TNI=4114}                                                   VXLAN


== Layer: SLB_NAT_LAYER ==
== Group: SLB_GROUP_NAT_IPv4_OUT ==

Rule                      Friendly name                Priority Flags                Type  Conditions      FlagsEx
----                      -------------                -------- -----                ----  ----------      -------
SLB_DEFAULT_UDP_STATELESS SLB_DEFAULT_UDP_STATELESS       65435 1 terminating          allow @{Protocols=17} 0
SLB_DEFAULT_RULE          SLB_DEFAULT_RULE                65535 3 terminating stateful allow None            0
```

# M3.3 Internal DNS (iDNS)

## Lab 1: Deploy iDNS

In this lab, we will configure iDNS and validate that the tenants are able to perform name resolution. The scripts below are using 10.184.108.1 by default. Verify within your lab environment the IP address of your DC that you want to enable for use within iDNS.

1. On each of the Network Controller VMs, we need to configure the TrustedHosts to trust the iDNS server IP address (DC01) that we will be configuring. Failure to do so will result in terminating error when trying to configure iDNS as NC will use WinRM to communicate to the iDNS server to push configuration changes.

```
Invoke-Command -ComputerName 'SDN-NC01','SDN-NC02','SDN-NC03' -ScriptBlock {
    Set-Item WSMan:\localhost\Client\TrustedHosts -Value '10.184.108.1'
}
```

2. Once TrustedHosts have been configured, we need to then failover the primary replica for vSwitchService. If running from a VM not running Network Controller, then add the -NetworkController parameter to the command below.

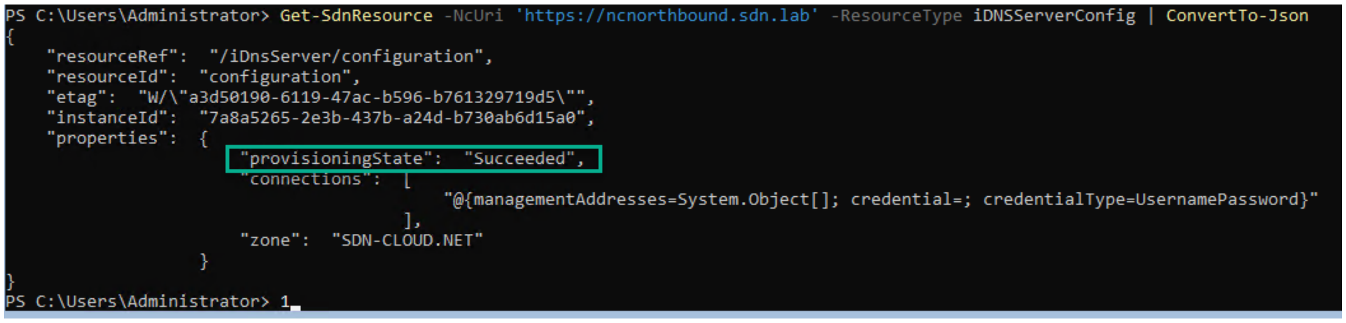
```
Move-SdnServiceFabricReplica -ServiceName
fabric:/NetworkController/VSwitchService
```

3. From one of the Network Controllers, run the following script:

```
$dnsForwarderIPAddress = '10.184.108.1'
$uri = "https://ncnorthbound.sdn.lab"
$domainCredential = Get-Credential -UserName 'SDN.LAB\Administrator' -Message
"Please provide password"

Import-Module NetworkController
$CredentialProperties = New-Object
Microsoft.Windows.NetworkController.CredentialProperties
$CredentialProperties.Type = "UsernamePassword"
$CredentialProperties.UserName = $domainCredential.UserName
$CredentialProperties.Value = $domainCredential.GetNetworkCredential().Password
$iDNSUserObject = New-NetworkControllerCredential -ConnectionURI $uri -
ResourceId "iDNSUser" -Properties $CredentialProperties -Force -
PassInnerException

$iDNSProperties = New-Object
Microsoft.Windows.NetworkController.InternalDNSServerProperties
$iDNSProperties.Connections += New-Object
Microsoft.Windows.NetworkController.Connection
$iDNSProperties.Connections[0].Credential = $iDNSUserObject
$iDNSProperties.Connections[0].CredentialType = $iDNSUserObject.properties.Type
$iDNSProperties.Connections[0].ManagementAddresses = $dnsForwarderIPAddress
$iDNSProperties.Zone = 'SDN-CLOUD.NET'

New-NetworkControllerIDnsServerConfiguration -ConnectionUri $uri -ResourceId
"iDNSConfiguration" -Properties $iDNSProperties -Force -PassInnerException -
Credential $domainCredential
```

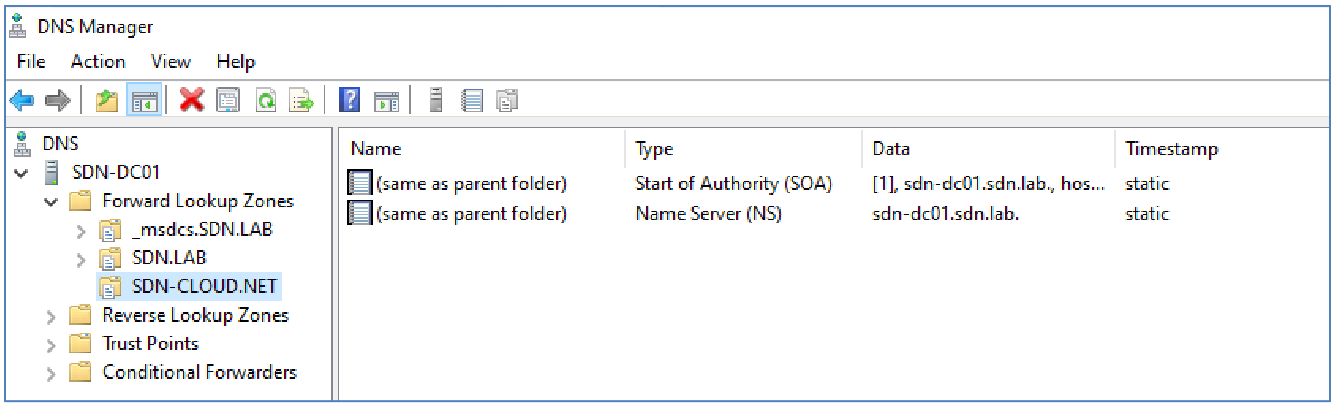
4. Validate that the resource is provisioned correctly.

```
Get-SdnResource -NcUri 'https://ncnorthbound.sdn.lab' -Resource iDNSServerConfig
| ConvertTo-Json
```



```
PS C:\Users\Administrator> Get-SdnResource -NcUri 'https://ncnorthbound.sdn.lab' -ResourceType iDNSServerConfig | ConvertTo-Json
{
    "resourceRef":  "/iDnsServer/configuration",
    "resourceId":  "configuration",
    "etag":  "W/\"a3d50190-6119-47ac-b596-b761329719d5\"",
    "instanceId":  "7a8a5265-2e3b-437b-a24d-b730ab6d15a0",
    "properties":  {
                       "provisioningState":  "Succeeded",
                       "connections":  [
                                           "@{managementAddresses=System.Object[]; credential=; credentialType=UsernamePassword}"
                                       ],
                       "zone":  "SDN-CLOUD.NET"
                   }
}
PS C:\Users\Administrator> 1
```

5. Navigate to DC01 and open DNS Manager. You should be able to see the **SDN-CLOUD.NET** zone under the forward lookup zones.



| Name | Type | Data | Timestamp |
|---|---|---|---|
| (same as parent folder) | Start of Authority (SOA) | [1], sdn-dc01.sdn.lab., hos... | static |
| (same as parent folder) | Name Server (NS) | sdn-dc01.sdn.lab. | static |

6. On each of the hypervisor hosts, run the following PowerShell script.

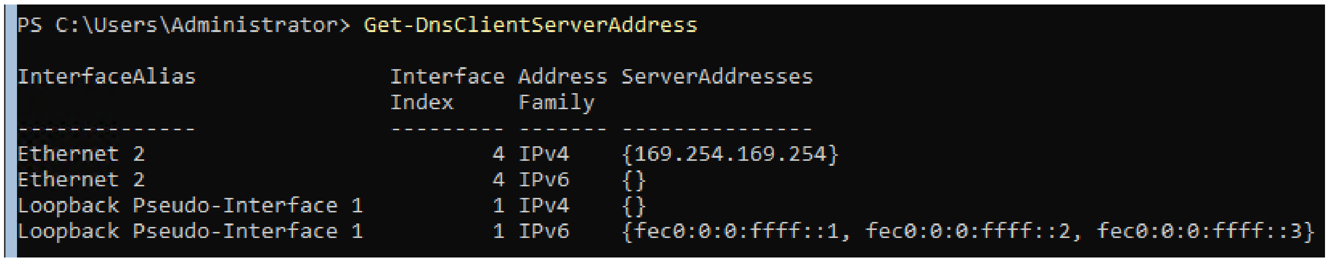
```
# configure iDNS Proxy Service on Hosts
# needs to be executed on each host

$dnsForwarderIPAddress = '10.184.108.1'

$dnsProxyServicePath =
'HKLM:\SYSTEM\CurrentControlSet\Services\NcHostAgent\Parameters\Plugins\Vnet\In
fraServices\DnsProxyService'

if (-NOT (Test-Path -Path $dnsProxyServicePath)) {
    New-Item -Path $dnsProxyServicePath -Force
}

New-ItemProperty -Path $dnsProxyServicePath -Name 'IP' -PropertyType String -
Value '168.63.129.16'
New-ItemProperty -Path $dnsProxyServicePath -Name 'ProxyPort' -PropertyType
DWORD -Value 53
New-ItemProperty -Path $dnsProxyServicePath -Name 'Port' -PropertyType DWORD -
Value 53
New-ItemProperty -Path $dnsProxyServicePath -Name 'MAC' -PropertyType String -
Value "aa-bb-cc-aa-bb-cc"

$dnsRegServicePath =
'HKLM:\SYSTEM\CurrentControlSet\Services\DNSProxy\Parameters'
if (-NOT (Test-Path -Path $dnsRegServicePath)) {
    New-Item -Path $dnsRegServicePath -Force
}

New-ItemProperty -Path $dnsRegServicePath -Name Forwarders -Value
$dnsForwarderIPAddress -PropertyType String
```

7. After the registry key have been set, restart NCHostAgent on each host by running **Restart-Service -Name NCHostAgent -Force**

## Lab 2: Examine tenant DNS settings

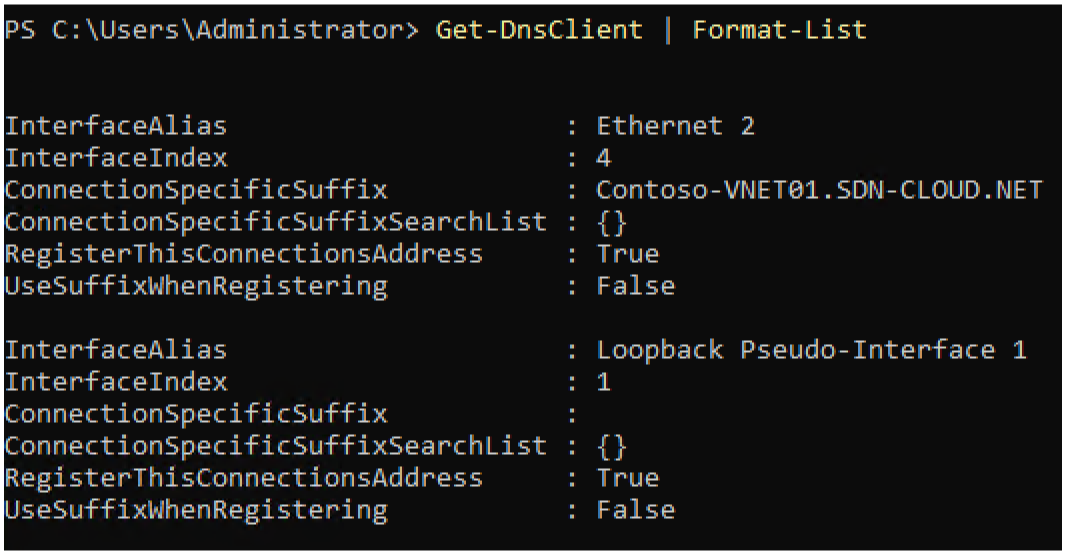
After the configuration has been put into effect, anytime the client boots up and performs DHCP process to get an IP address, other information such as what we configured in the DnsProxyService will be provided as part of the DHCP options.

1. Examine the DNS Client Server Address with **Get-DnsClientServerAddress.**



```
PS C:\Users\Administrator> Get-DnsClientServerAddress

InterfaceAlias               Interface Address ServerAddresses
                             Index     Family
--------------               --------- ------- ---------------
Ethernet 2                           4 IPv4    {169.254.169.254}
Ethernet 2                           4 IPv6    {}
Loopback Pseudo-Interface 1          1 IPv4    {}
Loopback Pseudo-Interface 1          1 IPv6    {fec0:0:0:ffff::1, fec0:0:0:ffff::2, fec0:0:0:ffff::3}
```

2. Examine the DNS Suffix using **Get-DnsClient | Format-List.** You should the ConnectionSpecificSuffix matches the SDN-CLOUD.NET zone we configured in iDNS configuration.



```
PS C:\Users\Administrator> Get-DnsClient | Format-List


InterfaceAlias                     : Ethernet 2
InterfaceIndex                     : 4
ConnectionSpecificSuffix           : Contoso-VNET01.SDN-CLOUD.NET
ConnectionSpecificSuffixSearchList : {}
RegisterThisConnectionsAddress     : True
UseSuffixWhenRegistering           : False

InterfaceAlias                     : Loopback Pseudo-Interface 1
InterfaceIndex                     : 1
ConnectionSpecificSuffix           :
ConnectionSpecificSuffixSearchList : {}
RegisterThisConnectionsAddress     : True
UseSuffixWhenRegistering           : False
```

# M3.4 Virtual network peering

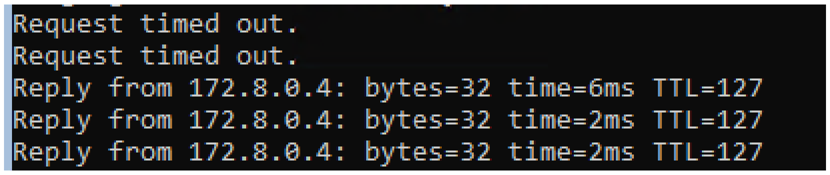
## Lab 1: Create VNET peering

In some instances, there may be a requirement to peer one or multiple Virtual Networks together. This is a common practice, especially in Azure to allow isolation/security between the workloads deployed on different Virtual Networks but want to access shared resources on a different Virtual Network.

1. Perform the same steps in M3.2 [Lab 1]() and [Lab 2]() to create a new Virtual Network and VM
    a. Virtual Network:
        i. Name: Contoso-VNET02
        ii. Address Space: 172.8.0.0/16
        iii. Subnet:
            1. Name: Subnet01
            2. Address Space: 172.8.0.0/24
    b. Virtual Machine:
        i. Name: Contoso-VM2
        ii. IP: 172.8.0.4
        iii. Host: (Different host than where VM1 resides)
2. RDP into Contoso-VM1 and Contoso-VM2 and enable ICMP in the firewall.

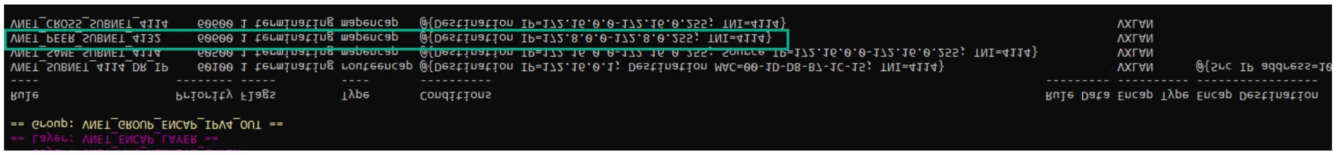
```
netsh advfirewall firewall add rule name="ICMP Allow incoming V4 echo request"
protocol=icmpv4:8,any dir=in action=allow
```

3. Once the firewall has been configured to allow ICMP, start a continuous ping between Contoso-VM1 and Contoso-VM2 using the remote VM's IP address. Ping -t ipaddress
4. Within WAC, navigate to Virtual networks → Inventory → Contoso-VNET01 → Settings
5. Click on Peerings, then +New
    a. Name: VNET01-VNET02-PEER
    b. Virtual networks: Contoso-VNET02
6. Leave other settings default, and click on Submit
7. Once created, do steps 4, 5 and 6 again, this time creating the peering in the opposite direction.
    a. Name: VNET02-VNET01-PEER
    b. Virtual networks: Contoso-VNET01
8. Once successful, navigate back to your RDP session to Contoso-VM# and observe the ping results. You should see ping now flowing between the two VMs



```
Request timed out.
Request timed out.
Reply from 172.8.0.4: bytes=32 time=6ms TTL=127
Reply from 172.8.0.4: bytes=32 time=2ms TTL=127
Reply from 172.8.0.4: bytes=32 time=2ms TTL=127
```

9. If you examine your VFP port configuration, you should see a new VNET_PEER_SUBNET_VSID rule programmed into the VNET_GROUP_ENCAP_IPv4 tables. Use the steps in [M3.2 Lab 4]() if you don't remember the commands offhand to examine the VFP port rules.



```
== Layer: VNET_ENCAP_LAYER ==
== Group: VNET_GROUP_ENCAP_IPv4_OUT ==

Rule                    Priority Flags         Type        Conditions                                                                          Rule Data Encap Type Encap Destination
----                    -------- -----         ----        ----------                                                                          --------- ---------- -----------------
VNET_SUBNET_4114_DR_IP     60100 1 terminating routeencap  @{Destination IP=172.16.0.1; Destination MAC=00-1D-D8-B7-1C-15; VNI=4114}                      VXLAN      @{Src IP address=10
VNET_SAME_SUBNET_4114      60500 1 terminating mapencap    @{Destination IP=172.16.0.0-172.16.0.255; Source IP=172.16.0.0-172.16.0.255; VNI=4114}         VXLAN
VNET_PEER_SUBNET_4132      60600 1 terminating mapencap    @{Destination IP=172.8.0.0-172.8.0.255; VNI=4114}                                              VXLAN
VNET_CROSS_SUBNET_4114     60600 1 terminating mapencap    @{Destination IP=172.16.0.0-172.16.0.255; VNI=4114}                                            VXLAN
```

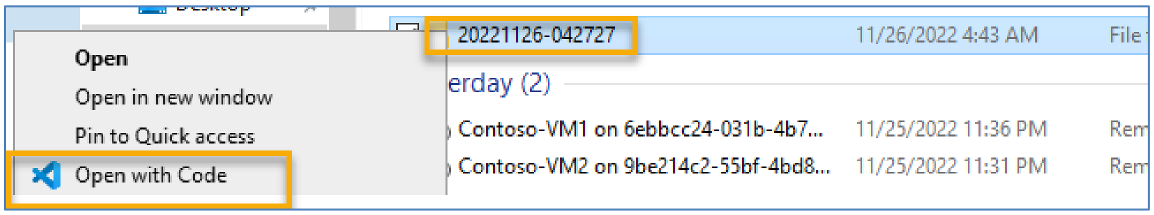
# Lab 2: Collect network traces

In some instances, network traces may need to be collected during a repro to best understand why something is not working as expected.

1. Logon to VM1 or VM2, and setup a continuous ping between the VM
2. Connect to DC01 and run the following script. It will enable tracing on the vmSwitch on Servers, pause for 15 seconds, stop tracing, then start a data collection.

```
Start-SdnNetshTrace -ComputerName 'SDN-HOST01','SDN-HOST02' -Role Server
Start-Sleep -Seconds 15
Stop-SdnNetshTrace -ComputerName 'SDN-HOST01','SDN-HOST02'

Start-SdnDataCollection -NetworkController SDN-NC01 -Role
NetworkController,Server -IncludeLogs -FromDate (Get-Date).AddHours(-1)
```

3. Once completed the results will be saved to directory under C:\Windows\Tracing\SdnDataCollection by default. There is a lot of information captured, and it is highly recommended to analyze the data using Visual Studio Code. Copy the folder back to the lab jump host and use Visual Studio Code to examine the data.



   - {datacollection}\SdnAPIResource: Contains the results taken from the NC northbound interface.
   - {datacollection}\{node}\NetworkTraces: Contains the latest available etl trace as that was identified in the default directory. There will also be a converted txt file.
   - {datacollection}\{node}\ConfigState: Contains critical information related configuration of SDN.
   - {datacollection}\{node}\ConfigState\VFP: Contains the VFP details for each port on the node.
4. Open the _netshTrace.etl file using Network Monitor and apply the following filter. Replace the IP addresses with what you are using for your VM.

```
description.contains("172.16.0.4")

and description.contains("172.8.0.4")
```

5. Unfortunately, Network Monitor does not parse VXLAN packets very well, however the biggest benefit is the parsing ability for ETW events. In the trace, you should see events related to HyperVVfpExt and HyperVVmSwitch that show packets entering/leaving the switches, as well as what happens with packets within VFP as they flow through the layer.

6. In some instances, you may want to see the VXLAN headers, in which case you can save the results as .cap file. File → Save As → Frame selection: All captured frames. Open the .cap file with Wireshark and apply the following filter:

```
ip.addr==172.16.0.4 and ip.addr==172.8.0.4 and icmp
```

7. Examine the results and the details captured in the trace.
    - What can you isolate within the packet headers?