



Software Defined Networking (SDN) in Azure Stack HCI training: Lab for Module 1: Network Controller

Microsoft Corporation

Published: March 15, 2024

Applies to

SDN training: Module 1: Network Controller

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
March 15, 2024	Initial release.

Contents

Applies to.....	1
Overview	5
M1.1 Network Controller Architecture	5
Lab 1: Examine Service Fabric Cluster	5
Lab 2: Disable and Enable Service Fabric Nodes.....	7
Lab 3: Examine an Unhealthy Cluster.....	9
Lab 4: Examine Cluster Authentication	9
M1.2 Northbound API	10
Lab 1: Determine NorthBound API URI.....	10
Lab 2: Examine SDN Fabric Resources	11
M1.3 Southbound API	12
Lab 1: Verify Southbound Certificates	13
M1.4 Network Controller Services	13
Lab 1: Get Service Fabric Service	13
Lab 2: Failover the Primary Replica.....	15
Lab 3: Get Process for Service Fabric Service	16
Lab 4: Examine IMOS Data.....	16
M1.5 Network Controller Host Agent.....	17
Lab 1: Verify Southbound Connectivity.....	17
Lab 2: Open vSwitch Database (OVSDB).....	19
M1.6 Component-Level Tracing.....	20
Lab 1: Examine Diagnostic Log Settings	20
Lab 2: Examine Logman Trace Settings.....	21
Lab 3: Examine Scheduled Tasks	22

Lab 4: Convert and View an ETW Trace.....23

Lab 5: Perform Data Collection..... 24

SDN LAB: M1 NETWORK CONTROLLER

Overview

The Network Controller is a distributed system that runs on a Service Fabric cluster, which provides high availability, scalability, and reliability. The Service Fabric cluster consists of multiple nodes, each hosting one or more replicas of the Network Controller services. The Network Controller services are responsible for managing the network resources and policies across the SDN infrastructure, such as virtual networks, subnets, gateways, load balancers, firewall rules, etc.

In this lab, you will learn how to access and examine the Service Fabric cluster that hosts the Network Controller, using tools such as PowerShell, and REST API. You will also learn how to check the health and status of the cluster, the nodes, and the services, and how to perform some basic operations such as failover and node maintenance. By the end of this lab, you should have a better understanding of the Network Controller architecture and functionality.

M1.1 Network Controller Architecture

As all operations are performed by the Network Controller, such as configuring SLB and GW VMs, routing, network interface operations, etc. it is critical first step to verify the health of the Network Controller as any critical issue impacting the Network Controller could cause all other behaviors/issues with the SDN infrastructure.

This section will explore validating the Service Fabric cluster health, as well as basic operations such as failover of primary replicas and disabling and enabling Service Fabric nodes.

Lab 1: Examine Service Fabric Cluster

The Service Fabric cluster will be composed of one or more Network Controller nodes that are running Service Fabric, with a minimum of three Network Controllers being the minimum for production purposes to provide high availability. To examine the Service Fabric cluster, you can leverage [ServiceFabric Module | Microsoft Learn](#). In addition, there are several functions included in the [SdnDiagnostics](#) that are commonly leveraged.

CommandType	Name	Version	Source
Function	Get-SdnServiceFabricApplicationHealth	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricClusterHealth	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricClusterManifest	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricLog	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricNode	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricReplica	2.2210....	sdndiagnostics
Function	Get-SdnServiceFabricService	2.2210....	sdndiagnostics
Function	Invoke-SdnServiceFabricCommand	2.2210....	sdndiagnostics
Function	Move-SdnServiceFabricReplica	2.2210....	sdndiagnostics

Connect into the Network Controller VM and run **Get-SdnInfrastructureInfo**. This will pre-populate the cache for the cmdlets below.

Examine the number of nodes within the cluster, as well as state of each node by running **Get-SdnServiceFabricNode | Format-Table**.

```
PS C:\Users\Administrator> Get-SdnServiceFabricNode | Format-Table
```

NodeDeactivationInfo	NodeName	IpAddressOrFQDN	NodeType	CodeVersion	ConfigVersion	NodeStatus	NodeUpTime	NodeDownTime	NodeUpAt
	SDN-NC02	SDN-NC02.SDN.LAB	SDN-NC02	7.1.400.9590	12.1.2.0	Up	00:08:21	00:00:00	11/19/2022 3:45:06 AM
	SDN-NC01	SDN-NC01.SDN.LAB	SDN-NC01	7.1.400.9590	12.1.2.0	Up	00:08:22	00:00:00	11/19/2022 3:45:05 AM
	SDN-NC03	SDN-NC03.SDN.LAB	SDN-NC03	7.1.400.9590	12.1.2.0	Up	00:08:22	00:00:00	11/19/2022 3:45:05 AM

Examine the health of the Service Fabric cluster by running **Get-SdnServiceFabricClusterHealth**.

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

```
NodeHealthStates      : {SDN-NC02: Ok, SDN-NC01: Ok, SDN-NC03: Ok}
ApplicationHealthStates : {fabric:/NetworkController: Ok, fabric:/System: Ok}
HealthEvents          : {}
HealthStatistics       :
                        Node           : 3 Ok, 0 Warning, 0 Error
                        Replica        : 31 Ok, 0 Warning, 0 Error
                        Partition      : 11 Ok, 0 Warning, 0 Error
                        Service        : 11 Ok, 0 Warning, 0 Error
                        DeployedServicePackage : 31 Ok, 0 Warning, 0 Error
                        DeployedApplication  : 3 Ok, 0 Warning, 0 Error
                        Application       : 1 Ok, 0 Warning, 0 Error

AggregatedHealthState  : Ok
UnhealthyEvaluations   : {}
```

Examine the health of the applications deployed on the cluster by running **Get-SdnServiceFabricApplicationHealth**.

```

PS C:\Users\Administrator> Get-SdnServiceFabricApplicationHealth

ServiceHealthStates      : {fabric:/NetworkController/FnmService: Ok, fabric:/NetworkController/GatewayManager: Ok, fabric:/NetworkController/HelperService:
                           Ok, fabric:/NetworkController/ServiceInsertion: Ok...}
DeployedApplicationHealthStates : {fabric:/NetworkController+SDN-NC02: Ok, fabric:/NetworkController+SDN-NC03: Ok, fabric:/NetworkController+SDN-NC01: Ok}
HealthEvents              : {SourceId: 'System.CM', Property: 'State', Ok, "Application has been created.", TimeToLive 10675199.02:48:05.4775807,
                           RemoveWhenExpired False, SequenceNumber 115, ReportId CM_7.0_1000: IsExpired False}
HealthStatistics          :
                           Replica      : 31 Ok, 0 Warning, 0 Error
                           Partition    : 11 Ok, 0 Warning, 0 Error
                           Service      : 11 Ok, 0 Warning, 0 Error
                           DeployedServicePackage : 31 Ok, 0 Warning, 0 Error
                           DeployedApplication  : 3 Ok, 0 Warning, 0 Error
ApplicationName           : fabric:/NetworkController
AggregatedHealthState     : Ok
UnhealthyEvaluations      : {}

```

In this Service Fabric cluster, there is only a single application called fabric:/NetworkController which all services reside under. In most other Service Fabric clusters, you may see dozens of applications deployed, with numerous services deployed under each application.

Lab 2: Disable and Enable Service Fabric Nodes

In certain instances, you may need to perform an operation against a Network Controller node where a reboot is required. To help ensure a graceful reboot, it is recommended to disable the node from within the cluster to perform a reboot. To do so, we will need to run Service Fabric cmdlets in this lab. The -NodeName is case-sensitive, so you may want to leverage **Get-SdnServiceFabricNode** or **Get-ServiceFabricNode** to make sure you grab the correct name.

```

Connect-ServiceFabricCluster
Disable-ServiceFabricNode -NodeName 'SDN-NC01' -Intent Restart

```

```

PS C:\Users\Administrator> Connect-ServiceFabricCluster
True

ConnectionEndpoint      :
FabricClientSettings    : {
    ClientFriendlyName      : PowerShell-f3cba9bc-34e5-4750-9a75-e69422d7c2bc
    PartitionLocationCacheLimit : 100000
    PartitionLocationCacheBucketCount : 1024
    ServiceChangePollInterval  : 00:02:00
    ConnectionInitializationTimeout : 00:00:02
    KeepAliveInterval         : 00:00:20
    ConnectionIdleTimeout      : 00:00:00
    HealthOperationTimeout     : 00:02:00
    HealthReportSendInterval   : 00:00:00
    HealthReportRetrySendInterval : 00:00:30
    NotificationGatewayConnectionTimeout : 00:00:30
    NotificationCacheUpdateTimeout : 00:00:30
    AuthTokenBufferSize       : 4096
}
GatewayInformation      : {
    NodeAddress              : SDN-NC01.SDN.LAB:49006
    NodeId                   : 131776f005569cd97baefb4db5222830
    NodeInstanceId           : 133133088677252683
    NodeName                 : SDN-NC01
}

PS C:\Users\Administrator> Disable-ServiceFabricNode -NodeName 'SDN-NC01' -Intent Restart

Confirm
Continue with this operation?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
Successfully accepted request for disabling node 'SDN-NC01'
PS C:\Users\Administrator>

```

Once the command has completed, run `Get-SdnServiceFabricNode` to check the state.

```

NodeName      : SDN-NC01
NodeId        : 131776f005569cd97baefb4db5222830
NodeInstanceId : 133133088677252683
NodeType      : SDN-NC01
NodeStatus    : Disabled
HealthState   : Ok
CodeVersion   : 7.1.409.9590
ConfigVersion : 12.1.2.0
IsSeedNode    : True
IpAddressOrFQDN : SDN-NC01.SDN.LAB
FaultDomain   : fd:/SDN-NC01
UpgradeDomain : SDN-NC01
NodeDeactivationInfo : EffectiveIntent : Restart
                    Status : Completed

                    TaskType : Client
                    TaskId  : 131776f005569cd97baefb4db5222830
                    Intent  : Restart

IsStopped     : False

```



```
Enable-ServiceFabricNode -NodeName 'SDN-NC01'
```

In some situations, you may encounter where Network Controller application is reporting unhealthy. If the majority of the nodes are in a bad state, this is known as Quorum Loss and Network Controller services will be in a read-only state and will not allow any write operations. This means any NB API calls will fail, in addition if live migration happens with virtual machine(s), the policies on the dataplane will become stale, resulting in packet loss for the virtual machine(s).

After the service has stopped, check the application and cluster health for Service Fabric.

[illegible]

Lab 4: Examine Cluster Authentication

Get-SdnServiceFabricClusterManifest

2. Examine Cert:\LocalMachine\My on the Network Controller nodes to see which certificates are present.

```
Get-ChildItem -Path 'Cert:\LocalMachine\My'
```

M1.2 Northbound API

There are numerous SDN resources exposed via the Northbound API. Some are related to the SDN Fabric itself, such as Servers, LoadBalancerManager/Config, LoadBalancerMuxes and Gateways. Other SDN resources may be related to the tenant dataplane configuration, such as NetworkInterfaces, VirtualNetworks, VirtualNetworkGateways, etc.

Lab 1: Determine NorthBound API URI

To examine the resources configured within Network Controller, you first need to determine the Northbound API endpoint that you will perform your REST operations against.

Connect to SDN-NC01 and run **Get-SdnNetworkController**. The FQDN of the NB API will match the ServerCertificate that is configured. Customers may use RestName (Dynamic) or RestIPAddress (Static IP) to define their Northbound API endpoint.

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

Node                : {SDN-NC01}
ClientAuthentication : None
ClientCertificateThumbprint :
ClientSecurityGroup  :
ServerCertificate    : [Subject]
                     CN=NCNORTHBOUND.SDN.LAB
                     [Issuer]
                     CN=NCNORTHBOUND.SDN.LAB
                     [Serial Number]
                     1EA4ED2CC3BAC6BD49C97B515DE2666F
                     [Not Before]
                     5/7/2020 9:30:02 PM
                     [Not After]
                     5/7/2030 9:40:00 PM
                     [Thumbprint]
                     DF9A69DB0D988DA6A6AA33E570BB2F9200A5D362
RestIPAddress        :
RestName              : NCNORTHBOUND.SDN.LAB
Version              : 12.0.10
```

You can also run **Get-SdnInfrastructureInfo**, which will also automatically determine the NB API endpoint. This information will be automatically cached as well within **Global:SdnDiagnostics** that can be referenced later. Additionally, some cmdlets within SdnDiagnostics module will automatically read from this cache, which makes working with the module simpler as required parameters will automatically be populated.

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

Name                           Value
----                           -
RestApiVersion                 V3
FabricNodes                    {SDN-NC01, SDN-HOST01.SDN.LAB, SDN-HOST02.SDN.LAB, SDN-GW01.SDN.LAB...}
NcUrl                          https://NCNORTHBOUND.SDN.LAB
Server                         {SDN-HOST01.SDN.LAB, SDN-HOST02.SDN.LAB}
Gateway                        {SDN-GW01.SDN.LAB, SDN-GW02.SDN.LAB}
SoftwareLoadBalancer           {SDN-MUX01.SDN.LAB, SDN-MUX02.SDN.LAB}
NetworkController              SDN-NC01
```

Lab 2: Examine SDN Fabric Resources

In this section, let's examine the Server resources by using **Get-SdnResource** cmdlet. If you have not run **Get-SdnInfrastructureInfo** already, please do so now to populate the NC URI and other environment variables.

Run **Get-SdnResource** with the **-Resource Servers** defined to return all the Server objects back from Network Controller.

Get-SdnResource -NcUri \$Global:SdnDiagnostics.EnvironmentInfo.NcUrl -Resource Servers

```
PS C:\Users\Administrator> Get-SdnResource -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUrl -ResourceType Servers

resourceRef : /servers/2795c924-e54b-4a0c-9f26-c7b8c1bdde2d
resourceId   : 2795c924-e54b-4a0c-9f26-c7b8c1bdde2d
etag         : W/"2615a063-9a59-40c1-a0bc-09b78fdacfb8"
instanceId   : f9fe6ccc-a2ca-44fd-816f-58db52f7f5b3
properties   : @{provisioningState=Succeeded; connections=System.Object[]; certificate=MIIDFCCAfygAwIBAgIQY7AaAZtku6VD102qBWKY
DAXLlNETi5MQUIwHhcNMjAwNTA3MjE1NjUzWhcNMzAwNTA3MjE1NjUxWjAdMRswGQYDVQQDBJTRE4tSE9TVDAxLlNETi5MQUIwggEiMA0GCSqG
6pjs0C4Qs16HTdTVS3Xs6MR9ak7WMP7nUwsmUBrXT592im/JR1QwD9zWwPRN1INv5fFmscuVnlsy9j+SyxPwzQ5J5EjKCtY2RGg/z4eh8vOuF7A
u/F1W1j4Uetj4ZQH02kL1a4RZDsui9Nh7cVsGpCd6jG+IRSEpqlrvFnt/11EBKNMu6T43ZKXGOV4+TiP5QvHgSd0FZmN+4qzBajq/NhMOX2GLxjc
AgMBAAGjUDBOMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAUBgggrBgEFBQcDAQYIKwYBBQUHAWIwHQYDVRR0BBYEFDCw4tK5ugtqUffGllzbaxSF
/++gJfq/gv2cvoq1HbKDUEJSPs6N0bAoZZHK5zhJnodVwZZAdSK8FaJTnoud0f60JszUsmPNjki0IvKJKnZvoDAdqJCzVKJv1e0KnMi5anznj6wT
tmB6UoxU9foxSYV/nSLK2Rkb8AKgzy7K7pHC9R+Ls5hdZjeyo0l6Cf1TsXNouLkLauUqTB3VfTZwJ4Z+51evZDBCKHo7iam0WjjIFG69N885YINT
o; auditingEnabled=System.Object[]; configurationState=; virtualNetworkInterfaces=System.Object[]; networkInterf

resourceRef : /servers/46a9a408-4e62-4989-8cb3-b85af0652431
resourceId   : 46a9a408-4e62-4989-8cb3-b85af0652431
etag         : W/"a81e9181-74d8-4528-a627-40d7b30f1219"
instanceId   : d0360c5a-44b1-414c-bca7-9db290cc110e
properties   : @{provisioningState=Succeeded; connections=System.Object[]; certificate=MIIDFCCAfygAwIBAgIQeyuFA8BBFrdGrk34W0wo
DAYLlNETi5MQUIwHhcNMjAwNTA3MjE1OTU4WhcNMzAwNTA3MjE1OTU2WjAdMRswGQYDVQQDBJTRE4tSE9TVDAxLlNETi5MQUIwggEiMA0GCSqG
dsyvlN6EnevHe1j5nPrA9Aw4psntXqN/wOFA8p0vyyw2FWD5WYILGveVvpwBpEHbSou4YGuqnN2rvkLQH8HJdpXHDntDOYz741x5+iJ9kjUY2E9c
Rm4Hrd3iDUDuNpV1e6AAPsH1ha5k++S6WGubm98+li7Y4YFFkd9F79XwJEG44sEzYeX06jWSDtz1e/C2fAyf/F7/2FphdwoZP7bH++jYKopInBGL/
AgMBAAGjUDBOMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAUBgggrBgEFBQcDAQYIKwYBBQUHAWIwHQYDVRR0BBYEFB/72bCwrtakaQUHw0znc8EC
aLGG/bXp0CTInSy3Ire1xx9N2IAKmEASx/fiLMGS/x/Ic-fUp/RAT2B+HAsMimlJ/pIt2M7sgmZeZM4cTP2Rajmp3qzI/h/4tbp88Un7cwvB2ODPE
OsHlzMpcdp3xwKFUY/I/61M7WrhEcbyjCzbqJiu7/hQgD1RpmcwXITD941DAbpfSxGEo4+PLbgNvd+xpD461BAZPbi3WJFxs8ov/9Zfo7/MIEYS
y; auditingEnabled=System.Object[]; configurationState=; virtualNetworkInterfaces=System.Object[]; networkInterf
```

```
Get-SdnResource -NCUri $Global:SdnDiagnostics.EnvironmentInfo.NCUri1 -ResourceRef /servers/46a9a408-4e62-4989-8cb3-b85af0652431
```

[illegible]

```
$server = Get-SdnResource -NcUri $Global:SdnDiagnostics.EnvironmentInfo.NcUri -
ResourceRef /servers/46a9a408-4e62-4989-8cb3-b85af0652431
$server.Properties
```

```
PS C:\Users\Administrator> $server = Get-SdnResource -NuRI $global:SdnDiagnostics.EnvironmentInfo.NuRI -ResourceRef /servers/46a9a408-4e62-4989-8cb3-b85af0652431
PS C:\Users\Administrator> $server.properties

provisioningState      : Succeeded
connections             : {@(managementAddresses=System.Object[]; credential=; credentialType=X509Certificate)}
certificate             : {1MDFDCCFAVQYABIA8QYFA8BBFRdGrk34Wm07yTANBghqkhiG9w0BAQsFADADMRsQvQVDDQVQ00DB1TR4EtSE9TVDAyL1NETi5MQUttWhhCNMjAwNTA3MjE10TU4WhhCNMzAwNTA3Mj
MRSuGvQVDDQVQ00DB1TR4EtSE9TVDAyL1NETi5MQUttWhhCNMjE1MA0GCGSgTS13DQCEAQAADIA8BDAwggEKAO1BQRC4tE13XUziU7gPbpblqXdsVylN6EnevHe1j5nPrA9Aw4psntxQn/wOF4
SUI1T3GvEvPwpoBfEHSoSz4YGuqn2rvklQ4H8DdpXDMtdD0Yz741x5s1j9k1JUV2E9Cwcei+TiU6HwCThMiFvcgvVbN1ek0X4AS5E1/RIA7fB1kb16D7Rm4hR31DUDUnpV1e6AAPsF
Gubh9n+1i7Y4YFFKd9F79XwJ3g445EzYeX06jWSdtz1e/C2fAyf/F7/2FphdwzP7bHb+JYKopINBGL/POYRX7svV2iY2D5W92PKkd2kPdmmbqgJvUbHhc+4qB6y1FagBBAAGjUD
pDcE1W/eQEAH/FoADaBgnVHUSFEfJAUBggBgEFBQOCDAQYIKwYBBQUHAWIeHQVDR0B0EYEF/72bWmtkAtKwH0wznC8B7C8SM7MA0GCGS134DQCEAQAADIA8BDAwggEKAO1BQRC4tE13XUziU7gPbpblqXdsVylN6EnevHe1j5nPrA9Aw4psntxQn/wOF4
K2xmxfD0S1hMcpdc3kxKFUY/I/6iM7WrhEcbyJCzbqJiu7/hQgDI8PmcwXITD941dAbpfSxGEO4+PLbgNwd+XPd461BAzPb13MJFxsG8ov/9ZfO/MiEYSQKcBAL1v9TkKp1DLtv5t
Y0g7qU8rPFHk3XQvZey

auditingEnabled         : {}
configurationState      : @status=Success; detailedInfo=System.Object[]; lastUpdatedTime=2022-09-15T07:12:05.7688554+00:00}
virtualNetworkInterfaces : {@(resourceRef=/networkInterfaces/SDN-GW01.SDN_LAB_BackEnd), @(resourceRef=/networkInterfaces/SDN-GW01.SDN_LAB_FrontEnd),
                           @(resourceRef=/networkInterfaces/Contoso-WM01_NIC0), @(resourceRef=/networkInterfaces/Fabrikam-WM01_NIC0)}
networkInterfaces       : {@(resourceRef=/servers/46a9a408-4e62-4989-8cb3-b85af0652431/networkInterfaces/SDNSwitch; resourceID=SDNSwitch;
                           etag=/"a819e181-74d8-4528-a627-4ad7b3k1219"/; instanceId=4b9aae2d-bdc4-47e6-91eb-d9b5d5a5f262; properties={}
```

1. Can you view/retrieve the resource properties for one of the networkInterface objects?
2. Try converting the properties to JSON using `| ConvertTo-Json -Depth 100` to see all the resource properties.
3. Examine other SDN fabric resources, such as Credentials, LogicalNetworks, LoadBalancerMuxes, LoadBalancerManagerConfig, Gateways

Communication with the hosts is over OVSDB protocol while communication with the SLB MUX devices is over

the WCF protocol. For communication with the Servers over OVSDb, Network Controller needs to present a certificate.

Lab 1: Verify Southbound Certificates

1. Examine the Southbound nodes (Servers and MUXes) to see which certificates are present under Cert:\LocalMachine\My to see which certificates are present.
2. If using self-signed certificates, examine the Cert:\LocalMachine\Root store to see which certificates are present.

M1.4 Network Controller Services

Network Controller is an application hosted on Service Fabric, with multiple services such as vSwitchService and SlbManagerService that make up Network Controller.

Lab 1: Get Service Fabric Service

Each Service Fabric Service can be exposed by running:

`Get-SdnServiceFabricService -ApplicationName 'fabric:/NetworkController' -ServiceTypeName ApiService`

```
PS C:\Users\Administrator> Get-SdnServiceFabricService -ApplicationName 'fabric:/NetworkController' -ServiceTypeName ApiService

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

Correspondingly, you can get the Partition Database for a particular service by running:

`Get-SdnServiceFabricPartition -ServiceTypeName ApiService`

```
PS C:\Users\Administrator> Get-SdnServiceFabricPartition -ServiceTypeName 'ApiService'

PartitionId           : ffc1f779-328c-4ae7-a81e-461185204b5d
PartitionKind         : Singleton
DataLossNumber        : 133132574696456340
ConfigurationNumber    : 12884901888
TargetReplicaSetSize  : 3
MinReplicaSetSize     : 3
LastQuorumLossDuration : 00:00:55
PrimaryEpoch         : System.Fabric.Epoch
ServiceKind           : Stateful
HealthState           : Ok
PartitionInformation   : System.Fabric.SingletonPartitionInformation
PartitionStatus       : Ready
```

To get the Service Fabric Replica, you can leverage `Get-SdnServiceFabricReplica`. You can also add `-Primary` switch if you want to only return the primary replica as Network Controller leverages stateful fabric services, which always use Primary and ActiveSecondary replicas.

```
Get-SdnServiceFabricReplica -ApplicationName 'fabric:/NetworkController' -
ServiceTypeName ApiService
```

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

```

ReplicaId      : 133132574711746383
ReplicaOrInstanceId : 133132574711746383
PartitionId    : ffc1f779-328c-4ae7-a81e-461185204b5d
ReplicaRole     : Primary
ServiceKind     : Stateful
Id              : 133132574711746383
ReplicaStatus   : Ready
HealthState     : Ok
ReplicaAddress  : SDN-NC03.SDN.LAB:0
NodeName        : SDN-NC03
LastInBuildDuration : 00:00:02

```

```

ReplicaId      : 133132574912107788
ReplicaOrInstanceId : 133132574912107788
PartitionId    : ffc1f779-328c-4ae7-a81e-461185204b5d
ReplicaRole    : ActiveSecondary
ServiceKind    : Stateful
Id             : 133132574912107788
ReplicaStatus  : Ready
HealthState    : Ok
ReplicaAddress : SDN-NC02.SDN.LAB:0
NodeName       : SDN-NC02
LastInBuildDuration : 00:00:02

```

```

ReplicaId      : 133132574912107789
ReplicaOrInstanceId : 133132574912107789
PartitionId    : ffc1f779-328c-4ae7-a81e-461185204b5d
ReplicaRole     : ActiveSecondary
ServiceKind     : Stateful
Id              : 133132574912107789
ReplicaStatus   : Ready
HealthState     : Ok
ReplicaAddress  : SDN-NC01.SDN.LAB:0
NodeName        : SDN-NC01
LastInBuildDuration : 00:00:02

```

Lab 2: Failover the Primary Replica

In some instances there may not be something working correctly where the policies you are expecting to be programmed into the southbound devices are just not happening. One of the easiest things to do if your Service Fabric cluster is not in quorum loss, is to perform a failover of the primary replica. This is only possible when you have more than 3 Network Controller nodes within your cluster.

```
Move-SdnServiceFabricReplica -ServiceTypeName 'ApiService'
```

[2DM-NC01] ԿԵՅԻՐՅ ԺՈՒ ԳՐԱՐԻՑ: ՎԵՐՄՈՒՔՐՈՒՄԸ ՈՂՂԻՆԵՎԻԻԶԱՆԻՑԵ ԴԱՅ ՐԵՍՈՒՄԵՐ ԺՈՒՄ 2DM-NC03 ԺՈՒ 2DM-NC01
ԲՅ Շ: ՈՂՂԵՆԶ/ԳՊՄԻՆԻՏՐԱԳՐՈՒ ՊՈՒՆ-ԶՈՒԶԵՆԻՑԵԳՐԱՐԻՑԿԵՅԻՐՅ -ԶԵՆԻՑԵԼԿԵՅԻՐՅՈՒՄԵ, ՎԵՐՄՈՒՔՐՈՒՄԸ,

Lab 3: Get Process for Service Fabric Service

Each of the Service Fabric Services has a corresponding process that is running on each of the replica nodes.

```
Get-Process -Name *SDN*
```

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

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
1855	97	189744	220240	68,153.95	9748	0	SDNAPI
1348	52	128524	145384	11.61	6568	0	SDNBR
2640	89	167468	196892	133.70	6908	0	SDNCTLR
1362	53	127752	143592	11.94	3128	0	SDNFNM
2452	67	146556	168716	59.14	9004	0	SDNFW
2230	65	186804	211844	29.77	7100	0	SDNGWM
921	41	97292	122792	37.30	4124	0	SDNHelper
2021	55	127908	145276	10.22	7428	0	SDNSI
2654	94	439940	296456	356.97	8916	0	SDNSLBM
615	44	93404	108972	75.20	1500	0	SDNUpdater
2784	76	125028	154372	94.44	10600	0	SDNVSM

If the processes are not running, this is an indication that something is wrong with the Service Fabric Cluster, Application or Services themselves. If they are not running, the services will not be able to push policies and rules down to NCHostAgent / OVSDB, which would directly impact accessing of resources within the dataplane.

Try This:

- Examine the SDNAPI process details. Where is the SDNAPI.exe located?
- Stop the FabricHostSvc Service. What happens to the SDN processes on that server?

Lab 4: Examine IMOS Data

Service Fabric leverages Key Value Store (KVS) to store the configuration state data. Network Controller adds an In-Memory Object Store (IMOS) schema which provides a database-like abstraction of KVS for the services within Network Controller application. IMOS data can be gathered from Network Controller by performing a PUT request to NetworkControllerState API endpoint to trigger the configuration dump. This operation will result in the primary replicas for each service to output its current IMOS configuration to

C:\Windows\Tracing\SdnDiagnostics\NetworkControllerState.

You can leverage **Get-SdnNetworkControllerState** which will perform the REST API operation, and then will pick up the corresponding service IMOS files from each Network Controller within the Service Fabric cluster.

```
Get-SdnNetworkControllerState -NetworkController 'SDN-NC01' -OutputDirectory
```


"C:\Windows\Tracing\SdnDataCollection\IMOS"

```
PS C:\Users\Administrator> Get-SdnNetworkControllerState -NetworkController SDN-NC01 -OutputDirectory C:\Windows\Tracing\SdnDataCollection\IMOS
[SDN-NC01] Detected that SDN-NC01 is local machine
[SDN-NC01] Copying C:\Windows\Tracing\SDNDiagnostics\NetworkControllerState\* to C:\Windows\Tracing\SdnDataCollection\IMOS\NetworkControllerState
PS C:\Users\Administrator> _
```

Once the command has completed, navigate to the directory (local from where command was executed from) and examine the output.

M1.5 Network Controller Host Agent

An extension of the Network Controller is the NCHostAgent that is installed on the Hyper-V hosts that are managed by the Network Controller. All policies are pushed from services hosted on Network Controller to the OVSDB tables within NCHostAgent to be programmed into the VFP in the dataplane. If for any reason the network changes do not appear to be happening and the Network Controller API is showing the correct state, then need to verify that the Network Controller can talk to the NCHostAgent.

Lab 1: Verify Southbound Connectivity

Verify that the NCHostAgent service is running by running **Get-Service -Name NcHostAgent** on one of the Hyper-V hosts.

```
PS C:\Users\administrator> Get-Service -Name NcHostAgent

Status      Name           DisplayName
-----
Running     NcHostAgent    NC Host Agent
```

When the NCHostAgent service is running, it will create a TLS connection from the Hyper-V host to the SDN API primary replica over TCP Port 6640.

NOTE: In Azure Stack HCI 23H2 systems, this has been changed to port 6645 that NCHostAgent is listening for and receiving connections from Network Controller. Examine HKLM:\SYSTEM\CurrentControlSet\Services\NCHostAgent\Connections to verify the port.

Get-NetTCPConnection -RemotePort 6640

```
PS C:\Users\Administrator> Get-NetTCPConnection -RemotePort 6640

LocalAddress      LocalPort RemoteAddress      RemotePort State      AppliedSetting OwningProcess
-----
10.184.108.3      54981    10.184.108.14      6640     Established Internet      3920
```

If you connect into the Remote Address, you can isolate the process that the TCP connection is related to by focusing on TCP Port 6640.

Get-NetTCPConnection -LocalPort 6640

Get-Process -Id 3968

```
PS C:\Users\Administrator> Get-NetTCPConnection -LocalPort 6640
```

LocalAddress	LocalPort	RemoteAddress	RemotePort	State	AppliedSetting	OwningProcess
10.184.108.14	6640	10.184.108.3	54981	Established	Datacenter	3968
10.184.108.14	6640	10.184.108.2	52657	Established	Datacenter	3968
0.0.0.0	6640	0.0.0.0	0	Listen		3968


```
PS C:\Users\Administrator> Get-Process -Id 3968
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
1807	106	179588	220856	...,716.78	3968	0	SDNAPI

After NHostAgent establishes a TCP connection with SDN API, Virtual Switch Manager and Firewall Service will create southbound connections to NHostAgent as well. If there are no tenant workloads that have been deployed, then there will not be a southbound connection from SDN vSwitch Manager service.

Get-NetTCPConnection -LocalPort 6640

```
PS C:\Users\Administrator> Get-NetTCPConnection -LocalPort 6640
```

LocalAddress	LocalPort	RemoteAddress	RemotePort	State	AppliedSetting	OwningProcess
10.184.108.3	6640	10.184.108.14	60381	Established	Datacenter	3920
10.184.108.3	6640	10.184.108.14	60380	Established	Datacenter	3920
0.0.0.0	6640	0.0.0.0	0	Listen		3920

If you connect into the Remote Address, you can isolate the process that the TCP connection is related to by identifying the OwningProcess for the TCP connection and comparing it with current running processes.

Get-NetTCPConnection -LocalPort 6640

Get-Process -Id #####

```
PS C:\Users\Administrator> Get-NetTCPConnection -LocalPort 60381
```

LocalAddress	LocalPort	RemoteAddress	RemotePort	State	AppliedSetting	OwningProcess
0.0.0.0	60381	0.0.0.0	0	Bound		3412
10.184.108.14	60381	10.184.108.3	6640	Established	Datacenter	3412


```
PS C:\Users\Administrator> Get-Process -Id 3412
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
2314	78	144976	175560	4,975.45	3412	0	SDNVSM

Try this:

- Failover the vSwitchService (SDNVSM) or FirewallService (SDNFW) primary replica on NC multi-node environment. What has happened with the TCP connections to the Hypervisor hosts?

Lab 2: Open vSwitch Database (OVSDDB)

Microsoft's Network Controller pushes policies down to the NC Host Agent running on each hypervisor host using Open vSwitch Database (OVSDDB) Management Protocol via the Southbound Interface (SBI). These policies are represented in schemas that are persisted in the host agent's database. A local ARP responder on the host is then able to catch and respond to all ARP requests from the VMs to provide the destination MAC address of the remote VM. The Host Agent database also contains the VTEP IP address of all hosts attached to the virtual subnet. The Host Agent programs mapping rules into the VFP extension of the Hyper-V Virtual Switch to correctly encapsulate and send the VM packet based on the destination VM.

For more information about OVSDDB, see [RFC 7047](#). For more information about Microsoft's use of OVSDDB, see [What's New in Hyper-V Network Virtualization in Windows Server 2016](#) or [Network Virtualization in the Windows Server 2016 Software Defined Networking \(SDN\) Stack](#).

The data represented in the tables is also stored in configuration files under %ProgramData%\Microsoft\Windows\NCHostAgent on each compute host. By default, there is a ms_vtep, Firewall, and ServiceInsertion configuration file. You can view the configuration on any of the hosts by opening an administrative command prompt and running `ovsdb-client.exe dump tcp:127.0.0.1:6641 database_name`.

Try running the command either locally on SDN-HOST01 or SDN-HOST02, or leveraging SdnDiagnostics module, from any computer.

`Invoke-SdnCommand -ComputerName SDN-HOST01 -ScriptBlock {ovsdb-client.exe dump tcp:127.0.0.1:6641 ms_vtep}`

```
PS C:\Users\Administrator> Invoke-SdnCommand -ComputerName SDN-HOST01 -ScriptBlock {ovsdb-client.exe dump tcp:127.0.0.1:6641 ms_vtep}
Global table
  _uuid                                cur_cfg db_version managers next_cfg other_config switches
-----
b566fc70-6623-4a2a-aa07-122c86ff8647 2317      []           []       2317      {}           [c0e0ef96-6a52-4095-a4ef-d7e4eafd86a0]

Logical_Binding_Stats table
  _uuid bytes_from_local bytes_to_local packets_from_local packets_to_local
-----
Logical_Router table
  _uuid description enable_logical_router name other_config static_routes switch_binding
-----
Logical_Switch table
  _uuid description encryption_credential_thumbprint name other_config static_routes tunnel_key
-----
Manager table
  _uuid inactivity_probe is_connected max_backoff other_config status target
```

Alternatively, SdnDiagnostics has several functions available that automatically take the raw database output to create a PSObject that can be more easily worked with.

```
Get-SdnOvsdbGlobalTable -ComputerName SDN-HOST01
```

```
PS C:\Users\Administrator> Get-SdnOvsdbGlobalTable -ComputerName SDN-HOST01

uuid           : b566fc70-6623-4a2a-aa07-122c86ff8647
switches       : c0e0ef96-6a52-4095-a4ef-d7e4eafd86a0
next_cfg       : 2317
cur_cfg        : 2317
PSComputerName : SDN-HOST01
RunspaceId     : 1c4b8822-043f-4f95-8f7a-22654442c9a2
```

To see which functions are available for enumerating the OVSDB data, run:

```
Get-Command -Module SdnDiagnostics -Name *OVSDB*
```

Try this:

- Stop NCHostAgent service on SDN-HOST01 or SDN-HOST02. Try to run OVSDB commands against the MS_VTEP or MS_Firewall database.

M1.6 Component-Level Tracing

Lab 1: Examine Diagnostic Log Settings

Diagnostic logging is configured using `Get-NetworkControllerDiagnostic` and `Set-NetworkControllerDiagnostic` cmdlets. To see the current diagnostics configuration, run `Get-NetworkControllerDiagnostic`.

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

LogScope           : All
DiagnosticLogLocation :
LogTimeLimitInDays  : 3
LogSizeLimitInMBs   : 15000
LogLevel           : Informational
```

When using the `Set-NetworkControllerDiagnostic` cmdlet, it is actually pushing the configuration into Service Fabric and forcing a cluster upgrade of the application to take the new log configuration settings. You can also see these log settings by examining the cluster manifest directly using `Get-SdnServiceFabricClusterManifest`.

```
<Section Name="Trace/Etw">
  <Parameter Name="Level" Value="4" />
</Section>
```

```

<Section Name="Diagnostics">
  <Parameter Name="ProducerInstances" Value="WinFabEtlFile,WinFabCrashDump,WinFabPerfCtrFolder" />
  <Parameter Name="MaxDiskQuotaInMB" Value="15000" />
  <Parameter Name="AppDiagnosticStoreAccessRequiresImpersonation" Value="false" />
</Section>
<Section Name="WinFabEtlFile">
  <Parameter Name="ProducerType" Value="EtlFileProducer" />
  <Parameter Name="IsEnabled" Value="true" />
  <Parameter Name="EtlReadIntervalInMinutes" Value="5" />
  <Parameter Name="DataDeletionAgeInDays" Value="3" />
</Section>
<Section Name="WinFabCrashDump">
  <Parameter Name="ProducerType" Value="FolderProducer" />
  <Parameter Name="IsEnabled" Value="true" />
  <Parameter Name="FolderType" Value="WindowsFabricCrashDumps" />
</Section>
<Section Name="WinFabPerfCtrFolder">
  <Parameter Name="ProducerType" Value="FolderProducer" />
  <Parameter Name="IsEnabled" Value="true" />
  <Parameter Name="FolderType" Value="WindowsFabricPerformanceCounters" />
  <Parameter Name="DataDeletionAgeInDays" Value="3" />
</Section>

```

These global configuration settings are pushed down to each SDN fabric node into respective registry location.

`Get-ChildItem -Path HKLM:\Software\Microsoft\NetworkController\SDN\Diagnostics`

```

PS C:\Users\Administrator> Get-ChildItem -Path HKLM:\Software\Microsoft\NetworkController\SDN\Diagnostics

Hive: HKEY_LOCAL_MACHINE\Software\Microsoft\NetworkController\SDN\Diagnostics

Name                           Property
----                           -
Parameters                     ChangeInErrorCode              : 0
                               DeviceType                    : 128
                               ErrorCode                          : 0
                               IsLoggingEnabled                : 1
                               IsSDNCtrlPrimaryNode            : 1
                               LastUpdatedTime                  : 1666729512
                               LogLevel                          : 4
                               LogmanBufferSize                  : 256
                               LogmanFileCreationIntervalInMin    : 20
                               LogmanFlushTimeInMin              : 2
                               LogSizeLimit                       : 15000
                               LogTimeLimit                       : 3
                               Password                          : {1, 0, 0, 0...}
                               RemoteLogLocation                 :
                               UserName                          :

```

Lab 2: Examine Logman Trace Settings

By default, ETW trace logging is enabled on any SDN fabric node. Each role will have its own unique ETW providers that tracing is enabled for.

Logman query SdnDiagnosticsTrace

```
PS C:\Users\Administrator> logman query SdnDiagnosticsTrace

Name: SdnDiagnosticsTrace
Status: Running
Root Path: C:\Windows\tracing\SDNDiagnostics\Logs\
Segment: On
Schedules: On
Segment Max Duration: 1200 second(s)
Run as: SYSTEM

Name: SdnDiagnosticsTrace\SDNDiagnosticsTrace
Type: Trace
Output Location: C:\Windows\tracing\SDNDiagnostics\Logs\SDNDiagnosticsTrace_000010.etl
Append: Off
Circular: Off
Overwrite: Off
Buffer Size: 256
Buffers Lost: 0
Buffers Written: 14
Buffer Flush Timer: 120
Clock Type: Performance
File Mode: File
```

In addition to the information above, there will be several other properties related to the unique ETW providers that tracing is enabled for.

```
Provider:
Name: Microsoft-Windows-NetworkController-Framework
Provider Guid: {80355850-C8ED-4336-ADE2-6595F9CA821D}
Level: 4 (win:Informational)
KeywordsAll: 0x0
KeywordsAny: 0xffffffffffff (Session3,Session2,Session1,Session0,Microsoft-Windows-NetworkController-Framework/A
amework/Operational,Microsoft-Windows-NetworkController-Framework/Analytic,Microsoft-Windows-NetworkController-Framework/Debu
,0x200,0x400,0x800,0x1000,0x2000,0x4000,0x8000,0x10000,0x20000,0x40000,0x80000,0x100000,0x200000,0x400000,0x800000,0x1000000,
20000000,0x40000000,0x80000000,0x100000000,0x200000000,0x400000000,0x800000000,0x1000000000,0x2000000000,0x4000000000,0x80000
0000,0x10000000000,0x20000000000,0x40000000000,0x80000000000,0x100000000000,0x200000000000,0x400000000000,0x800000000000,0x1000000000000,
,0x2000000000000,0x4000000000000,0x8000000000000)
Properties: 0
Filter Type: 0
```

Try this:

- Examine logman trace settings for other SDN roles to identify which providers are enabled by default.

Lab 3: Examine Scheduled Tasks

Scheduled tasks enable default logging on the respective SDN fabric in addition to preventing the logs from consuming an unlimited amount of space.

Enumerate current scheduled tasks on SDN-NC01

```
Get-ScheduledTask -TaskName "*SDN*"
```

```
PS C:\Users\Administrator> Get-ScheduledTask -TaskName "*SDN*"

TaskPath                                     TaskName                                     State
-----
\Microsoft\Windows\Network Controller\      SDN Diagnostics Task                       Ready
\Microsoft\Windows\PLA\                     SDNDiagnosticsTrace                        Running
```

To get more details regarding the task itself, you can pipe the output from above to | `Get-ScheduledTaskInfo`

```
PS C:\Users\Administrator> Get-ScheduledTask -TaskName "*SDN*" | Get-ScheduledTaskInfo

LastRunTime      : 10/25/2022 7:30:30 PM
LastTaskResult   : 0
NextRunTime      : 10/25/2022 8:00:00 PM
NumberOfMissedRuns : 0
TaskName         : SDN Diagnostics Task
TaskPath         : \Microsoft\Windows\Network Controller\
PSComputerName   :

LastRunTime      : 10/25/2022 5:31:31 PM
LastTaskResult   : 267009
NextRunTime      :
NumberOfMissedRuns : 0
TaskName         : SDNDiagnosticsTrace
TaskPath         : \Microsoft\Windows\PLA\
PSComputerName   :
```

Lab 4: Convert and View an ETW Trace

There are multiple methods within Microsoft to decode the .etl files into human readable format, however, customers and partners can decode these .etl traces gathered from SDN fabric as these ETW events are manifest based and can be decoded on the systems with the appropriate features and roles installed.

Connect to SDN-NC0# node, and convert one of the .etl trace files and examine the output.

```
$file = Get-ChildItem -Path $Global:SdnDiagnostics.Settings.DefaultLogDirectory -
Filter "*.etl" | sort LastWriteTime -Descending | select -First 1
$result = Convert-SdnEtwTraceToTxt -FileName $file.FullName -Overwrite Yes -Report No
$result

notepad $result.FileName
```

```

PS C:\Users\Administrator> $file = Get-ChildItem -Path $Global:SdnDiagnostics.Settings.DefaultLogDirectory -Filter "*.etl" | sort LastWriteTime
-Descending | select -First 1
PS C:\Users\Administrator> $result = Convert-SdnEtwTraceToTxt -FileName $file.FullName -Overwrite Yes -Report No
PS C:\Users\Administrator> $result

Status   FileName
-----
Success C:\Windows\tracing\SDNDiagnostics\Logs\SDNDiagnosticsTrace_000014.txt

PS C:\Users\Administrator> notepad $result.FileName
PS C:\Users\Administrator>

```

Alternatively, if you are not actively on the system and need to decode the files, you can leverage `netsh trace convert` which is native Windows command that will decode these files.

Lab 5: Perform Data Collection

In some instances, we cannot always get to the root cause of customer issues during a single troubleshooting session or may have reached a technical roadblock. In these instances, it is recommended to use **Start-SdnDataCollection** to gather a comprehensive set of data from the SDN fabric that can be analyzed. SdnDiagnostics module itself can be executed from any location if the user has appropriate permissions to access the Network Controller NB API and SDN fabric nodes.

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

```

PS C:\Users\Administrator> Start-SdnDataCollection -Role NetworkController,Server -IncludeLogs -FromDate (Get-Date).AddHours(-1)
[SDN-NC01] Starting SDN Data Collection
[SDN-NC01] Required: 10 GB | Available: 11.1802177429199 GB
[SDN-NC01] Results will be saved to C:\Windows\Tracing\SdnDataCollection\20221025-211834
[SDN-NC01] Node SDN-NC01 with role NetworkController added for data collection
[SDN-NC01] Node SDN-HOST01.SDN.LAB with role Server added for data collection
[SDN-NC01] Node SDN-HOST02.SDN.LAB with role Server added for data collection
[SDN-NC01] Performing cleanup of C:\Windows\Tracing\SdnDataCollection\Temp directory across SDN-NC01
[SDN-NC01] Collect configuration state details for NetworkController nodes: SDN-NC01
[SDN-NC01] Collect configuration state details for role NetworkController
[SDN-NC01] Required: 100 MB | Available: 11424.93359375 MB
[SDN-NC01] Collect general configuration state details
[SDN-NC01] Required: 100 MB | Available: 11424.36328125 MB
[SDN-NC01] Detected that SDN-NC01 is local machine
[SDN-NC01] Copying C:\Windows\Tracing\SDNDiagnostics\NetworkControllerState\* to C:\Windows\Tracing\SdnDataCollection\20221025-211834\NetworkControllerState
[SDN-NC01] Checking for any previous network traces and moving them into C:\Windows\Tracing\SdnDataCollection\Temp
[SDN-NC01] Collect service fabric logs for NetworkController nodes: SDN-NC01
[SDN-NC01] Collect Service Fabric logs between 10/25/2022 8:18:33 PM and 10/25/2022 9:19:50 PM UTC
[SDN-NC01] Required: 1 GB | Available: 11.1445999145508 GB

```

```

PS C:\Users\Administrator> netsh trace convert
[SDN-NC01] D9f9 c0j7ecfjou combjefeq
[SDN-NC01] b64f0wmtu8 c7e9uub 04 C:/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/1emb q7uecfouλ 9c0022 2DM-NC01' 2DM-H0210T'2DM'FVB' 2DM-H0210S'2DM'FVB
T83q/zqu-m02f0S'zqu'J9p
[SDN-NC01] coblyu8 //2DM-H0210S'2DM'FVB/C2/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/zqud9f9u02fice~lu9ce0nfbne~S0SS10S2'c2λ fo C:/Mtuqom2/lu9ciub/zqud9f9
[SDN-NC01] coblyu8 //2DM-H0210S'2DM'FVB/C2/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/1emb fo C:/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/S0SS10S2-S1T83q/zqu-m02f
T83q/zqu-m02f0T'zqu'J9p
[SDN-NC01] coblyu8 //2DM-H0210T'2DM'FVB/C2/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/zqud9f9u02fice~lu9ce0nfbne~S0SS10S2'c2λ fo C:/Mtuqom2/lu9ciub/zqud9f9
[SDN-NC01] coblyu8 //2DM-H0210T'2DM'FVB/C2/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/1emb fo C:/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/S0SS10S2-S1T83q/zqu-m02f
[SDN-NC01] coblyu8 C:/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/zqud9f9u02fice~lu9ce0nfbne~S0SS10S2'c2λ fo C:/Mtuqom2/lu9ciub/zqud9f9c0j7ecfjou/S0SS10S2-S
[SDN-NC01] D6f6efeq f94f 2DM-NC01 f2 j0c9J w9cmtu8

```

Logs by default will be saved under `C:\Windows\Tracing\SdnDiag\SdnDataCollection_{date-timestamp}`. The data collected will depend on which parameters were defined and what roles were collected. Take some time to explore the dataset that was collected to get a feel for the content that is collected. [Visual Studio Code](#) is useful

for this scenario as it allows you to open the parent directory and examine the resources. Extensive marketplace extensions make viewing certain file types much easier.