5

Managing Shared Data

In this chapter, we cover the following recipes:

* Setting up and securing an SMB file server
* Creating and securing SMB shares
* Accessing data on SMB shares
* Creating an iSCSI target
* Using an iSCSI target
* Configuring a DFS Namespace
* Configuring DFS Replication

# Introduction

Sharing data with other users on your network has been a feature of computer operating systems from the very earliest days of networking. This chapter looks at Windows Server 2019 features that enable you to share files and folders and to use the data that you've shared. This chapter follows on from [Chapter 4](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch04.html), Managing Windows Storage.

Microsoft's LAN Manager was the company's first network offering. It enabled client computers to create, manage, and share files in a secure manner. The protocol that LAN Manager used to provide this client/server functionality was an early version of the Server Message Block (SMB) protocol.

SMB is a file-level storage protocol running over TCP/IP. SMB enables you to share files and folders securely and reliably. To increase reliability for SMB servers, you can install a cluster and cluster the file server role. This is an active-passive solution and works great as long as the underlying data is accessible.

This chapter shows you how to implement and leverage the features of sharing data between systems, including SMB contained in Windows Server 2016. In the recipes in this chapter, you'll begin by creating and using basic SMB file sharing. Then you build an iSCSI infrastructure, which you'll leverage when building an SOFS.

You finish by looking at the Distributed File System (DFS). With DFS, you can provide the means to connect to multiple shared folders, held on a variety of servers through DFS Namespace. A DFS Namespace is the virtual view of the files and folders with a DFS installation.

In the first recipe, Securing an SMB file server, you harden the security on your SMB file server. Then, in the Creating and securing SMB shares and Accessing SMB shares recipes, you set up simple file-folder sharing and access the shared files. With the Creating an iSCSI target recipe, you create an iSCSI target on the SRV2 server, while in the Using an iSCSI target recipe, you make use of that shared iSCSI disk from FS1. iSCSI is a popular Storage Area Networking (SAN) technology, and these recipes show you how to use the Microsoft iSCSI initiator and target features.

There are two separate features under the banner of the DFS. DFS Namespaces allows you to create a logical folder structure that you distribute across multiple computers. DFS Replication replicates data held on DFS target folders to provide a transparent, fault-tolerant, and load-balancing DFS implementation. Note that DFS Replication is separate from the Storage Replica feature discussed in [Chapter 4,](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch04.html) Managing Windows Storage.

In the Configuring a DFS Namespace recipe, you'll set up a domain-based DFS Namespace. And then, you'll configure and set up DFS Replication in the Configuring DFS Replication recipe.

There are a number of servers involved in the recipes in this chapter—each recipe describes the specific server you use for that recipe. As with other chapters, all the servers are members of the Reskit.Org domain.

# Setting up and securing an SMB file server

The first step in creating a file server is to install the necessary features to the server, then harden it. You use the Add-WindowsFeature cmdlet to add the features necessary for a file server. You can then use the Set-SmbServerConfiguration cmdlet to improve the configuration.

Since your file server can contain sensitive information, you must take reasonable steps to avoid some of the common attack mechanisms and adopt best security practices. Security is a good thing but, as always, be careful! By locking down your SMB file server too hard, you can lock some users out of the server. SMB 1.0 has a number of weaknesses and, in general, should be removed. But, if you disable SMB 1.0, you may find that older computers (for example, those running Windows XP) lose the ability to access shared data. Before you lock down any of the server configurations, be sure to test your changes carefully.

## Getting ready

Run this recipe on FS1, a new server in the Reskit.Org domain.

## How to do it...

1. Add the FileServer features and RSAT tools to FS1:

$Features = 'FileAndStorage-Services','File-Services',

'FS-FileServer','RSAT-File-Services'

Add-WindowsFeature -Name $Features

1. Retrieve the SMB Server settings:

Get-SmbServerConfiguration

1. Turn off SMB1:

$CHT = @{

EnableSMB1Protocol = $false

Confirm = $false

}

Set-SmbServerConfiguration @CHT

1. Turn on SMB signing and encryption:

$SHT1 = @{

RequireSecuritySignature = $true

EnableSecuritySignature = $true

EncryptData = $true

Confirm = $false

}

Set-SmbServerConfiguration @SHT1

1. Turn off the default server and workstations shares:

$SHT2 = @{

AutoShareServer = $false

AutoShareWorkstation = $false

Confirm = $false

}

Set-SmbServerConfiguration @SHT2

1. Turn off server announcements:

$SHT3 = @{

ServerHidden = $true

AnnounceServer = $false

Confirm = $false

}

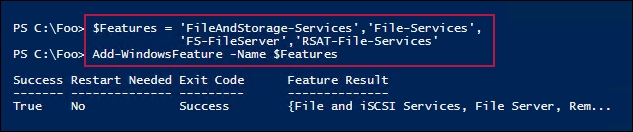
Set-SmbServerConfiguration @SHT3

1. Restart the service with the new configuration:

Restart-Service lanmanserver

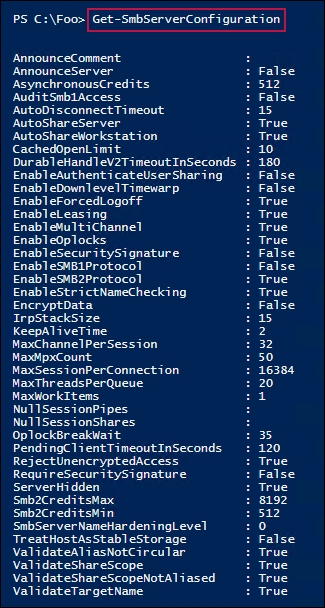
## How it works…

In step 1, you add four features to the FS1 server (FileAndStorage-Services,File-Services,FS-FileServer,RSAT-File-Services), which looks like this:



In step 2, after adding the necessary features to FS1, you get the SMB server configuration using the Get-SMBServerConfiguration cmdlet. This returns 43 separate configuration properties. You can change some of these to harden your SMB server or to accommodate unique aspects of your infrastructure. Some of these properties, however, are relatively obscure—if you don't know what they do, consider leaving them at their default values.

The output of this step looks like this:



In step 3, you turn off SMB1. In step 4, you configure the server to sign all SMB packets and to encrypt data transferred via SMB. SMB signing is particularly useful to reduce the risk of a man-in-the-middle attack. Requiring data encryption increases the security of your organization's data as it travels between server and client computers. Another benefit of using SMB encryption versus something such as IPSec is that deployment is just a matter of adjusting SMB server configuration.

Windows has a set of administrative shares it creates by default. In most cases, you can disable these. In step 5, you turn off the default server/workstation shares. It's important to note that when setting up DFS Replication, the DFS Replication cmdlets require access to these administrative shares.

With step 6, you also turn off server announcements, which reduces the visibility of your file server to hackers.

With those configuration items updated, in step 7, you restart the file server service lanmanserver. Note that restarting the service closes any active connections. Ensure you restart during a scheduled maintenance outage or when you're certain the server is inactive.

These final five steps created no output. After you update these configuration settings, you can use Get-SMBServerConfiguration to confirm the correct server settings are in place.

## There's more...

In this recipe, you hardened a full installation of Windows Server 2019. To further harden your file server, consider using Server Core for your hosts.

In step 3, you disabled SMB1. SMB1 is an older and less secure version of the SMB protocol and could represent an attack vector. The downside of disabling it is that older client computers only support SMB1 and could cease to access shared data if you disable SMB1. Older clients include Windows XP and Windows Server 2003. Windows Vista/Server 2008 and later versions of Windows have built-in support for SMB2. So, as long as you're running fully-supported clients and server systems, you should be able to turn off SMB1.

For large organizations, you should consider using the AuditSmb1Access configuration setting. This setting logs access your server via SMB1. To discover any older SMB clients that would be affected by disabling SMB1, you can search the SMB event log.

# Creating and securing SMB shares

With your file server service set up, the next step in deploying a file server is to create SMB shares and secure them. For decades, administrators have used the net.exe command to set up shared folders and to do a lot more. These continue to work, but you may find the new cmdlets easier to use, particularly if you're automating large-scale SMB server deployments.

This recipe looks at creating and securing shares on a Server 2019 platform using the PowerShell SMBServer module. You also use cmdlets from the NTFSSecurity module (a third-party module you download from the PS Gallery).

## Getting ready

You run this recipe on the file server (FS1) that you set up and hardened in the Setting up and securing your SMB server recipe. In this recipe, you share a folder (C:\Foo) on the file server. You created this folder previously. Then, you create a file in the C:\Foo folder you just shared and set the ACL for the files to be the same as for the share. You use the Set-SMBPathAcl cmdlet to do this. You then review the ACL for both the folder and the file.

This recipe uses a global security group, Sales, which you create in the Reskit.Org domain. See the introduction section in [Chapter 7](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch07.html), Managing Printers, for the script snippet you can use to create the groups, users, and group memberships used by this recipe.

In this recipe, you use the Get-NTFSAccess cmdlet from NTFSSecurity, a third-party module that you downloaded from the PowerShell Gallery. See the Managing NTFS permissions recipe for more details about this module and for instructions on how to download it.

## How to do it...

1. Discover existing shares and access rights:

Get-SmbShare -Name \* |

Get-SmbShareAccess |

Format-Table -GroupBy Name

1. Share a folder:

New-SmbShare -Name Foo -Path C:\Foo

1. Update the share to have a description:

$CHT = @{Confirm=$False}

Set-SmbShare -Name Foo -Description 'Foo share for IT' @CHT

1. Set folder enumeration mode:

$CHT = @{Confirm = $false}

Set-SMBShare -Name Foo -FolderEnumerationMode AccessBased @CHT

1. Set encryption on the Foo share:

Set-SmbShare –Name Foo -EncryptData $True @CHT

1. Remove all access to the Foo share:

$AHT1 = @{

Name = 'Foo'

AccountName = 'Everyone'

Confirm = $false

}

Revoke-SmbShareAccess @AHT1 | Out-Null

1. Add Reskit\Administrator to have Read access to the share:

$AHT2 = @{

Name = 'foo'

AccessRight = 'Read'

AccountName = 'Reskit\ADMINISTRATOR'

ConFirm = $false

}

Grant-SmbShareAccess @AHT2 | Out-Null

1. Add Full access for the OS:

$AHT3 = @{

Name = 'foo'

AccessRight = 'Full'

AccountName = 'NT Authority\SYSTEM'

Confirm = $False

}

Grant-SmbShareAccess @AHT3 | Out-Null

1. Set Creator/Owner to have Full access:

$AHT4 = @{

Name = 'foo'

AccessRight = 'Full'

AccountName = 'CREATOR OWNER'

Confirm = $False

}

Grant-SmbShareAccess @AHT4 | Out-Null

1. Grant Sales administrators Read access, and grant SalesAdmins Full access:

$AHT5 = @{

Name = 'Foo'

AccessRight = 'Read'

AccountName = 'Sales'

Confirm = $false

}

Grant-SmbShareAccess @AHT5 | Out-Null

$AHT6 = @{

Name = 'Foo'

AccessRight = 'Full'

AccountName = 'SalesAdmins'

Confirm = $false

}

Grant-SmbShareAccess @AHT6 | Out-Null

1. Review the ACL on the Foo share:

Get-SmbShareAccess -Name Foo |

Sort-Object AccessRight

1. Set the ACL file to be same as the shared ACL:

Set-SmbPathAcl -ShareName 'Foo'

1. Create a file in C\Foo:

'foo' | Out-File -FilePath C:\Foo\Foo.Txt

1. Set the ACL file to be same as the shared ACL:

Set-SmbPathAcl -ShareName 'Foo'

1. View the ACL folder using Get-NTFSAccess:

Get-NTFSAccess -Path C:\Foo |

Format-Table -AutoSize

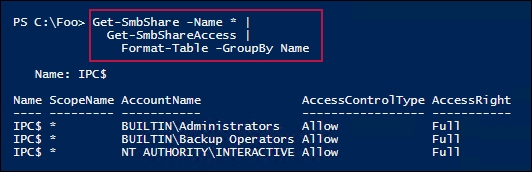
1. View the ACL file:

Get-NTFSAccess -Path C:\Foo\Foo.Txt |

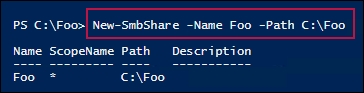
Format-Table -AutoSize

## How it works…

In step 1, you look at the existing shares and access rights, which looks like this:

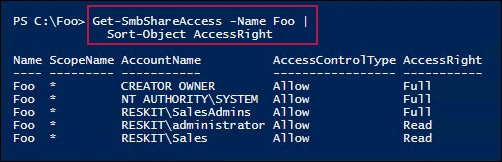


In step 2, you create a new SMB share (Foo) on the C:\Foo folder, which looks like this:



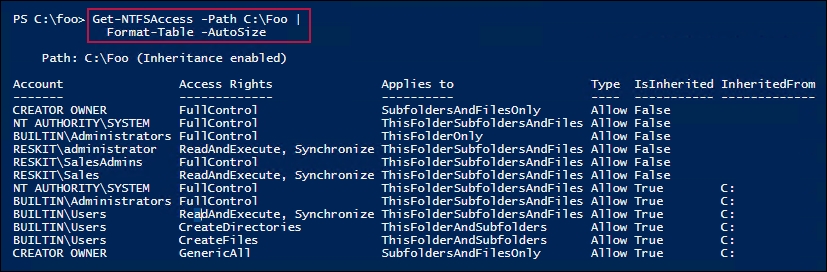
With step 3, you add a description to the share. In step 4, you set the share enumeration mode to AccessBased. In step 5, you set encryption for data sent to/from the Foo share. With step 6, you explicitly remove all access to the Foo share. In step 7, you enable Reskit\Administrator to have read-only access to the share. In step 8, you enable the OS to have full access to the share, while in step 9, you allow the creator or owner full access to files or folders in the share. In step 10, you grant all members of the Sales group read access to data on the share, and you grant members of the SalesAdmins group full access to the share. Step 3 through step 10 produce no output.

After configuring access to the share, in step 11, you use the Get-SMBShareAccess cmdlet to view the Foo share's ACL, which looks like this:

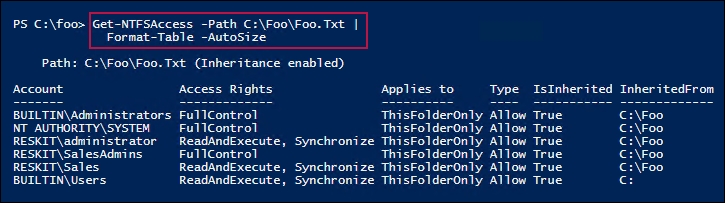


In step 12, you set the NTFS ACL file/folder to be same as the ACL for the share, which produces no output. With step 13, you create a new file in the C:\Foo folder. These two steps produce no output.

In step 14, you view the updated ACL on the C:\Foo folder itself, which looks like this:



Finally, in step 15, you view the ACL for the file you created in step 13, which looks like this:



## There's more...

In step 1, you examined the existing SMB shares. This step is run on the FS1 file server after you've hardened it (see the Creating and securing an SMB file server recipe). Thus, all the default shares (except the IPC$ share) aren't present on FS1.

The IPC$ share is also known as the null session connection. This session connection enables anonymous users to enumerate the names of domain accounts and network shares. The lanmanserver service creates this share by default, although you can turn it off. The IPC$ share is also used to support named pipe connections to your server.

In step 4, you set the enumeration mode on the Foo share to AccessBased. This means that when you're browsing folders and files within this share, you only see the objects you have access to. There is an improvement in security (as people can't see files, they have no access to), but this does introduce a small performance penalty.

In step 5, you set up this share to encrypt data sent to/from the share. This overrides the overall server configuration you set in the Setting up and securing an SMB file server recipe.

In step 14 and step 15, you examined the ACLs on the underlying folder and file after setting NTFS permissions on C:\Foo to be the same as for the Foo share. Since you didn't remove inheritance from the C:\Foo folder in this recipe, you can see that some users still have access (due to inheritance) to the files and files in the folder. To further secure this folder, you should remove NTFS inheritance from the C:\Foo folder.

## See Also

For details about IPC$ share, see <https://support.microsoft.com/help/3034016/ipc-share-and-null-session-behavior-in-windows>. Be careful if you chose to turn off the IPC$ share—test the resulting configuration very carefully.

# Accessing data on SMB shares

In the Creating and securing SMB shares recipe, you created a share on FS1. Files shared using SMB act and feel like local files when you access the share, for example, via Explorer.

In this recipe, you access the Foo share on FS1 from the CL1 Windows 10 system you created in [Chapter 1](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch01.html), Establishing a PowerShell Administrative Environment.

## Getting ready

You should have completed the Creating and securing SMB shares recipe. Additionally, you should have the CL1 Windows 10 system up and working—you created this system in [Chapter 1](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch01.html), Establishing a PowerShell Administrative Environment.

You should run this recipe in an elevated console.

## How to do it...

1. Examine the SMB client's configuration:

Get-SmbClientConfiguration

1. Set SMB signing from the client:

$CHT = @{Confirm=$false}

Set-SmbClientConfiguration -RequireSecuritySignature $True @CHT

1. Examine the SMB client's network interface:

Get-SmbClientNetworkInterface |

Format-Table Friendlyname, RSS\*, RD\*, Speed, IpAddresses

1. Examine the shares provided by FS1:

$FS1CS = New-CimSession -ComputerName FS1

Get-SmbShare -CimSession $FS1CS

1. Create a drive mapping, mapping R: to the share on the FS1 server:

New-SmbMapping -LocalPath R: -RemotePath \\FS1.Reskit.Org\Foo

1. View the shared folder mapping on CL1:

Get-SmbMapping

1. View the shared folder's contents:

Get-ChildItem -Path R:

1. View the existing connections:

Get-SmbConnection

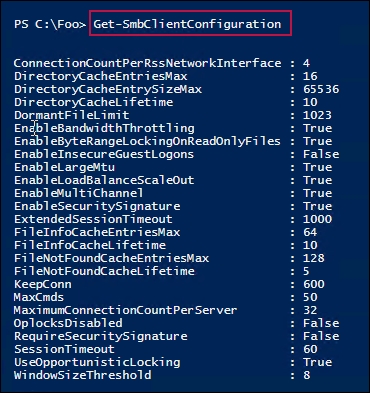
1. Show what files and folders are open on FS1:

Notepad R:\Foo.Txt # created in an earlier recipe

Get-SmbOpenFile -CimSession $FS1CS

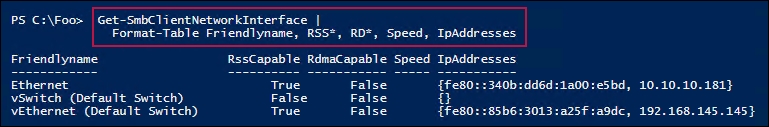
## How it works…

In step 1, you view the SMB client configuration, which looks like this:

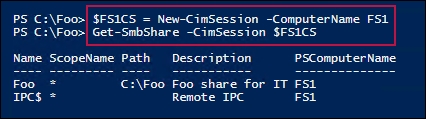


In step 2, you explicitly set the SBM to require the signing of SMB packets. This step creates no output.

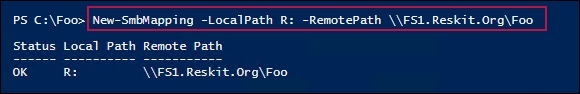
In step 3, you look at the SMB client's network interface details, which look something like this:



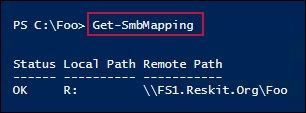
In step 4, you set up a CIM session to FS1, then use that session to determine the shares being offered by FS1, which looks like this:



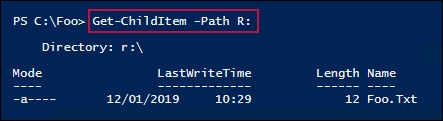
In step 5, you create a new client-side drive mapping, mapping the R: drive on CL1 to the \\FS1\Foo share. The output looks like this:



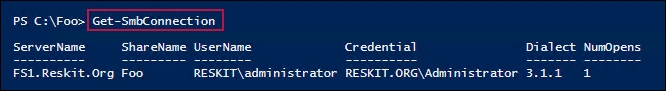
In step 6, you view the client-side drive mappings, which looks like this:



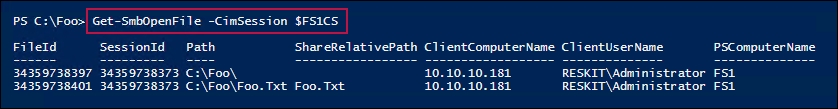
In step 7, you use the Get-ChildItem cmdlet to view the contents of the R: drive, and see what files exist on the Foo share of FS1. The output of this step looks like this:



In step 8, you open a file on R: in the Notepad drive and then use the CIM session (which you created in step 4) to view the open connections to FS1. The output looks like this:



In the final step, step 9, you view the files and folders that are open on FS1, which looks like this:



## There's more...

In step 4, you create a CIM session from CL1 to FS1, and check which shares are provided by FS1. There's no cmdlet equivalent of the net view <servername> command.

In step 7, since you have set permissions, a regular domain admin account has no access. It needs to be part of the Sales or Salesadmin groups.

In step 9, you used Get-SmbOpenFile to see the files open on FS1. As you can see, details of the open file and the computer making the connection are clearly shown. You get both a full path for each open file or folder, as well as a share-relative path. What the output shows is that R:\Foo.Txt from CL1 is C:\Foo\Foo.Txt on FS1.

# Creating an iSCSI target

iSCSI is an industry-standard protocol that implements block storage over a TCP/IP network. With iSCSI, the server, or initiator, provides a volume shared via iSCSI. Effectively, the shared volumes are iSCSI logical unit numbers. The iSCSI client then sees that disk as locally attached. From the iSCSI client, you can manage the disk just like locally-attached storage.

Windows Server 2019 includes both iSCSI target (server) and iSCSI initiator (client) features. You set up an iSCSI target on a server and then use an iSCSI initiator on a client system to access the iSCSI target. You can use both Microsoft and third-party initiators and targets, although if you mix and match, you need to test very carefully that the combination works in your environment.

With iSCSI, a target is a single disk that the client accesses using the iSCSI Client. An iSCSI target server hosts one or more targets, where each iSCSI target is equivalent to a LUN on a Fiber Channel SAN. The iSCSI initiator is a built-in component of Windows Server 2019 (and Windows 10). The iSCSI target feature is one you install optionally on Windows Server 2019.

You could use iSCSI in a cluster of Hyper-V servers. The servers in the cluster can use the iSCSI initiator to access an iSCSI target. Used via the Cluster Shared Volume, the shared iSCSI target is shared between nodes in a failover cluster that enables the VMs in that cluster to be highly available.

## Getting ready

In this recipe, you create an iSCSI target on the SRV1 server. Run this recipe from SRV1.

## How to do it...

1. Install the iSCSI target feature on SRV1:

Install-WindowsFeature FS-iSCSITarget-Server

1. Explore the iSCSI target server settings:

Get-IscsiTargetServerSetting

1. Create a folder on SRV1 to hold the iSCSI virtual disk:

$NIHT = @{

Path = 'C:\iSCSI'

ItemType = 'Directory'

ErrorAction = 'SilentlyContinue'

}

New-Item @NIHT | Out-Null

1. Create an iSCSI disk (that is, a LUN):

$LP = 'C:\iSCSI\SalesData.Vhdx'

$LN = 'SalesTarget'

$VDHT = @{

Path = $LP

Description = 'LUN For Sales'

SizeBytes = 100MB

}

New-IscsiVirtualDisk @VDHT

1. Create the iSCSI target:

$THT = @{

TargetName = $LN

InitiatorIds = 'DNSNAME:FS1.Reskit.Org'

}

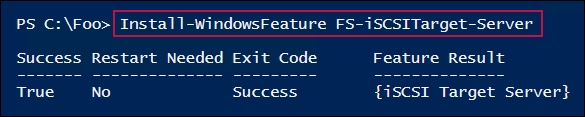
New-IscsiServerTarget @THT

1. Create the iSCSI disk target mapping:

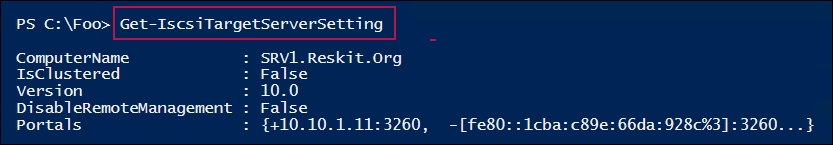
Add-IscsiVirtualDiskTargetMapping -TargetName $LN -Path $LP

## How it works…

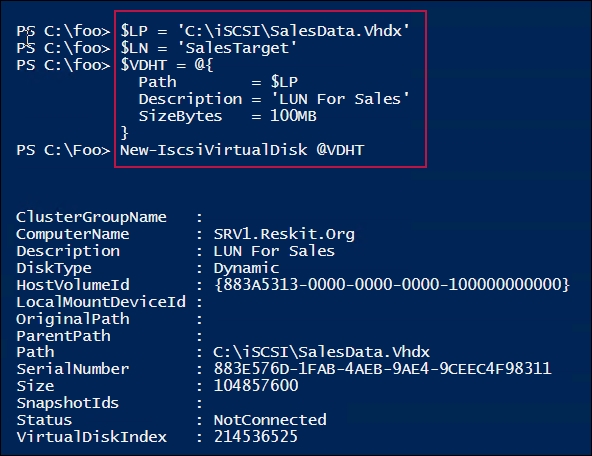
In step 1, you add the FS-iSCSITarget-Server feature to SRV1, which looks like this:



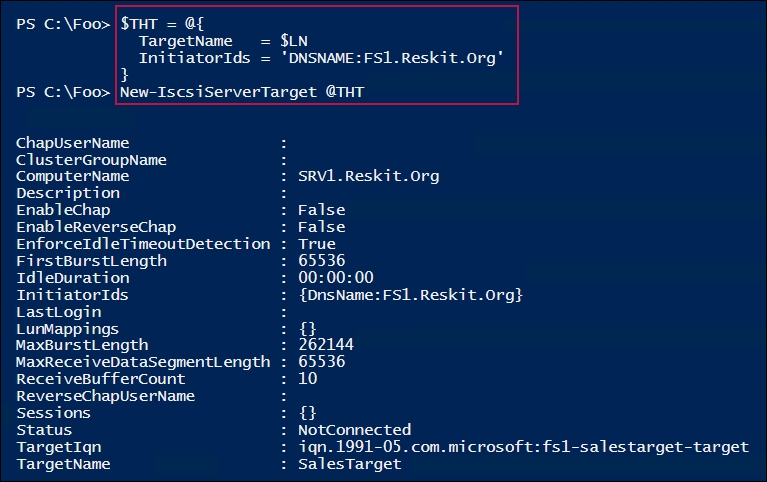
In step 2, you get the iSCSI target server settings for SRV1, which looks like this:



In step 3, which creates no output, you ensure you have a folder on SRV1 to hold the new iSCSI disk volume. In step 4, you create the iSCSI virtual disk, which looks like this:



In step 5, you create the iSCSI target on SRV1, which looks like this:



In step 6, you complete the creation of the iSCSI target by adding an iSCSI virtual disk target mapping.

## There's more...

In step 3, you create the new LUN, using New-IscsiVirtualDisk. When using this command, you must specify a VHDX file extension. Windows Server 2019 doesn't support VHD files for new iSCSI targets. If you have older VHD files you want to use as an iSCSI virtual disk, you can create a target that points to it. You just can't create new iSCSI virtual disks.

The virtual disk you created in step 4 is uninitialized and contains no filesystem. In order to use the iSCSI disk, you use the iSCSI initiator to mount and manage the drive as if it were local. You'll see this in the Using an iSCSI target recipe.

You can also increase security by using Challenge Handshake Authentication Protocol (CHAP) authentication. You can specify the CHAP username and password on both the initiator and the target to authenticate the connection to an iSCSI target. If the security of iSCSI traffic is an issue, you could consider securing iSCSI traffic using IPSec.

## See Also

If you aren't familiar with iSCSI and iSCSI targets, see <https://docs.microsoft.com/windows-server/storage/iscsi/iscsi-target-server> for an overview. And for more information on iSCSI, see <https://en.wikipedia.org/wiki/ISCSI>.

# Using an iSCSI target

Once you have an iSCSI target defined, as you did in the Creating an iSCSI target recipe, you can use it. Essentially, to use the disk, you connect to the iSCSI target server (that is, SRV1). Once you're connected, the Get-Disk cmdlet returns the iSCSI disk as though it were a local disk. You can then format and use the iSCSI disk as though it were local.

## Getting ready

This recipe uses the iSCSI target you created in the Creating an iSCSI target recipe. You use SRV1 as the iSCSI target and access the target from the iSCSI initiator (FS1).

## How to do it...

1. On FS1, set the iSCSI service to start automatically, then start the service:

Set-Service MSiSCSI -StartupType 'Automatic'

Start-Service MSiSCSI

1. Set up the portal to SRV1:

$PHT = @{

TargetPortalAddress = 'SRV1.Reskit.Org'

TargetPortalPortNumber = 3260

}

New-IscsiTargetPortal @PHT

1. Find and view the SalesTarget on the portal:

$Target = Get-IscsiTarget |

Where-Object NodeAddress -Match 'SalesTarget'

$Target

1. Connect to the target on SRV1:

$CHT = @{

TargetPortalAddress = 'SRV1.Reskit.Org'

NodeAddress = $Target.NodeAddress

}

Connect-IscsiTarget @CHT

1. View the iSCSI disk on SRV1 from FS1:

$ISD = Get-Disk |

Where-Object BusType -eq 'iscsi'

$ISD |

Format-Table -AutoSize

1. Turn the disk online and set it to read/write:

$ISD |

Set-Disk -IsOffline $False

$ISD |

Set-Disk -Isreadonly $False

1. Create a volume on the iSCSI disk on FS1:

$NVHT = @{

FriendlyName = 'SalesData'

FileSystem = 'NTFS'

DriveLetter = 'I'

}

$ISD |

New-Volume @NVHT

1. Use the drive as a local drive:

Set-Location -Path I:

New-Item -Path I:\ -Name SalesData -ItemType Directory |

Out-Null

'Testing 1-2-3' |

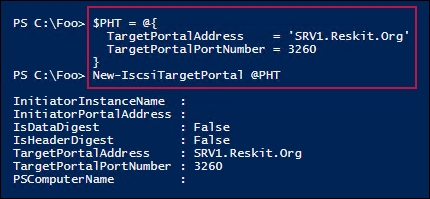
Out-File -FilePath I:\SalesData\Test.Txt

Get-ChildItem -Path I:\SalesData

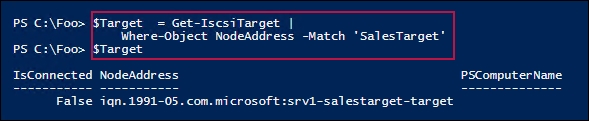
## How it works…

In step 1, you set the Microsoft iSCSI service to automatically start, then you start it. This step produces no output.

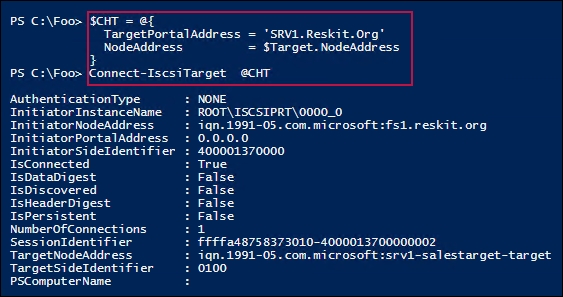
In step 2, you create a portal to the iSCSI server on SRV1, which looks like this:



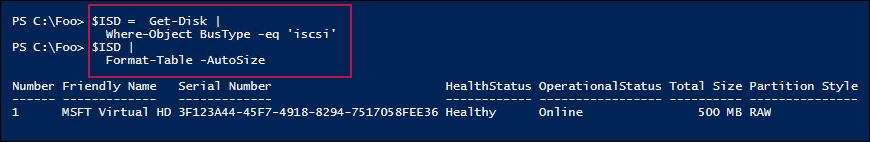
In step 3, you use the portal just created to get the SalesTarget iSCSI target on SRV1, which looks like this:



In step 4, you connect to the iSCSI target on SRV1, which looks like this:



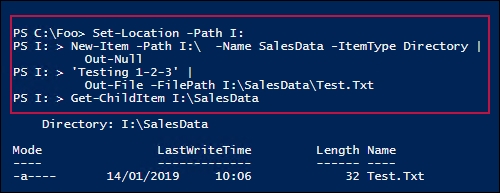
Once you've connected to the iSCSI target, you can view the disk drive provided via iSCSI, which looks like this:



In step 6, you turn the disk online and make it read/write. This step produces no output. In step 7, you create a new volume on this disk, which looks like this:



In the final step, step 8, you use the iSCSI disk just like it was a local one, which looks like this:



## There's more...

This recipe enabled you to use the Microsoft iSCSI initiator to connect to a Microsoft iSCSI-provided target. These built-in features work and are fine for simple use.

The iSCSI initiator and the iSCSI target features with Windows Server 2019 have seen little development or improvement since they were first released over a decade ago. You may find independent third-party iSCSI vendors that are more appropriate depending on your requirements.

# Configuring a DFS Namespace

The Distributed File System (DFS) is a set of services in Windows that enables you to create a structured replicated filestore on two or more servers within your organization. Microsoft first released DFS as an add-on to Windows NT 4.0. DFS has improved significantly since then.

In Windows Server 2019, DFS has two separate components. The first is DFS Namespace (DFSN). DFSN enables you to create a single contiguous namespace that refers to shares held on multiple servers. The second component, DFS Replication (DFSR), replicates data between DFS nodes in the DFS Namespace.

With DFS Namespaces, you can make use of shared folders stored on computers throughout the organization to create a single logically-structured namespace. This namespace appears to the user as a continuous and well-organized set of folders and subfolders, even though the actual shared data may be in a variety of independently-named shares on one or more computers in the organization.

Before you build your DFS Namespace, you need to create the shared folders that you wish to add to your DFS Namespace. The namespace design then determines which folder goes where within the namespace hierarchy. You also define the names of the folders in the namespace, and these can be different from the underlying file shares. When you view the DFS Namespace, the folders appear to reside on a single share that has multiple folders and subfolders. You navigate through the DFS Namespace and avoid needing to know the names of the actual servers and shares that physically hold the actual data.

It's important to note that using DFSN does not replicate any data between targets. Typically, you would use DFS Replication. If you need to replicate data, there are a variety of other tools available that may be more appropriate for your needs. For more information on file-synchronization tools, check out the following link: [https://en.wikipedia.org/wiki//Comparison\_of\_file\_synchronization\_software](https://en.wikipedia.org/wiki/Comparison_of_file_synchronization_software).

Both DFSN and DFSR have a supporting PowerShell module. The DFSN module helps you to manage the DFS Namespaces in your DFS implementation. You manage DFSR replication using the DFSR module.

In this recipe, you set up and configure a domain-based DFS Namespace on the SRV1 and SRV2 servers. You create additional DFS Namespace targets on other computers and add these to the DFS Namespace. In a later recipe, Configuring DFS Replication, you set up replication using DFSR.

## Getting ready

This recipe uses several systems: DC1, DC2, FS1, FS2, SRV1, and SRV2. Each of these systems hosts one or more shares that you create and use in this recipe as targets for the DFSN Namespace. You run this recipe from the CL1 Windows 10 (1809) system.

If your CL1 system is running an earlier edition of Windows 10, step 1 may not work for you, as the mechanism for adding RSAT tools has changed with 1809. If you're using Windows 10 1709 or 1803, check out <https://tfl09.blogspot.com/2018/10/installing-rsat-tools.html> for details on how to add the RSAT tools. If you're using any earlier version of Windows 10 for your testing, consider downloading an evaluation version ISO image of Windows Server 2019 and creating a VM using that ISO image.

In this recipe, you create a DFS Namespace, as set out in this table:

| Folder in DFSN Namespace | Target SMB share |
| --- | --- |
| \\Reskit.Org\ShareData\IT | n/a |
| \\Reskit.Org\ShareData\IT\ITData | \\FS1\ITData\  \\FS2\ITData\ |
| \\Reskit.Org\ShareData\IT\ITManagement | \\DC1\ITManagement  \\DC2\Mananagement |

## How to do it...

1. Add the DFSN RSAT Tools to CL1:

Get-WindowsCapability -Online -Name \*FileServices.Tools\* |

Add-WindowsCapability -Online |

Out-Null

1. Install DFS Namespace, DFS Replication, and the related management tools:

$IHT = @{

Name = 'FS-DFS-Namespace'

IncludeManagementTools = $True

}

Install-WindowsFeature @IHT -ComputerName SRV1

Install-WindowsFeature @IHT -ComputerName SRV2

1. View the DFSN module and the DFSN cmdlets:

Get-Module -Name DFSN -ListAvailable

1. Create folders and shares for DFS Root:

$SB = {

New-Item -Path C:\ShareData -ItemType Directory -Force |

Out-Null

$ACCESS = @{FullAccess = 'Everyone'}

New-SmbShare -Name ShareData -Path C:\ShareData @ACCESS

}

Invoke-Command -ComputerName SRV1, SRV2 -ScriptBlock $SB |

Out-Null

1. Create a DFS Namespace Root that points to \\SRV1\ShareData:

$NSHT = @{

Path = '\\Reskit.Org\ShareData'

TargetPath = '\\SRV1\ShareData'

Type = 'DomainV2'

Description = 'Reskit Shared Data DFS Root'

}

New-DfsnRoot @NSHT

1. Add a second target and view the results:

$NSHT2 = @{

Path = '\\Reskit.Org\ShareData'

TargetPath = '\\SRV2\ShareData'

}

New-DfsnRootTarget @NSHT2 | Out-Null

Get-DfsnRootTarget -Path \\Reskit.Org\ShareData

1. Create additional IT data shares and populate:

# Create FS1 folders/shares

$SB = {

# Create folder on FS1

New-Item -Path C:\IT2 -ItemType Directory | Out-Null

# Create share on FS1

New-SmbShare -Name 'ITData' -Path C:\IT2 -FullAccess Everyone

# Create a file in both the folder and therefore the share

'Root' | Out-File -FilePath C:\IT2\Root.Txt

}

Invoke-Command -ScriptBlock $SB -Computer FS1 | Out-Null

# Create FS2 folders/shares

$SB = {

New-Item -Path C:\IT2 -ItemType Directory | Out-Null

New-SmbShare -Name 'ITData' -Path C:\IT2 -FullAccess Everyone

'Root' | Out-File -FilePath c:\IT2\Root.Txt

}

Invoke-Command -ScriptBlock $SB -Computer FS2 | Out-Null

# Create DC1 folders/shares

$SB = {

New-Item -Path C:\ITM -ItemType Directory | Out-Null

New-SmbShare -Name 'ITM' -Path C:\ITM -FullAccess Everyone

'Root' | Out-File -Filepath c:\itm\root.txt

}

Invoke-Command -ScriptBlock $SB -Computer DC1 | Out-Null

# Create DC2 folders/shares

$SB = {

New-Item C:\ITM -ItemType Directory | Out-Null

New-SmbShare -Name 'ITM' -Path C:\ITM -FullAccess Everyone

'Root' | Out-File -FilePath c:\itm\root.txt

}

Invoke-Command -ScriptBlock $SB -Computer DC2

1. Create the DFS Namespace and set DFS targets:

$NSHT1 = @{

Path = '\\Reskit\ShareData\IT\ITData'

TargetPath = '\\FS1\ITData'

EnableTargetFailback = $True

Description = 'IT Data'

}

New-DfsnFolder @NSHT1 | Out-Null

$NSHT2 = @{

Path = '\\Reskit\ShareData\IT\ITData'

TargetPath = '\\FS2\ITData'

}

New-DfsnFolderTarget @NSHT2 | Out-Null

$NSHT3 = @{

Path = '\\Reskit\ShareData\IT\ITManagement'

TargetPath = '\\DC1\ITM'

EnableTargetFailback = $true

Description = 'IT Management Data'

}

New-DfsnFolder @NSHT3 | Out-Null

$NSHT4 = @{

Path = '\\Reskit\ShareData\IT\ITManagement'

TargetPath = '\\DC2\ITM'

}

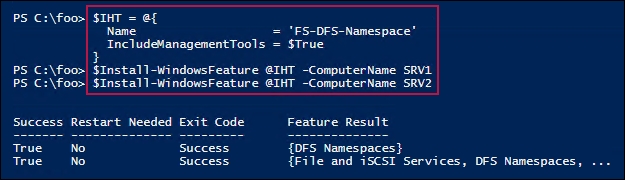
New-DfsnFolderTarget @NSHT4 | Out-Null

1. View the hierarchy:

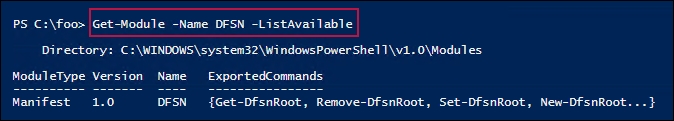
Get-ChildItem -Path \\Reskit.Org\ShareData\IT -Recurse

## How it works…

In step 1, you add the RSAT tools needed for DFSN (and DFS Replication), which creates no output. In step 2, you install the DFS Namespace feature to both SRV1 and SRV2, which looks like this:

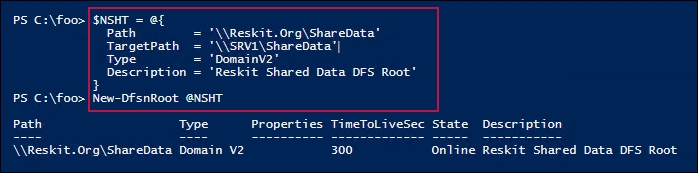


In step 3, you view the DFSN module, containing the key DFSN cmdlets, which looks like this:

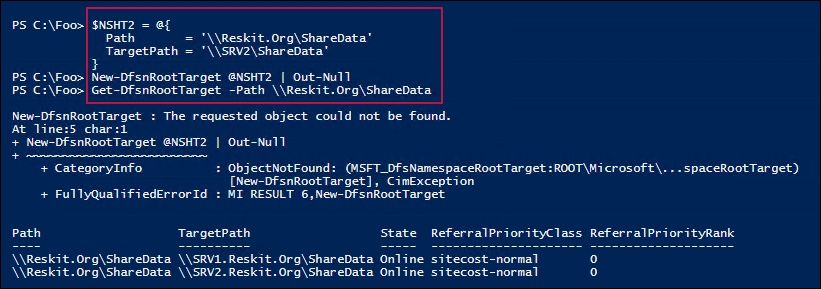


In step 4, you create the folders and shares needed to support the DFSN you're about to create. This step creates no output.

In step 5, you create the DFSN root that points to \\SRV1\ShareData. The output from this step looks like this:

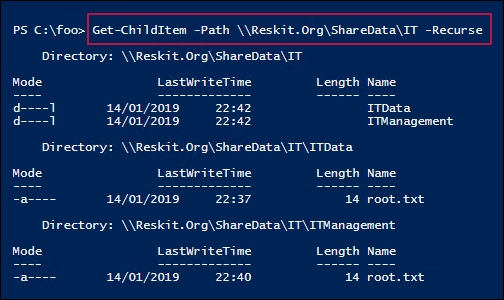


In step 6, you create a second target for the DFS root. Then you get the details of the DFSN root target. The output from this step looks like this:



In step 7, you create several additional shares and add data to the shares. In step 8, you create DFS Namespace targets for the newly-created shares. These two steps produce no output.

In step 9, you view the folders/files under the newly-created DFSN root, which look like this:



## There's more...

In step 1, you add the RSAT tools needed for DFSN (and DFS Replication). The methods for adding RSAT tools on Windows 10 have evolved. The technique used in the recipe is valid for Windows 10, version 1809 or later. If you're using an earlier version (such as 1803 or 1709), see <https://tfl09.blogspot.com/2018/10/installing-rsat-tools.html>.

In step 6, you create a second target. As you can see from the screenshot, this step creates the second DFSN root target, but it generates a CIM Exception. However, this error appears benign and, despite the error, the step does work successfully. This is an issue with the cmdlet, as reported here: <https://github.com/MicrosoftDocs/windows-powershell-docs/issues/541>.

# Configuring DFS Replication

DFSR is an efficient file-replication engine built into Windows Server 2019. You can use DFS Replication to replicate DFSN targets in an efficient manner, especially across low-bandwidth connections.

In DFSR, a replication group is a collection of computers, known as members. Each replication group member hosts replicated folders. Replicated folders are folders that DFSR ensures are synchronized. With DFS Replication groups, you can replicate the folders contained in your DFS Namespace.

A DFS replicated folder is a folder that DFSR keeps synchronized on each member. In the Configuring a DFS Namespace recipe, you created some folders that you need to replicate between each server. As the data changes in each replicated folder, DFSR replicates the changes across connections between the members of the replication group. The connections you set up between the members forms the replication topology.

Creating multiple replicated folders in a single replication group simplifies the process of deploying replicated folders because DFSR applies the topology, schedule, and bandwidth-throttling from the replication group to each replicated folder. Each replicated folder has many properties. These include file and subfolder filters that enable you to filter out different files and subfolders from each replicated folder.

In this recipe, you set up replication of the DFSN shared folders, created in the Configuring a DFS Namespace recipe. Then you'll test the replication in action.

## Getting ready

This recipe uses the same systems you used in the Configuring DFS Namespace recipe and assumes you have completed that recipe successfully. It sets up DFS Replication on the folders you created in the earlier recipe. You run this recipe on the Windows 10 system, CL1, that you set up in [Chapter 1](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20STY\9781789808537\OEBPS\ch01.html), Establishing a PowerShell Administrative Environment.

## How to do it...

1. Install the DFS-Replication feature on the key servers:

$SB = {

$IHT = @{

Name ='FS-DFS-Replication'

IncludeManagementTools = $true

}

Add-WindowsFeature @IHT

}

$ICHT = @{

ScriptBlock = $SB

ComputerName = 'DC1', 'DC2', 'FS1', 'FS2', 'SRV1', 'SRV2'

}

Invoke-Command @ICHT |

Format-Table -Property PSComputername,FeatureResult, Success

1. Turn on the administrative shares:

$SB2 = {

$SCHT = @{

AutoShareServer = $true

AutoShareWorkstation = $true

Confirm = $false

}

Set-SmbServerConfiguration @SCHT

"Restarting LanmanServer on $(hostname)"

Stop-Service -Name LanManServer -Force

Start-Service -Name LanManServer

}

$CN = @('DC1','DC2','FS1','FS2','SRV1','SRV2')

Invoke-Command -ScriptBlock $SB2 -ComputerName $CN

1. View the DFS cmdlets:

Get-Module -Name DFSR -ListAvailable

Get-Command -Module DFSR | Measure-Object

1. Create replication groups:

$RGHT1 = @{

GroupName = 'FSShareRG'

DomainName = 'Reskit.org'

Description = 'Replication Group for FS1, FS2 shares'

}

$RGHT2 = @{

GroupName = 'DCShareRG'

DomainName = 'Reskit.Org'

Description = 'Replication Group for DC1, DC2 shares'

}

New-DfsReplicationGroup @RGHT1 | Out-Null

New-DfsReplicationGroup @RGHT2 | Out-Null

1. Get replication groups in Reskit.Org:

Get-DfsReplicationGroup -DomainName Reskit.Org |

Format-Table

1. Add replication group members to FSShareRG:

$MHT1 = @{

GroupName = 'FSShareRG'

Description = 'ITData on FS1/2'

ComputerName = ('FS1','FS2')

DomainName = 'Reskit.Org'

}

Add-DfsrMember @MHT1

1. Add the DFSN folder to the FSShareRG replication group, thus replicating the \ITData share:

$RFHT1 = @{

GroupName = 'FSShareRG'

FolderName = 'ITData'

Domain = 'Reskit.Org'

Description = 'ITData on FS1/2'

DfsnPath = '\\Reskit.Org\ShareData\IT\ITData'

}

New-DfsReplicatedFolder @RFHT1 | Out-Null

1. Add replication group members to DCShareRG:

$MHT2 = @{

GroupName = 'DCShareRG'

Description = 'DC Server members'

ComputerName = ('DC1','DC2')

DomainName = 'Reskit.Org'

}

Add-DfsrMember @MHT2 |

Out-Null

1. Add DFSN folders to the DCShareRG replication group:

$RFHT2 = @{

GroupName = 'DCShareRG'

FolderName = 'ITManagement'

Domain = 'Reskit.Org'

Description = 'IT Management Data'

DfsnPath = '\\Reskit.Org\ShareData\IT\ITManagement'

}

New-DfsReplicatedFolder @RFHT2 |

Out-Null

1. View the replicated folders:

Get-DfsReplicatedFolder |

Format-Table -Property DomainName, GroupName,

FolderName, Description

1. Set the membership for the FSShareRG replication group:

$DMHT1 = @{

GroupName = 'FSShareRG'

FolderName = 'ITData'

ComputerName = 'FS1'

ContentPath = 'C:\IT2'

PrimaryMember = $true

Force = $true

}

Set-DfsrMembership @DMHT1 | Out-Null

$DMHT2 = @{

GroupName = 'FSShareRG'

FolderName = 'ITData'

ComputerName = 'FS2'

ContentPath = 'C:\IT2'

PrimaryMember = $false

Force = $true

}

Set-DfsrMembership @DMHT2 | Out-Null

1. Set the membership for the DCShareRG replication group:

$DMHT3 = @{

GroupName = 'DCShareRG'

FolderName = 'ITManagement'

ComputerName = 'DC1'

ContentPath = 'C:\ITM'

PrimaryMember = $true

Force = $true

}

Set-DfsrMembership @DMHT3 | Out-Null

$DMHT4 = @{

GroupName = 'DCShareRG'

FolderName = 'ITManagement'

ComputerName = 'DC2'

ContentPath = 'C:\ITM'

Force = $true

}

Set-DfsrMembership @DMHT4 | Out-Null

1. View the DFSR membership of the two replication groups:

Get-DfsrMembership -GroupName FSShareRG -ComputerName FS1, FS2 |

Format-Table -Property GroupName, ComputerName,

ComputerDomainName, ContentPath,

Enabled

Get-DfsrMembership -GroupName DCShareRG -ComputerName DC1, DC2 |

Format-Table -Property GroupName, ComputerName,

ComputerDomainName, ContentPath,

Enabled

1. Add replication connections for both replication groups:

$RCHT1 = @{

GroupName = 'FSShareRG'

SourceComputerName = 'FS1'

DestinationComputerName = 'FS2'

Description = 'FS1-FS2 connection'

DomainName = 'Reskit.Org'

}

Add-DfsrConnection @RCHT1| Out-Null

$RCHT2 = @{

GroupName = 'DCShareRG'

SourceComputerName = 'DC1'

DestinationComputerName = 'DC2'

Description = 'DC1-DC2 connection'

DomainName = 'Reskit.Org'}

Add-DfsrConnection @RCHT2 | Out-Null

1. Get the DFSR Membership and view it:

Get-DfsrMember |

Format-Table -Property Groupname, DomainName,

DNSName, Description

1. Update the DFSR configuration:

Update-DfsrConfigurationFromAD -ComputerName DC1, DC2, FS1, FS2

1. Check the existing folders to discover what's currently in the DFS share:

$Path = '\\Reskit.Org\ShareData\IT\ITManagement'

$Path1 = '\\DC1\itm'

$Path2 = '\\DC2\itm'

Get-ChiLditem -Path $Path

Get-ChiLditem -Path $Path1

Get-ChildItem -Path $Path2

1. Create files in the DFS share and re-check the underlying shares:

1..100 | foreach { "foo" |

Out-File \\Reskit.Org\ShareData\IT\ITManagement\Stuff$\_.txt}

$P = (Get-ChildItem -Path $Path | Measure-Object).count

$P1 = (Get-ChildItem -Path $Path1 | Measure-Object).count

$P2 = (Get-ChildItem -Path $Path2 | Measure-Object).count

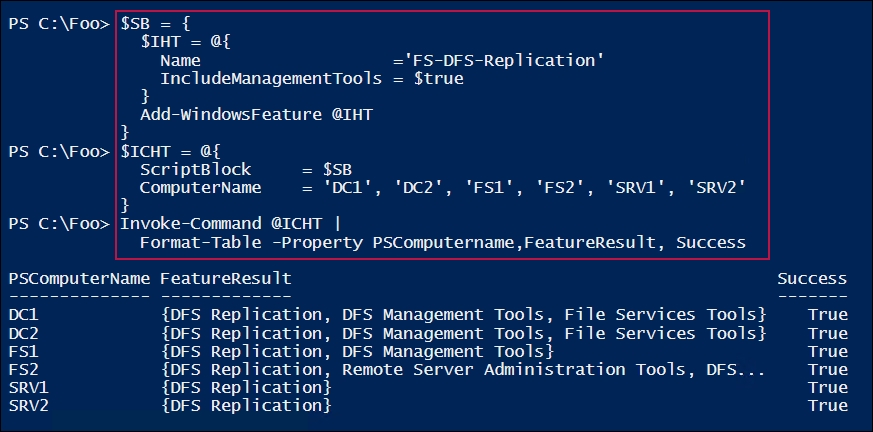
"$P objects in DFS root"

"$P1 objects on \\DC1"

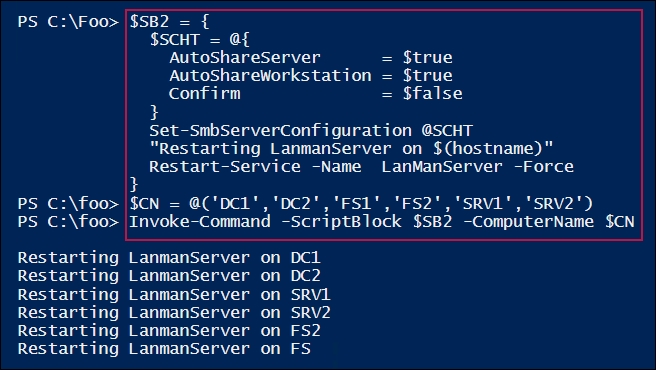
"$P2 objects on \\DC2"

## How it works…

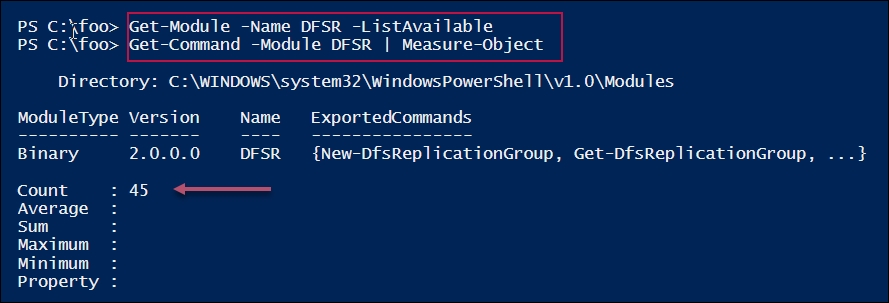
In step 1, you install the DFS Replication feature on several servers, which looks like this:



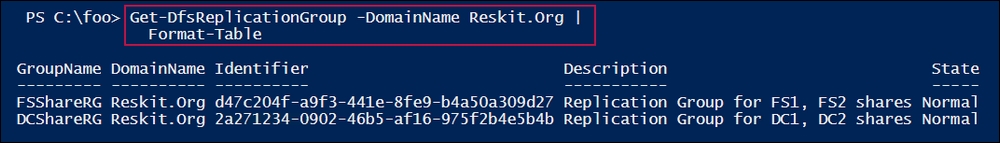
In step 2, after installing the DFSR feature, you configure Windows to update the SMB server configuration and then restart the target servers, which looks like this:



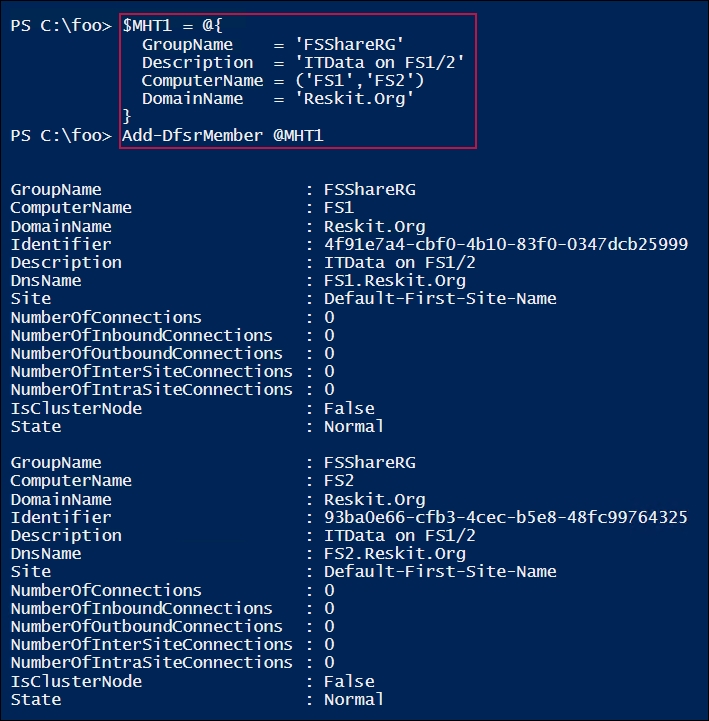
In step 3, you view the DFS module and see how many cmdlets are provided by the module. The output looks like this:



In step 4, which produces no output, you define the replication groups (FSShareRG and DCShareRG). In step 5, you view the domain-based DFSR replication groups, which look like this:



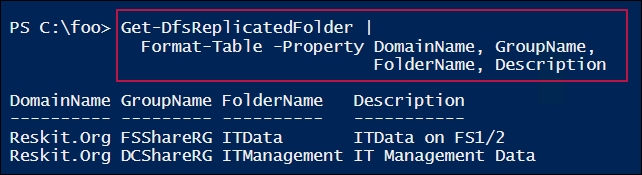
In step 6, you populate the FSShareRG replication group, which looks like this:



In step 7, you add the DFSN folder to the replication group, which produces no output.

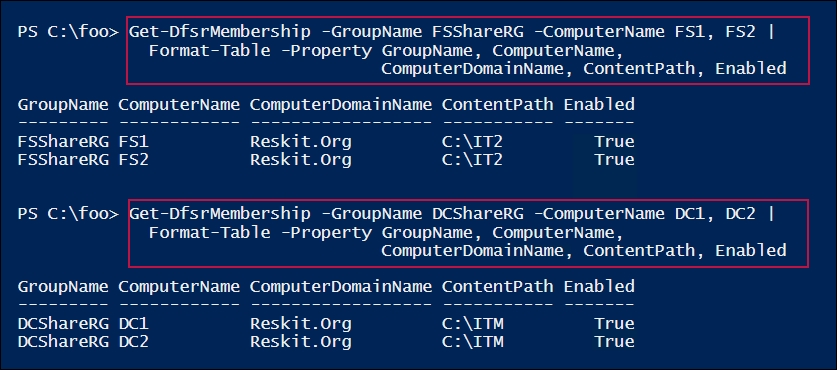
In step 8, you add the Add replication group members for DCShareRG and in step 9, you add the DFSN folders to the DCShareRG replication group. These two steps produce no output.

In step 10, you view the replicated folders in the Reskit.Org domain, which looks like this:

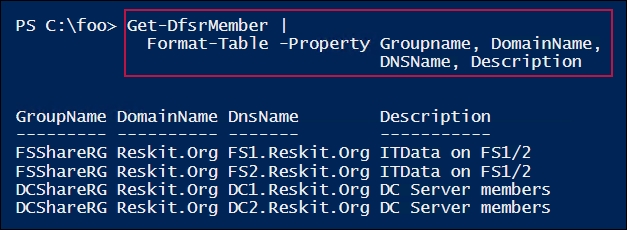


In step 11, you set the membership for the FSShareRG replication group, and in step 12, you set the membership for the DCShareRG replication group—these two steps produce no output.

In step 13, you view the DFSR membership of the two replication groups, which looks like this:

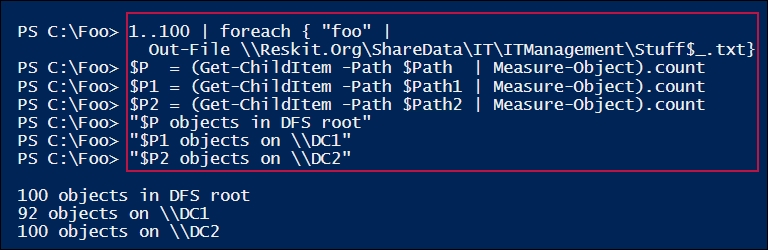


In step 14, you add connections for both replication groups, which produces no output. In step 15, you get the DFSR memberships of the groups in Reskit.Org and format the key properties, which looks like this:



In step 16, you force an update to the DFSR configuration, which produces no output. In step 17, you check to see that there are no files in the DFS share—this step produces no output.

In step 18, you test DFS Replication by creating a number of files on the DFS share (\\Reskit.Org\ShareData) and observe the number of files present in the underlying shares. The output, which is likely to vary, looks like this:



## There's more...

In step 2, you turn on the administrative shares. This is a requirement for setting up DFS Replication. Once you've set up replication, you may wish to turn off these administrative shares.

In step 8, for each shared folder in the FSShareRG replication group, you identify a primary member. Should a document be changed by multiple different members in the replication group, then DFS considers the copy on the primary master as definitive.

In step 11, you set up simple DFS Replication connections. DFS enables you to manage rich replication topologies and supports your configuring-replication schedules and bandwidth constraints.

For a comparison of DFS Replication and Storage Replica, see <https://www.petri.com/windows-server-2016-dfs-r-vs-storage-replica>.

In step 17, you check to see what files exist in the DFS share (that is, none). Then in step 18, you test the DFSR replication by creating 100 new files on the DFSN share, and then observe how many files exist in the two underlying shares. As you can see from the output, 100 files exist both on the DFS share and on DC1, but at the time of execution, only 92 files had replicated. If you'd waited a few more seconds after creating the files, you would have observed complete replication.

## See also….

The recipe sets up a simple set of replicated folders—four replicated folders on four servers based on the DFS Namespace created earlier. To extend this recipe, you could add other folders to the replication groups that weren't part of the DFS Namespace.

DFS Replication is one way to replicate files in an organization. DFS was designed for use over lower-bandwidth networks, thus in larger networks, DFS replicas might be out of sync. Also, DFS only replicates a file after it has been closed. With Server 2019, the Storage Replica feature is an alternative to DFSR. SR works at the block level, unlike DFSR, which operates at the file level. As a result, SR can replicate the changes to open files.