7

Managing Storage

This chapter covers the following recipes:

* Managing Disks
* Managing File Systems
* Exploring PowerShell Providers and the File System provider
* Managing Storage Replica
* Deploying Storage Spaces

# Introduction

Windows Server 2022 provides a range of features that allow access to various storage and storage devices. Windows supports spinning disks, USB memory sticks, and SSD devices (including MVMe SSD devices). These storage options provide you with great flexibility.

Before using a storage device to hold files, you need to create partitions or volumes on the device and then format these drives/volumes. Before formatting, you need to initialize a disk and define which partitioning method to use. You have two choices:

* Master Boot Record (MBR)
* GUID Partition Table (GPT)

These days, most PCs use the GUID Partition Table (GPT) disk type for hard drives and SSDs. GPT is more robust and allows for volumes bigger than 2 TB. The older MBR disk type is used typically by older PC and removable drives such as memory cards or external disk drives.

For a good discussion of the differences between these two mechanisms, see https://www.howtogeek.com/193669/whats-the-difference-between-gptand-mbr-when-partitioning-a-drive/.

Once you create a volume on a storage device, you can format the volume. Windows supports five filesystems you can use: ReFS, NTFS, exFAT, UDF, and FAT32. For details of the latter four, see https://learn.microsoft.com/windows/desktop/fileio/filesystemfunctionality-comparison. The ReFS filesystem is more recent and is based on NFTS but lacks some features your file server might need (for example, ReFS does not support encrypted files that you may wish to support. A benefit of the ReFS file system is the automatic integrity checking. For a comparison between the ReFS and NTFS filesystems, see https://www.iperiusbackup.net/en/refs-vs-ntfs-differences-and-performance-comparison-when-to-use/. You examine partitioning and formatting volumes in the Managing physical disks and volumes recipe.

NTFS (and ReFS) volumes allow you to create access control lists (ACLs) that control access to files and folders stored in Windows volumes. Each ACL has one or more Access Control Entries (ACE). Each ACE in an ACL defines a specific permission for some account (for example, setting Read only for members of the DNS Admins group). In general, you want ACLs to have as few ACEs as possible. Longer ACLs can be challenging to keep up to date, represent a potential security issue, and impact performance.

Managing ACLs with PowerShell is somewhat tricky. PowerShell lacks rich support for managing ACLs and ACL inheritance, although .NET does provide the necessary classes you need to manage ACLs. To simplify the management of ACLs on NTFS volumes, as you see in the Managing NTFS permissions recipe in Chapter 8, you can download and use a third-party module, NTFSSecurity.

Storage Replica is a feature of Windows Server that replicates any volume to a remote system. Storage Replica is only available with the Windows Server Data Center edition. In Managing Storage Replica, you create a replication partnership between two hosts and enable Windows Server to keep the replica up to date.

Storage Spaces is a technology provided by Microsoft in the Windows Client (Windows 10 and Windows 11) and recent versions of Windows Server that can help you protect against a disk drive’s failure. Storage Spaces provides software RAID, which you investigate in Deploying Storage Spaces.

## The systems used in the chapter

In this chapter, you use two servers: SRV1 and SRV2. Both are running Windows Server 2022 Data Center edition. The servers are member servers in the Reskit.Org domain served by the two Domain Controllers: DC1 and DC2, as shown here:



Figure 7.1: Hosts in use for this chapter

Insert image B18878\_07\_01.png

# Managing Disks

Windows Server 2022 requires a computer with at least one storage drive (in most cases, this is your C:\ drive). You can connect a storage device using different bus types, such as IDE, SATA, SAS, and USB. Before you can utilize a storage device in Windows, you need to initialize it and create a partitioning scheme.

You can use two partitioning schemes: the older format of Master Boot Record (MBR) and the more recent GUID Partition Table (GPT). The MBR scheme, first introduced with the PC DOS 2 in 1983, had some restrictions. For example, the largest partition supported with MBR is 2 TB. And to create more than four primary partitions, you would need an extended partition and then create additional partitions inside the extended partition. For larger disk devices, this can be inefficient

The GPT scheme enabled much larger drives (OS-imposed partition limits) and up to 128 partitions per drive. You typically use GPT partitioning with Windows Server. If you built the VMs for the servers you use to test the recipes, each VM has a single GPT partitioned (virtual) disk.

In this chapter, you use eight new virtual disk devices in the server, SRV1, and examine the disks and the partitions/volumes on SRV1.

## Getting ready

This recipe uses SRV1, a domain-joined host in the Reskit.Org domain, on which you have installed PowerShell 7 and VS Code. You also use SRV2 and should have DC1 online as well.

Before this chapter, you configured SRV1 with a single boot/system drive (the C: drive). The recipes in this chapter use eight additional disks. You must add these disks to the SRV1 host before running this recipe.

Assuming you are using Hyper-V to create your VMs, you can run the following script on your Hyper-V host to add the new disks to the SRV1 and SRV2 VMs.

# 0. Add new disks to the SRV1, SRV2 VMs

# Run this step on the VM host

# Assumes a single C: on SCSI Bus 0

# This step creates a new SCSI controller to hold the new disks

#

# 0.1 Turning off the VMs

Get-VM -Name SRV1, SRV2 | Stop-VM -Force

# 0.2 Getting Path for hard disks for SRV1, SRV2

$Path1   = Get-VMHardDiskDrive -VMName SRV1

$Path2   = Get-VMHardDiskDrive -VMName SRV2

$VMPath1 = Split-Path -Parent $Path1.Path

$VMPath2 = Split-Path -Parent $Path2.Path

# 0.3 Creating 8 virtual disks on VM host

0..7 | ForEach-Object {

  New-VHD -Path $VMPath1\SRV1-D$\_.vhdx -SizeBytes 64gb -Dynamic |

    Out-Null

  New-VHD -Path $VMPath2\SRV2-D$\_.vhdx -SizeBytes 64gb -Dynamic |

    Out-Null

}

# 0.4 Adding disks to SRV1, SRV2

# Create the next SCSI controller on SRV1/SRV2

Add-VMScsiController -VMName SRV1

[int] $SRV1Controller =

  Get-VMScsiController -VMName SRV1 |

    Select-Object -Last 1 |

      Select-Object -ExpandProperty ControllerNumber

Add-VMScsiController -VMName SRV2

[int] $SRV2Controller =

  Get-VMScsiController -VMName SRV1 |

    Select-Object -Last 1 |

       Select-Object -ExpandProperty ControllerNumber

# Now add the disks to each VM

0..7 | ForEach-Object {

  $DHT1 = @{

    VMName           = 'SRV1'

    Path             = "$VMPath1\SRV1-D$\_.vhdx"

    ControllerType   = 'SCSI'

    ControllerNumber = $SRV1Controller

  }

  $DHT2 = @{

    VMName           = 'SRV2'

    Path             =  "$VMPath2\SRV2-D$\_.vhdx"

    ControllerType   = 'SCSI'

    ControllerNumber =  $SRV2Controller

  }

  Add-VMHardDiskDrive @DHT1

  Add-VMHardDiskDrive @DHT2

}

# 0.5 Checking VM disks for SRV1, SRV2

Get-VMHardDiskDrive -VMName SRV1 | Format-Table

Get-VMHardDiskDrive -VMName SRV2 | Format-Table

# 0.6 Restarting VMs

Start-VM -VMName SRV1

Start-VM -VMName SRV2

If you download the scripts for this book from GitHub, the script for the recipe contains this preparation step.

Once you have created the eight new disks for the two VMs, you can begin the recipe on SRV1.

## How to do it...

1. Displaying the disks on SRV1

Get-Disk

1. Get the first usable disk

$Disk = Get-Disk |

           Where-Object PartitionStyle -eq Raw |

             Select-Object -First 1

$Disk | Format-List

1. Initializing the first available disk

$Disk |

  Initialize-Disk -PartitionStyle GPT

1. Re-displaying all disks in SRV1

Get-Disk

1. Viewing volumes on SRV1

Get-Volume | Sort-Object -Property DriveLetter

1. Viewing partitions on SRV1

Get-Partition

1. Examining details of a volume

Get-Volume | Select-Object -First 1 | Format-List

1. Examining details of a partition

Get-Partition | Select-Object -First 1 | Format-List

1. Formatting and initializing the second disk as MBR

$Disk2 = Get-Disk |

           Where-Object PartitionStyle -eq Raw |

             Select-Object -First 1

$Disk2 |

  Initialize-Disk -PartitionStyle MBR

1. Examining disks in SRV1

Get-Disk

## How it works...

In step 1, you use the Get-Disk command to view the disks available in SRV1, with output like this:

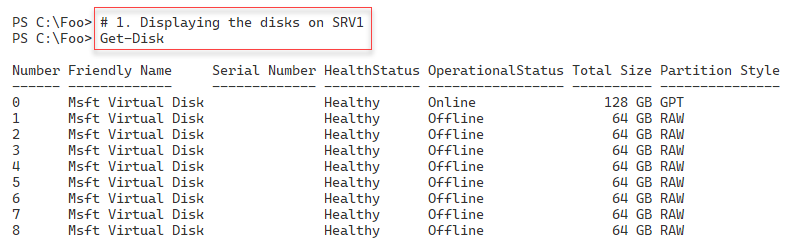


Figure 7.2: Displaying disk details

Insert image B18878\_07\_02.png

In step 2, you the first usable disks and examine the details of the disk, with output like this:



Figure 7.3: Displaying disk details of the first usable disk

Insert image B18878\_07\_03.png

In step 3, you initialize this disk using the GUID Partition Table (GPT) partition scheme. This step produces no output.

In step 4, you view the disks again on SRV1, with output like this;

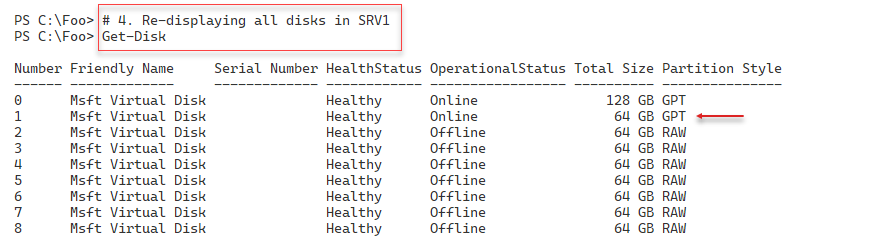


Figure 7.4: Viewing existing disk volumes on SRV1

Insert image B18878\_07\_04.png

In step 5, you use the Get-Volume command to view the partitions on SRV1, with the following output:

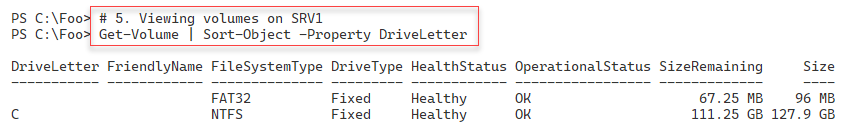


Figure 7.5: Viewing existing volumes on SRV1

Insert image B18878\_07\_05.png

In step 6, you use the Get-Partition command to view all the partitions on SRV1, which creates the following output:

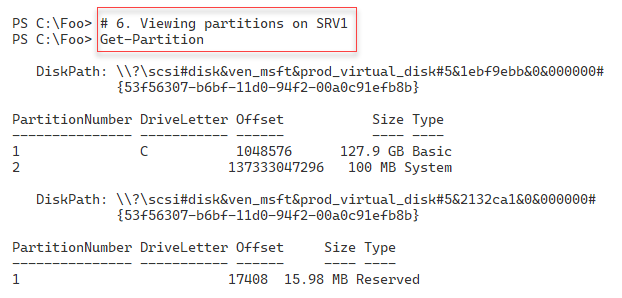


Figure 7.6: Viewing all partitions on SRV1

Insert image B18878\_07\_06.png

In step 7, you examine the properties of a disk, with output like this:

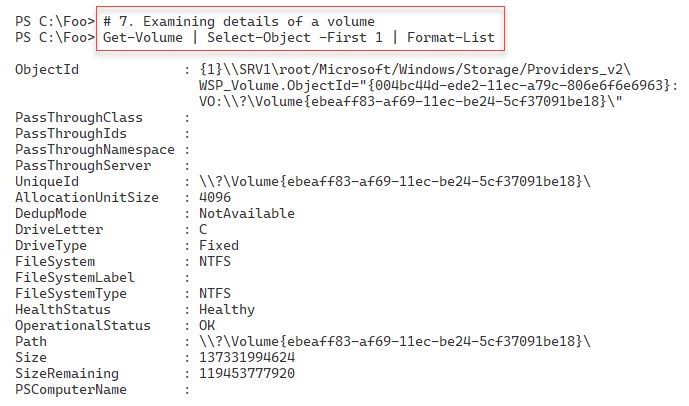


Figure 7.7: Viewing Volume properties

Insert image B18878\_07\_07.png

In step 8, you view the properties of a partition with output like this:

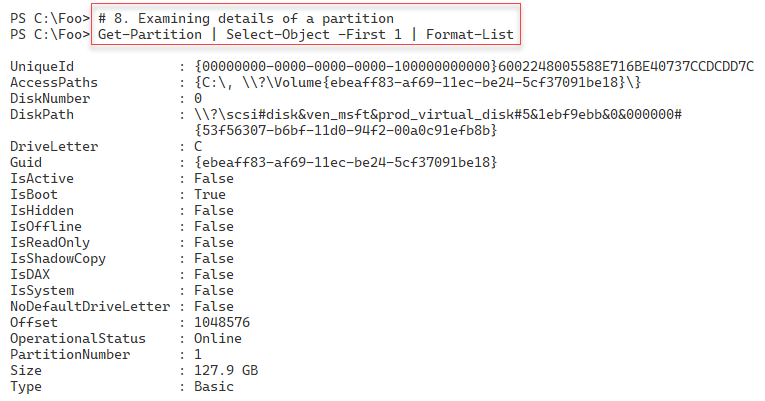


Figure 7.8: Viewing partition details

Insert image B18878\_07\_08.png

In step 9, you get the next unpartitioned disk and initialize it using the Master Boot Record (MBR) formatting scheme. This step creates no output.

In step 10, you again view the disks available on SRV1, including the two disks you just partitioned. The output looks like this:

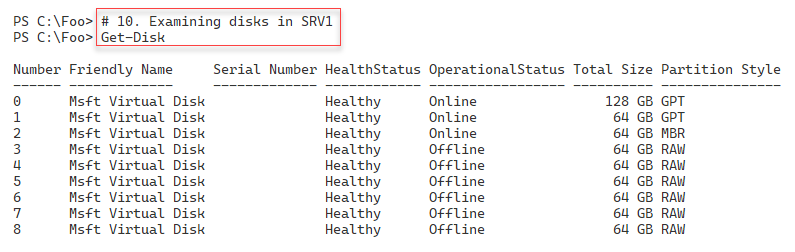


Figure 7.9: Viewing disks on SRV1

Insert image B18878\_07\_09.png

## There's more...

In step 1, you view all the disks in SRV1. The first disk, Disk 0, is your C: drive. You created this as part of the process of installing Windows Server 2022. In step 2, you get the first of the eight disk devices you added to SRV1 at the start of this recipe. However, before you can use this disk in Windows, you need to initialize the disk, which you do in step 3. When you initialize a disk, you specify the partitioning scheme, which in this case is GPT. Initializing the disk does not create any partitions or volumes. You can see, in Step 4, that the disk has a partition scheme.

In steps 5 and 6, you verify that there is only one volume or partition on SRV1 (i.e., the C: drive on disk 0).

The terms “partition” and “volume” in Windows are, in effect, the same thing. But due to history, you can use two different commands to create and manage volumes and partitions., In step 7, you review the properties of a volume, while in step 8, you examine the properties of a Windows disk partition.

In most cases, you initialize storage devices in your system using GPT, a more robust and flexible scheme. But you may wish to use the older MBR partition scheme, for example, for removable storage devices you intend to use on hosts that do not support GPT.

# Managing File Systems

To use a storage device, whether a spinning disk, CD/DVD device, or a solid-state device, you must format that device/drive with a file system. You must also have initialized the disk with a partitioning scheme, as you saw in the “Managing Disks” recipe

In most cases, you use NTFS as the file system of choice. It is robust and reliable and provides efficient access control. NTFS also provides file encryption and compression. An alternative is the ReFS file system. This file system might be a good choice for some specialized workloads. For example, you might use the ReFS file system on a Hyper-V host to hold VM virtual hard drives. Additionally, for interoperability with devices like video and still cameras, you might need to use the FAT, FAT32, or exFAT file system.

For more details on the difference between NTFS, FAT, FAT32, and ExFAT file systems, see https://medium.com/hetman-software/the-difference-between-ntfs-fat-fat32-and-exfat-file-systems-ec5172c60ccd. And for more information about the ReFS file system, seehttps://learn.microsoft.com/windows-server/storage/refs/refs-overview .

## Getting ready

This recipe uses SRV1, a domain-joined host in the Reskit.Org domain, on which you have installed PowerShell 7 and VS Code. You also have DC1 online. In the Managing Disks recipe, you added eight virtual disks to the SRV1. Then, in the Managing Disks recipe, you initialized the first two. In this recipe, you create specific volumes on these two disks.

## How to do it...

1. Getting the second disk

$Disk = Get-Disk | Select-Object -Skip 1 -First 1

$Disk | Format-List

1. Creating a new volume in this disk

$NewVolumeHT1   = @{

  DiskNumber    = $Disk.Disknumber

  DriveLetter  = 'S'

  FriendlyName = 'Files'

}

New-Volume @NewVolumeHT1

1. Getting the next available disk to use on SRV1

$Disk2 = Get-Disk |

           Where-Object PartitionStyle -eq 'MBR' |

             Select-Object -First 1

$Disk2 | Format-List

1. Creating 4 new partitions on the third (MBR) disk

$UseMaxHT= @{UseMaximumSize = $true}

New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter W -Size 1gb

New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter X -Size 15gb

New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter Y -Size 15gb

New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter Z @UseMaxHT

1. Formatting each partition

$FormatHT1 = @{

  DriveLetter        = 'W'

  FileSystem         = 'FAT'

  NewFileSystemLabel = 'w-fat'

}

Format-Volume @FormatHT1

$FormatHT2 = @{

  DriveLetter        = 'X'

  FileSystem         = 'exFAT'

  NewFileSystemLabel = 'x-exFAT'

}

Format-Volume @FormatHT2

$FormatHT3 = @{

  DriveLetter        = 'Y'

  FileSystem         = 'FAT32'

  NewFileSystemLabel = 'Y-FAT32'

}

Format-Volume  @FormatHT3

$FormatHT4 = @{

  DriveLetter        = 'Z'

  FileSystem         = 'ReFS'

  NewFileSystemLabel = 'Z-ReFS'

}

Format-Volume @FormatHT4

1. Getting all volumes on SRV1

Get-Volume | Sort-Object -Property DriveLetter

## How it works...

In step 1, you obtain the second disk on SRV1 and display the disk details. You initialized this disk previously using the GPT partitioning scheme. The output looks like this:

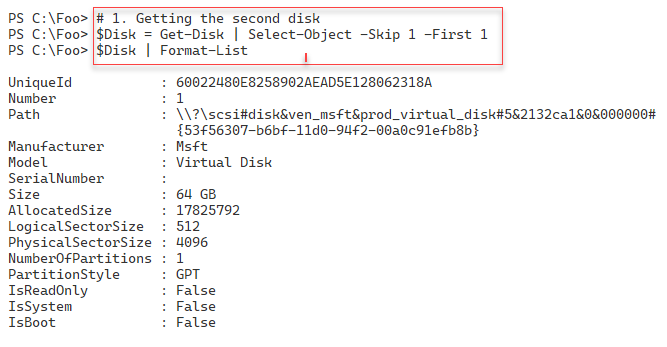


Figure 7.10: Getting the next RAW disk on SRV1

Insert image B18878\_07\_10.png

In step 2, you use the Create-Volume to create an S: volume/partition on the second disk in SRV1, with output like this:

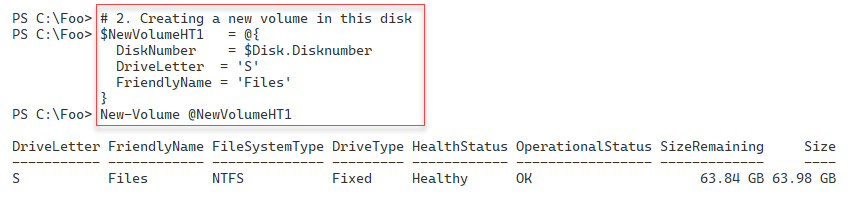


Figure 7.11: Creating the S: Volume

Insert image B18878\_07\_11.png

In step 3, you get the third disk on SRV1, the disk you initialized, in the prior recipe, with an MBR partitioning scheme. The output of this step is as follows:

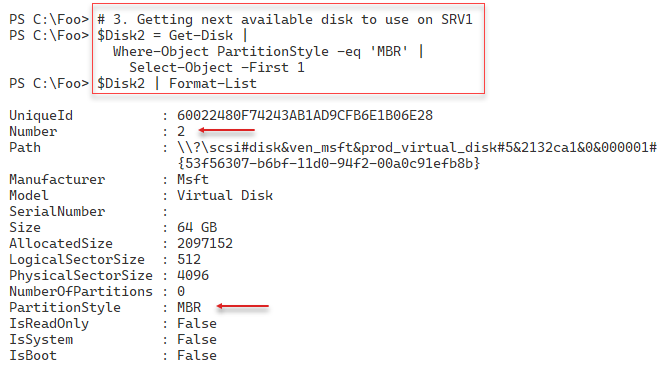


Figure 7.12: Getting the next disk

Insert image B18878\_07\_12.png

In step 4, you use the New-Partition cmdlet to create four new partitions on disk 2, with output like this:

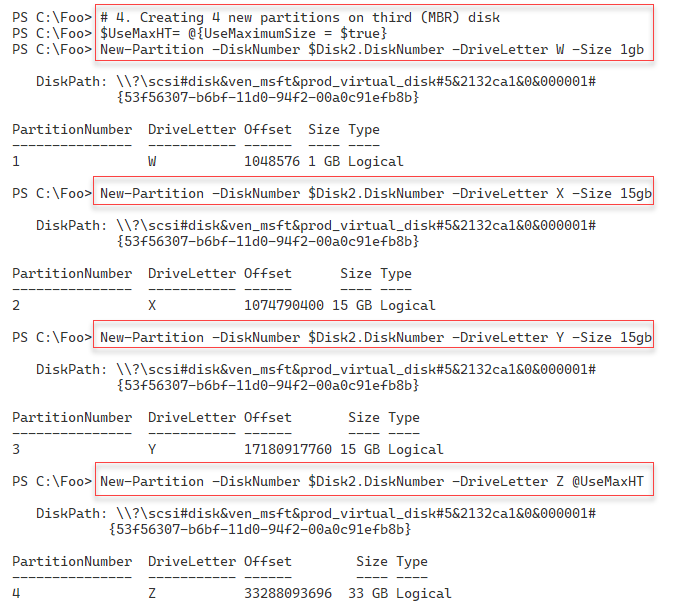


Figure 7.13: Creating partitions on Disk 2

Insert image B18878\_07\_13.png

Now that you have initialized this disk and created partitions, you need to format those partitions using a specific file system. In step 5, you format the partitions you created in the previous step with output like this:

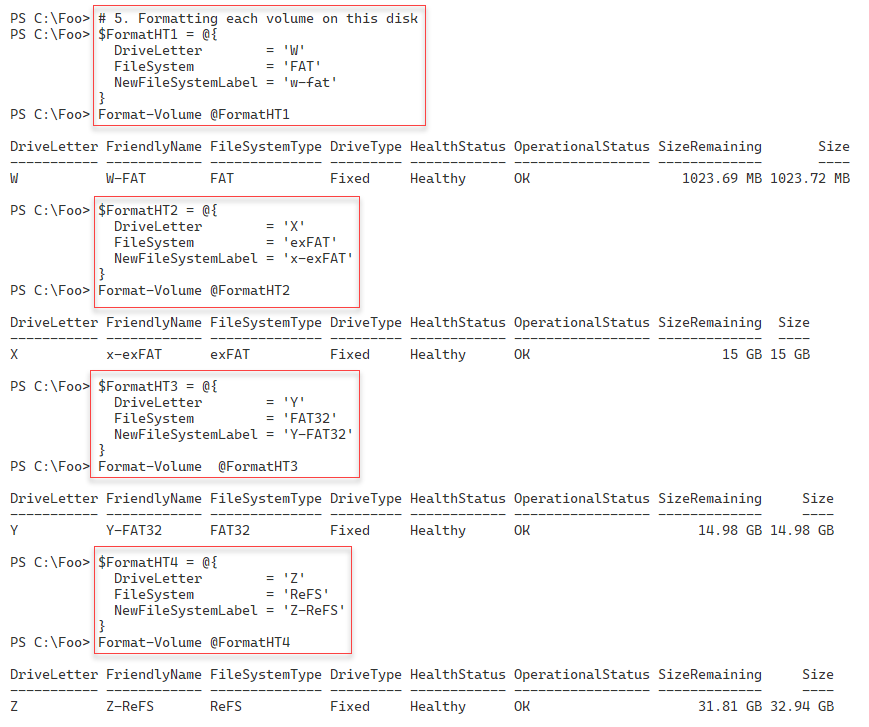


Figure 7.14: Formatting partitions on Disk 2

Insert image B18878\_07\_14.png

In step 7, you use the Get-Volume command to view the partitions you have created on SRV1 (thus far!). The output from this step is as follows:

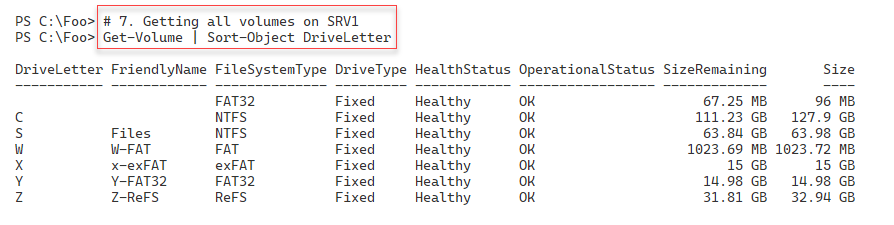


Figure 7.15: Viewing volumes on Disk 2

Insert image B18878\_07\_15.png

## There's more...

In step 1, you get the next available disk on the server. This disk is the second on SRV1, as seen in the previous recipe (Figure 7.9).

In step 4, you create four partitions on the disk. In this step, you create the $UseMaxHT hash table variable to simplify and shorten the final command. This step illustrates how you can mix parameters/values and hash tables when using a cmdlet. This approach can simplify some scripts.

In this recipe, with Disk 2, you used the MBR-partitioned disk and created four small partitions, each with a different file system. In practice, you would not usually create multiple MBR-based partitions on a single storage device.

# Exploring PowerShell providers and the File System Provider

One innovation in PowerShell that IT professionals soon learn to love is PowerShell providers. A provider is a component that provides access to specialized data stores for easy management. The provider makes the data appear in a drive with a path similar to how you access files in file stores.

PowerShell 7.2 comes with the following providers:

* **Registry**: provides access to registry keys and registry values (https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_registry\_provider.
* **Alias**: provides access to PowerShell'’s command aliases (https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_alias\_provider).
* **Environmen**t: provides access to Windows environment variables (https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_environment\_provider).
* **FileSystem**: provides access to files in a partition (https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_filesystem\_provider.
* **Function**: provides access to PowerShell'’s function definitions (https://lealrn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_function\_provider).
* **Variable:** provides access to PowerShell'’s variables (https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_variable\_provider).
* **Certificate**: provides access to the current user and local host's X.509 digital certificate stores (https://learn.microsoft.com/powershell/module/microsoft.powershell.security/about/about\_certificate\_provider).
* **WSMan**: provides a configuration surface for you to configure the WinRM service (https://kearn.microsoft.com/powershell/module/microsoft.wsman.management/about/about\_wsman\_provider).

The key advantage of PowerShell providers is that you do not need a set of cmdlets for each underlying data store. Instead, you use Get-Item or Get-ChildItem with any provider to return provider-specific data, as you can see in this recipe.

Other applications can add providers to a given host. For example, the IIS administration module creates an IIS: drive, and the Active Directory module creates an AD: drive.

And if you have organization'-unique data stores, you could create a customized provider. The SHiPS module, available from the PowerShell gallery, enables you to build a provider using PowerShell. As an example of the SHiPS platform’s capabilities, you can use a sample provider from the CimPSDrive module. This module contains a provider for the CIM repository. For more information on the SHiPS platform, see https://github.com/PowerShell/SHiPS/tree/development/docs. For more details on the CimPSDrive provider, see https://github.com/PowerShell/CimPSDrive.

## Getting ready

This recipe uses SRV1, a domain-joined host in the Reskit.Org domain. You used this server in previous recipes in this chapter. This recipe also uses the disks you added before the “Managing Disks” recipe.

## How to do it...

1. Getting PowerShell providers

Get-PSProvider

1. Getting drives from the registry provider

Get-PSDrive | Where-Object Provider -match 'Registry'

1. Looking at a registry key

$Path = 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'

Get-Item -Path $Path

1. Getting registered owner

(Get-ItemProperty -Path $Path -Name RegisteredOwner).RegisteredOwner

1. Counting aliases in the Alias: drive

Get-Item Alias:\* | Measure-Object

1. Finding aliases for Remove-Item

Get-ChildItem Alias:\* |

  Where-Object ResolvedCommand -match 'Remove-Item$'

1. Counting environment variables on SRV1

Get-Item ENV:\* | Measure-Object

1. Displaying the Windows installation folder

"Windows installation folder is [$env:windir]"

1. Checking on FileSystem provider drives on SRV1

Get-PSProvider -PSProvider FileSystem |

  Select-Object -ExpandProperty Drives |

    Sort-Object -Property Name

1. Getting home folder for FileSystem provider

$HomeFolder = Get-PSProvider -PSProvider FileSystem |

  Select-Object -ExpandProperty Home

$HomeFolder

1. Checking Function drive

Get-Module | Remove-Module -WarningAction SilentlyContinue

$Functions = Get-ChildItem -Path Function:

"Functions available [$($Functions.Count)]"

1. Creating a new function

Function Get-HelloWorld {'Hello World'}

1. Checking Function drive again

$Functions2 = Get-ChildItem -Path Function:

"Functions now available [$($Functions2.Count)]"

1. Viewing function definition

Get-Item Function:\Get-HelloWorld | Format-List \*

1. Counting defined variables

$Variables = Get-ChildItem -Path Variable:

"Variables defined [$($Variables.count)]"

1. Getting trusted root certificates for the local machine

Get-ChildItem -Path Cert:\LocalMachine\Root |

  Format-Table FriendlyName, Thumbprint

1. Examining ports in use by WinRM

Get-ChildItem -Path WSMan:\localhost\Client\DefaultPorts

Get-ChildItem -Path WSMan:\localhost\Service\DefaultPorts

1. Setting Trusted Hosts

Set-Item WSMan:\localhost\Client\TrustedHosts -Value '\*' -Force

1. Installing SHIPS and CimPSDrive modules

Install-Module -Name SHiPS, CimPSDrive -Force

1. Importing the CimPSDrive module and creating a drive

Import-Module -Name CimPSDrive

New-PSDrive -Name CIM -PSProvider SHiPS -Root CIMPSDrive#CMRoot

1. Examining BIOS using the CimPSDrive module

Get-ChildItem CIM:\Localhost\CIMV2\Win32\_Bios

## How it works...

In step 1, you use the Get-PSProvider to view all the PowerShell providers that are currently on SRV1. The output looks like this:

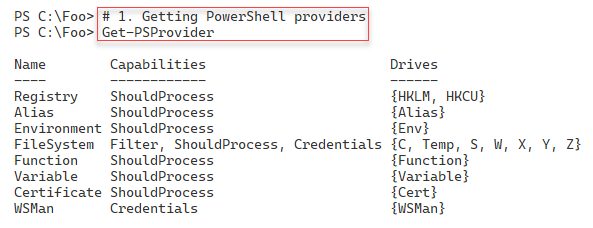


Figure 7.16: Viewing the PowerShell providers on SRV1

Insert image B18878\_07\_16.png

A provider can provide one or more drives. In step 2, you use the Get-PSDrive cmdlet to discover the drives currently provided by the Registry provider, with output like this:

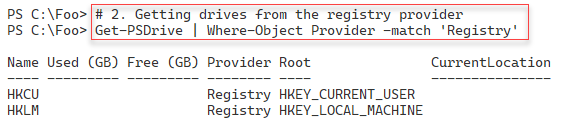


Figure 7.17: Viewing the drives in the registry provider

Insert image B18878\_07\_17.png

In step 4, you use the registry provider to retrieve the registry holding the Registered Owner. The Window installation process writes a value to the registry containing the name of the system’s owner. The output of this step looks like this:

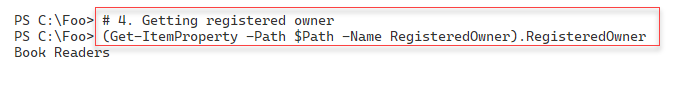


Figure 7.18: Obtaining the registered owner via the registry

Insert image B18878\_07\_18.png

In step 5, you use the PowerShell alias provider to count the number of aliases, with output like this:

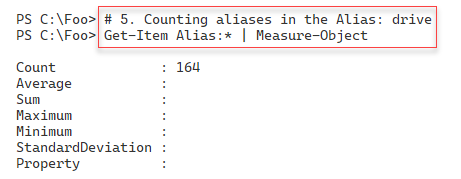


Figure 7.19: Counting the number of aliases on SRV1

Insert image B18878\_07\_19.png

In step 6, you use the Get-ChildItem command against the alias provider to discover the aliases to the Remove-Item command, with output like this:

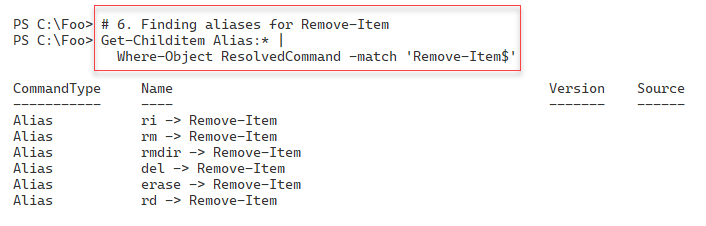


Figure 7.20: Discovering aliases to the Remove-Item command

Insert image B18878\_07\_20.png

In step 7, you use the Environment Variable provider to count the number of environment variables, with output like this:

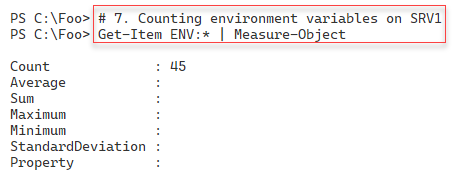


Figure 7.21: Counting the environment variables on SRV1

Insert image B18878\_07\_21.png

In step 8, you use the Environment variable to retrieve the name of the Windows installation folder, with output like this:

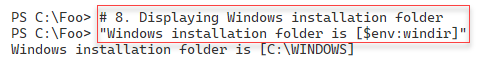


Figure 7.22: Counting the environment variables

Insert image B18878\_07\_22.png

A provider enables you to create drives, which you can think of as placeholders within the underlying data store. In step 9. you use the Get-PSProvider command to retrieve the drives surfaced by the file system provider, with output like this:

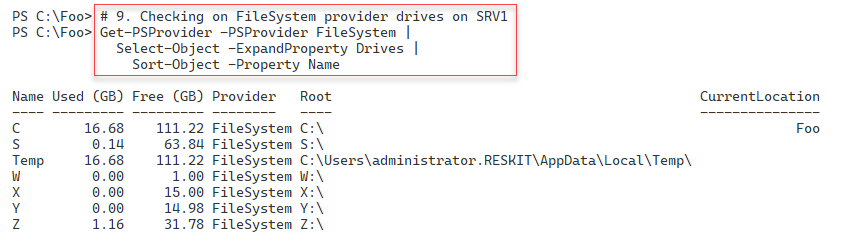


Figure 7.23: Getting Drives in the Filesystem provider

Insert image B18878\_07\_23.png

Each provider enables you to define a '‘home drive'’. You can use the Set‑Location command and specify a path of “~” to move to the home drive. In chapter 1, you created a new PowerShell profile and set the home drive for the file system provider (in Installing and Configuring PowerShell). In step 10, you get the home drive for the file system provider. The output of this step looks like this:

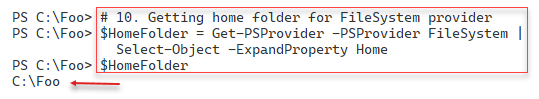


Figure 7.24: Getting the Filesystem provider home folder

Insert image B18878\_07\_24.png

In step 11, you remove all modules, removing any aliases defined by the loaded modules. Then you get and count the functions in the Function: drive, with output like this:

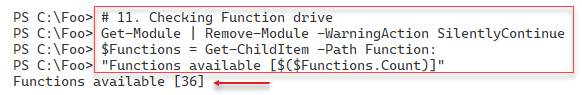


Figure 7.25: Getting and counting functions available

Insert image B18878\_07\_25.png

To test the alias provider, in step 12, you create a simple function. This step generates no output.

In step 13, you check the Function: drive again to view the number of functions available after adding a function. The output of this step looks like this:

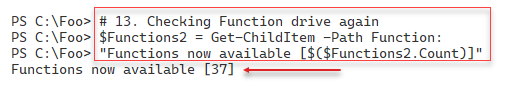


Figure 7.26: Getting and counting functions available again

Insert image B18878\_07\_26.png

Objects returned from the Function provider contain several properties, not the least of which is the function definition. In step 14, you view the function definition for the Get-HelloWorld function held in the Function: drive. The output looks like this:

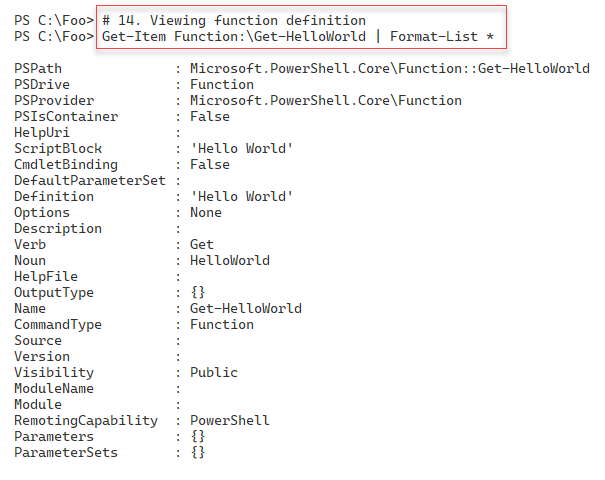


Figure 7.27: Getting function definition from Function: drive

Insert image B18878\_07\_27.png

The Variable: provider enables you to manipulate PowerShell variables. In step 15, you count the variables available in the current PowerShell session, with output like this:

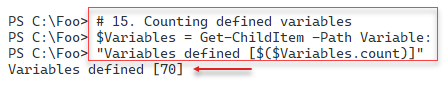


Figure 7.28: Counting variables on SRV1

Insert image B18878\_07\_28.png

The certificate provider enables you to manipulate X.509 digital certificates stored in the current user or system certificate stores. In step 16, you get the certificates from the current user’s trusted root certificate store, with output like this:

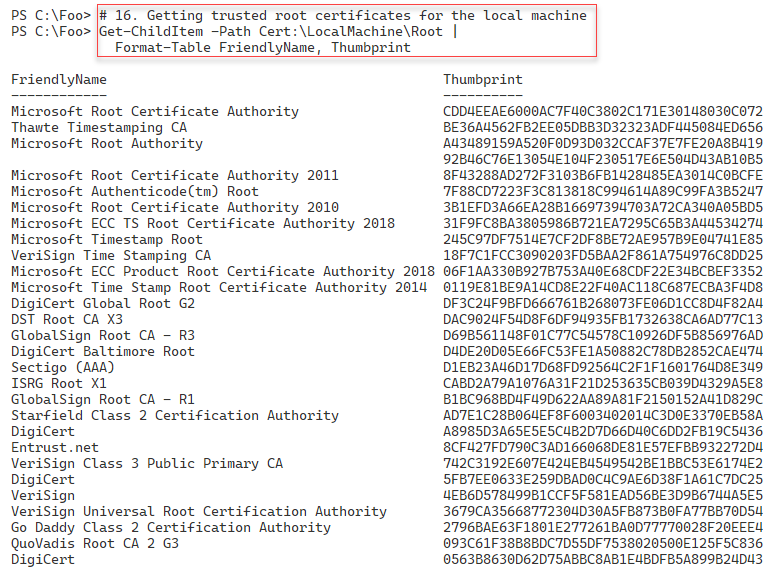


Figure 7.29: Getting certificates from the current user's trusted publisher certificate store

Insert image B18878\_07\_29.png

In PowerShell 7, the WinRM service provides the underpinnings to PowerShell remoting. Remoting uses WinRM to send PowerShell commands to a remote host and receive a response. With WinRM, you configure the WinRM client and WinRM server by updating items in the WSMan: drive. You can view the ports used by the WSMan client and WSMan server services on the SRV1 host, as shown in step 1. The output looks like this:

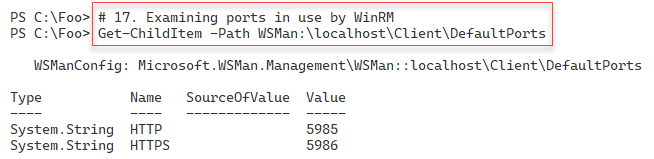


Figure 7.30: Getting WSMan service ports

Insert image B18878\_07\_30.png

To show how you can configure WinRM, in step 18, you set WinRM to trust explicitly any remote host by setting the TrustedHhosts item in the WSMan: drive. There is no output from this step.

Creating a provider in native C# is often a daunting step. But you can simplify the creation of customized providers by using the SHiPS module. One module created using SHiPS, is CimPSDrive. This module contains a CIM database provider enabling you to navigate the WMI database using PowerShell item commands. In step 20, you install the SHiPS and CimPSDrive modules, creating no output. Then in step 21, you import the CimPSDrive module and create a drive, which looks like this:

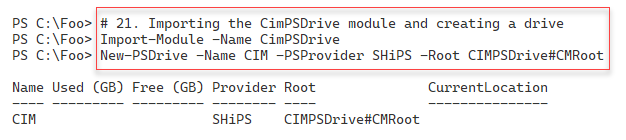


Figure 7.31: Importing the CimPSDrive module and creating a drive

Insert image B18878\_07\_31.png

In step 22, you use the newly created CIM PSDrive and examine the values of the Win32\_Bios class, with output like this:

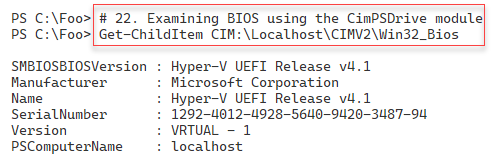


Figure 7.32: Viewing a WMI Class

Insert image B18878\_07\_32.png

## There's more...

In step 4, you use the registry provider to return the registered owner of this system. The Windows Installation process sets this value when you install the OS. If you use the Resource Kit build scripts on GitHub, the unattended XML files provide a user name and organization. Of course, you can change this in the XML or subsequently (by editing the registry.

In step 11, you check PowerShell’s Function: drive., This drive holds an entry for every function within a PowerShell session. Since PowerShell has no Remove-Function command, to remove a function from your PowerShell session, you remove a function by removing its entry (Remove-Item -Path Function:<functon to remover).

Step 16 lets you view the trusted root CA certificates in the local machine’s certificate store. Depending on your organization’s needs, you can add or remove certificates from this store – but be very careful. Microsoft maintains these root certificates as part of the Microsoft Root Certificate Program, which supports the distribution of root certificates, enabling customers to trust Windows products. You can read more about this program at https://learn.microsoft.com/security/trusted-root/program-requirements.

In step 18, you set the WinRM TrustedHosts item to “\*,” which means that whenever PowerShell negotiates a remoting connection with a remote host, it trusts the remote host machine is who it says it is and is not an imposter. Setting TrustedHosts like this can have security implications – you should be careful about when and where you modify this setting.

The SHiPS framework, which you install in step 19, is a module that helps you to develop a provider. This framework can be useful in enabling you to create new providers to unlock data in your organization. The framework is available, as you see in this recipe, from the PowerShell Gallery or from GitHub at: https://github.com/PowerShell/SHiPS. For a deeper explanation of the SHiPS framework, see https://4sysops.com/archives/create-a-custom-powershell-provider/.

# Managing Storage Replica

Storage Replica (SR) is a feature of Windows Server 2022 that replicates storage volumes to other systems. SR is only available with the Windows Server 2022 Datacenter edition.

You typically use Storage Replica to maintain a complete replica of one or more disk volumes, typically for disaster recovery. SR works on a volume basis. Once you configure SR and create a replication partnership, SR replicates all the files in a volume, for example, the F: Drive, to a disk on another host, for instance, SRV2. After setting up the SR partnership, as you update the F: drive, Windows automatically updates the target drive on SRV2. However, you cannot see the files on SRV2. An SR partnership also requires a drive on the source and destination hosts for internal logging.

## Getting ready

You use both SRV1 and SRV2 in this recipe. After adding and configuring additional virtual disks to this host, you run this recipe on SRV1, a domain-joined host in the Reskit.Org domain. You must have installed PowerShell 7 and VS Code on this host. This recipe uses the S: drive you created earlier on SRV1 in Managing disks, plus a new G: drive, which you created on disk number 3. You also create the corresponding disks on SRV2)

## How to do it...

1. Getting the disk number of the disk holding the S partition

$Part = Get-Partition -DriveLetter S

"S drive on disk [$($Part.DiskNumber)]"

1. Creating S: drive on SRV2

$ScriptBlock = {

  Initialize-Disk -Number $using:Part.DiskNumber -PartitionStyle GPT

  $NVHT = @{

   DiskNumber   =  $using:Part.DiskNumber

    FriendlyName = 'Files'

    FileSystem   = 'NTFS'

    DriveLetter  = 'S'

  }

  New-Volume @NVHT

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock

1. Creating content on S: on SRV1

1..100 | ForEach-Object {

  $NewFldr = "S:\CoolFolder$\_"

  New-Item -Path $NewFldr -ItemType Directory | Out-Null

  1..100 | ForEach-Object {

    $NewFile = "$NewFldr\CoolFile$\_"

    "Cool File" | Out-File -PSPath $NewFile

  }

}

1. Counting files/folders on S:

Get-ChildItem -Path S:\ -Recurse | Measure-Object

1. Examining the same S drive remotely on SRV2

$ScriptBlock2 = {

  Get-ChildItem -Path S:\ -Recurse |

    Measure-Object

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock2

1. Adding the storage replica feature to SRV1

Add-WindowsFeature -Name Storage-Replica -IncludeManagementTools |

  Out-Null

1. Adding the Storage Replica Feature to SRV2

$SB= {

  Add-WindowsFeature -Name Storage-Replica | Out-Null

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB

1. Restarting SRV2 and waiting for the restart

$RSHT = @{

  ComputerName = 'SRV2'

  Force        = $true

}

Restart-Computer @RSHT -Wait -For WinRM

1. Restarting SRV1 to finish the installation process

Restart-Computer

1. Creating a G: volume in disk 3 on SRV1

$ScriptBlock3 = {

  Initialize-Disk -Number 3 -PartitionStyle GPT | Out-Null

  $VolumeHT = @{

   DiskNumber   =  3

   FriendlyName = 'SRLOGS'

   DriveLetter  = 'G'

  }

  New-Volume @VolumeHT

}

Invoke-Command -ComputerName SRV1 -ScriptBlock $ScriptBlock3

1. Creating G: volume on SRV2

Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3

1. Viewing volumes on SRV1

Get-Volume | Sort-Object -Property Driveletter

1. Viewing volumes on SRV2

Invoke-Command -Computer SRV2 -ScriptBlock {

    Get-Volume | Sort-Object -Property Driveletter

}

1. Creating an SR replica partnership

$NewSRHT =  @{

  SourceComputerName       = 'SRV1'

  SourceRGName             = 'SRV1RG1'

  SourceVolumeName         = 'S:'

  SourceLogVolumeName      = 'G:'

  DestinationComputerName  = 'SRV2'

  DestinationRGName        = 'SRV2RG1'

  DestinationVolumeName    = 'S:'

  DestinationLogVolumeName = 'G:'

  LogSizeInBytes           = 2gb

}

New-SRPartnership @NewSRHT

1. Examining the volumes on SRV2

$ScriptBlock3 = {

  Get-Volume |

    Sort-Object -Property DriveLetter |

      Format-Table}

Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3

1. Reversing the replication

$ReverseHT = @{

  NewSourceComputerName   = 'SRV2'

  SourceRGName            = 'SRV2RG1'

  DestinationComputerName = 'SRV1'

  DestinationRGName       = 'SRV1RG1'

  Confirm                 = $false

}

Set-SRPartnership @ReverseHT

1. Viewing the SR Partnership

Get-SRPartnership

1. Examining the files remotely on SRV2

$ScriptBlock4 = {

  Get-ChildItem -Path S:\ -Recurse |

    Measure-Object

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock4

## How it works...

In step 1, you get the disk number of the disk holding the S: partition with output like this:

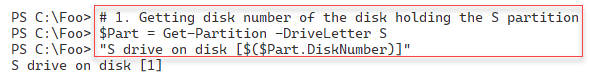


Figure 7.33: Viewing a WMI Class

Insert image B18878\_07\_33.png

In step 2, you create a new S: volume on SRV2. This step produces output like this:

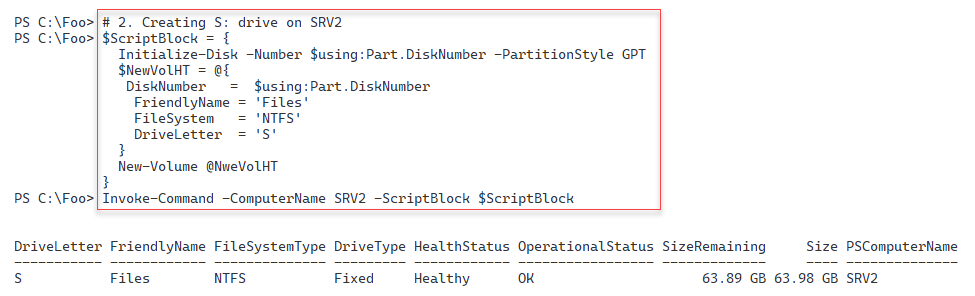


Figure 7.34: Creating S: on SRV2

Insert image B18878\_07\_34.png

In step 3, you create folders and files on SRV1, which creates no output. In step 4, you count the number of files and folders on the S: drive, with output like this:

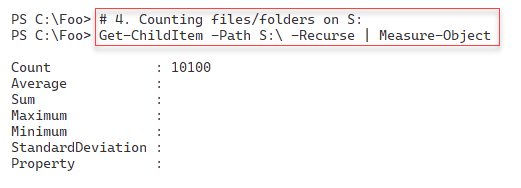


Figure 7.35: Viewing the files on S: on SRV1

Insert image B18878\_07\_35.png

In step 5, you examine the files and folders on the S: drive on SRV2, with output like this:

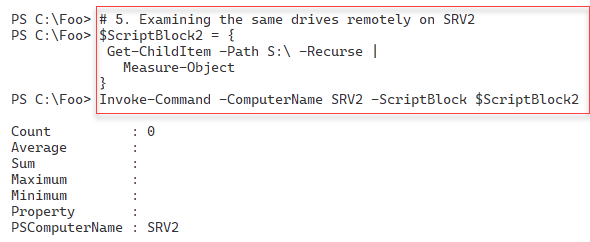


Figure 7.36: Viewing the files on SRV2

Insert image B18878\_07\_36.png

Now that you have the S: drive created on both SRV1 and SRV2 and have created content, in step 6, you add the Storage Replica feature to SRV1, creating the following output:

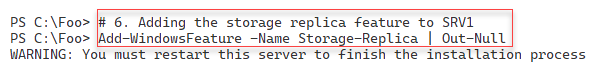


Figure 7.37: Adding Storage Replica to SRV1

Insert image B18878\_07\_37.png

In step 7, you add the Storage Replica feature to SRV2, generating similar output, like this:

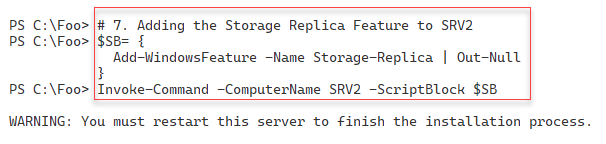


Figure 7.38: Adding Storage Replica to SRV2

Insert image B18878\_07\_38.png

In step 8, you restart SRV2. Then, in step 9, you restart SRV1. Neither step produces console output.

In step 10, you create a new G: volume on SRV1 (to hold Storage Replica log files), with output like this:

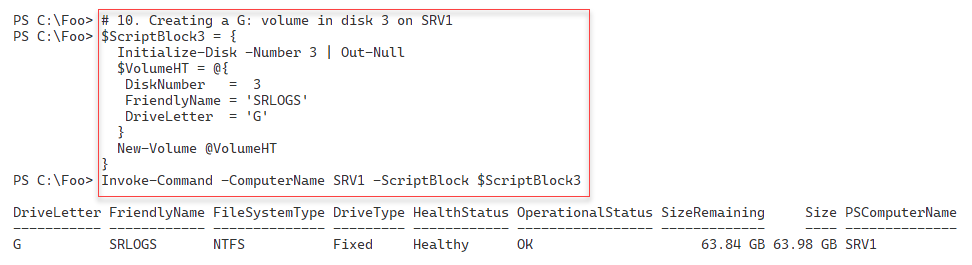


Figure 7.39: Creating a G: volume on SRV1

Insert image B18878\_07\_39.png

In step 11, you create a G: on SRV2, with output like this:

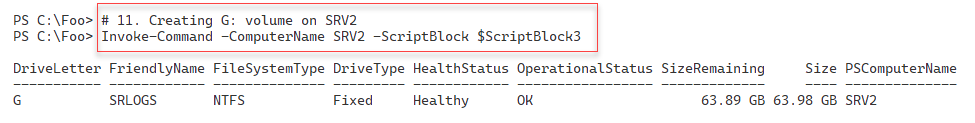


Figure 7.40: Creating a G: volume on SRV2

Insert image B18878\_07\_40.png

In step 12, you examine the volumes on SRV2, with output like this:

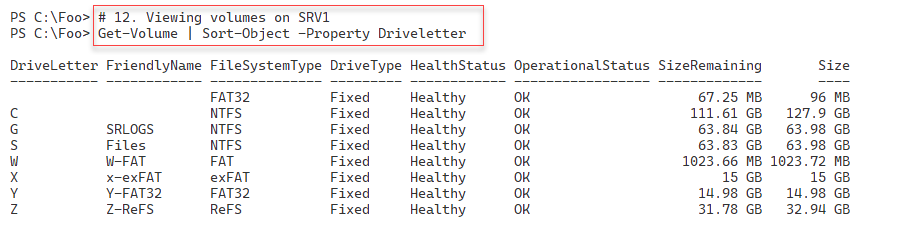


Figure 7.41: Viewing volumes on SRV1

Insert image B18878\_07\_41.png

In step 13, you examine the volumes on SRV2, with output like this:

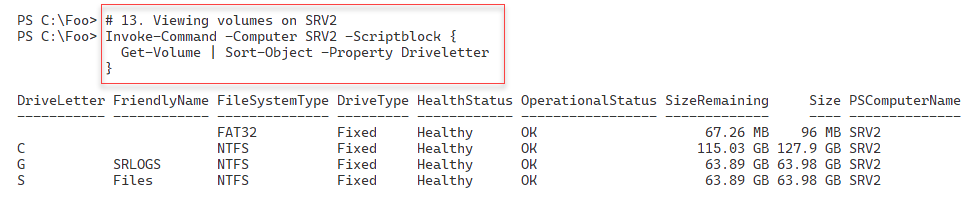


Figure 7.42: Viewing volumes on SRV2

Insert image B18878\_07\_42.png

In step 14, you use the New-SRPartnership command to create a new storage replica partnership creating the following output:

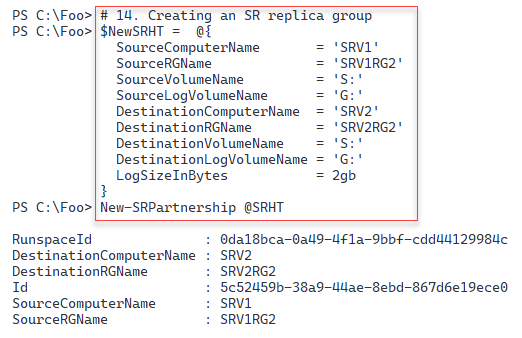


Figure 7.43: Creating a storage replica partnership

Insert image B18878\_07\_43.png

In step 15, you examine the volumes again on SRV2, with output like this:

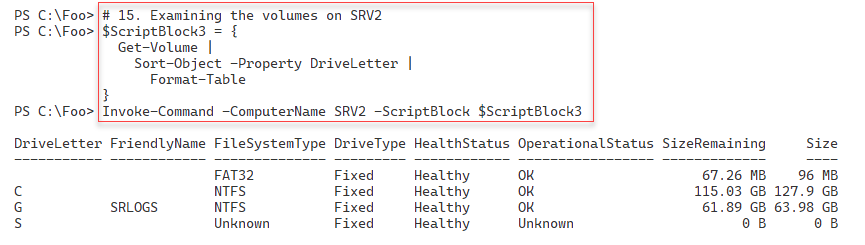


Figure 7.44: Examining the volumes on SRV2

Insert image B18878\_07\_44.png

Thus far, in this recipe, you have created an SR partnership, replicating the S: volume from SRV1 to SRV2. In the next step, step 16, you reverse the replication – replicating the S: volume from SRV2 to SRV1, which creates no output.

In step 17, you view the now-reversed SR partnership with the following output:

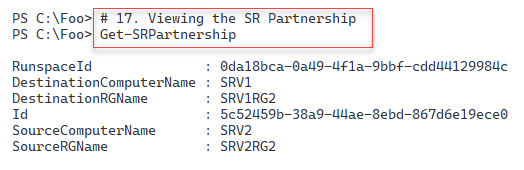


Figure 7.45: Examining the reversed SR partnership[

Insert image B18878\_07\_45.png

## There's more...

In the first five steps in this recipe, you create and review content on SRV1, which you intend to have Storage Replica replicate to SRV2. In practice, the data you are replicating would be the files in a file server or represent other files you want to synchronize.

In step 8, you reboot SRV2 remotely. If this is the first time you have rebooted SRV2 remotely using PowerShell, you may find that the command never returns even when SRV2 is demonstrably up and running. Just kill off the current PowerShell console (or close VS code) and open a new console to continue the recipe.

After creating the SR partnership, replication from SRV1 to reverse the replication in step 16. Before you reverse the replication, you should ensure that the initial replication has been completed – depending on the size of the SR replications, it could take quite a while. You can use the command (Get-SRGroup).Replicas.ReplicationStatus to check the status of the initial replication.

# Deploying Storage Spaces

Storage Spaces is a technology in Windows 10/11 and Windows Server that implements software RAID. You can add multiple physical drives to your server or workstation, and create fault-tolerant volumes for your host. You can read more about Storage Spaces at https://learn.microsoft.com/windows-server/storage/storage-spaces/overview.

You can use Storage Spaces on a single host or server to protect against unexpected disk drive failures. You should note that Storage Spaces is separate from Storage Spaces Direct (S2D). S2D enables you to create a virtual SAN with multiple hosts providing SMB3 access to a scale-out file server.

## Getting ready

You run this recipe on SRV1, a domain-joined host in the Reskit.Org domain. You also need DC1, a domain controller for the Reskit.org domain. This recipe uses the five virtual disks you added to SRV1 earlier in the chapter .at the start of the Managing Disks recipe.

## How to do it...

1. Viewing disks available for pooling

$Disks = Get-PhysicalDisk -CanPool $true

$Disks | Sort-Object -Property Deviceid

1. Creating a storage pool

$NewPoolHT = @{

  FriendlyName                 = 'RKSP'

  StorageSubsystemFriendlyName = "Windows Storage\*"

  PhysicalDisks                = $Disks

}

New-StoragePool @NewPoolHT

1. Creating a mirrored hard disk named Mirror1

$VDisk1HT = @{

  StoragePoolFriendlyName   = 'RKSP'

  FriendlyName              = 'Mirror1'

  ResiliencySettingName     = 'Mirror'

  Size                      = 8GB

  ProvisioningType          = 'Thin'

}

New-VirtualDisk @VDisk1HT

1. Creating a three-way mirrored disk named Mirror2

$VDisk2HT = @{

  StoragePoolFriendlyName    = 'RKSP'

  FriendlyName               = 'Mirror2'

  ResiliencySettingName      = 'Mirror'

  NumberOfDataCopies         = 3

  Size                       = 8GB

  ProvisioningType           = 'Thin'

}

New-VirtualDisk @VDisk2HT

1. Creating a volume in Mirror1

Get-VirtualDisk  -FriendlyName 'Mirror1' |

  Get-Disk |

    Initialize-Disk -PassThru |

      New-Partition -AssignDriveLetter -UseMaximumSize |

        Format-Volume

1. Creating a volume in Mirror2

Get-VirtualDisk  -FriendlyName 'Mirror2' |

  Get-Disk |

    Initialize-Disk -PassThru |

      New-Partition -AssignDriveLetter -UseMaximumSize |

        Format-Volume

1. Viewing volumes on SRV1

Get-Volume | Sort-Object -Property DriveLetter

## How it works...

In step 1, you examine the disks available for polling within SRV1. The output should look something like this:

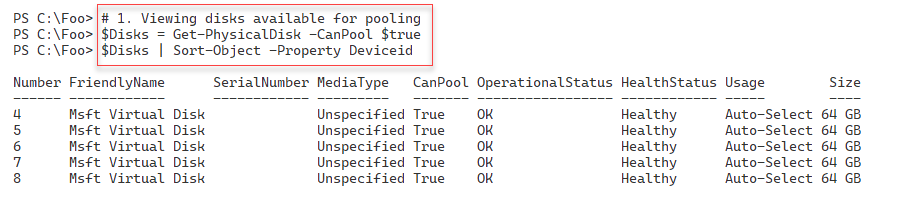


Figure 7.46: Viewing disks available for pooling on SRV1

Insert image B18878\_07\_46.png

In step 2, you use the New-StoragePool cmdlet to create a new storage pool using the five disks you discovered in the previous step. The output looks like this:

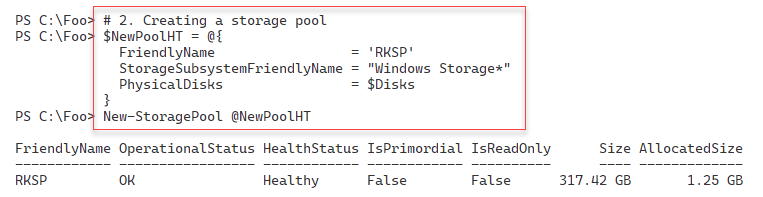


Figure 7.47: Creating a new storage pool

Insert image B18878\_07\_47.png

In step 3, you create a new Storage Space called Mirror1. This Storage Space is effectively a virtual disk within a storage pool. The output of this step looks like this:

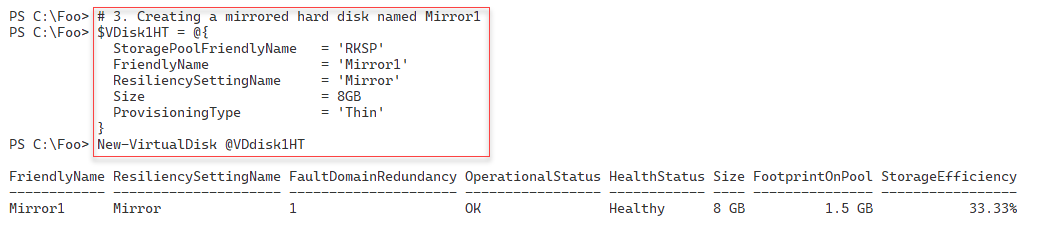


Figure 7.48: Creating a new mirrored disk

Insert image B18878\_07\_48.png

In step 4, you create a three-way mirrored disk called Mirror2. The output of this step looks like this:

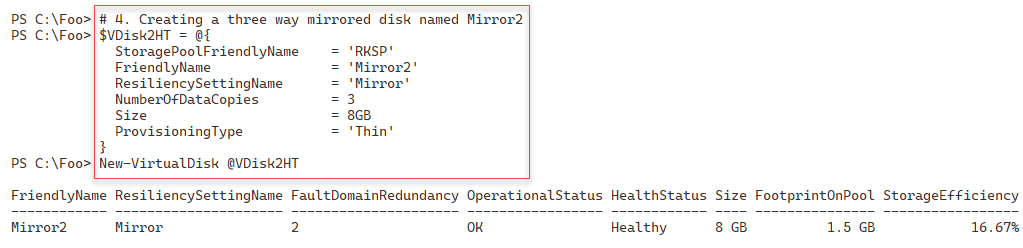


Figure 7.49: Creating a three-way mirrored storage space

Insert image B18878\_07\_49.png

In step 5, you create a new volume in the Mirror1 Storage Space, which looks like this:

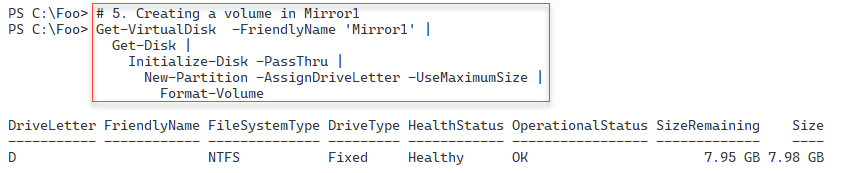


Figure 7.50: Creating a new disk volume inside the Mirror1 storage space

Insert image B18878\_07\_50.png

In step 6, you create another new volume. You create the volume in the three-way mirror Storage Space Mirror2, with output like this:

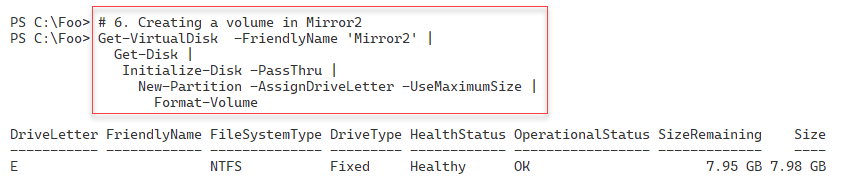


Figure 7.51: Creating a new disk volume inside the Mirror2 storage space

Insert image B18878\_07\_51.png

In the final step in this recipe, step 7, you use the Get-Volume command to view all the volumes available in SRV1, with output like this:

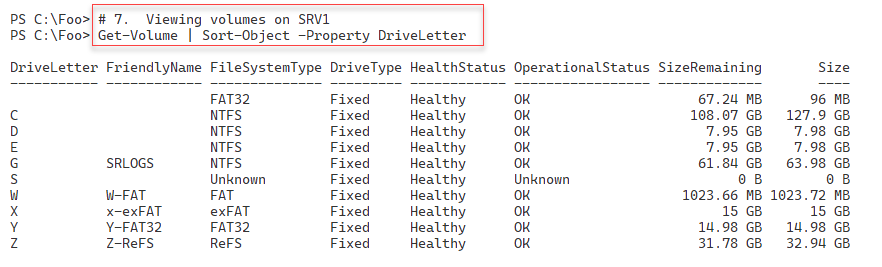


Figure 7.52: Viewing disk volumes available on SRV1

Insert image B18878\_07\_52.png

## There's more...

In step 1, you get the disks available for pooling with Storage Spaces. The output assumes you have performed the recipes earlier in this chapter (e.g., to create volumes, etc.).

In steps 5 and step 6, you create two new disk volumes in SRV1. The first is the D: drive, which you created in the mirror set Mirror1, and the second is the E: drive, a disk volume you create in the three-way mirror Storage Space. The drive you create in Mirror1 is resilient to losing a single disk, whereas the E: drive created in the Mirror2 storage space can sustain two disk failures and still provide full access to your information.