9

Managing Storage

In this chapter, we cover the following recipes:

* Managing physical disks and volumes
* Managing filesystems
* Exploring providers and the FileSystem provider
* Managing Storage Replica
* Deploying Storage Spaces

# Introduction

Windows Server 2022 provides a range of features that allows access to a wide variety of storage and storage devices. Windows supports spinning disks, USB memory sticks, and SSD devices (including NVMe SSD devices).

Before you can use a disk to hold files, you need to create partitions or volumes on the device and format them. When you first initialize a disk, you need to define which partitioning method to use. You have two choices:

Master Boot Record (MBR)

GUID Partition Table (GPT)

These days, most PCs use the 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 32-bit PCs, older PCs, 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/.

With a volume created, you can then format the disk volume. Windows supports five filesystems you can use: ReFS, NTFS, exFAT, UDF, and FAT32. For details of the latter four, refer to https://docs.microsoft.com/en-us/windows/desktop/fileio/filesystemfunctionality-comparison. The ReFS filesystem is more recent and is based on NTFS, but lacks some features a file server might need (it has no encrypted files). A benefit of the ReFS filesystem 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. Managing ACLs is somewhat tricky, and PowerShell itself lacks rich support for managing ACLs and ACL inheritance. To manage ACLs on NTFS volumes, as you see in the Managing NTFS file and folder permissions recipe in Chapter 10, Managing Shared Data, you can download and use a third-party module, NTFSSecurity.

Storage Replica is a feature of Windows Server (Datacenter only) that creates a disk on a remote system. 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 Window 10 and Windows Server that can help you protect against a disk drive’s failure. In effect, Storage Spaces provides software RAID, which you investigate in Deploying Storage Spaces.

# Managing physical disks and volumes

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

You can use two partitioning schemes: the older format of MBR and the more recent 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 just 2 TB. And creating more than four partitions required you to create an extended partition and create additional partitions inside the extended partition. The GPT scheme enabled much larger drives (partition limits are OS-imposed) and up to 128 partitions per drive. You typically use GPT partitioning with Windows Server.

In this chapter, you use eight new virtual disk devices in the server, SRV1 and create/use new volumes/partitions on those disks. At the start of this recipe, you add all eight virtual disks to the SRV1 VM. In the recipe itself, you use just the first two of these new disks.

## 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.

The recipes in this chapter make use of eight additional virtual disks. You can run the following script on the Hyper-V host to add the new disks to the SRV1 and SRV2 VMs:

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

# Run on VM host

# 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 disks to SRV1/2 for storage chapter

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

0..7 | ForEach-Object {

  $DHT1 = @{

    VMName           = 'SRV1'

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

    ControllerType   = 'SCSI'

    ControllerNumber = 0

  }

  $DHT2 = @{

    VMName           = 'SRV2'

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

    ControllerType   = 'SCSI'

    ControllerNumber =  0

  }

  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

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

## How to do it...

1. Getting the first new physical disk on SRV1

$Disks = Get-Disk |

           Where-Object PartitionStyle -eq Raw |

             Select-Object -First 1

$Disks | Format-Table -AutoSize

1. Initializing the first disk

$Disks |

  Where-Object PartitionStyle -eq Raw |

    Initialize-Disk -PartitionStyle GPT

1. Re-displaying all disks in SRV1

Get-Disk |

  Format-Table -AutoSize

1. Viewing volumes on SRV1

Get-Volume | Sort-Object -Property DriveLetter

1. Creating a F: volume in disk 1

$NVHT1 = @{

  DiskNumber   =  $Disks[0].DiskNumber

  FriendlyName = 'Files'

  FileSystem   = 'NTFS'

  DriveLetter  = 'F'

}

New-Volume @NVHT1

1. Creating two partitions in disk 2 - first create S volume

Initialize-Disk -Number 2 -PartitionStyle MBR

New-Partition -DiskNumber 2  -DriveLetter S -Size 32gb

1. Creating a second partition T on disk 2

New-Partition -DiskNumber 2  -DriveLetter T -UseMaximumSize

1. Formatting S: and T:

$NVHT1 = @{

  DriveLetter        = 's'

  FileSystem         = 'NTFS'

  NewFileSystemLabel = 'GD Shows'}

Format-Volume @NVHT1

$NVHT2 = @{

  DriveLetter        = 'T'

  FileSystem         = 'FAT32'

  NewFileSystemLabel = 'GD Pictures'}

Format-Volume @NVHT2

1. Getting partitions on SRV1

Get-Partition  |

  Sort-Object -Property DriveLetter |

    Format-Table -Property DriveLetter, Size, Type, \*name

1. Getting volumes on SRV1

Get-Volume |

  Sort-Object -Property DriveLetter

1. Viewing disks in SRV1

Get-Disk | Format-Table

## How it works...

In step 1, you use the Get-Disk cmdlet to get the first virtual disk in the SRV1 VM, one of the eight new disks you added previously. The output of this step looks like this:

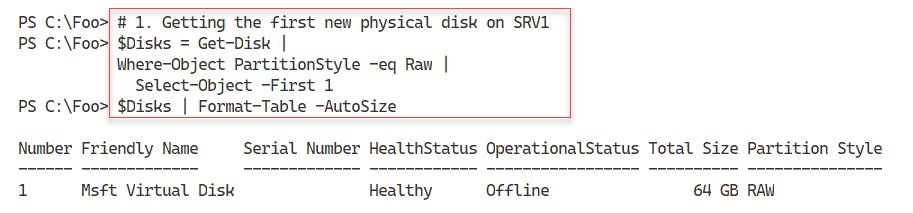


Figure 9.1: Getting the first new physical disk on SRV1

1. Insert image B42024\_09\_01.png

In step 2, you initialize the disk (disk number 1), which generates no output. Then, in step 3, you view all the disks in SRV1, with output that looks like this:

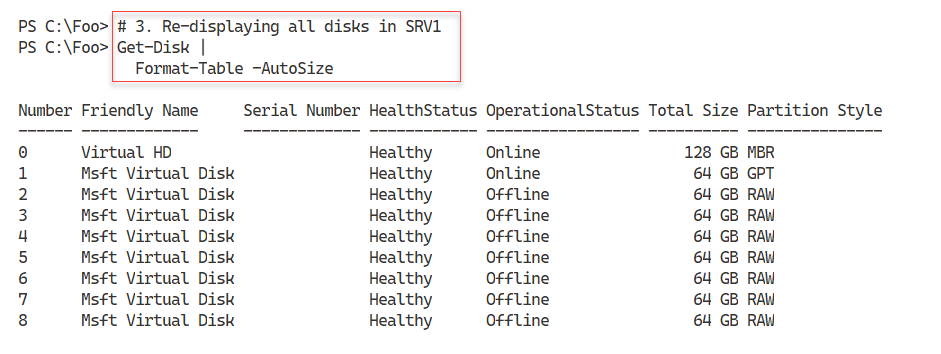


Figure 9.2: Viewing all the disks in SRV1

1. Insert image B42024\_09\_02.png

In step 4, you use the Get-Volume command to get the available volumes on SRV1, with output like this:

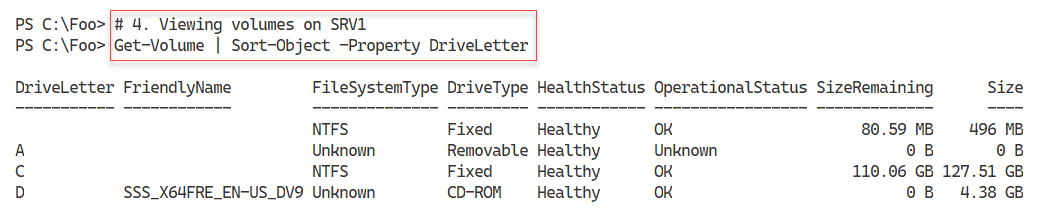


Figure 9.3: Viewing volumes in SRV1

1. Insert image B42024\_09\_03.png

With step 5, you create a new volume, and format it with the NTFS filesystem. The output from this step looks like this:

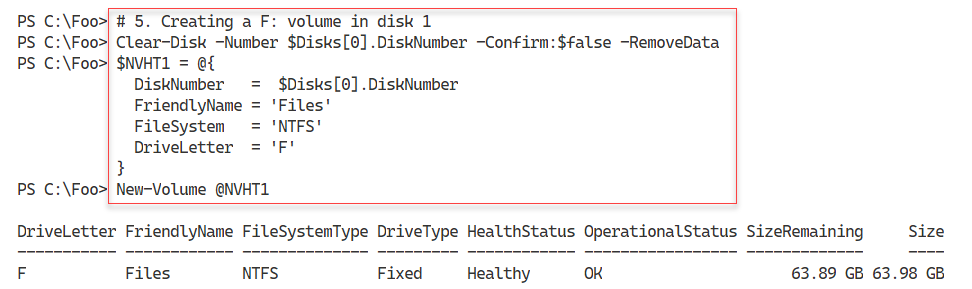


Figure 9.4: Creating an F: volume

1. Insert image B42024\_09\_04.png

In step 6, you initialize disk 2 and create a 32 GB partition with the S: drive. The output of this step looks like this:

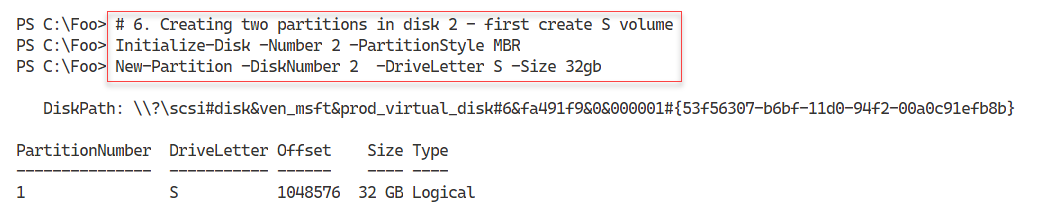


Figure 9.5: Creating an S: partition

1. Insert image B42024\_09\_05.png

In step 7, you create a second partition on disk 2, with output like this:

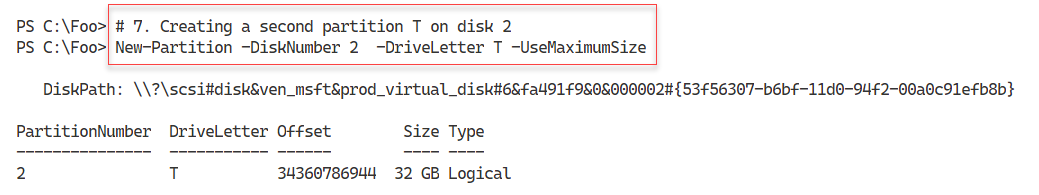


Figure 9.6: Creating a T: partition

1. Insert image B42024\_09\_06.png

In step 8, you format the newly created S: and T: drives. You use the NTFS filesystem for the S: drive, and the FAT32 filesystem for the T: drive, with output like this:

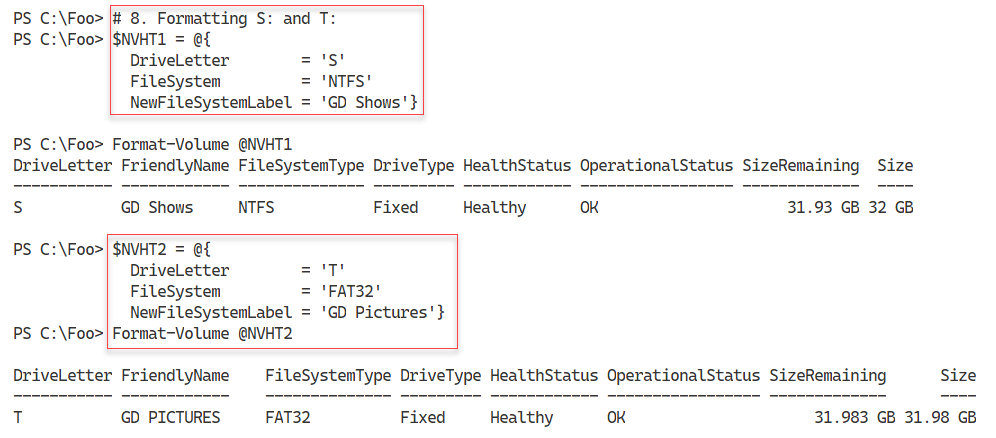


Figure 9.7: Formatting the S: and T: drives

**Insert image B42024\_09\_07.png**

In step 9, you view the partitions available on SRV1, with output like this:

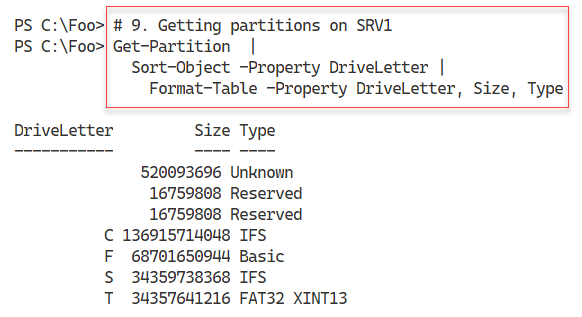


Figure 9.8: Getting partitions on SRV1

**Insert image B42024\_09\_08.png**

In step 10, you view the volumes on SRV1 using the Get-Volume command, with output like this:

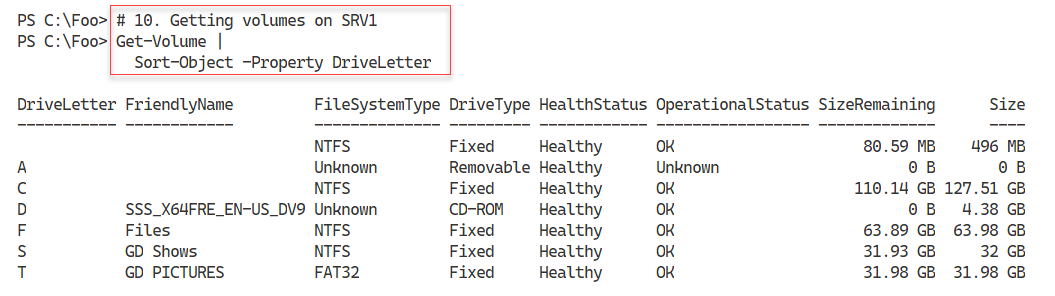


Figure 9.9: Getting volumes on SRV1

**Insert image B42024\_09\_09.png**

In the final step in this recipe, step 11, you view all the disks in SRV1, with output like this:



Figure 9.10: Viewing all the disks in SRV1

**Insert image B42024\_09\_10.png**

## There’s more...

With Windows (Windows 10 and Windows Server 2022), you create usable data drives using either the New‑Volume or New-Partition commands. The New-Volume cmdlet, shown in step 5, creates a partition on the specified disk, formats the partition with a filesystem, and gives it a label and a drive letter. In step 6, you initialize the disk with the MBR formatting and create a new partition in the disk. You create a second partition in step 7 and format the two partitions in step 8.

In step 10, you view the volumes on SRV1. Notice that in step 8, you specified a volume label for the T: volume with uppercase and lowercase letters. However, Windows converts this friendly name to all uppercase when you create the partition. This is by design.

# Managing filesystems

To make use of a "disk" device, whether a spinning disk, CD/DVD device, or a solid-state device, you must format that device/drive with a filesystem. In Windows, in addition to allowing you to specify which specific filesystem to use, you can also give the partition a drive letter and filesystem label while formatting the drive.

In most cases, you use NTFS as your filesystem of choice. It is robust and reliable and provides efficient access control as well as providing encryption and compression. ReFS might be a good choice for some specialized workloads, particularly on a physical Hyper-V host where you might use the ReFS filesystem on disks you use to hold your VMs’ virtual hard drives. For interoperability with things like video and still cameras, you may need to use the FAT, FAT32, or exFAT filesystems.

For more details on the difference between NTFS, FAT, FAT32, and ExFAT filesystems, see https://medium.com/hetman-software/the-difference-between-ntfs-fat-fat32-and-exfat-file-systems-ec5172c60ccd. And for more details on the ReFS filesystem, see https://docs.microsoft.com/en-us/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 physical disks and volumes recipe, you added eight virtual disks to the SRV1 VM and used the first two. In this recipe, you use the third of those disks.

## How to do it...

1. Getting disk to use on SRV1

$Disk = Get-Disk |

          Where-Object PartitionStyle -eq 'RAW' |

            Select-Object -First 1

1. Viewing disk

$Disk | Format-List

1. Viewing partitions on the disk

$Disk | Get-Partition

1. Initializing this disk and creating 4 partitions

Initialize-Disk -Number $Disk.DiskNumber -PartitionStyle GPT

New-Partition -DiskNumber $Disk.DiskNumber  -DriveLetter W -Size 15gb

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

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

$UMHT= @{UseMaximumSize = $true}

New-Partition -DiskNumber $Disk.DiskNumber  -DriveLetter Z @UMHT

1. Formatting each partition

$FHT1 = @{

    DriveLetter        = 'W'

    FileSystem         = 'FAT'

    NewFileSystemLabel = 'w-fat'

}

Format-Volume @FHT1

$FHT2 = @{

    DriveLetter        = 'X'

    FileSystem         = 'exFAT'

    NewFileSystemLabel = 'x-exFAT'

}

Format-Volume @FHT2

$FHT3 = @{

    DriveLetter        = 'Y'

    FileSystem         = 'FAT32'

    NewFileSystemLabel = 'Y-FAT32'

}

Format-Volume  @FHT3

$FHT4 = @{

    DriveLetter        = 'Z'

    FileSystem         = 'ReFS'

    NewFileSystemLabel = 'Z-ReFS'

}

Format-Volume @FHT4

1. Getting volumes on SRV1

Get-Volume | Sort-Object DriveLetter

## How it works...

In step 1, you get the first unused disk in SRV1 and store it in the $Disk variable. This step produces no output. In step 2, you view the disk object with output like this:



Figure 9.11: Viewing disk 3

**Insert image B42024\_09\_11.png**

In step 3, you view the partitions on disk 3. Since there are no partitions (yet) on this disk drive, this step produces no output.

In step 4, you create four partitions on disk 3. The first three occupy 15 GB each, with the fourth taking up all the remaining space in the disk. The output of this step looks like this:

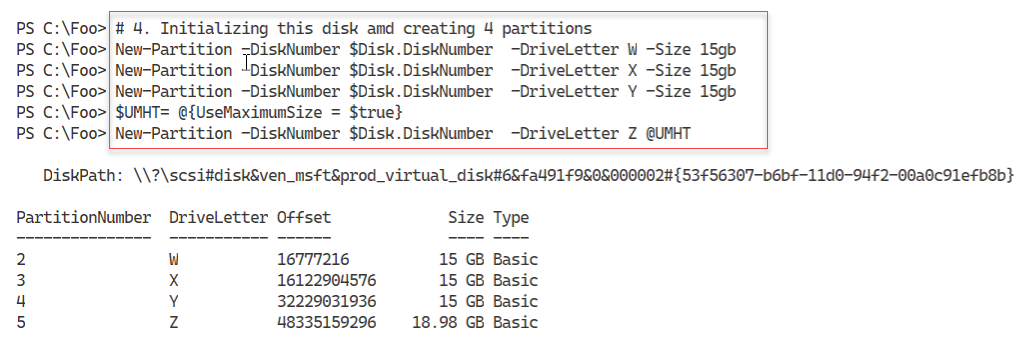


Figure 9.12: Creating partitions on disk 3

**Insert image B42024\_09\_12.png**

In step 5, you format each partition in disk 3 using a different filesystem. This includes attempting to create a format for the W: drive with the FAT filesystem, which generates an error since the largest partition you can use with FAT is 4 GB. The output looks like this:

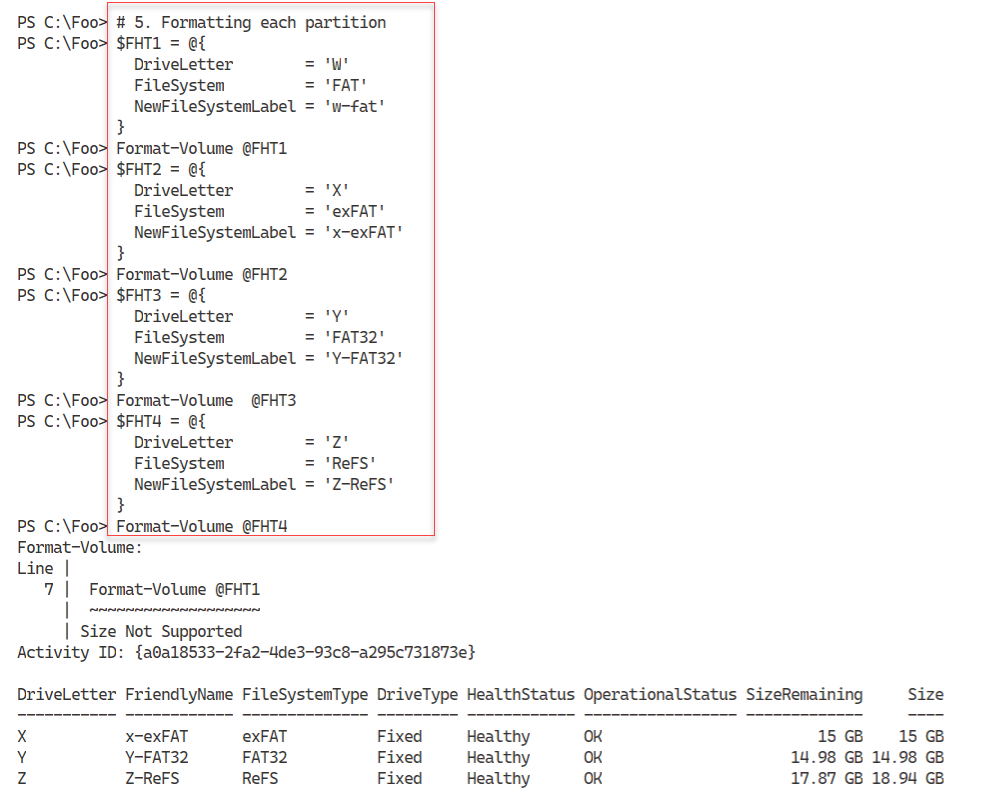


Figure 9.13: Formatting partitions on disk 3

**Insert image B42024\_09\_13.png**

In step 6, you use the Get-Volume command to get all the disk volumes in SRV1, which looks like this:

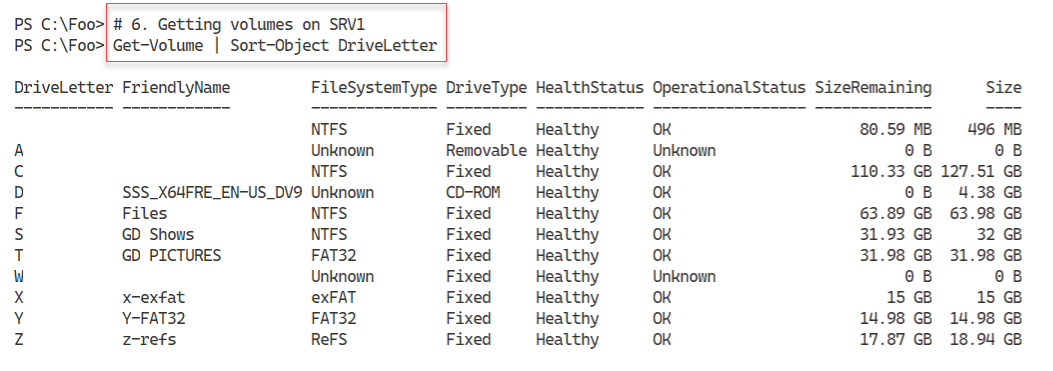


Figure 9.14: Viewing all volumes on SRV1

**Insert image B42024\_09\_14.png**

## There’s more...

In step 5, you attempt to format the W: partition with the FAT filesystem. You created this partition as 15 GB, which is too big for FAT, hence the error message you see. The other three partitions are successfully formatted. Notice that the friendly name for the Y: partition is in uppercase, although you supplied a friendly name with uppercase and lowercase characters. Format-Volume converts the name to all uppercase when formatting the partition with the FAT32 filesystem.

# Exploring providers and the FileSystem provider

One innovation in PowerShell that IT professionals soon learn to love is the PowerShell provider. 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 7.1 comes with the following providers:

* **Registry**: Provides access to registry keys and registry values (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_registry\_provider?view=powershell-7.1)
* **Alias**: Provides access to PowerShell’s command aliases (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_alias\_provider?view=powershell-7.1)
* **Environment**: Provides access to Windows environment variables (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_environment\_provider?view=powershell-7.1)
* **FileSystem**: Provides access to files in a partition (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_filesystem\_provider?view=powershell-7.1)
* **Function**: Provides access to PowerShell’s function definitions (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_function\_provider?view=powershell-7.1)
* **Variable**: Provides access to PowerShell’s variables (https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about\_variable\_provider?view=powershell-7.1)
* **Certificate**: Provides access to the current user and local host’s X.509 digital certificate stores (https://docs.microsoft.com/powershell/module/microsoft.powershell.security/about/about\_certificate\_provider?view=powershell-7.1)
* **WSMan**: Provides a configuration surface that configures the WinRM service (https://docs.microsoft.com/powershell/module/microsoft.wsman.management/about/about\_wsman\_provider?view=powershell-7.1)

With PowerShell providers, you do not need a set of cmdlets for each underlying data store. You can 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 if you have your own data stores, you can also create providers. 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 should have installed AD on this host and configured it as per earlier recipes in Chapter 5, Exploring .NET, and Chapter 6, Managing Active Directory. You used this server in previous recipes in this chapter.

## How to do it...

1. Getting providers

Get-PSProvider

1. Getting registry drives

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 from the registry

(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 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

$HF = Get-PSProvider -PSProvider FileSystem |

  Select-Object -ExpandProperty Home

"Home folder for SRV1 is [$HF]"

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

$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. Checking on available functions

Get-Item Variable:Function\*

1. Getting trusted root certificates for the local user

Get-ChildItem -Path Cert:\CurrentUser\TrustedPublisher

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

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 information

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

## How it works...

In step 1, you use the Get-PSProvider cmdlet to return the providers currently loaded in SRV1, with output like this:

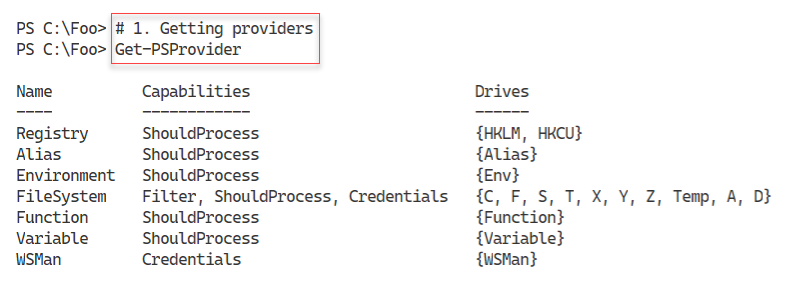


Figure 9.15: Viewing providers on SRV1

**Insert image B42024\_09\_15.png**

You can create a PowerShell provider drive within any provider (using New-PSDrive). To see the drives you have defined on SRV1 that use the Registry provider, in step 2, you use the Get-PSDrive command with output like this:

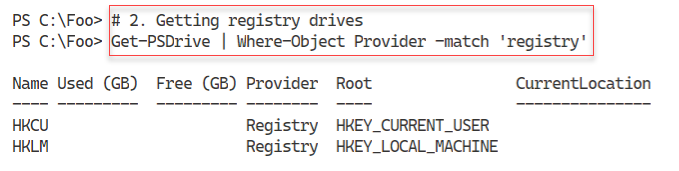


Figure 9.16: Getting registry drives

**Insert image B42024\_09\_16.png**

In step 3, you use the Registry provider and the HKLM: drive to get details of the current version of Windows, with output like this:

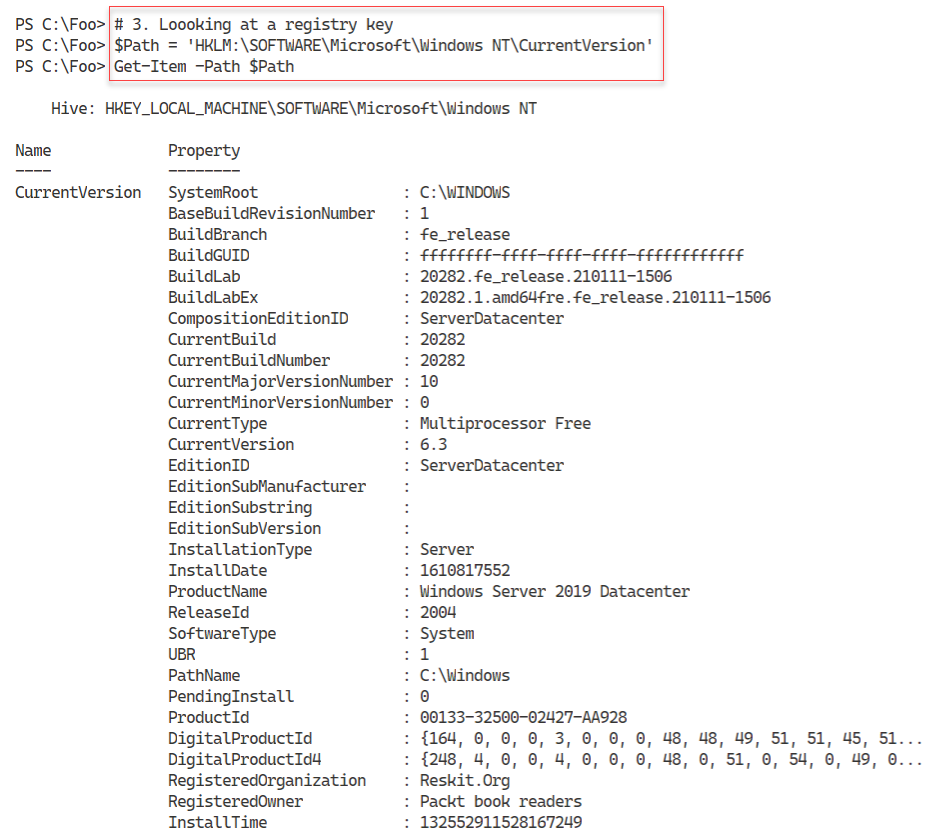


Figure 9.17: Examining a registry key

**Insert image B42024\_09\_17.png**

In step 4, you use the Get-ItemProperty cmdlet to get and display the registered owner of SRV1. Assuming that you have used the Reskit build scripts mentioned in the book's preface, the output of this step looks like this:

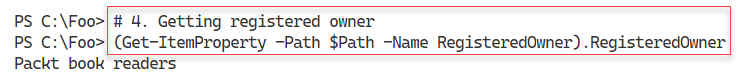


Figure 9.18: Viewing a registered owner on SRV2

**Insert image B42024\_09\_18.png**

By default, the Alias: drive contains all the aliases you define within a PowerShell session. In step 5, you retrieve and count the aliases in the Alias: drive, with output like this:

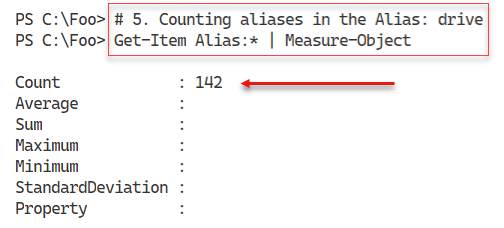


Figure 9.19: Counting aliases on SRV1

**Insert image B42024\_09\_19.png**

You can use the Alias: drive to discover aliases for a particular command. In step 6, you discover the aliases defined for the Remove-Item cmdlet. The output of this step looks like this:

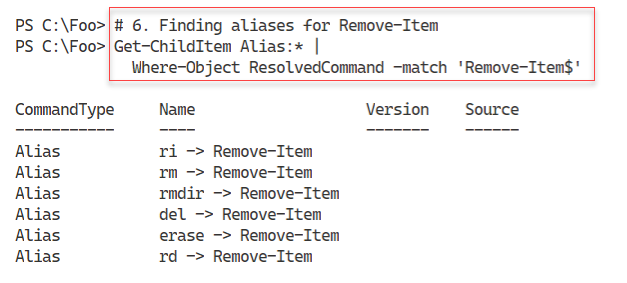


Figure 9.20: Finding aliases for Remove-Item

**Insert image B42024\_09\_20.png**

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

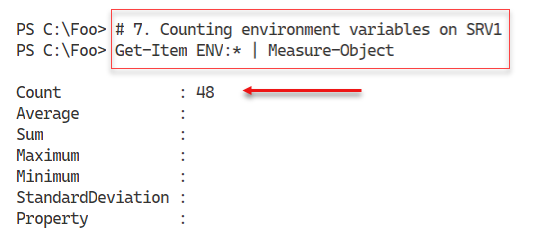


Figure 9.21: Counting environment variables on SRV1

**Insert image B42024\_09\_21.png**

In step 8, you use the ENV: drive to get the environment variable holding the Windows installation folder (typically, C:\Windows, but you have options). The output of this step looks like this:

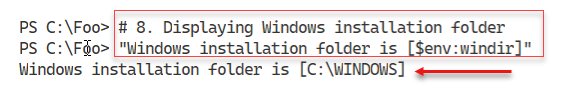


Figure 9.22: Getting the Windows installation folder

**Insert image B42024\_09\_22.png**

In step 9, you discover the FileSystem provider’s drives. These drives, shown in the following output, include the drives (partitions/volumes) you created in earlier recipes and look like this:

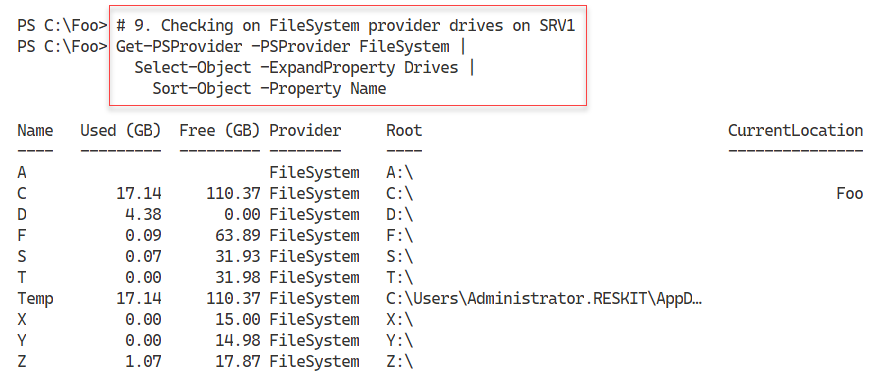


Figure 9.23: Getting Filesystem provider drives

**Insert image B42024\_09\_23.png**

Each provider enables you to define a “home drive.” If you set a home drive, you can use the Set‑Location command and specify a path of “~” to move to that home drive. You set the home drive for the filesystem provider in the $Profile file you set up in Chapter 1, Installing and Configuring PowerShell. In step 10, you get the home drive for the FileSystem provider. The output of this step looks like this:

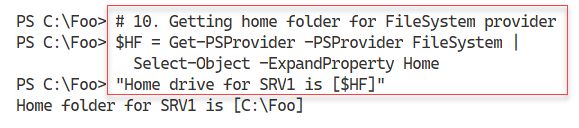


Figure 9.24: Getting the Filesystem provider home folder

**Insert image B42024\_09\_24.png**

In step 11, you remove all modules, and then get and count the functions in the Function: drive, with output like this:

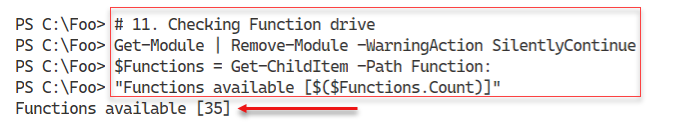


Figure 9.25: Getting and counting the functions available

**Insert image B42024\_09\_25.png**

In step 12, you create a simple function, which generates no output. In step 13, you re-check the Function: drive to discover your new function. The output of this step looks like this:

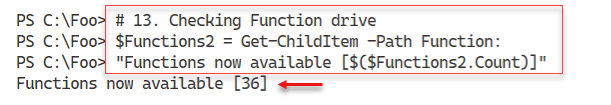


Figure 9.26: Getting and counting functions available

**Insert image B42024\_09\_26.png**

In step 14, you view the function definition for the Get-HelloWorld function contained in the Function: drive. The output looks like this:

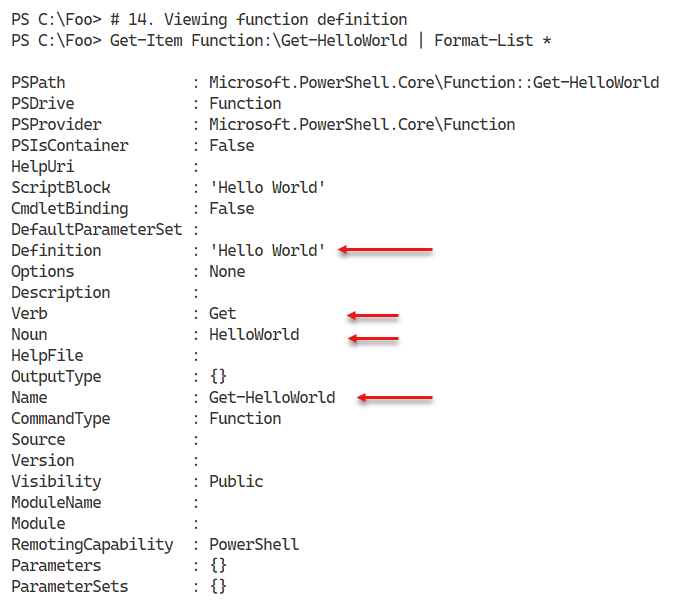


Figure 9.27: Getting function definition from the Function: drive

**Insert image B42024\_09\_27.png**

In step 15, you get and count the variables available in the Variable: drive. The output looks like this:

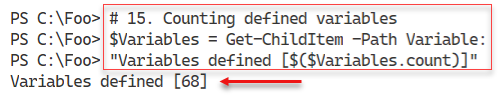


Figure 9.28: Getting and counting variables in the Variable: drive

**Insert image B42024\_09\_28.png**

In step 11 and step 13, you created two variables ($Functions and $Functions2). In step 16, specifying a wild card path, you search for variables in the Variable: drive that begin with “Function” with output like this:

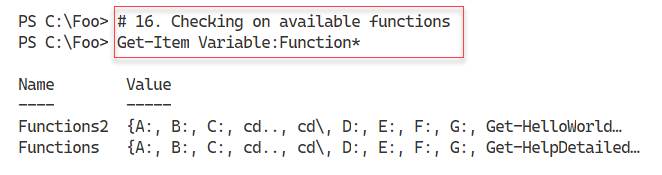


Figure 9.29: Getting variables, starting with “Function”

**Insert image B42024\_09\_29.png**

In step 17, you get the certificate from the current users’ trusted publisher store, with output like this:

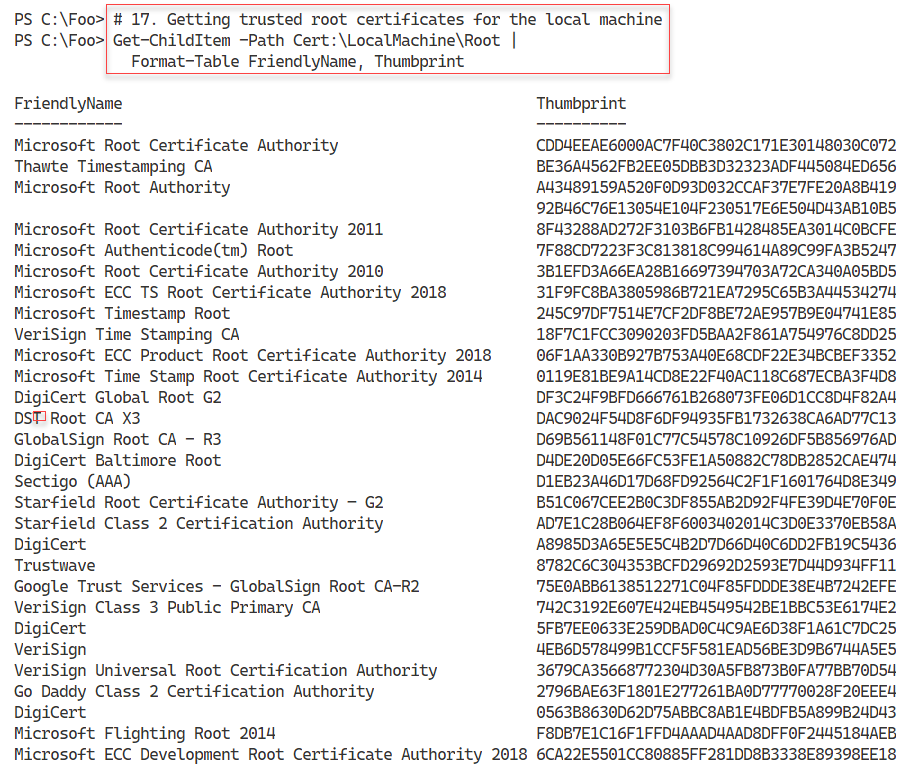


Figure 9.30: Getting certificates from the current user’s trusted publisher certificate store

**Insert image B42024\_09\_30.png**

The WinRM service implements PowerShell remoting. You configure the client and server sides of WinRM 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 18. The output looks like this:

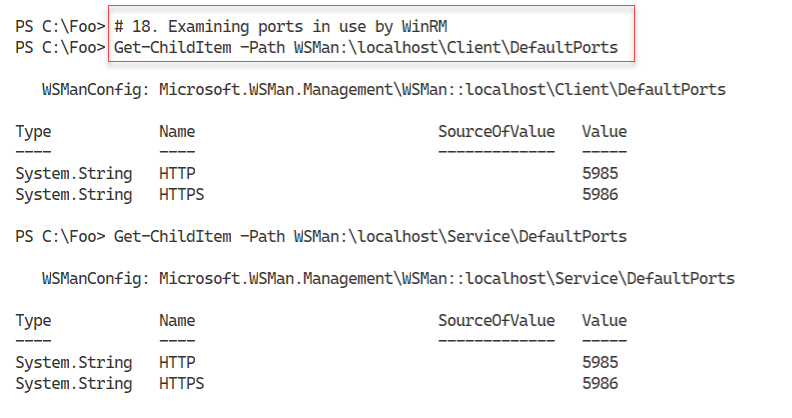


Figure 9.31: Getting WSMan service ports

**Insert image B42024\_09\_31.png**

In step 19, you set WinRM to trust any host by setting the TrustedHosts item in the WSMan drive. There is no output from this step.

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:

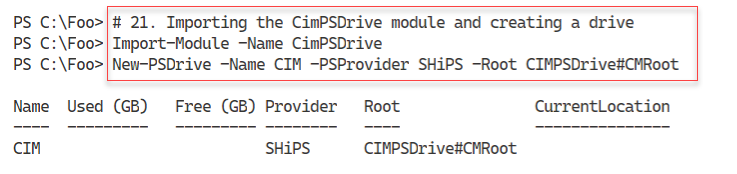


Figure 9.32: Importing the CimPSDrive module and creating a drive

**Insert image B42024\_09\_32.png**

In step 22, you use the CIM: drive to view the System BIOS details from WMI. The output looks like this:

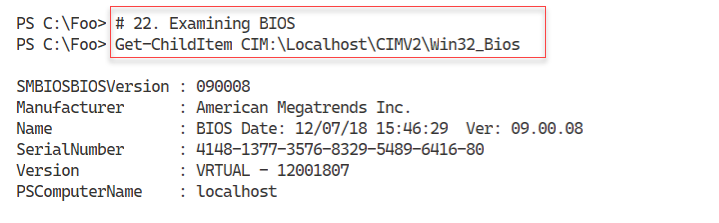


Figure 9.33: Using the CIM: drive

**Insert image B42024\_09\_33.png**

## There’s more...

In step 4, you use the registry provider to return the registered owner of this system. This value was set when you installed Windows. If you use the Resource Kit build scripts on GitHub, the unattended XML file provides a username and organization. Of course, you can change this in the XML or subsequently.

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 its entry (Remove-Item -Path Function:<function to remove>).

With step 17, you view the trusted root CA certificates in the local machine’s certificate store. These root certificates are maintained by Microsoft, 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://docs.microsoft.com/security/trusted-root/program-requirements.

In step 19, you set the WinRM TrustedHosts item to "\*". This 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. This has security implications – you should be careful about when and where you modify this setting.

The SHiPS framework, which you install in step 20, is a module that helps you to develop a provider. 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 more in-depth explanation of the SHiPS framework, see <https://4sysops.com/archives/create-a-custom-powershell-provider/>. This framework can be useful in enabling you to create new providers to unlock data in your organization.

# 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. With SR, you replicate all the files in one volume, for example, the F: drive, to a disk on another host, for example, SRV2. After setting up the SR partnership, as you update the F: drive, Windows automatically updates the target drive on SRV2, although you cannot see the files while Windows is replicating the volume (from SRV1 to SRV2). An SR partnership also requires a drive on the source and destination hosts for internal logging. SR is useful for maintaining a complete replica of one or more disks, typically for disaster recovery.

## Getting ready

You run this recipe on SRV1, a domain-joined host in the Reskit.Org domain, after adding and configuring additional virtual disks to this host. You must have installed PowerShell 7 and VS Code on this host. This recipe explicitly uses the F: drive you created earlier on SRV1 in Managing physical disks and volumes, plus a new G: drive, which you create in disk number 2 (and the corresponding disks on SRV2).

## How to do it...

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

$Part = Get-Partition -DriveLetter F

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

1. Creating F: volume on SRV2

$SB = {

  $NVHT = @{

   DiskNumber   =  $using:Part.DiskNumber

    FriendlyName = 'Files'

    FileSystem   = 'NTFS'

    DriveLetter  = 'F'

  }

  New-Volume @NVHT

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB

1. Creating content on F: on SRV1

1..100 | ForEach-Object {

  $NF = "F:\CoolFolder$\_"

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

  1..100 | ForEach {

    $NF2 = "$NF\CoolFile$\_"

    "Cool File" | Out-File -PSPath $NF2

  }

}

1. Showing what is on F: locally

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

1. Examining the same drives remotely on SRV2

$SB2 = {

  Get-ChildItem -Path F:\ -Recurse |

    Measure-Object

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB2

1. Adding the Storage Replica feature to SRV1

Add-WindowsFeature -Name Storage-Replica | 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 PowerShell

1. Restarting SRV1 to finish the installation process

Restart-Computer

1. Creating a G: volume in disk 2 on SRV1

$SB4 = {

  $NVHT = @{

   DiskNumber   =  2

   FriendlyName = 'SRLOGS'

   DriveLetter  = 'G'

  }

  Clear-Disk -Number 2 -RemoveData -Confirm:$False

  Initialize-Disk -Number 2 | Out-Null

  New-Volume @NVHT

}

Invoke-Command -ComputerName SRV1 -ScriptBlock $SB4

1. Creating a G: volume on SRV2

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB4

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 group

$SRHT =  @{

  SourceComputerName       = 'SRV1'

  SourceRGName             = 'SRV1RG'

  SourceVolumeName         = 'F:'

  SourceLogVolumeName      = 'G:'

  DestinationComputerName  = 'SRV2'

  DestinationRGName        = 'SRV2RG'

  DestinationVolumeName    = 'F:'

  DestinationLogVolumeName = 'G:'

  LogSizeInBytes           = 2gb

}

New-SRPartnership @SRHT

1. Examining the volumes on SRV2

$SB5 = {

  Get-Volume |

    Sort-Object -Property DriveLetter |

      Format-Table

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB5

1. Reversing the replication

$SRHT2 = @{

  NewSourceComputerName   = 'SRV2'

  SourceRGName            = 'SRV2RG'

  DestinationComputerName = 'SRV1'

  DestinationRGName       = 'SRV1RG'

  Confirm                 = $false

}

Set-SRPartnership @SRHT2

1. Viewing the SR partnership

Get-SRPartnership

1. Examining the files remotely on SRV2

$SB6 = {

  Get-ChildItem -Path F:\ -Recurse |

    Measure-Object

}

Invoke-Command -ComputerName SRV2 -ScriptBlock $SB6

## How it works...

In step 1, you get and display the disk number of the disk holding the F: volume on SRV1. The output looks like this:

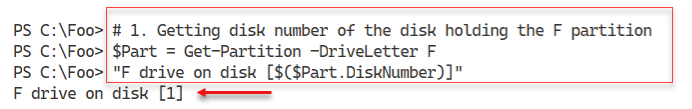


Figure 9.34: Getting the F: drive on SRV1

**Insert image B42024\_09\_34.png**

In step 2, you create an F: volume on SRV2, which looks like this:

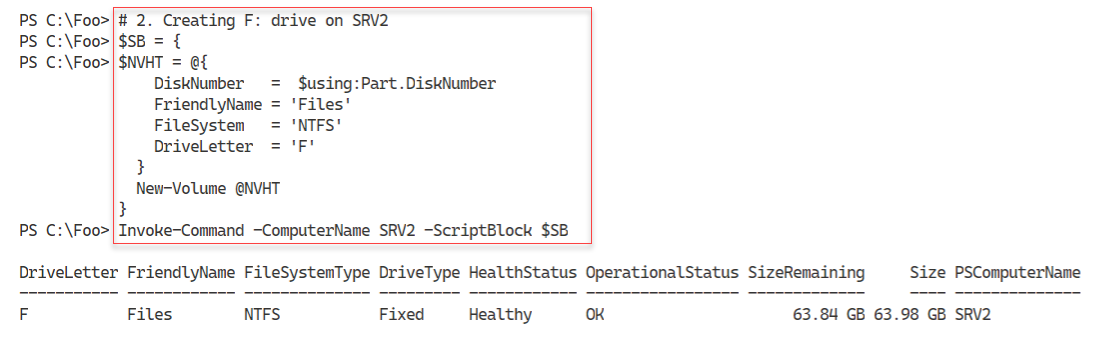


Figure 9.35: Creating F: on SRV2

**Insert image B42024\_09\_35.png**

In step 3, you create content in the F: drive on SRV1 by creating 100 folders and adding 100 files to each of those folders. This step creates no output. In step 4, you use Measure-Object to count the number of files and folders in the F: drive on SRV1, which looks like this:

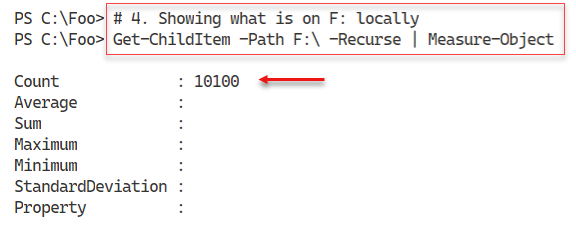


Figure 9.36: Measuring the files and folders on F: on SRV1

**Insert image B42024\_09\_36.png**

In step 5, you examine the F: drive on SRV2, which looks like this:

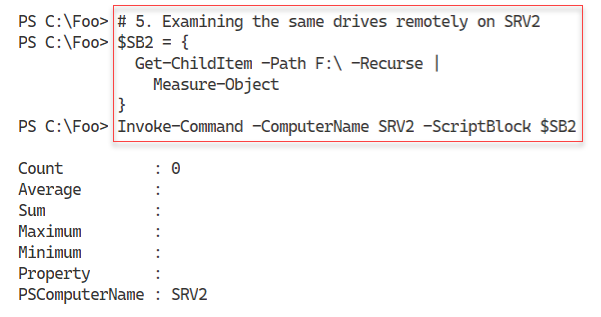


Figure 9.37: Measuring the files and folders on F: on SRV2

**Insert image B42024\_09\_37.png**

In step 6, you add the Storage Replica feature to SRV1, producing no output to the console. In step 7, you add the Storage Replica feature to SRV2, with output that looks like this:

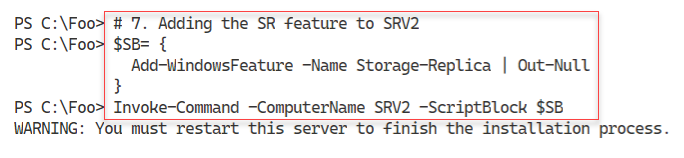


Figure 9.38: Adding Storage Replica to SRV2

**Insert image B42024\_09\_38.png**

You have to reboot SRV2 to complete Storage Replica’s installation, which you do in step 8. In step 9, you reboot SRV1, which completes the process of installing SR on both hosts.

After you reboot SRV2 (and SRV1), in step 10, you create a new volume on SRV1. This volume is to hold SR internal log files. The output looks like this:

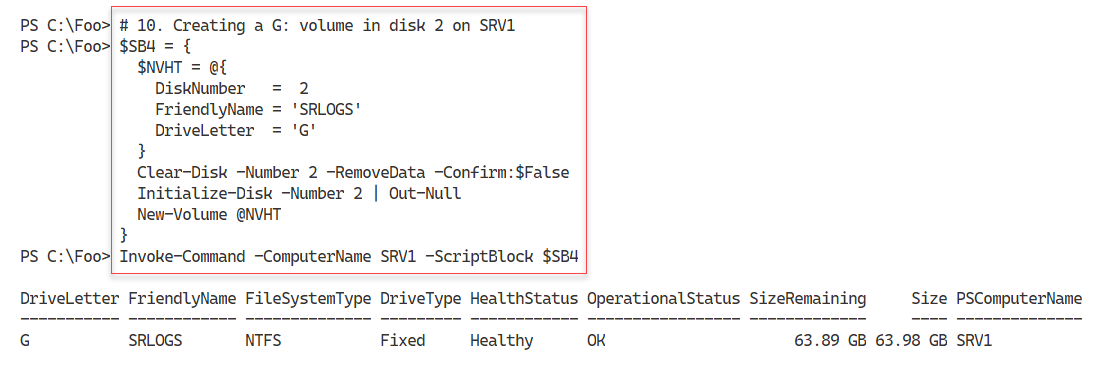


Figure 9.39: Adding a G: drive to SRV1

**Insert image B42024\_09\_39.png**

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

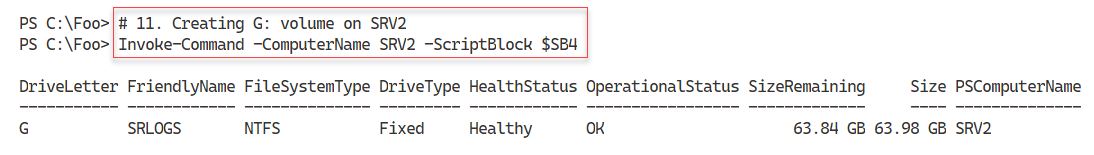


Figure 9.40: Adding a G: drive to SRV2

**Insert image B42024\_09\_40.png**

In step 12, you view the volumes on SRV1 with output like this:

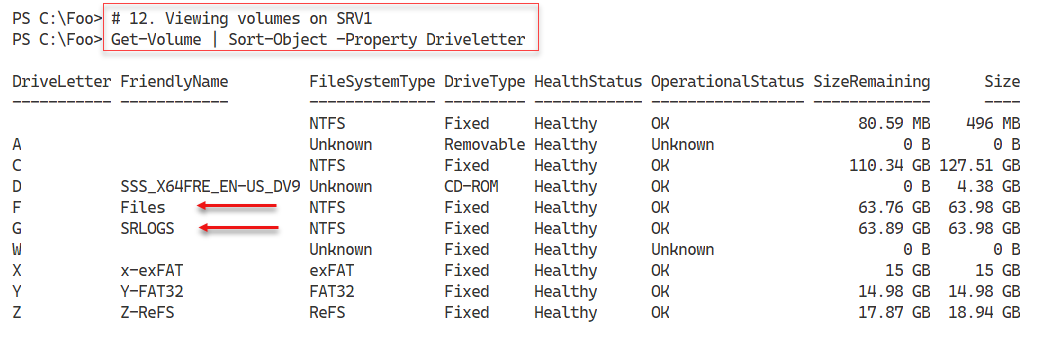


Figure 9.41: Viewing volumes on SRV1

**Insert image B42024\_09\_41.png**

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

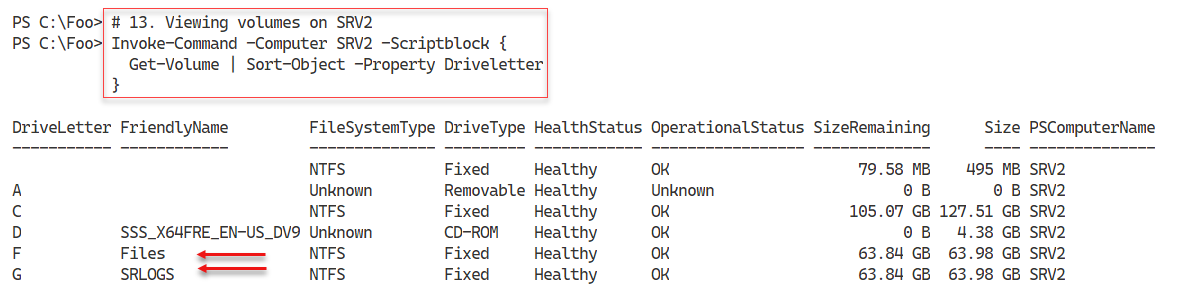


Figure 9.42: Viewing volumes on SRV2

**Insert image B42024\_09\_42.png**

In step 14, you create an SR replica group to replicate all the files on the F: drive in SRV1 to the F: drive on SRV2, using the G: drive on both hosts for internal SR logging. The output of this step is like this:

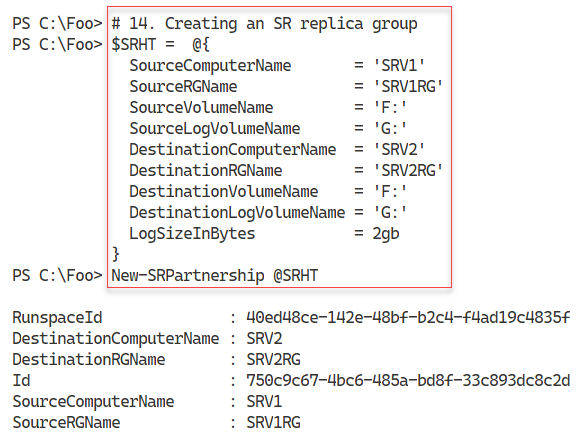


Figure 9.43: Creating a replica group

**Insert image B42024\_09\_43.png**

In step 15, you re-examine the volumes on SRV2, which looks like this:

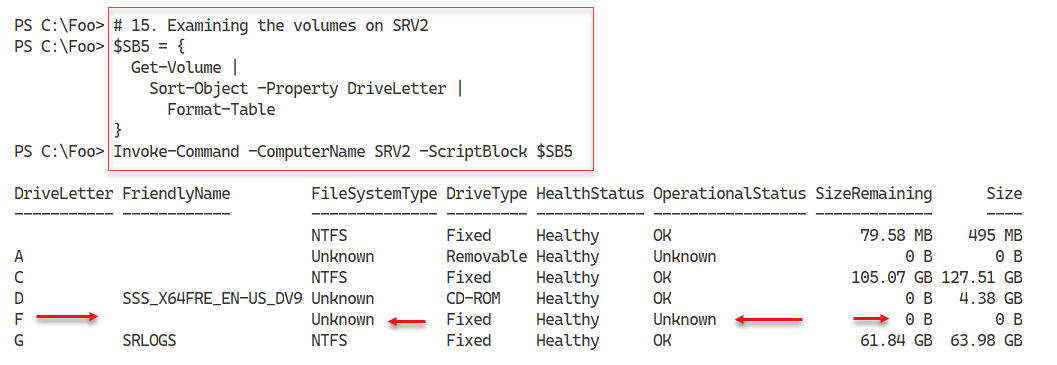


Figure 9.44: Viewing volumes on SRV2

**Insert image B42024\_09\_44.png**

With step 16, you reverse the replication – the files on SRV2 now get replicated to SRV1. This step creates no output. In step 17, you re-view the SR partnership with output like this:

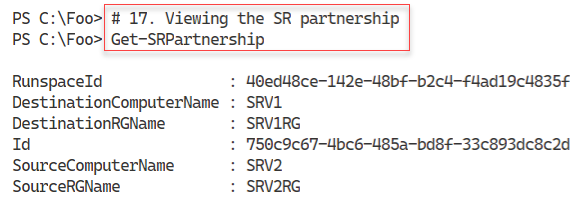


Figure 9.45: Viewing the SR partnership

**Insert image B42024\_09\_45.png**

In step 18, you view the files on the F: drive on SRV2, which looks like this:

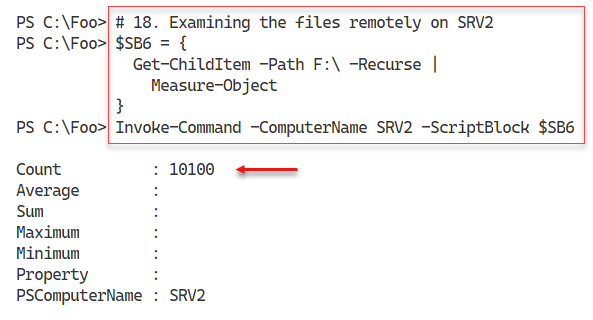


Figure 9.46: Viewing files on F: on SRV2

**Insert image B42024\_09\_46.png**

## There’s more...

In the first five steps in this recipe, you create 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 other files you want to synchronize.

In step 9, 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 SRV2, you 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 volume, it could take quite a while. You can use the (Get-SRGroup).Replicas.ReplicationStatus command to check the status of the initial replication.

# Deploying Storage Spaces

Storage Spaces is a technology in Windows 10 and Windows Server that implements software RAID. You can add multiple physical drives into your server or workstation, and create fault-tolerant volumes for your host. You can read more about Storage Spaces at https://docs.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** (aka **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 makes use of five of the virtual disks you added to SRV1 at the start of the Managing physical disks and volumes recipe.

## How to do it...

1. Viewing disks available for pooling

$Disks = Get-PhysicalDisk -CanPool $true

$Disks

1. Creating a storage pool

$SPHT = @{

    FriendlyName                 = 'RKSP'

    StorageSubsystemFriendlyName = "Windows Storage\*"

    PhysicalDisks                = $Disks

}

New-StoragePool @SPHT

1. Creating a mirrored hard disk named Mirror1

$VDHT1 = @{

  StoragePoolFriendlyName   = 'RKSP'

  FriendlyName              = 'Mirror1'

  ResiliencySettingName     = 'Mirror'

  Size                      = 8GB

  ProvisioningType          = 'Thin'

}

New-VirtualDisk @VDHT1

1. Creating a three-way mirrored disk named Mirror2

$VDHT2 = @{

  StoragePoolFriendlyName    = 'RKSP'

  FriendlyName               = 'Mirror2'

  ResiliencySettingName      = 'Mirror'

  NumberOfDataCopies         = 3

  Size                       = 8GB

  ProvisioningType           = 'Thin'

}

New-VirtualDisk @VDHT2

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 SRV2

Get-Volume | Sort-Object -Property DriveLetter

## How it works...

In step 1, you examine the disks available in SRV1 at the start of this recipe.

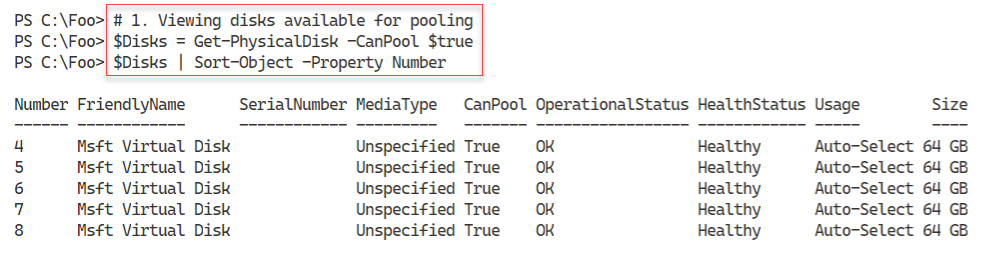


Figure 9.47: Viewing disks available for pooling on SRV1

**Insert image B42024\_09\_47.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:

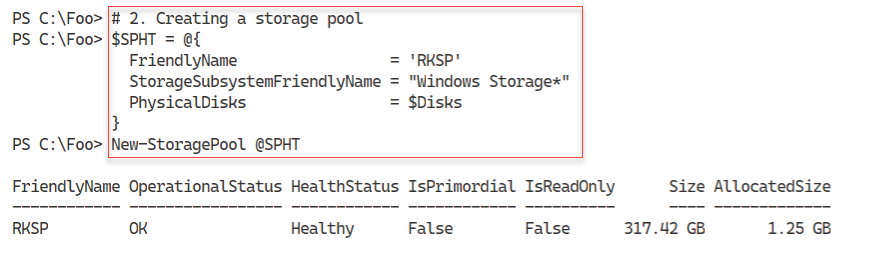


Figure 9.48: Creating a new storage pool

**Insert image B42024\_09\_48.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:

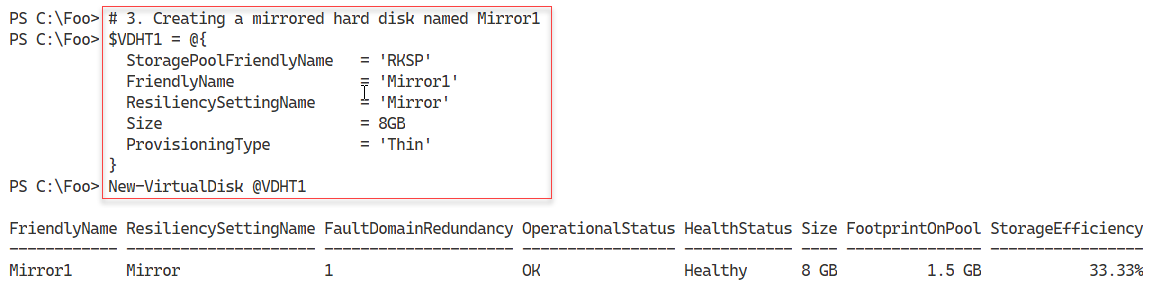


Figure 9.49: Creating a mirrored storage space

**Insert image B42024\_09\_49.png**

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

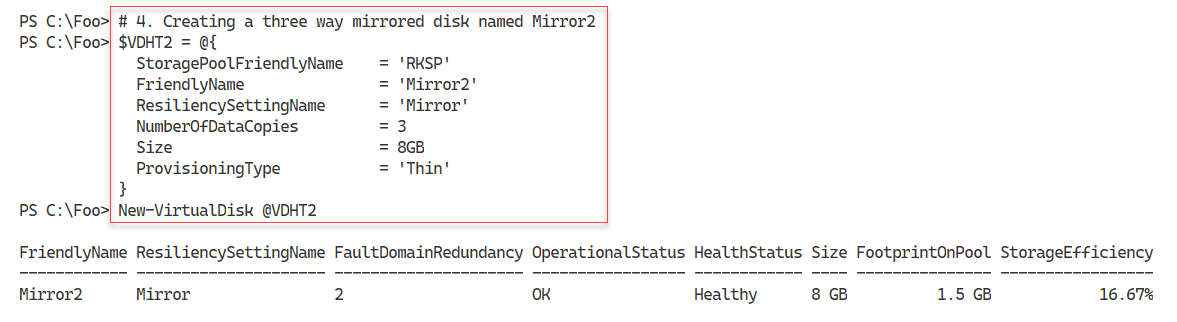


Figure 9.50: Creating a three-way mirrored storage space

**Insert image B42024\_09\_50.png**

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

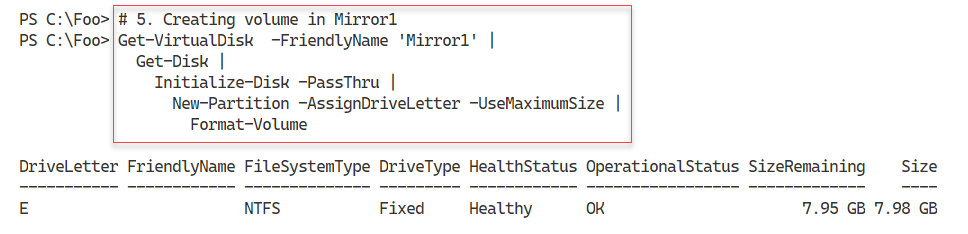


Figure 9.51: Creating a disk inside the Mirror1 Storage Space

**Insert image B42024\_09\_51.png**

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

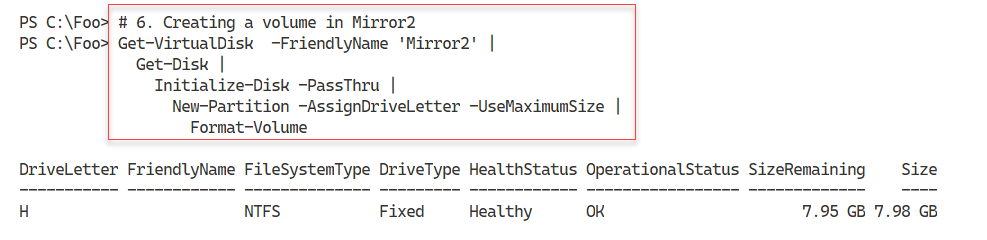


Figure 9.52: Creating a disk inside the Mirror2 Storage Space

**Insert image B42024\_09\_52.png**

In the final step in this recipe, step 7, you use the Get-Volume command to view all the volumes available in SRV2, with output like this:

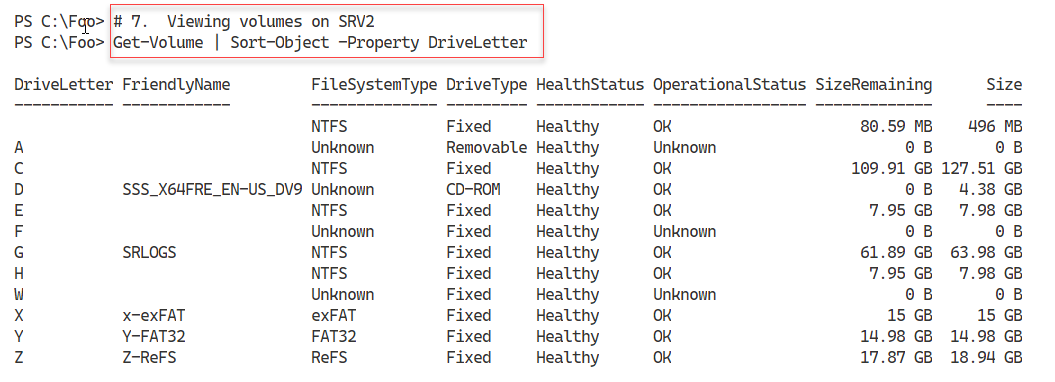


Figure 9.53: Viewing disks available on SRV2

**Insert image B42024\_09\_53.png**

## There’s more...

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