# 7

# Managing Storage

This chapter covers the following recipes:

- Managing Disks
- Managing File Systems
- Exploring PowerShell Providers and the FileSystem 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 NVMe 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 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 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/`.

Once you create a volume on a storage device, you can format the volume. Windows supports five file systems you can use: ReFS, NTFS, exFAT, UDF, and FAT32. For details on 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, which 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 file systems, see `https://www.iperiusbackup.net/en/refs-vs-ntfs-differences-and-performance-comparison-when-to-use/`. You'll examine partitioning and formatting volumes in the *Managing Disks* 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** (**ACEs**). 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'll 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 Datacenter edition. In *Managing Storage Replica*, you'll 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'll investigate in *Deploying Storage Spaces*.

## The systems used in the chapter

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

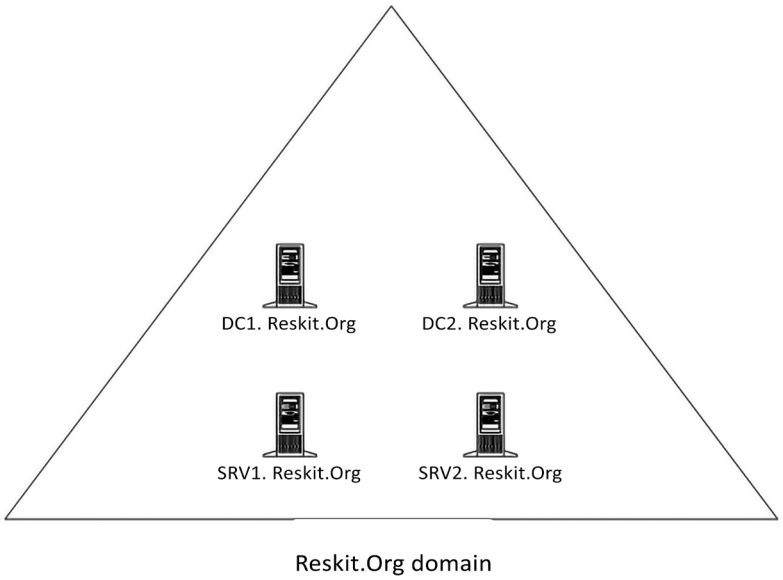


*Figure 7.1: Hosts in use for this chapter*

# 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 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 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'll 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
```

2. Get first usable disk

```
$Disk = Get-Disk |
           Where-Object PartitionStyle -eq Raw |
             Select-Object -First 1
$Disk | Format-List
```

3. Initializing the first available disk

```
$Disk |
  Initialize-Disk -PartitionStyle GPT
```

4. Re-displaying all disks in SRV1

```
Get-Disk
```

5. Viewing volumes on SRV1

```
Get-Volume | Sort-Object -Property DriveLetter
```

6. Viewing partitions on SRV1

```
Get-Partition
```

7. Examining details of a volume

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

8. Examining details of a partition

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

9. 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
```

10. 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:

```
PS C:\Foo> # 1. Displaying the disks on SRV1
PS C:\Foo> Get-Disk

Number Friendly Name     Serial Number HealthStatus OperationalStatus Total Size Partition Style
------ -------------     ------------- ------------ ----------------- ---------- ---------------
0      Msft Virtual Disk               Healthy      Online               128 GB GPT
1      Msft Virtual Disk               Healthy      Offline               64 GB RAW
2      Msft Virtual Disk               Healthy      Offline               64 GB RAW
3      Msft Virtual Disk               Healthy      Offline               64 GB RAW
4      Msft Virtual Disk               Healthy      Offline               64 GB RAW
5      Msft Virtual Disk               Healthy      Offline               64 GB RAW
6      Msft Virtual Disk               Healthy      Offline               64 GB RAW
7      Msft Virtual Disk               Healthy      Offline               64 GB RAW
8      Msft Virtual Disk               Healthy      Offline               64 GB RAW
```

*Figure 7.2: Displaying disk details*

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

```
PS C:\Foo> # 2. Get first usable disk
PS C:\Foo> $Disk = Get-Disk |
              Where-Object PartitionStyle -eq Raw |
                Select-Object -First 1
PS C:\Foo> $Disk | Format-List

UniqueId           : 60022480E8258902AEAD5E128062318A
Number             : 1
Path               : \\?\scsi#disk&ven_msft&prod_virtual_disk#
                     5&2132ca1&0&000000#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
Manufacturer       : Msft
Model              : Virtual Disk
SerialNumber       :
Size               : 64 GB
AllocatedSize      : 0
LogicalSectorSize  : 512
PhysicalSectorSize : 4096
NumberOfPartitions : 0
PartitionStyle     : RAW
IsReadOnly         : True
IsSystem           : False
IsBoot             : False
```

*Figure 7.3: Displaying disk details of the first usable disk*

In *step 3*, you initialize this disk using the GPT partition scheme. This step produces no output.

In *step 4*, you view the disks again on `SRV1`, with output like this:

```
PS C:\Foo> # 4. Re-displaying all disks in SRV1
PS C:\Foo> Get-Disk

Number Friendly Name     Serial Number HealthStatus OperationalStatus Total Size Partition Style
------ -------------     ------------- ------------ ----------------- ---------- ---------------
0      Msft Virtual Disk               Healthy      Online               128 GB GPT
1      Msft Virtual Disk               Healthy      Online                64 GB GPT
2      Msft Virtual Disk               Healthy      Offline               64 GB RAW
3      Msft Virtual Disk               Healthy      Offline               64 GB RAW
4      Msft Virtual Disk               Healthy      Offline               64 GB RAW
5      Msft Virtual Disk               Healthy      Offline               64 GB RAW
6      Msft Virtual Disk               Healthy      Offline               64 GB RAW
7      Msft Virtual Disk               Healthy      Offline               64 GB RAW
8      Msft Virtual Disk               Healthy      Offline               64 GB RAW
```

*Figure 7.4: Viewing existing disk volumes on SRV1*

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

```
PS C:\Foo> # 5. Viewing volumes on SRV1
PS C:\Foo> Get-Volume | Sort-Object -Property DriveLetter

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining      Size
----------- ------------ -------------- --------- ------------ ----------------- -------------      ----
                         FAT32          Fixed     Healthy      OK                     67.25 MB     96 MB
C                        NTFS           Fixed     Healthy      OK                   111.25 GB 127.9 GB
```

*Figure 7.5: Viewing existing volumes on SRV1*

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

```
PS C:\Foo> # 6. Viewing partitions on SRV1
PS C:\Foo> Get-Partition

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&1ebf9ebb&0&000000#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber DriveLetter Offset           Size Type
--------------- ----------- ------           ---- ----
1               C            1048576      127.9 GB Basic
2                           137333047296   100 MB System

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000000#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber DriveLetter Offset     Size Type
--------------- ----------- ------     ---- ----
1                           17408  15.98 MB Reserved
```

*Figure 7.6: Viewing all partitions on SRV1*

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

```
PS C:\Foo> # 7. Examining details of a volume
PS C:\Foo> Get-Volume | Select-Object -First 1 | Format-List

ObjectId             : {1}\\SRV1\root/Microsoft/Windows/Storage/Providers_v2\
                       WSP_Volume.ObjectId="{004bc44d-ede2-11ec-a79c-806e6f6e6963}:
                       VO:\\?\Volume{ebeaff83-af69-11ec-be24-5cf37091be18}\"
PassThroughClass     :
PassThroughIds       :
PassThroughNamespace :
PassThroughServer    :
UniqueId             : \\?\Volume{ebeaff83-af69-11ec-be24-5cf37091be18}\
AllocationUnitSize   : 4096
DedupMode            : NotAvailable
DriveLetter          : C
DriveType            : Fixed
FileSystem           : NTFS
FileSystemLabel      :
FileSystemType       : NTFS
HealthStatus         : Healthy
OperationalStatus    : OK
Path                 : \\?\Volume{ebeaff83-af69-11ec-be24-5cf37091be18}\
Size                 : 137331994624
SizeRemaining        : 119453777920
PSComputerName       :
```

*Figure 7.7: Viewing Volume properties*

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

```
PS C:\Foo> # 8. Examining details of a partition
PS C:\Foo> Get-Partition | Select-Object -First 1 | Format-List

UniqueId             : {00000000-0000-0000-0000-100000000000}6002248005588E716BE40737CCDCDD7C
AccessPaths          : {C:\, \\?\Volume{ebeaff83-af69-11ec-be24-5cf37091be18}\}
DiskNumber           : 0
DiskPath             : \\?\scsi#disk&ven_msft&prod_virtual_disk#5&1ebf9ebb&0&000000#
                       {53f56307-b6bf-11d0-94f2-00a0c91efb8b}
DriveLetter          : C
Guid                 : {ebeaff83-af69-11ec-be24-5cf37091be18}
IsActive             : False
IsBoot               : True
IsHidden             : False
IsOffline            : False
IsReadOnly           : False
IsShadowCopy         : False
IsDAX                : False
IsSystem             : False
NoDefaultDriveLetter : False
Offset               : 1048576
OperationalStatus    : Online
PartitionNumber      : 1
Size                 : 127.9 GB
Type                 : Basic
```

*Figure 7.8: Viewing partition details*

In *step 9*, you get the next unpartitioned disk and initialize it using the 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:

```
PS C:\Foo> # 10. Examining disks in SRV1
PS C:\Foo> Get-Disk

Number Friendly Name     Serial Number HealthStatus OperationalStatus Total Size Partition Style
------ -------------     ------------- ------------ ----------------- ---------- ---------------
0      Msft Virtual Disk               Healthy      Online                128 GB GPT
1      Msft Virtual Disk               Healthy      Online                 64 GB GPT
2      Msft Virtual Disk               Healthy      Online                 64 GB MBR
3      Msft Virtual Disk               Healthy      Offline                64 GB RAW
4      Msft Virtual Disk               Healthy      Offline                64 GB RAW
5      Msft Virtual Disk               Healthy      Offline                64 GB RAW
6      Msft Virtual Disk               Healthy      Offline                64 GB RAW
7      Msft Virtual Disk               Healthy      Offline                64 GB RAW
8      Msft Virtual Disk               Healthy      Offline                64 GB RAW
```

*Figure 7.9: Viewing disks on SRV1*

## 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, see `https://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, also in the *Managing Disks* recipe, you initialized the first two. In this recipe, you'll create specific volumes on these two disks.

## How to do it...

1. Getting second disk

```
$Disk = Get-Disk | Select-Object -Skip 1 -First 1
$Disk | Format-List
```

2. Creating a new volume in this disk

```
$NewVolumeHT1   = @{
  DiskNumber    = $Disk.Disknumber
  DriveLetter  = 'S'
  FriendlyName = 'Files'
}
New-Volume @NewVolumeHT1
```

3. Getting next available disk to use on `SRV1`

```
$Disk2 = Get-Disk |
           Where-Object PartitionStyle -eq 'MBR' |
             Select-Object -First 1
$Disk2 | Format-List
```

4. 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
```

5. 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
```

6. 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:

```
PS C:\Foo> # 1. Getting the second disk
PS C:\Foo> $Disk = Get-Disk | Select-Object -Skip 1 -First 1
PS C:\Foo> $Disk | Format-List

UniqueId            : 60022480E8258902AEAD5E128062318A
Number              : 1
Path                : \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000000#
                      {53f56307-b6bf-11d0-94f2-00a0c91efb8b}
Manufacturer        : Msft
Model               : Virtual Disk
SerialNumber        :
Size                : 64 GB
AllocatedSize       : 17825792
LogicalSectorSize   : 512
PhysicalSectorSize  : 4096
NumberOfPartitions  : 1
PartitionStyle      : GPT
IsReadOnly          : False
IsSystem            : False
IsBoot              : False
```

*Figure 7.10: Getting the next RAW disk on SRV1*

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

```
PS C:\Foo> # 2. Creating a new volume in this disk
PS C:\Foo> $NewVolumeHT1   = @{
             DiskNumber    = $Disk.Disknumber
             DriveLetter   = 'S'
             FriendlyName = 'Files'
           }
PS C:\Foo> New-Volume @NewVolumeHT1

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size
----------- ------------ -------------- --------- ------------ ----------------- -------------     ----
S           Files        NTFS           Fixed     Healthy      OK                    63.84 GB 63.98 GB
```

*Figure 7.11: Creating the S: volume*

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:

```
PS C:\Foo> # 3. Getting next available disk to use on SRV1
PS C:\Foo> $Disk2 = Get-Disk |
             Where-Object PartitionStyle -eq 'MBR' |
               Select-Object -First 1
PS C:\Foo> $Disk2 | Format-List

UniqueId           : 60022480F74243AB1AD9CFB6E1B06E28
Number             : 2
Path               : \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000001#
                     {53f56307-b6bf-11d0-94f2-00a0c91efb8b}
Manufacturer       : Msft
Model              : Virtual Disk
SerialNumber       :
Size               : 64 GB
AllocatedSize      : 2097152
LogicalSectorSize  : 512
PhysicalSectorSize : 4096
NumberOfPartitions : 0
PartitionStyle     : MBR
IsReadOnly         : False
IsSystem           : False
IsBoot             : False
```

*Figure 7.12: Getting the next disk*

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

```
PS C:\Foo> # 4. Creating 4 new partitions on third (MBR) disk
PS C:\Foo> $UseMaxHT= @{UseMaximumSize = $true}
PS C:\Foo> New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter W -Size 1gb

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000001#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset  Size Type
---------------  ----------- ------  ---- ----
1                W           1048576 1 GB Logical

PS C:\Foo> New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter X -Size 15gb

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000001#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset      Size Type
---------------  ----------- ------      ---- ----
2                X           1074790400 15 GB Logical

PS C:\Foo> New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter Y -Size 15gb

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000001#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset       Size Type
---------------  ----------- ------       ---- ----
3                Y           17180917760 15 GB Logical

PS C:\Foo> New-Partition -DiskNumber $Disk2.DiskNumber -DriveLetter Z @UseMaxHT

   DiskPath: \\?\scsi#disk&ven_msft&prod_virtual_disk#5&2132ca1&0&000001#
             {53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset        Size Type
---------------  ----------- ------        ---- ----
4                Z           33288093696  33 GB Logical
```

*Figure 7.13: Creating partitions on disk 2*

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:

```
PS C:\Foo> # 5. Formatting each volume on this disk
PS C:\Foo> $FormatHT1 = @{
             DriveLetter        = 'W'
             FileSystem         = 'FAT'
             NewFileSystemLabel = 'w-fat'
           }
PS C:\Foo> Format-Volume @FormatHT1

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining        Size
----------- ------------ -------------- --------- ------------ ----------------- -------------        ----
W           W-FAT        FAT            Fixed     Healthy      OK                  1023.69 MB 1023.72 MB

PS C:\Foo> $FormatHT2 = @{
             DriveLetter        = 'X'
             FileSystem         = 'exFAT'
             NewFileSystemLabel = 'x-exFAT'
           }
PS C:\Foo> Format-Volume @FormatHT2

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining  Size
----------- ------------ -------------- --------- ------------ ----------------- -------------  ----
X           x-exFAT      exFAT          Fixed     Healthy      OK                       15 GB 15 GB

PS C:\Foo> $FormatHT3 = @{
             DriveLetter        = 'Y'
             FileSystem         = 'FAT32'
             NewFileSystemLabel = 'Y-FAT32'
           }
PS C:\Foo> Format-Volume  @FormatHT3

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size
----------- ------------ -------------- --------- ------------ ----------------- -------------     ----
Y           Y-FAT32      FAT32          Fixed     Healthy      OK                  14.98 GB 14.98 GB

PS C:\Foo> $FormatHT4 = @{
             DriveLetter        = 'Z'
             FileSystem         = 'ReFS'
             NewFileSystemLabel = 'Z-ReFS'
           }
PS C:\Foo> Format-Volume @FormatHT4

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size
----------- ------------ -------------- --------- ------------ ----------------- -------------     ----
Z           Z-ReFS       ReFS           Fixed     Healthy      OK                  31.81 GB 32.94 GB
```

*Figure 7.14: Formatting partitions on disk 2*

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:

```
PS C:\Foo> # 7. Getting all volumes on SRV1
PS C:\Foo> Get-Volume | Sort-Object DriveLetter

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining       Size
----------- ------------ -------------- --------- ------------ ----------------- -------------       ----
                         FAT32          Fixed     Healthy      OK                    67.25 MB      96 MB
C                        NTFS           Fixed     Healthy      OK                   111.23 GB   127.9 GB
S           Files        NTFS           Fixed     Healthy      OK                    63.84 GB   63.98 GB
W           W-FAT        FAT            Fixed     Healthy      OK                  1023.69 MB 1023.72 MB
X           x-exFAT      exFAT          Fixed     Healthy      OK                       15 GB      15 GB
Y           Y-FAT32      FAT32          Fixed     Healthy      OK                    14.98 GB   14.98 GB
Z           Z-ReFS       ReFS           Fixed     Healthy      OK                    31.81 GB   32.94 GB
```

*Figure 7.15: Viewing volumes on disk 2*

## 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 FileSystem 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`).
- **Environment**: 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
```

2. Getting drives from the registry provider

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

3. Looking at a registry key

```
$Path = 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'
Get-Item -Path $Path
```

4. Getting the registered owner

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

5. Counting aliases in the `Alias:` drive

```
Get-Item Alias:* | Measure-Object
```

6. Finding aliases for `Remove-Item`

```
Get-ChildItem Alias:* |
  Where-Object ResolvedCommand -match 'Remove-Item$'
```

7. Counting environment variables on `SRV1`

```
Get-Item ENV:* | Measure-Object
```

8. Displaying the Windows installation folder

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

9. Checking on `FileSystem` provider drives on `SRV1`

```
Get-PSProvider -PSProvider FileSystem |
  Select-Object -ExpandProperty Drives |
    Sort-Object -Property Name
```

10. Getting home folder for the `FileSystem` provider

```
$HomeFolder = Get-PSProvider -PSProvider FileSystem |
  Select-Object -ExpandProperty Home
$HomeFolder
```

11. Checking `Function` drive

```
Get-Module | Remove-Module -WarningAction SilentlyContinue
$Functions = Get-ChildItem -Path Function:
"Functions available [$($Functions.Count)]"
```

12. Creating a new function

```
Function Get-HelloWorld {'Hello World'}
```

13. Checking the `Function` drive again

```
$Functions2 = Get-ChildItem -Path Function:
"Functions now available [$($Functions2.Count)]"
```

14. Viewing function definition

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

15. Counting the variables available

```
$Variables = Get-ChildItem -Path Variable:
"Variables defined [$($Variables.count)]"
```

16. Getting trusted root certificates for the local machine

```
Get-ChildItem -Path Cert:\LocalMachine\Root |
  Format-Table FriendlyName, Thumbprint
```

17. Examining ports in use by WinRM

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

18. Setting Trusted hosts

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

19. Installing `SHiPS` and `CimPSDrive` modules

```
Install-Module -Name SHiPS, CimPSDrive -Force
```

20. Importing the `CimPSDrive` module and creating a drive

```
Import-Module -Name CimPSDrive
New-PSDrive -Name CIM -PSProvider SHiPS -Root CIMPSDrive#CMRoot
```

21. Examining BIOS using the `CimPSDrive` module

```
Get-ChildItem CIM:\Localhost\CIMV2\Win32_Bios
```

## How it works...

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

```
PS C:\Foo> # 1. Getting PowerShell providers
PS C:\Foo> Get-PSProvider

Name        Capabilities                       Drives
----        ------------                       ------
Registry    ShouldProcess                      {HKLM, HKCU}
Alias       ShouldProcess                      {Alias}
Environment ShouldProcess                      {Env}
FileSystem  Filter, ShouldProcess, Credentials {C, Temp, S, W, X, Y, Z}
Function    ShouldProcess                      {Function}
Variable    ShouldProcess                      {Variable}
Certificate ShouldProcess                      {Cert}
WSMan       Credentials                        {WSMan}
```

*Figure 7.16: Viewing the PowerShell providers on SRV1*

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:

```
PS C:\Foo> # 2. Getting drives from the registry provider
PS C:\Foo> Get-PSDrive | Where-Object Provider -match 'Registry'

Name Used (GB) Free (GB) Provider Root               CurrentLocation
---- --------- --------- -------- ----               ---------------
HKCU                    Registry HKEY_CURRENT_USER
HKLM                    Registry HKEY_LOCAL_MACHINE
```

*Figure 7.17: Viewing the drives in the registry provider*

In *step 4*, you use the registry provider to retrieve the registry holding the registered owner. The Windows installation process writes a value to the registry containing the name of the system's owner. The output of this step looks like this:

```
PS C:\Foo> # 4. Getting registered owner
PS C:\Foo> (Get-ItemProperty -Path $Path -Name RegisteredOwner).RegisteredOwner
Book Readers
```

*Figure 7.18: Obtaining the registered owner via the registry*

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

```
PS C:\Foo> # 5. Counting aliases in the Alias: drive
PS C:\Foo> Get-Item Alias:* | Measure-Object

Count             : 164
Average           :
Sum               :
Maximum           :
Minimum           :
StandardDeviation :
Property          :
```

*Figure 7.19: Counting the number of aliases on SRV1*

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:

```
PS C:\Foo> # 6. Finding aliases for Remove-Item
PS C:\Foo> Get-Childitem Alias:* |
             Where-Object ResolvedCommand -match 'Remove-Item$'

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           ri -> Remove-Item
Alias           rm -> Remove-Item
Alias           rmdir -> Remove-Item
Alias           del -> Remove-Item
Alias           erase -> Remove-Item
Alias           rd -> Remove-Item
```

*Figure 7.20: Discovering aliases to the Remove-Item command*

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

```
PS C:\Foo> # 7. Counting environment variables on SRV1
PS C:\Foo> Get-Item ENV:* | Measure-Object

Count             : 45
Average           :
Sum               :
Maximum           :
Minimum           :
StandardDeviation :
Property          :
```

*Figure 7.21: Counting the environment variables on SRV1*

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

```
PS C:\Foo> # 8. Displaying Windows installation folder
PS C:\Foo> "Windows installation folder is [Senv:windir]"
Windows installation folder is [C:\WINDOWS]
```

*Figure 7.22: Counting the environment variables*

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 FileSystem provider, with output like this:

```
PS C:\Foo> # 9. Checking on FileSystem provider drives on SRV1
PS C:\Foo> Get-PSProvider -PSProvider FileSystem |
             Select-Object -ExpandProperty Drives |
               Sort-Object -Property Name

Name Used (GB) Free (GB) Provider   Root                                                CurrentLocation
---- --------- --------- --------   ----                                                ---------------
C         16.68    111.22 FileSystem C:\                                                         Foo
S          0.14     63.84 FileSystem S:\
Temp      16.68    111.22 FileSystem C:\Users\administrator.RESKIT\AppData\Local\Temp\
W          0.00      1.00 FileSystem W:\
X          0.00     15.00 FileSystem X:\
Y          0.00     14.98 FileSystem Y:\
Z          1.16     31.78 FileSystem Z:\
```

*Figure 7.23: Getting drives in the FileSystem provider*

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 FileSystem provider (in *Installing and Configuring PowerShell*). In *step 10*, you get the home drive for the FileSystem provider. The output of this step looks like this:

```
PS C:\Foo> # 10. Getting home folder for FileSystem provider
PS C:\Foo> $HomeFolder = Get-PSProvider -PSProvider FileSystem |
             Select-Object -ExpandProperty Home
PS C:\Foo> $HomeFolder
C:\Foo
```

*Figure 7.24: Getting the FileSystem provider home folder*

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:

```
PS C:\Foo> # 11. Checking Function drive
PS C:\Foo> Get-Module | Remove-Module -WarningAction SilentlyContinue
PS C:\Foo> $Functions = Get-ChildItem -Path Function:
PS C:\Foo> "Functions available [$($Functions.Count)]"
Functions available [36]
```

*Figure 7.25: Getting and counting functions available*

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:

```
PS C:\Foo> # 13. Checking Function drive again
PS C:\Foo> $Functions2 = Get-ChildItem -Path Function:
PS C:\Foo> "Functions now available [$($Functions2.Count)]"
Functions now available [37]
```

*Figure 7.26: Getting and counting functions available again*

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:

```
PS C:\Foo> # 14. Counting defined variables
PS C:\Foo> $Variables = Get-ChildItem -Path Variable:
PS C:\Foo> "Variables defined [$C$Variables.count)]"
Variables defined [70]
```

*Figure 7.27: Getting the function definition from the Function: drive*

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:

```
PS C:\Foo> # 15. Counting defined variables
PS C:\Foo> $Variables = Get-ChildItem -Path Variable:
PS C:\Foo> "Variables defined [$($Variables.count)]"
Variables defined [70]
```

*Figure 7.28: Counting variables on SRV1*

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:

```
PS C:\Foo> # 16. Getting trusted root certificates for the local machine
PS C:\Foo> Get-ChildItem -Path Cert:\LocalMachine\Root |
             Format-Table FriendlyName, Thumbprint

FriendlyName                                                Thumbprint
------------                                                ----------
Microsoft Root Certificate Authority                        CDD4EEAE6000AC7F40C3802C171E30148030C072
Thawte Timestamping CA                                      BE36A4562FB2EE05DBB3D32323ADF445084ED656
Microsoft Root Authority                                    A43489159A520F0D93D032CCAF37E7FE20A8B419
                                                            92B46C76E13054E104F230517E6E504D43AB10B5
Microsoft Root Certificate Authority 2011                   8F43288AD272F3103B6FB1428485EA3014C0BCFE
Microsoft Authenticode(tm) Root                             7F88CD7223F3C813818C994614A89C99FA3B5247
Microsoft Root Certificate Authority 2010                   3B1EFD3A66EA28B16697394703A72CA340A05BD5
Microsoft ECC TS Root Certificate Authority 2018            31F9FC8BA3805986B721EA7295C65B3A44534274
Microsoft Timestamp Root                                    245C97DF7514E7CF2DF8BE72AE957B9E04741E85
VeriSign Time Stamping CA                                   18F7C1FCC3090203FD5BAA2F861A754976C8DD25
Microsoft ECC Product Root Certificate Authority 2018       06F1AA330B927B753A40E68CDF22E34BCBEF3352
Microsoft Time Stamp Root Certificate Authority 2014        0119E81BE9A14CD8E22F40AC118C687ECBA3F4D8
DigiCert Global Root G2                                     DF3C24F9BFD666761B268073FE06D1CC8D4F82A4
DST Root CA X3                                              DAC9024F54D8F6DF94935FB1732638CA6AD77C13
GlobalSign Root CA - R3                                     D69B561148F01C77C54578C10926DF5B856976AD
DigiCert Baltimore Root                                     D4DE20D05E66FC53FE1A50882C78DB2852CAE474
Sectigo (AAA)                                               D1EB23A46D17D68FD92564C2F1F1601764D8E349
ISRG Root X1                                                CABD2A79A1076A31F21D253635CB039D4329A5E8
GlobalSign Root CA - R1                                     B1BC968BD4F49D622AA89A81F2150152A41D829C
Starfield Class 2 Certification Authority                   AD7E1C28B064EF8F6003402014C3D0E3370EB58A
DigiCert                                                    A8985D3A65E5E5C4B2D7D66D40C6DD2FB19C5436
Entrust.net                                                 8CF427FD790C3AD166068DE81E57EFBB932272D4
VeriSign Class 3 Public Primary CA                          742C3192E607E424EB4549542BE1BBC53E6174E2
DigiCert                                                    5FB7EE0633E259DBAD0C4C9AE6D38F1A61C7DC25
VeriSign                                                    4EB6D578499B1CCF5F581EAD56BE3D9B6744A5E5
VeriSign Universal Root Certification Authority             3679CA35668772304D30A5FB873B0FA77BB70D54
Go Daddy Class 2 Certification Authority                    2796BAE63F1801E277261BA0D77770028F20EEE4
QuoVadis Root CA 2 G3                                       093C61F38B8BDC7D55DF7538020500E125F5C836
DigiCert                                                    0563B8630D62D75ABBC8AB1E4BDFB5A899B24D43
```

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

In PowerShell 7, the WinRM service provides the underpinnings for 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:

```
PS C:\Foo> # 17. Examining ports in use by WinRM
PS C:\Foo> Get-ChildItem -Path WSMan:\localhost\Client\DefaultPorts

   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Client\DefaultPorts

Type            Name    SourceOfValue  Value
----            ----    -------------  -----
System.String   HTTP                   5985
System.String   HTTPS                  5986
```

*Figure 7.30: Getting WSMan service ports*

To show how you can configure WinRM, in *step 18*, you set WinRM to trust explicitly any remote host by setting the `TrustedHosts` 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:

```
PS C:\Foo> # 21. Importing the CimPSDrive module and creating a drive
PS C:\Foo> Import-Module -Name CimPSDrive
PS C:\Foo> New-PSDrive -Name CIM -PSProvider SHiPS -Root CIMPSDrive#CMRoot

Name Used (GB) Free (GB) Provider Root              CurrentLocation
---- --------- --------- -------- ----              ---------------
CIM                     SHiPS    CIMPSDrive#CMRoot
```

*Figure 7.31: Importing the CimPSDrive module and creating a drive*

In *step 22*, you use the newly created `CIM PSDrive` and examine the values of the `Win32_Bios` class, with output like this:

```
PS C:\Foo> # 22. Examining BIOS using the CimPSDrive module
PS C:\Foo> Get-ChildItem CIM:\Localhost\CIMV2\Win32_Bios

SMBIOSBIOSVersion : Hyper-V UEFI Release v4.1
Manufacturer      : Microsoft Corporation
Name              : Hyper-V UEFI Release v4.1
SerialNumber      : 1292-4012-4928-5640-9420-3487-94
Version           : VRTUAL - 1
PSComputerName    : localhost
```

*Figure 7.32: Viewing a WMI class*

## 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 its entry (`Remove-Item -Path Function:<functon` to `remove`).

`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 SR 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'll use both SRV1 and SRV2 in this recipe. After adding and configuring additional virtual disks to this host, you'll 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 disk number of the disk holding the S partition

```
$Part = Get-Partition -DriveLetter S
"S drive on disk [$($Part.DiskNumber)]"
```

2. 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
```

3. 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
  }
}
```

4. Counting files/folders on `S:`

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

5. Examining the `S:` drive remotely on `SRV2`

```
$ScriptBlock2 = {
  Get-ChildItem -Path S:\ -Recurse |
    Measure-Object
}
Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock2
```

6. Adding the storage replica feature to `SRV1`

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

7. Adding the storage replica feature to `SRV2`

```
$SB= {
  Add-WindowsFeature -Name Storage-Replica | Out-Null
}
Invoke-Command -ComputerName SRV2 -ScriptBlock $SB
```

8. Restarting `SRV2` and wait for the restart

```
$RSHT = @{
  ComputerName = 'SRV2'
  Force        = $true
}
Restart-Computer @RSHT -Wait -For WinRM
```

9. Restarting `SRV1` to finish the installation process

```
Restart-Computer
```

10. Creating a `G:` volume on 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
```

11. Creating a `G:` volume on `SRV2`

```
Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3
```

12. Viewing volumes on `SRV1`

```
Get-Volume | Sort-Object -Property Driveletter
```

13. Viewing volumes on `SRV2`

```
Invoke-Command -Computer SRV2 -ScriptBlock {
    Get-Volume | Sort-Object -Property Driveletter
}
```

14. 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
```

15. Examining the volumes on `SRV2`:

```
$ScriptBlock3 = {
  Get-Volume |
    Sort-Object -Property DriveLetter |
      Format-Table}
Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3
```

16. Reversing the replication

```
$ReverseHT = @{
  NewSourceComputerName   = 'SRV2'
  SourceRGName            = 'SRV2RG1'
  DestinationComputerName = 'SRV1'
  DestinationRGName       = 'SRV1RG1'
  Confirm                 = $false
}
Set-SRPartnership @ReverseHT
```

17. Viewing the SR partnership

```
Get-SRPartnership
```

18. 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:

```
PS C:\Foo> # 1. Getting disk number of the disk holding the S partition
PS C:\Foo> $Part = Get-Partition -DriveLetter S
PS C:\Foo> "S drive on disk [$C$Part.DiskNumber)]"
S drive on disk [1]
```

*Figure 7.33: Viewing a WMI class*

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

```
PS C:\Foo> # 2. Creating S: drive on SRV2
PS C:\Foo> $ScriptBlock = {
             Initialize-Disk -Number $using:Part.DiskNumber -PartitionStyle GPT
             $NewVolHT = @{
              DiskNumber   =  $using:Part.DiskNumber
               FriendlyName = 'Files'
               FileSystem   = 'NTFS'
               DriveLetter  = 'S'
             }
             New-Volume @NweVolHT
           }
PS C:\Foo> Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size PSComputerName
----------- ------------ -------------- --------- ------------ ----------------- -------------     ---- --------------
S           Files        NTFS           Fixed     Healthy      OK                      63.89 GB 63.98 GB SRV2
```

*Figure 7.34: Creating S: on SRV2*

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:

```
PS C:\Foo> # 4. Counting files/folders on S:
PS C:\Foo> Get-ChildItem -Path S:\ -Recurse | Measure-Object

Count             : 10100
Average           :
Sum               :
Maximum           :
Minimum           :
StandardDeviation :
Property          :
```

*Figure 7.35: Viewing the files on S: on SRV1*

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

```
PS C:\Foo> # 5. Examining the same drives remotely on SRV2
PS C:\Foo> $ScriptBlock2 = {
            Get-ChildItem -Path S:\ -Recurse |
              Measure-Object
           }
PS C:\Foo> Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock2

Count          : 0
Average        :
Sum            :
Maximum        :
Minimum        :
Property       :
PSComputerName : SRV2
```

*Figure 7.36: Viewing the files on SRV2*

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:

```
PS C:\Foo> # 6. Adding the storage replica feature to SRV1
PS C:\Foo> Add-WindowsFeature -Name Storage-Replica | Out-Null
WARNING: You must restart this server to finish the installation process
```

*Figure 7.37: Adding Storage Replica to SRV1*

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

```
PS C:\Foo> # 7. Adding the Storage Replica Feature to SRV2
PS C:\Foo> $SB= {
             Add-WindowsFeature -Name Storage-Replica | Out-Null
           }
PS C:\Foo> Invoke-Command -ComputerName SRV2 -ScriptBlock $SB

WARNING: You must restart this server to finish the installation process.
```

*Figure 7.38: Adding Storage Replica to SRV2*

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:

```
PS C:\Foo> # 10. Creating a G: volume in disk 3 on SRV1
PS C:\Foo> $ScriptBlock3 = {
             Initialize-Disk -Number 3 | Out-Null
             $VolumeHT = @{
              DiskNumber   =  3
              FriendlyName = 'SRLOGS'
              DriveLetter  = 'G'
             }
             New-Volume @VolumeHT
           }
PS C:\Foo> Invoke-Command -ComputerName SRV1 -ScriptBlock $ScriptBlock3

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size PSComputerName
----------- ------------ -------------- --------- ------------ ----------------- -------------     ---- --------------
G           SRLOGS       NTFS           Fixed     Healthy      OK                      63.84 GB 63.98 GB SRV1
```

*Figure 7.39: Creating a G: volume on SRV1*

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

```
PS C:\Foo> # 11. Creating G: volume on SRV2
PS C:\Foo> Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining     Size PSComputerName
----------- ------------ -------------- --------- ------------ ----------------- -------------     ---- --------------
G           SRLOGS       NTFS           Fixed     Healthy      OK                      63.89 GB 63.98 GB SRV2
```

*Figure 7.40: Creating a G: volume on SRV2*

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

```
PS C:\Foo> # 12. Viewing volumes on SRV1
PS C:\Foo> Get-Volume | Sort-Object -Property Driveletter

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining       Size
----------- ------------ -------------- --------- ------------ ----------------- -------------       ----
                         FAT32          Fixed     Healthy      OK                     67.25 MB      96 MB
C                        NTFS           Fixed     Healthy      OK                    111.61 GB   127.9 GB
G           SRLOGS       NTFS           Fixed     Healthy      OK                     63.84 GB   63.98 GB
S           Files        NTFS           Fixed     Healthy      OK                     63.83 GB   63.98 GB
W           W-FAT        FAT            Fixed     Healthy      OK                   1023.66 MB 1023.72 MB
X           x-exFAT      exFAT          Fixed     Healthy      OK                        15 GB      15 GB
Y           Y-FAT32      FAT32          Fixed     Healthy      OK                     14.98 GB   14.98 GB
Z           Z-ReFS       ReFS           Fixed     Healthy      OK                     31.78 GB   32.94 GB
```

*Figure 7.41: Viewing volumes on SRV1*

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

```
PS C:\Foo> # 13. Viewing volumes on SRV2
PS C:\Foo> Invoke-Command -Computer SRV2 -Scriptblock {
             Get-Volume | Sort-Object -Property Driveletter
           }

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining      Size PSComputerName
----------- ------------ -------------- --------- ------------ ----------------- -------------      ---- --------------
                         FAT32          Fixed     Healthy      OK                     67.26 MB    96 MB SRV2
C                        NTFS           Fixed     Healthy      OK                    115.03 GB 127.9 GB SRV2
G           SRLOGS       NTFS           Fixed     Healthy      OK                     63.89 GB 63.98 GB SRV2
S           Files        NTFS           Fixed     Healthy      OK                     63.89 GB 63.98 GB SRV2
```

*Figure 7.42: Viewing volumes on SRV2*

In *step 14*, you use the `New-SRPartnership` command to create a new Storage Replica partnership, creating the following output:

```
PS C:\Foo> # 14. Creating an SR replica group
PS C:\Foo> $NewSRHT =  @{
             SourceComputerName       = 'SRV1'
             SourceRGName             = 'SRV1RG2'
             SourceVolumeName         = 'S:'
             SourceLogVolumeName      = 'G:'
             DestinationComputerName  = 'SRV2'
             DestinationRGName        = 'SRV2RG2'
             DestinationVolumeName    = 'S:'
             DestinationLogVolumeName = 'G:'
             LogSizeInBytes           = 2gb
           }
PS C:\Foo> New-SRPartnership @SRHT

RunspaceId              : 0da18bca-0a49-4f1a-9bbf-cdd44129984c
DestinationComputerName : SRV2
DestinationRGName       : SRV2RG2
Id                      : 5c52459b-38a9-44ae-8ebd-867d6e19ece0
SourceComputerName      : SRV1
SourceRGName            : SRV1RG2
```

*Figure 7.43: Creating a Storage Replica partnership*

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

```
PS C:\Foo> # 15. Examining the volumes on SRV2
PS C:\Foo> $ScriptBlock3 = {
             Get-Volume |
               Sort-Object -Property DriveLetter |
                 Format-Table
           }
PS C:\Foo> Invoke-Command -ComputerName SRV2 -ScriptBlock $ScriptBlock3

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining      Size
----------- ------------ -------------- --------- ------------ ----------------- -------------      ----
                         FAT32          Fixed     Healthy      OK                     67.26 MB     96 MB
C                        NTFS           Fixed     Healthy      OK                    115.03 GB 127.9 GB
G           SRLOGS       NTFS           Fixed     Healthy      OK                     61.89 GB 63.98 GB
S                        Unknown        Fixed     Healthy      Unknown                    0 B      0 B
```

*Figure 7.44: Examining the volumes on SRV2*

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:

```
PS C:\Foo> # 17. Viewing the SR Partnership
PS C:\Foo> Get-SRPartnership

RunspaceId              : 0da18bca-0a49-4f1a-9bbf-cdd44129984c
DestinationComputerName : SRV1
DestinationRGName       : SRV1RG2
Id                      : 5c52459b-38a9-44ae-8ebd-867d6e19ece0
SourceComputerName      : SRV2
SourceRGName            : SRV2RG2
```

*Figure 7.45: Examining the reversed SR partnership*

## 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, you can reverse the replication. 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 the disks available for pooling

```
$Disks = Get-PhysicalDisk -CanPool $true
$Disks | Sort-Object -Property Deviceid
```

2. Creating a storage pool

```
$NewPoolHT = @{
  FriendlyName                = 'RKSP'
  StorageSubsystemFriendlyName = "Windows Storage*"
  PhysicalDisks               = $Disks
}
New-StoragePool @NewPoolHT
```

3. Creating a mirrored hard disk named `Mirror1`

```
$VDisk1HT = @{
  StoragePoolFriendlyName   = 'RKSP'
  FriendlyName              = 'Mirror1'
  ResiliencySettingName     = 'Mirror'
  Size                      = 8GB
  ProvisioningType          = 'Thin'
```

```
}
New-VirtualDisk @VDisk1HT
```

4. Creating a three-way mirrored disk named `Mirror2`

```
$VDisk2HT = @{
  StoragePoolFriendlyName    = 'RKSP'
  FriendlyName               = 'Mirror2'
  ResiliencySettingName      = 'Mirror'
  NumberOfDataCopies         = 3
  Size                       = 8GB
  ProvisioningType           = 'Thin'
}
New-VirtualDisk @VDisk2HT
```

5. Creating a volume in `Mirror1`

```
Get-VirtualDisk  -FriendlyName 'Mirror1' |
  Get-Disk |
    Initialize-Disk -PassThru |
      New-Partition -AssignDriveLetter -UseMaximumSize |
        Format-Volume
```

6. Creating a volume in `Mirror2`

```
Get-VirtualDisk  -FriendlyName 'Mirror2' |
  Get-Disk |
    Initialize-Disk -PassThru |
      New-Partition -AssignDriveLetter -UseMaximumSize |
        Format-Volume
```

7. 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:

```
PS C:\Foo> # 1. Viewing disks available for pooling
PS C:\Foo> $Disks = Get-PhysicalDisk -CanPool $true
PS C:\Foo> $Disks | Sort-Object -Property Deviceid

Number FriendlyName      SerialNumber MediaType    CanPool OperationalStatus HealthStatus Usage         Size
------ ------------      ------------ ---------    ------- ----------------- ------------ -----         ----
4      Msft Virtual Disk              Unspecified  True    OK                Healthy      Auto-Select 64 GB
5      Msft Virtual Disk              Unspecified  True    OK                Healthy      Auto-Select 64 GB
6      Msft Virtual Disk              Unspecified  True    OK                Healthy      Auto-Select 64 GB
7      Msft Virtual Disk              Unspecified  True    OK                Healthy      Auto-Select 64 GB
8      Msft Virtual Disk              Unspecified  True    OK                Healthy      Auto-Select 64 GB
```

*Figure 7.46: Viewing disks available for pooling on SRV1*

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:

```
PS C:\Foo> # 2. Creating a storage pool
PS C:\Foo> $NewPoolHT = @{
             FriendlyName                = 'RKSP'
             StorageSubsystemFriendlyName = "Windows Storage*"
             PhysicalDisks               = $Disks
           }
PS C:\Foo> New-StoragePool @NewPoolHT

FriendlyName OperationalStatus HealthStatus IsPrimordial IsReadOnly      Size AllocatedSize
------------ ----------------- ------------ ------------ ----------      ---- -------------
RKSP         OK                Healthy      False        False      317.42 GB       1.25 GB
```

*Figure 7.47: Creating a new storage pool*

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:

```
PS C:\Foo> # 3. Creating a mirrored hard disk named Mirror1
PS C:\Foo> $VDisk1HT = @{
             StoragePoolFriendlyName   = 'RKSP'
             FriendlyName              = 'Mirror1'
             ResiliencySettingName     = 'Mirror'
             Size                      = 8GB
             ProvisioningType          = 'Thin'
           }
PS C:\Foo> New-VirtualDisk @VDdisk1HT

FriendlyName ResiliencySettingName FaultDomainRedundancy OperationalStatus HealthStatus Size FootprintOnPool StorageEfficiency
------------ --------------------- --------------------- ----------------- ------------ ---- --------------- -----------------
Mirror1      Mirror                1                     OK                Healthy      8 GB          1.5 GB            33.33%
```

*Figure 7.48: Creating a new mirrored disk inside Storage Spaces*

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

```
PS C:\Foo> # 4. Creating a three way mirrored disk named Mirror2
PS C:\Foo> $VDisk2HT = @{
             StoragePoolFriendlyName    = 'RKSP'
             FriendlyName               = 'Mirror2'
             ResiliencySettingName      = 'Mirror'
             NumberOfDataCopies         = 3
             Size                       = 8GB
             ProvisioningType           = 'Thin'
           }
PS C:\Foo> New-VirtualDisk @VDisk2HT

FriendlyName ResiliencySettingName FaultDomainRedundancy OperationalStatus HealthStatus Size FootprintOnPool StorageEfficiency
------------ --------------------- --------------------- ----------------- ------------ ---- -------------- -----------------
Mirror2      Mirror                2                     OK                Healthy      8 GB          1.5 GB            16.67%
```

*Figure 7.49: Creating a three-way mirrored storage space*

In *step 5*, you create a new volume in the `Mirror1` storage space, which looks like this:

```
PS C:\Foo> # 5. Creating a volume in Mirror1
PS C:\Foo> Get-VirtualDisk  -FriendlyName 'Mirror1' |
             Get-Disk |
               Initialize-Disk -PassThru |
                 New-Partition -AssignDriveLetter -UseMaximumSize |
                   Format-Volume

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining    Size
----------- ------------ -------------- --------- ------------ ----------------- -------------    ----
D                        NTFS           Fixed     Healthy      OK                     7.95 GB 7.98 GB
```

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

In *step 6*, you create another new volume. You create the volume in the three-way mirror storage space, `Mirror2`, with output like this:

```
PS C:\Foo> # 6. Creating a volume in Mirror2
PS C:\Foo> Get-VirtualDisk  -FriendlyName 'Mirror2' |
             Get-Disk |
              Initialize-Disk -PassThru |
                New-Partition -AssignDriveLetter -UseMaximumSize |
                  Format-Volume

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining    Size
----------- ------------ -------------- --------- ------------ ----------------- -------------    ----
E                        NTFS           Fixed     Healthy      OK                     7.95 GB 7.98 GB
```

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

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:

```
PS C:\Foo> # 7.  Viewing volumes on SRV1
PS C:\Foo> Get-Volume | Sort-Object -Property DriveLetter

DriveLetter FriendlyName FileSystemType DriveType HealthStatus OperationalStatus SizeRemaining       Size
----------- ------------ -------------- --------- ------------ ----------------- -------------       ----
                         FAT32          Fixed     Healthy      OK                     67.24 MB      96 MB
C                        NTFS           Fixed     Healthy      OK                    108.07 GB   127.9 GB
D                        NTFS           Fixed     Healthy      OK                      7.95 GB    7.98 GB
E                        NTFS           Fixed     Healthy      OK                      7.95 GB    7.98 GB
G           SRLOGS       NTFS           Fixed     Healthy      OK                     61.84 GB   63.98 GB
S                        Unknown        Fixed     Healthy      Unknown                    0 B        0 B
W           W-FAT        FAT            Fixed     Healthy      OK                  1023.66 MB 1023.72 MB
X           x-exFAT      exFAT          Fixed     Healthy      OK                        15 GB      15 GB
Y           Y-FAT32      FAT32          Fixed     Healthy      OK                     14.98 GB   14.98 GB
Z           Z-ReFS       ReFS           Fixed     Healthy      OK                     31.78 GB   32.94 GB
```

*Figure 7.52: Viewing disk volumes available on SRV1*

## 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., creating 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.

# Join our community on Discord

Join our community's Discord space for discussions with the author and other readers:

`https://packt.link/SecNet`