4

Managing PowerShell 7 In The Enterprise

In this chapter, we cover the following recipes:

* Installing RSAT Tools on Windows Server
* Exploring package management
* Exploring PowerShellGet and PS Gallery
* Adding external modules
* Creating an internal PowerShell repository
* Establishing a code signing environment
* Implementing Just Enough Administration

# Introduction

Before you can begin to administer your Windows Server 2019 infrastructure, you need to create an environment in which you can use PowerShell to carry out the administration. That environment includes ensuring you have the tools you need close to hand, and ensuring the environment is as secure as possible.

To manage Windows roles and features as well as manage Windows itself with PowerShell, you need modules of PowerShell commands. You can manage most Windows features using PowerShell using the tools which come with the feature in question. You can install the tools with a feature - installing the ActiveDirectory module when you install Active Directory on a system. You can also load the tools separately and manage features remotely. The Remote Server Administration Tools (RSAT) are modules that allow you to manage Windows roles and features. In “Installing RSAT Tools on Windows Server”, you investigate the RSAT tools and how you can install them in Windows Server.

Although the RSAT tools provide much excellent functionality, they do not allow you to do everything you might wish. To fill the gaps, the PowerShell community has created many additional modules/commands which you can use as an alternative to PowerShell commands. To manage those you need package management which you examine in “Exploring Package Management”. In “Exploring PowerShellGet and PS Gallery” you look at one source of modules and examine how to find and utilize modules contained in the PS Gallery.

# Installing RSAT Tools

The RSAT tools are fundamental to administering the roles and features you can install on Windows Server. Each feature in Windows Server can, optionally have management tools and most do so. These tools can include PowerShell cmdlets, functions, and aliases. Some features also have older Win32 console applications. For the most part, you do not need the console applications since you can use the cmdlets, but that is not always the case. You may have older scripts that use those console applications.

You can also install the RSAT tools independently of a Windows Server feature on both Windows Server and the Windows Client (that is Windows 10). This recipe covers RSAT tool installation on Windows Server 2019.

You can also install the RSAT tools in Windows 10 and administer your servers remotely. The specific method of installing the RSAT tools varies with the specific version of Windows 10 you are using. For earlier Windows 10 editions, you can download the tools here https://www.microsoft.com/en-gb/download/details.aspx?id=45520.

In later editions of Window 10, beginning with the Windows 10 October Update, you install the RSAT tools using the “Features on Demand mechanism inside Windows 10. The URL in the previous paragraph has fuller details of how to install the RSAT tools on Windows 10.

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server 2019 Data Centre Edition.

## How to do it...

1. Displaying counts of available PowerShell commands

$CommandsBeforeRSAT = Get-Command -Module \*

$CmdletsBeforeRSAT = $CommandsBeforeRSAT  |

    Where-Object commandtype -eq 'Cmdlet'

$CommandCountBeforeRSAT = $CommandsBeforeRSAT.Count

$CmdletCountBeforeRSAT  = $CmdletsBeforeRSAT.Count

"On Host: [$(hostname)]"

"Total Commands available before RSAT installed [$CommandCountBeforeRSAT]"

"Cmdlets available before RSAT installed        [$CmdletCountBeforeRSAT]"

1. Getting command types returned by Get-Command

$CommandsBeforeRSAT |

  Group-Object -Property CommandType

1. Checking the object type details

$CommandsBeforeRSAT |

  Get-Member |

    Select-Object -ExpandProperty TypeName -Unique

1. Getting the collection of PowerShell modules and a count of modules before adding  
   the RSAT tools

$ModulesBefore = Get-Module -ListAvailable

1. Displaying a count of modules available before adding the RSAT tools

$CountOfModulesBeforeRSAT = $ModulesBefore.Count

"$CountOfModulesBeforeRSAT modules available"

1. Getting a count of features available on SRV1

Import-Module -Name ServerManager -WarningAction SilentlyContinue

$Features  = Get-WindowsFeature

$FeaturesI = $Features | Where-Object Installed

| Where-object installed

$RsatF     = $Features |

               Where-Object Name -Match 'RSAT'

$RSATFI    = $RSATF |

              Where-Object Installed

1. Displaying counts of features installed

"On Host [$(hostname)]"

"Total features available      [{0}]"  -f $Features.count

"Total features installed      [{0}]"  -f $FeaturesI.count

"Total RSAT features available [{0}]"  -f $RSATF.count

"Total RSAT features installed [{0}]"  -f $RSATFI.count

1. Adding ALL RSAT tools to SRV1

Get-WindowsFeature -Name \*RSAT\* |

  Install-WindowsFeature

1. Rebooting SRV1 then logging on as the local administrator

Restart-Computer -Force

1. Getting Details of RSAT tools now installed on SRV1

$FSRV1A   = Get-WindowsFeature

$IFSRV1A  = $FSRV1A | Where-Object Installed

$RSFSRV1A = $FSRV1A | Where-Object Installed |

              Where-Object Name -Match 'RSAT'

1. Displaying after effects

"After Installation of RSAT tools on SRV1"

"$($IFSRV1A.count) features installed on SRV1"

"$($RSFSRV1A.count) RSAT features installed on SRV1"

1. Displaying RSAT tools on SRV1

$MODS = "$env:windir\system32\windowspowerShell\v1.0\modules"

$SMMOD = "$MODS\ServerManager"

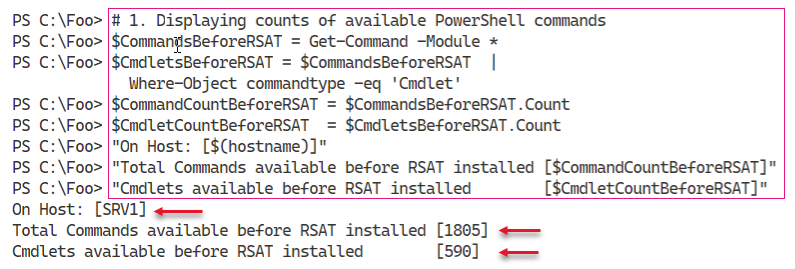
Update-FormatData -PrependPath "$SMMOD\\*.format.ps1xml"

Get-WindowsFeature |

  Where-Object Name -Match 'RSAT'

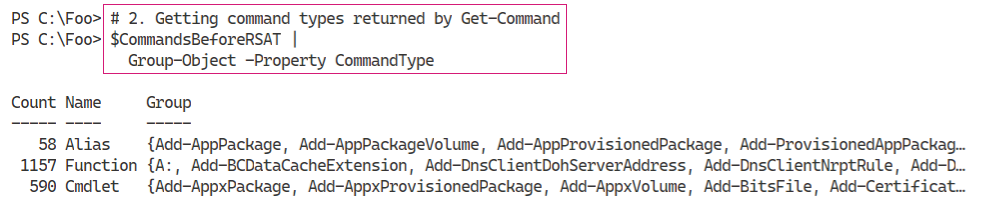
## How it works...

In step 1, you use the Get-Command command to obtain all the commands inside all modules on SRV1. The step then displays a count of the total number of commands available on SRV1 and how many actual cmdlets exist on SRV1 prior to installing the RSAT tools. The output of this step looks like this:



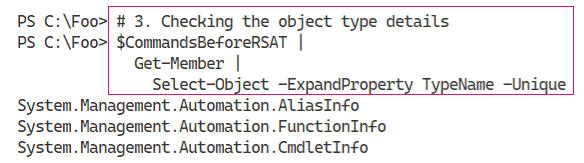
Insert image B42024\_04\_01.png

In step 2, you display a count of the types of commands available thus far on SRV1, which looks like this:



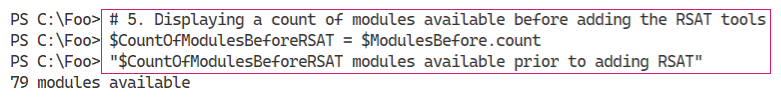
Insert image B42024\_04\_02.png

In PowerShell, when you use Get-Command, the cmdlet returns different objects to describe the different types of commands. As you saw in the previous step, there are three command types which PowerShell returns in different object classes. You can see the class names for those three command types in the output from step 3, which looks like this:



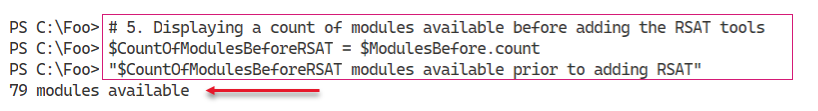
Insert image B42024\_04\_03.png

In step 4, which produces no output, you get all the modules available on SRV1. In step 5, you display a count of the number of modules available, which looks like this:



Insert image B42024\_04\_04.png

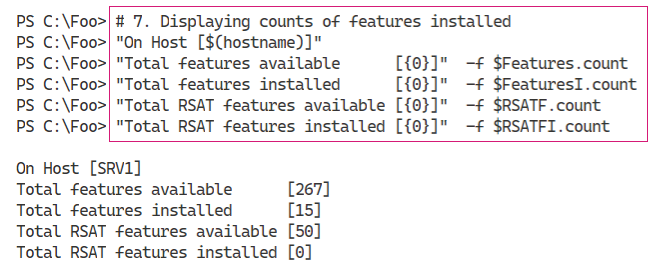
In step 5, you discover there are a total of 79 modules on SRV1 before you add the RSAT tools. The output of this step looks like this:



Insert image B42024\_04\_05.png

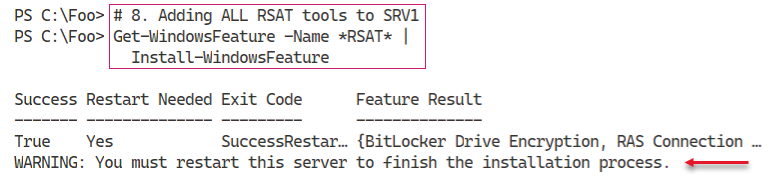
In step 6, you obtain counts of the features and features installed as well as the numbe of RSAT features available and installed. This step generates no output

In step 7, you display counts of the features available and installed, and the number of RSAT specific features available and installed, which looks like this:



Insert image B42024\_04\_06.png

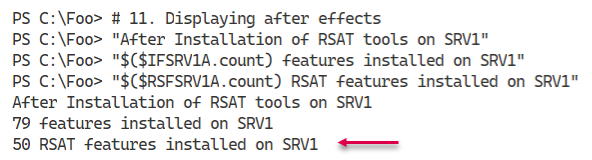
In step 8, you get and install all the RSAT features in Windows Server. This process does take a bit of time, and generates output like this:



Insert image B42024\_04\_07.png

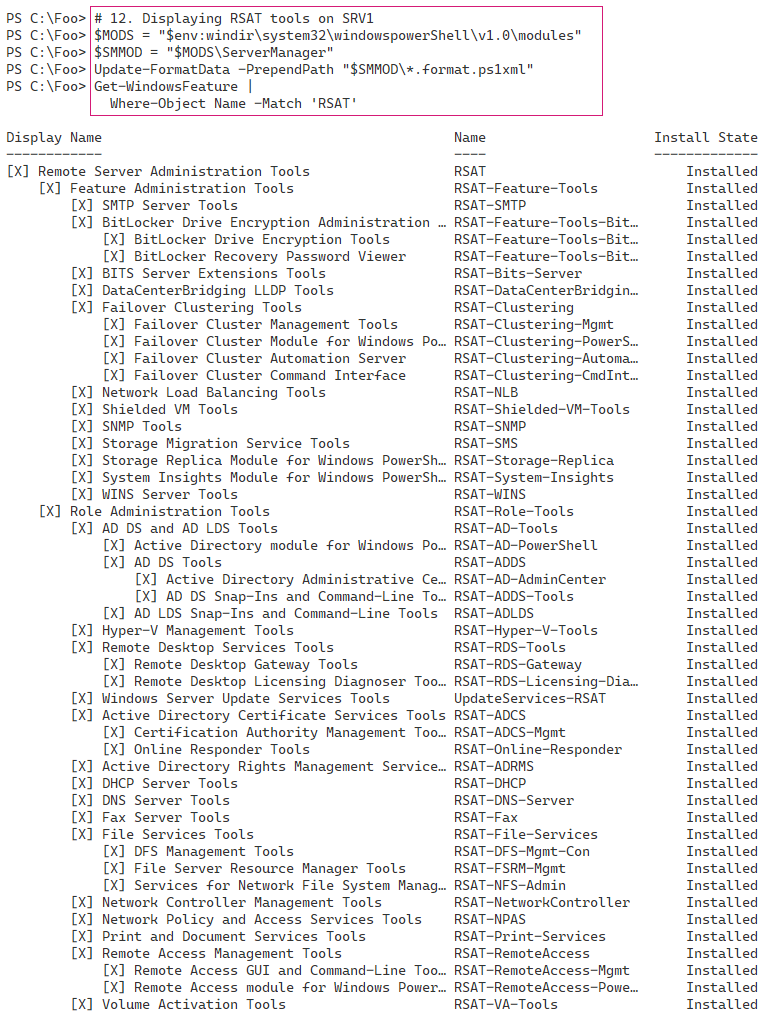
The installation of these tools requies a restart, as you can see in the figure above. Thus, in step 9 you restart the system. After the restart you login to SRV1 as an administrator to continue.

Now that you have added all the RSAT-Related Windows Features, you can begin to ge details of what you installed. In step 10, which creates no output, you get details of the features you just installed. and the commands they conain. In step 11, you display the count of RSAT features not available on SRV1, which looks like:



Insert image B42024\_04\_08.png

In step 12, you display the RSAT features you installed in an earlier step. The output of this step looks like this



Insert image B42024\_04\_09.png

## There's more...

The output from step 1 shows there are 1805 total commands and 590 Module based cmdlets available on SRV1, before adding the RSAT tools. The actual number may vary, depending on what additional tools, feature, or applications you might have added to SRV1 or the Windows Server version itself.

In step 2 and step 3, you find the kinds of commands available and the object type name PowerShell uses to describe these different command types. When you have the class names, you can use your favourite search engine to discover more details about each of these command types.

# Exploring Package Management

The PackageMangement PowerShell module provides tools that enable to download and install software packages from a variety of sources. The module, in effect, implements a provider interface that software package management systems use to manage software packages. You can use the cmdlets  
in the PackageMangement module to work with a variety of package management systems.

This module in effect, an API to package management providers such as PowerShellGet, discussed in the “Exploring PowerShellGet and PowerShell Gallery” recipe. The key function of the PackageMangement module is to manage the set of software repositories in which package management tools can search, obtain, install, and remove packages. The module enables you to discover and utilize software packages from a variety of sources (and potentially varying in quality).

This recipe explores the PackageManagement module from SRV1.

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

1. Reviewing the cmdlets in the PackageManagement module

Get-Command -Module PackageManagement

1. Reviewing installed providers with Get-PackageProvider

Get-PackageProvider |

  Format-Table -Property Name,

                         Version,

                         SupportedFileExtensions,

                         FromTrustedSource

1. Examining available Package Providers

$PROVIDERS = Find-PackageProvider

$PROVIDERS |

    Select-Object -Property Name,Summary |

      Format-Table -AutoSize -Wrap

1. Discovering and counting available packages

$PAGKAGES = Find-Package

"Discovered {0:N0} packages" -f $PAGKAGES.count

1. Showing first 5 packages discovered

$PAGKAGES  |

    Select-Object -First 5 |

      Format-Table -AutoSize -Wrap

1. Installing Chocolatier provider

Install-PackageProvider -Name Chocolatier -Force

1. Verifying Chocolatier is in the list of installed providers

Get-PackageProvider |

  Select-Object -Property Name,Version

1. Discovering Packages from Chocolatier

$Start = Get-Date

$CPackages = Find-Package -ProviderName Chocolatier -Name \*

"$($CPackages.Count) packages available from Chocolatey"

$End = Get-Date

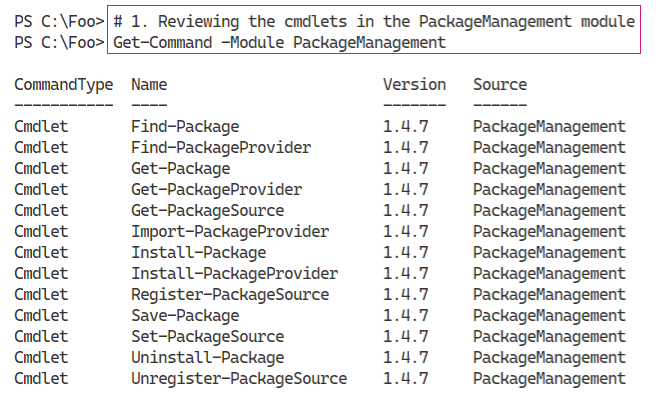
1. Displaying how long it took for Chocolatier

$Elapsed = $End - $Start

"Took {0:n3} seconds" -f $Elapsed.TotalSeconds

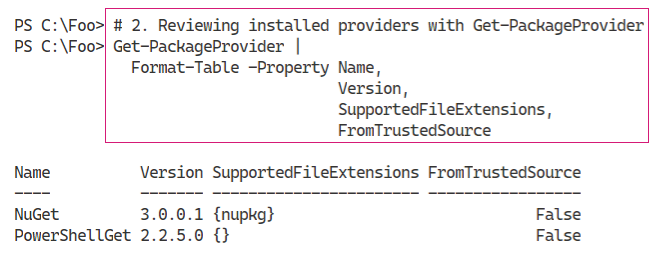
## How it works...

In step 1, you use Get-Command to view the commands provided by the PackageManagement module, which looks like this:



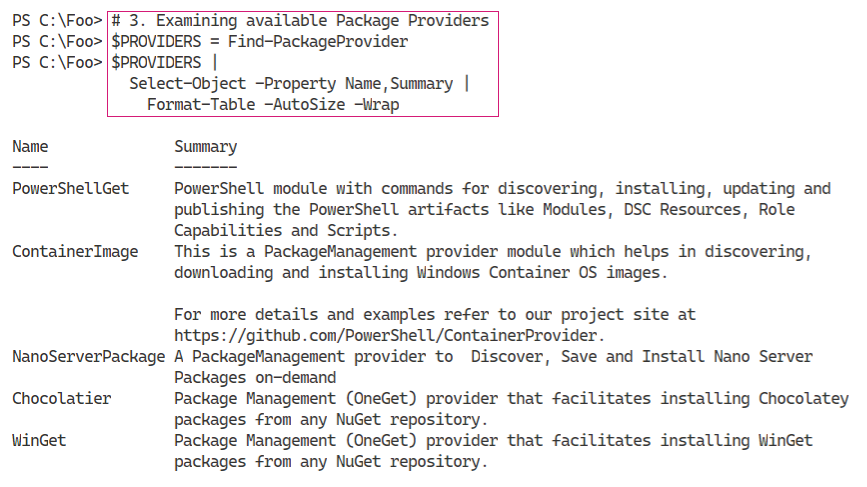
Insert image B42024\_04\_10.png

In step 2, you use the Get-PackageProvider cmdlet to discover the installed package providers,which looks like this



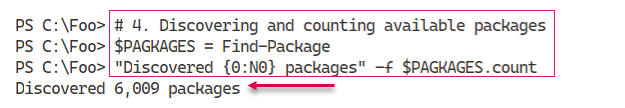
Insert image B42024\_04\_11.png

In step 3, you use the Find-PackageProvider to discover any other providers you can use. The output looks like this:



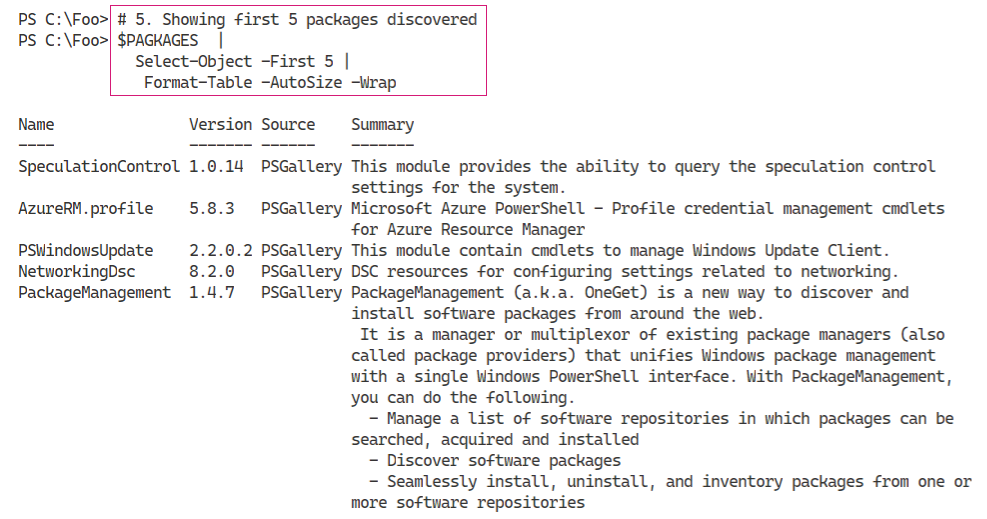
Insert image B42024\_04\_12.png

In step 4, you discover and count the packages you can find using Find-Packages, with output like this:



Insert image B42024\_04\_13.png

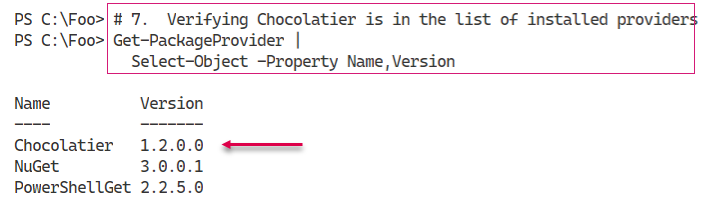
To illustrate some of the packages you just discovered, in step 5, you view the first 5 packages, which looks like this:



Insert image B42024\_04\_14.png

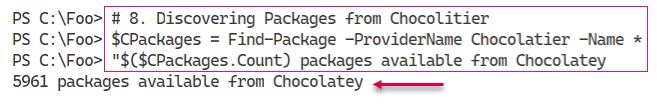
In step 6, you install the Chocolatier package provider, which gives you access via the package management to the Chocolately repository. Chocolately is a third-party application repository, although not directly supported by Microsoft. For more information about the Chocolately repository, see https://chocolatey.org/.

With step 7, you review the packaged providers now available on SRV1 to ensure Chocolatier is included, which looks like this:



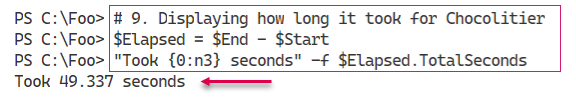
Insert image B42024\_04\_15.png

In step 8, you discover the packages available via the Chocolatier provider and display a count of packages. In this step, you also capture the date and time between the start and finish of finding packages. The output from this step looks likes this:



Insert image B42024\_04\_16.png

In step 9, you display the time taken to find the packages on Chocolately, which looks like this:



Insert image B42024\_04\_17.png

## There's more...

In step 3, you obtain a list of packages and then, in step 4, display them. The packages you see when you run this step are most likely to change - the package repositories are in a state of near-constant change. If you are looking for packages, the approach in these two steps is helpful. You download the list of packages and store it locally. Then you discover more about the existing packages without incurring the long time it takes to retriever the list of packages from any given repository. Later, in step 9, you display how long it took to obtain the list of packages from Chocolately. In your environment, this time may vary from that shown here, but it illustrates the usefulness of getting a list of all packages first before diving into discovery.

In step 6, you install another package provider, Chocolatier. This provider gives you access, via the package management commands, to the Chocolately repository. Chocolately provides you access to common application platforms and is much like apt-get in Linux (but for Windows). As always, be careful when obtaining applications or application components from any third-party repository since your vendors and partners may not provide full support in the case of an incident.

# Exploring PowerShellGet and PS Gallery

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

## How it works...

In step 1, you

Insert image B42024\_04\_19.png

## There's more...

# Adding External Modules

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

## How it works...

In step 1, you

Insert image B42024\_03\_35.png

## There's more...

# Creating an internal PowerShell repository

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

## How it works...

In step 1, you

Insert image B42024\_03\_35.png

## There's more...

# Establishing a code signing environment

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

## How it works...

In step 1, you

Insert image B42024\_03\_35.png

## There's more...

# Implementing Just Enough Administration

## Getting Ready

You run this recipe on SRV1, on which you have installed PowerShell 7 and VS Code. SRV1 is a workgroup server running Windows Server Data Centre Edition.

## How to do it...

## How it works...

In step 1, you

Insert image B42024\_03\_35.png

## There's more...