12

Using Containers

In this chapter, we cover the following recipes:

* Configuring a container host
* Deploying sample containers
* Deploying IIS in a container
* Using a Dockerfile to create a custom container

# Introduction

Containers, as a method of Unix virtualization, containers have been around for quite a while. To a large degree, containers serve as an approach for deploying applications popularized by the open-source Docker initiative. Windows Server 2022, Windows Server supports both Docker and Docker containerization integrated with Hyper-V.

With containers in Windows Server 2022, you perform most administration tasks not by using PowerShell cmdlets, but by using a command-line tool called docker.exe. For those used to PowerShell's object-oriented and task-focused approach, you may find this application hard to use. I daresay you are not alone. The docker.exe application works in PowerShell and you can, of course, use PowerShell to wrap the command.

With containers in Windows Server 2019, you need to download and install a number of components including container base images. These require an internet connection.

Containers provide scalability by enabling you to run multiple containers directly on top of Windows Server 2019. This takes up considerably fewer resources than if each container was contained in its own separate virtual machine (VM). In theory, running multiple containers on a single host could be a security risk in that malware could enable bad actors to access the contents of one container from another unfriendly container. To reduce those risks, you can run containers inside Hyper-V. With Hyper-V containers, the container is run inside a completely virtualized environment that provides additional hardware-level security, albeit at the price of performance. Hyper-V containers are also useful in a shared tenant environment, where one container host can run containers belonging to different organizations.

To deploy containers in Windows Server 2019, you need to provide both a container host (to run the container) and one or more container images, which Docker can run as a container. In the Configuring a container host recipe, you configure a host, CH1 to run containers.

Once you have configured a container host, it's a great idea to test that you can run containers successfully. There are a number of sample containers you can download to test out the basic container functionality (and the use of docker.exe). In the Deploying sample containers, you explore and download key base container images and sample containers.

With containers, you package applications inside a container, which then makes use of a shared kernel. The (single) shared kernel provides kernel-level features for all the containers deployed on a container host. The container then runs an application making use of the shared kernel.

A common application many customers containerize is as Internet Information Server (IIS). This is straightforward, as you can see in the Deploying IIS in a container recipe.

You can also build your own customized containers. To build a container you create a Dockerfile, a simple text file with build instructions. Docker.exe uses this file to build your customized container image for you to deploy. You look at creating and using an image using a Dockerfile in the snappily-named Using a Dockerfile to create and use a container recipe.

This chapter provides only an introduction to containers, images, docker.exe, and Dockerfile files. There is much more to explore with containers. Topics including Docker networking, Docker Swarm, and more are outside the scope of this book. To discover more about containers than we can fit here, look at Packt's book: Learning Windows Server Containers by Srikanth Machiraju. And, for more on the endearingly awful docker.exe application, take a look at Docker on Windows by Elton Stoneman.  
For more information on Windows containers, see this link: https://docs.microsoft.  
com/en-us/virtualization/windowscontainers/about/

The recipes in this chapter use the command-line tool docker.exe. For those familiar and comfortable with all of PowerShell's awesomeness, this is going to come as a bit of a shock. As you can observes, docker.exe has no tab completion, all output is minimal text blobs (no objects), parameter names seem random and curious, the online help is not very helpful, and the error reporting is downright atrocious. All in all, docker.exe takes time to get to grips with, is less easy to automate than other Windows features, and feels very, very slow even on a well-equipped workstation. But if you plan to user the awesome containers features in Windows Server, consider spending some time building a good framework and framework tools for your environment. It’s also worth noting that if you uyse most major search engines to discover aspects of containers, searches tend to yield a lot of useful pages, but focused on Linux as a container host and using tools and features no available under Windows..

# Configuring a container host

The first step in containerization is to configure a container host. The container host is a Windows host running Windows Server 2022 with the necessary container-related services and prerequisites installed and running. You can also run containers on Windows 10, but this is outside the scope of this chapter.

In this recipe, you install the components necessary for containers, including loading the Docker components

## Getting Ready

This recipe uses CH1, a domain joined server running Windows Server 2022. You have installed PowerShell 7 and VS Code on this host.

## How to do it...

1. Discovering the network adapter, adapter interface and adapter interface index

$IPType    = 'IPv4'

$Adapter   = Get-NetAdapter |  Where-Object Status -eq 'Up'

$Interface = $Adapter | Get-NetIPInterface -AddressFamily $IPType

$Index     = $Interface.IfIndex

Get-NetIPAddress -InterfaceIndex $Index -AddressFamily $IPType |

  Format-Table -Property Interface\*, IPAddress, PrefixLength

1. Setting a new IP address for the NIC

$IPHT = @{

  InterfaceIndex = $Index

  PrefixLength   = 24

  IPAddress      = '10.10.10.51'

  DefaultGateway = '10.10.10.254'

  AddressFamily  = $IPType

}

New-NetIPAddress @IPHT

1. Verifying the new IP address

Get-NetIPAddress -InterfaceIndex $Index -AddressFamily $IPType |

  Format-Table IPAddress, InterfaceIndex, PrefixLength

1. Setting DNS Server IP address

$CAHT = @{

  InterfaceIndex  = $Index

  ServerAddresses = '10.10.10.10'

}

Set-DnsClientServerAddress @CAHT

1. Verifying the new IP configuration

Get-NetIPAddress -InterfaceIndex $Index -AddressFamily $IPType |

  Format-Table

1. Testing that SRV2 can see the domain controller

Test-NetConnection -ComputerName DC1.Reskit.Org |

  Format-Table

1. Creating a credential for DC1

$U    = 'Reskit\Administrator'

$PPT  = 'Pa$$w0rd'

$PSC  = ConvertTo-SecureString -String $ppt -AsPlainText -Force

$Cred = [pscredential]::new($U,$PSC)

1. Setting WinRM on SRV2 to trust s

$TPPATH = 'WSMan:\localhost\Client\TrustedHosts'

Set-Item -Path $TPPATH -Value 'DC1' -Force

Restart-Service -Name WinRM -Force

1. Enabling non-secure updates to Reskit.Org DNS domain

$DNSSSB = {

  $SBHT = @{

    Name          = 'Reskit.Org'

    DynamicUpdate = 'NonsecureAndSecure'

}

  Set-DnsServerPrimaryZone @SBHT

}

Invoke-Command -ComputerName DC1 -ScriptBlock $DNSSSB -Credential $Cred

1. Ensuring SRV2 registers within the Reskit.Org DNS zone

$DNSCHT = @{

  InterfaceIndex                 = $Index

  ConnectionSpecificSuffix       = 'Reskit.Org'

  RegisterThisConnectionsAddress = $true

  UseSuffixWhenRegistering       = $true

}

Set-DnsClient  @DNSCHT

1. Registering host IP address at DC1

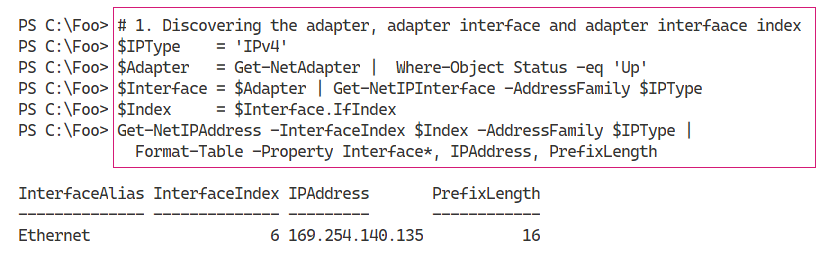
Register-DnsClient

1. Testing the DNS server on DC1.Reskit.Org resolves SRV2

Resolve-DnsName -Name SRV2.Reskit.Org -Type 'A' -Server DC1.Reskit.Org

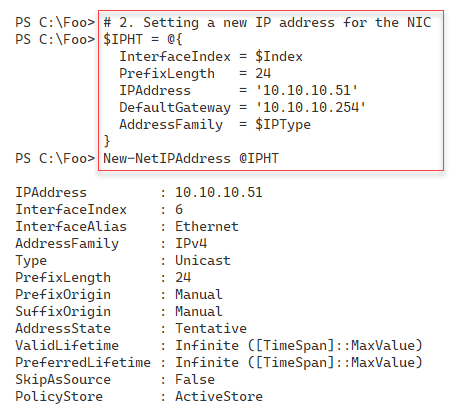
## How it works...

In step 1, you use Get-NetAdapter and Get-NetIPAddress to determine the IP address of this server. Then you display the resultant address, which looks like this:



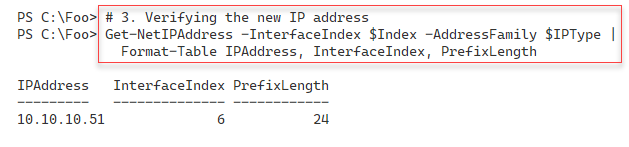
1. Insert image B1672\_01\_01.png

In step 2, you use the New-NetIPAddress cmdlet to set a static IP address on SRV2. The output looks like this:



1. Insert image B1672\_01\_02.png

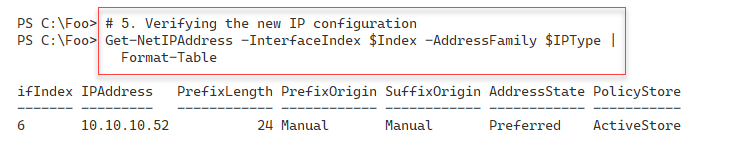
To double-check that you have configured SRV2 with the correct IP address configuration, you can use the Get-NetIPaddress cmdlet. The output looks like this:



1. Insert image B1672\_01\_03.png

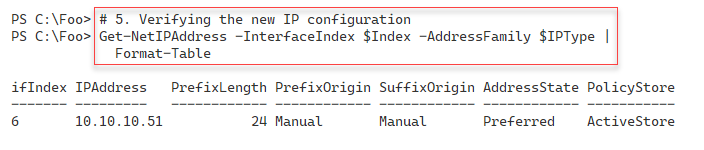
Besides setting an IP address, subnet mask and default gateway, you also need to configure SRV2 with a DNS server address. In step 4, you use the Set‑DnsClientServerAddress cmdlet, which creates no output.

To check the updated IP configuration on SRV2, in step 5, you verify the configuration by (re)-using the Get-Get-NetIPAddress cmdlet, with output like this:



1. Insert image B1672\_01\_04.png

In step 6, you use Test-interconnection to ensure SRV2 can connect to DC1, the domain controller in the Reskit.Org domain with this as the output:



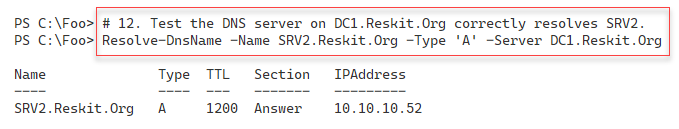
1. Insert image B1672\_01\_05.png

To enable SRV2, a workgroup computer to run commands on DC1, you need correct Windows credentials. In step 7, which creates no output, you create credentials for the Administrator user in Reskit.Org.

With step 8, you configure the WinRM service to trust DC1 explicitly. This step creates no output.

In step 9, you reconfigure the DNS Service on DC1 to enable non-secure updates to the Reskit.Org domain. In step 10, you configure SRV2 to register itself within the Reskit.Org zone on DC1. And then, in step 11, you register SRV2’s IP address within the DNS service on DC1. These three steps also produce no output.

In the last step, step 12, you query the DNS service to resolve the domain name SRV2.Reskit.Org. This step produces the following output:



1. Insert image B1672\_01\_06.png

In this recipe, you configured a workgroup server to have a static IP address. You also configure the DNS service to enable SRV2 to register a DNS record within the Reskit.Org domain. In most production scenarios, you would join SRV2 to the domain, in which case DNS registration just works without needing step 7 through step 11.

## There's more...

In step 5, you verify SRV2’s IP address. This test does not check SRV2’s DNS configuration. To check the DNS address as well, you could use Get-NetIPConfiguration.

In step 7, you create a credential to enable you to run commands on DC1. In this step, you use the enterprise/domain administrator account. In production, a better approach would be to create another user with a subset of the Enterprise Admin’s group's permissions then use that user to perform step 9.

In step 8, you configure WinRM to trust the DNS Server, DC1. This configuration is necessary for a workgroup environment because, by default, workgroup computers do not trust other servers when using PowerShell remoting (as you do in a later step). PowerShell remoting, by default, performs mutual authentication. Kerberos provides the mutual authentication in a domain environment, while in a workgroup environment, you could use SSL to connect to DC1. By configuring SRV2 to trust DC1, you are disabling authentication of DC1 by SRV2. In a protected environment, like you have your set of Reskit.Org servers, this is acceptable. In production and especially in larger environments, a better approach is to enable SSL for remoting to hosts in separate security realms.

# Deploying sample containers

In today’s connected world, network connectivity is vital. When you add a new server to your infrastructure, it is useful to ensure that the server can connect to and use the network.

In this recipe, you perform necessary network connectivity tests on the newly installed SRV2 host. You should ensure that full connectivity exists before adding a server to the domain.

## Getting Ready

This recipe uses SRV2, a workgroup host. You gave this host a static IP address in “Configuring IP Addressing”.

## How to do it...

1. Verifying SRV2 itself is up, and that Loopback is working

Test-Connection -ComputerName SRV2 -Count 1 -IPv4

1. Testing connection to local host's WinRM port

Test-NetConnection -ComputerName SRV2 -CommonTCPPort WinRM

1. Testing basic connectivity to DC1

Test-Connection -ComputerName DC1.Reskit.Org -Count 1

1. Checking  connectivity to SMB port on DC1

Test-NetConnection -ComputerName DC1.Reskit.Org -CommonTCPPort SMB

1. Check connectivity to the LDAP port on DC1

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 389

1. Examining the path to a remote server on the Internet

$NCHT = @{

  ComputerName     = 'WWW.Packt.Com'

  TraceRoute       = $true

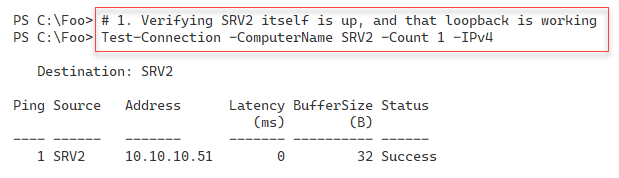
  InformationLevel = 'Detailed'

}

Test-NetConnection @NCHT    # Check our wonderful publisher

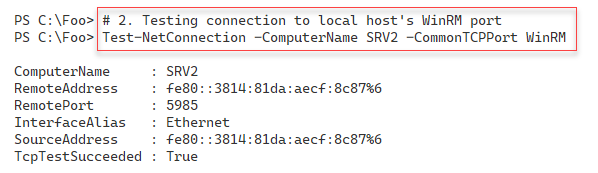
## How it works...

In step 1, you verify that SRV2’s Loopback adapter works and that the basic TCP/IP stack is up and working. The output looks like this:



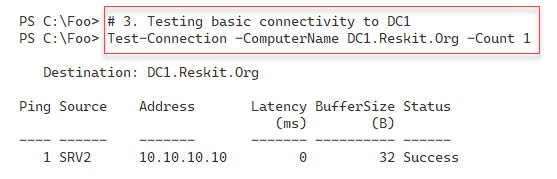
1. Insert image B1672\_01\_07.png

In step 2, you check to ensure that the WinRM port is open and working, with output like this:



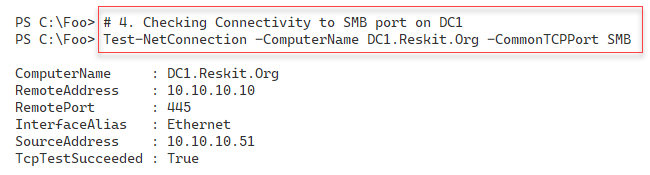
1. Insert image B1672\_01\_08.png

In the Reskit.Org network, DC1 is a domain controller and a DNS server. In step 3, you test the connectivity to this critical enterprise server, with output like this:



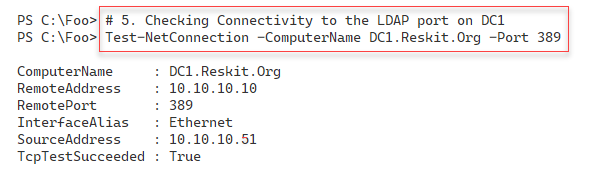
1. Insert image B1672\_01\_09.png

In any domain environment hosts need to access the Sysvol share on a DC to download group policy.POL files. In step 4, you test that SRV2 can access the SMB port, port 445, on the DC, with output like this:



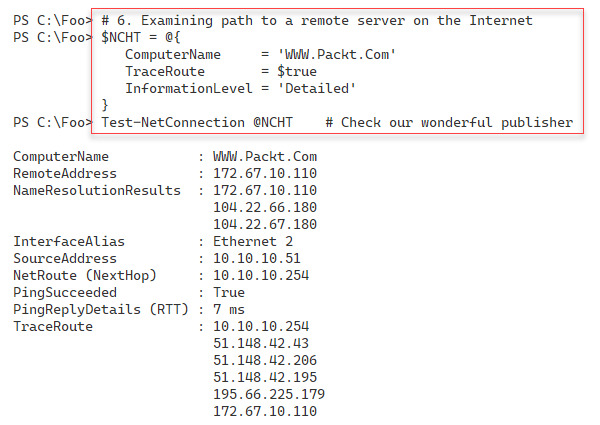
1. Insert image B1672\_01\_10.png

In step 5, you test that SRV2 can access DC1 on the LDAP port, port 389, with the following output:



1. Insert image B1672\_01\_11.png

In step 6, you check connectivity with the Internet and test the network path to the publisher’s website at www.packt.com. The output is:



1. Insert image B1672\_01\_12.png

## There's more...

This recipe's steps confirm the host can accept connections over WinRM and can contact the DC for core activities. You could add several additional tests, such as testing you can access the DNS server and resolve DNS queries.

In step 6, you test the Internet connectivity to our publisher’s website (www.packt.com). Since we are just testing the connectivity to this site, you only need to specify the actual computer name and do not need the HTTP or HTTPS prefix.

# Deploying IIS in a container

In previous recipes, you configured SRV2 with a static IP address and tested its connectivity. Each server needs a unique IP address and other configuration options to configure on a server by server basis. You can also configure client computers running Windows 10 or other OSs manually, although this can be a huge and challenging task in large organisations.

Dynamic Host Configuration Protocol (DHCP) enables a DHCP client to get its IP configuration and other networking details automagically from a DHCP server. DHCP automates IP configuration and avoids the work and avoids the inevitable issues involved with manual IP configuration.

Windows and most other client operating systems, including Linux and Apple Macs, have a built-in DHCP client. Windows Server also includes a DHCP Server service you can install to provide DHCP services to the clients. You can install DHCP using Server Manager and configure the service using the DHCP GUI application. Alternatively, you can automate the installation of DHCP as you can see in this recipe. In the next recipe, “Configure DHCP Scopes and Options”, you configure the DHCP service to issue IP addresses in a specific range. You also configure DHCP to provide DHCP clients with other IP address configuration options, such as the subnet mask, default gateway, and the DNS server IP address or addresses.

## Getting Ready

This recipe uses DC1, a domain controller in the Reskit.Org domain. You should have installed AD on this host and configured it as per earlier recipes in Chapter 5 and Chapter 6.

## How to do it...

1. Install the DHCP Feature on DC1 and add the Management tools

Import-Module -Name ServerManager -WarningAction SilentlyContinue

Install-WindowsFeature -Name DHCP -IncludeManagementTools

1. Adding DC1 to trusted DHCP Servers and add the DHCP Security Group

Import-Module -Name DHCPServer -WarningAction SilentlyContinue

Add-DhcpServerInDC

Add-DHCPServerSecurityGroup

1. Letting DHCP know it's all configured

$DHCPHT = @{

Path = 'HKLM:\SOFTWARE\Microsoft\ServerManager\Roles\12'

Name = 'ConfigurationState'

Value = 2

}

Set-ItemProperty @DHCPHT

1. Restarting DHCP Server

Restart-Service -Name DHCPServer –Force

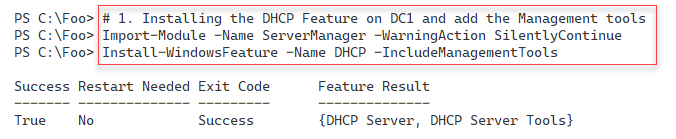
1. Testing service availability

Get-Service -Name DHCPServer |

Format-List -Property \*

## How it works...

In step 1, you import the Server Manager module and use Install-WindowsFeature to add the DHCP server service to DC1. The output from this step looks like this:



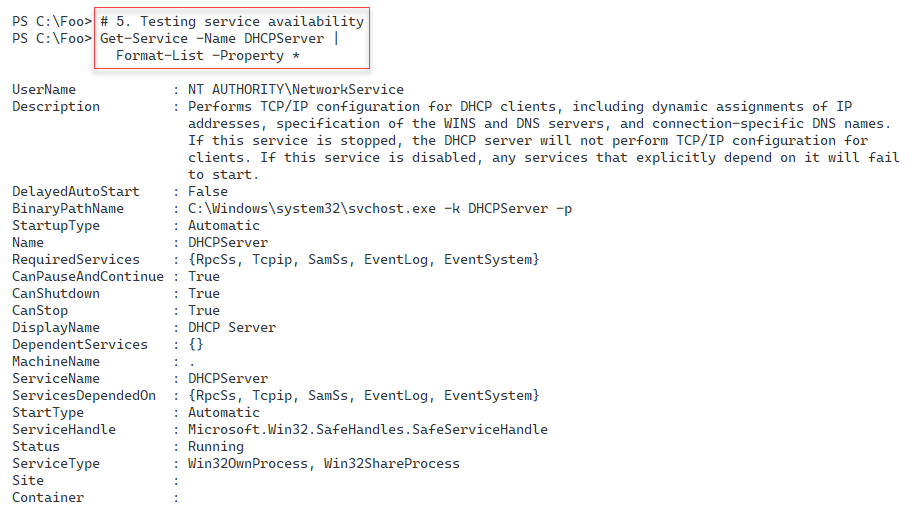
1. Insert image B1672\_01\_13.png

In step 2, you add DC1 to the set of authorized DHCP servers in the domain and add the DHCP security groups to the DHCP server. The groups that this command adds are DHCP Users and DHCP Administrators security groups. For more details on these groups, see https://secureidentity.se/delegate-dhcp-admins-in-the-domain/.

In step 3, you set a registry entry to tell Windows that all post-deployment DHCP configuration activities are complete. The GUI installation process takes you through this automatically. When installing via PowerShell, you need to set the registry entry to complete the configuration.

With you have completed the configuration activities, you restart the DHCP service. Once restarted, the DHCP service can issue IP configuration to DHCP clients. For this to happen, you must also have specified the configuration information you specify in the “Configure DHCP Scopes and Options” recipe. Step 2, step 3. and step 4 produce no output.

In step 5, you complete this recipe by ensuring that the DHCP service has started. The output of this step looks like this:



1. Insert image B1672\_01\_14.png

## There's more...

When the Windows DHCP service starts, it checks to ensure the server is on the DHCP server list authorized in the domain. The DHCP service does not start on any non-authorized DHCP server. By adding DC1 to the list of authorized servers can help to guard against rogue DHCP Servers.

In step 5, you check the DHCP Service. Get-Service's output details the service, including a description and the path name to the actual service executable. The DHCP service does not run in its own process. Instead, it runs inside svchost.exe. It is for this reason that you do not see the service explicitly when you use Get-Process.

# Using a Dockerfile to create a custom container

Installing DHCP is simple, as you see in “Installing DHCP”. You add the Windows feature and then carry out two small configuration steps. In most cases, you probably do not need to take these extra steps. The extra steps enable you to use the relevant security groups and avoid the Server Manager GUI message that there are configuration steps not yet performed.

Before your DHCP server can provide IP address configuration information to DHCP clients, you need to create a DHCP scope and DHCP options. A DHCP scope is a range of DHCP addresses that your DHCP server can give out for a given IP subnet. DHCP options are specific configuration options your DHCP server provides, such as the DNS server's IP address and the IPv4 default gateway.

You can set DHCP options a scope level or at a server level, depending on your organisation's needs. For example, you would most likely specify a default gateway in the Scope options, with DNS server address(es) set at the server level.

In this recipe, you create a new scope for the 10.10.10.0/24 subnet and specify both scopes and server level

## Getting Ready

You run this recipe on DC1, a domain controller in the Reskit.Org domain after installing the DHCP server service. You must have installed PowerShell 7 and VS Code on this host.

## How to do it...

1. Importing the DHCP Server module

Import-Module DHCPServer -WarningAction SilentlyContinue

1. Creating an IPv4 scope

$SCOPEHT = @{

Name = 'ReskitOrg'

StartRange = '10.10.10.150'

EndRange = '10.10.10.199'

SubnetMask = '255.255.255.0'

ComputerName = 'DC1.Reskit.Org'

}

Add-DhcpServerV4Scope @SCOPEHT

1. Getting IPV4 Scopes from the server

Get-DhcpServerv4Scope -ComputerName DC1.Reskit.Org

1. Setting server-wide option values

$OPTION1HT = @{

ComputerName = 'DC1.Reskit.Org' # DHCP Server to Configure

DnsDomain = 'Reskit.Org' # Client DNS Domain

DnsServer = '10.10.10.10' # Client DNS Server

}

Set-DhcpServerV4OptionValue @OPTION1HT

1. Setting a scope specific option

$OPTION2HT = @{

ComputerName = 'DC1.Reskit.Org' # DHCP Server to Configure

Router = '10.10.10.254'

ScopeID = '10.10.10.0'

}

Set-DhcpServerV4OptionValue @OPTION2HT

1. Viewing DHCP server options

Get-DhcpServerv4OptionValue | Format-Table -AutoSize

1. Viewing scope specific options

Get-DhcpServerv4OptionValue -ScopeId '10.10.10.10' |

Format-Table -AutoSize

1. Viewing DHCPv4 option definitions

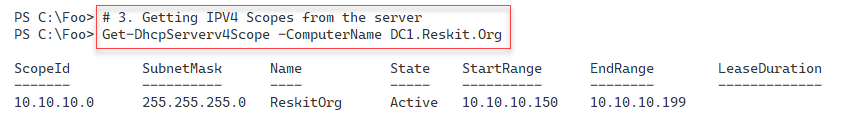
Get-DhcpServerv4OptionDefinition | Format-Table -AutoSize

## How it works...

In step 1, you import the DHCPServer module. When you installed DHCP (in “Installing DHCP”), you added the management tools, including this module. However, the DHCP team have not yet made this module compatible with PowerShell 7. This step, which produces no output, loads the module using the Windows PowerShell compatibility solution.

In step 2, you create a new DHCP scope for IPV4 addresses. The scope enables the DHCP server to issue IP addresses in the range of 10.10.10.150 - 10.10.10.199 range. This step produces no output.

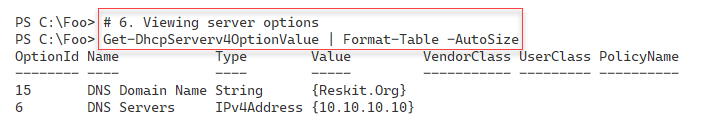
In step 3, you use Get-DHCPServerIPV4Scope to retrieve details of all the DHCP scopes you have defined on DC1. The output of this step looks like this:



1. Insert image B1672\_01\_15.png

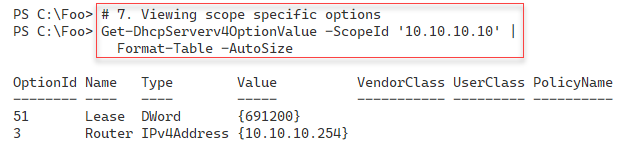
In step 4, you set two server-wide options, creating no output. These are options and values offered to all clients of any DHCP scope defined on this server. In step 5, you specify a scope option. This step also produces no output.

In step 6, you view the DHCP Server-wide options, with output looks like this;



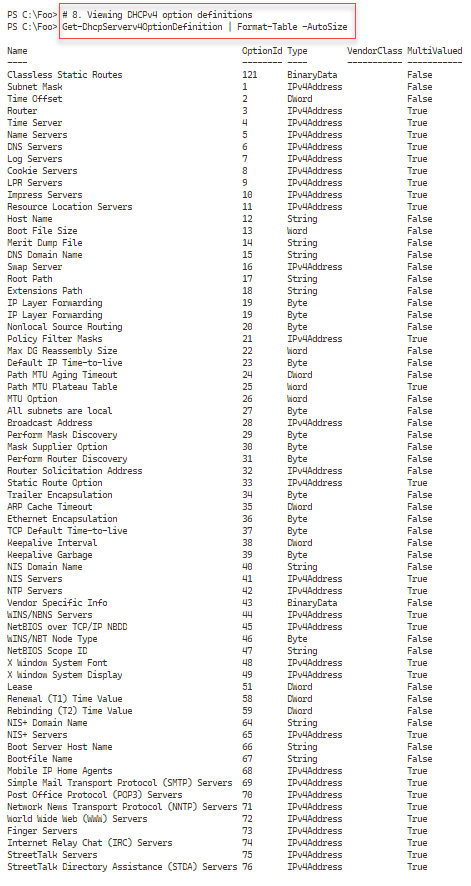
1. Insert image B1672\_01\_16.png

With step 7, you view the options you have set on the 10.10.10.10 scope, which looks like this:



1. Insert image B1672\_01\_17.png

There are 66 DHCP options you can use to provide option values to DHCP Clients. Most of these options are of little use in most cases but provide support for niche and uncommon scenarios. To view the set of options defined by default, in step 8, which looks like this:



1. Insert image B1672\_01\_18.png

## There's more...