9

Managing Windows Internet Information Server

In this chapter, we cover the following recipes:

* Installing IIS
* Configuring IIS for SSL
* Managing TLS cipher suites
* Configuring a central certificate store
* Configuring IIS bindings
* Managing IIS logging and log files
* Managing IIS applications and application pools
* Analyzing IIS log files
* Managing and monitoring Network Load Balancing

# Introduction

Internet Information Services (IIS) is a Windows feature that implements an extensible web server. IIS was first introduced as an add-on for Windows NT 4.0, and has been the focus of substantial development ever since. IIS is an add-on feature that Microsoft has built into both Windows Server 2019 and Windows 10.

With IIS in Windows Server, you can deploy both internet-facing public websites and sites on your internal intranet. You can integrate IIS with enterprise applications, including SharePoint, Exchange, and System Center.

IIS provides a platform for a variety of web-based applications. With IIS, you can provide a simple, HTML-based static website, as well as rich, multi-tiered applications. You can combine the applications running on IIS with backend databases, such as Microsoft SQL Server.

Like other Windows Server features, there is good PowerShell cmdlet coverage for IIS. You can make use of two PowerShell modules: the WebAdministration module, introduced in earlier versions of Windows Server, and the IISAdministration module, which provides additional cmdlets. While many PowerShell modules work well in PowerShell Core, neither of the IIS modules work in PowerShell Core. As a workaround (if you are using PowerShell Core), you can use PowerShell remoting to run these cmdlets on the remote IIS server.

This chapter covers how to install, configure, manage, and maintain IIS on Windows Server 2019. While you can load and use IIS in Windows 10, the focus, in this chapter, is on Windows Server 2019.

# Installing IIS

Before you can use IIS, you must install it onto your host. Like other roles and features of Windows Server 2019 that are covered in this book, you install IIS by using the Install-WindowsFeature cmdlet. Once you have installed the web server, you can take a look at the host.

## Getting ready

This recipe uses SRV1 and assumes a fresh installation. If you have used SRV1 to test previous recipes, you may need to remove the Web-Server feature before you run this recipe. Also, you should have the Windows Server 2019 installation DVD in the D: drive of SRV1.

## How to do it…

1. Add the Web-Server feature, sub-features, and tools to SRV1, as follows:

$FHT = @{

Name = 'Web-Server'

IncludeAllSubFeature = $true

IncludeManagementTools = $true

Source = "D:\sources\sxs"

}

Install-WindowsFeature @FHT

1. See what features are installed:

Get-WindowsFeature -Name Web\* | Where-Object Installed

1. Check the IIS administration modules:

$Modules = @('WebAdministration', 'IISAdministration')

Get-Module -Name $Modules -ListAvailable

1. Get a count of how many commands are in each module:

$C1 = (Get-Command -Module WebAdministration |

Measure-Object |

Select-Object -Property Count).Count

$C2 = (Get-Command -Module IISAdministration |

Measure-Object |

Select-Object -Property Count).Count

"$C1 commands in WebAdministration Module"

"$C2 commands in IISAdministration Module"

1. Get details of the IIS provider contained in the WebAdministration module:

Import-Module -Name WebAdministration

Get-PSProvider -PSProvider WebAdministration

1. You can find out what is in the IIS: drive with the following command:

Get-ChildItem -Path IIS:\

1. You can find out what is in the Sites folder with the following command:

Get-Childitem -Path IIS:\Sites

1. Look at the default website, as follows:

$IE = New-Object -ComObject InterNetExplorer.Application

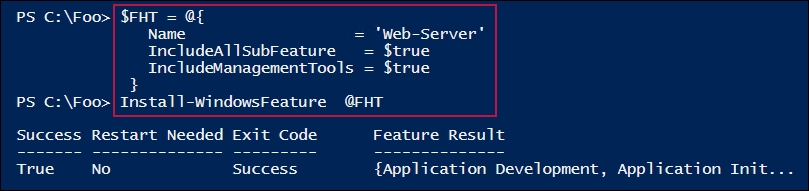
$URL = 'HTTP://SRV1'

$IE.Navigate2($URL)

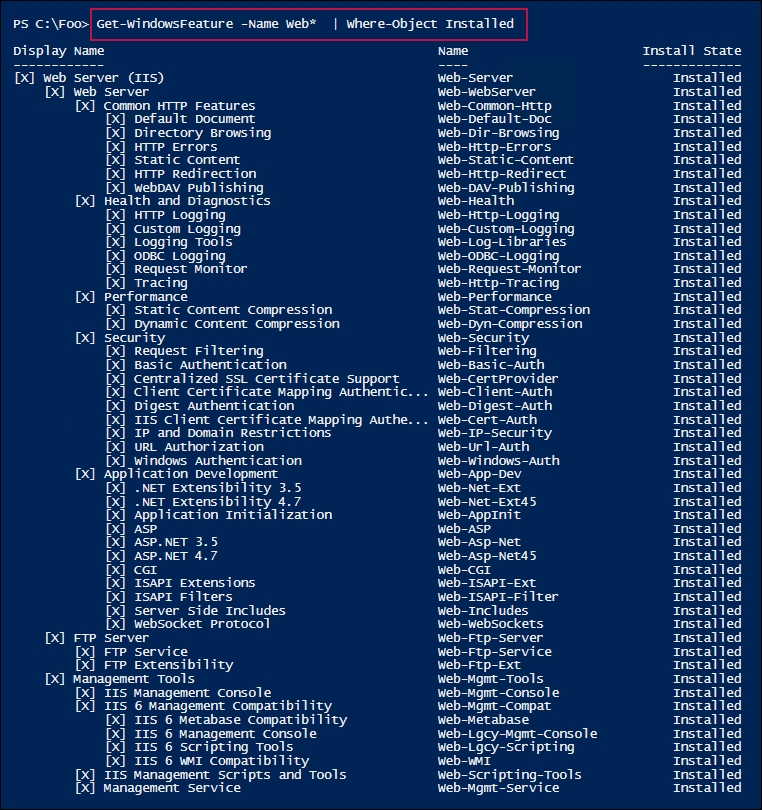
$IE.Visible = $true

## How it works…

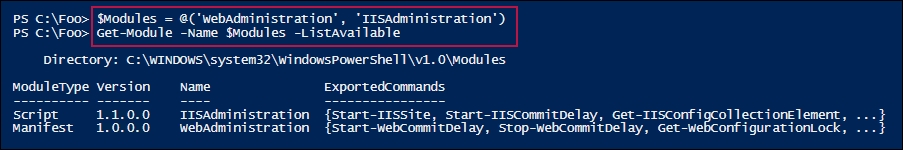
In step 1, you use the Install-WindowsFeature cmdlet to install IIS, as well as a number of the web server sub-features and management tools, which look like this:



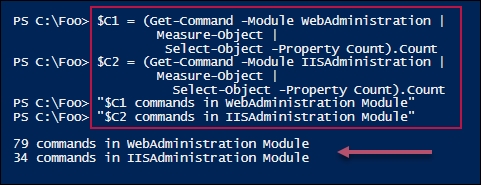
In step 2, you use the Get-WindowsFeature cmdlet to retrieve the web server related features installed on SRV1, the output for which looks like this:



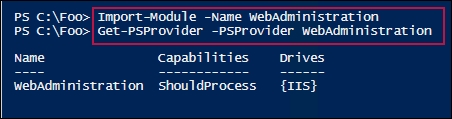
In step 3, you get the IIS-related modules on SRV1, which produces the following output:



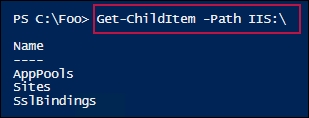
In step 4, you get a count of the number of commands in the WebAdministration and IISAdministrtion modules, as follows:



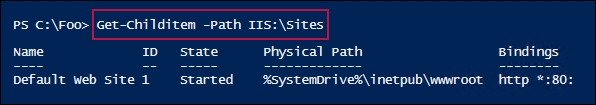
In step 5, you import the WebAdministration module, which loads the IIS provider. Then you get details of the provider, as follows:



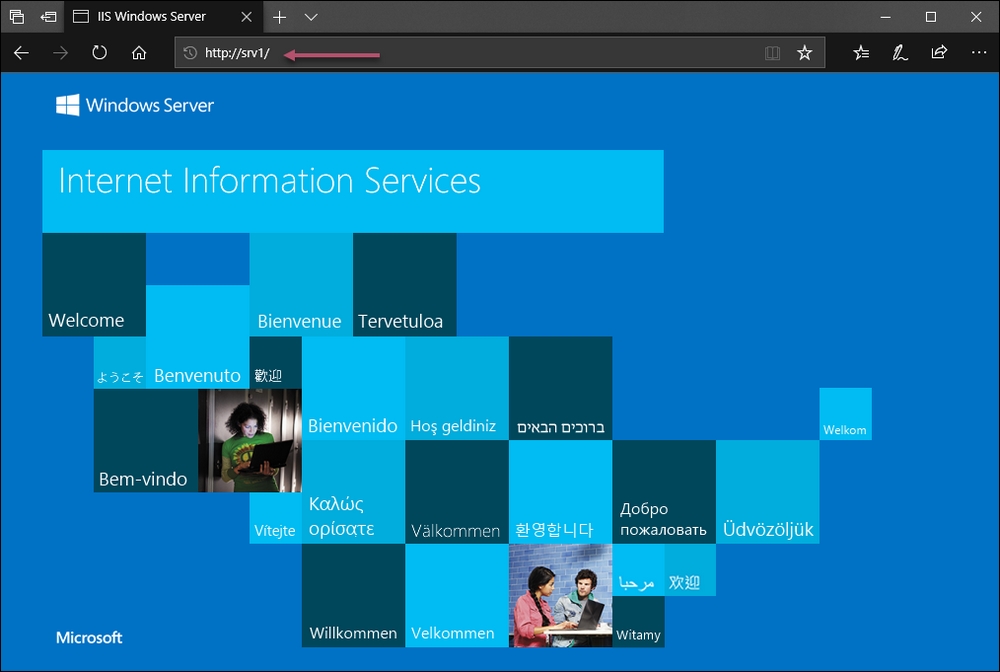
In step 6, you use the provider to view the contents of IIS:, which looks as follows:



In step 7, you view the contents of IIS:\Sites, which looks like this:



In step 8, you use Internet Explorer to view the web service on SRV1, which looks like this:



## There's more...

In step 1, you installed the IIS management tools. These tools include the IIS GUI tool, plus the WebAddministration and IISAdministration PowerShell modules.

In step 5, you import the WebAdministration module manually. In addition to loading the cmdlets/functions contained in the module, when you import the module, PowerShell loads the WebAdministration PowerShell provider. This provider enables you to browse aspects of the web server, including the sites, application pools, and SSL bindings on the host. You use this feature in later recipes in this chapter. When you use any of the cmdlets in the WebAdministration module, PowerShell, by default, auto-loads the module. But if you only want to use the provider, you have to manually load the module first, as shown in this recipe.

In step 8, you view the default website that the installation process adds for you. Browsing to the server is a great way to determine that IIS is loaded and is running on SRV1.

# Configuring IIS for SSL

Traffic between a web browser and a web server on the internet, or even within a corporate intranet, is open, and can be intercepted. To avoid the data being compromised, you can make use of protocols built into your web browser, along with IIS, to provide encryption, as well as authentication.

In the 1990s, Netscape Communications developed a protocol that provided some necessary security, in the form of the Secure Socket Layer (SSL) protocol. SSL 1.0 was never commercially released, while SSL 2.0 and SSL 3.0 were developed and released, but are now deprecated as unsafe.

Transport Layer Security (TLS) was developed openly as the next version of SSL. TLS 1.0 is essentially SSL 3.1. In 2014, Google identified a serious vulnerability in both SSL 3.0 and TLS 1.0. That leaves TLS 2.0 as the best protocol to deploy, and it is the only one installed by default with IIS in Windows Server 2019.

These days, SSL, as a protocol, is being deprecated in favor of TLS. Most major websites no longer actually use the SSL protocol. Nevertheless, we refer to such websites as using SSL, and we continue to use the HTTPS scheme, since end users cannot explicitly choose between SSL and TLS.

When the user specifies a URL beginning with HTTPS:, the browser contacts the server on port 443. The browser and server then negotiate which security protocol to use (for example, TLS 2.0) and which cipher suite to use to protect the data being transferred. A cipher suite is a distinct set of algorithms to provide for key exchange and the encryption algorithms to be used for both bulk encryption and hashing.

In order to set up IIS for secure transfer, first, you need a certificate. The certificate identifies the server by name and specifies what the certificate can be used for. Public and private keys are associated with the certificate.

If you are setting up IIS as an internal web server, then you should use your internal Certificate Authority (CA) to create the web server certificate. If your web server is to be internet-facing, then you should get a certificate from a public CA. Remember that the certificate should be issued (and signed) by a CA that is explicitly trusted by any client accessing the secure site. Many public CAs around the world are automatically trusted by most modern browsers. Additionally, you can configure workstations and servers to enroll the root CA certificate for your internal CA automatically.

In this recipe, you use self-signed certificates. This works wonderfully in a test environment, but should never be used in production. The technique that you use in this recipe first generates a self-signed certificate, which you then copy into the local machine's trusted root store. This action makes the local machine trust the self-signed certificate. Should you access the server from any other host, the browser generates certificate errors, since those other machines do not trust the certificate.

## Getting ready

You need to run this recipe on SRV1, after you have installed IIS, as you did in the Installing IIS recipe.

## How to do it...

1. Import the WebAdministration module:

Import-Module -Name WebAdministration

1. Create a self-signed certificate:

$CHT = @{

CertStoreLocation = 'CERT:\LocalMachine\MY'

DnsName = 'SRV1.Reskit.Org'

}

$SSLCert = New-SelfSignedCertificate @CHT

1. Copy the certificate to the root store on SRV1:

$C = 'System.Security.Cryptography.X509Certificates.X509Store'

$AL = 'Root', 'LocalMachine'

$Store = New-Object -TypeName $C -ArgumentList $AL

$Store.Open('ReadWrite')

$Store.Add($SSLcert)

$Store.Close()

1. Create a new SSL binding on the Default Web Site:

New-WebBinding -Name 'Default Web Site' -Protocol https -Port 443

1. Assign the certificate that was created earlier to this new binding:

$SSLCert | New-Item -Path IIS:\SslBindings\0.0.0.0!443

1. View the site using HTTPS:

$IE = New-Object -ComObject InterNetExplorer.Application

$URL = 'https://SRV1.Reskit.Org'

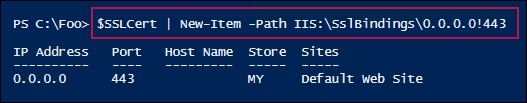
$IE.Navigate2($URL)

$IE.Visible = $true

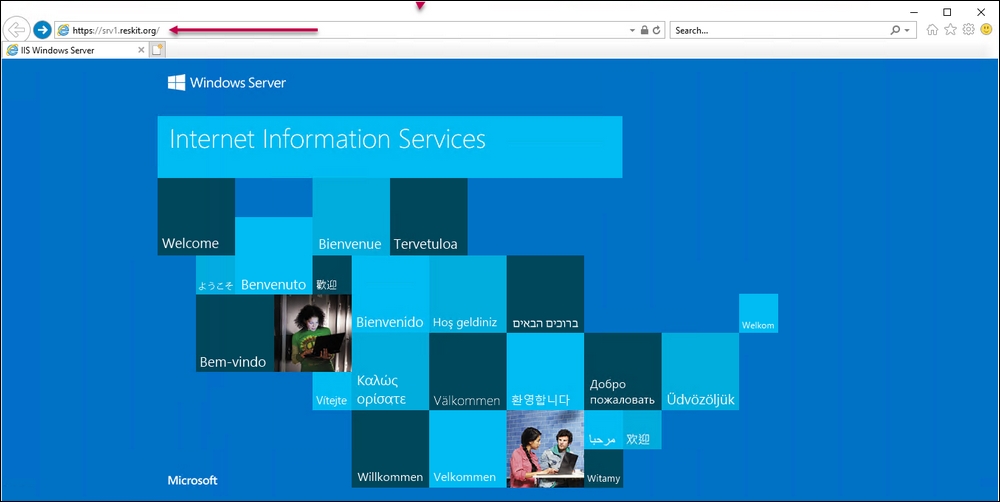
## How it works…

In step 1, you import the web administration module manually. In step 2, you create a self-signed certificate for SRV1, and in step 3, you copy that newly created certificate to the local machine's root certificate store. In step 4, you create a new binding for port 443 on the default website. These steps produce no output.

In step 5, you assign your self-signed certificate to the default website, which looks like this:



In step 6, you use HTTPS to view the default website on SRV1, which looks like this:



## There's more...

In this recipe, you manually import the WebAdministration module (in step 1). You need to do this because in step 5, you use the IIS Provider to bind the certificate to the SSL/TLS port (that is, 443). Since you were not using any of the cmdlets in the module, you have to import the module before you can use the provider.

In step 3, you use the .NET framework to copy the self-signed certificate into the local server's trusted root certificate store. This enables SRV1 to trust the self-signed certificate. You have to use .NET because the PowerShell certificate provider does not support a copy operation. As an alternative, you could use the Export-Certificate and Import-Certificate to export the certificate to a file, and then re-import it.

The output shown for step 6 in this recipe is identical to the output for step 8 in the Install IIS recipe, except that in this recipe, you used TLS to view the web page.

# Managing TLS cipher suites

A cipher suite is a specific set of methods or algorithms that provide functions, including key exchange, bulk encryption, hashing, and creating message digests. Numerous Windows services, such as TLS, SSH, and IPSEC, make use of cipher suites when communicating with other hosts. With TLS, you can use the TLS cipher suite cmdlets to manage the cipher suites that your IIS web server is going to negotiate (or not).

Once the browser connects to the web server, the web server and the browser negotiate and choose the best cipher suite that both sides can support. If the browser only asks for cipher suites that the web server does not support, then the server terminates the communication.

By default, Windows Server 2019 supports 31 cipher suites, providing different algorithms and key lengths. In this recipe, you retrieve the cipher suites on Windows Server 2019, and both enable and disable a specific cipher suite.

## Getting ready

This recipe makes use of SRV1, after you have run the Installing IIS recipe.

## How to do it...

1. Get the cipher suites on SRV1 and display them, as follows:

Get-TlsCipherSuite |

Format-Table Name, Exchange, Cipher, Hash, Certificate

1. Find the cipher suites that support 3DES with the following command:

Get-TlsCipherSuite -Name 3DES |

Format-Table Name, Exchange, Cipher, Hash, Certificate

1. Disable the 3DES-based cipher suites:

Foreach ($CS in (Get-TlsCipherSuite -Name '3DES'))

{Disable-TlsCipherSuite -Name $CS.Name}

1. Check whether any cipher suites that support 3DES remain:

Get-TlsCipherSuite 3DES |

Format-Table Name, Exchange, Cipher, Hash, Certificate

1. Re-enable the 3DES-based cipher suite:

Enable-TlsCipherSuite -Name TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

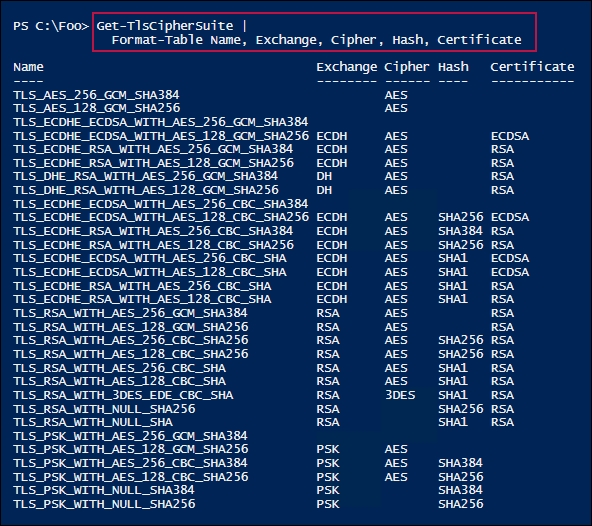
1. Check for enabled cipher suites that support 3DES:

Get-TlsCipherSuite 3DES |

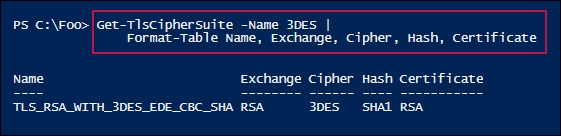
Format-Table -Property Name, Exchange, Cipher, Hash, Certificate

## How it works…

In step 1, you get the available cipher suites on SRV1, which looks like this:

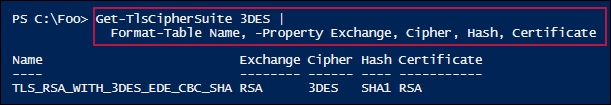


In step 2, you discover which cipher suites utilize 3DES, which looks like this:



In step 3, you disable any cipher suites based on 3DES, which produces no output. In step 4, you re-query to check for any 3DES based cipher suites—having just removed that suite, this step produces no output. In step 5, you re-enable the cipher suite named TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA (which you previously disabled). This step also produces no output.

In the final step, step 6, you check to see which TLS cipher suites are enabled that support 3DES, which looks like this:



## There's more...

In step 5, you re-enabled the 3DES-based cipher suite that you disabled in step 3. You enabled the cipher suite using its full name. There is no cmdlet that can show you which cipher suites you have on your system that are now disabled—you can only see which ones are specifically enabled, so be careful when disabling cipher suites you may subsequently need to re-enable!

## See also

For more details on cipher suites in Windows, see <https://docs.microsoft.com/windows/desktop/secauthn/cipher-suites-in-schannel>.

The 3DES cryptographic algorithm uses a block cipher mechanism based on 64-bit blocks. The Sweet32 Birthday attacks show the vulnerability potential of using block ciphers with small block sizes, which includes 3DES. For this reason, and especially if you are using IIS on a large-scale, internet-facing server, you should consider disabling the 3DES-based cipher, as shown in the recipe. For more information on block ciphers and the Sweet32 Birthday attacks, see <https://sweet32.info/>.

# Configuring a central certificate store

If you host multiple secure servers at the internet scale, you may find that certificate management can be challenging. Each time you add a new IIS host into your infrastructure, you need to ensure that all of the correct certificates are in place on that host and the correct web binding (binding the certificates to IIS) is in place for each secure site. To add to the workload, you need to deal with certificate expiration and the renewal of certificates across each IIS server that utilizes those certificates.

Windows 2019 includes a feature, the Central Certificate Store (CCS), that simplifies managing certificates. With CCS, you store certificates in a central location, such as on an SMB file share, and use IIS to load certificates from the central CCS share.

In this recipe, you configure SRV1 to use a new share on DC1, which holds the CCS SMB share. You create the certificate store, create a new certificate for SRV1, and move that certificate to the central certificate share on DC1. You then configure IIS to make use of the central store, rather than using the local certificate stores, like you did in the Configuring IIS for SSL recipe.

## Getting ready

This recipe uses two servers: SRV1 is an IIS server and DC1 is configured to hold the SSL CCS. You should have both servers up and running. Also, this recipe assumes that you have IIS at least partly loaded and set up for SSL (in other words, you should have run the Install IIS and Configure IIS for SSL recipes in advance). This recipe does check and ensure that the needed features are added to SRV1. You should also ensure that the Active Directory (AD) cmdlets are loaded on SRV1.

## How to do it...

1. Remove the existing certificates from SRV1:

Get-ChildItem Cert:\localmachine\My |

Where-Object Subject -Match 'SRV1.Reskit.Org' |

Remove-Item -ErrorAction SilentlyContinue

Get-ChildItem Cert:\localmachine\root |

Where-Object Subject -match 'SRV1.Reskit.Org' |

Remove-Item

1. Remove SSL web bindings, if any exist:

Import-Module -Name WebAdministration

Get-WebBinding |

Where-Object protocol -EQ 'https' |

Remove-WebBinding

Get-ChildItem IIS:\SslBindings |

Where-Object Port -eq 443 |

Remove-Item

1. Create a shared folder and share it on DC1:

$SB = {

If ( -NOT (Test-Path c:\SSLCerts)) {

New-Item -Path c:\SSLCerts -ItemType Directory |

Out-Null}

$SHAREHT= @{

Name = 'SSLCertShare'

Path = 'C:\SSLCerts'

FullAccess = 'Everyone'

Description = 'SSL Certificate Share'

}

New-SmbShare @SHAREHT

'SSL Cert Share' | Out-File C:\SSLCerts\Readme.Txt

}

Invoke-Command -ScriptBlock $SB -ComputerName DC1 |

Out-Null

1. Check the share on DC1:

New-SmbMapping -LocalPath X: -RemotePath \\DC1\SSLCertShare |

Out-Null

Get-ChildItem -Path X:

1. Add new SSL certificates to the root certificate store on SRV1:

$SSLHT = @{

CertStoreLocation = 'CERT:\LocalMachine\MY'

DnsName = 'SRV1.Reskit.Org'

}

$SSLCert = New-SelfSignedCertificate @SSLHT

$C = 'System.Security.Cryptography.X509Certificates.X509Store'

$NOHT = @{

TypeName = $C

ArgumentList = 'Root','LocalMachine'

}

$Store = New-Object @NOHT

$Store.Open('ReadWrite')

$Store.Add($SSLcert)

$Store.Close()

1. Export the certificate to a PFX file:

$CertPW = 'SSLCerts101!'

$SSHT = @{

String = $CertPW

Force = $true

AsPlainText = $True

}

$Certpwss = ConvertTo-SecureString @SSHT

$CertHT = @{

Cert = $SSLCert

FilePath = 'C:\SRV1.Reskit.Org.pfx'

Password = $Certpwss

}

Export-PfxCertificate @CertHT

1. Move the certificate to the SSLCertShare share on DC1:

$MHT = @{

Path = 'C:\SRV1.Reskit.Org.pfx'

Destination = '\\DC1\SSLCertShare\SRV1.Reskit.Org.Pfx'

Force = $True

}

Move-Item @MHT

1. Install the CCS feature on SRV1:

Install-WindowsFeature Web-CertProvider | Out-Null

1. Create a new user for the certificate sharing:

$User = 'Reskit\SSLCertShare'

$Password = 'Pa$$w0rd'

$SSHT2 = @{

String = $Password

AsPlainText = $true

$Force = $True

}

$PSS = ConvertTo-SecureString @SSHT2

$NewUserHT = @{

AccountPassword = $PSS

Enabled = $true

PasswordNeverExpires = $true

ChangePasswordAtLogon = $false

SamAccountName = 'SSLCertShare'

UserPrincipalName = 'SSLCertShare@Reskit.Org'

Name = 'SSLCertShare'

DisplayName = 'SSL Cert Share User'

}

New-ADUser @NewUserHT

1. Configure the SSL CSS in the registry:

$IPHT = @{

Path = 'HKLM:\SOFTWARE\Microsoft\IIS\CentralCertProvider\'

Name = 'Enabled'

Value = 1

}

Set-ItemProperty @IPHT

$IPHT.Name = 'CertStoreLocation'

$IPHT.Value = '\\DC1\SSLCertShare'

Set-ItemProperty @IPHT

1. Enable the SSL CCS, as follows:

$WCHT = @{

CertStoreLocation = '\\DC1\SSLCertShare'

UserName = $User

Password = $Password

PrivateKeyPassword = $Certpw

}

Enable-WebCentralCertProvider @WCHT

$CPHT = @{

UserName = 'Reskit\SSLCertShare'

Password = $Password

PrivateKeyPassword = $Certpw

}

Set-WebCentralCertProvider @CPHT

1. Set up SSL for a default site:

New-WebBinding -Name 'Default Web Site' -Protocol https -Port 443

$SslCert | New-Item -Path IIS:\SslBindings\0.0.0.0!443

1. Remove the certificate from SRV1:

Get-ChildItem Cert:\LocalMachine\My |

Where-Object Subject -Match 'SRV1.RESKIT.ORG' |

Remove-Item -Force

1. Now, view the website with SSL, as follows:

$IE = New-Object -ComObject InterNetExplorer.Application

$URL = 'HTTPS://SRV1.Reskit.Org/'

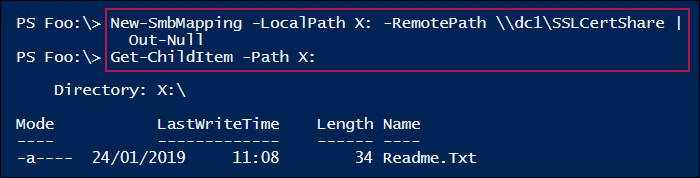
$IE.Navigate2($URL)

$IE.Visible = $true

## How it works…

In step 1, you remove all certificates from the LocalMachine's MY certificate store. In step 2, you remove any SSL bindings for SRV1. In step 3, you create a new folder to hold centrally provided certificates, and share it on DC1. These three steps produce no output.

In step 4, you check the contents of the \\DC1\SSLCertShare share, which looks like this:



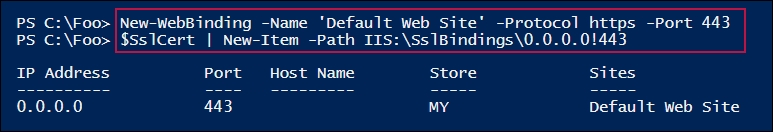
In step 5, you create a self-signed certificate in the LocalMachine\MY certificate store. This certificate is then copied into the LocalMachine\Root certificate share. This step produces no output.

In step 6, you export the newly created self-signed certificate to a PFX file, which looks like this:

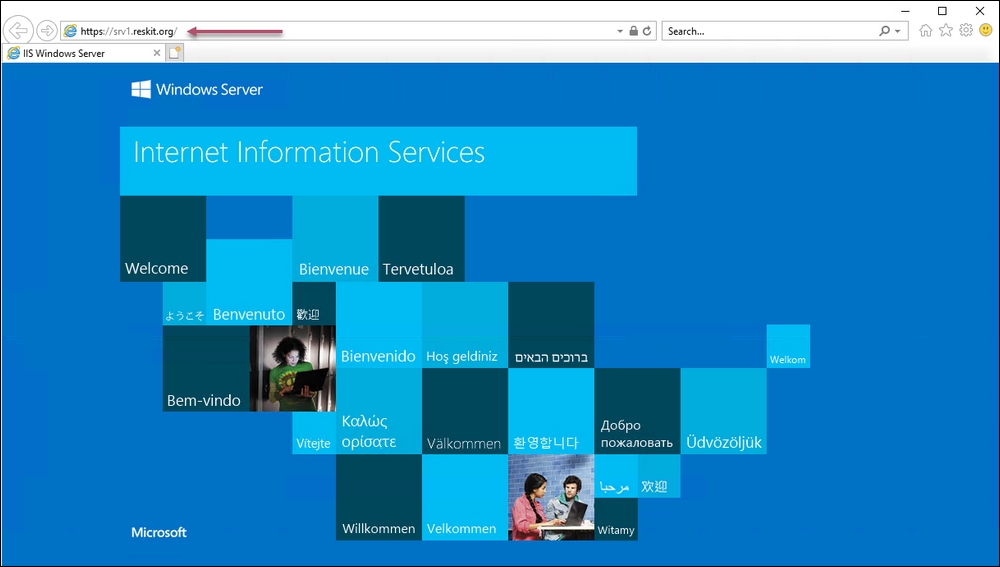


In step 7, you move the PFX file containing the certificate to the SSLCertShare share on DC1. In step 8, you install the Web-CertProvider feature to SRV1. In step 9, you create a new user in the AD for certificate sharing. In step 10, you configure the SSL CCS in the registry on SRV1. Then, in step 11, you enable the SSL CCS. These five steps produce no output.

In step 12, you set an initial web binding, which looks like this:



With step 13, you remove the certificate from the local certificate store, producing no output. Finally, in step 14, you use Internet Explorer to navigate to the default website on SRV1 using HTTPS:, which looks like this:



## There's more...

In step 5, you created a self-signed certificate in the LocalMachine\MY certificate store; then, you copied this certificate into the LocalMachine\Root certificate share. The copy operation has the effect of making SRV1 trust the newly created certificate. In production, you would create a certificate from a trusted Certificate Authority and avoid copying the certificate (since the system would already trust that CA).

# Configuring IIS bindings

In IIS, a binding consists of an IP address, a port, and a host header on which the web server listens for requests made to that website. The binding tells IIS how to route inbound HTTP/HTTPS requests.

In this recipe, you create a new website on SRV1 and add bindings to enable the site. In this recipe, you only bind for HTTP.

## Getting ready

You need to run this recipe on SRV1 after installing IIS (which you did in the Installing IIS recipe).

## How to do it...

1. Import the WebAdministration module:

Import-Module -Name WebAdministration

1. Create and populate a new page:

$SitePath = 'C:\inetpub\www2'

New-Item $SitePath -ItemType Directory | Out-Null

$page = @'

<!DOCTYPE html>

<html>

<head><title>Main Page for WWW2.Reskit.Org</title></head>

<body><p><center>

<b>HOME PAGE FOR WWW2.RESKIT.ORG</b></p>

This is the root page for this site

</body></html>

'@

$PAGE | Out-File -FilePath $SitePath\INDEX.HTML | Out-Null

1. Create a new website that uses host headers:

$WSHT = @{

PhysicalPath = $SitePath

Name = 'WWW2'

HostHeader = 'WWW2.Reskit.Org'

}

New-Website @WSHT

1. Create a DNS record on DC1 for WWW2.Reskit.Org:

Invoke-Command -Computer DC1.Reskit.Org -ScriptBlock {

$DNSHT = @{

ZoneName = 'Reskit.Org'

Name = 'www2'

IpAddress = '10.10.10.50'

}

Add-DnsServerResourceRecordA @DNSHT

}

1. Finally, show the site, as follows:

$IE = New-Object -ComObject InterNetExplorer.Application

$URL = 'Http://WWW2.Reskit.Org'

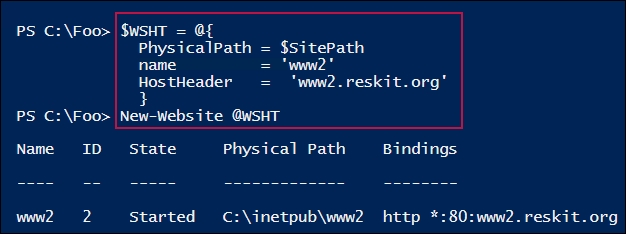
$IE.Navigate2($URL)

$IE.Visible = $true

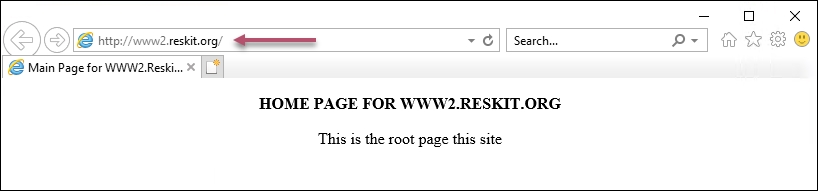
## How it works…

In step 1, you import the WebAdministration module explicitly, in order to load the IIS provider. In step 2, you create a new page on SRV1 that serves as the new site's home page. These two steps produce no output.

In step 3, you create a new IIS website on SRV1; namely, WWW2.Reskit.Org. The output of this step looks like this:



In step 4, you create a new DNS record to point to this new site, which produces no output. In step 5, you use Internet Explorer to navigate to the new site, which looks like this:



## There's more...

By default, while you can have as many HTTP-based sites as you want on a given machine, you can only have one HTTPS site. This is because the details of the site that the browser is asking for are inside the encrypted content, and thus, can only be actioned once decrypted.

To overcome this limitation, IIS uses a TLS feature known as Server Name Indication (SNI). SNI allows the name of the hostname being contacted to be specified during the SSL/TLS handshake. This, in turn, enables IIS to support more than one secure site. To use SNI, the browser or web client, as well as the web server, must support SNI. Modern web browsers support SNI, which has been a feature of IIS on Windows Server for many versions.

In step 4, you created a DNS A record that points to the site at 10.10.10.50. If your site is on a different IP address, change this input accordingly.

## See also

SNI has been a feature of IIS for some time. See <https://docs.microsoft.com/iis/get-started/whats-new-in-iis-8/iis-80-server-name-indication-sni-ssl-scalability> for details of the SNI feature in IIS in Windows Server. You can find more details on SNI in general at <http://en.wikipedia.org/wiki/Server_Name_Indication>.

# Managing IIS logging and log files

By default, every time IIS receives a request from a client, it logs that request to a log file. You can use PowerShell to modify this behavior. You can turn off logging, change the logging frequency, or changing the folder where IIS stores its log files.

Log files are a great place to look when troubleshooting a web server or when analyzing your website's traffic. IIS web server logs are also useful for capacity planning and to help you to analyze the behavior of the traffic (for example, where is it coming from, what clients are being used, which sites/pages are being accessed, and so on).

By default, IIS creates a separate log file every day. This has advantages, but on a busy web server with many sites, managing log files can become a challenge. A web server that has been up and running for a month would have 30 separate log files. You may wish to change log files less frequently.

Also, since IIS puts log files inside the C:\Windows folder by default, you may wish to change the location of log files. You could move the log files to a separate folder in your C:\ drive, or to a separate drive altogether, depending on the sizes of your IIS logs.

In this recipe, you configure logging in IIS using PowerShell and the IIS provider that you load from the WebAdministration module. This technique uses Set-ItemProperty to configure IIS logging.

## Getting ready

This recipe assumes that you have installed IIS, as per the Installing IIS recipe.

## How to do it...

1. Import the web administration module to ensure that the IIS provider is loaded:

Import-Module WebAdministration

1. Look at where you are currently storing log files:

$IPHT1 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logfile.directory'

}

$LogfileLocation = (Get-ItemProperty @IPHT1).value

$LF = [System.Environment]::ExpandEnvironmentVariables("$LF")

Get-ChildItem $LogFileFolder -Recurse

1. Change the folder to C:\IISLogs:

$IPHT2 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logfile.directory'

}

Set-ItemProperty @IPHT2 -Value 'C:\IISLogs'

New-Item -Path C:\IISLogs -ItemType Directory |

Out-Null

1. Change the logging style, as follows:

$IPHT3 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logFile.logFormat'

}

Set-ItemProperty @IPHT3 -Value 'W3C'

1. Change logging change file frequency:

$IPHT3 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logFile.period'

}

Set-ItemProperty @IPHT3 -Value Weekly

1. Change the logging to use a maximum log size:

$IPHT4 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logFile.period'

}

Set-ItemProperty @IPHT4 -Value 'MaxSize'

$Size = 1GB

$IPHT5 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logFile.truncateSize'

}

Set-ItemProperty @IPHT5 -Value $size

1. Disable logging, as follows:

$IPHT5 = @{

Path = 'IIS:\Sites\Default Web Site'

Name = 'logFile.enabled'

}

Set-ItemProperty @IPHT5 -Value $False

1. Delete all of the log files over 30 days old, as shown here:

$LogDirs = Get-ChildItem -Path IIS:\Sites |

Get-ItemProperty -Name logFile.directory.value |

Select -Unique

$Age = 30 # days to keep log files

$DaysOld = (Get-Date).AddDays(-$Age) # how long ago that was

Foreach ($LogDir in $LogDirs){

$Dir = [Environment]::ExpandEnvironmentVariables($LogDir)

Get-ChildItem -Path $Dir -Recurse -ErrorAction SilentlyContinue |

Where-Object LastWriteTime -lt $DaysOld |

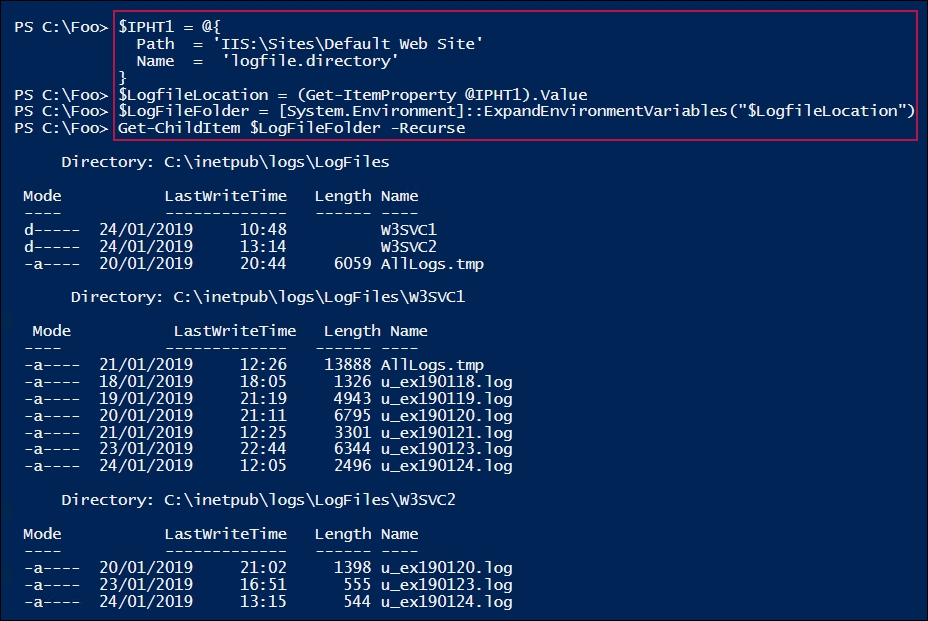
Remove-Item

}

## How it works…

In step 1, you import the WebAdministration module explicitly, which loads the IIS provider, creates an IIS: PSDrive on your system, and produces no output.

In step 2, you use the IIS provider to retrieve the location of IIS logs for SRV1 and display the existing log files. Depending on how much you have used the two websites created in this chapter's recipes (that is, the default website created in the Installing IIS recipe and the WWW2 site created in the Configuring IIS bindings recipe), the output might look like this:



In step 3, you create a folder at C:\IISLogs and change the log file folder to this one. In step 4, you change the logging type to W3C; in step 5, you set the frequency that IIS uses to change log files, and in step 6, you set a maximum log file size. In step 7, you disable logging for the default website. Finally, in step 8, you remove any log files over 30 days old. These six steps produce no output.

## There's more...

In step 2, you look at the log files created so far. Your output is likely to be different, depending on how many clients you have used to access the two websites created in the recipes in this chapter. In the output, you may notice that the logging file folder root (C:\inetpub\logs\LogFiles) has two sub-folders that both contain individual daily log files. The log files that you see in the W3SVC1 sub-folder relate to the default website, while the log files in the W3SVC2 sub-folder relate to the WWW2.Reskit.Org site.

In step 3, you changed the folder that holds the IIS logs. In production, you may choose to hold IIS log files on separate disks, which is a best practice.

In step 4, you adjusted the log file format for IIS logging to W3C. You have several options for log file formats. See <https://docs.microsoft.com/iis/manage/provisioning-and-managing-iis/configure-logging-in-iis> for more information on IIS log file formats.

In step 6, you change the logging to use a maximum size log file. This does keep the size of log files in check, but means that you may not record all events.

In step 7, you disable logging for the default website. This would enable your IIS server to be a little more efficient (due to not having to log events) and means that runaway disk space usage is less likely. As with most Windows logging, IIS logging is turned on by default, but you can easily turn it off or reconfigure it, based on your requirements.

In step 8, you deleted any log files over 30 days old. Instead of deleting them, you may wish to copy them to a central site for more in-depth analysis. The regular logs are great for simple analysis, but for longer term capacity planning, having more data could be useful.

You may also want to keep your log files on your web servers for less than 30 days. To avoid the logs from clogging up your web servers, you can update this recipe to copy the oldest log file(s) to a central repository.

Log files can consume a significant amount of space. You might also consider saving them to a folder that you compress using NTFS file compression, or by using something like WinZIP or WinRAR to compress the files (and decompress them, if you need to do more analysis).

## See also

For more information on how to resolve application issues using IIS log files, see <https://www.sumologic.com/blog/log-management-analysis/iis-logs-troubleshooting/>.

# Managing IIS applications and application pools

In the earliest versions of IIS, all of the web pages and sites on a given system ran in a single Windows process. This meant that one application, if not written well, could cause issues with other applications. An application with a memory or handle leak would eventually require a restart of the single process (or even a reboot of the server).

In later versions of IIS, Microsoft added the concept of web applications and application pools to IIS. With IIS, a web application is a set of one or more URLs (web pages). For example, the pages for the WWW2.Reskit.Org example that you created in the Configuring IIS bindings recipe are stored in C:\inetpub\www2 on SRV1. You can configure IIS to run different web applications inside of independent worker processes. This means that your default website and the WWW2 site could run inside of totally different worker processes, and issues with one are not going to affect the other.

An application pool is a set of worker processes that IIS uses to run a specific application. You can run one or more applications within a given application pool, or run each application in separate application pools. Technically, a website and a web application are not the same, but in many cases, different websites end up being distinct applications.

The application pool feature provides application isolation, enabling you to run possibly unstable applications independently of others. And since you can configure an application pool to run more than one worker process, application pools provide a degree of scalability (taking use of multiple cores on modern processors).

With application pools, IIS can spawn numerous threads in each worker process that IIS runs in parallel, which takes advantage of today's multi-core processors. IIS can create and destroy worker processes on demand, adding more when the workload is higher, and destroying them when they are not needed.

You can also set up the worker processes to be recycled by IIS (that is, stop, then restart). Thus, if an unstable application contains a memory leak (something quite possible when using older ISAPI technologies, for example), recycling the process returns the leaked resources back to the OS. Thus, even a very poorly written application can run reasonably well inside IIS.

There are a variety of conditions that you can set to trigger recycling on an application pool. You can set a schedule of when to recycle; you can recycle if the private memory exceeds a predetermined value (for example, 1 GB), or after a certain number of requests (such as recycling the application pool after 1 million hits).

For fuller details, see <https://technet.microsoft.com/library/cc745955.aspx>.

This page relates to IIS 7, but the details are still the same for the version of IIS shipped both with Windows 10 and Server 2019. Another nice feature of application pools is that you can configure each application pool with separate credentials, which provides increased security of IIS applications. For example, an HR application could run using the credentials Reskit\HRApp, while you could configure an accounting web application to run as Reskit\AccountApp. You could then set up Access Control Lists on various resources (files, SQL databases, and so on) based on these user IDs.

In this recipe, you create a new IIS web application based on the WWW2 site that you created in the Configuring recipe. The recipe also creates and configures an application pool that hosts the application/website.

## Getting ready

You need to run this recipe on SRV1, which you configured with IIS (created in the Installing IIS recipe) and with the WWW2 site (created in the Configuring IIS bindings recipe).

## How to do it...

1. Import the web administration module:

Import-Module -Name WebAdministration

1. Create a new application pool:

New-WebAppPool -Name WWW2Pool

1. Create a new application in the pool:

$WAHT = @{

Name = 'WWW2'

Site = 'WWW2'

ApplicationPool = 'WWW2Pool'

PhysicalPath = 'C:\inetpub\WWW2'

}

New-WebApplication @WAHT

1. View the application pools on SRV1:

Get-IISAppPool

1. Set the application pool restart time, as follows:

$IPHT1 = @{

Path = 'IIS:\AppPools\WWW2Pool'

Name = 'Recycling.periodicRestart.schedule'

}

Clear-ItemProperty @IPHT1

$RestartAt = @('07:55', '19:55')

New-ItemProperty @IPHT1 -Value $RestartAt

1. Set the application pool maximum private memory, as follows:

$IPHT2 = @{

Path = 'IIS:\AppPools\WWW2Pool'

Name = 'Recycling.periodicRestart.privatememory'

}

Clear-ItemProperty @IPHT2

[int32] $PrivMemMax = 150mb

Set-ItemProperty -Path 'IIS:\AppPools\WWW2Pool' `

-Name Recycling.periodicRestart.privateMemory `

-Value $PrivMemMax

1. Set the maximum number of requests before a recycle and view, as follows:

$IPHT3 = @{

Path = 'IIS:\AppPools\WWW2Pool'

Name = 'Recycling.periodicRestart.requests'

}

Clear-ItemProperty @IPHT3

[int32] $MaxRequests = 104242

Set-ItemProperty @IPHT3 -Value $MaxRequests

Get-ItemProperty @IPHT3

1. Recycle the application pool immediately:

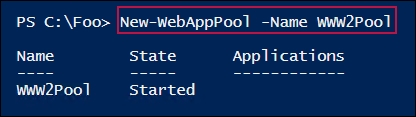
$Pool = Get-IISAppPool -Name WWW2Pool

$Pool.Recycle()

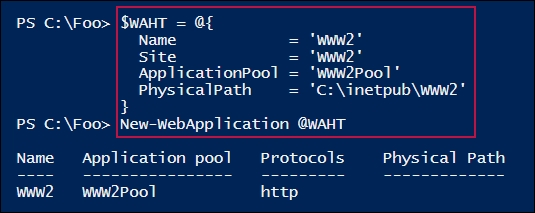
## How it works…

In step 1, you import the WebAdministration module explicitly. This loads the IIS provider, creates an IIS: PSDrive on your system, and produces no output.

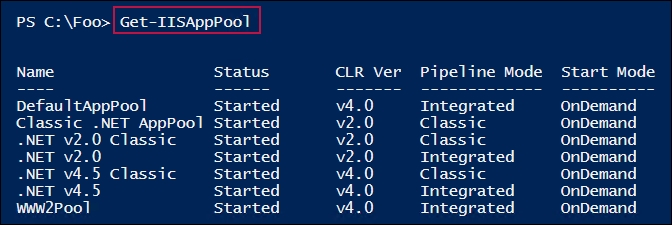
In step 2, you create a new application pool (WWW2Pool). This application pool points to the WWW2 site that you created earlier. While that site is currently just a single page, you could extend it, in which case the application would encompass all pages in the folder. This step has output that looks like this:



Once you have created the application pool, you can create a new web application to host the WWW2 site created earlier. In step 3, you create an application within the just created application pool. The output looks like this:

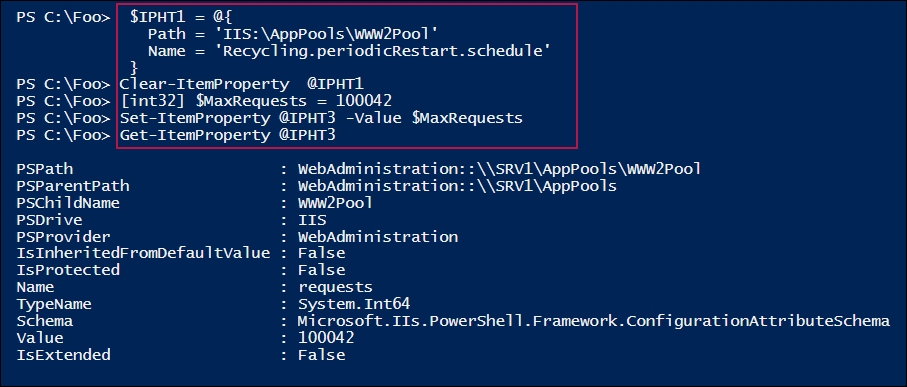


In step 4, you review the existing web application pools on SRV1 (including the WWW2Pool that you just created). The output for this looks like the following:

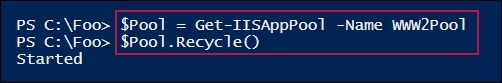


In step 5, you configure IIS to recycle the application pool at 07:55, and again at 19:55. In step 6, you configure the application to have a maximum size of 150 MB. In step 7, you specify that after a certain number of requests, IIS should automatically recycle the app pool. These three steps produce no output.

In step 7, you set the maximum requests to 100,042 to trigger an application pool recycle and view the result, which looks like this:



Finally, in step 8, you recycle the pool immediately. This produces the following output:



## There's more...

In step 2, you create a new application pool using the New-WebAppPool cmdlet. However, there is no Get-WebAppPool cmdlet to enable you to view the application pools. Instead, as you can see in step 4, you use the Get-IISAppPool cmdlet. That's because the Get-IISAppPool comes from the IISAdministration module—so much for consistency.

In step 4, you can see a variety of existing web pools. These show the IIS application pools created both by default, and by other recipes in this book. The application pool is an important feature to enable you to run multiple web applications on a single server and avoid application interference. As a part of deploying IIS, you might consider removing all but the necessary application pools.

In step 5, step 6, and step 7, you configure the application pool properties. You achieve this by setting the item properties within the IIS provider. Where you want to configure pool properties, you set the relevant item property on the application pool item for the pool.

These steps make use of the WebAdministration provider. The item properties that you set are translated by the provider into the XML that actually drives IIS. For more information on the WebAdministration provider, see <https://technet.microsoft.com/library/ee909471(v=ws.10).aspx>.

# Analyzing IIS log files

IIS logs each request that it receives from a client. If someone uses a browser to navigate to HTTP://SRV1.Reskit.Org, then details of that interaction are logged to a text file. By default, IIS stores log entries in files within the C:\inetpub\logs\LogFiles folder, but you can change the location, as you saw in the Managing IIS logging and log files recipe.

The log files that IIS generates are therefore a great source of information about who is using your web servers, and for what. Details such as the client's IP address, the HTTP verb (GET, POST, and so on), the page requested, and more, are all in the log.

In this recipe, you process the logs on SRV1 to see which clients are connecting to your server and what client software they are using.

## Getting ready

You run this recipe on/against SRV1, a web server that you have configured and used in other recipes in this chapter. In order to get useful data from this recipe, you need log files, and that means using one (and preferably more) client to access the web server on SRV1. You can use any of the other virtual machines (VMs) that you have to access the default IIS website on SRV1.

## How to do it...

1. Define the location of the log files and a temporary filename:

$LogFolder = 'C:\inetpub\logs\LogFiles\W3SVC1'

$LogFiles = Get-ChildItem $LogFolder\\*.log -Recurse

$LogTemp = "C:\inetpub\logs\LogFiles\W3SVC1\AllLogs.tmp"

1. Create a ($Logs) array to hold each useful line of each log file:

$Logs = @() # Create empty array

# Remove the comment lines

$LogFiles |

ForEach { Get-Content $\_ |

Where-Object {$\_ -notLike "#[D,F,S,V]\*" } |

Foreach { $Logs += $\_ } # add log entry to $Logs array

}

1. Build a better CSV file header:

$LogColumns = ( $LogFiles |

Select-Object -First 1 |

Foreach { Get-Content $\_ |

Where-Object {$\_ -Like "#[F]\*" } } )

$LogColumns = $LogColumns -replace "#Fields: ", ""

$LogColumns = $LogColumns -replace "-",""

$LogColumns = $LogColumns -replace "\(",""

$LogColumns = $LogColumns -replace "\)",""

1. Save the updated log entries to the temporary file:

$NL = [Environment]::NewLine

$P = [System.String]::Join( [Environment]::NewLine, $Logs)

$S = "{0}{1}{2}" -f $LogColumns, $NL,$P

Set-Content -Path $LogTemp -Value $S

1. Read the reformatted logs as a CSV file:

$Logs = Import-Csv -Path $LogTemp -Delimiter " "

1. View the client IP addresses:

$Logs |

Sort-Object -Property cip |

Select-Object -Property CIP -Unique

1. View the user agent instances used to communicate with SRV1:

$Logs |

Sort-Object -property csUserAgent |

Select-Object -Property csUserAgent -Unique

1. View the access frequency of each user agent:

$Logs |

Sort-Object -Property csUserAgent |

Group-Object csuseragent |

Sort-object -Property Count -Desc |

Format-Table -Property Count, Name

1. Who is using what:

$Logs |

Select-Object -Property CIP, CSUserAgent -Unique |

Sort-Object -Property CIP

## How it works…

In step 1, you define the location of the IIS log files that you are interested in and collect the logs for the specific website (that is, the default website on SRV1). You also create the name of a temporary file that you use to hold the log details.

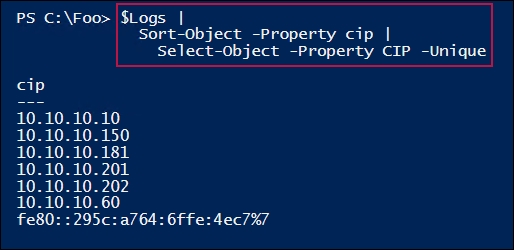
In step 2, you build an array ($Logs), which contains all of the actual log events, with the comments stripped out.

In step 3, you build a more useful header line for the temporary log file.

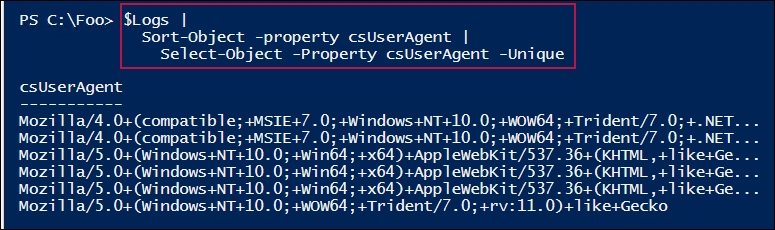
Next, in step 4, you save the actual log events, along with the updated header, into the temporary CSV file.

Then, in step 5, you read in the temporary CSV file. These first five steps produce no output.

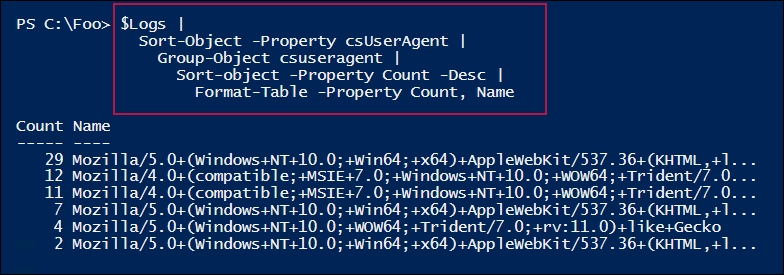
In step 6, you extract the different client IP addresses representing the individual clients that accessed SRV1, which looks like this:



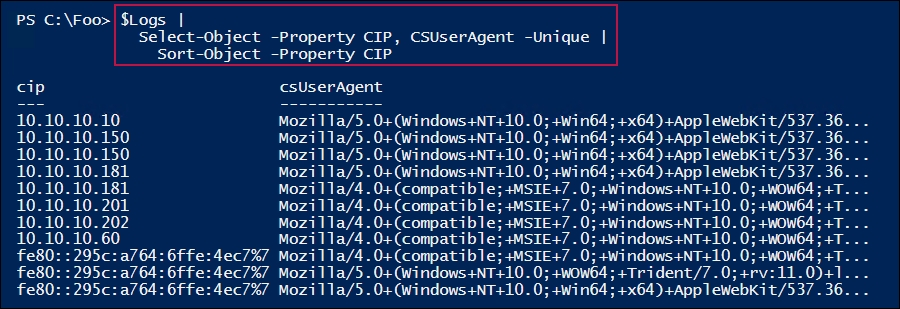
In step 7, you extract the names of the different user agents (browsers, and so on) that were used by clients to access http://SRV1. The output of this step looks like this:



In step 8, you examine the frequency of user agent usage, which looks like this:



Finally, in step 9, you look at what IP address is using which user agent, which looks like this:



## There's more...

In the Managing IIS logging and log files recipe, you might have changed the location of the IIS log files. If so, you may need to adjust the value of $LogFolder set in step 1.

In step 3, you create an updated and more useful CSV header line. This, amongst other things, gives the data columns more helpful names, which simplifies the processing you do, in step 6, step 7, and step 8. You could augment the updates to column names to be even more useful in your environment.

In the last three steps, you examine the user agent names supplied to SRV1 when the client systems connected. At connection time, each web client provides the name of the user agent, which is what IIS logs. In step 7, you see the different agent names, while in step 8, you look at how often each user agent is used. Finally, in step 9, you examine which IP address uses which user agent.

In step 8, you sort the user agent usage by IP address. Since PowerShell stores the IP address as a string, you can see that 10.10.10.202 comes before 10.10.10.60, which is, of course, how string sorting works in PowerShell. Additionally, you could extend this step to perform a DNS lookup (using Resolve-DnsName) to get the hostname and add the hostname to the output.

# Managing and monitoring Network Load Balancing

Network Load Balancing (NLB) is a feature of Windows and IIS that allows for multiple hosts to host the same website. The NLB cluster distributes all traffic to the cluster on the individual hosts.

NLB provides both scalability and fault tolerance. If you add additional nodes, the cluster is able to handle more traffic. And if a node should fail, the remaining nodes take the traffic, albeit at a potentially lower performance level.

NLB is a versatile feature. You can use NLB to load balance traffic from the web, over FTP, firewalls, proxies, and VPNs. Performance is acceptable, although many users prefer to use hardware load balancers.

In this recipe, you create a new NLB cluster (ReskitNLB) that load balances between two hosts (NLB1, NLB2). The recipe creates a simple, single-page site on each system, and then provides load balancing and failover of the NLB site.

In this recipe, you create a single-document site. The single document differs on each server, which is useful to show which server accepted and processed any given request. In production, you would want all of the nodes to have the same content, providing a seamless experience.

You run the first part of this recipe on NLB1. Once you have the NLB cluster, you can view it from another host (in this case, DC1).

## Getting ready

This recipe uses two new Windows 2019 servers: NLB1 and NLB2. You also use the server DC1, which is the domain controller in the Reskit.Org domain, and is also a DNS server for the domain. Each server is required to have static IP addresses; otherwise, you see an error when attempting to create the NLB cluster.

## How to do it...

1. Install the Web-Server (and .NET 3.5) feature on NLB1, NLB2:

$IHT1 = @{

Name = 'Web-Server'

IncludeManagementTools = $True

IncludeAllSubFeature = $True

Source = 'D:\sources\sxs'

}

Install-WindowsFeature @IHT1 -ComputerName NLB1

Install-WindowsFeature @IHT1 -ComputerName NLB2

1. Now, add the NLB feature to NLB1, NLB2:

$IHT2 = @{

Name = 'NLB'

IncludeManagementTools = $True

IncludeAllSubFeature = $True

}

Install-WindowsFeature @IHT -ComputerName NLB1 | Out-Null

Install-WindowsFeature @IHT -ComputerName NLB2 | Out-Null

1. Confirm that the NLB and Web-Server features are loaded on both NLB systems:

$SB = {

Get-WindowsFeature Web-Server, NLB

}

Invoke-Command -ScriptBlock $SB -ComputerName NLB1, NLB2 |

Format-table -Property DisplayName,PSComputername,Installstate

1. Create the NLB cluster, beginning with NLB1:

$NLBHT1 = @{

InterFaceName = 'Ethernet'

ClusterName = 'ReskitNLB'

ClusterPrimaryIP = '10.10.10.55'

SubnetMask = '255.255.255.0'

OperationMode = 'Multicast'

}

New-NlbCluster @NLBHT1

1. Add NLB2 to the ReskitNLB cluster:

$NLBHT2 = @{

NewNodeName = 'NLB2.Reskit.Org'

NewNodeInterface = 'Ethernet'

InterfaceName = 'Ethernet'

}

Add-NlbClusterNode @NLBHT2

1. Create the following network firewall rule:

$SB = {

$NFTHT =@{

DisplayGroup = 'File and Printer Sharing'

Enabled = 'True'

}

Set-NetFirewallRule @NFTHT

}

Invoke-Command -ScriptBlock $SB -ComputerName NLB1

Invoke-Command -ScriptBlock $SB -ComputerName NLB2

1. Create a default document, with different content on each machine:

'NLB Cluster: Hosted on NLB1' |

Out-File -FilePath C:\inetpub\wwwroot\Index.Html

'NLB Cluster: Greetings from NLB2' |

Out-File -FilePath \\NLB2\C$\inetpub\wwwroot\Index.Html

1. Check the VIP address for the NLB cluster:

Get-NlbClusterVip

1. Add a DNS A record for the cluster:

$SB = {

$NAHT = @{

Name = 'ReskitNLB'

IPv4Address = '10.10.10.55'

ZoneName = 'Reskit.Org'

}

Add-DnsServerResourceRecordA @NAHT

}

Invoke-Command -ComputerName DC1 -ScriptBlock $SB

# DO REMAINDER OF THIS RECIPE FROM DC1

1. View the NLB cluster node details from DC1:

Get-NlbClusterNode -HostName NLB1.Reskit.Org

1. View the NLB site from DC1, as follows:

Start-Process 'HTTP://ReskitNLB.Reskit.Org'

1. Stop one node (the one that responded in step 11):

Stop-NlbClusterNode -HostName NLB1

Clear-DnsClientCache

1. View the cluster node details on NLB1:

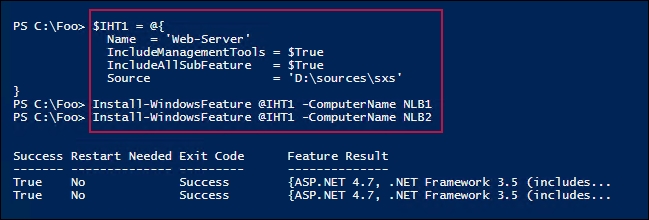
Get-NlbClusterNode -HostName NLB1

1. View the site again (from DC1):

Start-Process 'HTTP://ReskitNLB.Reskit.Org'

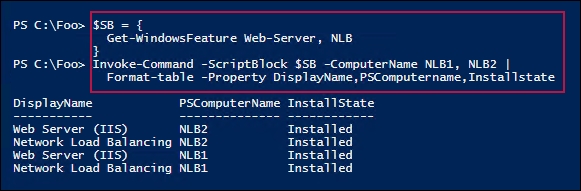
## How it works…

In step 1, you add the web service, tools, and sub-features to NLB1 and NLB2, which looks like this:

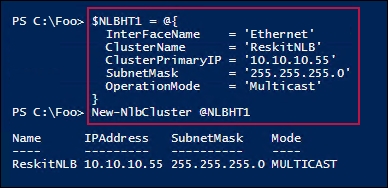


In step 2, you also install the web server feature on NLB1 and NLB2, which produces no output.

In step 3, you check to see whether IIS and NLB are loaded on both NLB1 and NLB2, which looks like this:

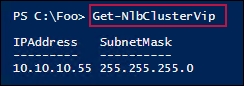


In step 4, you configure NLB to run on NLB1, which looks like this:



In step 5, you add NLB2.Reskit.Org to the ReskitNLB cluster. In step 6, you create firewall rules for the cluster. In step 7, you create the contents of the default document for both NLB1 and NLB2. These three steps produce no output.

In step 8, with the configuration of the NLB cluster, you retrieve the cluster's VIP address, which looks like this:



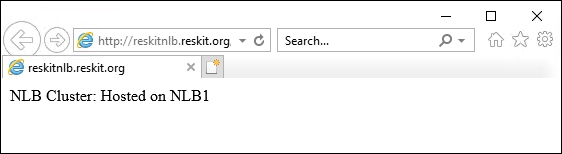
In step 9, you complete the configuration of the NLB cluster on NLB1 by creating an A record for the cluster name, which produces no output.

Once these steps have been completed, you have set up the NLB cluster. To test it, run the remaining steps in this recipe on DC1.

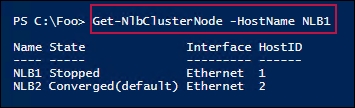
In step 10, you view the cluster node status, as follows:



In step 11, you view the NLB site from DC1, which looks like this:



In step 12, you stop NLB on NLB1, which produces no output. In step 13, you get the NLB cluster node details for the NLB cluster, showing that NLB1 is down, which looks like this:



After shutting down NLB, you re-browse on DC1 to the cluster, in order to see the following:



## There's more...

This recipe uses two new servers (NLB1 and NLB2). You could also run this recipe on other servers, as appropriate; for example, the SRV1 and SRV2 servers that are used elsewhere in this book.

In step 3, you create the NLB cluster. Because NLB1 and NLB2 have just one network adapter, you create the cluster with an operation mode of multicast. Had you used unicast, Windows would have effectively killed off the normal connection to these systems. In production, you would probably want two NICs inside each NLB cluster member.

In step 12, you stop a node in the ReskitNLB load balancing cluster. You then view (in step 13) the status of the nodes in the cluster by using the Get-NlbClusterNode cmdlet. After stopping NLB1, you can review the cluster to see the document from NLB2 (in step 14).

In step 7, you create the contents of the default document for both NLB1 and NLB2. In this recipe, you deliberately create different content for each document on both NLB cluster members. This shows you which host is handling the request for the default document on the NLB cluster.

If you need to take a node down (for example, to install a patch), you might want to do this during a maintenance window, when no one would be using the website provided by the cluster. If necessary, you can take one node down, perform any maintenance, and then restore the cluster member. All the while, the cluster continues to work. While a node is not part of the cluster, the cluster is less performant, but this is a fact to consider when deciding to maintain a live cluster (that is, during the day) or by waiting until your next setting any maintenance window.

In step 11, you view the site via the NLB cluster (from DC1), which shows that NLB1 is supplying the page. If you are testing this, you may find that NLB2 responds to the initial request (step 11); if so, in step 12, shut down NLB2 instead.

Note

If you run these tests on either of the cluster members, NLB resolves the cluster to the local site. Thus, running this from NLB1 would always pick NLB1, whereas running it from another host, such as DC1, you would see the desired behavior.