2

Managing Windows Network Services with PowerShell

In this chapter we will cover the following recipes:

- Configuring static networking
- Installing domain controllers
- Configuring zones in DNS
- Configuring DHCP scopes
- Configuring DHCP server failover
- Converting DHCP addresses to static
- Building out a PKI environment
- Creating AD users
- Searching for and reporting on AD users
- Finding expired computers in AD
- Creating and e-mailing a superuser report

Introduction

Setting up a new Active Directory environment can be either exciting or boring. If you have rarely built out new domain and networking environments, the process is probably new and very exciting. However, if you are constantly building out new environments for test labs or other business needs, the process can be fairly long and drawn out. Instead, you are mostly interested in automating the process to require minimal user input and maintain consistency between builds.

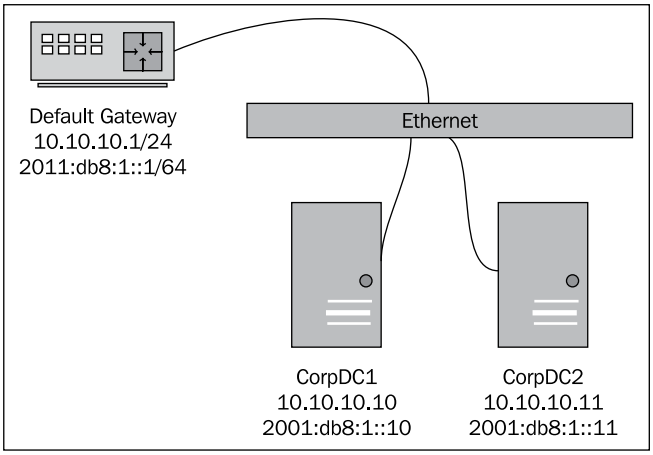
This chapter covers the installation and configuration of Active Directory, DNS, DHCP, and Certificate Services. This chapter should cover everything necessary to prepare an environment as a fully functioning Active Directory domain for use in labs or new domain environments.

Configuring static networking

TCP/IP is the primary technology used for communicating between computers today. When first building out an environment, one of the first items to accomplish is to define and apply an IP addressing scheme. Once the addressing scheme is defined, we can create static addresses for our first servers. Later, we will configure DHCP in case static addressing is not desired for all of the systems in your environment.

Getting ready

From the following diagram we can see that we have already defined our addressing scheme using both IPv4 and IPv6. At the start of our network, we have a router acting as a default gateway, and we will configure two servers in preparation for becoming domain controllers. The default gateway router is already statically assigned with IPv4 and IPv6 addresses:



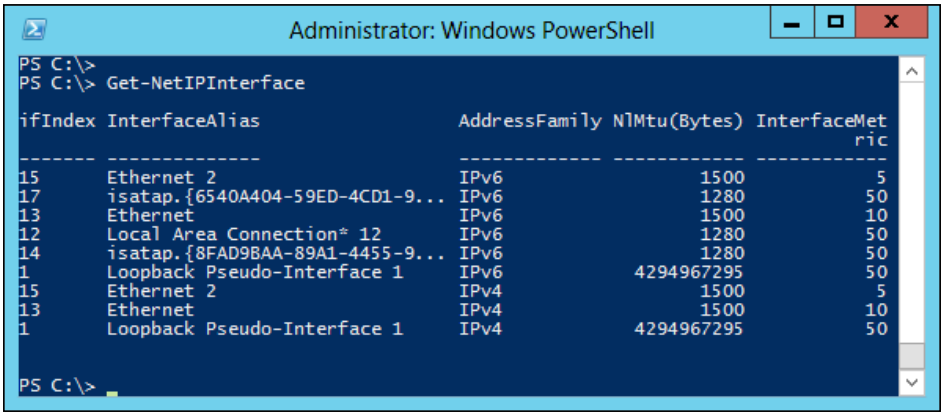
All three of these components are connected to a common Ethernet segment to communicate with each other.

Before defining any networking configuration, we should confirm that our addresses do not conflict with other networks in our environment. Even when building out isolated environments, it is best to use different network addresses in case of accidental conflict with production environments.

How to do it...

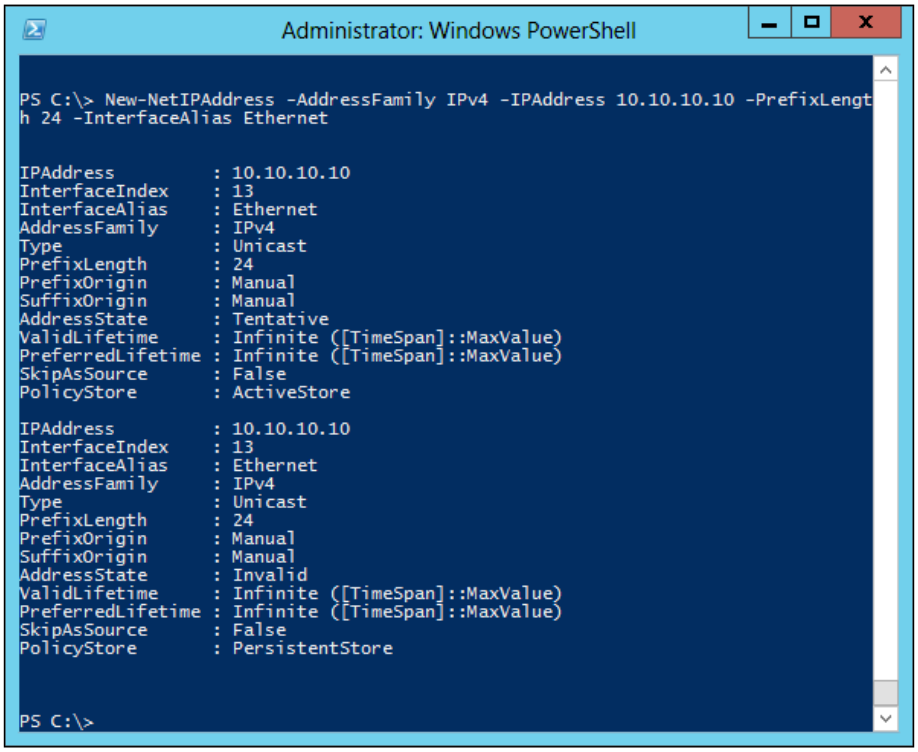
Carry out the following steps to configure static networking:

1. Find the interface to set by executing `Get-NetIPInterface`:



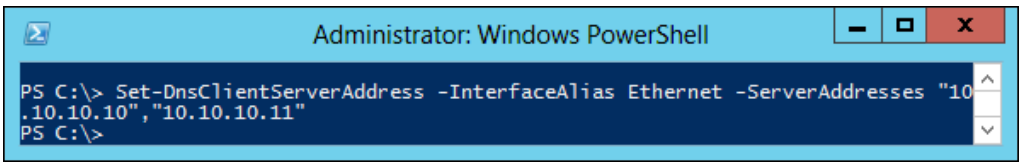
2. Set the IP information using `New-NetIPAddress`:

```
New-NetIPAddress -AddressFamily IPv4 -IPAddress 10.10.10.10
-PrefixLength 24 -InterfaceAlias Ethernet
```



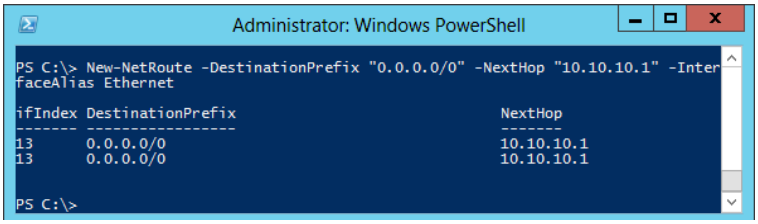
3. Set DNS Servers using `Set-DnsClientServerAddress`:

```
Set-DnsClientServerAddress -InterfaceAlias Ethernet
-ServerAddresses "10.10.10.10","10.10.10.11"
```



4. Set the default route using `New-NetRoute`:

```
New-NetRoute -DestinationPrefix "0.0.0.0/0" -NextHop "10.10.10.1"
-InterfaceAlias Ethernet
```



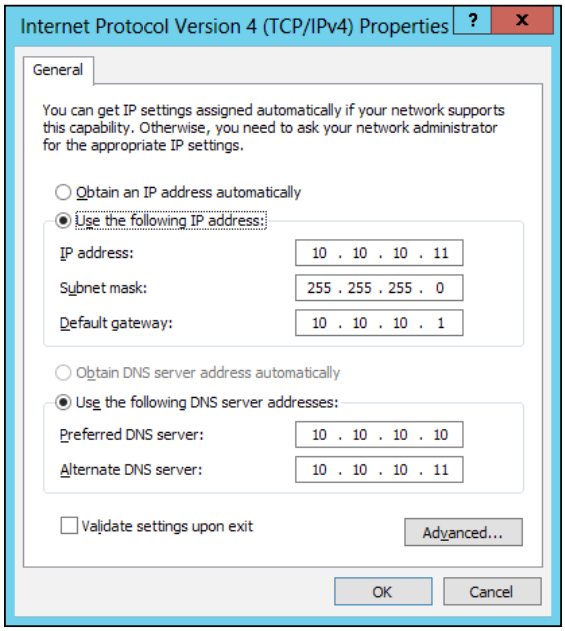
How it works...

In the first step we list out the network adapters available on the server. Windows Servers often include several network adapters of different types, and depending on the features installed, there can be several more. By executing `Get-NetworkIPInterface`, we list the interface names and indexes that we will use to identify the specific interface we desire to configure.

The second and third steps use `New-NetIPAddress` and `Set-DnsClientServerAddress` to configure the identified interface with IPv4 address and DNS targets for the specified interface.

The last step uses `New-NetRoute` to define a network route. The `-DestinationPrefix 0.0.0.0/0` parameter identifies this route as the default route, or default gateway. The `-NextHop 10.10.10.1` parameter is the router address to forward traffic into if another route does not take precedence.

The following screenshot shows the IPv4 address properties after finalizing configuration via PowerShell:



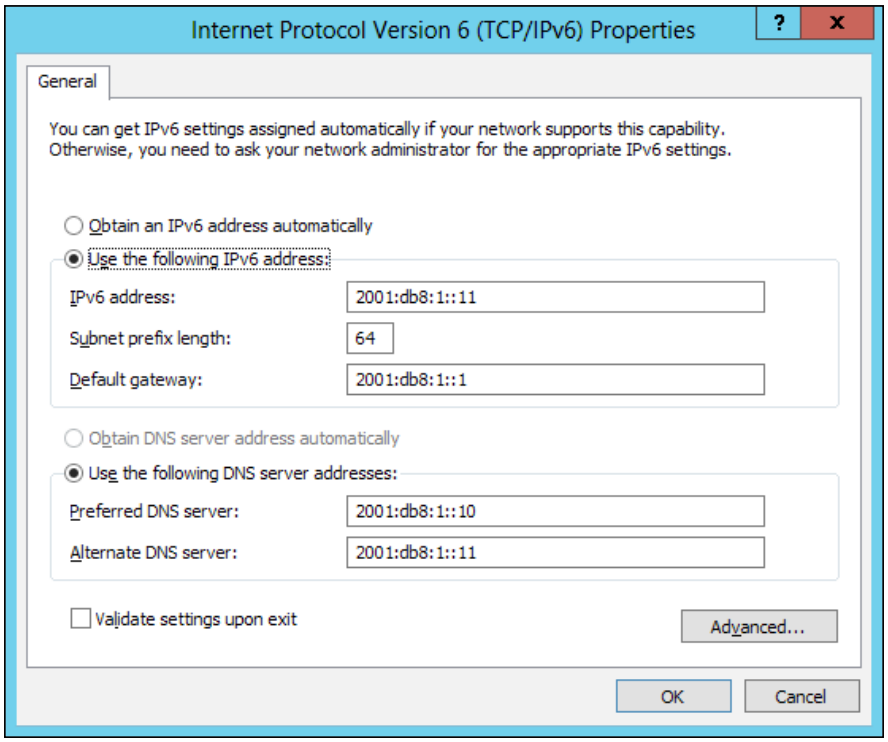
There's more...

There are a few more features provided by PowerShell. They are as follows:

- **IPv6 addressing**: In addition to configuring IPv4, PowerShell can also configure IPv6 addresses. The process for configuring static IPv6 addressing is exactly the same as IPv4, the only change is the addresses themselves.

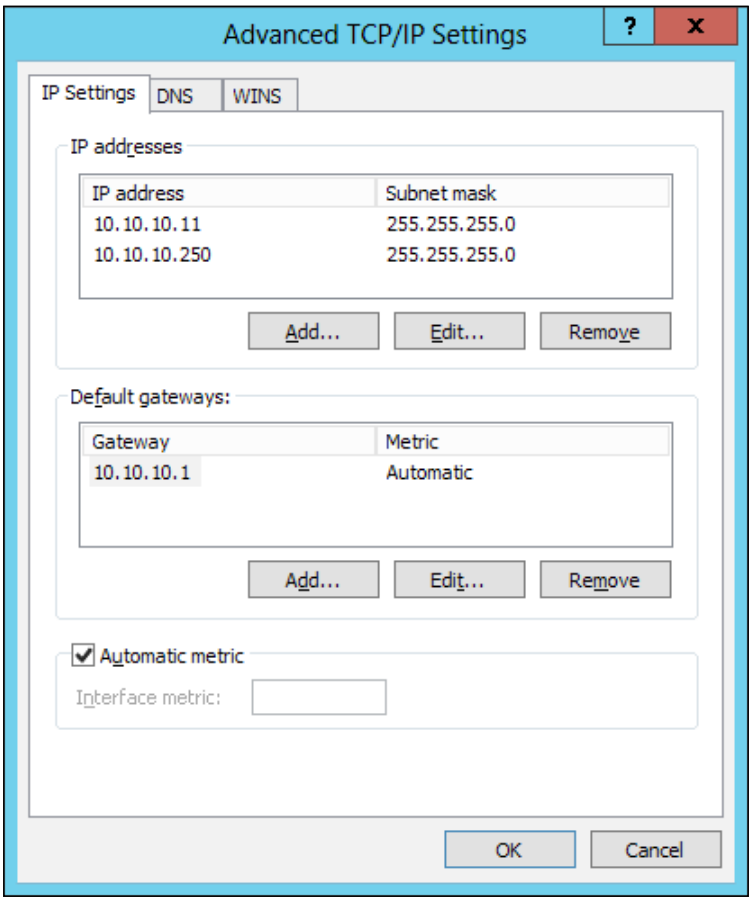
  Following are examples of configuring IPv6 on the same host. Note that both IPv4 and IPv6 addressing can coexist on the same server without issue:

```
New-NetIPAddress -AddressFamily IPv6 -IPAddress 2001:db8:1::10 `
-PrefixLength 64 -InterfaceAlias Ethernet
New-NetRoute -DestinationPrefix ::/0 -NextHop 2001:db8:1::1 `
-InterfaceAlias Ethernet
Set-DnsClientServerAddress -InterfaceAlias Ethernet `
-ServerAddresses "2001:db8:1::10","2001:db8:1::11"
```

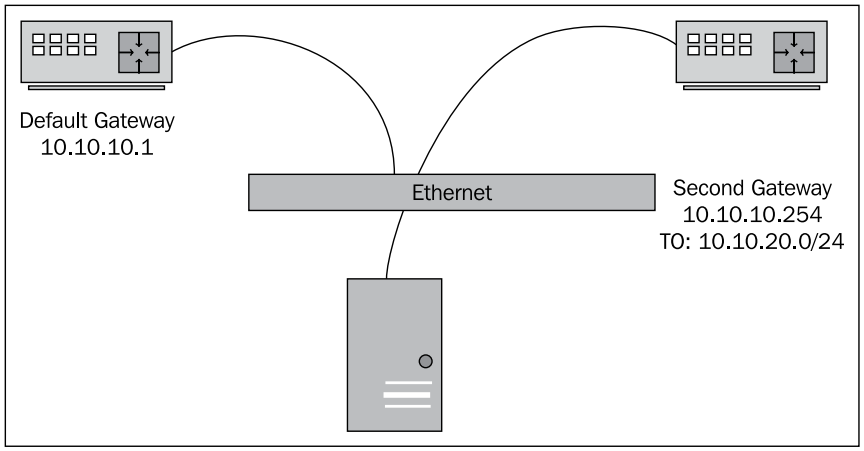


- **Additional IP addresses**: By using the `New-NetIPAddress` function, an interface can be configured with multiple IP addresses simultaneously. This configuration is often used for clustering or load balancing within Windows. Following is an example of configuring an additional address:

```
New-NetIPAddress -AddressFamily IPv4 -IPAddress 10.10.10.250
-PrefixLength 24 -InterfaceAlias Ethernet
```



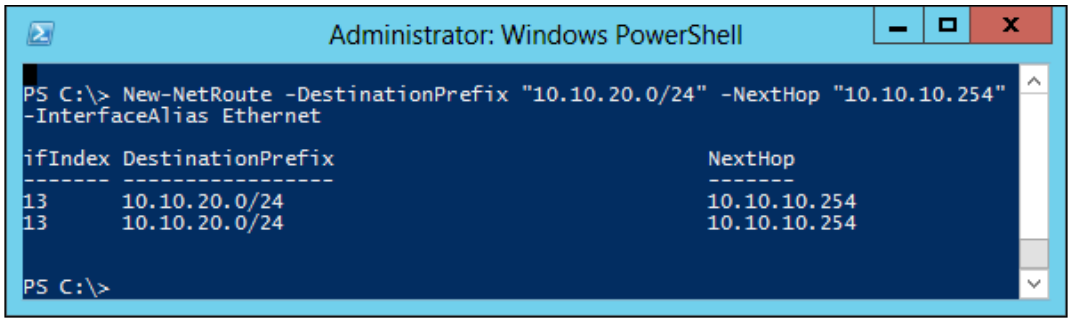
- **Additional routes**: Windows has the ability to route network packets to more locations than the default gateway. Say for instance, there are two routers on your network: the default gateway and a second gateway. The second gateway is used to access the `10.10.20.0/24` network, and the Windows server needs to be configured to route to it:



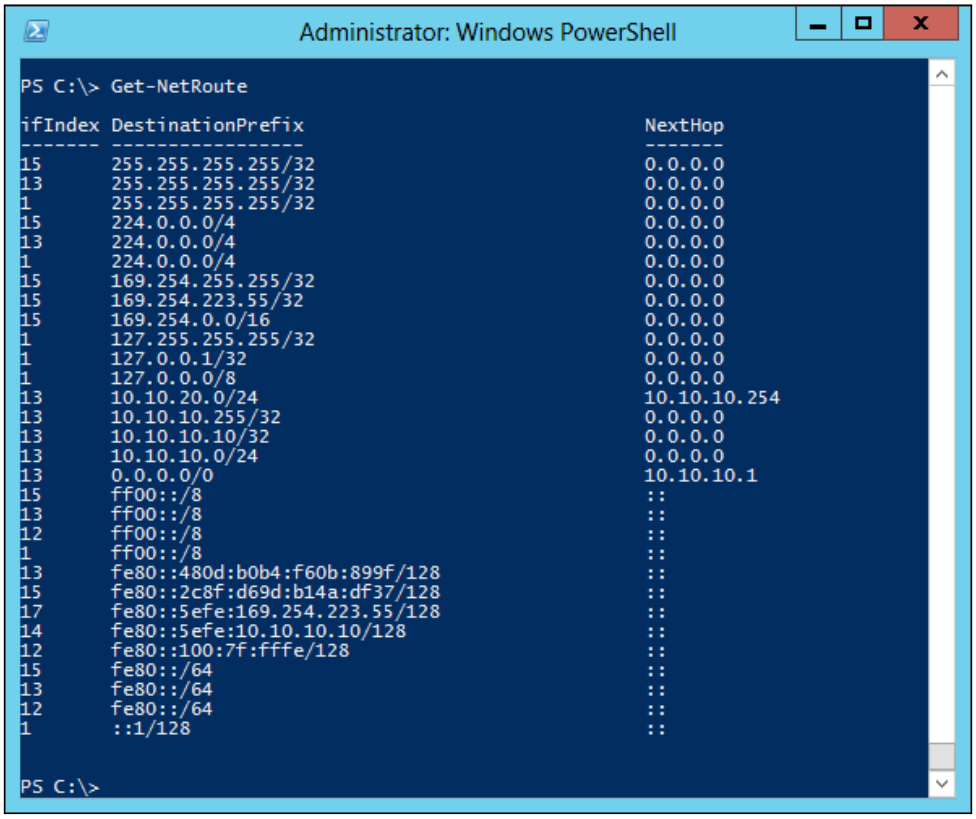
By executing the `New-NetRoute` command again, with the `-DestinationPrefix` and `-NextHop` addresses changed appropriately, we add a specific route to the server:

```
New-NetRoute -DestinationPrefix "10.10.20.0/24" -NextHop
"10.10.10.254" -InterfaceAlias Ethernet
```

In some cases, such as a dedicated management network, the secondary network may be connected to a different network interface. If that is the situation, change the `–InterfaceAlias` parameter to target the second interface.



The full list of routes can be viewed by running `Get-NetRoute`. This will return all IPv4, IPv6, default, and static routes that are defined on the system:



Installing domain controllers

Once the TCP/IP networking is set up and working, the next step to tackle is installing the domain controllers. In a Windows Active Directory domain, the domain controllers can be viewed as the core of the network. Domain controllers provide user authentication, group policy information, time synchronization, and access to Active Directory objects. Additionally, domain controllers often provide several network services such as DNS, DHCP, certificate services, and more.

This recipe will set up and install the first domain controller, creating a new domain in a new forest. Once completed, the second domain controller will be remotely installed and promoted. Additionally, we will install DNS on both domain controllers to provide name resolution services.

Getting ready

This recipe assumes a server and networking configuration setup similar to the prior recipe. We will be working with newly installed servers without any additional roles or software installed. To complete these tasks, you will need to log on to the server as the local administrator.

How to do it...

Carry out the following steps to install the domain controller:

1. As an administrator, open a PowerShell.
2. Identify the Windows Features to install:

```
Get-WindowsFeature | Where-Object Name -like *domain*
Get-WindowsFeature | Where-Object Name -like *dns*
```

3. Install the necessary features:

```
Install-WindowsFeature AD-Domain-Services, DNS –
IncludeManagementTools
```

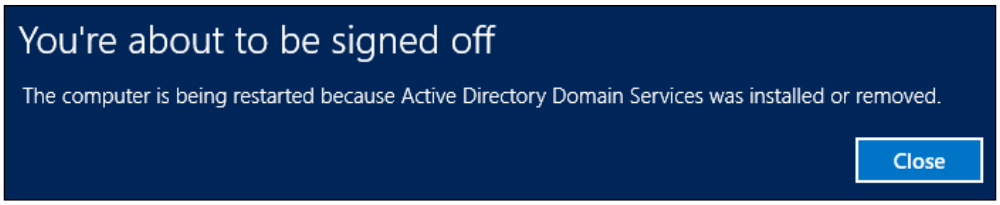
4. Configure the domain:

```
$SMPass = ConvertTo-SecureString 'P@$$w0rd11' –AsPlainText -Force
Install-ADDSForest -DomainName corp.contoso.com –
SafeModeAdministratorPassword $SMPass –Confirm:$false
```

How it works...

The first step executes the `Get-WindowsFeature` Cmdlet to list the features necessary to install domain services and DNS. If you are unsure of the exact names of the features to install, this is a great method to search for the feature names using wildcards. The second step uses `Install-WindowsFeature` to install the identified features, any dependencies, and any applicable management tools.

The third step calls `Install-ADDSForest` to create a new domain/forest named `corp.contoso.com`. Before promoting the server to a domain controller, we create a variable named `$SMPass`, which will hold a secure string that can be used as a password when promoting the server. This secure string is then passed as `-SafeModeAdministratorPassword` to the server, allowing access to the server if the domain services fail to start in the future:



You will see a notice similar to the preceding screenshot when installation is finished. The system will automatically restart and the domain controller install will be complete.

There's more...

The following lists what more can be done with the domain controller:

- **Joining a computer to domain**: Once the domain has been created, computers can be joined to the domain manually or via automation. The following example shows how to use PowerShell to join the `CorpDC2` computer to the `corp.contoso.com` domain.

```
$secString = ConvertTo-SecureString 'P@$$w0rd11' -AsPlainText
-Force
$myCred = New-Object -TypeName PSCredential -ArgumentList "corp\
administrator", $secString
Add-Computer -DomainName "corp.contoso.com" -Credential $myCred –
NewName "CORPDC2" –Restart
```

  Similar to creating the domain, first a `$secString` variable is created to hold a secure copy of the password that will be used to join the computer to the domain. Then a `$myCred` variable is created to convert the username/password combination into a `PSCrededntial` object that will be used to join the computer to the domain. Lastly, the `Add-Computer` Cmdlet is called to join the computer to the domain and simultaneously, rename the system. When the system reboots, it will be connected to the domain.

- **Push install of domain controller**: It is normally considered best practice to have at least two **domain controllers** (**DCs**) for each domain. By having two DCs, one can be taken offline for maintenance, patching, or as the result of an unplanned outage, without impacting the overall domain services.

Once a computer has been joined to the domain, promoting the system to a DC can be performed remotely using PowerShell:

```
Install-WindowsFeature –Name AD-Domain-Services, DNS
-IncludeManagementTools –ComputerName CORPDC2
Invoke-Command –ComputerName CORPDC2 –ScriptBlock {
$secPass = ConvertTo-SecureString 'P@$$w0rd11' -AsPlainText –Force
$myCred = New-Object -TypeName PSCredential -ArgumentList "corp\
administrator", $secPass
$SMPass = ConvertTo-SecureString 'P@$$w0rd11' –AsPlainText –Force
Install-ADDSDomainController -DomainName corp.contoso.com –
SafeModeAdministratorPassword $SMPass -Credential $myCred –
Confirm:$false
}
```

First, the Domain and DNS services and appropriate management tools are installed on the remote computer. Then, using the `Invoke-Command` Cmdlet, the commands are executed remotely to promote the server to a domain controller and reboot.

To create a new domain/forest, we used the `Install-ADDSForest` command. To promote a computer into an existing domain/forest, we use the `Install-ADDSDomainController` command.

Configuring zones in DNS

Windows domains rely heavily on DNS for name resolution and for finding appropriate resources. DNS is composed primarily of zones, each of which contains records. These zones and records provide name to address and address to name resolution for clients.

Here we will install and configure the DNS service and configure zones for servicing clients.

Getting ready

This recipe assumes a server and networking configuration similar to what is created in the first recipe. For DNS services to operate, the server does not need to be a member of an Active Directory domain, and in some scenarios, such as internet facing systems, Active Directory membership is discouraged.

We will be configuring our DNS servers with the following zones:

Zone	Type
corp.contoso.com	AD integrated
10.10.10.in-addr.arpa	AD integrated reverse lookup
20.168.192.in-add.arpa	AD integrated reverse lookup
contoso.com	Standard primary
fabrkam.com	Conditional forwarder to 192.168.99.1
corp.adatum.com	Secondary zone referencing 192.168.1.1

How to do it...

Carry out the following steps to configure zones in DNS:

1. Identify features to install:

```
Get-WindowsFeature | Where-Object Name -like *dns*
```

2. Install DNS feature and tools (if not already installed):

```
Install-WindowsFeature DNS -IncludeManagementTools –
IncludeAllSubFeature
```

3. Create a reverse lookup zone:

```
Add-DnsServerPrimaryZone –Name 10.10.10.in-addr.arpa –
ReplicationScope Forest
Add-DnsServerPrimaryZone –Name 20.168.192.in-addr.arpa –
ReplicationScope Forest
```

4. Create a primary zone and add static records:

```
Add-DnsServerPrimaryZone –Name contoso.com –ZoneFile contoso.com.
dns
Add-DnsServerResourceRecordA –ZoneName contoso.com –Name www –
IPv4Address 192.168.20.54 –CreatePtr
```

5. Create a conditional forwarder:

```
Add-DnsServerConditionalForwarderZone -Name fabrikam.com
-MasterServers 192.168.99.1
```

6. Create a secondary zone:

```
Add-DnsServerSecondaryZone -Name corp.adatum.com -ZoneFile corp.
adatum.com.dns -MasterServers 192.168.1.1
```

How it works...

The first two steps may have already been completed if your DNS server coexists on the domain controller. When viewing the output of `Get-WindowsFeature` in the first step, if `Install State` for the DNS features equals `Installed`, the roles are already installed. If the roles are already installed, you can still attempt to reinstall them without causing issues.

The third step creates two AD-integrated reverse lookup zones named `10.10.10.in-addr.arpa` and `20.168.192.in-addr.arpa`. These zones are used for IP-to-Name resolution for servers in the `10.10.10.0/24` (internal) and `192.168.20.0/24` (DMZ or untrusted) subnets. These reverse lookup zones are not automatically created when installing DNS or Active Directory and it is the administrator's responsibility to create it.

It is considered a best practice to have a reverse lookup zone for all networks in your organization. This eases many operational tasks and some network tools fail to work properly if the reverse lookup zones don't exist.

The fourth step creates a standard primary zone named `contoso.com`. This zone is different from the `corp.contoso.com` zone that was automatically created during creation of the domain. This new zone will be used to host records used in an untrusted or DMZ environment. In this example we created a static record `www.contoso.com`, configured it with a target IP address, and configured the reverse lookup record as well.

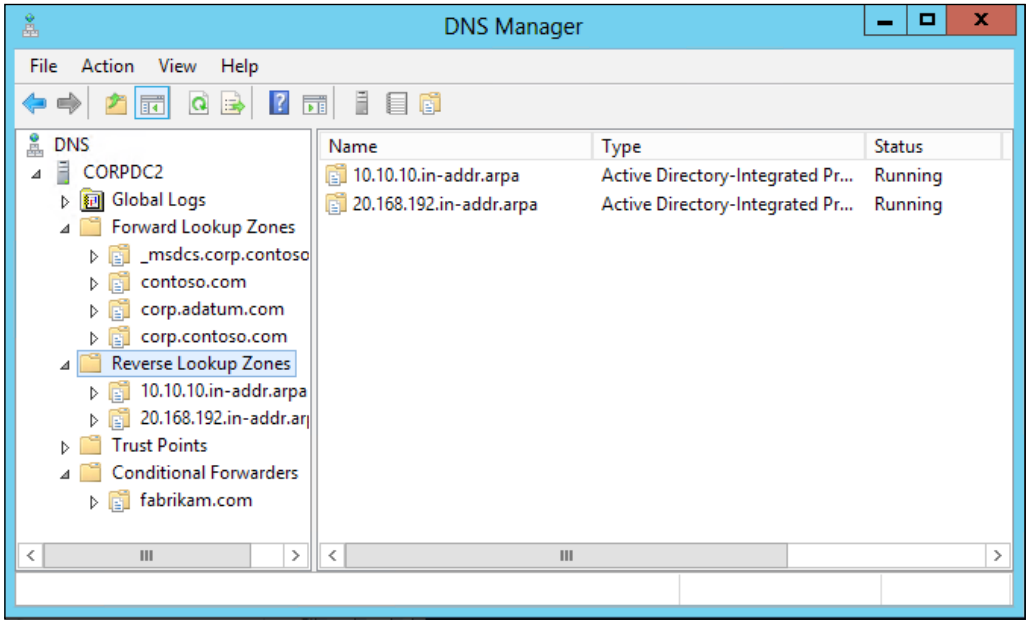
The steps shown here are an example of creating a primary zone. Additional steps may be needed to fully secure a DNS server that is accessible by the outside world.

Additionally, standard primary zones cannot be AD-integrated and do not automatically replicate to other DNS servers. To replicate a standard primary zone, a secondary zone must be created on the target DNS server and authorized to replicate.

The fifth step creates a conditional forwarder named `fabrikam.com`. A conditional forwarder simply identifies the domain request and forwards it to the appropriate master servers.

The sixth step creates a secondary zone named `corp.adatum.com`. Unlike primary zones, secondary zones are read-only, and they only hold a copy of the zone data as pulled from the master server. To add or update records in this zone, the changes must be made at the master server, and then replicated to the secondary.

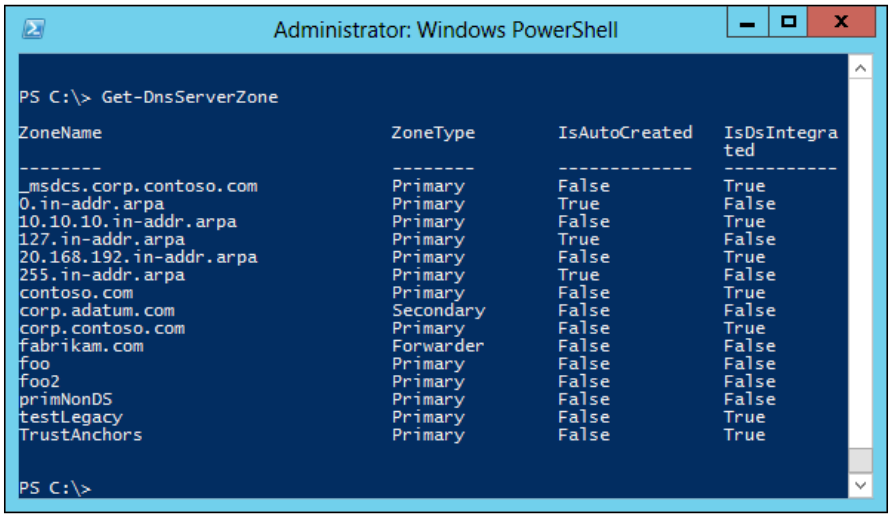
Unlike primary zones and conditional forwarders, secondary zones cannot be AD-integrated and do not automatically replicate to other DNS servers in the domain. This means that the secondary zones must be configured on each DNS server that will host the zone.



There's more...

The following lists the additional features of zones in DNS:

- **Listing all zones**: A full list of DNS zones on a server can be returned by executing the `Get-DnsServerZone` function:



- **Updating DNS records**: When updating static records there are two options: delete and recreate, and update. The following is a simple function that gets a current resource record from DNS, updates it, and commits it back to DNS:

```
Function Update-DNSServerResourceRecord{
    param(
    [string]$zoneName = $(throw "DNS zone name required")
    ,[string]$recordName = $(throw "DNS record name required")
    ,[string]$newIPv4Address = $(throw "New IPv4Address required")
    )
    # Get the current record from DNS
    $oldRecord = Get-DnsServerResourceRecord -ZoneName $zoneName
-Name $recordName
    Write-Host "Original Value: " $oldRecord.RecordData.
IPv4Address

    # Clone the record and update the new IP address
    $newRecord=$oldRecord.Clone()
    $newRecord.RecordData.IPv4Address = [ipaddress]$newIPv4Address
```

```
    # Commit the changed record
    Set-DnsServerResourceRecord -ZoneName $zoneName
-OldInputObject $oldRecord -NewInputObject $newRecord
    Write-Host "New Value: " (Get-DnsServerResourceRecord
-ZoneName $zoneName -Name $recordName).RecordData.IPv4Address
}
```

Configuring DHCP scopes

As an alternative to statically assigned TCP/IP addresses, Windows supports the **Dynamic Host Configuration Protocol** (**DHCP**). This service allows for provisioning of IP addresses, default gateways, DNS information, and even more advanced information such as boot servers.

This recipe will set up the basic DHCP features on a domain controller and configure an initial DHCP scope.

Getting ready

This recipe assumes a server, networking, and domain configuration similar to what is created in the *Installing domain controllers* recipe.

How to do it...

Carry out the following steps to configure DHCP scopes:

1. Install DHCP and management tools:

```
Get-WindowsFeature | Where-Object Name -like *dhcp*
Install-WindowsFeature DHCP -IncludeManagementTools
```

2. Create a DHCP scope

```
Add-DhcpServerv4Scope -Name "Corpnet" -StartRange 10.10.10.100
-EndRange 10.10.10.200 -SubnetMask 255.255.255.0
```

3. Set DHCP options

```
Set-DhcpServerv4OptionValue -DnsDomain corp.contoso.com -DnsServer
10.10.10.10 -Router 10.10.10.1
```

4. Activate DHCP

```
Add-DhcpServerInDC -DnsName corpdc1.corp.contoso.com
```

How it works...

The first step uses `Install-WindowsFeature` to install the DHCP feature and management tools on the currently logged on system. Once installed, the second step calls `Add-DHCPServerv4Scope` to create a DHCP scope named `Corpnet`, providing dynamic IPs on the `10.10.10.0/24 subnet`.

The third step uses `Set-DhcpServerv4OptionValue` to set up common DHCP options, such as the DNS servers and default gateway address. This command can include other common options such as the DNS domain name, WinsServer, and Wpad location. Additionally, any extended DHCP option ID can be configured using the `Set-DhcpServerv4OptionValue` command.

The last step calls `Add-DHCPServerInDC` to activate the DHCP service on the computer in Active Directory. This authorizes the DHCP service to provide addresses to clients in the domain.

There's more...

The following lists the additional features of DHCP:

- **Adding DHCP reservations**: In addition to creating and activating DHCP scopes, we can also create reservations in DHCP. A reservation matches a network adapter's MAC address to a specific IP address. It is similar to using a static address, except the static mapping is maintained on the DHCP server:

```
Add-dhcpserverv4reservation –scopeid 10.10.10.0 –ipaddress
10.10.10.102 –name test2 –description "Test server" –clientid 12-
34-56-78-90-12
Get-dhcpserverv4reservation –scopeid 10.10.10.0
```

- **Adding DHCP exclusions**: Additionally, we can create DHCP exclusions using PowerShell. An exclusion is an address, or range of addresses that the DHCP server won't provide to clients. Exclusions are often used when individual IP addresses within the scope have been statically assigned:

```
Add-DhcpServerv4ExclusionRange –ScopeId 10.10.10.0 –StartRange
10.10.10.110 –EndRange 10.10.10.111
Get-DhcpServerv4ExclusionRange
```

Configuring DHCP server failover

Prior to Server 2012, there were limited methods of ensuring DHCP was redundant and always available to service requests. One of the most common methods was to split DHCP scopes between multiple servers, with each server providing a subset of the scope. If one system was unavailable, the other system was still able to provide a subset of addresses. However, this caused problems because if a DHCP server was unavailable, there may not be enough addresses available to service all of your clients. Other redundancy options involved clustering or other expensive technologies that were difficult to manage.

In Server 2012 DHCP server failover is a built-in feature. This feature allows servers to share a common DHCP database to provide leases and provide redundancy. To use DHCP failover, the DHCP feature just needs to be installed and configured across servers. This recipe will walk through the configuration of DHCP failover.

Getting ready

This recipe assumes a server, networking, and domain configuration similar to what is created in the *Installing domain controllers* recipe. A minimum of two servers will be needed to configure as DHCP servers. Additionally, it assumes one of the domain controllers already has DHCP installed and configured.

How to do it...

Carry out the following steps to configure DHCP server failover:

1. Install DHCP on the second server either locally or remotely:

```
Install-WindowsFeature dhcp -IncludeAllSubFeature -ComputerName
corpdc2
```

2. Authorize DHCP on the second server:

```
Add-DhcpServerInDC -DnsName corpdc2.corp.contoso.com
```

3. Configure DHCP failover:

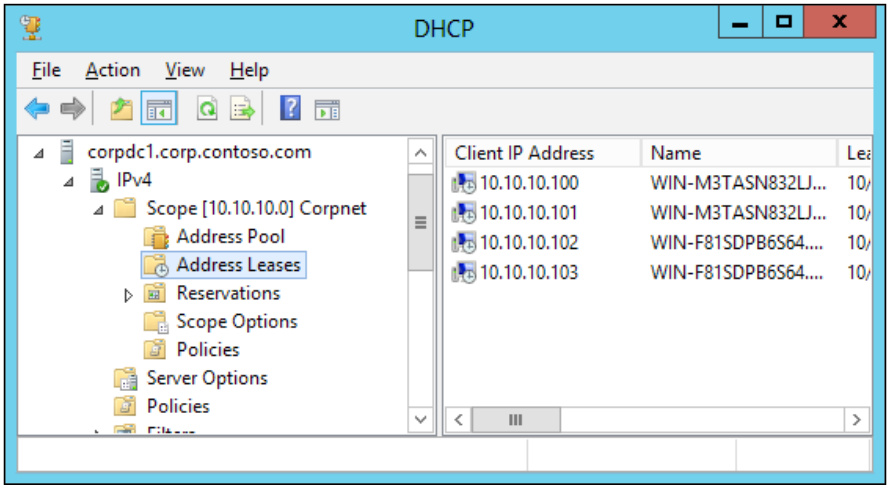
```
Add-DhcpServerv4Failover -ComputerName corpdc1 -PartnerServer
corpdc2 -Name Corpnet-Failover -ScopeId 10.10.10.0 -SharedSecret
'Pa$$w0rd!!'
```

How it works...

The first and second steps are responsible for installing and authorizing DHCP on `CorpDC2`. This is the same process used in the previous recipe to install DHCP on the first domain controller. Once installed, we use `Add-DhcpServerInDC` to authorize the server to act as a DHCP server.

The third step calls `Add-DHCPServerv4Failover` to configure DHCP failover across `CorpDC1` and `CorpDC2`. This command identifies the scope `10.10.10.0` for failover and configures a shared key for authenticating communication between the servers.

At this point the failover configuration is complete and both DHCP servers will begin providing addresses. If you open the DHCP administration console, you will see that both domain controllers have DHCP installed and servicing clients. Additionally, you will see that both servers have the same client lease information, making the solution truly redundant:



Converting DHCP addresses to static

While DHCP is an easy way to manage network addresses, especially, in dynamic environments, it does have its drawbacks. If something happens on your physical network or to your DHCP server, clients may not be able to receive or renew their addresses. And due to the dynamic nature of DHCP, addresses may change, causing issues with firewalls and DNS records.

This is normally fine for desktop environments, but in server environments, we want to minimize any possibility for an outage. As such, at some point you may want to convert your dynamically addressed hosts to use static addresses.

Getting ready

This recipe assumes a basic server configuration with a single interface using a single IP address via DHCP. The script works best when run locally on the target server.

How to do it...

Log on to the target server interactively and execute the following script:

```
# Identify all adapters that recieved an address via DHCP
$adapters = Get-WmiObject -Class Win32_NetworkAdapterConfiguration |
Where-Object {($_.IPAddress) -and $_.DHCPEnabled -eq 'True' }

# Iterate through each adapter
foreach($adapter in $adapters)
{
    # Get current adapter and IP information
    $adapIndex = $adapter.InterfaceIndex
    $ipAddress = $adapter.IPAddress[0]
    $subnetMask = $adapter.IPSubnet[0]
    $defaultGateway = $adapter.DefaultIPGateway[0]
    $prefix = (Get-NetIPAddress -InterfaceIndex $adapIndex –
AddressFamily IPv4).PrefixLength
    $dnsServers = $adapter.DNSServerSearchOrder
    [ipaddress]$netAddr = ([ipaddress]$ipAddress).Address -band
([ipaddress]$subnetMask).Address

    # Identify the DHCP server
    $dhcpServer = $adapter.DHCPServer
    $dhcpName = ([System.Net.DNS]::GetHostEntry($dhcpServer)).HostName

    # Add an exclusion to DHCP for the current IP address
    Invoke-Command -ComputerName $dhcpName -ScriptBlock{
        Add-DhcpServerv4ExclusionRange –ScopeId $args[0] –StartRange
$args[1] –EndRange $args[1]
    } -ArgumentList $netAddr.IPAddressToString, $ipAddress

    # Release the DHCP address lease
    Remove-NetIPAddress -InterfaceIndex $adapIndex -Confirm:$false

    # Statically assign the IP and DNS information
    New-NetIPAddress -InterfaceIndex $adapIndex -AddressFamily
IPv4 -IPAddress $ipAddress -PrefixLength $prefix -DefaultGateway
$defaultGateway
    Set-DnsClientServerAddress -InterfaceIndex $adapIndex
-ServerAddresses $dnsServers
}
```

How it works...

The first part of the script queries WMI for all network adapters that both have an active IP address, and are using DHCP. The results from the WMI query are placed into a variable named `$adapters` and are iterated in a `for each` loop, where the adapter and IP information is collected.

A network adapter can hold multiple IP addresses, but this script is only capable of handling the first IPv4 address of each adapter.

Once all of the network information is collected, `Invoke-Command` is used to connect to the DHCP server that issued the address and creates an exclusion. The exclusion record's start and end address is the IP address assigned to the client. This prevents the IP address from being reused by another host at a later time.

Lastly, the adapter is changed to a static address. `Remove-NetIPAddress` is used to release the DHCP address from the interface. Once cleared, `New-NetIPAddress` is used to statically configure the interface with the same IPv4 address, subnet, and gateway that was previously held. Finally, `Set-DnsClientServerAddress` assigns the DNS server addresses.

There's more...

This script can be run against a system remotely using a `PSSession`, with the exception of creating the DHCP exclusion. When using a `PSSession` to a remote computer, you cannot create another session to a third computer. As such, the script will run and successfully set the local interfaces to static, but it won't exclude DHCP from providing those addresses to another client.

Building out a PKI environment

Windows Active Directory domains are a great way to authenticate users and computers. Using a central store of accounts and passwords, requests can be easily authenticated, and accounts can be quickly added, updated, or removed as needed. While this is a great method for authentication within the domain, it does not work as well outside of the domain. Situations, where the domain controller may not be accessible, where the authority of the domain controller is in question, or when accessing resources outside of a domain, call for alternative authentication methods.

Certificates allow for creation of an authentication infrastructure by using a series of trusts. Instead of joining a domain, and thereby trusting the domain controllers, you trust a **Certificate Authority** (**CA**). The CA is responsible for handing out certificates that authenticate the user or computer. By trusting the CA, you implicitly trust the certificates it produces.