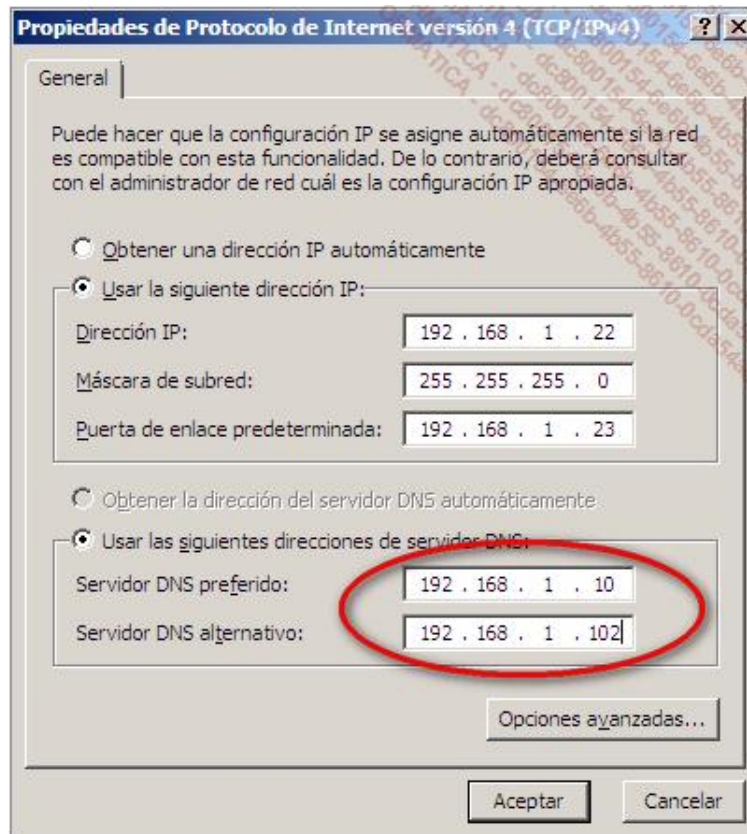


Actualización de la configuración de red de un conjunto de equipos

Problemática

Acabamos de efectuar una migración de nuestros servidores DNS y hemos tenido que instalar nuevas máquinas en sustitución de las antiguas. Por lo tanto para completar esta migración, será preciso actualizar la configuración de red de todos nuestros servidores para que tengan en cuenta este cambio. Puesto que nuestros servidores tienen una configuración de red estática, un script será el encargado de automatizar esta modificación de configuración. Esto nos evitará modificar manualmente este parámetro en cada uno de nuestros servidores. En este estudio de caso, supondremos que el cortafuegos de la red y de los equipos remotos dejan pasar el ping así como las peticiones WMI.

Veamos en la interfaz gráfica los campos que necesitaremos actualizar:



Parámetros de red a cambiar

Solución

Utilizar WMI para consultar y modificar remotamente la configuración de red.

En una primera etapa, vamos a centrarnos en hacer un script que recupere la configuración DNS de una máquina y que devuelva un objeto personalizado. Lo llamaremos `Get-DNSConfiguration.ps1`. Después, en una segunda etapa, haremos el script que nos permita modificar la configuración DNS, lo llamaremos `Set-DNSConfiguration.ps1`. Ambos tendrán como entrada un parámetro para indicar el nombre del equipo sobre el que actuar.

Get-DNSConfiguration.ps1

```
# Get-DNSConfiguration - v.1
#requires -version 2

param ($ComputerName)
```

```

if (Test-Connection $ComputerName -Quiet -Count 1)
{
    # recuperación de interfaces de red activas
    $cnxActivas =
Get-WmiObject Win32_NetworkAdapter -Computer $ComputerName |
    Where {$_.NetConnectionStatus -eq 2}

    # recuperación de la configuración de las interfaces de red activas
    $confActiva =
Get-WmiObject Win32_NetworkAdapterConfiguration -Comp $ComputerName |
    Where { ($_.index -eq $($cnxActivas.index)) }
    $resultado = $confActiva.DNSServerSearchOrder
    $Ping = 'OK'
}
else
{
    $resultado = $null
    $Ping = 'NOK'
}

$objResultado = New-Object PSObject
$objResultado | add-Member -memberType NoteProperty -name HostName -value
$ComputerName
$objResultado | add-Member -memberType NoteProperty -name Ping -value $ping
$objResultado | add-Member -memberType NoteProperty -name DNS -value
$resultado
$objResultado

```

Hagamos una prueba:

```
PS > .\Get-DNSConfiguration.ps1 W2K3R2SRV
```

HostName	Ping	DNS
-----	----	---
W2K3R2SRV	OK	{192.168.1.23, 192.168.1.24}

Perfecto, ¡el script funciona!

Algunas aclaraciones

La palabra reservada `Requires` nos indica que este script sólo funciona con como mínimo la versión 2 de PowerShell. En efecto, haremos una llamada al comando `Test-Connection` y este último no existe en PowerShell v1.

`Param` define la variable `ComputerName`, ésta contendrá el nombre del equipo recuperado en la línea de comandos. Utilizamos, a continuación, el comando `ping` para determinar si el equipo que recibirá la petición WMI está en línea. Para acortar el tiempo de respuesta hemos posicionado el parámetro `Count` a 1, lo que limita a un único envío de ping (por defecto se envían 3 ping). El parámetro `Quiet` permite obtener un valor de retorno de tipo booleano en lugar de un objeto de tipo `Win32_PingStatus`, como por defecto.

Llega después la parte esencial del script: la petición WMI, o más bien deberíamos decir las peticiones. La primera enumera las interfaces de red que se encuentran en estado «conectado». La segunda utiliza otra clase WMI para listar la configuración de red de las interfaces. Le aplicamos un filtro para que nos devuelva únicamente la configuración de las interfaces de red «activas» (conectadas) el identificador de las cuales hemos recuperado con la primera petición.

Consultaremos la propiedad `DNSServerSearchOrder`, ésta nos devolverá la parametrización DNS de la configuración de red activa.

Para concluir, crearemos un objeto personalizado de tipo PSObject y le añadiremos algunas propiedades. Esto permitirá a nuestro script devolver un objeto en lugar de simplemente texto. Usted comprenderá más adelante el interés de hacer esto.

Prueba a escala real

Para probar nuestro script en un número importante de equipos, lo ideal sería ir a buscar el nombre de los equipos directamente a Active Directory Domain Services o algo más sencillo, para empezar, a un fichero de texto.

En este caso, podríamos escribir lo siguiente:

```
PS > Get-Content archEquipos.txt | Foreach {./Get-DNSConfiguration.ps1 $_}
```


Sin embargo, si hemos importado en nuestra sesión los commandlets del módulo Active Directory (véase el Capítulo Ejecución remota - Comunicaciones remotas Windows PowerShell) entonces podremos escribir esto:

```
PS > Get-ADComputer -Filter * | Foreach {./Get-DnsConfiguration $_.Name}
```

En los dos casos, veamos lo que obtendremos:

HostName	Ping	DNS
-----	----	---
W2K8R2VM	OK	{127.0.0.1}
WIN7_US_X64	OK	{192.168.1.23}
WINXP	NOK	
WIN2K8X64	OK	{192.168.1.23}
W2K3R2SRV	OK	{192.168.1.23, 192.168.1.24}
WINXPPSV2	OK	{192.168.1.23}
EXCH2K10SRV	OK	{192.168.1.23, 192.168.1.24}
WINXPPSV1	OK	{192.168.1.23}

Interesante, ¿no? Podemos comentar de paso que la máquina llamada «WINXP» no ha respondido al ping, por lo que está en estado «NOK». Por lo tanto, no hemos efectuado peticiones WMI sobre ella con el fin de evitar un timeout, lo que nos hace ganar tiempo en la ejecución de nuestro script.

 Los valores para la propiedad DNS aparecen entre llaves, esto significa que el resultado obtenido es de tipo tabla (array).

Optimización PowerShell v2 con las funciones avanzadas

Las funciones avanzadas permiten a los scripts comportarse como commandlets nativos. La utilización de estas funciones permitiría a nuestro script aceptar nativamente la tubería como flujo de entrada, lo que nos evitaría tener que hacer un *Foreach* en la línea de comandos y lo simplificaría considerablemente. Además, tendremos siempre la posibilidad de pasar un nombre de equipo al parámetro Computer.

Observamos ahora las modificaciones introducidas en nuestro script del que hemos cambiado de paso el número de versión:

```
# Get-DNSConfiguration - v.2
#requires -version 2

[CmdletBinding()]
param (
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [String[]]$Computers
)
```

```

Process
{
    foreach ($computerName in $Computers)
    {
        if (Test-Connection $ComputerName -Quiet -Count 1)
        {
            # recuperación de interfaces de red activas
            $cnxActivas =
Get-WmiObject Win32_NetworkAdapter -Computer $ComputerName |
            Where {$_.NetConnectionStatus -eq 2}

            # recuperación de la configuración de las interfaces de red activas
            $confActiva =
Get-WmiObject Win32_NetworkAdapterConfiguration -Comp $ComputerName |
            Where { ($_.index -eq $($cnxActivas.index)) }
            $resultado = $confActiva.DNSServerSearchOrder
            $Ping = 'OK'
        }
        else
        {
            $resultado = $null
            $Ping = 'NOK'
        }

        $objResultado = New-Object PSObject
        $objResultado | add-Member -memberType NoteProperty -name HostName
-value $ComputerName
        $objResultado | add-Member -memberType NoteProperty -name Ping -value
$ping
        $objResultado | add-Member -memberType NoteProperty -name DNS -value
$resultado
        $objResultado
    }
}

```

Las modificaciones realizadas en este script son las siguientes:

- Añadimos [CmdletBinding()] para indicar el uso de las funciones avanzadas,
- Adición a nivel del bloque Param de [Parameter(Mandatory=\$true, ValueFromPipeline=\$true)], con el fin de obligar la entrada de un parámetro y autorizar la recepción de parámetros vía la tubería,
- Adición de un bloque Process, necesario en las funciones avanzadas que utilizan la tubería. Hubiéramos podido también añadir los bloques Begin y End.
- Adición de un bucle Foreach para tratar el caso de los parámetros múltiples.

Ahora si tenemos una lista de equipos en un archivo, podemos escribir esto:

```
PS > Get-Content archEquipos.txt | ./Get-DNSConfiguration.ps1
```

O también esto otro:

```
PS > ./Get-DNSConfiguration.ps1 -Computers (Get-Content archEquipos.txt)
```

Y si conseguimos los equipos a partir de Active Directory :

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name |  
.\Get-DNSConfiguration.ps1
```

Será necesario pasar por un filtro de tipo Select-Object con el fin de recuperar únicamente una propiedad de objeto: el nombre. En caso contrario se pasarían las propiedades de todos los objetos a la tubería lo que tendría como consecuencia la generación de un error.

Volvamos a nuestro estudio del caso. Ahora debemos construir el script que llevará a cabo la modificación de la configuración de red. Observaremos éste:

```
# Set-DNSConfiguration  
#requires -version 2  
  
param (  
    $ComputerName = $(throw "¡Debe especificar un nombre de equipo!"),  
    $DNSServerList = @( '192.168.1.30', '192.168.1.40' )  
)  
  
# recuperación de interfaces de red activas  
$cnxActivas = Get-WmiObject Win32_NetworkAdapter -Computer $ComputerName |  
    Where { $_.NetConnectionStatus -eq 2 }  
  
# recuperación de la configuración de las interfaces de red activas  
$confActiva =  
Get-WmiObject Win32_NetworkAdapterConfiguration -Comp $ComputerName |  
    Where { ( $_.index -eq $($cnxActivas.index) ) }  
  
$resultado = $confActiva.SetDNSServerSearchOrder($DNSServerList)  
  
if ($resultado.returnValue -eq 0)  
{  
    Write-Host "$ComputerName: ¡Actualización DNS con éxito!"  
-foreground Yellow  
}  
else  
{  
    Write-Host "$ComputerName: ¡Fallo de la actualización DNS!"  
-foreground Yellow  
}
```

Esta vez nuestro script es casi idéntico a la primera versión del script `Get-DNSConfiguration.ps1` excepto que en lugar de recuperar una propiedad aplica un método. Se trata del método `SetDNSServerSearchOrder`. Este toma como parámetro un objeto de tipo tabla de cadenas de caracteres que contienen las direcciones IP de nuestros nuevos servidores DNS.

Podemos señalar también que tenemos un parámetro de más: `DNSServerList`. Este último se inicializa con los valores; estos últimos constituyen entonces los valores por defecto.

Observemos nuestro script en acción:

```
PS > ./Set-DNSConfiguration -ComputerName WinXPPSv2  
  
WinXPPSv2: ¡Actualización DNS con éxito!
```

Acabamos de aplicar la nueva configuración DNS con los valores por defecto en la máquina WinXPPSv2. Verificamos si ha

funcionado correctamente:

```
PS > ./Get-DNSConfiguration -ComputerName WinXPPSv2
```

HostName	Ping	DNS
-----	----	---
WinXPPSv2	OK	{192.168.1.30, 192.168.1.40}

¡Perfecto, pasemos ahora a un nivel superior! Supongamos que queremos ser selectivos en la manera de aplicar los cambios. Queremos aplicar los nuevos parámetros únicamente a las máquinas que tienen en su configuración DNS la dirección IP 192.168.1.24.

Antes de modificar nada miramos primero la configuración general de nuestro parque de equipos:

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name | .\Get-DNSConfiguration.ps1
```

HostName	Ping	DNS
-----	----	---
W2K8R2VM	OK	{127.0.0.1}
WIN7_US_X64	OK	{192.168.1.23}
WINXP	NOK	
WIN2K8X64	OK	{192.168.1.23}
W2K3R2SRV	OK	{192.168.1.23, 192.168.1.24}
WINXPPSV2	OK	{192.168.1.30, 192.168.1.40}
EXCH2K10SRV	OK	{192.168.1.23, 192.168.1.24}
WINXPPSV1	OK	{192.168.1.23}

Probemos con el filtro siguiente:

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name | .\Get-DNSConfiguration.ps1 |
    Where { ($_.DNS -contains '192.168.1.24') }
```

HostName	Ping	DNS
-----	----	---
W2K3R2SRV	OK	{192.168.1.23, 192.168.1.24}
EXCH2K10SRV	OK	{192.168.1.23, 192.168.1.24}

Hemos obtenido las máquinas haciendo referencia a su configuración de red, un servidor DNS con la dirección 192.168.1.24. Ahora vamos a aplicar únicamente a este grupo de máquinas la nueva configuración DNS, como se muestra a continuación:

```
PS > $a = Get-ADComputer -Filter * | Select Name -Expand Name |
    .\Get-DNSConfiguration.ps1 | Where { ($_.DNS -contains '192.168.1.24') }

PS > $a | foreach { .\Set-DNSConfiguration.ps1 -ComputerName $_.HostName `
    -DNS @('192.168.1.50', '192.168.1.60') }
```

Acabamos de asignar una configuración DNS concreta a nuestra selección de máquinas. Lo que nos da al final el resultado siguiente:

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name | .\Get-DNSConfiguration.ps1
```

HostName	Ping	DNS
-----	----	---

W2K8R2VM	OK	{127.0.0.1}
WIN7_US_X64	OK	{192.168.1.23}
WINXP	NOK	
WIN2K8X64	OK	{192.168.1.23}
W2K3R2SRV	OK	{192.168.1.50, 192.168.1.60}
WINXPPSV2	OK	{192.168.1.30, 192.168.1.40}
EXCH2K10SRV	OK	{192.168.1.50, 192.168.1.60}
WINXPPSV1	OK	{192.168.1.23}

Mmm, todo esto no parece muy homogéneo... Vamos, hagamos un último esfuerzo para poner todas nuestras máquinas idénticas, dejando los valores por defecto de la función Set-DNSConfiguration:

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name |
    foreach { .\Set-DNSConfiguration.ps1 -ComputerName $_ }
```

```
W2K8R2VM: ¡Actualización DNS con éxito!
WIN7_US_X64: ¡Actualización DNS con éxito!
WIN2K8X64: ¡Actualización DNS con éxito!
W2K3R2SRV: ¡Actualización DNS con éxito!
EXCH2K10SRV: ¡Actualización DNS con éxito!
WINXPPSV1: ¡Actualización DNS con éxito!
```

Listo...

```
PS > Get-ADComputer -Filter * | Select Name -Expand Name | .\Get-DNSConfigur
ation.ps1
```

HostName	Ping	DNS
-----	----	---
W2K8R2VM	OK	{192.168.1.30, 192.168.1.40}
WIN7_US_X64	OK	{192.168.1.30, 192.168.1.40}
WINXP	NOK	
WIN2K8X64	OK	{192.168.1.30, 192.168.1.40}
W2K3R2SRV	OK	{192.168.1.30, 192.168.1.40}
WINXPPSV2	OK	{192.168.1.30, 192.168.1.40}
EXCH2K10SRV	OK	{192.168.1.30, 192.168.1.40}
WINXPPSV1	OK	{192.168.1.30, 192.168.1.40}