Prácticas de Infraestructura Informática

Bloque 3: Infraestructuras de scripting y de gestión en plataformas Windows

Sesión 4 - Infraestructura WMI/CIM

Objetivos

El objetivo de esta sesión es introducir la infraestructura WMI (Windows Management Instrumentation). Se trata de una tecnología de gestión implementada de forma estándar en plataformas Windows. Esta tecnología se puede utilizar de múltiples formas. En esta práctica se manejará la infraestructura WMI mediante scripts de PowerShell.

Desarrollo

- **Introducción a WMI**

  WMI es una infraestructura software disponible en todos los sistemas Windows actuales, que proporciona una plataforma para obtener información y gestionar de una forma estandarizada sistemas Windows, ya sea de forma local o remota. Precisamente su habilidad para gestionar equipos de forma remota es lo que proporciona a WMI una gran potencia y flexibilidad, siendo junto con el Directorio Activo una tecnología esencial para la gestión de grandes parques de clientes y servidores Windows.

  WMI es la implementación desarrollada por Microsoft del estándar CIM (Common Information Model). CIM es un estándar desarrollado por la industria TI, cuyo objetivo es proporcionar una definición común de información de gestión para sistemas, redes, aplicaciones y servicios, y que ha sido diseñado para facilitar la interoperabilidad entre todo tipo de sistemas de TI.

  Información sobre WMI

  Microsoft proporciona una plataforma de recursos on-line para usuarios finales, programadores y profesionales de TI conocida como Microsoft Docs (docs.microsoft.com). En esta plataforma se proporciona una información suficientemente detallada para comenzar a trabajar con la tecnología WMI.

  - Para encontrar esta información, introduce en Google ***`Microsoft Docs WMI Start Page`*** y lee los apartados sobre el propósito y la aplicabilidad de esta tecnología.

  Arquitectura de WMI

  Antes de empezar a utilizar esta tecnología es esencial comprender su arquitectura. En la figura de más abajo se muestra la arquitectura de WMI.

  En la arquitectura de WMI se distinguen tres tipos de elementos, que se describen a continuación:

  1) **Infraestructura WMI**. Ésta consta de dos elementos fundamentales: el repositorio y el manejador de objetos.

     El repositorio es un subsistema cuyo objetivo es almacenar un registro de todas aquellas clases que son utilizadas en el contexto de la plataforma WMI. Una parte de las clases registradas en el repositorio proporcionan funcionan genéricas de la infraestructura WMI, como por ejemplo, gestionar las conexión a un equipo remoto, o registrar eventos ante los que poner en marcha determinadas acciones. Otra parte de las clases registradas en el repositorio es aportada por los *proveedores*, que son los componentes que gestionan objetos concretos del sistema, y que se describen más abajo.

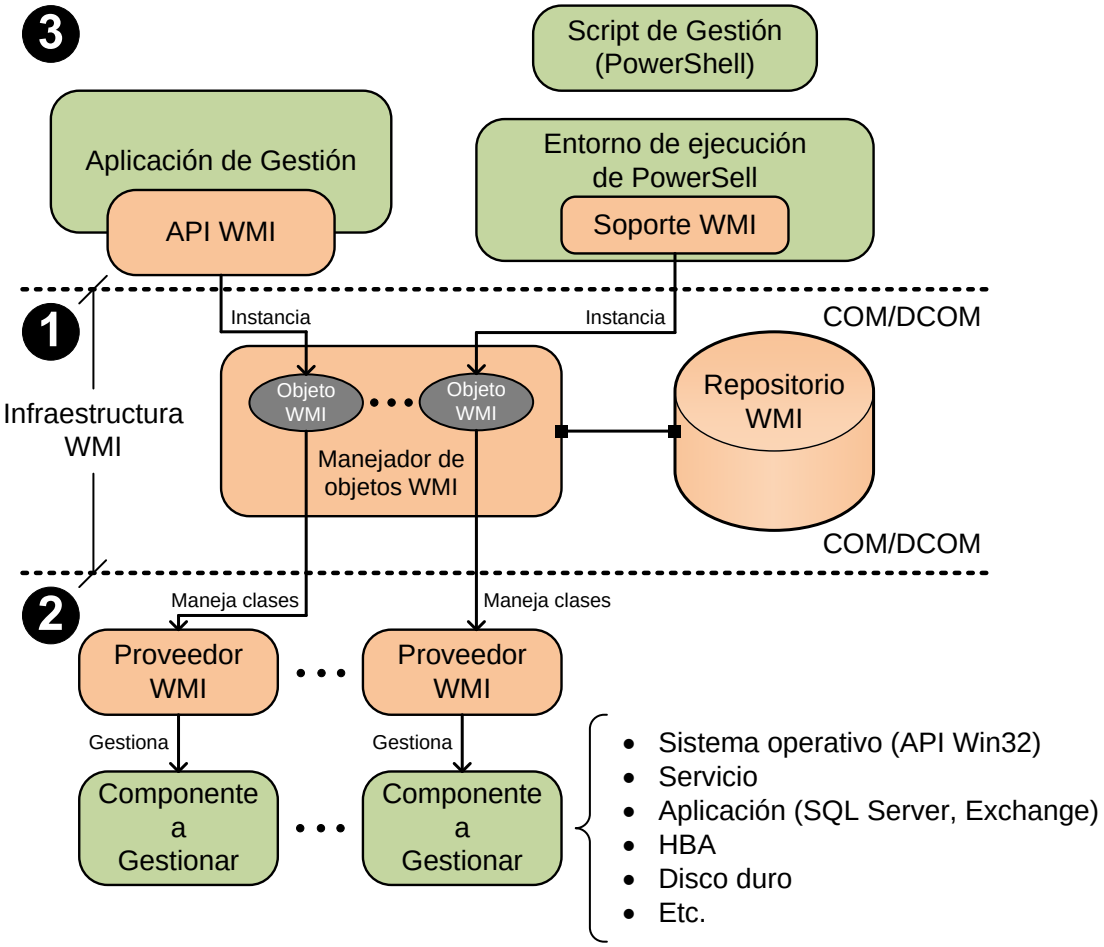
     El manejador de objetos se implementa mediante un servicio de Windows. En concreto se trata del *Servicio Instrumental de Administración* de Windows. El manejador de objetos implementa

los mecanismos necesarios para que las aplicaciones de gestión (parte 3 de la figura) se puedan conectar al repositorio e instanciar objetos a partir de las clases registradas en dicho repositorio. Los objetos WMI instanciados se cargan en la memoria del manejador de objetos.

2) **Proveedores**. Un proveedor es un objeto software diseñado para integrase en la infraestructura WMI, y que está orientado a monitorizar un componente del sistema o a actuar sobre el mismo. El componente manejado puede ser físico o lógico. Ejemplos de objetos manejados son un disco duro, una impresora, un programa instalado en el sistema, una cuenta de usuario, el propio sistema operativo, el propio computador como entidad completa etc. Las clases implementadas en un proveedor se registran en el repositorio WMI. De esta forma, dichas clases pueden ser utilizadas por las aplicaciones de gestión para monitorizar o gestionar objetos del sistema.

   La infraestructura WMI incluye un número significativo de proveedores estándar. A modo de ejemplo, el *Win32 provider* permite manejar un amplio conjunto de funciones del sistema operativo. Aparte de los proveedores incluidos de forma estándar, la infraestructura WMI proporciona las herramientas necesarias para integrar nuevos proveedores. De esta forma, la infraestructura WMI puede adaptarse sin limitación alguna a la gestión de cualquier tipo de elemento físico o lógico instalado en una plataforma Windows.

3) **Aplicaciones de gestión**. Estas aplicaciones son las que usan la infraestructura WMI, y pueden ubicarse, o bien en el mismo ordenador en el que se encuentra la infraestructura WMI a utilizar, o bien en un ordenador remoto. Estas aplicaciones pueden escribirse en diferentes lenguajes de programación. Si se escriben en C++ o en C# utilizan librerías que implementan la API WMI. Esta API está orientada a gestionar la infraestructura WMI desde la aplicación e incluye operaciones como conectarse al repositorio WMI e instanciar objetos en el mismo. Otra posibilidad para manejar la infraestructura WMI es utilizar lenguajes de scripting como PowerShell. En este caso, el entorno de ejecución de scripts incluye el soporte necesario para manejar la infraestructura WMI. Gracias a ello, el manejo de la infraestructura WMI desde PoweShell resulta de gran simplicidad.



- **La primara parte de esta práctica se realizará en la máquina virtual PLX-S-FS. Arranca esta máquina. Inicia sesión en ella con el administrador local.**

  (1) EJERCICIO. Abre la herramienta *Servicios* (accesible a través del menú *Herramientas administrativas*) y busca el servicio *Instrumental de Administración* de Windows. En primer lugar, comprueba que el servicio se encuentra iniciado. Después, averigua cuál es el nombre corto del servicio. Indícalo a continuación:

  El servicio *Instrumental de Administración de Windows* es el centro de las operaciones de WMI. Tiene la habilidad de instanciar las clases definidas en el repositorio a petición de las aplicaciones de gestión. Las aplicaciones de gestión actúan sobre los proveedores instanciando sus clases en el servicio *Instrumental de Administración de Windows*. Así, este servicio actúa como un intermediario entre las aplicaciones de gestión y los proveedores.

Proveedores WMI

Los proveedores WMI son los elementos que permitir interaccionar con los diferentes componentes y objetos de un sistema Windows. Los proveedores proporcionan clases para manejar los objetos con los que interaccionan.

Proveedores estándar

La infraestructura WMI viene con un conjunto de proveedores instalados por defecto.

- Para obtener información de los proveedores disponibles por defecto (estándar), en la página web que abriste anteriormente, navega hasta *WMI Reference -> WMI Providers -> WMI Providers*. Esto lleva a una página en la que se muestran todos los proveedores disponibles.

Win32 Provider

El proveedor Win32 tiene por objetivo proporcionar un mecanismo de gestión del sistema operativo en su conjunto, proporcionando un extenso número de funciones para manejar las funciones de la API Win32. Dicha API es el mecanismo fundamental para acceder a los servicios del sistema operativo Windows.

- Pulsa sobre el enlace correspondiente al *Win32 provider*. Las clases disponibles en este proveedor se organizan por funcionalidades, tales como *Computer System Hardware Classes*, *Operating System Classes*, etc.

  A continuación, se explorará una clase concreta del *Win32 provider*. Se trata de la clase utilizada para obtener información de los discos duros instalados en el sistema.
- La clase buscada se encuentra en el subgrupo *Computer System Hardware Classes*. Despliega este subgrupo. Se observa, a su vez, que este subgrupo se encuentra organizado en un conjunto de categorías de dispositivos. La categoría correspondiente a los discos duros es *Mass Storage Classes*. Abre esta categoría para observar las clases disponibles para manejar este tipo de dispositivo. La clase correspondiente al tipo de dispositivo disco duro es *Win32_DiskDrive*.
- Despliega la clase *Win32_DiskDrive*. Inicialmente se proporciona una breve descripción del cometido de la clase, a continuación su definición sintáctica, y finalmente sus miembros, que son los métodos y las propiedades. En el caso de esta clase se puede observar que hay dos métodos definidos, aunque no se encuentran actualmente implementados.

  (2) EJERCICIO. Explora las propiedades de la clase Win32_DiskDrive y contesta a las siguientes preguntas:

  A) ¿Qué posibles valores puede tomar la propiedad *InterfaceType*?

  B) ¿Existe alguna propiedad en esta clase cuyo tipo de acceso sea diferente de solo lectura (*read-only*)?

C) ¿Qué implica la respuesta anterior?

```
A)
B)
C)
```

Espacios de nombres

Con objeto de facilitar el nombrado de las clases, éstas se organizan en espacios de nombres (*namespaces*). Para acceder a una determinada clase será necesario indicar el espacio de nombres al que pertenece. En la ayuda de cada clase se indica el espacio de nombres correspondiente a dicha clase.

(3) EJERCICIO. Determina el espacio de nombres al que pertenece la clase *Win32_DiskDrive*. Puedes encontrar esta información en el apartado *Requirements*.

• Manejo de WMI desde PowerShell

Para ilustrar el manejo de WMI desde PowerShell, se utilizará la clase Win32_DiskDrive, que será utilizada para explorar la información de los discos de la máquina PLX-S-FS. En este momento, esta máquina tiene dos discos, uno de sistema y otro de datos. No obstante, inicialmente, para entender mejor el funcionamiento de Win32_DiskDrive, es más aconsejable empezar analizando una máquina con un solo disco. Debido a ello, antes de comenzar este apartado de la práctica, vas a quitar el disco de datos de la máquina.

- Apaga la máquina PLX-S-FS
- En el Administrador de Hiper-V, abre la configuración de la máquina virtual. Debes observar que la máquina tiene dos discos instalados bajo la controladora SCSI. Se trata de los discos PLX-S-FS.vhdx y PLX-S-FS-DATA.vhdx. El primero es el disco de sistema, y el segundo, el de datos.
- Desconecta de la máquina el disco de datos.
- Arranca la máquina.

CmdLet Get-CimInstance

El comando *Get-CimInstancet* es utilizado para instanciar las clases de WMI, ya sea en el ordenador local o en un ordenador remoto. Una vez que se obtiene acceso a un objeto de WMI, éste es manejado igual que cualquier otro objeto gestionado por PowerShell.

- En primer lugar, es necesario conocer el funcionamiento del comando *Get-CimInstance*. En la MV PLX-S-FS, los ficheros de ayuda de PowerShell no han sido instalados, debido a ello, mejor buscar la ayuda en Internet. En el navegador del sistema anfitrión, introduce el nombre de este CmdLet. Esto te llevará directamente a la ayuda del comando. Una vez revisada la ayuda, realiza el siguiente ejercicio:

  (4) EJERCICIO. En una consola de PowerShell, ejecuta el comando *Get-CimInstance* para instanciar la clase *Win32_DiskDrive* en el ordenador local. No asignes el objeto u objetos obtenidos por PowerShell a ninguna variable. Ejecuta este comando utilizando los siguientes parámetros: -ClassName, -Namespace y -ComputerName. Para indicar el computador local se utiliza el carácter punto (.). Indica el comando ejecutado a continuación.

  La salida del comando anterior no muestra el contenido de todas las propiedades del objeto instanciado, sino solo de algunas de ellas. Cada objeto muestra un listado de propiedades por defecto. En el caso de este objeto, PowerShell muestra las propiedades *DeviceID*, *Caption*, *Partitions*, *Size*, *Model*, *PSComputerName*. En este caso, el objeto instanciado corresponde al disco de sistema de la máquina virtual, que en este momento, es el único disco de la máquina.

(5) EJERCICIO. ¿Qué número de particiones indica el objeto Win32_DiskDrive?

- Para ver cuáles son estas particiones, se puede utilizar el *administrador de discos*. Abre *Administración de equipos -> Administración de discos*. Debes observar que hay tres particiones en el disco, si bien solo una de ellas tiene asignada letra de volumen.
- Cierra la herramienta *Administración de equipos*.
- Para ver todas las propiedades del objeto *Win32_DiskDrive*, ejecuta el comando Get-CimInstnace de la misma forma que en el Ejercicio 4, si bien añadiendo el parámetro -Property *.

(6) EJERCICIO. Indica el nombre de las propiedades que describen la geometría del disco. Si tienes dudas, pregúntale a tu profesor.

- Comprueba que todas las propiedades indicadas forman parte de la definición de la clase *Win32_DiskDrive*, según se muestra en el apartado Syntax de la ayuda de WMI para esta clase.

  El objeto de tipo *Win32_DiskDrive* instanciado por Get-CimInstance debe tener todas las propiedades indicadas en la ayuda de WMI para ese tipo de objeto, si bien Get-CimInstance añade alguna propiedad adicional, como por ejemplo, *PSComputerName* (que contiene el nombre del sistema sobre el que se ejecuta el comando) o *CimClass* (que contiene una cadena de caracteres con el nombre de la clase)

(7) EJERCICIO. ¿Qué valor almacena la propiedad *PSComputerName*?

Dicho valor debe ser consistente con el valor pasado al parámetro –ComputerName en la ejecución de Get-CimInstance.

(8) EJERCICIO. Indica los valores de las propiedades que se muestran a continuación:

BytesPerSector:
InterfaceType:
Model:

(9) EJERCICIO. Mirando la ayuda de WMI, indica el tipo de dato correspondiente a cada una de las tres propiedades indicadas en la pregunta anterior.

Tipo de *BytesPerSector*:
Tipo de *InterfaceType*:
Tipo de *Model*:

(10) EJERCICIO. Explica la razón del valor almacenado en la propiedad *InterfaceType*.

Un dato importante de los discos es el tamaño (capacidad). Vas a buscar este dato en la información proporcionada por la clase *Win32_DiskDrive*. No obstante, primero, vas a obtener esta información utilizando el administrador de discos, accesible desde la herramienta *Administración de equipos*.

- Abre el *Administrador de discos*. Observa que el sistema tiene instalado un disco, nombrado como Disco 0. Entonces, contesta la siguiente pregunta:

(11) EJERCICIO. ¿Cuál es el tamaño del disco 0 expresado en GB?

- En la clase *Win32_DiskDrive*, el tamaño del disco se almacena en la propiedad *Size*, y se expresa en bytes.

  (12) PREGUNTA. Indica el número de bytes almacenado en la propiedad *Size*.

  Para comprobar que el valor almacenado en *Size* es consistente con la capacidad mostrada en el *Administrador de discos*, debes saber que Windows utiliza potencias de 2 para expresar los múltiplos del byte, es decir, 1 GB = 2^30 = (1024*1024*1024) bytes.

- En la línea de comandos de PowerShell, divide el valor constado en la Pregunta 12 (propiedad *Size*) entre (1024 * 1024 * 1024), y comprueba que obtienes el resultado esperado (muy aproximadamente).

Objetos WMI / CIM

Cuando se instancia una clase WMI / CIM mediante Get-CimInstance, este comando devuelve un objeto o una colección de objetos correspondientes a la clase instanciada. Entonces, el objeto u objetos devueltos pueden ser almacenados en una variable. Para probar esto, volverás a utilizar la clase *Win32_DiskDrive*.

- Ejecuta el siguiente comando:

  > $a = Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –ComputerName .

  El comando anterior obtiene un objeto de la clase *Win32_DiskDrive* y lo almacena en la variable $a.

- Para ver el tipo de objeto almacenado en la variable $a, puedes utilizar el comando Get-Member.

  (13) EJERCICIO. Indica, a continuación, el tipo de objeto almacenado en la variable $a.

  Una vez el objeto WMI está en una variable, se puede acceder a sus propiedades de la misma forma que a cualquier otro objeto de PowerShell.

  (14) EJERCICIO. Ejecuta un comando Write-Host que imprima en la consola el nombre del disco (propiedad *Name*). Escribe el comando a continuación.

- El resultado del comando anterior debe ser `\\.\PHYSICALDRIVE0`.

  Hasta este momento, el instanciado de la clase Win32_DiskDrive ha devuelto un objeto simple, ya que la máquina virtual PLX-S-FS cuenta con un solo disco. En este punto, vas a volver a agregar el disco de datos a la máquina virtual, de modo que pasará a tener dos discos instalados. Entonces, analizarás el nuevo comportamiento que producirá el instanciado de la clase Win32_DiskDrive.

- Apaga la máquina PLX-S-FS.
- En el Administrador de Hyper-V, abre la configuración de la máquina virtual. Agrega el disco PLX-S-FS-DATA.vhdx como segundo disco conectado a la controladora SCSI.
- Arranca la máquina.
- Abre el *Administrador de discos* y comprueba que hay dos discos instalados, el Disco 0 y el Disco 1.
- Con objeto de volver a instanciar la clase *Win32_DiskDrive*, en una consola de PowerShell ejecuta el siguiente comando:

  > Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –ComputerName .

- Debes observar que la salida del comando anterior proporciona información sobre los dos discos instalados en el sistema. Observa que el nombre (propiedad *Name*) del primer disco es `\\.\PHYSICALDRIVE0`, y el nombre del segundo, `\\.\PHYSICALDRIVE1`. Asimismo, observa que el número de particiones en el primer disco (propiedad *Partitions*) es 3, mientras que el número de particiones en el segundo disco es 1, tal y como puedes comprobar en el *Administrador de discos*.

  Si en este punto, se guarda el valor devuelto por el comando Get-CimInstance en una variable, lo que se almacena en la variable es un array de dos objetos, correspondiendo cada objeto del array a uno de los discos.

- Ejecuta el siguiente comando.

  > $a = Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –ComputerName .

  (15) EJERCICIO. Ejecuta un comando Write-Host que imprima en la consola el nombre del primer disco (propiedad *Name*). Escribe el comando a continuación.

  (16) EJERCICIO. Ejecuta un comando Write-Host que imprima en la consola el nombre del segundo disco. Escribe el comando a continuación.

• Instanciado remoto de clases WMI/CIM

En los ejemplos anteriores se ha utilizado el comando Get-CimInstance para instanciar clases en el ordenador local y, consecuentemente, obtener información de dicho ordenador. Una de las grandes ventajas de WMI es que permite trabajar con máquinas remotas de la misma forma que con la máquina local.

Para el instanciado remoto de clases WMI/CIM, Windows puede utilizar diversos mecanismos. No obstante, la recomendación de Microsoft es que se utilice la infraestructura conocida como Windows Remote Management (WinRM), cuyo funcionamiento se basa en el protocolo WS-MAN. WinRM ya ha sido introducido en la sesión de prácticas *Fundamentos de PowerShell (II)*. En dicha sesión se explicó que la ejecución remota de comandos PowerShell utiliza WinRM como infraestructura de interconexión entre el ordenador cliente y el ordenador remoto.

De forma estándar, el comando Get-CimInstance utiliza WinRM para conectarse a la infraestructura WMI de un equipo remoto. No obstante, para poder gestionar de forma remota un equipo, WinRM debe estar habilitado en el equipo. En el caso de Windows Server 2019, WinRM está habilitado por defecto.

- Para verificar que WinRM se encuentra configurado para la administración remota del equipo, en PLX-S-FS, ejecuta el siguiente comando.

  `> winrm quickconfig`

  Deberás obtener la siguiente respuesta:

  ```
  El servicio WinRM ya está ejecutándose en esta máquina.
  WinRM ya está configurado para administración remota en este equipo.
  ```

  Una vez comprobado que PLX-S-FS está configurada para ser gestionada remotamente, en este apartado de la práctica se procederá a acceder a la infraestructura WMI de PLX-S-FS de forma remota. El acceso remoto se realizará desde el ordenador cliente PLX-C-51. Asimismo, para poder llevar a cabo determinadas operaciones, será necesario utilizar las credenciales de algún usuario del domino. Debido a ello, el controlador de dominio debe estar operativo.

- Arranca PLX-S-DC. No es necesario que inicies sesión. En el momento que se muestre la pantalla de bloqueo, el controlador de dominio estará plenamente operativo.
- Arranca PLX-C-51 e inicia sesión con el usuario local *Alumno*.

- Comprueba que hay conectividad entre PLX-C-51 y PLX-S-FS para que se pueda llevar a cabo el acceso remoto.

Sesiones CIM

Una forma de gestionar el instanciado remoto de clases WMI/CIM en un equipo es establecer una sesión remota con el equipo, y utilizar dicha sesión en el comando Get-CimInstance para instanciar las clases en el equipo. Windows gestiona las sesiones remotas mediante la infraestructura WinRM.

- Para establecer una sesión CIM se utiliza el cmdlet New-CimSession. Busca en Internet la ayuda del cmdlet New-CimSession. Observa la estructura de parámetros del comando. En el ejemplo que realizarás en esta sección, utilizarás los parámetros -ComputerName y -Credencial.
- Lee la ayuda sobre el parámetro -ComputerName. Entonces contesta la siguiente pregunta:

  (17) EJERCICIO. Indica los tres formatos en los que se puede especificar el nombre de un computador en el parámetro -ComputerName.

- Lee la ayuda sobre el parámetro -Credential. Puedes observar que en este parámetro se indica el usuario cuyas credenciales se utilizan para crear la sesión remota. Como el equipo PLX-S-FS forma parte de un dominio, en este parámetro utilizarás un usuario del dominio.
- A continuación, vas a crear una sesión remota para instanciar clases WMI/CIM en PLX-S-FS. Para ello, primero probarás usando las credenciales de un usuario cualquiera del dominio, por ejemplo, las de *Prof1*. Respecto al nombre del equipo, utilizarás el nombre NetBIOS, o sea, PLX-S-FS. Entonces, en PLX-C-51, ejecuta el siguiente comando:

  > $Se = New-CimSession –Credential "practicas\Prof1" –ComputerName PLX-S-FS

  El comando solicita que se introduzca la contraseña del usuario. Introduce la contraseña solicitada.

  Se produce el error Acceso denegado. En este caso, el problema es que el usuario del dominio *Prof1* no tiene privilegios suficientes para crear una sesión CIM con la máquina PLX-S-FS. Para poder crear este tipo de sesiones se requiere un usuario con privilegios de administración en el dominio, como por ejemplo, el Administrador del dominio.
- Repite el comando anterior, pero en el parámetro -Credential, utiliza "practicas\Administrador".
- En este caso la sesión CIM se crea sin problemas.
- Para ver las sesiones CIM abiertas, en PLX-C-51, ejecuta el comando siguiente:

  > Get-CimSession

  Puedes observar que se asigna un Id a cada sesión, que se va incrementando en la medida que se crean nuevas sesiones. También se observa que el protocolo usado por defecto es el WSMAN.
- Debes obtener la misma información de la variable $Se. Para ello, ejecuta

  > $Se
- Ahora, utilizando la sesión establecida con PLX-S-FS, se procede a instanciar remotamente en este sistema la clase *Win32_DiskDrive*. Para ello, en PLX-C-51, ejecuta el comando siguiente:

  > Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –CimSession $Se

  El comando debe ejecutarse correctamente, mostrando la información de los discos ubicados en el servidor PLX-S-FS.
- Para eliminar la sesión abierta, en PLX-C-51, ejecuta el comando siguiente:

  > Remove-CimSession –Id #

  Donde # es el número del *Id* de la sesión abierta.
- Finalmente, en PLX-C-51, ejecuta

  > Get-CimSession

El comando no debe retornar ningún valor, ya que en este punto no debe haber ninguna sesión CIM abierta en el sistema.

Instanciado de clases WMI/CIM con el parámetro -ComputerName

El comando Get-CimInstance, además de proporcionar el parámetro –CimSession para indicar el ordenador remoto, también proporciona como alternativa el parámetro –ComputerName. Cuando se utiliza este último parámetro, el ordenador cliente establece una sesión CIM temporal con el ordenador remoto, que permanece abierta solo durante la ejecución del comando. Adicionalmente, cuando se utiliza el parámetro –ComputerName, las credenciales presentadas al ordenador remoto son las del usuario que ejecuta el comando Get-CimInstance en el ordenador cliente.

- Para probar el parámetro –ComputerName, en PLX-C-51, ejecuta el comando siguiente:

  > Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –ComputerName PLX-S-FS
- Observarás que ocurre un error de ejecución. Entre las causas posibles se indica "El nombre de usuario o la contraseña especificada no son válidos". Esto es lo que está ocurriendo. En el ordenador cliente, tienes una sesión iniciada con el usuario local *Alumno*. Por consiguiente, las credenciales presentadas por Get-CimInstance a la máquina PLX-S-FS son las de un usuario que no pertenece al dominio, y que por tanto, no puede llevar a cabo ninguna operación en PLX-S-FS.

  La solución al problema anterior es utilizar en la máquina cliente un usuario del dominio con privilegios suficientes para instanciar clases WMI/CIM remotamente en PLX-S-FS. El usuario que tiene asignados estos derechos es el *Administrador* del dominio.
- En PLX-C-51, cierra la sesión abierta con el usuario local *Alumno* y abre una nueva sesión con el *Administrador* del dominio.
- Ahora, en PLX-C-51 volverás a ejecutar el comando que falló antes. Abre una consola de PowerShell. Observa que para el usuario Administrador, la consola de PowerShell se presenta con fondo negro, en vez de azul. Ahora, ejecuta el comando siguiente:

  > Get-CimInstance –ClassName Win32_DiskDrive –NameSpace Root\CIMV2 –ComputerName PLX-S-FS

  En este caso, la ejecución se llevará a cabo con éxito, debido a que las credenciales presentadas por la máquina cliente a la máquina remota (PLX-S-FS) son las del *Administrador del dominio*.

Reflexión

La infraestructura WMI, junto con WinRM, proporcionan un mecanismo esencial para la gestión de grandes parques de equipos Windows de forma centralizada. En esta sesión de prácticas simplemente se ha probado el instanciado de clases de forma remota, pero este mecanismo es el utilizado por la inmensa mayoría de herramientas utilizadas en el ámbito de la gestión centralizada de plataformas Windows. Por otra parte, una vez más, se comprueba cómo la infraestructura de Directorio Activo proporciona un soporte inestimable en la gestión de credenciales de acceso a sistemas Windows.

• Ejercicio (script básico)

En este apartado se plantea la realización de un script elemental que obtiene información básica del equipo local, utilizando la infraestructura WMI. Para ello, se utiliza la clase *Win32_ComputerSystem*.

- Este script se realizará y probará en el máquina PLX-C-51, que ya debe estar arrancada. No obstante, en este punto, el usuario que tiene sesión abierta en este equipo es el *Administrador del dominio*. Como el script a realizar en este ejercicio se ejecutará localmente, no es necesario utilizar este usuario (las buenas prácticas de seguridad indican que el *Administrador del dominio* solo debe utilizarse cuando sea estrictamente necesario). Debido a ello, cierra la sesión abierta con el *Administrador del dominio* e inicia una nueva sesión con el usuario local *Alumno*.
- En la máquina PLX-C-51, ya se dispone de una carpeta para almacenar *scripts*. Se trata de la carpeta *C:\Scripts*.

- La clase *Win32_ComputerSystem* pertenece al *Win32 Provider*, grupo *Operating System Classes*, categoría *Operating System Settings*. Navega por la ayuda de WMI hasta posicionarte en esta clase. Podrás observar que esta clase proporciona información general sobre el sistema, como por ejemplo el nombre del equipo, el dominio al que pertenece, etc.
- En primer lugar se instanciará la clase sin almacenarla en ninguna variable. Se trata de ver la información almacena en dicha clase. Para ello, en una consola de PowerShell, ejecuta el siguiente comando:

  > Get-CimInstance –ClassName Win32_ComputerSystem –NameSpace Root\CIMV2 –Property *
- En el comando anterior se ha omitido el parámetro -ComputerName. Cuando se omite este parámetro, su valor por defecto es el ordenador local.
- Observa que entre los datos proporcionados por esta clase está el dominio y el nombre del equipo. Comprueba que el valor de estas propiedades es consistente.
- La propiedad *Model* indica el tipo de sistema sobre el que se ejecuta el sistema operativo.

(18) EJERCICIO. ¿Qué indica dicha propiedad?

(19) EJERCICIO. Realiza un *script* llamado *ComputerInfo.ps1* que muestre en la consola la siguiente información:

Nombre de equipo: XXX
Dominio: YYY
Rol en el dominio: ZZZ

Donde XXX, YYY y ZZZ son los valores obtenidos de las propiedades correspondientes de la clase *Win32_ComputerSystem*. No obstante, en el caso de la propiedad *Rol en el dominio*, el valor numérico proporcionado por la propiedad debe interpretarse, de modo que el script muestre una cadena de caracteres explicativa según lo indicado en la ayuda de WMI. Así, si el valor de la propiedad es 0, el script debe mostrar *Standalone Workstation*, etc.

El script se almacenará en la carpeta *C:\Scripts*.

Se recomienda escribir el script utilizando la herramienta PowerShell ISE.

NOTAS:

– Para convertir el número proporcionado por la propiedad *DomainRole* en la cadena de caracteres correspondiente se utilizará la sentencia *switch*. Para obtener ayuda de esta sentencia, utiliza el comando *Get-Help about_Switch*.

- **Ejercicios para la evaluación de scripts**

  Script *FullComputerInfo.ps1*. Se trata de programar un script que proporcione información variada relativa al computador local en el que se ejecuta. La información proporcionada por el *script* se almacena en un fichero de texto llamado *CompunterName_Info.txt*, donde *ComputerName* es el nombre del computador en el que se ejecuta. El fichero se almacena siempre en el directorio *C:\Temp*. Antes de generar el informe, el script debe realizar dos comprobaciones: 1) que *C:\Temp* existe y 2) que en este directorio no existe un fichero con el mismo nombre que el que va a ser generado por el *script*. Si estas condiciones se cumplen, el fichero informativo se crea y el script envía un mensaje a la consola indicando que se ha creado con éxito el fichero informativo *CompunterName_Info.txt*. Si las condiciones anteriores no se cumplen, el script genera un mensaje informativo del problema ocurrido y aborta su ejecución.

  A continuación se proporciona la estructura del informe a generar por el *script*:

```
----------------------------- Identificación del equipo ---------------------------------
Nombre equipo:                XXX
Dominio:                      XXX

----------------------------- Información del sistema operativo -------------------------
Edición de Windows:           XXX
Fecha de instalación:         XXX
Directorio de Windows:        XXX
Arquitectura del SO:          XXX
Tipo de producto:             XXX [Expresado en cadena de caracteres, según ayuda]

----------------------------- Listado de discos -----------------------------------------
Nº de discos:                 XXX

============ Disco 0 =========
Nombre:                       XXX
Modelo:                       XXX
Tamaño (en GB):               XXX

...

============ Disco N =========
Nombre:                       XXX
Modelo:                       XXX
Tamaño (en GB):               XXX


----------------------------- Listado de volúmenes --------------------------------------
Nº de volúmenes:              XXX

============ Volumen 0 =======
Letra asignada:               XXX
Capacidad (en MB):            XXX
Espacio libre (en MB):        XXX
Tipo de drive:                XXX [Expresado en cadena de caracteres, según ayuda]
Sistema de ficheros:          XXX

...

============ Volumen N ==========
Letra asignada:               XXX
Capacidad (en MB):            XXX
Espacio libre (en MB):        XXX
Tipo de drive:                XXX [Expresado en cadena de caracteres, según ayuda]
Sistema de ficheros:          XXX
```

  NOTAS:

  – En la estructura del informe se observa que los valores a mostrar se escriben todos en una posición alineada. Para conseguir esto, se compondrá cada línea a escribir utilizando formateo extendido de cadenas, según se ha estudiado en la sesión de prácticas anterior.

  – Algunas clases WMI, al ser instanciadas, pueden devolver o bien un objeto simple, o bien un array de objetos, siendo necesario hacer un tratamiento diferenciado en cada uno de estos casos. Para conocer si el resultado del instanciado de una clase WMI es un objeto simple o un array de

objetos, debe utilizarse la propiedad IsArray del método GetType(), aplicable a cualquier variable. A continuación, se proporciona una ejemplo:

```
$a = 1, 2, 3
$a.GetType().Isarray
True
$a = 1
$a.GetType().Isarray
False
```

- Para conocer los discos instalados en el sistema debe utilizarse la clase *Win32_DiskDrive*. Esta clase debe tratarse de la forma apropiada utilizando GetType().Isarray.
- Para conocer los volúmenes instalados en el sistema debe utilizarse la clase *Win32_Volume*. Esta clase debe tratarse de la forma apropiada utilizando GetType().Isarray.

Script *GestionaServicio.ps1*. Se trata de programar un script que permita gestionar un servicio de Windows. El servicio se debe gestionar mediante la clase *Win32_Service* de la infraestructura WMI. En este script se gestionan solamente servicios del ordenador local. En primer lugar el script solicita al usuario el nombre del servicio que se desea gestionar. Entonces el script determina si el nombre introducido corresponde a algún servicio instalado en el sistema. En el caso de que no exista esta correspondencia, el script envía a la consola un mensaje indicando el problema y aborta la ejecución. En caso de que sí exista, se muestra un menú de opciones indicando las operaciones que se pueden llevar a cabo con el servicio. Las opciones se numeran del 1 al 5. El script debe esperar a que el usuario introduzca uno de estos números y, a continuación, llevar a cabo la operación correspondiente. Una vez realizada la operación, el script termina. El script debe analizar si la opción introducida por el usuario es válida, es decir, del 1 al 5. En caso contrario, debe indicar que se ha introducido una opción inválida y terminar.

Las operaciones posibles a realizar con el servicio son las siguientes:

1) Obtener una descripción del cometido del servicio
2) Indicar el modo de arranque *(startmode)* del servicio
3) Indicar el estado del servicio
4) Arrancar el servicio
5) Parar el servicio

NOTAS:

- Una clase WMI puede representar a una colección de objetos del sistema. Por ejemplo, la clase Win32_Service puede representar a todos los servicios en ejecución en el sistema. Sin embargo, en este script interesa trabajar con un servicio concreto: aquel cuyo nombre es introducido por el usuario en la consola. Para obtener un objeto concreto de una clase, WMI proporciona un mecanismo conocido como Query. En esencia, se trata de una pregunta que se hace a la infraestructura WMI para obtener un objeto o una colección de ellos correspondientes a una determinada clase. Las preguntas se formulan utilizando un lenguaje muy simple e intuitivo conocido como WQL *(Wmi Query Language)*. En la ayuda de PowerShell existe un tema describiendo este lenguaje, junto con un amplio conjunto de ejemplos acerca de la realización de *"queries"*. Se trata del tema *about_WQL*. (NOTA: en varios ejemplos de este tema de ayuda se utiliza el comando Get-WmiObject. Se trata de una versión equivalente, pero desfasada, del comando Get-CimInstance utilizado en estas prácticas.)
- Para determinar si una *"query"* realizada mediante el comando Get-CimInstance obtiene un objeto válido, se puede comparar la variable receptora del objeto con el valor *$null*. Si la variable contiene *$null*, significa que la *"query"* no ha obtenido ningún objeto válido.
- Los objetos CIM retornados por Get-CimInstance tienen características especiales. Debido a ello, los métodos de estos objetos no se pueden invocar directamente a partir de los objetos correspondientes mediante el operador punto (.), sino que deben invocarse utilizando el cmdLet

Invoke-CimMethod. Puedes buscar la sintaxis, así como diversos ejemplos de uso de este cmdLet, en la ayuda de PoweShell.

SCRIPT DEMOSTRATIVO:

Antes de empezar a programar el script *GestionaServicio.ps1*, debes analizar y probar en tu propio sistema el script demostrativo que se presenta en este punto.

El objetivo de este script es ilustrar dos mecanismos que serán necesarios en la programación de *GestionaServicio.ps1*. Se trata de la realización de *"queries"* mediante el parámetro –Query, y de la invocación de métodos en objetos CIM. El script demostrativo hace uso de la clase *Win32_Directory*. Puedes buscar la descripción de esta clase en la ayuda de WMI. La clase *Win32_Directory* representa un directorio del sistema de archivos del ordenador. El script realiza una operación muy simple: renombra una carpeta, que debe estar ubicada en el directorio *C:\Temp*, y cuyo nombre es introducido a través de la consola. Para realizar esta operación, el script utiliza el método *Rename* de la clase *Win32_Directory*.

En el script demostrativo se utilizan comandos que usan varias líneas. Para indicar a PowerShell que un comando continúa en la línea siguiente se utiliza el carácter (`). Este carácter debe estar precedido obligatoriamente por un espacio.

En el script se observa que el comando Invoke-CimMethod devuelve un objeto, cuya propiedad *RetrunValue* indica el código retornado por el método, según se describe en la ayuda del objeto WMI/CIM.

```
Write-Host "Este script renombra directorios ubicados en C:\Temp`n"

$dir = Read-Host "Introduce el nombre del directorio a renombrar"

$dir_fullPath = "C:\\Temp\\" + $dir
# NOTA: El caracter \, por ser especial, debe precederse de otro carácter \

# Obtener directorio
$O_Directory = Get-CimInstance -Query "Select * from Win32_Directory where name = '$dir_fullPath'" `
                               -NameSpace root/cimv2

if ( $O_Directory -ne $null )
{

   $newDirName = Read-Host ("Introduce el nuevo nombre para el directorio " + $dir)

   $newDirName_fullPath = "C:\\Temp\\" + $newDirName

   $ret = Invoke-CimMethod -InputObject $O_Directory `
                           -MethodName "Rename" `
                           -Arguments @{ Filename = $newDirName_fullPath }

   if ( $ret.ReturnValue -eq 0 )
   {
      Write-Host ("El cambio de nombre se ha realizado con éxito")
   }
   else
   {
      Write-Host ("El cambio de nombre no ha podido realizarse")
   }
}
else
{
   Write-Host("El directorio indicado no existe")
}
```