Es

¿Por qué PowerShell?

Warren Frame
& Don Jones
Principal Authors

PowerShell.org

Why PowerShell? (Spanish)

The DevOps Collective, Inc.

Este libro está a la venta en
http://leanpub.com/why-powershell-spanish

Esta versión se publicó en 2018-10-28

Este es un libro de Leanpub. Leanpub anima a los autores y publicadoras con el proceso de publicación. Lean Publishing es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener feedback del lector hasta conseguir tener el libro adecuado.

También por The DevOps Collective, Inc.

Creating HTML Reports in Windows PowerShell

A Unix Person's Guide to PowerShell

The Big Book of PowerShell Error Handling

DevOps: The Ops Perspective

Ditch Excel: Making Historical and Trend Reports in PowerShell

Secrets of PowerShell Remoting

The Big Book of PowerShell Gotchas

The Monad Manifesto, Annotated

Why PowerShell?

Windows PowerShell Networking Guide

The PowerShell + DevOps Global Summit Manual for Summiteers

Secrets of PowerShell Remoting (Spanish)

DevOps: The Ops Perspective (Spanish)

The Monad Manifesto: Annotated (Spanish)

Creating HTML Reports in PowerShell (Spanish)

The Big Book of PowerShell Gotchas (Spanish)

The Big Book of PowerShell Error Handling (Spanish)

DevOps: WTF?

PowerShell.org: History of a Community

Índice general

¿Por qué PowerShell?

Escrito por Warren Frame y Don Jones

Un vistazo increíblemente conciso del porque Windows PowerShell es importante, desde una perspectiva técnica y de negocios.

¿Ha sido útil este libro? El (los) autor (es) le pide (n) que haga una donación deducible de impuestos (en los EE.UU., consulte sus leyes si vive en otro lugar) de cualquier cantidad a The DevOps Collective[1] para apoyar su trabajo.

** Revise las actualizaciones! ** Nuestros ebooks se actualizan a menudo con contenido nuevo y corregido. Los hacemos disponibles de tres maneras::

- Nuestra rama principal GitHub organization[2], con un repositorio para cada libro. Visite https://github.com/devops-collective-inc/

[1] https://devopscollective.org/donate/
[2] https://github.com/devops-collective-inc

- Nuestra GitBook page[3], donde puede navegar por los libros en línea, o descargarlos en formato PDF, EPUB o MOBI. Utilizando el lector en línea, puede saltar a capítulos específicos. Visite https://www.gitbook.com/@devopscollective
- En LeanPub[4], donde se pueden descargar como PDF, EPUB, o MOBI (login requerido), y "comprar" los libros haciendo una donación a DevOps. También puede elegir recibir notificaciones de actualizaciones. Visite https://leanpub.com/u/devopscollective

GitBook y LeanPub generan la salida del formato PDF ligeramente diferente, por lo que puede elegir el que prefiera. LeanPub también le puede notificar cada vez que liberamos alguna actualización. Nuestro repositorio de GitHub es el principal; los repositorios en otros sitios suelen ser sólo espejos utilizados para el proceso de publicación. GitBook normalmente contendrá nuestra última versión, incluyendo algunos bits no terminados; LeanPub siempre contiene la más reciente "publicación liberada" de cualquier libro.

[3] https://www.gitbook.com/@devopscollective
[4] https://leanpub.com/u/devopscollective

Una breve reseña

Los entusiastas de PowerShell a menudo se encuentran explicando por qué alguien con responsabilidades de TI debe aprender PowerShell. Decidimos escribir esto como referencia para el futuro.

No discutiremos porque PowerShell por encima de otros lenguajes de Microsoft, como VBScript o archivos de lotes (batch), o lenguajes de uso general como Python o Perl. Hay un lugar para todos estos lenguajes, pero si trabajas en entornos Microsoft, PowerShell es un lenguaje importante para aprender.

También es importante entender que Microsoft tiene un gran compromiso con PowerShell. Llego para quedarse, y de hecho la empresa está construyendo más y más de sus soluciones de gestión utilizándolo. Hasta cierto punto, Microsoft incluso se ha apartado un poco del desarrollo de herramientas especializadas, en favor de crear estas mismas utilizando PowerShell. Eso es algo muy significativo.

Sigamos adelante.

¿Por qué el Scripting? ¿Por qué un Shell?

Antes de sumergirnos en PowerShell, abordemos la importancia del scripting y de la automatización, una faceta integral de PowerShell.

Probablemente usted conozca este XKCD comic[5] o algo similar para justificar el Scripting. Mientras que el ahorro de tiempo es sin duda un factor importante del scripting y la automatización, no es la única justificación.

Otros factores a considerar:

- **Consistencia**. Una solución de secuencias de comandos ejecutará siempre el mismo script de manera exacta. Sin riesgo de errores tipográficos, o de olvidarse completar la tarea, o haciendo la tarea incorrectamente. *Reducir el error humano.*
- **Registros de auditoría**. Hay muchas tareas en las que sería útil tener un registro de auditoría, tal vez incluyendo qué tarea se realizó, resultados importantes, errores que ocurrieron, cuándo se ejecutó la tarea, quién la ejecutó y así sucesivamente. Los scripts pueden proporcionar estos registros, y en PowerShell v5 y posteriores, el propio Shell cuenta con amplias capacidades de registro.
- **Código modular**. Al principio usted podría pasar más tiempo escribiendo una función particular de lo que justifique el ahorro de tiempo, pero generalmente usted podrá reutilizar o tomar prestadas ideas de ese mismo código más adelante.
- **Documentación**. ¿Se tiene documentación para la tarea? ¿Está actualizada? Un script bien escrito y comentado puede servir como una base útil de documentación, que podría no

[5] http://xkcd.com/1205/

existir para una tarea manual. En algunos casos, el script puede documentar el proceso que automatiza, ayudando a preservar el conocimiento institucional.

- **Educación**. Los administradores que pueden automatizar las tareas, al final se vuelven expertos en tecnologías como resultado de sus conocimientos. Eso los convierte en mejores planificadores, arquitectos, solucionadores de problemas y operadores, lo cual resulta beneficioso para la organización.
- **Delegación**. Con una solución basada en scripts, típicamente se pueden delegar más funciones a los equipos mejor preparados. Con PowersShell v3 y posteriores, estos scripts permiten la delegación de tareas de forma extremadamente granular, ayudando al equipo a ser más eficiente y sensible.

La moraleja de la historia es que el Scripting y la automatización son importantes, y se convierten en factores importantes y de alto valor para aprender PowerShell.

¿Por qué PowerShell?

Microsoft describe PowerShell como "un shell de línea de comandos basado en tareas con un lenguaje de scripting … construido en .NET Framework". ¿Qué es lo grandioso de PowerShell? ¿Por qué debería usarlo?

PowerShell es un shell de línea de comandos y un lenguaje de secuencias de comandos (ambos)

"Apague incendios" rápidamente utilizando comandos existentes o personalizados de PowerShell, sin necesidad de compilar el código. Desarrolle sus soluciones desde la línea de comandos. Escriba scripts rápidamente, los usará muchas veces. Cree scripts legibles y documentados, capaces de ejecutarse en ambientes de producción. Le ayudaran a mantener sus servicios/ambientes durante años.

¿Cuál es el costo de esta inversión? Aprender PowerShell. Bastante razonable teniendo en cuenta que es probable que tenga que hacerlo independientemente de su lenguaje actual de programación, suponiendo que trabaja con el ecosistema de Microsoft.

PowerShell puede interactuar con un sin número de tecnologías

.NET Framework, registro de Windows, COM, WMI, ADSI. Exchange, Sharepoint, System Center, Hyper-V, SQL. VMware vCenter, Cisco UCS, Citrix XenApp y XenDesktop, REST APIs, XML,

CSV, JSON, sitios Web, Excel, aplicaciones Office. C # y otros lenguajes, DLLs y otros binarios, incluyendo Linux o herramientas Unix. Un lenguaje que puede trabajar con estas diversas tecnologías además de integrarlas puede ser increíblemente valioso.

Windows no está basado en texto. Tarde o temprano tendrá que hacer algo que no puede hacer con las herramientas de NIX y otros lenguajes basados en texto. Muchas de las tecnologías con las que PowerShell puede interactuar simplemente no tienen interfaces basadas en texto, y ni siquiera pueden ser accesibles directamente desde lenguajes más formales como Perl o Python.

El mensaje aquí es que PowerShell es el mejor "pegamento" que Microsoft nos ha proporcionado para unir sistemas diversos. Es mejor que las anteriores Shells basadas en Windows porque entiende y funciona con la naturaleza del API de Windows en sí, lo cual es muy diferente de lo que hicieron los Shells anteriores basados en texto.

PowerShell está basado en objetos

Esto nos da una flexibilidad increíble. Filtrar, clasificar, medir, agrupar, comparar o tomar otras acciones sobre objetos a medida que pasan a través de la tubería (pipeline). Trabaje con propiedades y métodos en lugar de texto en bruto.

Si ha pasado tiempo programando y "descifrando" una salida basada en texto, sabe lo frustrante que puede ser. ¿Cuál delimitador utilizo? ¿Hay un delimitador? ¿Qué pasa si un resultado en particular tiene una entrada en blanco para una columna? ¿Necesito contar los caracteres en cada columna? ¿Este conteo variará dependiendo de la salida? Con objetos todo eso está resuelto, y hace que sea muy sencillo encadenar comandos y datos a través de diversas tecnologías.

PowerShell llegó para quedarse

Microsoft está poniendo todo su empeño detrás de PowerShell.

El soporte de PowerShell es un requisito en los criterios comunes de ingeniería de Microsoft, y un producto de servidor no se puede liberar sin una interfaz de PowerShell. Eso significa que muy pocos productos de servidor de Microsoft no pueden ser manejados desde PowerShell - y los pocos que no pueden, podrán hacerlo pronto.

Otros proveedores de tecnologías Microsoft también tienen soporte para PowerShell. Esto incluye IBM, Cisco, Citrix, VMware, NetApp, Dell, y docenas más.

En muchos casos Microsoft utiliza PowerShell para construir las consolas de administración GUI para sus productos. Algunas tareas no se pueden realizar en el GUI y sólo se pueden completar desde PowerShell. Esto podría ser un problema: *en un número cada vez mayor de situaciones, no se puede administrar el producto totalmente a menos que utilice PowerShell.* Y esto se extiende a ofertas basadas en la nube como Azure y Office 365, también.

Consolidar y multiplicar su aprendizaje

Su recompensa por aprender PowerShell es una habilidad mejorada para controlar y automatizar las muchas tecnologías con las que se integra. Puede usar el mismo conjunto de comandos para filtrar, exportar, redireccionar, modificar, extender y realizar acciones contra la salida para todas estas tecnologías. Puede aprovechar esas mismas habilidades de PowerShell y llevarlas en cualquier dirección que necesite: Hyper-V, vCenter, SQL, AD, XenApp y más.

Su recompensa por aprender herramientas específicas o ejecutables como net.exe o schtasks.exe, es la capacidad de trabajar con esas

herramientas. Ni más, ni menos. En cambio, con PowerShell su inversión en el aprendizaje se convierte en un enorme ecosistema de múltiples capacidades.

En Windows, PowerShell realmente es la única opción

VBScript está obsoleto, y su desarrollo se detuvo hace mucho. VBScript ya era anémico en términos de las cosas que podría "tocar", lo que lo convierte en un "pegamento" pobre para conectar sistemas y procesos.

Y nada más se acercó a las capacidades de VBScript. Python, Perl, <ponga el nombre que desee aquí> - son grandes en Linux, predominantemente basados en sistemas operativos orientados a texto, pero poco útiles en Windows, ya que no podían acceder a los muchos y variados usos de las APIs de Windows para su gestión.

PowerShell consolida todos esos APIs en una única interfaz, en gran medida coherente, que se centra en las operaciones de sistemas y su administración.

La historia de negocio

Si usted ha ignorado PowerShell hasta ahora o estaba escéptico al respecto, vamos a ver lo que Microsoft ha hecho.

En la **versión 1**, PowerShell surgió como la primera interfaz de gestión diseñada específicamente para la automatización administrativa.

En la **versión 2**, PowerShell ganó capacidades nativas de administración remota, permitiendo la administración remota de cualquier servidor o cliente que ejecute PowerShell. El "alcance" de PowerShell se extendió a cientos de APIs de administración, lo que permitió una gestión del mundo real. El producto también maduró en un lenguaje de escritura increíblemente simple y potente que se puede utilizar para construir scripts profesionales de automatización.

En la **versión 3**, PowerShell mejoro al permitir ejecutar tareas de larga duración de manera desconectada, apátrida - llamada flujos de trabajo *(Workflows)*. El alcance del producto se extendió aún más, cubriendo todas las principales plataformas de servidores de Microsoft, apoyando las ofertas de Microsoft de la nube. Para esta versión, PowerShell era una cosa muy real, tanto es así que muchos GUIs de Microsoft comenzaron a usar PowerShell "bajo el capó".

En la **versión 4**, PowerShell extendió su "alcance" al integrarse con una nueva tecnología: La configuración de estado deseada (Desired State Configuration). DSC permite a los administradores describir en un archivo de texto, cómo debe configurarse un equipo. Aprovechando la inversión existente en PowerShell, DSC pone la máquina en un estado configurado y la mantiene allí.

En la **versión 5**, PowerShell maduró DSC y amplió sus capacidades de "fabricación de herramientas" en el espacio de desarrollador profesional. Con soporte en Visual Studio, DSC comenzó a abarcar

un espectro mucho más amplio de usuarios, desde administradores de nivel básico hasta desarrolladores avanzados.

El punto es que Microsoft ha estado ampliando las capacidades de PowerShell claramente desde el lanzamiento de la versión 1 en 2006. Lo ha hecho *como nunca antes* en lenguajes como VBScript, y lo ha hecho manteniendo su consistencia y eficiencia.

Adicional a todo esto, PowerShell ha inspirado todo un ecosistema de proveedores de soporte, y una comunidad global bastante entusiasta. Los administradores hoy son, más que nunca, capaces de obtener asistencia, respuestas e incluso soluciones producidas por vendedores y por la propia comunidad.

¿Dónde se puede aprender más?

Hay una gran cantidad de información sobre PowerShell.

- Empiece en powershell.org[6], un sitio web propiedad de la comunidad que alberga un foro bastante popular de Q&A. La organización también ofrece numerosos libros electrónicos gratuitos[7], dirige el evento anual PowerShell Summit[8] en América del Norte y Europa, además de un repositorio de DSC en GitHub, También organiza un concurso anual de juegos de scripting y muchas cosas más.
- Cualquier persona con experiencia en desarrollo o secuencias de comandos debería leer PowerShell in Action v2[9]. Está escrito por el co-diseñador y autor principal de PowerShell, Bruce Payette, y es la referencia estándar. Proporciona la mejor narración que obtendrá de artículos cortos pero detallados en la Web, y da una idea de algunas de las decisiones de diseño detrás del lenguaje. Es perfectamente aplicable hoy a pesar de estar escrito para PowerShell v2. Windows PowerShell for Developers[10] es una buena lectura para los más experimentados.
- Aquellos que no tienen experiencia en secuencias de comandos o desarrollo pueden querer empezar con una lectura más ligera, como Learn Windows PowerShell in a Month of Lunches[11].

[6] http://powershell.org
[7] http://powershell.org/wp/ebooks
[8] http://powershellsummit.org
[9] http://www.manning.com/payette2/
[10] http://shop.oreilly.com/product/0636920024491.do
[11] http://manning.com/jones3/

- ¿Quiere aprender sobre la marcha? PowerShell incluye todo lo que se necesita para aprender directamente desde el shell. Get-Command, Get-Help, Get-Member, y Select-Object le ayudarán a explorar y aprender PowerShell.
- ¿Prefiere videos? Los inventores del producto Jeffrey Snover y Jason Helmick organizaron dos sesiones gratuitas de PowerShell que han demostrado ser muy populares: Getting Started with PowerShell 3.0[12] y Advanced Tools and Scripting with PowerShell 3.0[13]. O echa un vistazo al canal de YouTube de PowerShell.org[14], donde se destacan vídeos técnicos y registros de sesión de cada evento PowerShell Summit.

[12] http://channel9.msdn.com/Series/GetStartedPowerShell3
[13] http://channel9.msdn.com/Series/advpowershell3
[14] http://youtube.com/powershellorg

¿Por qué PowerShell Remoting? (Mientras respondemos otros "por qué"...)

Una pregunta común es, "¿por qué deberíamos habilitar PowerShell Remoting?"

Antes de responder hay que entender un par de cosas - va a parecer un poco grosero. Lo siento.

- PowerShell Remoting ha existido desde 2008. Si usted se está preguntando esto hasta ahora, entonces usted está haciendo un trabajo "modesto" en la gestión de su entorno. En 2008 también aparecieron los relojes inteligentes, Microsoft (antes Windows) Azure, el Roadster de Tesla, "Frozen" de Disney, las bombillas led con precios razonables, y los modelos de iPhone 3G, 3GS, 4, 4S, 5, 5S, y 6. Sólo en caso de que se los haya perdido también.
- Las tecnologías de la información son una industria en constante cambio. Las decisiones perfectamente razonables que tomó en 2003 van a necesitar ser revisadas periódicamente, debido al cambio mencionado anteriormente.

Una vez aclarado esto, vamos a hablar brevemente de ...

¿Qué es PowerShell Remoting?

Remoting es simplemente una manera en que las herramientas de administración en una computadora se hablan con servicios en otra computadora, de modo que pueda administrar remotamente esos servicios.

Remoting se basa en http, y utiliza un protocolo llamado WS-Management (WS-MAN). Requiere que los servidores tengan un único puerto abierto para enrutar todo el tráfico de gestión entrante a través de ese puerto. El tráfico de WS-MAN puede ser registrado, puede ser enviado a través de un Proxy, a través de servidores de seguridad (proporcionados por terceros), y puede ser completamente cifrado por medio de SSL.

Como todo el tráfico es http, Remoting es esencialmente un transmisor de texto de ida y vuelta. Remoting simplemente especifica la forma de intercambiar un texto (generalmente una variante de XML) para que las herramientas y servicios puedan entenderse mutuamente.

El control remoto no afecta en modo alguno la seguridad de la red ni las configuraciones predeterminadas. Por defecto, no se transmiten nombres de usuario o contraseñas, estén cifrados o no. En un entorno que no sea de dominio, puede crear una variante en el que se transmitirán contraseñas, pero si usted quisiera cifrarlas por SSL, puede utilizar https.

Remoting no le otorga a nadie privilegios especiales. La tecnología literalmente no permite que alguien haga algo que no tiene permiso para hacer, a menos que haya pasado por una configuración bastante compleja conocida como administración delegada. En ese caso, puede habilitar a usuarios específicos para realizar tareas específicas que normalmente no podrían hacerlo, pero sólo a través del canal específico y la interfaz que ha configurado.

Comparando Remoting con sus antecesores

Antes de Remoting, la mayoría de la administración remota de Windows se gestionaba a través de llamadas remotas de procedimiento (Remote Procedure Calls), o RPCs. Estas llamadas solían utilizar un puerto de hopping, por lo que era increíblemente difícil su gestión a través de un firewall. En contraste, con PowerShell Remoting usted solo tiene que ocuparse de un único puerto.

Muchas organizaciones hoy en día simplemente instalan herramientas de administración en servidores y luego usan el escritorio remoto para "administrar de forma remota".En realidad es una muy mala idea, y es una de las principales razones por las que un servidor de Windows requiere tantos parches, tantos reinicios, y está expuesto a otras tantas cosas. El código necesario para soportar un GUI completo, y por lo tanto RDP, es enorme - y eso significa que los parches son inevitables. Ejecutar herramientas de administración *en el servidor*, normalmente bajo credenciales de administrador, deja abierta la puerta a un ataque.

En pocas palabras, la mayoría de las organizaciones parecen estar preocupadas por las "implicaciones" de seguridad de la conexión remota de PowerShell. Las implicaciones son **significativas**: la comunicación remota es **mucho más segura** de lo que usted ha estado haciendo. También es más eficiente, impone menos sobrecarga en los servidores y permite que los estos funcionen con un sistema operativo más "delgado" que requiera menos parches, menos reinicios y menos Service Packs. Esos servidores también arrancarán más rápido, ejecutarán menos procesos, ejecutarán menos servicios y consumirán menos espacio de almacenamiento para el sistema operativo. También, esos servidores demandarán menos sobrecarga de memoria para el sistema operativo, lo que significa que usted podrá empaquetar más de ellos en un host de virtualización. Suena horrible, ¿verdad?

Pero esta es la mejor razón

Simplemente, la principal razón para habilitar Remoting es que usted no tiene ninguna otra opción. Microsoft se ha movido firmemente en esta dirección, y ahora Remoting viene habilitado por defecto en Windows Server 2012 y posteriores. La compañía también *está desactivando* el escritorio remoto de forma predeterminada, lo que debería indicarle algo.

Además, en los servidores Nano, *el registro en la consola no es ni siquiera una opción*, ya sea que utilice el escritorio remoto o no. No hay ningún inicio de sesión local. Remoting es *literalmente su única opción* para administrar el servidor. Y así será en el futuro.

Ejecutar un entorno de Windows sin habilitar Remoting - al menos en los servidores - es como conducir un coche sin querer presionar el acelerador. No es muy divertido, y no vas a llegar muy lejos.