HANSER

Leseprobe

Holger Schwichtenberg

Windows PowerShell 5.0

Das Praxisbuch

ISBN (Buch): 978-3-446-44643-4

ISBN (E-Book): 978-3-446-44815-5

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-44643-4

sowie im Buchhandel.

Inhalt

Vorwort zur fünften Auflage

Liebe Leserin, lieber Leser,

willkommen zur fünften Auflage dieses PowerShell-Buchs! Das vor Ihnen liegende Buch behandelt die Windows-PowerShell-Version 5.0 von Microsoft sowie ergänzende Werkzeuge von Microsoft und Drittanbietern (z. B. PowerShell Community Extensions). Das Buch ist aber auch geeignet, wenn Sie noch PowerShell 2.0, 3.0 oder 4.0 einsetzen. Welche Funktionen neu hinzugekommen sind, wird jeweils erwähnt.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 43 Jahre alt und habe im Fachgebiet Wirtschaftsinformatik promoviert. Ich lebe (in Essen, im Herzen des Ruhrgebiets) davon, dass mein Team und ich im Rahmen unserer Firma *www.IT-Visions.de* anderen Unternehmen bei der Entwicklung von .NET- und PowerShell-Anwendungen beratend und schulend zur Seite stehen. Zudem entwickeln wir im Rahmen der 5Minds IT-Solutions GmbH & Co. KG Software (*www.5Minds.de*) im Auftrag von Unternehmen.

Es ist mein Hobby und „Nebenberuf", IT-Fachbücher zu schreiben. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 63. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Mountain Biking, Lauf-Sport, Fotografie und Reisen.

Natürlich verstehe ich das Bücherschreiben auch als Werbung für die Arbeit unserer Unternehmen und wir hoffen, dass der ein oder andere von Ihnen uns beauftragen wird, Sie durch Beratung, Schulung und Auftragsentwicklung zu unterstützen.

Wer sind Sie?

Damit Sie den optimalen Nutzen aus diesem Buch ziehen können, möchte ich – so genau es mir möglich ist – beschreiben, an wen sich dieses Buch richtet. Hierzu habe ich einen Fragebogen ausgearbeitet, mit dem Sie schnell erkennen können, ob das Buch für Sie geeignet ist.

Sind Sie Systemadministrator in einem Windows-Netzwerk?	○ Ja ○ Nein
Laufen die für Sie relevanten Computer mit den von PowerShell 3.0, 4.0 oder 5.0 unterstützten Betriebssystemen? (Windows 7/8/8.1/10, Windows Server 2008/2008 R2/2012/2012 R2/2016)	○ Ja ○ Nein
Sie besitzen zumindest rudimentäre Grundkenntnisse im Bereich des (objektorientierten) Programmierens?	○ Ja ○ Nein
Wünschen Sie einen kompakten Überblick über die Architektur, Konzepte und Anwendungsfälle der PowerShell?	○ Ja ○ Nein
Sie können auf Schritt-für-Schritt-Anleitungen verzichten?	○ Ja ○ Nein
Sie können auf formale Syntaxbeschreibungen verzichten und lernen lieber an aussagekräftigen Beispielen?	○ Ja ○ Nein
Sie erwarten nicht, dass in diesem Buch **alle** Möglichkeiten der PowerShell detailliert beschrieben werden?	○ Ja ○ Nein
Sind Sie, nachdem Sie ein Grundverständnis durch dieses Buch gewonnen haben, bereit, Detailfragen in der Dokumentation der PowerShell, von .NET und WMI nachzuschlagen, da das Buch auf 1000 Seiten nicht alle Details erläutern kann?	○ Ja ○ Nein

Wenn Sie alle obigen Fragen mit „Ja“ beantwortet haben, ist das Buch richtig für Sie. In anderen Fällen sollten Sie sich erst mit einführender Literatur beschäftigen.

Was ist neu in diesem Buch?

Gegenüber der vorherigen Auflage zur PowerShell 4.0 wurde das Buch um die neuen Funktionen in PowerShell 5.0 erweitert und inhaltlich optimiert. Zudem wurden die bestehenden Inhalte des Buchs an einigen Stellen erweitert und ich habe Praxisbeispiele aus meinen Arbeiten für Kunden und an unserer eigenen IT-Infrastruktur aus den letzten Jahren hinzugefügt.

Sind in diesem Buch alle Features der PowerShell beschrieben?

Die PowerShell umfasst mittlerweile über 1400 Commandlets mit jeweils zahlreichen Optionen. Zudem gibt es unzählige Erweiterungen mit vielen hundert weiteren Commandlets. Zudem existieren zahlreiche Zusatzwerkzeuge. Es ist allein schon aufgrund der Vorgaben des Verlags für den Umfang des Buchs nicht möglich, alle Commandlets hier auch nur zu erwähnen. Zudem habe ich – obwohl ich selbst fast jede Woche mit der PowerShell in der Praxis arbeite – immer noch nicht alle Commandlets jemals eingesetzt. Ich beschreibe in diesem Buch, was ich selbst in der Praxis verwende.

Wem ist zu danken?

Folgenden Personen möchte ich meinen Dank für ihre Mitwirkung an diesem Buch aussprechen:

- meinem Kollegen und Freund Peter Monadjemi, der rund 100 Seiten mit Beispielen zu der Vor-Vor-Auflage dieses Buchs beigetragen hat (Themen: Workflows, Bitlocker, ODBC, Hyper-V, DNS-Client, Firewall und SQL-Server-Administration),
- Frau Bauer-Schiewek und Frau Hasselbach, die dieses Buchprojekt beim Carl Hanser Verlag koordinieren und vermarkten,
- Frau Petra Kienle, die meine Tippfehler gefunden und sprachliche Ungenauigkeiten eliminiert hat,
- meiner Frau und meinen Kindern dafür, dass sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

Woher bekommen Sie die Beispiele aus diesem Buch?

Unter *http://www.powershell-doktor.de/leser* biete ich ein **ehrenamtlich betriebenes** Webportal für Leser meiner Bücher an. In diesem Portal können Sie

- die Codebeispiele aus diesem Buch herunterladen,
- eine PowerShell-Kurzreferenz „Cheat Sheet“ (zwei DIN-A4-Seiten als Hilfe für die tägliche Arbeit) kostenlos herunterladen,
- Feedback zu diesem Buch geben (Bewertung abgeben und Fehler melden) und
- technische Fragen in einem Webforum stellen.

Alle registrierten Leser erhalten auch Einladungen zu kostenlosen Community-Veranstaltungen sowie Vergünstigungen bei unseren öffentlichen Seminaren zu .NET und zur PowerShell. Bei der Registrierung müssen Sie das Kennwort **The Man in the High Castle** angeben.

Wo kann ich mich schulen oder beraten lassen?

Unter der E-Mail-Adresse *buero@IT-Visions.de* stehen mein Team und ich für Anfragen bezüglich Schulung, Beratung und Entwicklungstätigkeiten zur Verfügung.

Zum Schluss des Vorworts ...

wünsche ich Ihnen viel Spaß und Erfolg mit der PowerShell 5.0!

Dr. Holger Schwichtenberg

Essen, im Januar 2016

Über den Autor Dr. Holger Schwichtenberg

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Leiter des Berater- und Dozententeams bei *www.IT-Visions.de*

www.IT-Visions.de®
Dr. Holger Schwichtenberg

- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5minds IT-Solutions GmbH & Co. KG (*www.5minds.de*)

5Minds
IT-SOLUTIONS

- Über 60 Fachbücher beim Carl Hanser Verlag, bei Microsoft Press und Addison-Wesley sowie mehr als 850 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006–2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal *heise.de* (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z. B. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET Cologne, VS One, NRW.Conf, Net.Object Days, Windows Forum)
- Zertifikate und Auszeichnungen von Microsoft:
  - Microsoft Most Valuable Professional (MVP)
  - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Microsoft .NET Framework, Visual Studio, C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Technologien

- Einführung von .NET Framework und Visual Studio/Migration auf .NET
- Webanwendungsentwicklung mit IIS, ASP.NET und JavaScript/AJAX
- Enterprise .NET, verteilte Systeme/Webservices mit .NET
- Relationale Datenbanken, XML, Datenzugriffsstrategien
- Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework
- Windows PowerShell (WPS) und Windows Management Instrumentation (WMI)

- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA)
  - Betrieb diverser Community-Websites: *www.dotnetframework.de*, *www.entwicklerlexikon.de*, *www.windows-scripting.de*, *www.aspnetdev.de* u. a.
- Firmenwebsites: *http://www.IT-Visions.de* und *http://www.5minds.de*
- Weblog: *http://www.dotnet-doktor.de*
- Kontakt: *buero@IT-Visions.de* sowie *Telefon 02 01-64 95 90-0*

Über den Co-Autor Peter Monadjemi

Peter Monadjemi ist freiberuflicher Trainer und Autor mit den Schwerpunkten PowerShell, Windows-Server-Administration und Softwareentwicklung auf der Grundlage des .NET Frameworks. Er kennt die PowerShell bereits seit der Zeit, in der sie als Vorabversion noch unter dem Projektnamen „Monad“ lief – ein Codename, der für ihn eine Art „Wink des Schicksals“ war. Seitdem hat er viel Zeit mit der PowerShell verbracht und Hunderte von Skripten geschrieben. Er lebt in Esslingen am Neckar.

12 Fernausführung (Remoting)

Eine der schmerzlich vermissten Funktionen in der ersten Windows PowerShell (Version 1.0) war die generelle Unterstützung für Fernzugriffe auf andere Systeme bzw. Fernverwaltung von entfernten Systemen. Mit den Bordmitteln der PowerShell 1.0 konnte man im Wesentlichen nur über WMI via Distributed COM (DCOM) Daten von anderen Systemen abrufen. Eine generelle Möglichkeit zur Fernausführung von Commandlets und Skripten gab es nicht. Hier hat Microsoft nun in der Version 2.0 deutlich nachgebessert.

Seit PowerShell 2.0 gibt es über das Protokoll WS-Management (kurz: WS-Man) die Fernausführungsmöglichkeit („PowerShell Remoting“) für einzelne Commandlets und ganze Skripte.

WS-Management ist ein Netzwerkprotokoll auf Basis von XML-Webservices unter Verwendung des Simple Object Access Protocol (SOAP). WS-Management dient dem Austausch von Verwaltungsinformationen zwischen (heterogenen) Computersystemen. WS-Management ist ein Standard der Desktop Management Task Force (DMTF), der im Jahr 2006 verabschiedet wurde.

WS-Management bietet eine enge Verbindung zum Web Based Enterprise Management (WBEM) alias Windows Management Instrumentation (WMI).

Microsoft bietet eine Implementierung von WS-Management unter dem Namen „Windows Remote Management (WinRM)“ seit Windows XP und Windows Server 2003.

In Vista und Windows Server 2008 ist die Version 1.1 der Implementierung enthalten. Windows 7 und Windows Server 2008 R2 enthalten Version 2.0. Für Windows XP, Windows Vista sowie Windows Server 2003 (inkl. R2) und Windows Server 2008 gibt es ein Add-On für WinRM v2 mit Namen „Windows Management Framework“. Seit Windows 8.x und Windows Server 2012 (inkl. R2) gibt es WinRM 3.0, das es auch als Add-On für Windows 7 und Windows Server 2008/2008 R2 gibt *http://www.microsoft.com/en-us/download/details.aspx?id=34595*.

Für einen Fernaufruf müssen sowohl der lokale (der Aufrufer, der Client) als auch der entfernte Computer (der Aufgerufene, der Server) Windows Remote Management (WinRM) ab Version 2.0 unterstützen. Außerdem muss die PowerShell (ab Version 2.0) auf beiden Systemen installiert sein. WinRM benutzt im Standard die Ports 5985 (HTTP) und 5986 (HTTPS). Die Authentifizierung erfolgt im Normalfall über Kerberos, alternativ sind auch Basisauthentifizierung, Digest und NTLM möglich.

Die Verbindung zwischen Client und Server kann permanent oder temporär sein.

Mit PowerShell Remoting ist nicht nur ein Fernaufruf eines Computers, sondern auch gleichzeitig mehrerer Computer möglich. So kann man z. B. ein Skript gleichzeitig auf mehreren entfernten Systemen starten.

12.1 RPC-Fernabfrage ohne WS-Management

Einige Commandlets in der PowerShell besitzen eingebaute Fernabfragemöglichkeiten abseits von WS-Management. Diese Commandlets haben einen Parameter „-Computername“ und die Fernaufrufmöglichkeiten des Betriebssystems, die auf einem Remote Procedure Call (RPC) basieren.

Folgende Commandlets besitzen den Parameter „-Computername“ und arbeiten mit DCOM:

- `Clear-EventLog`
- `Limit-EventLog`
- `Get-Counter`
- `New-EventLog`
- `Get-EventLog`
- `Remove-EventLog`
- `Get-HotFix`
- `Restart-Computer`
- `Get-Process`
- `Show-EventLog`
- `Get-Service`
- `Show-Service`
- `Get-WinEvent`
- `Stop-Computer`
- `Get-WmiObject` (schon in PowerShell 1.0 vorhanden)
- `Write-EventLog`

Der Fernaufruf mit vorgenannten Commandlets funktioniert auch, wenn WS-Management nicht installiert und konfiguriert ist.

TIPP: Derartige Commandlets findet man mit:

```
Get-Command | where { $_.parameters.keys -contains "ComputerName" -and
$_.parameters.keys -notcontains "Session"}
```

Der folgende Befehl ermittelt vom Computer „F170“ alle Dienste, die mit dem Buchstaben „i“ beginnen.

```
Get-Service -ComputerName F170 i*
```

Die Abfrage mehrerer Computer ist nur nacheinander durch Übergabe in der Pipeline möglich, da man bei diesen Commandlets bei `Computername` kein Array als solchen übergeben kann.

Falsch:

```
Get-Service -ComputerName F173, F170 i*
```

Richtig:

```
"F171", "F172", "F173" | % { Get-Service i* -ComputerName $_ }| ft Name, status,
-machinename
```

Über das Attribut `MachineName` kann man jeweils sehen, welcher der abgefragten Computer das Ergebnis geliefert hat.

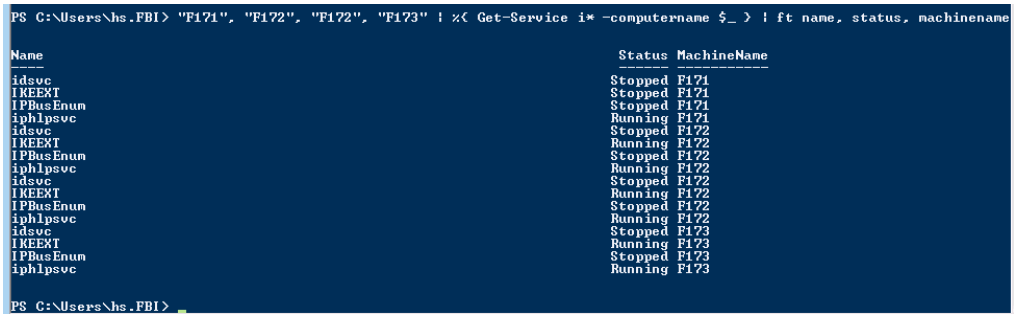

Bild 12.1 Abfrage der Dienste auf zwei Computern

12.2 Anforderungen an PowerShell Remoting

Für einen Fernaufruf via PowerShell Remoting müssen sowohl der lokale (der Aufrufer, der Client) als auch der entfernte Computer (der Aufgerufene, der Server) folgende Voraussetzungen erfüllen:

- Microsoft .NET Framework 2.0 oder höher
- Windows PowerShell 2.0 oder höher
- Windows Remote Management (WinRM) 2.0 oder höher

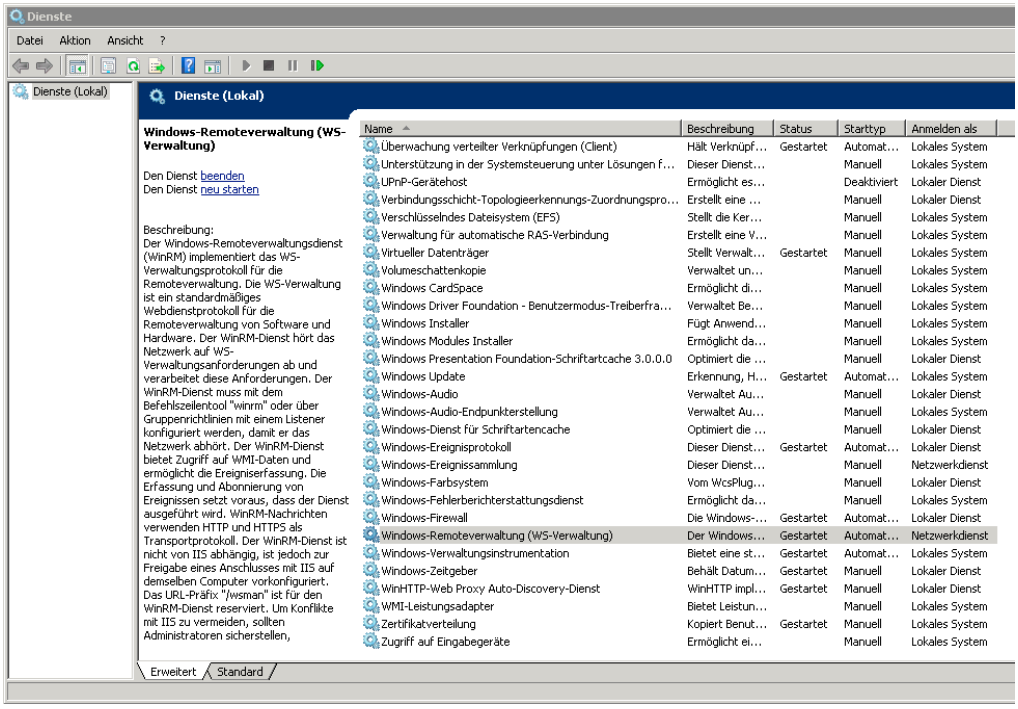

Bild 12.2 Der WinRM-Dienst in der Diensteliste

12.3 Rechte für PowerShell-Remoting

Fernaufrufe sind sowohl domänenintern als auch domänenübergreifend (durch Vertrauensstellungen oder gleiche Benutzername-Kennwort-Kombination) möglich, im zweiten Fall allerdings nur unter expliziter Angabe von Benutzernamen und Kennwort, selbst wenn auf dem Zielsystem eine zu dem aufrufenden System identische Benutzername-Kennwort-Kombination existiert. Fernaufrufe können nur Benutzer ausführen, die auf dem Zielsystem zur Administratorengruppe gehören. Man kann das Recht zum Fernaufruf durch Änderung der sogenannten Sitzungskonfigurationen steuern.

Auch auf dem lokalen Computer werden Administratorrechte für einige Aktionen im Zusammenhang mit dem Fernaufruf benötigt. Dies sind insbesondere die Einrichtung von WS-Management und die Konfiguration von PowerShell-Sitzungen. Auch ein Fernaufruf gegen den eigenen Computer („Loopback-Aufruf“) erfordert Administratorrechte.

12.4 Einrichten von PowerShell Remoting

Im Auslieferungszustand der PowerShell sind PowerShell-Fernaufrufe deaktiviert. Mit `Enable-PSRemoting` konfiguriert man einen Computer zum Empfang von Fernaufrufen von anderen Rechnern. Dieser Befehl startet den WinRM-Systemdienst (Listener), konfiguriert die Windows PowerShell und trägt das Protokoll WS-Management als Ausnahme in der Windows Firewall ein. `Enable-PSRemoting` ist nicht notwendig auf Computern, die nur (!) PowerShell-Befehle an andere Rechner senden wollen. Zum Ausführen von `Enable-PSRemoting` muss man auf dem System Administrator sein.

```
PS C:\Users\Administrator> Enable-PSRemoting

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable remote management of this computer by using the Windows Remote
Management (WinRM) service.
 This includes:
    1. Starting or restarting (if already started) the WinRM service
    2. Setting the WinRM service startup type to Automatic
    3. Creating a listener to accept requests on any IP address
    4. Enabling Windows Firewall inbound rule exceptions for WS-Management traffic (for http only).

Do you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
WinRM is already set up to receive requests on this computer.
WinRM is already set up for remote management on this computer.

Confirm
Are you sure you want to perform this action?
Performing the operation "Set-PSSessionConfiguration" on target "Name: microsoft.powershell SDDL:
O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;RM)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This lets selected users remotely run Windows
PowerShell commands on this computer.".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Confirm
Are you sure you want to perform this action?
Performing the operation "Set-PSSessionConfiguration" on target "Name: microsoft.powershell.workflow SDDL:
O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;RM)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This lets selected users remotely run Windows
PowerShell commands on this computer.".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Confirm
Are you sure you want to perform this action?
Performing the operation "Set-PSSessionConfiguration" on target "Name: microsoft.powershell32 SDDL:
O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;RM)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This lets selected users remotely run Windows
PowerShell commands on this computer.".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Confirm
Are you sure you want to perform this action?
Performing the operation "Set-PSSessionConfiguration" on target "Name: microsoft.windows.servermanagerworkflows SDDL:
O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This lets selected users remotely run Windows
PowerShell commands on this computer.".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
PS C:\Users\Administrator>
```

Bild 12.3 Erstkonfiguration von PowerShell Remoting mit Enable-PSRemoting

TIPP: Zum Unterdrücken der Nachfragen geben Sie ein:

```
Enable-PSRemoting -force
```

Zum Testen der Einrichtung geben Sie ein:

```
New-PSSession
```

Dann sollte das nachstehende Ergebnis erscheinen.

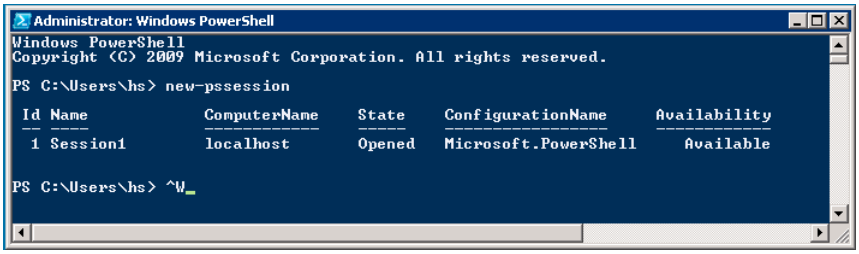

Bild 12.4 Testen der Fernverbindungskonfiguration mit New-PSSession

HINWEIS: In Domänen kann WinRM über die Gruppenrichtlinie „Computer Configuration\Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM service“ konfiguriert werden.

Ein Problem bei der Einrichtung könnte sein, dass diese mit der Fehlermeldung „Set-WSManQuickConfig : WinRM firewall exception will not work since one of the network connection types on this machine is set to Public. Change the network connection type to either Domain or Private and try again.“ abbricht. Ein Blick in das Netzwerkcenter von Windows wird dann zeigen, dass es tatsächlich ein „Public“-Netzwerk gibt (siehe Beispiel in der Abbildung).

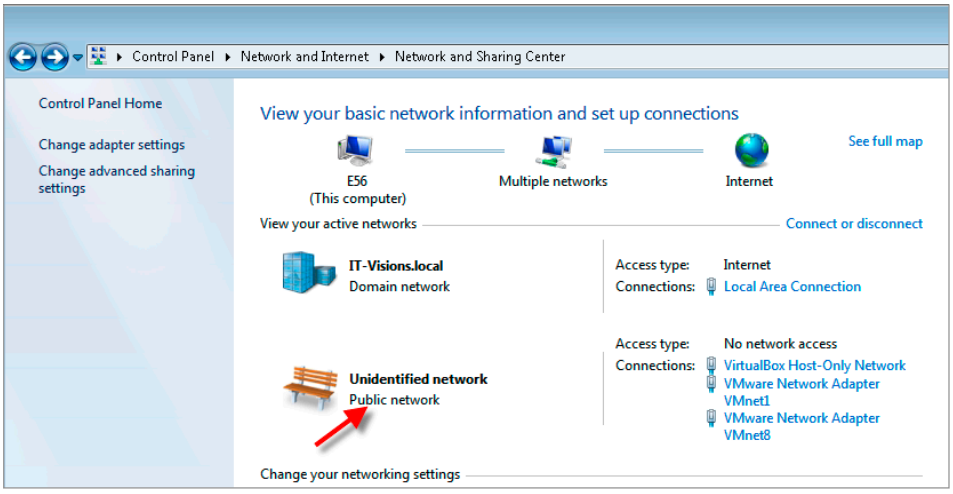

Bild 12.5 Ein „Public Network“ verhindert die Einrichtung des PowerShell Remoting.

Lösen kann man dies über secpol.msc (siehe Abbildung 12.6).

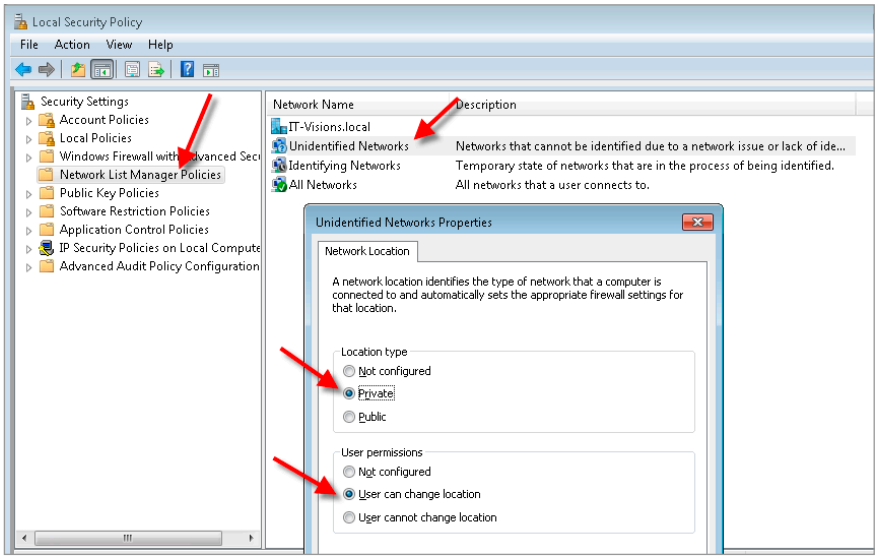

Bild 12.6 Lösung des „Public Network"-Problems

TIPP: Seit PowerShell-Version 4.0 kann man alternativ beim `Enable-PSRemoting`-Commandlet auch den Parameter `SkipNetworkProfileCheck` angeben, wodurch die Prüfung auf das Netzwerkprofil entfällt.

Zum Deaktivieren des PowerShell Remoting gibt es das Commandlet:

```
Disable-PSRemoting
```

Durch `Disable-PSRemoting` wird der WinRM-Dienst aber nicht beendet.

12.5 Überblick über die Fernausführungs-Commandlets

Die wichtigsten Commandlets für PowerShell Remoting sind:

- `Enter-PSSession`: Starten einer Fernausführungssitzung im Telnet-Stil
- `Invoke-Command`: Fernausführung eines einzelnen PowerShell-Commandlets oder eines Skripts
- `New-PSSession`: Erstellen einer permanenten Verbindung für die Fernausführung

Alle oben genannten Commandlets bieten ein Attribut `-Computername`. Bei `Enter-PSSession` kann man nur einen Computer angeben, bei `Invoke-Command` und `New-PSSession` auch ein Array mehrerer Computer.

HINWEIS: Im Standard vorgesehen ist die Angabe von Computernamen. IP-Adressen können alternativ verwendet werden, erfordern aber ein anderes Authentifizierungsverfahren, da das im Standard verwendete Kerberos keine IP-Adressen unterstützt (siehe Kapitel zur Sicherheit bei PowerShell Remoting).

12.6 Interaktive Fernverbindungen im Telnet-Stil

Mit dem Commandlet `Enter-PSSession` eröffnet man eine interaktive Sitzung zu einem entfernten System im Stil des Telnet-Protokolls. Anzugeben ist der Computername, z. B.

```
Enter-PSSession -Computername F170
```

Nach erfolgreicher Ausführung des Befehls wird der Computername vor der PowerShell-Eingabeaufforderung angezeigt. Alle eingegebenen Befehle werden nun auf dem entfernten System ausgeführt. Alle Ausgaben landen auf dem lokalen System.

Testen kann man zum Beispiel, indem man mit `[System.Environment]::MachineName` den Computernamen abruft.

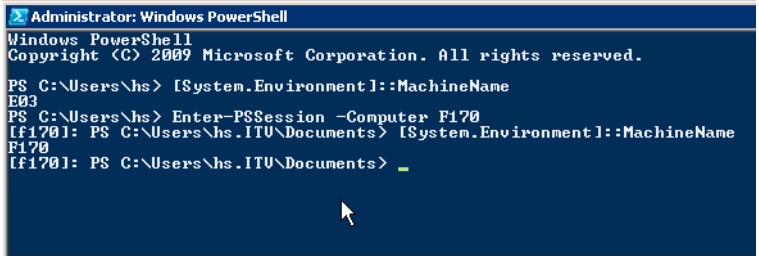

Bild 12.7 Aufbau einer interaktiven Fernsitzung

HINWEIS: Man kann sich in einer interaktiven Sitzung immer nur mit genau einem entfernten System verbinden. Man kann aber auf einem System mehrere PowerShell-Fenster öffnen und sich darin mit jeweils einem anderen entfernten System verbinden.

Möglichkeiten einer Fernsitzung

In einer Fernsitzung kann jegliche Form von Änderungen durchgeführt werden, sowohl durch Ausführung von Commandlets, z. B.

```
(Get-service BITS) | start-service
```

als auch durch den Aufruf von Methoden

```
(Get-service BITS).Start()
```

Man kann auch Windows-Prozesse starten. Zu beachten ist jedoch, dass man auf dem entfernten System Benutzeroberflächen von diesen Prozessen nicht sehen kann, selbst wenn man dort lokal angemeldet ist.

Zum Verlassen der Fernsitzung gibt man ein:

```
Exit-PSSession
```

12.7 Fernausführung von Befehlen

Um einen einzelnen Befehl auf einem entfernten System auszuführen, kann man auch das Commandlet `Invoke-Command` mit dem Parameter `-Computername` verwenden. Beim Parameter `-ScriptBlock` kann man einen oder mehrere (durch Zeilenumbruch oder Semikolon getrennte) Befehle angeben und auch Pipelines nutzen. Nicht nur Commandlets, auch klassische Kommandozeilenbefehle sind möglich.

Listing 12.1 [2_Aufbauwissen\Remoting\WPS2_Remoting_Script.ps1]

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Service b* }
Invoke-Command -ComputerName F170 -scriptblock { Get-Service | sort status | ft name,
status }
Invoke-Command -computer F170 -Scriptblock { "Computername: " + [System.
Environment]::MachineName ; "Zeit: " + [DateTime]::Now ; "Sprache: " +
(Get-Culture) }
Invoke-Command -computer F170 -Scriptblock { ping www.it-visions.de }
```

Natürlich kann man den Skriptblock vorher auch in einer Variable speichern.

Listing 12.2 [2_Aufbauwissen\Remoting\WPS2_Remoting_Script.ps1]

```
$cmd = {
"Rechnername: $([System.Environment]::MachineName)"
"Benutzername: $([System.Environment]::UserDomainname + "\" + [System.
Environment]::Username)"
 }
Invoke-Command PC199 -ScriptBlock $cmd
```

Ein Sonderfall ist gegeben, wenn der Skriptblock unterschiedlich für verschiedene Rechner sein soll. Dann muss der Skriptblock mit `[scriptblock]::Create()` erzeugt werden.

Listing 12.3 [2_Aufbauwissen\Remoting\Remoting_DynamicScriptBlock.ps1]

```
$pcListe = "D140","D141"
foreach($pc in $pcListe)
{
"Abfrage bei $pc"
```

```
$scriptblock = [scriptblock]::Create( `
'"PC ' + $pc + ' nennt sich "' + `
'+ [System.Environment]::MachineName')

"Erzeugtes Skript..."
$scriptblock
"Skript wird gesendet..."
$ergebnis = Invoke-Command -ScriptBlock $scriptblock -ComputerName $pc
$ergebnis
}
```

HINWEIS: Alle Commandlets oder Anwendungen, die in dem Skriptblock gestartet werden, müssen auf dem Zielsystem verfügbar sein.

Auch hier wird das Ergebnis auf dem lokalen System angezeigt, oft mit der zusätzlichen Spalte „PSComputerName“, die den Namen des aufgerufenen Computers enthält.

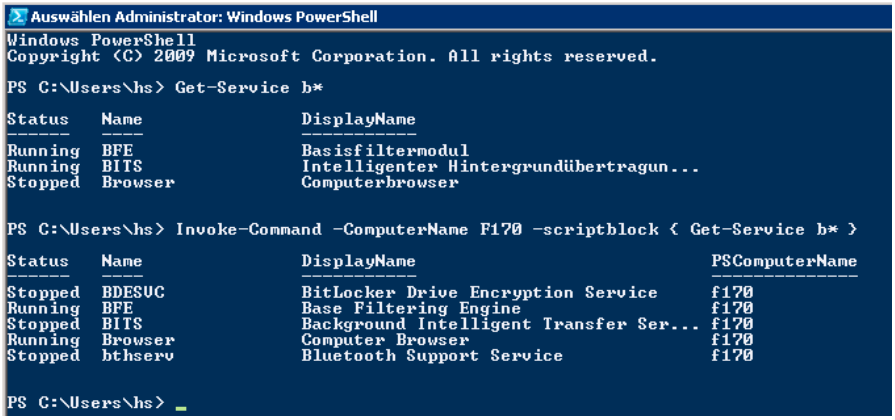

Bild 12.8 Lokaler Aufruf versus entfernter Aufruf

Allerdings muss man beachten, dass die Ergebnismenge keineswegs die gleiche Struktur wie bei einem lokalen Aufruf hat. Die Objekte in der Pipeline sind nicht vom Typ `System.ServiceProcess.ServiceController`, sondern `Deserialized.System.ServiceProcess.ServiceController`. Zwischen den Rechnergrenzen hat, für den Transport über das Netzwerk, eine Serialisierung/Deserialisierung der Objekte stattgefunden. Dabei sind die Methoden der Objekte „verloren“ gegangen. Methodenaufrufe wie

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service bits }).Start()
```

sind also nicht möglich!

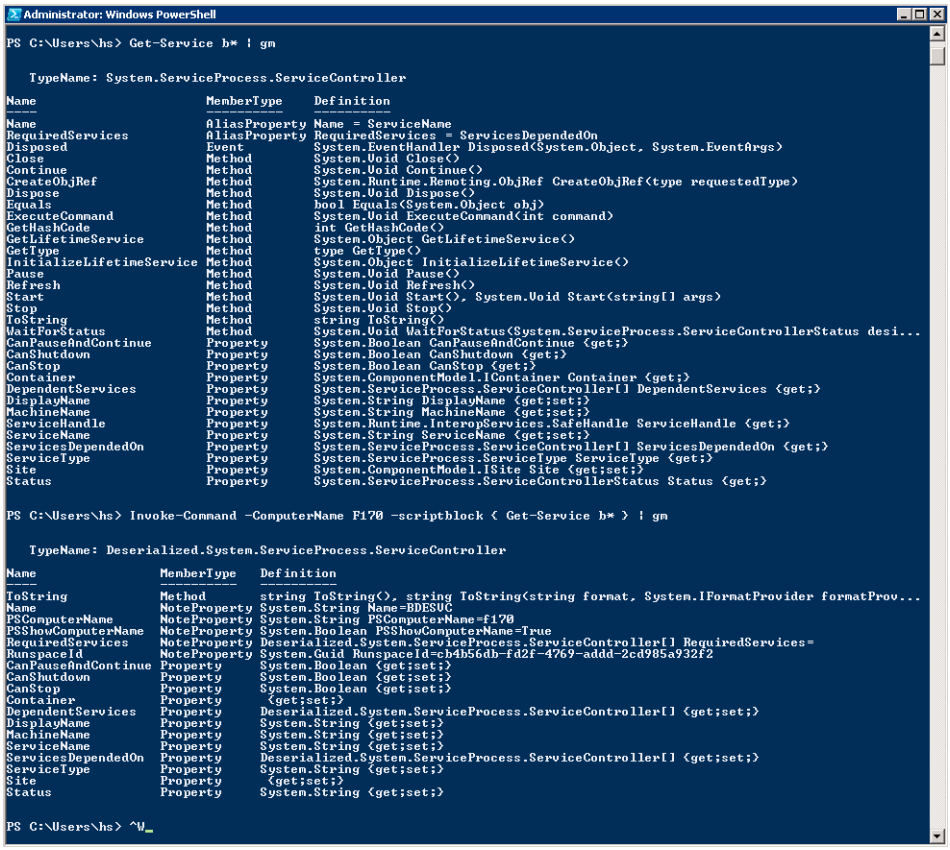

Bild 12.9 Get-Member nach einem lokalen und einem entfernten Aufruf von Get-Service

ACHTUNG: Beim Pipelining kann man in eine sehr tiefe Falle tappen. Ein Benutzer, der merkt, dass

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service Bits
}).Start()
```

nicht funktioniert, würde wohl auf

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service Bits }) |
Start-Service
```

ausweichen wollen.

Dieser Befehl würde ohne Fehlermeldung abgeschlossen – er hätte aber nicht getan, was gewünscht war. In diesem Fall würde der „BITS“-Dienst auf dem lokalen System, nicht auf dem entfernten System gestartet. Der Grund liegt darin, dass `Start-Service` das `PSComputerName`-Attribut ignoriert und nur den Namen des Dienstes berücksichtigt. Die folgende Bildschirmabbildung liefert den Beweis.

```
PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS }

Status   Name               DisplayName                            PSComputerName
------   ----               -----------                            --------------
Stopped  BITS               Background Intelligent Transfer Ser... f170

PS C:\Users\hs> Get-Service BITS

Status   Name               DisplayName
------   ----               -----------
Stopped  BITS               Intelligenter Hintergrundübertragun...

PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS } | Start-Service
PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS }

Status   Name               DisplayName                            PSComputerName
------   ----               -----------                            --------------
Stopped  BITS               Background Intelligent Transfer Ser... f170

PS C:\Users\hs> Get-Service BITS

Status   Name               DisplayName
------   ----               -----------
Running  BITS               Intelligenter Hintergrundübertragun...
```

Bild 12.10 Unerwartetes Verhalten beim Fernaufruf

Viele Commandlets funktionieren so, auch weitreichende wie `Remove-Item`. Der folgende Befehl löscht also nicht Textdateien auf dem entfernten, sondern gleichnamige Dateien auf dem lokalen System!

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Item d:\Daten\*.txt } | remove-
Item
```

Richtig ist hier, den Befehl `Remove-Item` mit in den Skriptblock zu nehmen:

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Item d:\Daten\*.txt | remove-
Item }
```

Grund für diese Falle ist, dass die von `Get-Item` gelieferten Dateiobjekte zwar von der PowerShell um den Parameter `PSComputername` angereichert wurden, aber das Commandlet `Remove-Item` diese Zusatzinformation leider ignoriert.

HINWEIS: Einige Commandlets in der PowerShell, darunter `Get-Process` und `Get-Service`, bieten auch noch einen kürzeren Weg für die Fernabfrage an. Bei diesen Commandlets kann man ein einzelnes entferntes System über den Parameter `–Computer` angeben, z. B. `Get-Process -Computer F111` (Details siehe Kapitel „Fernausführung/Fernabfrage ohne WS-Management“).

Vorteil dieser Methode ist, dass man dafür nicht WS-Management braucht, also auch ältere Betriebssysteme abfragen kann, für die es kein WS-Management gibt. Nachteil ist, dass sich die Fernabfrage immer nur auf den einzelnen Befehl bezieht. Man kann weder Befehlsfolgen noch Skripte angeben. Außerdem kann man immer nur ein einzelnes entferntes System ansprechen.

Noch ein Tipp: Sie können auch einen Skriptblock mit Zeilenumbrüchen an der Konsole erfassen. Dazu müssen Sie aber innerhalb des Skriptblocks SHIFT + EINGABE drücken. Erst wenn Sie den Skriptblock mit der schließenden geschweiften Klammer beendet haben, drücken Sie nur noch EINGABE. Wenn Sie vorzeitig nur auf EINGABE drücken, wird der Befehl sofort ausgeführt.

```
PS T:\> invoke-command -computer D142 -ScriptBlock {
>>> if (-not (test-path c:\temp)) { md c:\temp; "Temp angelegt!" }
>>> dir c:\
>>> }


    Verzeichnis: C:\


Mode                 LastWriteTime         Length Name                                 PSComputerName
----                 -------------         ------ ----                                 --------------
d-----        19.01.2016     17:16                temp                                 D142
Temp angelegt!
d-----        22.08.2013     17:52                PerfLogs                             D142
d-r---        22.12.2015     22:00                Program Files                        D142
d-----        19.01.2016     16:45                Program Files (x86)                  D142
d-----        19.01.2016     17:16                temp                                 D142
d-r---        06.12.2015     19:53                Users                                D142
d-----        22.12.2015     21:20                Windows                              D142


PS T:\>
```

Bild 12.11 Eingabe eines Skriptblocks mit Umbrüchen an der Konsole

12.8 Parameterübergabe an die Fernausführung

Oft braucht der auf dem entfernten System auszuführende Skriptblock Werte, die der aufrufende Computer besitzt. Das erste Beispiel zeigt, wie es falsch ist. `$VM` und `$Path` sind leer bei der Ausführung auf Server79.

```
$VM = "D140"
$Path = "\\D123\backup"

Invoke-Command -ComputerName Server79 -ScriptBlock {
# falsch. $VM ist leer !!!
"Sichern der VM $vm nach $Path..."
}
```

Richtig ist, den Inhalt von `$VM` und `$Path` per Parameter `-Argumentlist` an `Invoke-Command` zu übergeben und dann in dem fernauszuführenden Skriptblock darauf mit `$args[0]` und `$args[1]` zuzugreifen.

```
$VM = "D140"
$Path = "\\D123\backup"
#richtig!
Invoke-Command -ComputerName Server79 -ArgumentList $vm,$Path  -ScriptBlock {
$VM = $args[0]
$Path = $args[1]
"Sichern der VM $vm nach $Path..."
}
```

12.9 Fernausführung von Skripten

Mit `Invoke-Command` kann man natürlich ein auf dem entfernten System vorhandenes Skript starten, z. B.:

```
Invoke-Command -computer F170 -scriptblock { d:\Skripte\WPS2_Computername.ps1 }
```

Voraussetzung ist natürlich, dass auf dem entfernten System die Skriptausführung erlaubt ist und alle für das Skript benötigten Dateien dort sind.

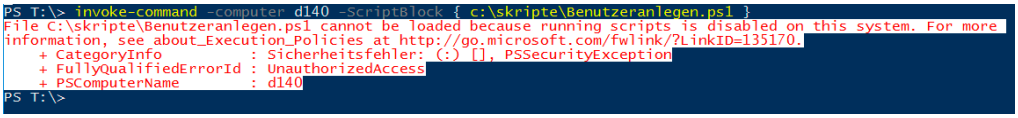

Bild 12.12 Fehlermeldung, wenn das Starten des Skripts auf einem entfernten System nicht erlaubt ist

Mit folgendem Befehl kann man die Skriptausführung auf einem entfernten System aktivieren:

```
Invoke-Command -computer F170 -scriptblock { Set-executionpolicy unrestricted }
```

Das folgende Listing zeigt ein Beispiel, bei dem man die Skriptausführung aktiviert, ein Skript kopiert, dann ausführt und anschließend das Skript löscht und die Skriptausführung wieder deaktiviert.

Listing 12.4 Entfernte Skriptausführung durch Skriptkopieren [2_Aufbauwissen/Remoting/WPS2_Remoting_Script.ps1]

```
$computer = "PC170"
$lokalesSkript = "H:\TFS\Demos\PowerShell\2_Aufbauwissen\Remoting\WPS2_Computername.
ps1"
"Start Session..."
$s = New-PSSession -ComputerName $computer

"Enable Script Execution on remote System..."
Invoke-Command -Session $s -scriptblock { Set-executionpolicy unrestricted }

"Copy Script..."
Copy-Item $lokalesSkript "\\$computer\c$\temp\wps2_Computername.ps1"

"Start Script..."
Invoke-Command -Session $s -scriptblock { c:\temp\WPS2_Computername.ps1 }

"Delete Script..."
Remove-Item "\\$computer\c$\temp\wps2_Computername.ps1"

"Disable Script Execution on remote System..."
Invoke-Command -Session $s -scriptblock { Set-executionpolicy default }

"End Session!"
Remove-PSSession $s
```

Das gleiche Ergebnis kann man aber auch viel einfacher haben, denn das Commandlet `Invoke-Command` bietet auch die Möglichkeit, ein lokales Skript auf den entfernten Computer zu übertragen und dort zu starten:

```
Invoke-Command -computer PC170 -FilePath H:\WPS2_Computername.ps1
```

12.10 Ausführung auf mehreren Computern

Das Commandlet `Invoke-Command` bietet auch die Möglichkeit, mehrere Computer in Form eines Arrays (eine durch Komma getrennte Liste) anzugeben.

Beispiel 1: Setzen von Datum und Uhrzeit auf mehreren Computern:

```
Invoke-Command -computer F170, F171, F172, F173 -Script { Set-date -date }
```

```
PS C:\> Invoke-Command -computer F170, F171, F172, F173  -Script { set-date -date "24.6.2009 15:10:00"
 -cred $cred

Mittwoch, 24. Juni 2009 15:10:00
Mittwoch, 24. Juni 2009 15:10:00
Mittwoch, 24. Juni 2009 15:10:00
Mittwoch, 24. Juni 2009 15:10:00
```

Bild 12.13 Ausführen von Set-Date auf mehreren Computern

Beispiel 2: Auslesen von c:\Temp auf mehreren Computern

```
$computer = F170, F171, F172, F173, F174
Invoke-Command -computer $computer { Get-childitem c:\temp }
```

TIPP: Für den lokalen Computer kann man „localhost“ oder „.“ verwenden.

Das Ergebnis ist eine Gesamtliste der Ergebnisse von allen genannten Computern. Die Objekte in der Menge besitzen ein zusätzliches Attribut (NoteProperty) `PSComputerName`, das den Namen des Computers zeigt, der das Objekt geliefert hat. Dadurch ist ein Filtern/Sortieren/Gruppieren auf dem aufrufenden Computer möglich.

HINWEIS: Die PowerShell fragt bei den einzelnen Computern in der Reihenfolge an, wie sie im Array angegeben sind. Die Reihenfolge der Ergebnisse ist jedoch abhängig davon, wann die Ergebnisse eintreffen.

Bild 12.14 Ergebnismenge eines Abrufs von Dienst-Objekten von drei Computern

TIPP: Bei der Ausgabe von einigen Klassen (z. B. auch bei ServiceController) wird PSComputerName automatisch ausgegeben. Dies kann man durch `-HideComputername` unterdrücken.

```
Invoke-Command -Computer F170, F171 -ScriptBlock { Get-Service I* }
-HideComputername
```

Abbrechen eines Fernbefehls

Zum Abbrechen eines entfernt ausgeführten Befehls kann man wie bei lokalen Befehlen die Tastenkombination **STRG + C** verwenden.

12.11 Sitzungen

`Invoke-Command` erzeugt im Standard eine temporäre Verbindung. Alle Definitionen (Variablen und Funktionen), die im Rahmen der Ausführung von `Invoke-Command` auf einem entfernten System erzeugt wurden, sind nach Ende des Befehls wieder ungültig.

Die Alternative ist eine permanente Verbindung (Sitzung). Eine Sitzung (engl. Session, alias „PSSession“) ist Host für die PowerShell, in der die PowerShell Befehle ausführt. Aus der Sicht von Windows ist eine Session ein Prozess.

Eine Sitzung gestaltet man durch das Laden von Snap-Ins und Modulen sowie durch die Definition von Variablen, Funktionen und Aliasen. Alle diese Einstellungen leben so lange, wie die Sitzung dauert.

Beim Start der PowerShell durch PowerShell.exe wird automatisch eine Sitzung erzeugt (Default Session). Durch Commandlets kann man weitere Sitzungen auf dem lokalen Computer oder entfernten Computern erzeugen.

HINWEIS: Alle Fernaufrufe der PowerShell erfolgen im Rahmen einer Sitzung. Die PowerShell unterscheidet temporäre Sitzungen (mit `Invoke-Command` unter Angabe eines Computernamens) und permanente Sitzungen (mit `Invoke-Command` unter Angabe eines Sitzungsobjekts, das vorher mit `New-PSSession` erzeugt wurde).

Commandlets zur Sitzungsverwaltung

Es folgt ein Überblick über die Commandlets zur Sitzungsverwaltung:

- `New-PSSession`: Erzeugen einer neuen Sitzung auf dem lokalen oder einem entfernten Computer
- `Get-PSSession`: Liste aller Sitzungen, die aus der aktuellen Sitzung heraus gestartet wurden (zeigt aber nicht Sitzungen, die andere Computer auf dem lokalen Computer geöffnet haben)
- `Remove-PSSession`: entfernt eine Session oder alle Sessions (`Remove-PSSession *`)
- `Enter-PSSession`: Start einer interaktiven Sitzung auf dem lokalen oder einem entfernten Computer
- `Exit-PSSession`: Ende einer interaktiven Sitzung
- `Disable-PSSessionConfiguration`: Sperren einer/aller Sitzungskonfigurationen
- `Enable-PSSessionConfiguration`: Entsperren einer/aller Sitzungskonfigurationen
- `Get-PSSessionConfiguration`: Auflisten der Sitzungskonfigurationen
- `Register-PSSessionConfiguration`: permanente Registrierung einer Sitzungskonfiguration
- `Set-PSSessionConfiguration`: Setzen von Eigenschaften einer Sitzungskonfiguration
- `Unregister-PSSessionConfiguration`: Löschen einer Sitzungskonfiguration

Sitzungen erstellen

Eine Sitzung erzeugt man über `New-PSSession`:

```
$s = New-PSSession  -computername F171
```

Auf diese offene Sitzung muss man im Commandlet `Invoke-Command` Bezug nehmen. Ein Computername ist dann nicht mehr erforderlich.

```
Invoke-Command -session $s -scriptblock {$p = Get-Process }
```

Man kann auch permanente Sitzungen zu mehreren Computern aufbauen, indem man bei `New-PSSession` mehrere Computer angibt:

```
$s = New-PSSession  -computername F173, D144, D145
Invoke-Command -session $s -scriptblock {Get-culture}
```

Alternativ kann man auch mehrere einzelne Sitzungen erstellen und diese bei `Invoke-Command` angeben.

```
"Sitzungen erstellen..."
$s1 = New-PSSession  -ComputerName F173
$s2 = New-PSSession  -ComputerName D144
$s3 = New-PSSession  -ComputerName D145

"Fernzugriff auf alle drei Rechner..."
Invoke-Command -Session $s1, $s2, $s3 -ScriptBlock { Get-Service spooler }
```

Die PowerShell-Profilskripte werden weder in temporären noch in permanenten Verbindungen automatisch auf dem entfernten System geladen. Bei Bedarf müssen sie explizit gestartet werden, z. B.:

```
Invoke-Command -session $s {. "$home\Documents\WindowsPowerShell\Microsoft.
PowerShell_profile.ps1"}
```

Schließen von Sitzungen

Sitzungen werden automatisch geschlossen, wenn die Sitzung (Elternsitzung) beendet wird, aus der heraus die Sitzung (Kindsitzung) gestartet wurde. Entfernte Sitzungen enden zudem automatisch, wenn entfernte Computer für vier Minuten lang nicht mehr erreichbar sind. Manuell kann man eine Sitzung mit `Remove-PSSession` schließen.

Sitzungskonfigurationen

Eine Sitzungskonfiguration legt durch zahlreiche Einstellungen fest, wer eine Sitzung aufbauen darf und welche Befehle in der Sitzung zur Verfügung stehen. Typische Einstellungen sind:

- Benutzer, die sich mit dem Computer entfernt verbinden dürfen
- Größe der Objekte, die die entfernten Benutzer übertragen dürfen
- Festlegung der verfügbaren Commandlets und Funktionen

HINWEIS: Die Konfiguration einer Sitzung ist nur möglich, wenn die PowerShell als Administrator gestartet wurde.

Die Standardkonfiguration trägt den Namen „Microsoft.PowerShell“. Auf 64-Bit-Computern gibt es zusätzlich „Microsoft.PowerShell32“. Seit Windows Server 2008 R2 gibt es außerdem „Microsoft.ServerManager“.

Die verfügbaren Konfigurationen mit zahlreichen Details zeigt:

```
Get-PSSessionconfiguration | fl
```

oder alternativ

```
dir wsman:localhost/plugin
```

Zugriffsrechte für Fernaufrufe

Im Standard können nur Administratoren Fernaufrufe ausführen. Man kann aber die Zugriffsrechteliste (Access Control List – ACL) der Sitzungskonfiguration ändern.

```
Set-PSSessionConfiguration Microsoft.PowerShell -ShowSecurityDescriptorUI
```

Etwas kurios für eine kommandozeilenbasierte Shell ist, dass sich dadurch ein Windows-Fenster öffnet, wie man es von den Zugriffsrechtelisten von Windows Explorer und der Windows-Registrierungsdatenbank kennt. Rein kommandozeilenbasiert kann man arbei-

ten, indem man im Parameter `SecurityDescriptorSDDL` eine SDDL-Zeichenkette (SDDL = Security Descriptor Definition Language) übergibt.

HINWEIS: Das Commandlet `Disable-PSSessionConfiguration` verändert die Zugriffsrechtelisten für eine oder alle vorhandenen Sitzungskonfigurationen so, dass kein Benutzer mehr Rechte für den Fernaufruf hat.

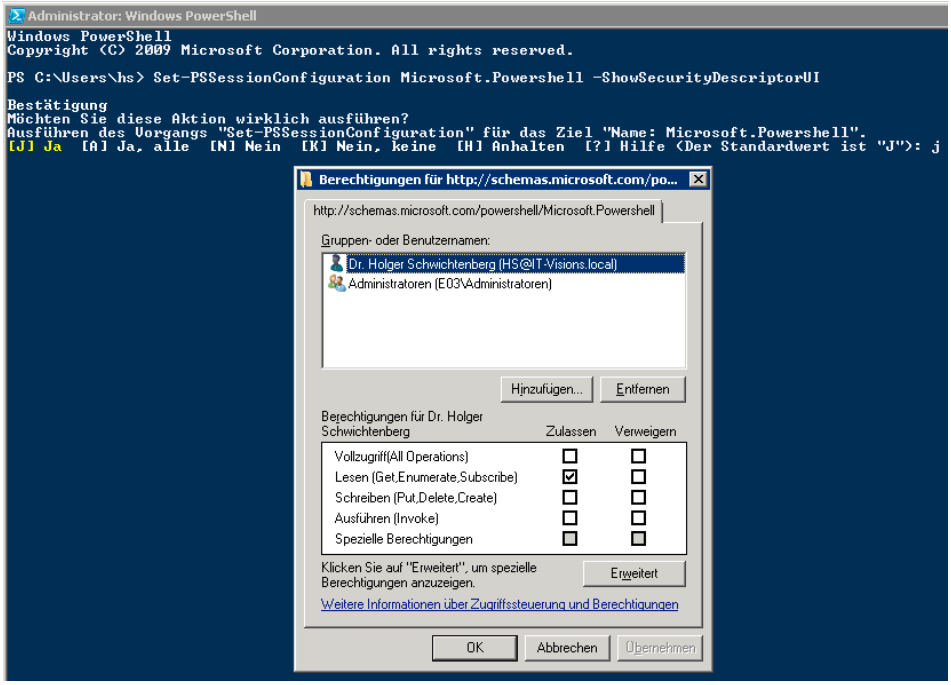

Bild 12.15 Ändern der ACL für die PowerShell-Sitzungskonfiguration

Beispiel:

```
disable-PSSessionConfiguration -name Microsoft.PowerShell
```

Die Blockade kann man rückgängig machen mit `Enable-PSSsessionConfiguration`.

Mit `Register-PSSessionConfiguration` kann man eine neue Konfiguration erzeugen. Diese wird permanent auf dem Computer gespeichert. Der erforderliche Neustart des WS-Management-Dienstes wird auf Nachfrage ausgeführt.

Der folgende Befehl legt eine neue Konfiguration unter dem Namen „FBIConfig“ an mit einer Erhöhung der Datenmenge auf 200 MB (Standard sind 50 MB) und einem Skript, das beim Starten der Sitzung ausgeführt werden soll. Das Skript muss lokal auf dem System vorhanden sein.

```
register-PSSessionConfiguration -name FBIConfig
-MaximumReceivedDataSizePerCommandMB 200
-StartupScript h:\Skripte\WPS2_Remoting_SessionStartSkript.ps1
```

Anschließend kann man eine Sitzung mit Bezug auf diese Konfiguration starten:

```
$s = New-PSSession  -ConfigurationName FBIConfig
```

Auch `Invoke-Command` hat den Parameter `-ConfigurationName`:

```
Invoke-Command -ConfigurationName "FBIConfig" -scriptblock { Start-Service BITS }
-computer F173
```

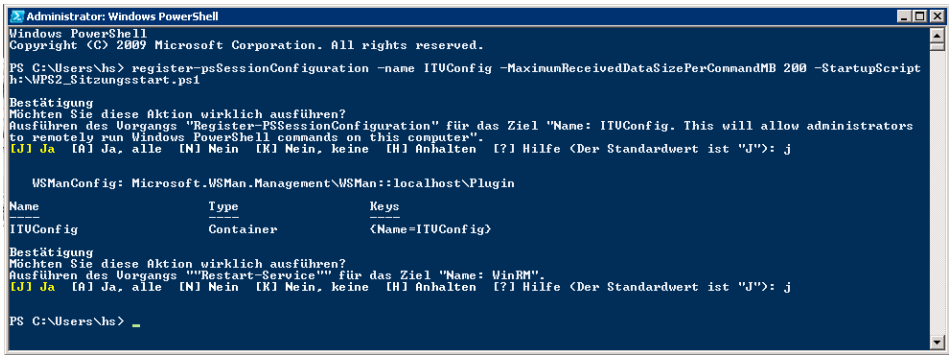

Bild 12.16 Ablauf der Registrierung einer Sitzungskonfiguration

In `$PSSessionConfigurationName` ist der Name der Konfiguration abgelegt, die verwendet wird, wenn man den Parameter `-ConfigurationName` nicht angibt. Den Inhalt dieser Variablen kann man ändern.

Zum Löschen einer Konfiguration verwendet man:

```
Unregister-PSSessionConfiguration -name FBIConfig
```

12.12 Implizites Remoting

Implizites Remoting ist eine weitere Form des PowerShell Remoting, bei der für eine entfernt auszuführende PowerShell-Funktion ein Commandlet auf dem Client als Alias verwendet wird.

Man erstellt eine PowerShell-Remoting-Session:

```
$session = New-PSSession D140
```

Man sendet dann in der Sitzung eine Funktion zu dem entfernten System. Ausgeführt wird dadurch dort nichts.

```
Invoke-Command $session -scriptblock  { function Get-ComputerName { [System.
Environment]::MachineName }}
```

Nun kann man diese Funktion auf dem entfernten System im Rahmen der Sitzung aufrufen:

```
Invoke-Command $session  {Get-ComputerName}
```

Alternativ kann man – via `Import-PSSession` – das implizite Remoting einrichten. Dadurch entsteht ein lokales Alias-Commandlet, das aber nicht lokal ausgeführt wird, sondern im Rahmen der Sitzung zum Fernsystem gesendet wird:

```
Import-PSSession -Session $session -CommandName Get-ComputerName
```

Die oben gesendete Funktion auf dem fernen System kann nun so gestartet werden, ohne Angabe der Sitzung oder des Namens des Fernsystems:

```
Get-ComputerName
```

ACHTUNG: Der Einsatz von implizitem PowerShell Remoting kann nicht zu den empfehlenswerten Vorgehensweisen („Best Practies“) zählen, weil man dem Commandlet nicht ansehen kann, wo es ausgeführt wird. Das kann zu Unklarheiten führen und zu langsamen Ausführungszeiten, die nicht direkt erkennbar sind.

12.13 Zugriff auf entfernte Computer außerhalb der eigenen Domäne

Der Zugriff auf entfernte Computer außerhalb der eigenen bzw. einer vertrauenden Domäne ist möglich.

Grundsätzlich gibt es zwei Möglichkeiten für den domänenübergreifenden Zugriff:

- Expliziter Eintrag des Zielsystems in die Liste vertrauter Systeme
- Einrichten von Secure Socket Layer (SSL) für die HTTP-Kommunikation alias HTTPS

Herleitung des Problems

Wenn man einen domänenübergreifenden Aufruf versucht (z. B. `Enter-PSSession F171`), wird man auf folgende Fehler stoßen:

> *„Die Anforderung kann von WinRM nicht verarbeitet werden. Bei Verwendung der Kerberos-Authentifizierung ist der folgende Fehler aufgetreten: Der Netzwerkpfad wurde nicht gefunden. Mögliche Ursachen:*
>
> *Der angegebene Benutzername oder das angegebene Kennwort ist ungültig. – Kerberos wird verwendet, wenn keine Authentifizierungsmethode und kein Benutzername angegeben werden.*
>
> *Kerberos akzeptiert Domänenbenutzernamen, aber keine lokalen Benutzernamen.*

Der Dienstprinzipalname (Service Principal Name, SPN) für den Remotecomputernamen und -port ist nicht vorhanden.

Der Clientcomputer und der Remotecomputer befinden sich in unterschiedlichen Domänen, zwischen denen keine Vertrauensbeziehung besteht.

Wenn Sie die oben genannten Ursachen überprüft haben, probieren Sie folgende Aktionen aus: – Suchen Sie in der Ereignisanzeige nach Ereignissen im Zusammenhang mit der Authentifizierung.

Ändern Sie die Authentifizierungsmethode; fügen Sie den Zielcomputer der Konfigurationseinstellung ‚TrustedHosts' für WinRM hinzu, oder verwenden Sie den HTTPS-Transport. Beachten Sie, dass Computer in der TrustedHosts-Liste möglicherweise nicht authentifiziert sind."

Dieser ausführliche Fehlertext liefert schon recht genaue Hinweise auf das Problem: Die Authentifizierungsmethode Kerberos funktioniert nur in Domänen. Es ist also eine andere Authentifizierungsmethode zu wählen.

Verfügbare Standardauthentifizierungsmethoden sind Default (= Kerberos), Basic, Negotiate (= Aushandlung zwischen Client und Server mit dem Simple and Protected GSSAPI Negotiation Mechanism – SPNEGA), NegotiateWithImplicitCredential, Credssp, Digest, Kerberos.

Ein neuer Versuch könnte dann also Digest oder Basic sein, also:

```
Enter-PSSession F171 -Authentication Digest
```

oder

```
Enter-PSSession F171 -Authentication Basic
```

Nun lautet der Fehlertext in beiden Fällen:

„Der WinRM-Client kann die Anforderung nicht verarbeiten. Wenn der Basic- oder Digest-Authentifizierungsmechanismus verwendet wird, müssen Anforderungen den Benutzernamen und das Kennwort enthalten. Fügen Sie den Benutzernamen oder das Kennwort hinzu, oder ändern Sie den Authentifizierungsmechanismus, und wiederholen Sie die Anforderung."

Dies bedeutet also, dass die Daten des am lokalen System angemeldeten Benutzers nicht automatisch übermittelt werden (selbst wenn es auf dem Zielsystem eine gleiche Kombination aus Benutzername und Kennwort gibt).

Nun ein dritter Versuch, wobei man Client und Server die Authentifizierung aushandeln lässt:

```
Enter-PSSession F171 -Authentication Negotiate -credential F171\hs
```

Die PowerShell zeigt den Authentifizierungsdialog …

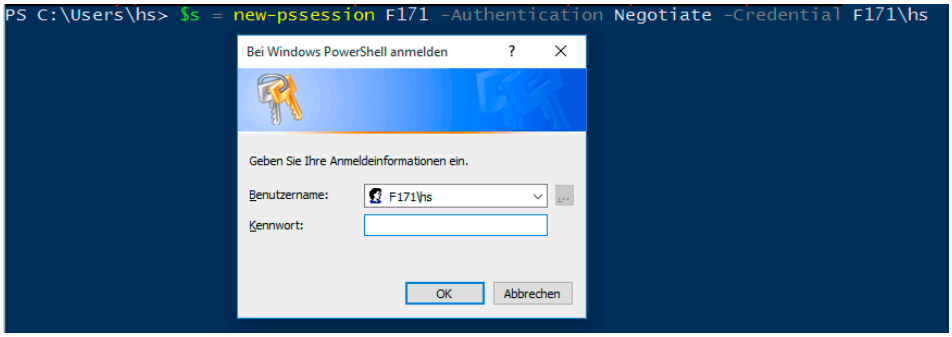

Bild 12.17 Authentifizierungsdialog

und dann wieder einen Fehler:

> *Der WinRM-Client kann die Anforderung nicht verarbeiten. Wenn das Authentifizierungsschema nicht Kerberos ist oder der Clientcomputer nicht Mitglied einer Domäne ist, muss der HTTPS-Datentransport verwendet werden, oder der Zielcomputer muss der TrustedHosts-Konfigurationseinstellung hinzugefügt werden. Verwenden Sie winrm.cmd, um TrustedHosts zu konfigurieren. Beachten Sie, dass Computer in der TrustedHosts-Liste möglicherweise nicht authentifiziert sind.*

Auch hier ist die Anweisung klar: Entweder ist HTTPS zu verwenden oder aber ein Eintrag in TrustedHosts vorzunehmen. Letzteres ist einfacher.

Eintrag in die Liste vertrauter Systeme

Diesen Eintrag kann man über den WSMan-Navigationsprovider der PowerShell recht elegant auf dem Client vornehmen:

```
cd WSMan:\localhost\Client
Set-Item trustedhosts "F170, F171, F172, F173, F174, F175" -force
Restart-Service winrm
```

Alternativ können auch IP-Adressen statt Rechnernamen angegeben werden.

Die Veränderungen der Eigenschaft „TrustedHost“ erfordern eigentlich eine Rückbestätigung, daher das `-force`. TrustedHost ist eine Liste der zu vertrauenden Rechner (Rechnername oder IP-Adresse). Das Skript muss mit Administratorrechten gestartet werden.

Ausführen von Befehlen auf entfernten Computern außerhalb der eigenen Domäne

Danach kann man dann mit einem Befehl wie folgt einen Fernzugriff ausführen, wobei die PowerShell explizit nach Benutzername und Kennwort fragen wird, selbst wenn auf dem Zielsystem die gleiche Benutzername-Kennwort-Kombination existiert.

```
Invoke-Command F171 { Get-ChildItem c:\ } -authentication negotiate -credential
F171\hs
```

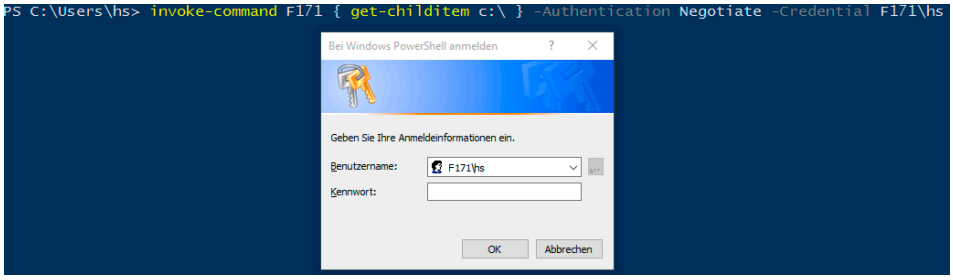

Bild 12.18 Nachfrage der PowerShell beim Zugriff auf Computer, die nicht zur (vertrauten) Domäne gehören

Man kann auch eine dauerhafte Sitzung zu einem entfernten Computer, der nicht zur Domäne gehört, erstellen:

```
$s = New-PSSession  F171 -authentication negotiate -credential F171\hs
```

TIPP: Die Anmeldedaten immer wieder eingeben zu müssen, kann man vermeiden, indem man sich die Anmeldedaten in einer Variablen merkt:

```
$cred = Get-Credential
```

Hinweis: Man muss die Anmeldedaten nicht pro Computer eingeben. Sofern Benutzername und Kennwort auf allen Systemen gleich sind, kann man auch Folgendes schreiben:

Listing 12.5 Setzen des Datums mit Set-Date auf mehreren Nicht-Domänencomputern, die die gleiche Benutzername-Kennwort-Kombination haben [WPS2_Remoting_MultiComputer_SetDate.ps1]

```
$cred = Get-Credential
$s = New-PSSession  -auth negotiate -cred $cred -computer F170, F171, F171,
F173, F174
Invoke-Command -Script { Set-date -date "24.6.2009 15:20:00" } -session $s
```

12.14 Verwaltung des WS-Management-Dienstes

In PowerShell ist der WS-Management-Dienst (WS-Man) über einen PowerShell-Provider (mit Namen „WSMan“) administrierbar. Im Standard erscheint in der Liste das Laufwerk „WSMan“.

```
PS C:\Users\hs> get-psdrive

Name           Used (GB)     Free (GB) Provider      Root                                               CurrentLocation
----           ---------     --------- --------      ----                                               ---------------
A                                      FileSystem    A:\
Alias                                  Alias
C                  10,90         86,65 FileSystem    C:\                                                Users\hs
cert                                   Certificate   \
Env                                    Environment
Function                               Function
G                                      FileSystem    G:\
H                   ,09         97,57 FileSystem    H:\
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
S                  5,24        227,65 FileSystem    S:\
T                   ,09         37,48 FileSystem    T:\
Variable                               Variable
WSMan                                  WSMan
```

Bild 12.19 Liste der PowerShell-Provider in PowerShell

In diesem Laufwerk kann man wie bei anderen Providern mit den Standard-Navigations-Commandlets wie `Get-ChildItem` (`dir`) und `Get-Item`/`Set-Item` arbeiten.

```
PS C:\Users\hs>
PS C:\Users\hs> cd wsman:\localhost
PS WSMan:\localhost> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Name                     Value                                                Type
----                     -----                                                ----
MaxEnvelopeSizekb        150                                                  System.String
MaxTimeoutms             60000                                                System.String
MaxBatchItems            32000                                                System.String
MaxProviderRequests      4294967295                                           System.String
Client                                                                        Container
Service                                                                       Container
Shell                                                                         Container
Listener                                                                      Container
Plugin                                                                        Container
ClientCertificate                                                             Container
```

Bild 12.20 Auflisten von WSMan:/localhost

Im Standard erscheint unterhalb der Laufwerkswurzel WSMan: nur „localhost“. Durch das Commandlet `Connect-WSMan` unter Angabe eines Computernamens kann man hier aber weitere Computer integrieren und im Folgenden ansteuern.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\hs> dir wsman:


   WSManConfig:

ComputerName                                  Type
------------                                  ----
localhost                                     Container


PS C:\Users\hs> connect-wsman -ComputerName F170
PS C:\Users\hs> dir wsman:


   WSManConfig:

ComputerName                                  Type
------------                                  ----
localhost                                     Container
f170                                          Container


PS C:\Users\hs> ^R_
```

Bild 12.21
Einsatz von Connect-WSMan

Die folgende Befehlsfolge zeigt das Setzen der vertrauenswürdigen Computer, zu denen eine Authentifizierung mit Basisauthentifizierung oder Digest toleriert werden soll.

```
cd WSMan:\localhost\Client
Set-Item trustedhosts "F170, F171, F172, F173, F174, F175" -force
Restart-Service winrm
```

Über das WSMan:-Laufwerk kann man auch den verwendeten Port ändern. Den aktuell verwendeten Port listet man auf mit:

```
Dir wsman:\localhost\listener\*\Port
```

Auf dem Zielsystem ändert man den Port mit (hier wird geändert auf Port 80):

```
set-item wsman:\localhost\listener\*\Port -value 80 -Force
```

Auf dem Client kann man dann den Port auf Aufruf angeben:

```
Enter-PSSession PC199 -Port  80
```

12.15 PowerShell Direct für Hyper-V

Ein virtuelles System können Sie grundsätzlich genauso wie ein physikalisches System fernverwalten über RPC oder PowerShell-Remoting. Für PowerShell-Remoting muss dies auf dem Zielsystem aktiviert sein.

Ab Windows 10 und Windows Server 2016 bietet Microsoft für in Hyper-V gehostete virtuelle Maschinen eine Vereinfachung unter dem Namen „PowerShell Direct“ an. Hier ist eine PowerShell-Remoting-Verbindung nicht mehr notwendig. Auch die Firewall muss nicht geöffnet sein. Die PowerShell redet direkt über den VMBus von Hyper-V.

Voraussetzungen sind:

- Der Hyper-V-Host ist Windows 10 oder Windows Server 2016 oder höher.
- Der Hyper-V-Gast ist Windows 10 oder Windows Server 2016 oder höher.
- Das Gastbetriebssystem muss laufen.
- In dem Gastbetriebssystem muss der „Hyper-V VM Session Service“ (vmicvmsession) laufen.
- Der Aufrufer muss Hyper-V-Administrator sein.
- Die PowerShell-Konsole muss mit Administratorrechten laufen.
- Der Aufrufer braucht ein Benutzerkonto auf dem Gastbetriebssystem. Dieses Konto muss aber keine Administratorrechte haben!

Die Commandlets `New-PSSession` und `Invoke-Command` haben dafür die neuen Parameter `-VMName` und `-VMGUID` erhalten. Auf eine der beiden Weisen identifiziert der Nutzer die anzusprechende virtuelle Maschine.

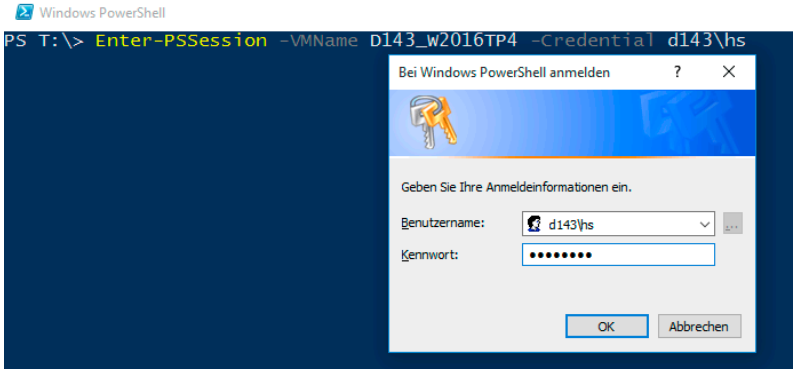

Bild 12.22 Initiieren einer Sitzung mit PowerShell Direct

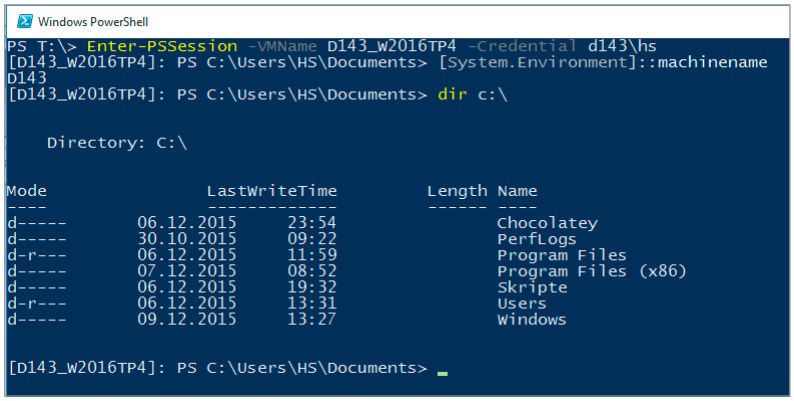

Bild 12.23 Nutzen einer PowerShell-Direct-Sitzung

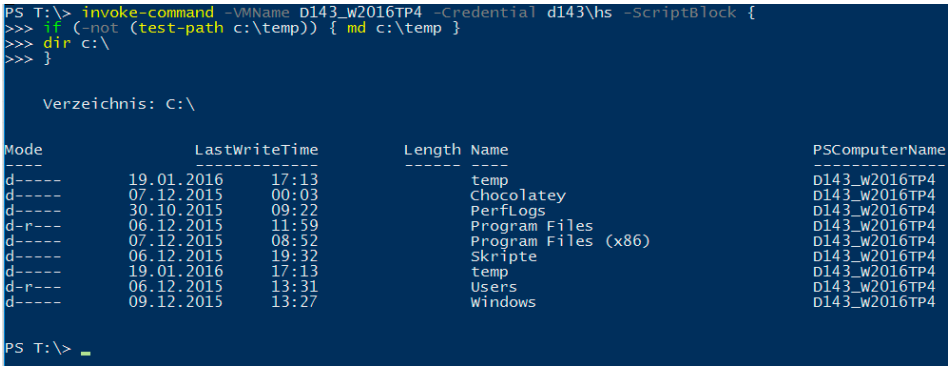

Bild 12.24 Ausführung eines Skriptblocks via PowerShell Direct

Die Befehle, die PowerShell Direct verwenden, müssen direkt auf dem Hyper-V-Host ausgeführt werden. Es ist nicht möglich, sich mit PowerShell Remoting mit dem Hyper-V-Host zu verbinden und diesem dann den Auftrag zu einer PowerShell-Direct-Verbindung zu geben. Diesen Versuch quittiert die PowerShell mit „Specified method is not supported."

TIPP: Auch bei PowerShell Direct ist es möglich, das Kennwort für das Gastbetriebssystem im Quellcode eines Skripts zu verankern – wenn man sich der Risiken bewusst ist.

```
$VMName = "D130"
$cred = new-object System.Management.Automation.PSCredential -ArgumentList
"D130\hs", (ConvertTo-SecureString "geheim+123" -AsPlainText -force)
Enter-PSSession -vmname D130 -Credential $cred
```

12.16 Praxisbeispiel zu PowerShell Direct

Das folgende Praxisskript zeigt, wie man eine frisch installierte virtuelle Maschine vom Hyper-V-Host aus konfiguriert:

- Festlegen einer statischen IP-Adresse
- Setzen des DNS-Servers
- Umbenennen des Computers
- Beitritt zu einer Domäne
- Neustart
- Alle PowerShell-Skripte erlauben
- PowerShell Remoting aktivieren
- Zugriff per RDP aktivieren

Danach ist dann eine Fernadministration des Rechners als neues Domänenmitglied per RDP und PowerShell Remoting möglich!

Listing 12.6 [2_Aufbauwissen\Remoting\WPS5_PowerShell_Direct_PCKonfig.ps1]

```
# Konfiguration einer Windows-VM in HyperV
# (C) Holger Schwichtenberg, www.IT-Visions.de, 2013-2016
# Die folgenden Eingabedaten müssen vor dem Start angepasst werden!

# Zugang zur VM
$VMName = "F131"
# Kennwörter hier nur zu Testzwecken im Skript. Müssen ggf. interaktiv abgefragt
werden!
$cred = new-object System.Management.Automation.PSCredential -ArgumentList "$VMName\
administrator", (ConvertTo-SecureString "fbi+123" -AsPlainText -force)
```

```
$credDomain = new-object System.Management.Automation.PSCredential -ArgumentList
"fbi\administrator", (ConvertTo-SecureString ‚fbi+123' -AsPlainText -force)

# Daten für neue Konfiguration
$ip = "192.168.1.131"
$DNSServer = "192.168.1.111"
$Gateway = "192.168.1.253"
$Domainname = "fbi.local"

# Skript-Einstellungen
$ErrorActionPreference = "stop"

########################

# Hilfsfunktion, die auf Verbindung zum Gast-Betriebssystem wartet
function WaitFor-PSDirect([string]$VMName, $cred){
   Write-Output "Warte auf PowerShell Direct-Verbindung zu $VMName mit Benutzer
$($cred.username)"
   while ((invoke-command -VMName $VMName -Credential $cred {"Test"} -ea
SilentlyContinue) -ne "Test") {Sleep -Seconds 1}
   "$VMName ist verfügbar via PowerShell Direct!"
   }

######################## Block 1:
# Festlegen einer statischen IP-Adressen
# Setzen des DNS-Servers
# Umbenennen des Computers
# Beitritt zu einer Domäne
# Neustart

$scriptblock1 = {
param($Name,$IP,$credDomain,$Gateway,$DNSServer,$Domainname)

$InterfaceName = (Get-netadapter | select-object -first 1).name
"Bisheriger Name $([System.Environment]::machinename) mit Netzadapter $netzadapter"
Get-NetIPAddress
"Neuer Name: $Name mit IP $ip Gateway $Gateway DNSServer $DNSServer"
Set-NetIPInterface -InterfaceAlias $InterfaceName -Dhcp Disabled
New-NetIPAddress -InterfaceAlias $InterfaceName -IPAddress  $IP -DefaultGateway
$Gateway -PrefixLength 24 | out-null
Set-DnsClientServerAddress -InterfaceAlias $InterfaceName -ServerAddresses $DNSServer
ping $ip
"Rename Computer und Domain Join..."
Add-Computer -NewName $Name -Verbose -DomainName $Domainname -Credential $credDomain
"PowerShell-Direkt-Skript #1 ist fertig!"
}

# Block 1 ausführen
"Konfiguriere VM $vmname - Schritt 1"
Invoke-Command -VMName $VMName -Credential $cred -ScriptBlock $scriptblock1
-ArgumentList $vmname,$ip,$credDomain,$Gateway,$DNSServer,$Domainname
"Reboot"

# Neustart und dann warten, bis Betriebssystem wieder zugreifbar!
Restart-VM $VMName -Force
WaitFor-PSDirect $VMName $cred

########################
```

```
# Alle PowerShell-Skripte erlauben
# PowerShell Remoting aktivieren
# Zugriff per RDP aktivieren

$scriptblock2 = {
param()
"PowerShell-Skripte erlauben..."
Set-ExecutionPolicy Unrestricted
"PowerShell-Remoting erlauben..."
Enable-psremoting -force -SkipNetworkProfileCheck
"RDP erlauben..."
set-ItemProperty -Path ‚HKLM:\System\CurrentControlSet\Control\Terminal Server'-name
"fDenyTSConnections" -Value 0
Enable-NetFirewallRule -DisplayGroup "Remote Desktop"
set-ItemProperty -Path ‚HKLM:\System\CurrentControlSet\Control\Terminal Server\
WinStations\RDP-Tcp' -name "UserAuthentication" -Value 1

"PowerShell-Direkt-Skript #2 ist fertig!"
}

# Block 2 ausführen
"Konfiguriere VM $vmname - Schritt 2"
Invoke-Command -VMName $VMName -Credential $cred -ScriptBlock $scriptblock2
"FERTIG: Fernadministration des Rechners $vmname als neues Domänenmitglied per RDP und
PowerShell Remoting nun möglich!"
```

Stichwortverzeichnis

D

E

F

G

J

K

L

M

N

Q

R

S

T

X

Y

Z