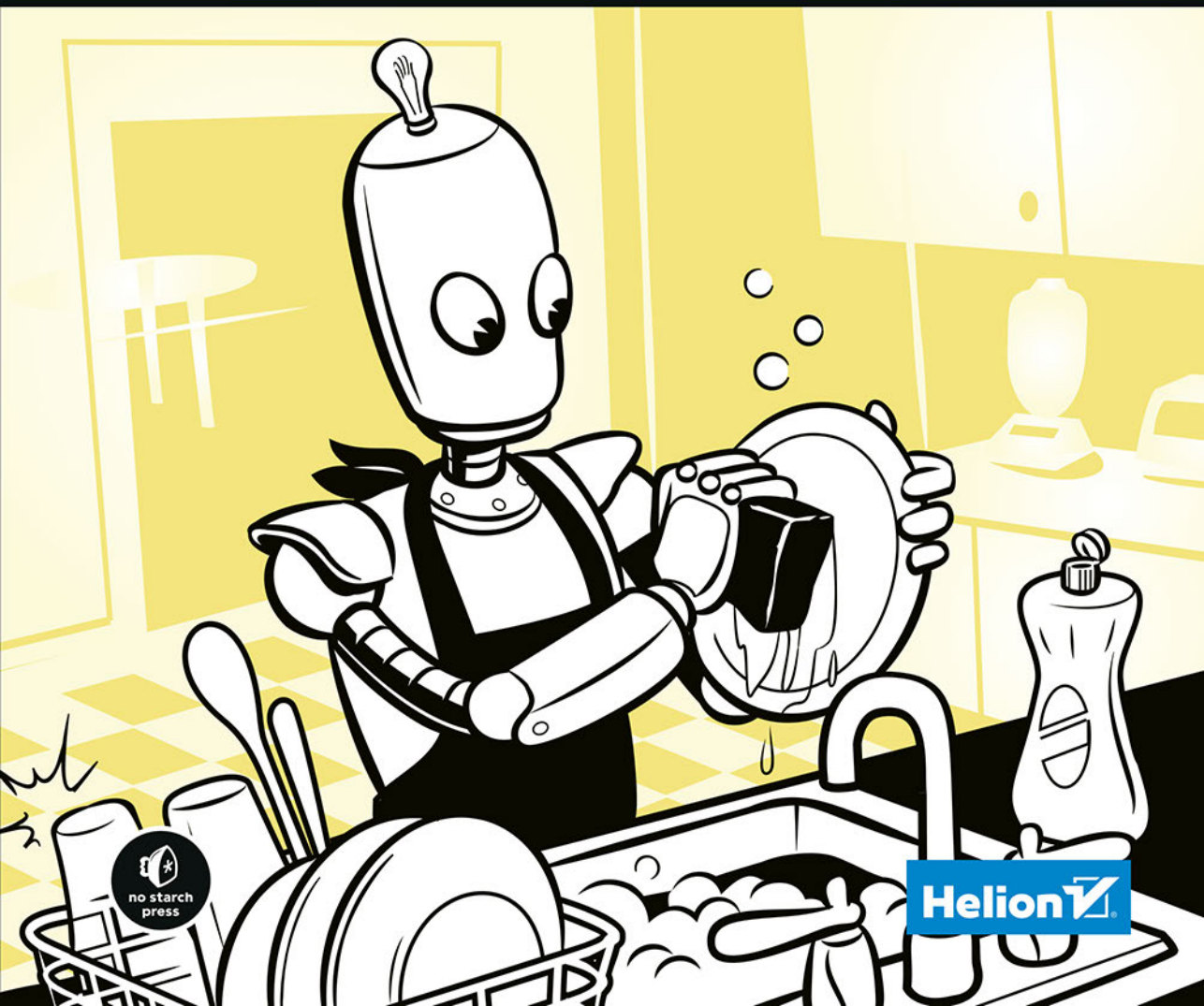
POWERSHELL DLA ADMINISTRATORÓW SYSTEMÓW

PROSTA AUTOMATYZACJA ZADAŃ

ADAM BERTRAM



Tytuł oryginału: PowerShell for Sysadmins: Workflow Automation Made Easy

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-7291-7

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/psadsy
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/psadsy.zip

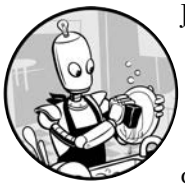
Printed in Poland.

Spis treści

Część III
Budowanie swojego własnego modułu

8

Zdalne uruchamianie skryptów



JEŻELI JESTEŚ JEDYNYM INFORMATYKIEM PRACUJĄCYM W MAŁEJ FIRMIE CZY ORGANIZACJI, PRAWDOPODOBNIE MASZ CO NAJMNIEJ KILKA SERWERÓW DO ZARZĄDZANIA. JEŻELI MASZ SKRYPT, który musisz uruchomić, możesz zalogować się do każdego z serwerów, otworzyć konsolę powłoki PowerShell i ręcznie uruchomić tam swój skrypt. Ale możesz zaoszczędzić dużo czasu, uruchamiając jeden skrypt, który zdalnie wykona określone zadanie na każdym serwerze. W tym rozdziale dowiesz się, jak uruchamiać polecenia na zdalnych hostach przy użyciu mechanizmów powłoki PowerShell.

Usługi zdalne powłoki PowerShell (ang. *PowerShell remoting*) to mechanizm, który umożliwia użytkownikowi zdalne uruchamianie poleceń w sesji na jednym lub wielu komputerach jednocześnie. *Sesja*, a dokładniej `PSSession`, jest terminem związanym z pracą zdalną powłoki PowerShell, odnoszącym się do środowiska, w którym PowerShell działa na komputerze zdalnym, gdzie wykonywane są polecenia. Popularne narzędzie `psexec` z pakietu SysInternals firmy Microsoft, choć działające w nieco inny sposób, wykorzystuje tę samą koncepcję: przygotowujesz kod, który działa na komputerze lokalnym, a następnie przesyłasz ten kod do komputera zdalnego i wykonujesz go tam tak, jakbyś siedział przy jego klawiaturze i ekranie.

Większość tego rozdziału poświęcimy na omawianie zagadnień związanych z sesjami zdalnymi. Dowiesz się, czym one są, jak ich używać i co zrobić, gdy już zdalnie zrobisz to, co chciałeś zrobić — ale najpierw musisz się dowiedzieć kilku ciekawych rzeczy o blokach skryptów (ang. *scriptblocks*).

Uwaga *Firma Microsoft wprowadziła zdalne wykonywanie poleceń w wersji 2 powłoki PowerShell, która jest oparta na usłudze WinRM (ang.* Windows Remote Management*). Z tego powodu określenie* WinRM *jest czasami używane w odniesieniu do zdalnych usług powłoki PowerShell.*

Praca z blokami skryptów

Mechanizm zdalnych usług powłoki PowerShell w szerokim zakresie wykorzystuje *bloki skryptów*, które — podobnie jak funkcje — są fragmentami kodu spakowanego w pojedyncze elementy wykonywalne. Bloki skryptu różnią się jednak od funkcji na kilka kluczowych sposobów: przede wszystkim są anonimowe (nie mają swoich nazw) i można je przypisywać do zmiennych.

Aby lepiej poznać te różnicę, rozważmy następujący przykład. Zdefiniujemy funkcję o nazwie `New-Thing`, która za pomocą polecenia `Write-Host` wyświetla jakiś tekst w konsoli (listing 8.1).

Listing 8.1. Kod funkcji New-Thing, która wyświetla kod w oknie konsoli

```
function New-Thing {
    param()
    Write-Host "Cześć! Tu funkcja New-Thing!"
}

New-Thing
```

Jeżeli uruchomisz ten skrypt, powinieneś zobaczyć, że w oknie konsoli wyświetlony zostanie tekst „Cześć! Tu funkcja New-Thing!”. Zwróć jednak uwagę, że aby uzyskać ten rezultat, musimy wywołać funkcję `New-Thing`.

Podobny rezultat możemy osiągnąć za pomocą bloku skryptu, który zostanie najpierw przypisany do zmiennej, jak pokazano na listingu 8.2.

Listing 8.2. Tworzenie bloku skryptu i przypisanie go do zmiennej $newThing

```
PS> $newThing = { Write-Host "Cześć! Jestem blokiem skryptu!" }
```

Aby zbudować blok skryptu, wystarczy umieścić kod, który chcesz wykonać, w nawiasach klamrowych. Skoro nasz blok skryptu jest teraz przechowywany w zmiennej `$newThing`, możesz sobie pomyśleć, że aby go wykonać, wystarczy wywołać tę zmienną, jak pokazano na listingu 8.3.

Listing 8.3. Tworzenie i próba wykonania bloku skryptu

```
PS> $newThing = { Write-Host "Cześć! Jestem blokiem skryptu!" }
PS> $newThing
Write-Host "Cześć! Jestem blokiem skryptu!"
```

Niestety, jak widać, powłoka PowerShell traktuje zawartość zmiennej `$newThing` jako zwykłą wartość. Nie rozpoznaje, że ciąg znaków `Write-Host` jest poleceniem, które powinna wykonać, i zamiast tego wyświetla zawartość bloku skryptu.

Aby poinformować powłokę PowerShell, że powinna uruchomić kod znajdujący się w zmiennej, musisz użyć znaku `&` (ang. *ampersand*), po którym powinieneś umieścić nazwę zmiennej. Na listingu 8.4 przedstawiono wynik takiej operacji.

Listing 8.4. Wykonywanie bloku skryptu

```
PS> & $newThing
Cześć! Jestem blokiem skryptu!
```

Znak `&` informuje powłokę PowerShell, że zawartość umieszczona między nawiasami klamrowymi to kod, który powinien zostać uruchomiony. Znak `&` jest jednym ze sposobów wykonania bloku kodu; nie pozwala jednak na dostosowanie polecenia, co będzie potrzebne podczas korzystania z komunikacji zdalnej powłoki PowerShell do pracy na komputerach zdalnych. W następnej sekcji omówimy zatem inny sposób wykonywania bloków skryptów.

Zastosowanie polecenia Invoke-Command do wykonywania kodu w systemach zdalnych

Pracując z usługami zdalnymi powłoki PowerShell, będziesz w większości przypadków korzystać z dwóch najważniejszych poleceń: `Invoke-Command` i `New-PSSession`. W tej sekcji poznasz polecenie `Invoke-Command`, a w następnej omówimy polecenie `New-PSSession`.

`Invoke-Command` jest prawdopodobnie poleceniem, którego przy połączeniach zdalnych będziesz używać najczęściej. Można z niego korzystać na dwa główne sposoby. Pierwszy z nich to *działania ad hoc*, czyli uruchamianie pojedynczych poleceń czy jednowierszowych sekwencji poleceń, mających na celu wykonanie na komputerze zdalnym określonego, prostego zadania. Drugi sposób wykorzystuje pełne sesje interaktywne. Oba sposoby omówimy w dalszej części tej sekcji.

Przykładem zdalnego działania ad hoc jest uruchomienie wybranej usługi na komputerze zdalnym za pomocą polecenia `Start-Service`. Podczas wykonywania takiego działania przy użyciu polecenia `Invoke-Command` powłoka PowerShell tworzy w tle sesję zdalną i następnie zamyka ją, gdy tylko polecenie zostanie wykonane. Takie rozwiązanie nieco ogranicza to, co możesz zrobić zdalnie z użyciem polecenia `Invoke-Command`, dlatego w następnej sekcji dowiesz się, jak tworzyć własne zdalne sesje.

Na razie jednak przekonasz się, jak polecenie `Invoke-Command` wykonuje zdalne działania ad hoc. Aby to zrobić, uruchom konsolę powłoki PowerShell, wpisz polecenie `Invoke-Command` i naciśnij klawisz *Enter*, jak pokazano na listingu 8.5.

Listing 8.5. Uruchamianie polecenia Invoke-Command bez żadnych parametrów

```
PS> Invoke-Command

cmdlet Invoke-Command at command pipeline position 1
Supply values for the following parameters:
ScriptBlock:
```

Po wykonaniu tego polecenia powłoka powinna „poprosić" o wskazanie bloku skryptu. Wykorzystamy tutaj polecenie `hostname`, które zwróci nazwę zdalnego komputera, na którym polecenie zostanie uruchomione.

Aby przekazać blok skryptu z poleceniem `hostname`, musisz użyć wymaganego parametru, `ComputerName`. Poinformuje ono polecenie `Invoke-Command`, na którym komputerze zdalnym ma zostać uruchomione to polecenie, tak jak to zostało pokazane na listingu 8.6 (pamiętaj, że aby to przykładowe polecenie zadziałało poprawnie, zarówno mój komputer, jak i komputer zdalny `WEBSRV1` musiały być częścią tej samej domeny Active Directory (AD), a dodatkowo moje konto użytkownika musiało mieć uprawnienia administratora na komputerze `WEBSRV1`).

Listing 8.6. Uruchamianie prostego polecenia Invoke-Command

```
PS> Invoke-Command -ScriptBlock { hostname } -ComputerName WEBSRV1
WEBSRV1
```

Zwróć uwagę, że wynikiem działania polecenia `hostname` jest tutaj nazwa zdalnego komputera — w tym przypadku jest to serwer o nazwie `WEBSRV1`. Właśnie wykonałeś zdalnie swoje pierwsze polecenie!

Uwaga *Jeżeli spróbujesz wykonać takie polecenie na komputerze zdalnym z systemem operacyjnym starszym niż Windows Server 2012 R2, może się okazać, że nie będzie to działać zgodnie z oczekiwaniami. W takim przypadku musisz najpierw włączyć usługi zdalne powłoki PowerShell. Począwszy od systemu Windows Server 2012 R2 usługi zdalne powłoki PowerShell są włączone domyślnie, a usługa WinRM działa z otwartymi wszystkimi niezbędnymi portami zapory i odpowiednimi prawami dostępu. Jeżeli jednak używasz starszej wersji systemu Windows, musisz to zrobić ręcznie. W takiej sytuacji, zanim będziesz mógł użyć polecenia `Invoke-Command`, powinieneś na komputerze zdalnym uruchomić sesję konsoli z podwyższonym poziomem uprawnień i włączyć usługi zdalne, wykonując polecenie `Enable-PSRemoting`. Możesz również użyć polecenia `Test-WSMan`, aby upewnić się, że usługi zdalne powłoki PowerShell są skonfigurowane poprawnie i dostępne.*

Uruchamianie lokalnych skryptów na komputerach zdalnych

W poprzedniej sekcji uruchomiłeś proste polecenie na komputerze zdalnym. Polecenia `Invoke-Command` możesz także używać do wykonywania całych skryptów. W takiej sytuacji zamiast parametru `Scriptblock` powinieneś użyć parametru `FilePath` i jako jego wartość podać ścieżkę do skryptu na lokalnym komputerze. Polecenie `Invoke-Command` odczyta zawartość skryptu z dysku lokalnego, a następnie wykona ten kod na komputerze zdalnym. Wbrew powszechnemu przekonaniu sam skrypt nie jest wykonywany na komputerze zdalnym.

Aby to zademonstrować, załóżmy, że na swoim komputerze masz skrypt o nazwie *GetHostName.ps1*, który znajduje się w katalogu głównym *C:*. W naszym skrypcie znajduje się tylko jedno polecenie: `hostname`. Teraz chcesz uruchomić ten skrypt na komputerze zdalnym, aby zwrócił jego nazwę. Zauważ, że choć nasz przykładowy skrypt jest niezwykle prosty, polecenie `Invoke -Command` zupełnie nie dba o zawartość skryptu i po prostu wykona wszystko, co w nim znajdzie.

Aby uruchomić skrypt, powinieneś przekazać plik skryptu do polecenia `Invoke-Command` za pomocą parametru `FilePath` (listing 8.7).

Listing 8.7. Wykonywanie lokalnego skryptu na komputerze zdalnym

```
PS> Invoke-Command -ComputerName WEBSRV1 -FilePath C:\GetHostName.ps1
WEBSRV1
```

Polecenie `Invoke-Command` uruchamia na komputerze `WEBSRV1` kod znajdujący się w lokalnym skrypcie *GetHostName.ps1* i zwraca wyniki jego działania z powrotem do sesji lokalnej.

Zdalne używanie zmiennych lokalnych

Choć usługi zdalne powłoki PowerShell potrafią same zadbać o bardzo wiele szczegółów, powinieneś uważać podczas używania zmiennych lokalnych w sesji zdalnej. Załóżmy, że na komputerze zdalnym znajduje się plik *C:\File.txt*. Ponieważ ścieżka do tego pliku może się w pewnym momencie zmienić, postanowiłeś ją przypisać do zmiennej, na przykład o nazwie `$serverFilePath`:

```
PS> $serverFilePath = 'C:\File.txt'
```

Załóżmy teraz, że wewnątrz bloku skryptu, który będzie wykonany na komputerze zdalnym, chciałbyś się odwołać do pliku *C:\File.txt*. Na listingu 8.8 możesz zobaczyć, co się stanie, gdy spróbujesz odwołać się do tego pliku za pośrednictwem utworzonej przed chwilą zmiennej lokalnej.

Listing 8.8. Odwołania do zmiennych lokalnych nie działają w sesjach zdalnych

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { Write-Host "Ścieżka
➥do pliku to $serverFilePath" }
Ścieżka do pliku to
```

Zwróć uwagę, że zmienna `$serverFilePath` nie posiada wartości, ponieważ w trakcie wykonywania bloku skryptu taka zmienna na komputerze zdalnym po prostu nie istnieje! Podczas definiowania zmiennej w skrypcie lub z poziomu konsoli taka zmienna jest przechowywana w określonym obszarze działania (ang. *runspace*), który jest kontenerem używanym przez powłokę PowerShell do przechowywania informacji o sesji. Z obszarami działania możesz zetknąć się na przykład w sytuacji, kiedy na swoim komputerze uruchomisz dwie sesje powłoki PowerShell w tym samym czasie, w pierwszej z nich zdefiniujesz nową zmienną, a następnie z poziomu drugiej będziesz próbował się do niej odwołać. Taka operacja zakończy się niepowodzeniem, ponieważ nowo zdefiniowana zmienna istnieje tylko w obszarze działania pierwszej sesji powłoki i nie jest dostępna dla drugiej sesji.

Domyślnie zmienne, funkcje i inne elementy nie są dostępne w wielu obszarach działania, aczkolwiek istnieje kilka sposobów na ich udostępnianie. Przykładowo, są dwa główne sposoby przesyłania zmiennych do komputera zdalnego.

Przekazywanie zmiennych za pomocą parametru ArgumentList

Aby pobrać wartość zmiennej lokalnej i przekazać ją do zdalnego bloku skryptu, możesz użyć parametru `ArgumentList` polecenia `Invoke-Command`. Parametr ten umożliwia przekazanie tablicy `$args` zawierającej szereg wartości lokalnych, których następnie możesz użyć w kodzie bloku skryptu. Aby pokazać, jak to działa, na listingu 8.9 przekazujemy zmienną lokalną `$serverFilePath`, zawierającą ścieżkę do pliku *C:\File.txt*, do zdalnego bloku skryptów, a następnie odwołujemy się do niej poprzez tablicę `$args`.

Listing 8.9. Zastosowanie tablicy $args do przekazywania zmiennych lokalnych do sesji zdalnej

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { Write-Host "Ścieżka
↪do pliku to  $($args[0])" } -ArgumentList $serverFilePath
Ścieżka do pliku to C:\File.txt
```

Jak widać, ścieżka do pliku *C:\File.txt* została teraz wyświetlona w sesji zdalnej. Dzieje się tak, ponieważ przekazaliśmy zmienną `$serverFilePath` za pomocą parametru `ArgumentList` i wewnątrz skryptu zastąpiliśmy odwołanie do zmiennej `$serverFilePath` odwołaniem do tablicy `$args[0]`. Jeżeli chcesz przekazać więcej niż jedną zmienną do bloku skryptu, możesz dodać kolejną wartość do parametru `ArgumentList` i odpowiednio odwoływać się do następnego elementu tablicy `$args`.

Przekazywanie zmiennych za pomocą instrukcji $using

Innym sposobem przekazywania wartości zmiennych lokalnych do zdalnego bloku skryptu jest użycie instrukcji `$using`. Aby uniknąć używania parametru `ArgumentList`, możesz dodawać instrukcję `$using` do dowolnej nazwy zmiennej lokalnej. Dzięki temu, zanim powłoka PowerShell wyśle blok skryptu do komputera

zdalnego, wyszuka miejsca, w których użyłeś instrukcji `$using`, i rozwinie wszystkie oznaczone nią zmienne lokalne wewnątrz bloku skryptu.

Na listingu 8.10 zamieszczono zmodyfikowaną wersję kodu z listingu 8.9, gdzie zamiast parametru `ArgumentList` użyte zostało wyrażenie `$using:serverFilePath`.

Listing 8.10. Zastosowanie instrukcji $using w odwołaniach do zmiennych lokalnych w sesji zdalnej

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { Write-Host " Ścieżka
↪do pliku to  $using:serverFilePath" }
Ścieżka do pliku to C:\File.txt
```

Jak widać, wyniki działania poleceń przedstawionych na listingach 8.9 i 8.10 są takie same.

Instrukcja `$using` wymaga mniej pracy i jest bardziej intuicyjna, ale kiedy zaczniemy tworzyć pakiet Pester, przeznaczony do testowania skryptów, przekonasz się, że być może będziesz musiał powrócić do używania parametru `ArgumentList` — w przeciwnym razie Pester nie będzie miał możliwości oszacowania wartości zmiennej przekazywanej za pomocą instrukcji `$using`. Kiedy używasz parametru `ArgumentList`, zmienne przekazywane do sesji zdalnej są definiowane lokalnie, dzięki czemu Pester może je poprawnie zinterpretować. Jeżeli teraz nie do końca rozumiesz, o co w tym chodzi, to nie przejmuj się — wszystko nabierze sensu podczas czytania rozdziału 9. Najważniejsze, że póki co instrukcja `$using` działa doskonale!

Teraz, gdy masz już podstawową wiedzę na temat polecenia `Invoke-Command`, możemy przejść do najważniejszych zagadnień związanych z sesjami zdalnymi.

Praca z sesjami zdalnymi

Jak już wspominaliśmy, usługi zdalne powłoki PowerShell wykorzystują koncepcję *sesji*. Kiedy nawiązujesz połączenie ze zdalnym komputerem, powłoka PowerShell otwiera na komputerze zdalnym *lokalną sesję*, dzięki której możesz wykonywać polecenia i skrypty. Nie musisz znać zbyt wielu technicznych szczegółów związanych z tworzeniem i działaniem sesji. Wystarczy pamiętać, że możesz utworzyć zdalną sesję, połączyć się z nią, rozłączyć się z nią i że zachowuje ona stan, w jakim ją zostawiłeś. Sesja zdalna nie zostanie usunięta, dopóki jej nie zakończysz.

W poprzednim podrozdziale, kiedy uruchamiałeś polecenie `Invoke-Command`, powłoka PowerShell tworzyła nową sesję, wykonywała przekazywany kod i automatycznie kończyła sesję. W tym podrozdziale dowiesz się, jak tworzyć tak zwane *pełne sesje* (ang. *full sessions*), czyli sesje, w których możesz bezpośrednio wprowadzać polecenia. Używanie polecenia `Invoke-Command` do jednorazowego wykonywania poleceń ad hoc sprawdza się całkiem dobrze, ale nie jest zbyt efektywne, gdy trzeba uruchomić wiele poleceń, których nie można zmieścić w jednym bloku skryptu. Na przykład jeżeli pracowałbyś nad dużym skryptem, który wykonuje jakieś operacje lokalnie, pobiera informacje z zewnętrznych źródeł, używa

tych informacji w sesji zdalnej, pobiera informacje z sesji zdalnej do wykorzystania lokalnie, a następnie powraca do sesji lokalnej, musiałbyś utworzyć skrypt, który będzie wielokrotnie uruchamiał polecenie `Invoke-Command`. Jeszcze więcej problemów pojawi się, kiedy będziesz musiał ustawiać jakieś zmienne w sesji zdalnej i później używać ich ponownie. Wówczas polecenie `Invoke-Command` w takiej formie, w jakiej używałeś go dotychczas, zupełnie się nie sprawdzi — zamiast tego potrzebujesz zdalnej sesji, która pozostanie aktywna nawet wtedy, gdy się z nią chwilowo rozłączysz.

Tworzenie nowej sesji

Aby utworzyć trwałe połączenie z komputerem zdalnym za pomocą powłoki PowerShell, powinieneś użyć polecenia `New-PSSession`, które utworzy pełną sesję z komputerem zdalnym i udostępni Ci ją na komputerze lokalnym.

Aby utworzyć nową sesję `PSSession`, powinieneś zastosować polecenie `New-PSSession` z parametrem `ComputerName`, jak pokazano na listingu 8.11. W tym przykładzie komputer, na którym zostało uruchomione to polecenie, znajduje się w tej samej domenie Active Directory, co serwer `WEBSRV1`, a moje konto, na którym jestem zalogowany, ma uprawnienia administratora na serwerze `WEBSRV1`. Aby połączyć się ze zdalnym komputerem przy użyciu parametru `ComputerName` (jak na listingu 8.11), konto użytkownika musi mieć prawa lokalnego administratora lub przynajmniej należeć do grupy *Użytkownicy zarządzania zdalnego* na komputerze zdalnym. Jeżeli Twoje konto użytkownika nie należy do domeny AD, możesz zastosować parametr `Credential` polecenia `New-PSSession`, który umożliwia przekazanie obiektu `PSCredential` zawierającego poświadczenia logowania pozwalające na uwierzytelnienie na komputerze zdalnym.

Listing 8.11. Tworzenie nowej sesji PSSession

```
PS> New-PSSession -ComputerName WEBSRV1
Id  Name    ComputerName    ComputerType    State    ConfigurationName
↪Availability
--  ----    ------------    ------------    -----    -----------------
↪------------
 3  WinRM3  WEBSRV1         RemoteMachine   Opened   Microsoft.PowerShell
 ↪Available
```

Jak widać, polecenie `New-PSSession` zwraca gotową sesję. Po nawiązaniu połączenia i utworzeniu sesji możesz powracać do niej za pomocą polecenia `Invoke-Command`, z tym że zamiast używać parametru `ComputerName`, tak jak w poleceniach ad hoc, powinieneś użyć parametru `Session`.

W wierszu wywołania polecenia `Invoke-Command` musisz podać parametr `Session` z argumentem w postaci obiektu sesji. Aby wyświetlić listę wszystkich aktywnych sesji, powinieneś użyć polecenia `Get-PSSession`. Na listingu 8.12 wyniki działania polecenia `Get-PSSession` zostały przypisane do zmiennej.

Listing 8.12. Wyświetlanie sesji zdalnych utworzonych na komputerze lokalnym

```
PS> $session = Get-PSSession
PS> $session
Id  Name    ComputerName    ComputerType    State    ConfigurationName
↪Availability
--  ----    ------------    ------------    -----    -----------------
↪------------
 6  WinRM6  WEBSRV1         RemoteMachine   Opened   Microsoft.PowerShell
 ↪Available
```

Ponieważ polecenie `New-PSSession` uruchomiliśmy do tej pory tylko raz, na listingu widać tylko jedną sesję `PSSession`. Jeżeli masz utworzonych wiele sesji, za pomocą parametru `Id` (zobacz wyniki działania polecenia `Get-PSSession`) możesz wskazać sesję, której chcesz użyć w poleceniu `Invoke-Command`.

Wywoływanie poleceń w sesji

Teraz, gdy nasza przykładowa sesja została przypisana do zmiennej, możemy przekazywać ją do polecenia `Invoke-Command`, za pomocą którego będziemy wywoływać polecenia w tej sesji, tak jak pokazano na listingu 8.13.

Listing 8.13. Wykorzystanie istniejącej sesji do wykonania polecenia na komputerze zdalnym

```
PS> Invoke-Command -Session $session -ScriptBlock { hostname }
WEBSRV1
```

Zapewne zauważyłeś, że tym razem polecenie to zostało wykonane znacznie szybciej niż wtedy, gdy wykonywałeś je w trybie ad hoc. Dzieje się tak, ponieważ teraz polecenie `Invoke-Command` nie musi już tworzyć nowej sesji. Utworzenie pełnej sesji nie tylko powoduje, że poszczególne polecenia są wykonywane szybciej, ale także daje Ci dostęp do większej liczby funkcji. Na przykład, jak pokazano na listingu 8.14, w pełnej sesji możesz ustawić na komputerze zdalnym wybrane zmienne, a potem powrócić do tej sesji bez ich utraty.

Listing 8.14. Zmienne zachowują swoje wartości pomiędzy kolejnymi połączeniami do sesji

```
PS> Invoke-Command -Session $session -ScriptBlock { $foo = 'Mam nadzieję, że
↪tu będziesz!' }
PS> Invoke-Command -Session $session -ScriptBlock { $foo }
Mam nadzieję, że tu będziesz!
```

Dopóki sesja pozostaje otwarta, możesz w niej robić wszystko, czego potrzebujesz, a jej stan nie ulegnie zmianie. Jednak dotyczy to tylko bieżącej sesji lokalnej. Jeżeli rozpoczniesz kolejną sesję powłoki PowerShell, nie będziesz mógł kontynuować połączenia zdalnego od miejsca, w którym przerwałeś. Sesja zdalna

będzie nadal aktywna, ale odwołanie do niej na komputerze lokalnym zostanie utracone. W takim przypadku sesja PSSession przejdzie w stan rozłączenia (ang. *disconnected*), o którym powiemy więcej w następnej sekcji.

Otwieranie sesji interaktywnych

Na listingu 8.14 pokazaliśmy, jak wykorzystać cmdlet Invoke-Command do wysyłania poleceń do zdalnego komputera i pobierania wyników ich działania. Taki sposób wykonywania poleceń zdalnych można porównać do uruchamiania niemonitorowanego skryptu. Jeżeli chcesz uruchomić interaktywną konsolę dla sesji zdalnej — na przykład w celu rozwiązywania problemów — możesz użyć polecenia Enter-PSSession.

Polecenie Enter-PSSession umożliwia użytkownikowi interaktywną pracę z sesją. Może ono utworzyć własną sesję lub wykorzystać istniejącą sesję, utworzoną wcześniej za pomocą polecenia New-PSSession. Jeżeli nie wskażesz sesji, której chcesz użyć, polecenie Enter-PSSession utworzy nową sesję i rozpocznie oczekiwanie na dalsze polecenia (listing 8.15).

Listing 8.15. Utworzenie zdalnej sesji interaktywnej

```
PS> Enter-PSSession -ComputerName WEBSRV1
[WEBSRV1]: PS C:\Users\Adam\Documents>
```

Zwróć uwagę, że po nawiązaniu połączenia znak zachęty powłoki PowerShell zmienia się na [WEBSRV1]: PowerShell, co wskazuje, że nie wykonujesz już poleceń lokalnie, ale w sesji zdalnej. W tym momencie możesz uruchomić dowolne polecenie, tak jakbyś siedział przy konsoli zdalnego komputera. Taka interaktywna praca ze zdalnymi sesjami to świetny sposób na uniknięcie konieczności używania aplikacji Remote Desktop Protocol (RDP) do wykonywania różnych zadań na komputerze zdalnym, takich jak konfiguracja czy rozwiązywanie problemów.

Rozłączanie i ponowne nawiązywanie połączenia z sesjami

Jeżeli zamkniesz konsolę PowerShell, a następnie otworzysz ją ponownie i spróbujesz użyć polecenia Invoke-Command w sesji, z którą wcześniej pracowałeś, pojawi się komunikat o błędzie pokazany na listingu 8.16.

Listing 8.16. Próba wykonania polecenia w rozłączonej sesji

```
PS> $session = Get-PSSession -ComputerName websrv1
PS> Invoke-Command -Session $session -ScriptBlock { $foo }
Invoke-Command : Because the session state for session WinRM6, a617c702-ed92-
↪4de6-8800-40bbd4e1b20c, websrv1 is not equal to Open, you cannot run
↪a command in the session. The session state is Disconnected.
At line:1 char:1
+ Invoke-Command -Session $session -ScriptBlock { $foo }
--pozostałe wiersze zostały pominięte--
```

Powłoka PowerShell może odnaleźć sesję PSSession na komputerze zdalnym, ale nie potrafi odszukać jej uchwytu na komputerze lokalnym, co oznacza, że sesja jest rozłączona. Dzieje się tak, kiedy w poprawny sposób nie zamkniesz zdalnej sesji PSSession. Istniejące sesje można rozłączyć za pomocą polecenia Disconnect-PSSession.

Wszystkie utworzone wcześniej sesje możesz wyczyścić, pobierając je za pomocą polecenia Get-PSSession, a następnie przesyłając do polecenia Disconnect-PSSession (listing 8.17). Możesz również rozłączać pojedyncze, wybrane sesje, wskazując je za pomocą parametru Session polecenia Disconnect-PSSession.

Listing 8.17. Rozłączanie sesji PSSession

```
PS> Get-PSSession | Disconnect-PSSession
Id  Name    ComputerName    ComputerType    State         ConfigurationName
↪Availability
--  ----    ------------    ------------    -----         -----------------
↪------------
4   WinRM4  WEBSRV1         RemoteMachine   Disconnected  Microsoft.
↪PowerShell  None
```

Aby prawidłowo rozłączyć sesję, należy przekazać nazwę komputera zdalnego do parametru Session. Możesz to zrobić, wykonując polecenie Disconnect-PSSession -Session *obiekt_sesji* lub przesyłając istniejącą sesję za pomocą potoku bezpośrednio z wyników działania polecenia Get-PSSession, jak na listingu 8.17.

Jeżeli chcesz ponownie połączyć się z sesją rozłączoną za pomocą polecenia Disconnect-PSSession, zamknij konsolę PowerShell, uruchom ją ponownie, a następnie użyj polecenia Connect-PSSession, jak na listingu 8.18. Pamiętaj, że możesz wyświetlać i łączyć się tylko z tymi sesjami, które zostały utworzone na Twoim koncie. Nie możesz zobaczyć sesji utworzonych przez innych użytkowników.

Listing 8.18. Ponowne podłączanie się do rozłączonej wcześniej sesji

```
PS> Connect-PSSession -ComputerName websrv1
[WEBSRV1]: PS>
```

Po nawiązaniu połączenia powinieneś być w stanie uruchomić dowolny kod na komputerze zdalnym, tak jakbyś nigdy nie zamykał konsoli.

Jeżeli nadal otrzymujesz komunikat o błędzie, być może masz niezgodne wersje powłoki PowerShell. Rozłączone sesje działają tylko wtedy, gdy komputer lokalny i serwer zdalny używają tej samej wersji powłoki PowerShell. Na przykład jeżeli na komputerze lokalnym używasz powłoki PowerShell 5.1, ale serwer zdalny, z którym się łączysz, korzysta z innej wersji powłoki PowerShell, nieobsługującej rozłączonych sesji (na przykład PowerShell v2 lub starszej), rozłączone sesje nie będą działać. Pamiętaj, że zawsze warto upewnić się, że zarówno komputer lokalny, jak i serwer zdalny korzystają z tej samej wersji powłoki PowerShell.

Aby przekonać się, która wersja powłoki PowerShell jest zainstalowana na danym komputerze, powinieneś sprawdzić wartość zmiennej `$PSVersionTable`, która przechowuje informacje o wersji (listing 8.19).

Listing 8.19. Wyświetlanie informacji o wersji powłoki PowerShell na komputerze lokalnym

```
PS> $PSVersionTable
Name                           Value
----                           -----
PSVersion                      5.1.15063.674
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.15063.674
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Aby sprawdzić wersję powłoki PowerShell działającej na komputerze zdalnym, powinieneś uruchomić polecenie `Invoke-Command` na tym komputerze, przekazując mu w bloku skryptu nazwę zmiennej `$PSVersionTable`, jak pokazano na listingu 8.20.

Listing 8.20. Wyświetlanie informacji o wersji powłoki PowerShell na komputerze zdalnym

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { $PSVersionTable }
Name                           Value
----                           -----
PSRemotingProtocolVersion      2.2
BuildVersion                   6.3.9600.16394
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0}
PSVersion                      4.0
CLRVersion                     4.0.30319.34014
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
```

Pamiętaj, aby przed odłączeniem się od sesji sprawdzić, czy obie wersje powłoki są zgodne; w ten sposób można uniknąć utraty cennych wyników pracy w systemie zdalnym.

Usuwanie sesji za pomocą polecenia Remove-PSSession

Za każdym razem, gdy polecenie `New-PSSession` tworzy nową sesję, istnieje ona zarówno na serwerze zdalnym, jak i na komputerze lokalnym. Możesz utworzyć wiele sesji na wielu serwerach jednocześnie, a jeżeli niektóre z tych sesji nie będą już używane, być może będziesz musiał je zamknąć i wyczyścić. Możesz to zrobić za pomocą polecenia `Remove-PSSession`, które nawiązuje połączenie z komputerem zdalnym, zamyka sesję i usuwa lokalny uchwyt sesji `PSSession` (o ile taki uchwyt istnieje). Przykład takiego polecenia został pokazany na listingu 8.21.

Listing 8.21. Usuwanie sesji PSSession

```
PS> Get-PSSession | Remove-PSSession
PS> Get-PSSession
```

Jak łatwo zauważyć, po zamknięciu sesji ponowne uruchomienie polecenia `Get-PSSession` nie zwraca już żadnych rezultatów. Oznacza to, że na komputerze lokalnym nie ma już żadnych sesji zdalnych.

Mechanizm zdalnego uwierzytelniania powłoki PowerShell

Jak dotąd ignorowaliśmy kwestię uwierzytelniania. Domyślnie, jeżeli komputer lokalny i zdalny znajdują się w tej samej domenie i oba mają włączone usługi zdalne powłoki PowerShell, nie ma potrzeby jawnego uwierzytelniania. Ale jeżeli tak nie jest, zdalna sesja musi zostać jakoś uwierzytelniona.

Dwa najczęściej używane sposoby uwierzytelniania na komputerach zdalnych wykorzystywane przez powłokę PowerShell to użycie protokołu Kerberos lub CredSSP. Jeżeli jesteś w domenie Active Directory, prawdopodobnie korzystasz już z mechanizmu biletów Kerberos, czy o tym wiesz, czy nie. Active Directory i niektóre systemy Linux używają dziedzin systemu Kerberos, które wystawiają bilety uwierzytelniające klientom. Bilety takie są następnie wykorzystywane do uzyskiwania dostępu do zasobów sieciowych i weryfikowane na kontrolerach domeny.

Z drugiej strony uwierzytelnianie CredSSP nie potrzebuje Active Directory. Protokół CredSSP został wprowadzony w systemie Windows Vista i wykorzystuje dostawcę usług poświadczeń po stronie klienta (ang. *client-side credential service provider* — CSP), aby umożliwić aplikacjom delegowanie poświadczeń użytkownika do komputerów zdalnych. CredSSP nie wymaga systemu zewnętrznego, takiego jak kontroler domeny, do uwierzytelnienia dwóch systemów.

W środowisku Active Directory usługi zdalne powłoki PowerShell korzystają z protokołu Kerberos, za pomocą którego Active Directory przeprowadza wszystkie uwierzytelnienia. Powłoka PowerShell używa konta, na które jesteś zalogowany lokalnie, do uwierzytelnienia na komputerze zdalnym — tak jak wiele innych usług.

Jeżeli jednak nie pracujesz w środowisku Active Directory, możesz być zmuszony nieco zmienić typ uwierzytelniania; na przykład w sytuacji, gdy musisz połączyć się z komputerami zdalnymi przez sieć Internet albo w sieci lokalnej, ale za pomocą lokalnych poświadczeń na komputerze zdalnym. Powłoka PowerShell obsługuje wiele metod uwierzytelniania, lecz najczęściej używanym protokołem — poza Kerberosem — jest CredSSP, który umożliwia komputerowi lokalnemu delegowanie poświadczeń użytkownika do komputera zdalnego. Cała koncepcja jest podobna do zasady działania protokołu Kerberos, z tym że w przypadku CredSSP usługa Active Directory nie jest potrzebna.

W środowisku Active Directory zwykle nie musisz używać innego typu uwierzytelniania, ale czasami może to się okazać niezbędne, więc najlepiej gdy będziesz na to przygotowany. W tym podrozdziale poznasz jeden z typowych problemów z uwierzytelnianiem i dowiesz sie, jak go obejść.

Problem drugiego przeskoku

Problem drugiego przeskoku (ang. *double hop problem*) jest znany od czasu, gdy firma Microsoft dodała usługi zdalne do powłoki PowerShell. Pojawia się, gdy uruchamiasz kod w sesji zdalnej, a następnie próbujesz uzyskać dostęp do innych zasobów zdalnych z poziomu tej sesji. Na przykład jeżeli w swojej sieci masz kontroler domeny o nazwie `DC` i chcesz wyświetlić pliki znajdujące się w jego katalogu głównym *C:* przy użyciu udziału administracyjnego *C$*, możesz to bez problemu zrobić zdalnie z poziomu komputera lokalnego (listing 8.22).

Listing 8.22. Wyświetlanie listy plików za pośrednictwem udziału UNC

```
PS> Get-ChildItem -Path '\\dc\c$'
    Directory: \\dc\c$
Mode                        LastWriteTime     Length     Name
----                        -------------     ------     ----
d-----                10/1/2019 12:05 PM                FileShare
d-----               11/24/2019  2:28 PM                inetpub
d-----               11/22/2019  6:37 PM                InstallWindowsFeature
d-----                4/16/2019  1:10 PM                Iperf
```

Problem pojawia się, gdy utworzysz sesję `PSSession` do innego komputera i spróbujesz ponownie uruchomić to samo polecenie, tak jak pokazano na listingu 8.23.

Listing 8.23. Próba dostępu do udziałów sieciowych z poziomu sesji zdalnej

```
PS> Enter-PSSession -ComputerName WEBSRV1
[WEBSRV1]: PS> Get-ChildItem -Path '\\dc\c$'
ls : Access is denied
--pozostałe wiersze zostały pominięte--
[WEBSRV1]: PS>
```

W takim przypadku powłoka PowerShell informuje, że dostęp jest zabroniony — nawet jeżeli wiesz, że Twoje konto użytkownika ma dostęp do tego zasobu. Dzieje się tak, ponieważ podczas korzystania z domyślnego uwierzytelniania Kerberos usługi zdalne powłoki PowerShell nie przekazują tych poświadczeń do innych zasobów sieciowych. A więc PowerShell nie wykonuje przeskoku do kolejnego zasobu sieciowego. Powłoka PowerShell działa zgodnie z mechanizmami systemu Windows i ze względów bezpieczeństwa odmawia delegowania tych poświadczeń, w wyniku czego zwraca komunikat o odmowie dostępu.

Podwójny przeskok z uwierzytelnianiem CredSSP

W tej sekcji dowiesz się, jak ominąć problem podwójnego przeskoku — celowo używamy tutaj określenia „ominąć" zamiast „naprawić". Firma Microsoft ostrzega, że korzystanie z uwierzytelniania CredSSP może stanowić poważny problem bezpieczeństwa, ponieważ poświadczenia przekazane do pierwszego komputera są automatycznie używane dla wszystkich połączeń z tego komputera. Oznacza to, że jeżeli bezpieczeństwo tego komputera zostanie naruszone, to przejęte poświadczenia mogą zostać użyte do łączenia się z innymi komputerami w sieci. Nie zmienia to jednak w niczym faktu, że choć istnieje kilka innych, mniej lub bardziej złożonych rozwiązań tego problemu, takich jak mechanizm KDC (ang. *Kerberos Constrained Delegation*), wielu użytkowników decyduje się na wykorzystanie uwierzytelniania CredSSP, ponieważ jest ono łatwe w użyciu.

Przed zastosowaniem CredSSP powinieneś go włączyć zarówno po stronie klienta, jak i na serwerze za pomocą polecenia `Enable-WsManCredSSP` wykonanego w sesji powłoki PowerShell z poziomu administratora. To polecenie posiada parametr `Role`, który pozwala określić, czy uwierzytelnianie CredSSP jest włączane po stronie klienta, czy po stronie serwera. Najpierw powinieneś włączyć CredSSP po stronie klienta, jak pokazano na listingu 8.24.

Listing 8.24. Włączanie obsługi uwierzytelniania CredSSP po stronie klienta

```
PS> Enable-WSManCredSSP ❶-Role ❷Client ❸-DelegateComputer WEBSRV1
CredSSP Authentication Configuration for WS-Management
CredSSP authentication allows the user credentials on this computer to be
➥sent to a remote computer. If you use CredSSP authentication for
➥a connection to a malicious or compromised computer, that machine will
➥have access to your username and password. For more information, see the
➥Enable-WSManCredSSP Help topic.
Do you want to enable CredSSP authentication?
[Y] Yes   [N] No   [S] Suspend   [?] Help (default is "Y"): y

cfg           : http://schemas.microsoft.com/wbem/wsman/1/config/client/auth
lang          : en-US
Basic         : true
Digest        : true
Kerberos      : true
Negotiate     : true
Certificate   : true
CredSSP       : true
```

Uwierzytelnianie CredSSP po stronie klienta możesz włączyć, przekazując wartość `Client` ❷ do parametru `Role` ❶. Oprócz tego używamy również wymaganego parametru `DelegateComputer` ❸, ponieważ powłoka PowerShell musi „wiedzieć", które komputery będą mogły korzystać z delegowanych poświadczeń. Aby zezwolić na delegowanie poświadczeń do wszystkich komputerów, możesz użyć symbolu gwiazdki (*), ale ze względów bezpieczeństwa lepiej to będzie

zrobić tylko dla komputerów, z którymi będziesz pracować, w tym przypadku jest to serwer WEBSRV1.

Po włączeniu uwierzytelniania CredSSP na kliencie musisz zrobić to samo na serwerze (listing 8.25). Na szczęście zamiast nawiązywać połączenie typu Remote Desktop czy korzystać z fizycznej konsoli serwera, możesz po prostu otworzyć nową sesję zdalną, a następnie włączyć uwierzytelnianie CredSSP z poziomu tej sesji.

Listing 8.25. Włączanie uwierzytelniania CredSSP na serwerze zdalnym

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { Enable-WSManCredSSP
➥-Role Server }
CredSSP Authentication Configuration for WS-Management
CredSSP authentication allows the server to accept user credentials from
➥a remote computer. If you enable CredSSP authentication on the server,
➥the server will have access to the username and password of the client
➥computer if the client computer sends them. For more information, see the
➥Enable-WSManCredSSP Help topic.
Do you want to enable CredSSP authentication?
[Y] Yes [N] No [?] Help (default is "Y"): y

#text
-----
False
True
True
False
True
Relaxed
```

Dzięki temu, że włączyłeś uwierzytelnianie CredSSP zarówno po stronie klienta, jak i na serwerze, klient może delegować poświadczenia użytkownika do serwera zdalnego, a serwer zdalny będzie w stanie je obsłużyć. Teraz możesz spróbować ponownie uzyskać dostęp do zdalnych zasobów sieciowych z poziomu sesji zdalnej (listing 8.26). Pamiętaj, że jeżeli kiedykolwiek będziesz musiał wyłączyć uwierzytelnianie CredSSP, możesz to zrobić przy użyciu polecenia Disable-WsmanCredSSP.

Listing 8.26. Dostęp do zasobów sieciowych z poziomu sesji uwierzytelnionej za pomocą CredSSP

```
PS> Invoke-Command -ComputerName WEBSRV1 -ScriptBlock { Get-ChildItem -Path
➥'\\dc\c$' } ❶-Authentication Credssp ❷-Credential (Get-Credential)
cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential

    Directory: \\dc\c$
Mode                LastWriteTime     Length     Name
➥PSComputerName
```

```
----                 -------------     ------     ----                     ----
↪-----------
d-----         10/1/2019 12:05 PM                FileShare
↪WEBSRV1
d-----         11/24/2019 2:28 PM                inetpub
↪WEBSRV1
d-----         11/22/2019 6:37 PM                InstallWindowsFeature
↪WEBSRV1
d-----          4/16/2019 1:10 PM                Iperf
↪WEBSRV1
```

Zauważ, że musisz wyraźnie wskazać, że polecenie `Invoke-Command` (lub `Enter-PSSession`) powinno używać uwierzytelniania CredSSP ❶, a oba polecenia — któregokolwiek z nich używasz — wymagają przekazania poświadczeń, które możesz pozyskać, używając polecenia `Get-Credential` ❷ (zamiast domyślnego protokołu Kerberos).

Po wykonaniu polecenia `Invoke-Command` i przekazaniu za pomocą polecenia `Get-Credential` nazwy konta użytkownika i hasła dostępu do udziału `c$` na serwerze `DC` możesz się przekonać, że polecenie `Get-ChildItem` działa zgodnie z oczekiwaniami!

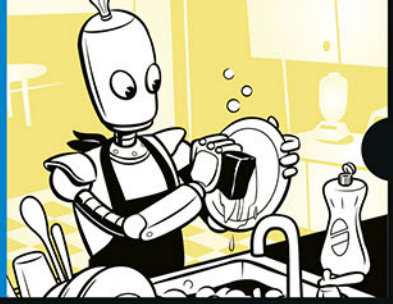
Podsumowanie

Korzystanie z usług zdalnych powłoki PowerShell to zdecydowanie najłatwiejszy sposób wykonywania poleceń w systemach zdalnych. Jak mogłeś się sam przekonać w tym rozdziale, usługi zdalne powłoki PowerShell są łatwe w użyciu i całkiem intuicyjne. Kluczem do sukcesu jest dobre zrozumienie koncepcji bloku skryptów i reguł wykonywania znajdującego się w nich kodu — a przy odrobinie chęci zdalne wykonywanie kodu stanie się dla Ciebie drugą naturą.

W części III tej książki — gdzie zbudujesz swój własny, niezawodny moduł powłoki PowerShell — będziemy używać usług zdalnych tej powłoki w prawie każdym poleceniu. Jeżeli zatem miałeś jakieś problemy ze zrozumieniem tego rozdziału, powtórz go lub spróbuj samodzielnie poeksperymentować ze zdalnym wykonywaniem poleceń. Wypróbuj różne scenariusze, spróbuj coś zepsuć, potem napraw to i zrób wszystko, co w Twojej mocy, aby zrozumieć działanie usług zdalnych powłoki PowerShell. To jedna z najważniejszych umiejętności, których możesz się nauczyć z tej książki.

W rozdziale 9. omówimy inne ważne zagadnienie: testowanie kodu za pomocą pakietu Pester.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion



OSZCZĘDŹ CZAS. AUTOMATYZUJ. URUCHOM POWERSHELL!

Nazwą PowerShell określa się dwa powiązane składniki: powłokę wiersza poleceń, instalowaną domyślnie praktycznie we wszystkich nowoczesnych systemach operacyjnych, oraz język skryptowy powłoki. Oba te elementy tworzą potężne i uniwersalne narzędzie, którego możesz używać do automatyzacji niemal wszystkiego: od szybkiego restartu setki serwerów po zbudowanie kompletnego systemu kontrolującego centrum danych. Aby nie tracić czasu na żmudne, powtarzalne obowiązki administratora, musisz tylko biegle posługiwać się konsolą PowerShell.

To praktyczny podręcznik dla administratorów systemów i inżynierów, którzy chcą zautomatyzować zadania związane z utrzymywaniem środowisk serwerowych, prowadzeniem testów albo automatyzacją potoków ciągłej integracji. Zawiera wprowadzenie do języka powłoki PowerShell, informacje o sposobach uruchamiania poleceń na zdalnych komputerach, a także techniki pracy z typowymi domenami. Pokazuje również, w jaki sposób można budować własne moduły PowerShell, aby zastosować je do automatyzacji całych laboratoriów lub środowisk testowych, tworzenia maszyn wirtualnych Hyper-V, instalacji systemów operacyjnych oraz wdrażania i konfigurowania serwerów IIS i SQL.

Teorię uzupełnia mnóstwo przykładów kodu, wskazówek i wyjaśnień, ułatwiających rozpoczęcie samodzielnej automatyzacji zadań administracyjnych.

W książce między innymi:

- wszechstronne korzystanie ze środowiska testowego PowerShell Pester
- analiza danych strukturalnych, praca z Active Directory, Azure i AWS
- projektowanie i budowa modułu PowerShell
- automatyzacja wdrożeń systemu Windows
- proste wdrażanie i konfiguracja serwerów WWW i SQL

Adam Bertram od ponad dwudziestu lat zajmuje się zawodowo IT, świetnie zna się też na prowadzeniu biznesu w sieci. Jest przedsiębiorcą i znakomitym informatykiem, uhonorowanym przez Microsoft tytułem MVP, a także trenerem, autorem książek i treści marketingowych dla wielu firm technologicznych. Założył popularną platformę rozwoju kariery IT TechSnips.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-7291-7
9 788328 372917
Cena: 69,00 zł
no starch press