

In diesem Kapitel lernen Sie die Windows PowerShell von Microsoft (im Folgenden einfach nur „PowerShell“ genannt) als eine moderne Alternative zur klassischen Windows-Befehlszeile, den Stapeldateien und dem Windows Scripting Host (WSH) kennen. Es wird deutlich werden, dass die PowerShell sehr viel mehr ist als eine klassische „Command-Shell“ und Administratoren durch sie Möglichkeiten erhalten, die über das Eingeben von Befehlen und Ausführen von Skripten hinausgehen. Diese Möglichkeiten bestehen z. B. in dem Einbeziehen unterschiedlicher Datenquellen als Teil einer Befehlskette (etwa Daten, die das Betriebssystem liefert, die Ergebnisse von Datenbankabfragen, Rückgabewerte von Webservice-Aufrufen usw.), dem Verknüpfen der abgerufenen Daten und dem Umwandeln in Standardformate (CSV, HTML, XML oder JSON), die eine Weiterarbeitung und Darstellung der abgerufenen Daten in anderen Anwendungen erlauben. Die PowerShell ist zudem weniger eine klassische Konsolenanwendung, sondern eher ein „Funktionsbaustein“, der in unterschiedliche Host-Anwendungen eingebaut wird. Die PowerShell ist daher ein „polymorphes Werkzeug“, das sich in unterschiedlichen Situationen und Umgebungen einsetzen lässt. Als klassische Command-Shell genauso wie als Teil einer Windows- oder Webanwendung oder als ein Windows-Workflow, der über einen längeren Zeitraum eine festgelegte Folge von Anweisungen ausführt. Über PowerShell-Remoting besteht die Möglichkeit, einen PowerShell-Prozess auf anderen Computern im Netzwerk zu starten und damit Befehle und Skripte auf diesen Computern auszuführen. Dank PowerShell-Remoting wird die Reichweite einer administrativen Lösung daher auf das gesamte Netzwerk erweitert. Anders als die klassische Command-Shell `cmd.exe`, die seit ihrer Einführung mit Windows NT im Jahr 1993 lediglich ein Zubehörprogramm unter vielen ist, besitzt die PowerShell bei Microsoft auch eine strategische Bedeutung. Es ist ein erklärtes Ziel des Konzerns, dass ein Großteil der Funktionalität von Windows Server per PowerShell ansprechbar sein soll. Damit lassen sich Automatisierungsszenarien auch auf jene Bereiche ausweiten, wie z. B. das Bereitstellen von Windows-Installationen im

Unternehmensnetzwerk, die bislang nur mit speziellen Befehlszeilentools oder über die GUI abbildbar waren. Es spricht daher vieles dafür, dass die PowerShell schnell zu einem Werkzeug werden wird, das Sie viele Jahre begleiten wird.

2.1 Wie alles anfang – ein kurzer Blick zurück

Am Anfang stand eine Vision. *Jeffrey Snover*, ein Entwicklungsingenieur bei Microsoft in Redmond, hatte eine klare Vorstellung davon, wie ein Verwaltungswerkzeug aussehen müsste, das auch in Zukunft allen Ansprüchen der modernen Windows Server-Administration gerecht werden würde. Motiviert wurde er vermutlich durch die in diesem Jahr erschienene Anwendungsplattform .NET Framework¹ und den Umstand, dass Microsoft diesen wichtigen Bereich viele Jahre vernachlässigt hatte und die Konkurrenz, vor allem in Gestalt von Linux, in diesem Punkt einiges mehr zu bieten hatte.

Die fünf Forderungen, die *Mr. Snover* bereits im Jahre 2002 formuliert hatte, hat er in seinem *Monad-Manifesto* zusammengefasst („Monad“ war der erste Codename der späteren PowerShell²), das unter <http://www.jsnover.com/blog/2011/10/01/monad-manifesto> als PDF-Datei zur Verfügung steht. In seinem Papier beschreibt *Mr. Snover* fünf Punkte, die eine künftige „Universal-Shell“ als Eigenschaften besitzen sollte:

1. Ein Automatisierungsmodell (das auf dem .NET Framework basiert).
2. Eine Command Shell (die ebenfalls auf dem .NET Framework basiert).
3. Ein Managementmodell, das Klassen zur Verfügung stellt, mit deren Hilfe sich im Alltag häufig auftretende Anforderungen (z. B. Authentifizierung bei einem Remote-Zugriff) umsetzen lassen.
4. Remote-Scripting auf der Basis von Webservice-Aufrufen.
5. Eine (natürlich auf dem .NET Framework) basierende Managementkonsole als Nachfolger der mit Windows 2000 eingeführten Computermanagementkonsole.

Von den fünf Punkten sind vier in der aktuellen Version der PowerShell bereits umgesetzt. Lediglich ein Nachfolger der betagten Computermanagementkonsole (MMC) wurde bis heute nicht realisiert³. Da *Mr. Snover* inzwischen als „Lead Architect“ die Windows Server-Gruppe bei Microsoft leitet, könnte aus einer neuen Managementkonsole auf der Basis von PowerShell-Befehlen vielleicht noch etwas werden.

Die erste Version der PowerShell erschien im November 2006. Auch wenn sich am Kern bis heute nichts grundsätzlich geändert hat, dürfte damals noch nicht abzusehen gewesen sein, welchen Funktionsumfang die PowerShell ein paar Jahre später besitzen

¹ Microsofts Antwort auf Java.

² Namensähnlichkeiten zum Namen des Autors dieses Buches sind rein zufällig.

³ Zeitweise war ein solcher Nachfolger unter dem Codenamen „Aspek“ in Arbeit, doch das Projekt wurde offenbar wieder eingestellt.

würde. Umfasste die Version 1.0 bescheidene 129 interne Befehle (Cmdlets), stehen bei Windows Server 2012 R2 und Windows 8.1 aktuell ca. 2500 PowerShell-Befehle als Teil des Betriebssystems zur Verfügung (wenn alle Features des Betriebssystems installiert wurden). Auch wenn Quantität nicht automatisch Qualität bedeuten muss, macht die Zahl eindrucksvoll deutlich, wie funktional reichhaltig die PowerShell als Werkzeug für Administratoren in der aktuellen Windows-Version geworden ist. Dabei muss berücksichtigt werden, dass jeder PowerShell-Befehl eine einheitliche Syntax besitzt, die Hilfe zu jedem Befehl Beispiele umfasst und automatisch zur Verfügung steht, sobald das entsprechende Windows Feature hinzugefügt wurde. Der Einarbeitungsaufwand ist damit deutlich geringer als es die große Zahl an Befehlen eventuell suggeriert Tab. 2.1.

2.2 Die Stärken der PowerShell

Als die Version 1.0 der PowerShell im Herbst 2006 auf der Bildfläche erschien, war es nicht so, dass es bis dahin ein Mangel an Administrationswerkzeugen gegeben hätte. Mit *Cmd.exe*, den darauf aufbauenden Stapeldateien und dem *Windows Scripting Host* (WSH) waren zwei Automatisierungswerkzeuge bei Windows bereits fest eingebaut. Durch populäre Tools wie *AutoIt* und *Kixtart*, und verschiedene proprietäre Tools wurde das Angebot abgerundet. Wem das nicht genügte, der konnte *PerlScript*, *ooREXX* oder die Unix-Tools

Tab. 2.1 Die Versionsgeschichte der Windows PowerShell

Version	Erscheinungstermin	Besonderheiten
1.0	November 2006	Die erste Version besaß noch wenige Cmdlets, so dass sie nicht viel mehr als eine moderne Form der Eingabeaufforderung war
2.0	November 2009	Mit dieser Version kamen wichtige Eigenschaften dazu, vor allem PowerShell-Remoting
3.0	Mai 2012	Mit dieser Version kam die Workflow-Unterstützung hinzu, Verbesserungen bei WMI und zahlreichen Cmdlets und eine vereinfachte Abfragesyntax bei Where-Object
4.0	Oktober 2013	Mit dieser Version setzt die PowerShell mit der „Desired State Configuration“ (DSC) einen weiteren Schwerpunkt, der die Art und Weise, wie in Zukunft Server in großer Zahl in einem Netzwerk konfiguriert werden, stark verändern könnte
5.0	Mai 2014 als Vorabversion	Mit dieser Version führt Microsoft erstmalig eine Paketverwaltung (OneGet) ein und verbessert DSC. Diese Version soll, wie auch künftige Versionen, unabhängig von einer Windows Server-Version veröffentlicht werden ^a

^a Wenn Sie diese Zeilen lesen, könnte die Version bereits offiziell verfügbar sein

im Rahmen von *CygWin* benutzen. Eine weitere Alternative zu den etablierten Werkzeugen musste daher ein paar innovative Eigenschaften besitzen.

Es sind genau drei Stärken, die die PowerShell attraktiv machen:

1. Die Entwickler der PowerShell haben sich einfache, aber wirksame Konventionen ausgedacht, z. B. für die Namensvergabe von Befehlen, die das Einarbeiten in die PowerShell deutlich erleichtern. Diese tragen zur Konsistenz bei der Befehlssyntax bei, die wiederum die Einarbeitung erleichtert.
2. Viele Funktionen sind „selbst entdeckbar“ (im Original heißt dieses Merkmal „Auto Discovery“). Darunter fallen Kleinigkeiten wie der Umstand, dass die Namen von Befehlen, deren Parameter und teilweise auch deren mögliche Werte per [Tab]-Taste abgefragt werden können. Dazu zählt der Umstand, dass Befehle durch Metadaten ergänzt werden, über die sich z. B. Informationen über ihre Parameter abrufen lassen.
3. Anders als die vielen kleinen Befehlszeilentools, die es bei Windows Server seit vielen Versionen gibt, besitzt die PowerShell auch eine strategische Bedeutung. Das bedeutet unter anderem, dass die PowerShell Bestandteil aller Windows Server-Produkte ist und kontinuierlich weiterentwickelt wird.

Ein weiterer Aspekt, der inzwischen für die PowerShell spricht, ist der Umstand, dass es sehr viel Know-how im Internet gibt. Um es einmal überspitzt zu formulieren: Es dürfte schwer sein auf eine Problemstellung zu kommen, für die nicht irgendjemand aus der weltweiten PowerShell-Community, die aber eher ein loser Verbund als ein organisiertes Gebilde ist, bereits eine Lösung gefunden und im Rahmen seines Blogs oder auf den bekannten PowerShell-Skriptportalen veröffentlicht hat.

- **Tipp** Eine sehr gute Informationsquelle ist das englischsprachige *PowerShell-Magazine* (<http://www.powershellmagazine.com>).

2.2.1 Beispiele für die Konsistenz der PowerShell-Befehle

Wie vorteilhaft sich die Konsistenz der PowerShell-Befehlssyntax auswirkt, in dem sie die Eingabe von Befehlen und das Erlernen der Befehlssyntax erleichtert, sollen die folgenden Beispiele für Systemabfragen deutlich machen. Im Folgenden werden eine Reihe von Abfragen vorgestellt, die sich auf unterschiedliche Themenbereiche beziehen. Dabei geht es nur um den allgemeinen Aufbau eines Befehls. Die Details werden in den folgenden Kapiteln vorgestellt.

Der erste Befehl gibt die Eckdaten zu allen Prozessen zurück, die aktuell mehr als 50 MB Arbeitsspeicher belegen:

```
| Get-Process | Where-Object WS -gt 50MB
```

Der nächste Befehl gibt die Eckdaten zu allen Systemdiensten zurück, die aktuell nicht laufen:

```
| Get-Service | Where-Object Status -ne "Running"
```

Der folgende Befehl listet alle Hyper-VMs auf, die aktuell ausführen:

```
| Get-VM | Where-Object Status -eq "Running"
```

Der folgende Befehl listet ebenfalls virtuelle Maschinen auf, dieses Mal aber basierend auf einer VMWare-Virtualisierung:

```
| Get-VM | Where-Object Status -eq "Running"
```

Wenn die vier vorgestellten Befehle eines gemeinsam haben, dann dass sie sich bezüglich ihrer Schreibweise sehr ähnlich sind.

Der folgende Befehl entspricht dem ersten Befehl, nur dass dieses Mal die erweiterte Abfragesyntax verwendet wird, die bei der PowerShell 2.0 die einzige Option war:

```
| Get-Process | Where-Object { $_.WS -gt 50MB }
```

Ab der Version 3.0 steht bei *Where-Object* alternativ die bereits vorgestellte vereinfachte Schreibweise zur Auswahl.

Auch wenn die Details zu den einzelnen Befehlen an dieser Stelle noch keine Rolle spielen sollen, haben die Beispiele eines deutlich gemacht: Die Schreibweise ist in allen Fällen ähnlich bis sogar identisch, da es für PowerShell-Commands eine Reihe von Konventionen gibt. Dazu gehört vor allem die Namensregel, nach der sich der Name jedes Cmdlets aus zwei Bestandteilen zusammensetzt, die immer mit einem Bindestrich getrennt werden: Einem Verb bzw. verbähnlichen Wort, das eine Aktion bezeichnet, und einem Hauptwort, das den Gegenstand der Aktion benennt. Einfach und effektiv. Mehr zu den Konventionen erfahren Sie in Kap. 4, in dem die PowerShell-Commands vorgestellt werden.

2.3 Ein technischer Überblick über die PowerShell

Für reine Anwender (Administratoren, IT-Pros und alle, die die PowerShell in erster Linie als Anwendungswerkzeug sehen) ist die PowerShell eine Host-Anwendung (z. B. PowerShell-Konsole oder PowerShell ISE), die sie als klassische Blackbox verwenden. Man gibt Befehle ein, führt Skripte aus, das Innenleben spielt keine Rolle. Aus der Perspektive

eines Software-Entwicklers besteht die PowerShell aus mehreren .NET Framework-Assemblies (Dateien mit der Erweiterung *.Dll*, die sog. „Managed Code“ enthalten, der von der .NET-Laufzeit ausgeführt wird), die von einem Host-Programm geladen werden. Microsoft stellt mit *Powershell.exe* (Konsole) und *Powershell ISE.exe* (eine Windows-Anwendung) gleich zwei Hosts als Teil von Windows zur Verfügung. Auch andere Hersteller, kleine Softwarefirmen, Administratoren mit Entwicklerkenntnissen und theoretisch auch jeder „Hobby-Programmierer“ kann die PowerShell-Assemblies mit wenig Aufwand in seine C#- oder Visual Basic-Programme einbauen und damit die PowerShell-Befehle als Teil seiner Anwendung ausführen. Dieses „Hosting“ der PowerShell-Engine wird in Kap. 25 („PowerShell für Entwickler“) an einem kleinen Beispiel vorgestellt.

2.3.1 Die Rolle des .NET Frameworks (.NET-Laufzeit)

Das .NET Framework, auch mit .NET-Laufzeit oder einfach nur .NET bezeichnet (wobei der unscheinbare Punkt wie „dot“ ausgesprochen wird), ist der Unterbau der PowerShell. Das .NET Framework ist eine Laufzeitumgebung, die von Microsoft als Antwort auf die Bedrohung des Windows-Monopols durch *Java* entwickelt und im Jahr 2002, begleitet von einer großen Marketingkampagne, veröffentlicht wurde⁴. Heute verfolgt der Konzern bekanntlich andere Interessen (Stichwort: „Mobile and Cloud first“) und setzt auf andere Techniken, das .NET Framework gibt es als festen Bestandteil jeder Windows-Version aber nach wie vor. Es besteht im Kern aus einer virtuellen Maschine, der *Common Language Runtime* (CLR), die „Managed Code“ ausführt. Managed Code besteht aus Befehlen der Microsoft-Programmiersprache IL (*Intermediate Language*) und ist in einer Assembly-Datei enthalten. Die PowerShell basiert auf einer Reihe solcher Assemblies (u. a. *System.Management.Automation.dll*), womit sich der Kreis schließt. Die Entwickler der PowerShell haben sich damals für das .NET Framework als Unterbau entschieden, da mit der .NET-Laufzeit von Anfang an eine reichhaltige Anwendungsfunktionalität zur Verfügung steht, die sowohl von der PowerShell als auch von PowerShell-Skripten genutzt werden kann. Außerdem ist das direkte Einbetten von z. B. Programmbefehlen der Programmiersprachen C# oder Visual Basic in einem PowerShell-Skript möglich. Dies erweitert zum einen die Möglichkeiten von PowerShell-Anwendern und macht die PowerShell zum anderen auch für Entwickler attraktiv. Die Abhängigkeit der PowerShell zur .NET-Laufzeit war in der Anfangszeit insofern ein Problem, da die .NET-Laufzeit damals noch separat verteilt werden musste und nicht als gegeben vorausgesetzt werden konnte.

⁴ Java ist eine Kombination von Programmiersprache und Laufzeitumgebung, die Mitte der 90er Jahre von der Firma Sun veröffentlicht wurde und heute zu Oracle gehört. Java versprach damals, dass eine Anwendung nur einmal entwickelt werden musste und danach auf allen Plattformen und im Browser ausführen konnte. Wären alle Anwender auf Java umgestiegen, wäre Windows überflüssig geworden. Wie wir heute wissen, hat Java am „Windows-Monopol“ nicht allzu viel geändert.

Spätestens seit Windows Server 2008 R2 und Windows 7 ist sie ein fester Bestandteil des Betriebssystems und steht damit immer zur Verfügung.

Tabelle 2.2 stellt die PowerShell-Versionen der Version des .NET Framework gegenüber, auf der die jeweilige Version aufsetzt. Diese Abhängigkeit muss bei der Installation der PowerShell berücksichtigt werden.

2.3.2 PowerShell als Interpreter

PowerShell-Skripte werden grundsätzlich interpretiert ausgeführt. Das bedeutet konkret, dass ein PowerShell-Skript Befehl für Befehl abgearbeitet wird. Es gibt keinen Compiler, der aus PowerShell-Befehlen vor der Ausführung IL-Code oder gar Maschinencode macht (auch nicht von anderen Firmen). Allerdings ist die PowerShell seit der Version 3.0 kein klassischer Interpreter mehr. Jeder Befehl wird bereits während der Eingabe in einen sog. „Abstrakten Syntaxbaum“ (engl. „Abstract Syntax Tree“, kurz AST) zerlegt, der die Bestandteile des Befehls als Tokens (Symbole) enthält und bei der Ausführung Token für Token abgearbeitet wird. Bei der PowerShell ISE macht sich dieser Aspekt u. a. dadurch bemerkbar, dass Befehle bereits unmittelbar nach der Eingabe angezeigt und z. B. auch fehlende geschweifte Klammern in einem Skript als Fehler angezeigt werden.

2.3.3 PowerShell für andere Plattformen?

Die PowerShell gibt es für Windows ab Version XP SP3 und Windows Server 2003 in der Version 2.0 bzw. ab Windows 7 und Windows Server 2008 R2 für alle übrigen Versionen. Es gibt sie außerdem für Windows RT. Andere Plattformen werden nicht unterstützt. Ein vor vielen Jahren begonnenes Open Source-Projekt mit dem Namen „Pash“, das das Ziel hatte, die PowerShell (auf der Basis des zum .NET Framework kompatiblen Open Source Frameworks *Mono*) nach Linux zu portieren, kam über erste Ansätze nicht hinaus⁵. Von

Tab. 2.2 Die PowerShell-Versionen und die erforderlichen .NET Framework-Versionen

PowerShell	.NET Framework-Version
1.0	2.0
2.0	2.0
3.0	4.0
4.0	4.5
5.0	4.5

⁵ Inzwischen hat das von *Igor Moolchnik* ursprünglich gestartete Projekt einen neuen Besitzer und unter *GitHub* ein neues zu Hause gefunden und wird offenbar weiterentwickelt – <https://github.com/Pash-Project/Pash>.

Microsoft gibt es zur Zeit (Stand: Mai 2014) keinerlei Bestrebungen, die PowerShell auf andere Plattformen zu portieren⁶. Eine Ausnahme ist das Thema „Desired State Configuration“ (DSC).

Im Zusammenhang mit der *Open Management Infrastructure-Initiative* (OMI) hat Microsoft bereits vor einigen Jahren den Quellcode für einen CIM-Manager freigegeben, mit dem sich (WMI-) Abfragen und Befehle gegen alle Geräte ausführen lassen, auf denen dieser CIM-Manager implementiert wurde. Auf der Grundlage eines solchen OMI-CIM-Server hat Microsoft im Mai 2014 „Windows PowerShell Desired State Configuration for Linux“ angekündigt. Damit lassen sich im Rahmen der mit PowerShell 4.0 eingeführten Desired State Configuration (Kap. 20) Konfigurationseinstellungen auch auf Linux-Server übertragen.

2.3.4 Die Rolle des PowerShell-Hosts

Der (PowerShell-) Host ist die Anwendung, die die einzelnen PowerShell-Bibliotheken (Assemblies) unter einem Dach zusammenfasst und damit für den Anwender zugänglich macht. Abbildung 2.1 stellt den Zusammenhang zwischen dem PowerShell-Host und dem Unterbau in einer Übersicht dar. Microsoft stellt mit *Powershell.exe* (Konsole) und *PowerShell_ISE.exe* (Windows) zwei Hosts zur Verfügung. Es gibt eine Reihe weiterer Hosts, bei denen es sich in erster Linie um Alternativen zur PowerShell ISE handelt, wie *Power GUI*, *PowerGUI Script Editor*, *PowerShell Plus* und *PowerShell Studio*.

Jede Host bietet dieselbe PowerShell-Funktionalität an, implementiert aber unter Umständen nicht denselben Funktionsumfang, was z. B. die Möglichkeiten der Ausgabe betrifft.

2.4 Download und Installation

Microsoft stellt die Versionen 2.0 bis (aktuell) 4.0 als Updates zur Verfügung, die aber nicht automatisch verteilt, sondern von der Microsoft-Webseite heruntergeladen werden müssen. Der Download heißt aber nicht „PowerShell“, sondern „Windows Management Framework“ (WMF), das neben der PowerShell weitere Komponenten, wie Windows Remoteing (WinRM), umfasst (Abb. 2.2). Tabelle 2.3 stellt die einzelnen Downloads mit ihren KB-Nummern zusammen.

Der Umstand, dass man zwischen mehreren Versionen der PowerShell (in erster Linie 3.0 und 4.0) wählen kann und daher auf mehreren Arbeitsplätzen im Unternehmen unterschiedliche Versionen installiert sein können, wirft natürlich zahlreiche Fragen auf. Diese werden im Folgenden im Stile eines FAQ beantwortet. Soviel bereits vorab: „Probleme“

⁶ Wenngleich PowerShell-Erfinder *Jeffrey Snover* auf einer Community-Konferenz im Mai 2014 angedeutet hat, dass die PowerShell eines Tages Open Source werden könnte.

Die Rolle des PowerShell-Hosts

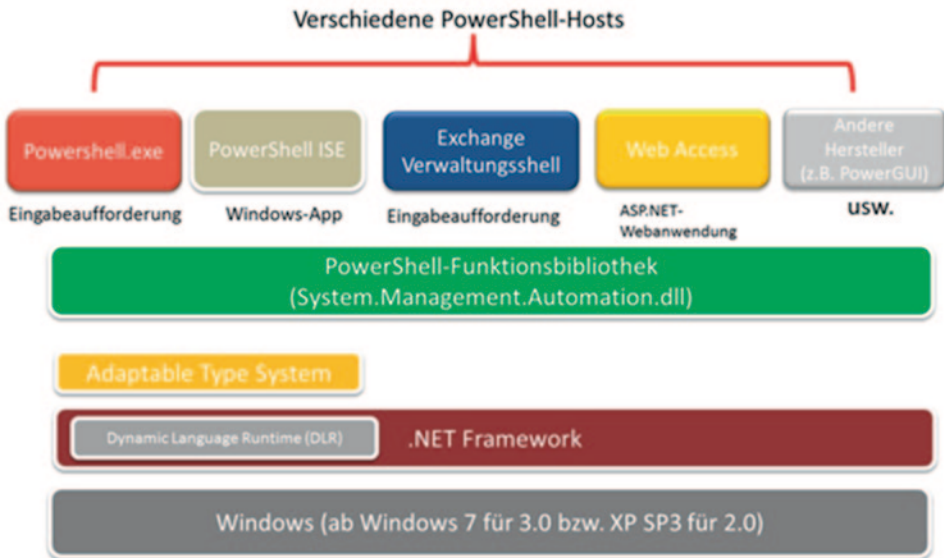


Abb. 2.1 Das Zusammenspiel zwischen Host und PowerShell-Funktionalität

Bestandteile des Windows Management Frameworks 3.0

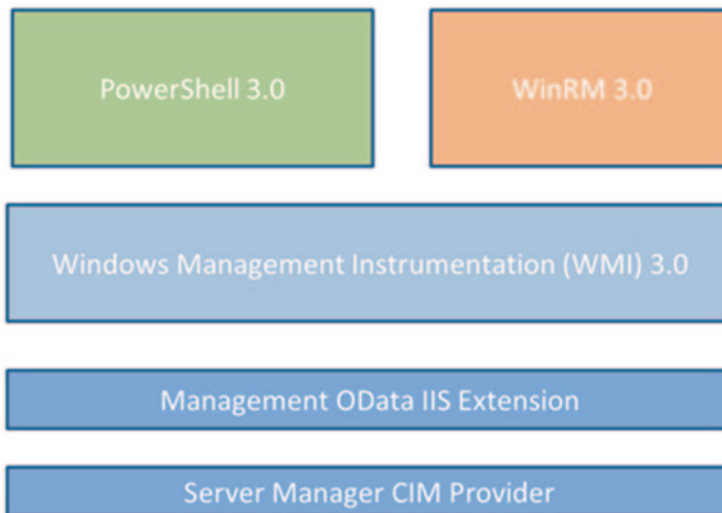


Abb. 2.2 Die Bestandteile des Windows Management Framework 3.0

Tab. 2.3 PowerShell-Downloads im Überblick

PowerShell-Version	KB-Nummer	Voraussetzungen
1.0	KB926140/KB926139	Windows XP, Windows Server 2003
2.0	KB968930	Windows XP SP3, Windows Server 2008
3.0	KB2506143/KB2506146	Windows 7 SP1, Windows Server 2008
4.0	KB2819745/KB2799888	Windows 7 SP1, Windows Server 2008 R2 SP1, Windows Server 2012

sollte der Umstand, dass ein Skript auf einem Arbeitsplatz von einer Version 2.0, auf dem anderen Arbeitsplatz von einer Version 4.0 der PowerShell ausgeführt wird, nur in seltenen Ausnahmefällen bereiten (und dann lassen sie sich auch mit einfachen Mitteln lösen).

Frage Kann ein mit der PowerShell 2.0 erstelltes Skript unter den Nachfolgeversionen ausgeführt werden?

Antwort Grundsätzlich ja. Mit jeder Nachfolgeversion gibt es eine Reihe von kleineren „breaking changes“ (also Änderungen, die dazu führen können, dass ein Skriptbefehl nicht mehr so funktioniert wie unter der Vorgängerversion und daher zu einem Fehler führen kann), doch sind dies Spezialfälle, die im Praxisalltag nur selten auftreten. Die „breaking changes“ werden von Microsoft in einer Begleitdatei der aktuellen Version dokumentiert.

Frage Kann ein mit der PowerShell 4.0 erstelltes Skript von der PowerShell 2.0 ausgeführt werden?

Antwort Grundsätzlich ja, es darf nur keine Befehle und Operatoren verwenden, die erst mit einer Nachfolgeversion eingeführt wurden. Die PowerShell bietet für diesen Fall das Schlüsselwort *#requires*, über das die Versionsnummer angegeben wird, die für die Ausführung eines Skripts vorausgesetzt wird. Dadurch kann der Fall nicht auftreten, dass ein Skript, das Befehle verwendet, die es bei der PowerShell 2.0 noch nicht gibt, von einem PowerShell 2.0-Host ausgeführt wird und die Fehler erst im Verlauf der Skriptausführung auftreten.

Frage Spielt die Plattform (32 und 64 Bit) für die Ausführung eines PowerShell-Skripts eine Rolle?

Antwort Nein, sofern keine plattformspezifischen Merkmale angesprochen werden.

PowerShell für die Windows-Administration

Ein kompakter und praxisnaher Überblick

Monadjemi, P.

2014, XXII, 521 S. 80 Abb., Softcover

ISBN: 978-3-658-02963-0