



Mit den richtigen PowerShell-Befehlen sichern Sie Ihren XenServer

Citrix XenServer 5.6 über die PowerShell sichern Backup per Skript

von Martin Pajonk

Citrix XenServer erfreut sich immer größerer Beliebtheit. Das liegt zum einen am kostenlosen Einstieg in die Virtualisierungswelt und zum anderen am durchaus überzeugenden Funktionsumfang. Doch gibt es derzeit noch zu wenige Alternativen im Backupbereich. Hier lassen sich die Hersteller ihre Lösungen teuer bezahlen – der geldwerte Vorteil von XenServer ist so schnell wieder aufgebraucht. Genau hier setzt eine Kombination aus Microsoft PowerShell und den kommandozeilenbasierten XenCenter-Tools an. Wie Sie damit virtuelle Systeme sichern und wiederherstellen, zeigt Ihnen dieser Workshop.

Die Sicherung und Wiederherstellung von XenServer-Snapshots ist mit ein wenig Handarbeit und ohne eine weitere Softwarelösung möglich. Hierzu wird lediglich die PowerShell, das Citrix XenCenter und das “XenServerPSSnapIn” benötigt. Dem zu Grunde liegt natürlich ein installierter Citrix XenServer inklusive der virtuellen Maschinen, die er beheimatet. Nachfolgend eine kurze Übersicht zu den verwendeten Komponenten.

Auf den aktuellen Microsoft Betriebssystemen, Windows Server 2008 R2 oder Windows 7, findet sich bereits die aktuelle Version der Windows PowerShell. Diese lässt sich aber auch für ältere Semester der OS-Familie problemlos von der Microsoft Website in der Version 2.0 herunterladen und nachinstallieren [1]. Die PowerShell stellt eine Reihe von nützlichen Commandlets (kurz Cmdlets) zur Verfügung und bietet die Möglichkeit, diese auch in Form von PowerShell Snap-Ins zu erweitern. Ein Cmdlet besteht aus einem Verb, einem

Substantiv und einer optionalen Parameterliste wie *Get-ChildItem*. Hilfe zu Cmdlets erhalten Sie entweder durch das Ausführen des Cmdlets *Get-Help {cmdlet} [-params]* oder auf der TechNet Seite von Microsoft [2].

Citrix XenServer und XenCenter installieren

Für die Erstellung der Skripte haben wir in unserem Workshop-Szenario zunächst einen Citrix XenServer in der Version 5.6 eingerichtet. Weiterhin wurden zu Testzwecken mehrere virtuelle Maschinen eingespielt. Das XenCenter läuft in Version 5.6 auf einer Windows 7-Maschine. Die dazugehörige grafische Benutzeroberfläche ermöglicht es, die eingetragenen XenServer und die dort beheimateten virtuellen Maschinen auf einfache Art und Weise zu überwachen und zu administrieren. Sowohl den XenServer als auch das XenCenter können Sie über die Citrix-Seite [3] beziehen. Sollte Hilfe bei der Installation vonnöten sein, ist diese ebenfalls auf der Citrix Seite zu finden [4].

Bei der Installation des XenServers wird auf dem Host automatisch ein Kommandozeilen-basiertes Tool namens “xe” mit installiert. Dieses gibt es auch als eigenständiges Remote-Tool für Windows. Für Windows-Clients wird es bei der Installation von XenCenter aufgespielt und befindet sich im üblichen Installationspfad des XenCenters “C:\Programme\Citrix\XenCenter”. Unter Linux ist der Installer als rpm-Paket verfügbar. Darüber hinaus ist “xe” Bestandteil der Serverinstallations-CD und kann von dieser installiert werden. Zur Vereinfachung der Benutzung sollten Sie den Installationspfad von “xe” in die globale PATH-Variable eintragen. Eine umfangreiche Dokumentation zum Command Line Interface finden Sie unter [5].

Das PowerShell Snap-In für XenServer wird uns im weiteren Verlauf bei der Auswahl der zu sichernden virtuellen Maschinen behilflich sein. Es steht auf der Citrix Community-Seite zum Download bereit [6].

XenServer PowerShell-Snap-In

Das “XenServerPSSnapIn” besteht aus einer umfangreichen Sammlung von Cmdlets, die für die einfache Administration von XenServern mit Hilfe der PowerShell gedacht sind. In der Kombination mit dem CLI-Tool “xe” können Sie damit praktisch alle denkbaren Aufgaben automatisieren. Vor der Installation sollten Sie jedoch die Rechte innerhalb der PowerShell anpassen – weitere Informationen hierzu erhalten Sie durch die Eingabe von *Get-Help about_execution_policies*. Für unser Beispiel setzen wir die Rechte zur Ausführung von Skripten durch *Set-ExecutionPolicies RemoteSigned*. Die hier bereitgestellten Cmdlets halten sich leider nicht an den vorgeschriebenen Standard für die Benennung. Als Grund werden potenzielle Konflikte mit Konkurrenzprodukten angeführt. Falls Sie keine derartigen Konflikte befürchten, finden Sie im Installationspfad das Skript *Initialize-Aliases.ps1*, das die inkorrekte Namensgebung wieder geradebiegt. Im weiteren Verlauf verwenden wir die nicht standardkonformen Bezeichnungen, da sie in beiden Fällen immer funktionieren sollten. Nach der Installation des XenServerPSSnapIn ist es erforderlich, das Snap-In in der Konsole bekannt zu machen:

```
PS H:\>Get-PSSnapIn -Registered
```

```
Name: PowerGUI
PSVersion: 1.0
Description:
```

```
Name: XenServerPSSnapIn
PSVersion: 2.0
```

```
Description: Citrix XenServer PowerShell SnapIn
```

Verlief die Installation erfolgreich, erscheint nach der Eingabe von *Get-PSSnapIn -Registered* das XenServerPSSnapIn auf der Konsole. Nun muss es nur noch in der PowerShell angemeldet werden. Das erfolgt durch die Eingabe von *Add-PSSnapIn XenServerPSSnapIn*. Eine erneute Kontrolle durch die Eingabe von *Get-PSSnapIn* zeigt eine Liste

der in der aktuellen PowerShell angemeldeten Snap-Ins. Das XenServerPSSnapIn kommt mit vielen Cmdlets daher. Sie können sie sich alle durch die Eingabe von *Get-Command -Module XenServerPSSnapIn* anzeigen lassen. Nachfolgend eine kurze Beschreibung der drei Cmdlets, die in den Backup- und Restore-Skripten Verwendung finden.

- Connect-XenServer: Wie der Name erahnen lässt, dient dieses Cmdlet dazu, eine Verbindung mit dem XenServer aufzubauen. Als Parameter werden die URL, der Benutzer und das Passwort benötigt. Weitere Parameter und Informationen zu diesem Cmdlet erhalten Sie durch die Eingabe von *Get-Help Connect-XenServer*. Eine erfolgreiche Verbindung hat die Rückgabe eines Xen-API.XenObject-Objekts zur Folge. Die genaue Syntax finden Sie sowohl im weiteren Verlauf des Artikels als auch in der Hilfe zu Connect-XenServer.
- Get-XenServer:VM: Mit Hilfe dieses Cmdlets werden alle auf dem XenServer sich befindenden virtuellen Maschinen ausgegeben, inklusive Templates, Snapshots und Controll Domains. Zu beachten ist hierbei, dass zuvor eine Verbindung mit dem XenServer mittels Connect-XenServer aufgebaut wurde. Wie später zu sehen ist, kann mit Hilfe des “Where-Object”-Cmdlets und einer Pipe auf simple Art und Weise die Ausgabe, der Output des Get-XenServer:VM Cmdlets gefiltert werden. Auch hier liefert das Cmdlet ein Objekt zurück, diesmal vom Typ “System.Array”.
- Disconnect-XenServer: Über das dritte Cmdlet müssen nicht viele Worte verloren werden. Es schließt eine bestehende Verbindung mit einem XenServer, der über den URL-Parameter spezifiziert werden kann.

Entwicklungsumgebung und Aufbau der Skripte

Für das Erstellen der Skripte können Sie jeden beliebigen Texteditor verwenden. Trotzdem ist zu erwähnen, dass Microsoft der PowerShell eine recht nützliche Entwicklungsumgebung – das “Windows

PowerShell Integrated Scripting Environment” oder kurz ISE – spendiert hat. Dieses verfügt sogar über rudimentäre Debuggingfunktionen, womit das Auffinden von Fehlern um einiges vereinfacht wird. Als eine Alternative bietet sich zudem die PowerGUI an, die mit Code-Vervollständigung und einer Anzeige der aktuellen Variablen daherkommt. Doch nun zu den Skripten selbst. Diese sind auf vier Dateien verteilt worden.

1. *config.ps1*
2. *xen.ps1*
3. *backup.ps1*
4. *restore.ps1*

Die Namensgebung der einzelnen Dateien verrät bereits, wofür die einzelnen Skripte gut sind. In der Datei *config.ps1* sind alle benötigten Informationen hinterlegt, um mit einem speziellen Server Kontakt aufzunehmen oder wo die zu speichernden Dateien abgelegt werden sollen. Die Pfade beziehen sich auf den Host, auf dem auch die Skripte aufgerufen werden. Die Zeile “Set-PSDebug -Strict” dient dazu, nicht initialisierte Variablen zu melden, so dass leicht zu übersehende Tippfehler bei der Benennung von Variablen von vorneherein vermieden werden.

Die Datei *xen.ps1* bündelt alle benötigten Funktionen für die Sicherung und den Export von Snapshots und für die

```
# Variablen müssen vor Benutzung
# initialisiert werden
Set-PSDebug -strict

# citrix server
$_ip = '192.168.23.42'
$_url = 'http://' + $_ip
$_user = 'root'
$_pw = 'geheim'

# Sicherungspfad für Backupdateien
$_storePath = 'C:\STORE\PATH'

# Beinhaltet wiederherzustellende
# Snapshots
$_restorePath = 'C:\PATH\TO\FILES'

# Logging Pfad und Dateiname
$_logPath = 'C:\PATH\TO\LOG\STORAGE'
$_logFileName = 'my.log'
```

Listing 1: *config.ps1*





```

$_scriptRoot = "C:\PATH\TO\SCRIPTS\"

# externe Skripte includieren
. ($_scriptRoot + "config.ps1")
. ($_scriptRoot + "xen.ps1")

trap
{
    "Hier ist etwas furchtbar schief gegangen: " + $_
}

# Eine Liste zu sichernden VMS holen
$vmList = GetVmList "vm"

if ($vmList -ne "") {
    foreach ($vm in $vmList) {
        DoSnapshot $vm | Out-Null
    }
} else {
    throw "Leere VM Liste übergeben"
}

# Eine Liste zu exportierender Snapshots holen
$snapshotList = GetVmList "snapshot"

if ($snapshotList -ne "") {
    foreach ($snapshot in $snapshotList){
        ExportSnapshot $snapshot
    }
} else {
    throw "Leere Snapshotliste übergeben"
}

```

Listing 2: backup.ps1



Wiederherstellung von Templates und ist somit die zentrale Datei bei diesem Unterfangen. Die Skripte *backup.ps1* und *restore.ps1* bedienen sich der anderen beiden Skripte, um die ihnen angedachten Aufgaben zu erfüllen. In beiden wird einleitend zunächst der Skriptpfad gesetzt und anschließend die benötigten Dateien *config.ps1* und *xen.ps1* eingebunden.

Snapshot erstellen

Um Snapshots erstellen zu können, müssen Sie zunächst in Erfahrung bringen, wovon Snapshots gemacht werden sollen. Hierzu dient die Funktion "GetVmList" in der Datei *xen.ps1*. Innerhalb der Funktion legen Sie anhand des übergebenen Parameters fest, wonach gefiltert werden soll. Wie Sie dem Quellcode entnehmen können, sorgt der Parameter "vm" nach dem Verbindungsaufbau dafür, dass eine Liste jener Ressourcen zurückgegeben wird, die kein Snapshot, Template oder Kontrolldomäne sind. Im Detail geschieht dies durch das Anwenden des Cmdlets "Where-Object", das mit Hilfe des Pipeoperators an das "Get-XenServer:VM"-Cmdlet angehängt wird. Das Ergebnis des

Cmdlets "Get-XenServer:VM" wird dann an ein "Where-Object" übergeben, das anhand der an ihn weitergereichten Filterregeln die gewünschten Ergebnisse liefert. Beim "\$_"-Konstrukt handelt es sich um eine in PowerShell vordefinierte Variable, die auf das zuletzt übergebene Objekt zugreift. Durch den Punktoperator "." wird auf Eigenschaften des Objekts zugegriffen. Mit wenig Aufwand ist es auch problemlos möglich, die Auswahl der zu sichernden virtuellen Maschinen den eigenen Bedürfnissen anzupassen. Nach dem Zugriff auf den XenServer wird die Verbindung wieder geschlossen und das Ergebnis von der Funktion wieder zurückgegeben.

In der Signatur der Funktion werden nun die benötigten Parameter mit den Defaultwerten aus der Konfigurationsdatei besetzt. Durch die Angabe von konkreten Werten lassen sich über eine foreach-Schleife auch mehrere XenServer mit dieser Funktion nach potentiellen Snapshot-Kandidaten durchforsten. Die von "GetVmList" zurückgegebene Liste wird iteriert und von jedem Kandidaten nun mit Hilfe der Funktion "DoSnapshot" ein Snapshot auf dem Server angefertigt. In der Funktion "DoSnapshot" passieren zwei Dinge: Es wird der Name des Snapshots generiert und durch den Aufruf von

```

xe -s $ip -u $user -pw $pw vm-
snapshot vm=$uuid new-name-
label=$snapshotNameLabel

```

der Snapshot erzeugt. Als Ergebnis erhalten Sie eine uuid, die den erstellten Snapshot eindeutig identifiziert.

Snapshot exportieren

Ebenso wie beim Erzeugen der Snapshots muss dem Skript bekannt sein, welche Snapshots es exportieren soll. Hierzu nutzen Sie die Funktion "GetVmList". Dieses Backup-Skript gibt alle auf dem XenServer vorhandenen Snapshots zurück. Die so erhaltene Liste wird abermals durchlaufen und der Export der zu

vor erstellten Snapshots beginnt mit Hilfe der Funktion "ExportSnapshot". Der eigentliche Export der Snapshots findet in der Zeile

```

$exportResult = xe -s $ip -u $user
-pw $pw snapshot-export-to-template
snapshot-uuid=$uuid
filename=$fileName

```

statt. Dadurch wird ein Template des angelegten Snapshots erzeugt und als XVA-Image unter dem in "\$filename" angegebenen Dateinamen gespeichert.

```

function ExportSnapshot([XenAPI.XenObject] $snapshot,
[string] $ip = $_ip, [string] $user = $_user, [string]
$password = $_pw, [string] $storePath = $_storePath)
{
    if (!$snapshot) {
        throw "kein Snapshot für Export übergeben"
    }

    $uuid = $snapshot.uuid
    $fileName = $storePath + $snapshot.name_label + ".xva"

    $exportResult = xe -s $ip -u $user -pw $pw
    snapshot-export-to-template snapshot-
    uuid=$uuid filename=$fileName

    if ($exportResult -eq "Export succeeded") {
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $uuid + " - export - [ OK ]") -Encoding utf8 -Append
        RemoveSnapshot $snapshot $ip $user $pw
    } else {
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $uuid + " - export - [ FAIL ]") -Encoding utf8 -Append
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $uuid + " - export - " + $exportResult) -Encoding utf8
        -Append
    }
}

function RemoveSnapshot([XenAPI.XenObject] $snapshot,
[string] $ip = $_ip, [string] $user = $_user, [string]
$password = $_pw)
{
    if ($snapshot.is_a_snapshot) {
        $uuid = $snapshot.uuid
        $removeResult = xe -s $ip -u $user -pw $pw
        snapshot-uninstall snapshot-uuid=$uuid force=$true
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $uuid + " - remove - [ OK ]") -Encoding utf8 -Append
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $uuid + " - remove - " + $removeResult) -Encoding utf8
        -Append
    } else {
        Out-File -FilePath ($logPath + $_logFileName) -
        InputObject ("$(Get-Date -Format yyyyMddHHmmss) - " +
        $snapshot.uuid + " - remove - [ FAIL ]") -Encoding utf8
        -Append
    }
}

```

Listing 3: xen.ps1 (Teil 1)



```

function GetVmList([string] $trigger = "vm", [string] $url = $_url, [string] $user = $_user, [string] $pw = $_pw)
{
    Connect-XenServer -url $url -username $user -password $pw | Out-Null

    switch ($trigger.ToLower())
    {
        "vm" { $vmList = Get-XenServer:VM | Where-Object {!$_is_a_snapshot -and !$_is_a_template -and
            !$_is_control_domain} }
        "snapshot" { $vmList = Get-XenServer:VM | Where-Object {$_is_a_snapshot -eq $true} }
        default { throw "trigger [$trigger] in Funktionssignatur nicht definiert" }
    }

    Disconnect-XenServer -url $url

    return $vmList
}

function DoSnapshot([XenAPI.XenObject] $vm, [string] $ip = $_ip, [string] $user = $_user, [string] $pw = $_pw)
{
    if (!$vm) {
        throw "keine VM übergeben"
    }

    # Snapshotnamen mit Zeitstempel generieren
    [string]$snapshotNameLabel = "$(Get-Date -Format yyyyMMddHHmmss)_" + $vm.name_label

    $uuid = $vm.uuid
    # gib die erzeugte Uuid zurück
    return xe -s $ip -u $user -pw $pw vm-snapshot vm=$uuid new-name-label=$snapshotNameLabel
}

function ImportSnapshot([string] $fileName, [string] $ip = $_ip, [string] $user = $_user, [string] $pw = $_pw)
{
    if ($fileName -ne '') {
        $filePath = $_restorePath + $fileName

        $msg = "$(Get-Date -Format yyyyMMddHHmmss) - " + $filePath + " - import -"

        $uuid = xe -s $ip -u $user -pw $pw vm-import filename=$filePath
        if ($uuid -ne '') {
            Out-File -FilePath ($_logPath + $_logFileName) -InputObject ($msg + " [ OK ]") -Encoding utf8
            -Append
            CreateVm $uuid $fileName
        } else {
            Out-File -FilePath ($_logPath + $_logFileName) -InputObject ($msg + " [ FAIL ]") -Encoding utf8
            -Append
        }
    }
    else {
        throw "Kein Dateiname für Import eingegeben"
    }
}

function CreateVm([string] $uuid, [string] $fileName, [string] $ip = $_ip, [string] $user = $_user, [string] $pw = $_pw)
{
    if ($uuid -eq "") {
        throw "keine uuid angegeben"
    }

    if ($fileName -eq "") {
        throw "kein Dateinamen übergeben"
    }

    # Namen für neue VM erstellen
    $newNameLabel = $fileName.TrimEnd('.xva')

    $newUuid = xe.exe -s $ip -u $user -pw $pw vm-install new-name-label=$newNameLabel template-uuid=$uuid

    # erstellte VM starten
    xe.exe -s $ip -u $user -pw $pw vm-start vm=$newUuid
}

```

Listing 4: xen.ps1 (Teil 2 ab function GetVmList)



Nach dem erfolgreichen Export können Sie den Snapshot vom XenServer entfernen. Dies geschieht durch die Funktion "RemoveSnapshot". Das Löschen der Snapshots auf dem XenServer initiieren Sie durch den Aufruf der Zeile

```
xe -s $ip -u $user -pw $pw snapshot-uninstall snapshot-uuid=$uuid force=$true
```

Diese Zeile ist allerdings mit Vorsicht zu genießen. Normalerweise ist eine Bestätigung des Benutzers beim Ausführen dieses Kommandos vonnöten, um den Löschvorgang anzustoßen. Diese Sicherheitsabfrage schalten Sie im Skript durch den Zusatz "force=true" aus. Als nächster Punkt sei erwähnt, dass jede an das Kommando übergebene uuid vom Server entfernt wird, unabhängig davon, ob es sich dabei um einen Snapshot handelt oder nicht. So sollten Sie sich durch die Formulierungen innerhalb des abgesetzten Befehls *snapshot-uninstall snapshot-uuid* nicht auf der sicheren Seite wähnen – schließlich ist nicht davon auszugehen, dass etwas anderes als ein Snapshot gelöscht werden könnte. Nach Beendigung des Skriptes lohnt noch ein prüfender Blick in das angefertigte Logfile. In der *config.ps1* wurde es unter dem Namen *my.log* angelegt und befindet sich im Verzeichnis der exportierten Templates.

Snapshot wiederherstellen

Das Wiederherstellen der Snapshots aus den gesicherten Templates geht in zwei Schritten vor sich.

1. Die wiederherzustellenden Templates importieren.
2. Das Erstellen von virtuellen Maschinen aus den importierten Templates.

Im Skript *restore.ps1* werden zunächst alle Dateien mit Hilfe des Cmdlets "Get-ChildItem" in der Variable "\$backupFiles" abgelegt, die in *config.ps1* über "\$_restorePath" definiert sind. Dank des Cmdlets können Sie sich das aktuelle Verzeichnis anzeigen lassen, ganz im Stil des klassischen dir-Befehls. Nebenbei erwähnt



ist *dir* der Alias für *Get-ChildItem* in der PowerShell. Da für unsere Zwecke die Namen der vorhandenen Dateien völlig ausreichen, teilen wir dies dem Cmdlet über den Parameter “-Name” mit. Weiterhin sind nur die Dateien mit der Endung *.xva* interessant, was Sie durch ein *-Include *.xva* an das Cmdlet weitergeben. Für jede der von “*Get-ChildItem*” zurückgelieferten Dateien wird die Funktion “*ImportSnapshot*” aufgerufen. Durch den CLI-Aufruf *xe -s \$ip -u \$user -pw \$pw vm-import filename=\$filePath* werden die Templatedateien auf den XenServer hochgeladen und stehen ab dann bereit, um installiert zu werden.

Ist das Template hochgeladen, geht die Wiederherstellung einer virtuellen Maschine recht schnell vonstatten. Der CLI-Aufruf

```
xe.exe -s $ip -u $user -pw $pw
vm-install new-name-label=$newName
Label template-uuid=$uuid
```

stellt die virtuelle Maschine aus dem bereits hochgeladenen Template wieder her. Ist die virtuelle Maschine installiert, können Sie diese per CLI direkt starten. Dies geschieht durch den Aufruf

```
xe.exe -s $ip -u $user -pw $pw
vm-start vm=$newUuid
```

Welche virtuelle Maschine gestartet wird, lässt sich entweder durch die Angabe des Namens oder der *uuid* bestimmen.

Automatisierung der Snapshotlösung

Um das regelmäßige Ausführen von Backups zu automatisieren, bietet sich in der Microsoft-Welt der Aufgabenplaner an. Sie finden ihn unter “Start / Alle Programme / Zubehör / Systemprogramme”. Das Hinzufügen einer neuen Aufgabe sollte kein Problem darstellen. Hierbei ist zu beachten, dass bei der Erstellung der Aufgabe die Aktion “Programm/Skript starten” ausgewählt ist. Als das auszuführende Programm geben Sie die PowerShell an: *C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe*. Sie wird mit den Argumenten *-command “Add-PSSnapin XenServer PSSnapIn; & C:\PATH\TO\SCRIPT\backup.ps1”* gestartet, die Sie in dem dafür vorgesehenen Feld “Argumente hinzufügen (optional)” eintragen. Mit dem Parameter “-command” sorgen Sie dafür, dass die folgenden Instruktionen wie in einem geöffneten PowerShell-Fenster interpretiert werden. Als Erstes stellen Sie damit sicher, dass das *XenServerPSSnapIn* geladen ist. Anschließend wird das Backupskript ausgeführt und Sie müssen nur noch die Frequenz angeben, mit der die Backups ausgeführt werden sollen. Bestätigen Sie diese Aktion nun zweimal mit “OK”.

erst alle Abbilder erstellt und anschließend nacheinander exportiert werden, vielmehr kann die Sicherung auch so gestaltet werden, dass jede virtuelle Maschine komplett abgebildet und exportiert wird, bevor weitere Sicherungen vorgenommen werden.

Anstatt alle gesicherten Templates aus einem Verzeichnis wiederherzustellen, kann es komfortabler sein, die betroffenen Templates in einer Textdatei zusammenzufassen und über diese importieren zu lassen. Nach der Wiederherstellung dürften die importierten Templates in den meisten Fällen nicht mehr vonnöten sein. Sie können diese nach der erfolgreichen Erstellung automatisch vom System entfernen. Hierdurch halten Sie auch die Redundanz der vorhandenen Templates auf dem XenServer in Grenzen. Abhängig von der Anzahl und Größe der zu sichernden virtuellen Maschinen sowie dem möglichen Maximaldurchsatz des vorhandenen Netzwerks kann es nötig werden, die Sicherung auf mehrere Slots zu verteilen. Es kann durchaus vorkommen, dass eine Sicherungskopie einer 75 GByte großen virtuellen Maschine zwischen 90 und 120 Minuten oder noch länger in Anspruch nimmt. Bei einer Handvoll virtueller Maschinen bieten sich Backups von Hand über das XenCenter an. Wesentlich sicherer und komfortabler ist es allerdings, diese Aufgabe zu automatisieren. (dr)

Fazit

Die hier vorgestellten Skripte sind ein Anfang und sollten an die eigenen Bedürfnisse angepasst beziehungsweise erweitert werden [7]. Im Zusammenspiel mit dem Windows-Aufgabenplaner ist es ohne weiteres möglich, gezielt und regelmäßige Backups von virtuellen Maschinen anzufertigen. Der Einsatz des *XenServerPSSnapIns* erlaubt eine unkomplizierte Auswahl und Export der zu sichernden virtuellen Maschinen. Da Backupskripte den Großteil ihrer Zeit ohne Aufsicht ablaufen, ist es sicher von Vorteil, wenn Ausnahmebehandlung implementiert ist. Abhängig von der vorhandenen Infrastruktur kann es erforderlich sein, dass nicht zu-

```
$_scriptRoot = "C:\PATH\TO\SCRIPTS\"

# externe Skripte includieren
. ($_scriptRoot + "config.ps1" )
. ($_scriptRoot + "xen.ps1" )

trap
{
    "Hier ist etwas furchtbar schief gegangen: " + $_
}

$backupFiles = Get-Childitem $_restorePath -Name -Include *.xva

if ($backupFiles -ne '') {
    foreach ($backupFileName in $backupFiles) {
        ImportSnapshot $backupFileName
    }
}
```

Listing 5: restore.ps1



- [1] Download der PowerShell 2.0 <http://support.microsoft.com/kb/968929>
- [2] Hilfe zu Cmdlets <http://it-a.eu/aap32>
- [3] XenServer und XenCenter-Download <http://it-a.eu/aap33>
- [4] Installationshilfe zu XenServer und XenCenter <http://it-a.eu/aap34>
- [5] Dokumentation zum Command Line Interface <http://it-a.eu/aap35>
- [6] PowerShell Snap-In für XenServer <http://community.citrix.com/cdn/xs/sdks>
- [7] Citrix XenServer Administrators Guide <http://it-a.eu/aap37>

Links

