



# Whitepaper Desired State Configuration (DSC) und Windows Server 2016

– Ein Einstieg –

© Gerhard Glenk  
IT Consulting  
Josef-Simon-Str. 35  
90473 Nürnberg  
E-Mail: [gerhard.glenk@online.de](mailto:gerhard.glenk@online.de)

Version 1.0 – Juni 2017

Veröffentlicht bei:



Rachfahl IT-Solutions GmbH & CO. KG  
Heiligenhaus 21  
59969 Hallenberg  
Website: <https://www.hyper-v-server.de>

## Inhaltsverzeichnis

Vorwort .....	4
Skripte zum Downloaden .....	4
1 DSC-Grundlagen.....	5
1.1 Das Modell, das hinter DSC steckt.....	5
1.2 Konfigurationsskripte .....	7
1.2.1 Die Ressourcen-Eigenschaften <code>Ensure</code> und <code>DependsOn</code> .....	8
1.2.2 Kompilieren der Konfiguration .....	8
1.2.3 MOF-Dateien.....	9
1.2.4 Anwenden der kompilierten Konfiguration .....	9
1.3 DSC Ressourcen .....	11
1.3.1 Die PowerShell Gallery.....	11
1.3.2 Community Websites.....	12
1.3.3 Eigene Ressourcen .....	12
1.3.4 Versionsverwaltung .....	13
1.4 Konfiguration des LCM (Local Configuration Manager) .....	14
1.5 Spielereien mit dem Einführungsbeispiel.....	15
1.6 DSC in Projektteams – Partielle Konfigurationen.....	15
1.7 Nächste Schritte .....	16
2 VMs im Hyper-V, Partielle Konfigurationen und ein DSC Web Pull Server .....	17
2.1 Die Lab Systemumgebung .....	17
2.1.1 Die Physik.....	17
2.1.2 Die Hyper-V Umgebung .....	17
2.1.3 Betriebssystem Images für die VMs.....	18
2.1.4 Einsammeln und Installieren benötigter DSC-Ressourcen .....	20
2.2 Ein DSC Konfigurationsskript zum Erzeugen von VMs im Hyper-V .....	22
2.2.1 Parameterliste.....	22
2.2.2 Importieren benötigter DSC-Ressourcen.....	23
2.2.3 Variablendefinitionen .....	23
2.2.4 Die Konfiguration im Detail.....	23
2.2.4.1 Verzeichnisse für die VM erzeugen.....	23
2.2.4.2 Die Startdatei für die VM erzeugen.....	23
2.2.4.3 Setup Dateien für die VM in die Startdatei kopieren.....	24
2.2.4.4 Die VM im Hyper-V erzeugen.....	24
2.2.4.5 Dem Netzwerkadapter der VM einen vernünftigen Namen geben.....	25
2.2.5 Eine erste Test-VM.....	26

2.3	Bereitstellen VM-spezifischer Konfigurationsdaten am Beispiel eines Web Pull Servers.....	27
2.3.1	Was ist ein DSC Web Pull Server?.....	27
2.3.2	Bereitstellen der Konfigurationsdaten.....	27
2.4	Die Konfiguration unseres DSC Web Pull Servers DSC-PS01 mit partiellen Konfigurationen ..	29
2.4.1	Die Betriebssystemkonfiguration (OSconfig).....	30
2.4.2	Die Anwendungskonfiguration (APPconfig) für unseren Web Pull Server DSC-PS01.....	31
2.4.3	Die Konfiguration des Local Configuration Managers (LCMconfig) für unseren DSC Web Pull Server DSC-PS01.....	34
2.4.4	Das Initiieren der DSC-Konfiguration in der VM nach der Windows Installation .....	35
2.5	Der DSC Web Pull Server DSC-PS01 entsteht .....	36
2.5.1	Start und Konfiguration .....	36
2.5.2	Funktionsprüfung.....	37
2.6	Ein einfacher Pull Client.....	38
2.6.1	Konfigurationsskripte und DSC-Ressourcen erstellen und einsammeln .....	38
2.6.2	Konfigurationsskripte und DSC-Ressourcen auf dem Pull Server publizieren .....	40
2.6.2.1	Aufbereiten der MOF-Konfigurationsdateien .....	40
2.6.2.2	Aufbereiten der DSC Ressourcen Module.....	40
2.6.2.3	Es geht einfacher.....	41
2.6.3	Erstellen des Pull Clients .....	42
2.7	Der Pull Server als Report Server.....	45
2.7.1	Senden von Report Daten.....	45
2.7.2	Auswerten der Report Daten .....	46
2.8	Troubleshooting .....	49
2.8.1	Die Ereignisanzeige (Event Viewer) in der VM.....	49
2.8.2	Protokolldateien .....	49
2.8.3	Praxistipp für unser Vorgehen .....	50
3	Wie geht's weiter?.....	50
Anhang A:	Die Skripte zum Downloaden .....	51

## Vorwort

Die neueste Version der Server Plattform Windows Server 2016 bringt im Bereich “Verwaltung und Automatisierung” einige interessante Neuerungen und Weiterentwicklungen mit, die bei den Highlights meistens nicht oder nur am Rande erwähnt werden. Ich meine damit z.B. das Thema “Desired State Configuration (DSC)”. DSC gibt es zwar schon länger, aber mit dem Server 2016 wird diese Technologie meines Erachtens zukünftig an Bedeutung gewinnen. Sicher wird sich der eine oder andere nun fragen, was sich dahinter verbirgt und was man damit anfangen kann.

In diesem Dokument will ich deshalb etwas näher auf diese Technologie eingehen, indem ich zunächst ein paar Grundlagen beschreiben und dann darauf aufbauend eine Lab-Umgebung mit virtuellen Maschinen im Hyper-V schaffen werde, die ich als Basis für eine Private Cloud verwenden kann.

## Skripte zum Downloaden

In diesem Dokument finden Sie viele Code Schnipsel aus PowerShell Skripten in Form von Bildschirmfotos. Um Ihnen lästige Tipparbeit beim Nachvollziehen der Szenarien zu ersparen, habe ich die Skripte in eine Datei mit dem Namen *DSC-Lab.zip* gepackt, die Sie von der URL

<https://www.hyper-v-server.de/wp-content/uploads/2017/02/DSC-Lab.zip>

herunterladen können. Details zum Inhalt dieser .ZIP-Datei und deren Handhabung finden Sie im [Anhang A](#) dieses Dokuments.

Bitte beachten Sie, dass diese Skripte auf meine später noch beschriebene [Test- und Entwicklungs-umgebung](#) ausgerichtet sind. Falls Sie eine andere Systemumgebung verwenden, müssen Sie gegebenenfalls an der einen oder anderen Stelle Laufwerks- und / oder Pfadangaben anpassen.

## 1 DSC-Grundlagen

Bevor wir jedoch beginnen, unsere Wolke mit DSC aufziehen zu lassen, sollten wir uns erst mal mit den Grundlagen dieser Technologie vertraut machen, was ich mit diesem ersten Beitrag versuchen will.

DSC wurde erstmals mit der PowerShell 4.0 eingeführt und basierte auf dem Windows Management Framework (WMF) 4.0, das standardmäßig in Windows 8.1 und Windows Server 2012 R2 enthalten war. Für den Windows Server 2016 in allen Editionen einschließlich dem Nano Server und auch Windows 10 in der aktuellen Version 1607 wurde die Technologie weiterentwickelt und steht nun in der Version 5.0 zur Verfügung. Für ältere Windows Versionen (ab Windows 7 bzw. Windows Server 2008) steht [WMF 5 zum Download](#) bei Microsoft zur Verfügung. Dieser und die weiteren Artikel beziehen sich auf die Version 5.0. Wenn Sie die Beispiele nachvollziehen wollen, empfehle ich Ihnen, Ihr System unbedingt auf diese neue Version zu aktualisieren. Über die globale PowerShell Variable `$PSVersionTable` können Sie die WMF / PowerShell Version, die auf Ihrem System installiert ist, überprüfen. Beim Windows Server 2016 werden aktuell folgende Informationen angezeigt:

```
PS C:\windows\system32> $PSVersionTable
Name                           Value
----                           -
PSVersion                       5.1.14393.1198
PSEdition                       Desktop
PSCompatibleVersions             {1.0, 2.0, 3.0, 4.0...}
BuildVersion                     10.0.14393.1198
CLRVersion                       4.0.30319.42000
WSManStackVersion                3.0
PSRemotingProtocolVersion        2.3
SerializationVersion            1.1.0.1
```

### 1.1 Das Modell, das hinter DSC steckt

Bei vielen Projekten, an denen ich in der Vergangenheit mitgearbeitet habe, war die Beschreibung der Konfiguration der beteiligten Systeme und Komponenten eine zentrale Aufgabe. Es wurden detaillierte Checklisten erstellt, in denen Schritt für Schritt die notwendigen Aktionen dokumentiert wurden, um später eine einmal erzeugte Umgebung erneut aufbauen zu können und auch Vorgehensweisen für Fehleranalyse und Korrektur festzulegen. Diese Checklisten waren sicherlich hilfreich, wenn sich neue Mitarbeiter in ein Projekt einarbeiten mussten. Sie hatten jedoch einen Schönheitsfehler: Sie mussten immer von einem Menschen abgearbeitet werden und konnten nicht direkt von einem Computer ausgeführt werden.

Mit DSC (Desired State Configuration) kann dieser Medienbruch beseitigt werden. Spezielle PowerShell-Sprachelemente erlauben es, die einzelnen Konfigurationsschritte für die beteiligten Systeme und die darauf benötigten Komponenten in PowerShell **Konfigurationsskripten** zu beschreiben und diese dann bei Bedarf an die jeweiligen Zielsysteme zur Abarbeitung zu verteilen.

Auf den Zielsystemen gibt es einen lokalen Prozess – den **Local Configuration Manager (LCM)** – der die gesendeten Konfigurationsskripte entgegennimmt und die darin enthaltenen Konfigurationsschritte ausführt.

Damit der LCM die erhaltenen Konfigurationsschritte ausführen kann, benötigt er als **Ressourcen** bezeichnete Komponenten, die in PowerShell-Modulen (.PSM1-Dateien) bereitgestellt werden müssen und die die Logik für die eigentlichen Konfigurationsschritte enthalten. Sie bieten außerdem entsprechende Parameter z.B. für Pfad-, Benutzer- oder komponentenspezifische Eigenschaften an. Der LCM entnimmt den erhaltenen Konfigurationsskripten die Werte dieser Parameter und startet damit die Ressourcen.

Der LCM startet aber nicht nur die Ressourcen für die Erstkonfiguration, sondern er ruft auch in konfigurierbaren Zeitintervallen die Ressourcen auf, um zu überprüfen, ob die Konfiguration noch den

ursprünglichen Parametern entspricht. Bei Abweichungen kann er dann die Ressource veranlassen, die ursprünglichen Einstellungen wieder herzustellen.

Bleibt die Frage, wie der LCM seine Konfigurationsskripte erhält. Es gibt 2 Betriebsmodi: Den **Push** und den **Pull** Modus. Im Push Modus müssen die Konfigurationsskripte und Ressourcen von einem Administrator manuell an den LCM übermittelt und gestartet werden. Im Pull Modus wird der LCM eines Zielsystems einmalig angewiesen, sich von einem zentralen **Pull-Server** regelmäßig seine für ihn bestimmten Konfigurationsskripte und Ressourcen zu holen und diese mit der vorhandenen Konfiguration abzugleichen. Das Ergebnis kann dann an einen zentralen Report-Server zurückgemeldet werden.

Jetzt könnte der Einwand kommen, warum man für die Konfiguration eines Systems einen solchen Aufwand über mehrere Stufen betreiben soll. Für ein einzelnes System mag dies berechtigt sein, da kommt man oftmals mit speziell geschriebenen "klassischen" PowerShell-Skripten schneller ans Ziel. Aber betrachten wir mal die Situation eines Service Providers (egal ob in einer Public Cloud oder in einem unternehmenseigenen Rechenzentrum), der auf Anforderung mehrere – unter Umständen Hunderte oder Tausende – Systeme mit der gleichen Konfiguration bereitstellen, überwachen und im Fehlerfall korrigieren soll. Hier können die Automatismen von DSC eine große Hilfe sein. Und genau solche Massenszenarien hatte man beim Design von DSC im Fokus.

Eine weitere Frage möchte ich hier auch gleich ansprechen: Wie schaut es denn aus mit DSC in Nicht-Windows Umgebungen wie Linux oder IOS? Nun, die Antwort ist ganz einfach. Microsoft hat vor einiger Zeit ganz offiziell die PowerShell und auch DSC als Open Source zur Verfügung gestellt und damit die Möglichkeit geschaffen, PowerShell und DSC auch in einer Nicht-Windows Welt einzusetzen. Details dazu würden hier jedoch zu weit führen. Aber hier ist schon mal der Link zur offiziellen MSDN-Dokumentation für DSC und Linux: [Erste Schritte mit DSC für Linux](#).

## 1.2 Konfigurationsskripte

Der erste Schritt, um mit DSC zu arbeiten, besteht im Erstellen von *Konfigurationsskripten*. Es handelt sich dabei um PowerShell Skripte (.PS1-Dateien), die eine *Configuration* definieren. Schauen wir uns als Einstieg ein einfaches Konfigurationsskript an:

```
Configuration DscDemo # Name der Configuration (wie bei einer PowerShell Funktion)
{
    Param # optionale Parameterliste
    (
        [Parameter()]
        [string[]]$nodeName = 'localhost'
    )

    # Wir sollten alle PowerShell Module, aus denen wir Ressourcen verwenden,
    # zuerst importieren.
    # PSDesiredStateConfiguration ist das Standardmodul von windows, das die
    # 'Out of the Box' Ressourcen enthält, zu finden unter
    # C:\Windows\System32\WindowsPowerShell\v1.0\Modules\PSDesiredStateConfiguration
    Import-DscResource -ModuleName 'PSDesiredStateConfiguration'

    $DemoFolder = "C:\DscDemo"

    # Mindestens 1 Node Block muss vorhanden sein, der das Zielsystem angibt.
    Node $nodeName
    {
        # Zu konfigurierende Ressourcen
        File DscDemoDir # Erzeugen eines Verzeichnisses
        {
            Type = 'Directory'
            DestinationPath = $DemoFolder
            Ensure = 'Present'
        }
        File DscDemoFile # Erzeugen einer Textdatei mit Inhalt
        {
            Type = 'File'
            DestinationPath = Join-Path -Path $DemoFolder -ChildPath "DemoFile.txt"
            Contents = 'Dies ist ein Demo-Text.'
            Ensure = 'Present'
            # Textdatei nur Erzeugen, wenn die Konfiguration der obigen
            # Ressource erfolgreich war
            DependsOn = '[File]DscDemoDir'
        }
    }
}

# Aufruf der Konfiguration, damit sie kompiliert wird
DscDemo
```

Eine *Configuration* ist eine spezielle Form einer PowerShell Funktion, die mit dem Schlüsselwort *Configuration* beginnt. Den Namen der *Configuration* können wir frei wählen wie für Funktionen (hier *DscDemo*) und wir können auch eine Parameterliste festlegen. Innerhalb des *Configuration* Blocks können beliebige PowerShell-Befehle stehen. Es muss aber mindestens ein *Node* Block vorhanden sein, mit dem der Name des Zielsystems festgelegt wird, für das die *Configuration* gelten soll. Innerhalb des *Node* Blocks können jetzt die Ressourcen, die auf dem Knoten konfiguriert werden sollen, mit ihren Parametern aufgeführt werden. Ein Ressourcen-Block beginnt immer mit dem Namen einer Ressource gefolgt von einer Bezeichnung des Ressourcen-Blocks. Die Bezeichnung eines Ressourcen-Blocks können wir frei wählen. Das vorstehende Beispiel enthält 2 Ressourcen Blöcke. Beide beziehen sich auf die Standard-Ressource *File* aus dem DSC Standardmodul *PSDesiredStateConfiguration*. Der erste Ressourcenblock erhält die Bezeichnung *DscDemoDir*, den zweiten benennen wir *DscDemoFile*.

Zur Erläuterung: Die *File* Ressource ermöglicht verschiedene Dateisystem-Operationen wie z.B. das Anlegen oder Löschen von Verzeichnissen oder Dateien mit Inhalt.

Eine Ressource offeriert verschiedene Eigenschaften bzw. Properties, die für das Konfigurieren mit Werten versehen werden müssen. Die *File* Ressource erwartet z.B. eine Eigenschaft für den Typ der Dateisystem-Operation (Verzeichnis- oder Datei-Operation) und dann natürlich einen Pfadnamen für die Operation (Property *DestinationPath*).

Beim Erstellen eines Konfigurationskripts empfiehlt es sich, alle PowerShell-Module, die Ressourcen enthalten, auf die wir in den Ressourcen-Blöcken Bezug nehmen, gleich am Beginn der `Configuration` mit `Import-DscResource` zu importieren (siehe Zeile 14 im vorstehenden Beispiel). `Import-DscResource` ist ein dynamisches Schlüsselwort, das nur innerhalb eines `Configuration` Blocks erkannt wird. Damit ermöglichen wir es insbesondere der PowerShell ISE bereits beim Erstellen von Ressourcen-Blöcken, jeweils die für eine Ressource verfügbaren Eigenschaften als Liste im Editor anzuzeigen und auch gleich eine Syntaxprüfung durchzuführen. Außerdem kann damit die PowerShell angewiesen werden, eine bestimmte Version eines Moduls einzubinden, falls mehrere Versionen des Moduls auf dem System existieren.

### 1.2.1 Die Ressourcen-Eigenschaften `Ensure` und `DependsOn`

`Ensure`: Diese Eigenschaft kann bei vielen Ressourcen angegeben werden. Ihr kann als Stringwert entweder `'Present'` oder `'Absent'` zugewiesen werden. `'Present'` bedeutet, dass die Konfiguration wie angegeben vorhanden sein muss. Falls nicht, wird sie erzeugt. Mit `'Absent'` wird angegeben, dass die Konfiguration entfernt werden soll, falls sie vorhanden ist.

In unserem Beispiel wird also mit dem ersten Ressource-Block ein Verzeichnis erzeugt, falls es noch nicht vorhanden ist. Mit dem zweiten Ressource-Block wird in diesem Verzeichnis eine Textdatei mit einem bestimmten Inhalt erstellt. Hier sorgt die Angabe von `'Present'` nicht nur dafür, dass die Datei vorhanden ist, sondern ihr Inhalt auch dem Wert der `Content` Eigenschaft entspricht.

`DependsOn`: Normalerweise werden die Ressourcen-Blöcke in der Reihenfolge abgearbeitet, in der sie im Konfigurationskript aufgeführt sind. Mit der `DependsOn` Eigenschaft kann diese Reihenfolge beeinflusst werden. Somit können Abhängigkeiten zwischen Ressourcen festgelegt werden. Ein Ressourcen-Block wird erst dann abgearbeitet, wenn die bei `DependsOn` angegebenen Ressourcen-Blöcke zuvor erfolgreich ausgeführt wurden. Der Eigenschaft kann als Wert ein Stringarray zugewiesen werden, das Verweise auf andere Ressource-Blöcke enthält. Jeder Eintrag im Stringarray muss die Form haben: `'[<RessourceName>]<RessourceblockBezeichner>'`. In unserem Beispiel wird also der Block `DscDemoFile` nur dann berücksichtigt, wenn der Ressourcen-Block `DscDemoDir` zuvor erfolgreich ausgeführt wurde.

### 1.2.2 Kompilieren der Konfiguration

Unser Konfigurationskript speichern wir jetzt in einer PowerShell (.PS1) Datei (z.B. `DscDemo.ps1`). Um die `Configuration` für die weitere Verarbeitung vorzubereiten, müssen wir sie kompilieren. Hierzu muss die `Configuration` aufgerufen werden wie eine PowerShell-Funktion. Es gibt mehrere Möglichkeiten dazu:

Über die PowerShell-Konsole per "Dot Sourcing": Führen Sie folgende PowerShell-Befehle aus:

```
. .\DscDemo.ps1 # Lädt die Konfiguration in den globalen Arbeitsraum der PowerShell
DscDemo        # Aufruf der Konfiguration, gegebenenfalls mit Parametern
```

In der PowerShell ISE mit der Taste `F5` bzw. über den Menüpunkt "Run Script"

Direkt im Konfigurationskript – siehe letzte Zeile im obigen Beispiel:

```
PS D:\DSC-Lab> .\DscDemo.ps1

Verzeichnis: D:\DSC-Lab\DscDemo

Mode                LastWriteTime         Length Name
----                -
-a----            16.05.2017   14:40           2950 localhost.mof

PS D:\DSC-Lab>
```



Beim Kompilieren geschieht folgendes:

- Alle Variablen werden aufgelöst und eingebettete Befehle werden ausgeführt.
- Im aktuellen Verzeichnis wird ein Unterordner mit dem Namen der Konfiguration erzeugt.
- Für jeden *Node*-Block wird in diesem Unterverzeichnis eine Datei *NodeName.MOF* erzeugt.

Anmerkung: Will man das Ergebnis der Kompilierung in einem beliebigen anderen Ordner als in einem Unterordner des aktuellen Verzeichnisses ablegen, so kann beim Aufruf der `Configuration` zum Kompilieren über den Parameter `-OutputPath` der gewünschte Pfad angegeben werden, also z.B.

```
PS D:\DSC-Lab> . .\DscDemo.ps1
PS D:\DSC-Lab> dscdemo -OutputPath C:\Temp

Verzeichnis: C:\Temp

Mode                LastWriteTime         Length Name
----                -
-a----             16.05.2017   14:46           2950 localhost.mof
```

Der Parameter `-OutputPath` steht immer zur Verfügung. Er muss nicht in der Parameterliste der `Configuration` aufgeführt sein.

### 1.2.3 MOF-Dateien

Die beim Kompilieren einer `Configuration` erzeugten MOF-Dateien enthalten nun alle Informationen für den Local Configuration Manager (LCM) des jeweiligen Zielsystems. Doch was sind MOF-Dateien?

MOF steht für *Managed Object Format*. Es handelt sich um ein Text-Dateiformat, das von der [Distributed Management Task Force \(DMTF\)](#) – einer herstellerübergreifenden Organisation, der auch Microsoft angehört – definiert wurde mit dem Ziel, plattformübergreifend Softwarekomponenten und Systeme zu konfigurieren und zu verwalten. So gesehen, können DSC-Konfigurationsskripte auch für Nicht-Windows Systeme erstellt werden, z.B. für [verschiedene Linux](#) Derivate, sofern diese MOF-Dateien unterstützen. Andererseits besteht auch die Möglichkeit, mit Tools von Fremdherstellern (z.B. *Chef* oder *Puppet*) Konfigurationen zu beschreiben, sie als MOF-Dateien bereitzustellen und mit DSC weiterzuverarbeiten.

### 1.2.4 Anwenden der kompilierten Konfiguration

Ich habe weiter oben bereits beschrieben, dass die Konfiguration eines Systems durch dessen **Local Configuration Manager (LCM)** durchgeführt wird und dass hierzu die Konfigurationsdaten (genauer die .MOF-Dateien) und die benötigten Ressourcen verfügbar sein müssen. Dazu gibt es 2 Betriebsmodi für den LCM, den Push und den Pull Mode. Für den Pull Mode benötigen wir einen zentralen **Pull Server**, von dem sich der LCM selbständig die Informationen besorgt.

Aktuell haben wir jedoch noch keinen solchen zentralen Pull Server. Um unsere Beispiel-Konfiguration anwenden zu können, müssen wir deshalb auf den Push Mode zurückgreifen. Dazu gibt es das Cmdlet `Start-DscConfiguration`. Diesem Cmdlet übergeben wir als Parameter den Pfadnamen des beim Kompilieren der Konfiguration erzeugten Ordners mit den .MOF-Dateien. Über weitere Parameter können wir den Ablauf beeinflussen, z.B. warten bis die Aktion beendet ist (`-Wait`), ein Protokoll mitlaufen lassen (`-Verbose`) oder Erzwingen, dass die Konfiguration vollständig neu gestartet wird und eine bereits zuvor angestoßene (und möglicherweise wegen Fehlern “hängengebliebene”) Instanz beendet wird (`-Force`). Um unser Beispiel anzuwenden, können wir also nach dem Kompilieren folgenden Befehl eingeben und erhalten das gezeigte Ergebnis:

```
PS D:\DSC-Lab> Start-DscConfiguration -Path .\DscDemo -wait -Force -Verbose
```

```
AUSFÜHRLICH: Vorgang "CIM-Methode aufrufen" mit den folgenden Parametern durchführen,
"methodName" = SendConfigurationApply, 'className' = MSFT_D
SCLocalConfigurationManager, 'namespaceName' =
root/Microsoft/Windows/DesiredStateConfiguration".
AUSFÜHRLICH: Vom Computer 'M6500PRO' mit Benutzer-SID 'S-1-5-21-2204300359-590443393-
3658835497-1001' ist ein LCM-Methodenaufruf eingegangen.
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenFestlegen]
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenRessource] [[File]DscDemoDir]
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenTesten ] [[File]DscDemoDir]
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoDir] Das System kann die
angegebene Datei nicht finden.
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoDir] Zugehörige(s)
Datei/Verzeichnis: C:\DscDemo.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenTesten ] [[File]DscDemoDir] in 0.0160 Sekunden.
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenFestlegen] [[File]DscDemoDir]
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoDir] Das System kann die
angegebene Datei nicht finden.
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoDir] Zugehörige(s)
Datei/Verzeichnis: C:\DscDemo.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenFestlegen] [[File]DscDemoDir] in 0.0160 Sekunden.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenRessource] [[File]DscDemoDir]
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenRessource] [[File]DscDemoFile]
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenTesten ] [[File]DscDemoFile]
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoFile] Das System kann die
angegebene Datei nicht finden.
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoFile] Zugehörige(s)
Datei/Verzeichnis: C:\DscDemo\DemoFile.txt.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenTesten ] [[File]DscDemoFile] in 0.0000 Sekunden.
AUSFÜHRLICH: [M6500PRO]: LCM: [ StartenFestlegen] [[File]DscDemoFile]
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoFile] Das System kann die
angegebene Datei nicht finden.
AUSFÜHRLICH: [M6500PRO]: [[File]DscDemoFile] Zugehörige(s)
Datei/Verzeichnis: C:\DscDemo\DemoFile.txt.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenFestlegen] [[File]DscDemoFile] in 0.0150 Sekunden.
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenRessource] [[File]DscDemoFile]
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenFestlegen]
AUSFÜHRLICH: [M6500PRO]: LCM: [ BeendenFestlegen] in 0.2620 Sekunden.
AUSFÜHRLICH: Vorgang "CIM-Methode aufrufen" wurde abgeschlossen.
AUSFÜHRLICH: Die Ausführung des Konfigurationsauftrags hat 0.425 Sekunden gedauert.
```

Wie man sieht, ruft das Cmdlet *Start-DscConfiguration* den LCM auf dem Zielsystem (in unserem Fall *localhost*) auf und übergibt ihm den Pfad für den Ordner mit der .MOF-Datei. Um die Ressource *File*, die in diesem Beispiel verwendet wird, brauchen wir uns dabei nicht zu kümmern, da es sich um eine Standard-Ressource handelt, die in jedem DSC-fähigen System verfügbar ist.

### 1.3 DSC Ressourcen

Ich habe es weiter oben schon erwähnt: Damit eine DSC-Konfiguration vom Local Configuration Manager (LCM) auf einem Zielsystem durchgeführt werden kann, müssen dort die Ressourcen, die in einem Konfigurationskript mit Parametern angegeben sind, in Form von PowerShell Modulen verfügbar sein. Aber wo verstecken sich diese DSC-Module?

Es gibt in der PowerShell eine Environmentvariable `$env:PSModulePath`, die eine Liste von Pfaden enthält, in denen der LCM nach Modulen sucht, u.a.

```
C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
```

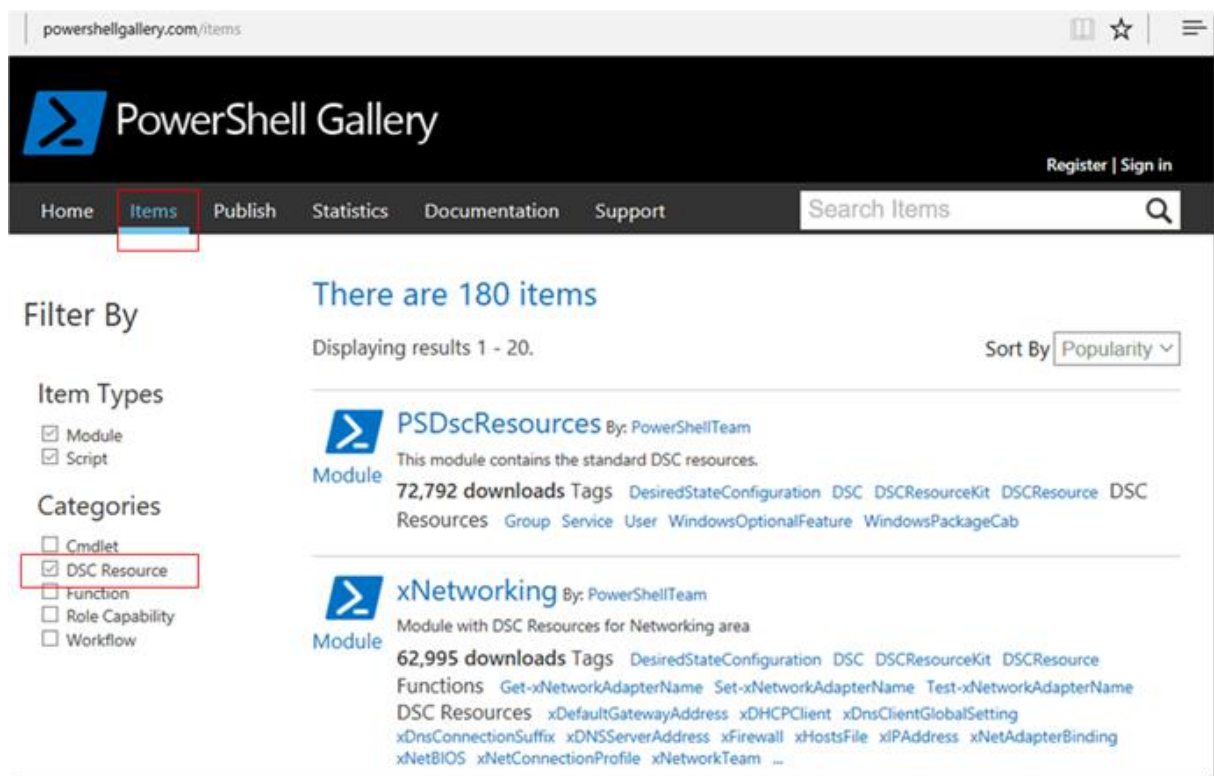
- hier liegen die DSC-Module, die standardmäßig mit Windows installiert werden. Sie ermöglichen das Konfigurieren einiger grundlegender Funktionen des Betriebssystems wie z.B. Datei-, Registry-, Prozess- oder Benutzer-Operationen sowie die Installation von Windows Betriebssystemkomponenten. Eine Übersicht findet man in der [MSDN-Library](#).

```
C:\Program Files\WindowsPowerShell\Modules\
```

- hier sollten die DSC-Module liegen, die vom Benutzer zusätzlich bereitgestellt werden. Doch woher bekommt man die?

#### 1.3.1 Die PowerShell Gallery

Für weiterführende Aufgaben gibt es DSC-Ressourcen zum Download im Internet. Microsoft veröffentlicht in regelmäßigen Abständen sogenannte "DSC Resource Kits". Dabei handelt es sich allerdings nicht um komplette Download-Pakete, sondern vielmehr um eine Auflistung von verfügbaren DSC-Ressourcen in der [PowerShell Gallery](#). Eine aktuelle Liste erhält man, wenn man in der [PowerShell Gallery](#) bei der Suche unter "Items" die Kategorie "DSC Resource" markiert.



The screenshot shows the PowerShell Gallery website interface. The 'Items' tab is selected in the navigation menu. The main content area displays 'There are 180 items' and 'Displaying results 1 - 20'. A 'Filter By' sidebar on the left shows 'Item Types' (Module, Script) and 'Categories' (Cmdlet, DSC Resource, Function, Role Capability, Workflow). The 'DSC Resource' category is checked. The search results list two modules: 'PSDscResources' (72,792 downloads) and 'xNetworking' (62,995 downloads). Both are by PowerShellTeam and include tags like 'DesiredStateConfiguration', 'DSC', and 'DSCResourceKit'.

Für das Herunterladen einzelner Ressourcen stellt die PowerShell direkt ein eigenes Cmdlet *Install-Module* zur Verfügung, z.B. aus dem vorstehenden Screenshot:

```
PS D:\DSC-Lab> Install-Module -Name 'xNetworking' -Force
```

*Install-Module* sucht standardmäßig in der [PowerShell Gallery](#) nach dem Modul mit dem bei *-Name* angegebenen Namen und kopiert es in den Pfad

```
C:\Program Files\WindowsPowerShell\Modules\
```

Mit *-Force* können wir die aktuelle Version der Ressource herunterladen, auch wenn bereits eine ältere Version vorhanden ist ("Side by Side" Installation). Auf die Versionsverwaltung von DSC-Ressourcen werden wir gleich zu sprechen kommen. *Install-Module* bietet noch weitere Parameter an wie z.B. die Internet-Adresse eines anderen Repositoriums als die PowerShell Gallery oder die explizite Angabe einer Versionsnummer der DSC-Ressource. Ich werde bei meinen Beispielen noch darauf zurückkommen.

### 1.3.2 Community Websites

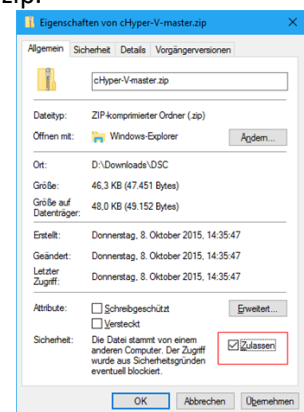
DSC ist wie die gesamte PowerShell ein Open Source Software (OSS) Angebot von Microsoft. Entsprechend gibt es neben den "offiziellen" Microsoft Quellen auch eine Reihe von Community Websites, auf denen man weitere DSC-Ressourcen findet oder die Derivate bereits vorhandener Module anbietet. Gute Quellen sind die Seiten des [PowerShell Teams](#) von Microsoft (<https://github.com/PowerShell>), die Seiten von [PowerShell Magazin](#) (<http://www.PowerShellmagazine.com>) und die der [PowerShell.org Community](#) (<https://PowerShell.org>). Und natürlich wird eine Bing oder Google Suche weitere Quellen aufzeigen.

Die auf Community Websites angebotenen DSC-Ressourcen lassen sich nicht immer mit *Install-Module* installieren, sondern werden häufig in Form von .ZIP-Dateien angeboten. Um solche Ressourcen nutzen zu können, muss man die .ZIP-Dateien manuell herunterladen und dann in den Standardpfad

```
C:\Program Files\WindowsPowerShell\Modules\
```

entpacken. Ein paar Tipps dazu:

- Vergessen Sie nach dem Herunterladen nicht, eine Virenprüfung durchzuführen!
- Falls der Name der .ZIP-Datei nicht dem Namen des PowerShell-Moduls entspricht, geben sie ihr vor dem Entpacken den passenden Namen. Ich habe z.B. vor einiger Zeit aus dem Github Repository von PowerShell.Org eine ZIP-Datei cHyper-V-master.zip heruntergeladen, die eine Ressource cHyperV enthält. Also benennen wir die Datei zunächst um in cHyper-V.zip.
- Aus Sicherheitsgründen werden von den aktuellen Windows Versionen heruntergeladene Dateien mit einer Sicherheitssperre versehen, die man vor dem Entpacken entfernen sollte. Vergisst man dies, sind die entpackten Daten ebenfalls mit dieser Sperre belegt und es kommt zu Fehlermeldungen beim Zugriff, die nicht sofort auf die Sperre hindeuten. Zum Aufheben der Sperre rufen Sie im Explorer die Eigenschaften der Datei auf und setzen auf der Registerkarte Allgemein einen Haken im Kästchen Zulassen.
- Jetzt können Sie den Inhalt der .ZIP-Datei in das Modul-Verzeichnis der PowerShell entpacken, z.B. mit



```
Expand-Archive -Path ".\cHyper-V.zip" -DestinationPath "C:\Program Files\WindowsPowerShell\Modules"
```

### 1.3.3 Eigene Ressourcen

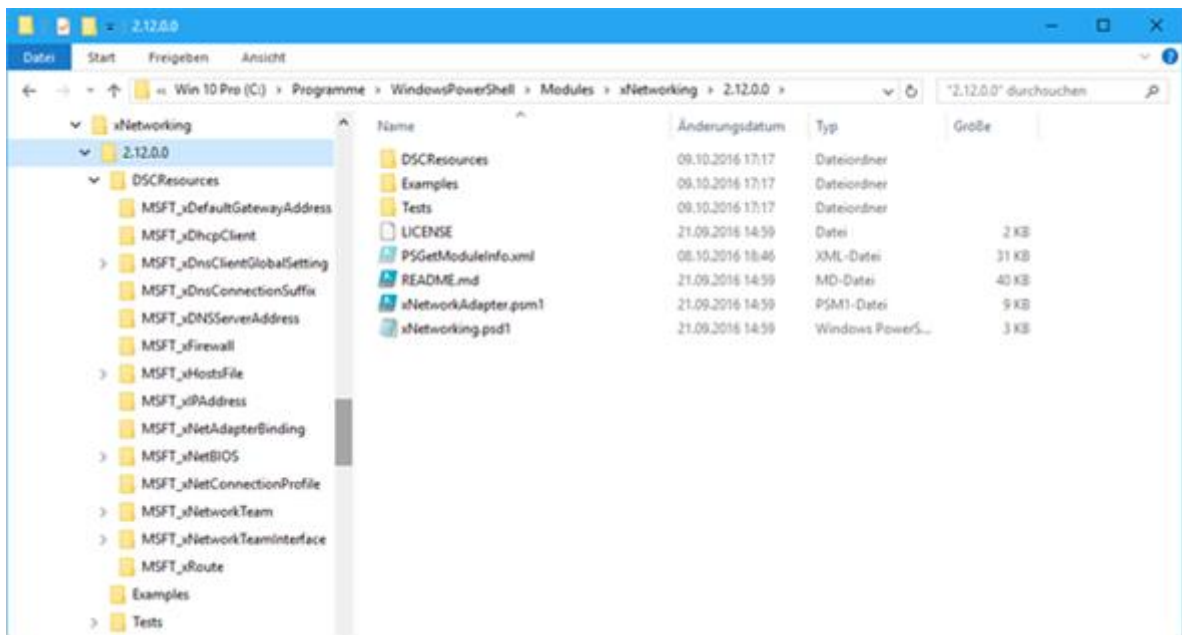
Auch wenn es mittlerweile sehr viele DSC Ressourcen im Internet gibt, kann man auf eine Anforderung stoßen, zu der man nichts Geeignetes findet. In diesem Fall ist es durchaus möglich, sich selbst PowerShell Module mit geeigneten Ressourcen zu erstellen. Spezielle Entwicklungswerkzeuge sind dafür nicht notwendig, es genügt ein PowerShell Editor wie z.B. die PowerShell ISE.

Die selbst geschriebenen Module können dann in das globale Modulverzeichnis "C:\Program Files\WindowsPowerShell\Modules" zusammen mit den notwendigen Verwaltungsinformationen kopiert werden und stehen sofort für die Verwendung in Konfigurationskripten zur Verfügung.

Im Verlauf meines angedachten Cloud Projekts werde ich sicherlich auf dieses Thema ausführlicher zurückkommen.

#### 1.3.4 Versionsverwaltung

Mit der PowerShell 5 – also mit Windows Server 2016 bzw. Windows 10 Version 1607 – wurde eine Versionsverwaltung für DSC-Ressourcen eingeführt. Dadurch ist es möglich PowerShell-Module gleichen Namens, aber mit verschiedenen Versionsnummern auf dem gleichen System zu halten ("Side by Side" Installation). Dies ist sehr einfach und übersichtlich gelöst. Unter dem Verzeichnis des Moduls gibt es für jede Version einen eigenen Unterordner mit der jeweiligen Versionsnummer. Erst in diesem sind dann die Dateien der jeweiligen Modulversion abgelegt. Das Cmdlet *Install-Module* erzeugt diese Verzeichnisstruktur automatisch, sofern die Download-Quelle eine Versionsnummer definiert. Beispielsweise wird beim Herunterladen der Ressource *xNetworking* aus der PowerShell Gallery mit *Install-Module* folgende Verzeichnisstruktur erzeugt



Ich habe bei den Konfigurationsskripten empfohlen, alle Module, aus denen DSC-Ressourcen benötigt werden, gleich am Anfang der *Configuration* mit dem Befehl *Import-DscResource* aufzuführen. Insbesondere muss dies geschehen, wenn verschiedene Versionen des gleichen Moduls auf dem System vorhanden sind. Um die richtige Version zu referenzieren, gibt es bei *Import-DscResource* einen zusätzlichen Parameter *-ModuleVersion*, bei dem die Versionsnummer als String anzugeben ist:

```
Import-DscResource -Module xNetworking -ModuleVersion 3.2.0.0
```

## 1.4 Konfiguration des LCM (Local Configuration Manager)

Der LCM (Local Configuration Manager) ist integraler Bestandteil der PowerShell in allen aktuellen Windows Versionen. Um ihn verwenden zu können, sind also keine Nachinstallationen notwendig. Allerdings muss bzw. kann er konfiguriert werden. Dazu erstellen wir ein spezielles Konfigurations-skript, in dem der `Configuration` Block mit dem Attribut `[DSCLocalConfigurationManager()]` markiert ist. Innerhalb einer solchen LCM-Konfiguration steht nur eine begrenzte Auswahl an Ressourcen zur Verfügung. Eine der wichtigsten Ressourcen heißt `Settings`. Über sie kann festgelegt werden, ob der LCM im Push oder Pull Mode arbeiten soll. Über weitere Eigenschaften der `Settings` Ressource kann z.B. festgelegt werden, ob der LCM regelmäßig bereits konfigurierte Ressourcen überprüft und diese gegebenenfalls korrigiert. Eine minimale LCM-Konfiguration wie sie für unser Einführungsbeispiel ausreichend ist, könnte wie folgt aussehen:

```
[DSCLocalConfigurationManager()]
configuration LCMConfig
{
    Node localhost
    {
        Settings
        {
            RefreshMode = 'Push' # alternativ: 'Pull'
            ConfigurationMode = 'ApplyAndMonitor' # Alternativ: 'ApplyOnly', 'ApplyAndAutoCorrect'

            # viele weitere Einstellungen verfügbar
        }
    }
}
LCMconfig
```

Eine solche LCM-Konfiguration müssen wir wie üblich kompilieren. Interessant dabei ist, dass jetzt für jeden Node-Block im Verzeichnis mit dem Kompilierungsresultat eine `.MOF`-Datei mit dem Namen `<Nodename>.meta.mof` erzeugt wird.

```
PS D:\DSC-Lab> D:\DSC-Lab\LCMconfig.ps1

Verzeichnis: D:\DSC-Lab\LCMConfig

Mode                LastWriteTime         Length Name
----                -
-a----             01.06.2017   11:43           1162 localhost.meta.mof
```

Um diese Konfiguration anzuwenden, können wir jetzt jedoch nicht das übliche Cmdlet `Start-DscConfiguration` verwenden. Vielmehr gibt es hierfür ein eigenes Cmdlet `Set-DscLocalConfigurationManager`.

```
Set-DscLocalConfigurationManager -Path .\LCMConfig -Force -Verbose

AUSFÜHRlich: Ausführen des Vorgangs "Start-DscConfiguration: SendMetaConfigurationApply" für
das Ziel "MSFT_DSCLocalConfigurationManager".
AUSFÜHRlich: Vorgang "CIM-Methode aufrufen" mit den folgenden Parametern durchführen,
"methodName" = SendMetaConfigurationApply, 'className' = MS
FT_DSCLocalConfigurationManager, 'namespaceName' =
root/microsoft/windows/DesiredStateConfiguration".
AUSFÜHRlich: Vom Computer 'M6500PRO' mit Benutzer-SID 'S-1-5-21-2204300359-590443393-
3658835497-1001' ist ein LCM-Methodenaufwurf eingegangen.
AUSFÜHRlich: [M6500PRO]: LCM: [ StartenFestlegen]
AUSFÜHRlich: [M6500PRO]: LCM: [ StartenRessource] [MSFT_DSCMetaConfiguration]
AUSFÜHRlich: [M6500PRO]: LCM: [ StartenFestlegen] [MSFT_DSCMetaConfiguration]
AUSFÜHRlich: [M6500PRO]: LCM: [ BeendenFestlegen] [MSFT_DSCMetaConfiguration] in 0.0060
Sekunden.
AUSFÜHRlich: [M6500PRO]: LCM: [ BeendenRessource] [MSFT_DSCMetaConfiguration]
AUSFÜHRlich: [M6500PRO]: LCM: [ BeendenFestlegen]
AUSFÜHRlich: [M6500PRO]: LCM: [ BeendenFestlegen] in 0.1000 Sekunden.
AUSFÜHRlich: Vorgang "CIM-Methode aufrufen" wurde abgeschlossen.
AUSFÜHRlich: "Set-DscLocalConfigurationManager" wurde in 0.262 Sekunden beendet.
```

Zum Anzeigen der aktuellen LCM-Konfiguration gibt es ebenfalls ein eigenes Cmdlet `Get-DscLocalConfigurationManager`:

## Get-DscLocalConfigurationManager

```

ActionAfterReboot      : ContinueConfiguration
AgentId                : 1A4F8F00-BC75-11E5-8360-B8AC6F79C4F4
AllowModuleOverwrite  : False
CertificateID          :
ConfigurationDownloadManagers : {}
ConfigurationID       :
ConfigurationMode     : ApplyAndMonitor
ConfigurationModeFrequencyMins : 15
Credential             :
DebugMode              : {NONE}
DownloadManagerCustomData :
DownloadManagerName   :
LCMCompatibleVersions : {1.0, 2.0}
LCMState               : Idle
LCMStateDetail        :
LCMVersion             : 2.0
StatusRetentionTimeInDays : 10
SignatureValidationPolicy : NONE
SignatureValidations  : {}
MaximumDownloadSizeMB : 500
PartialConfigurations :
RebootNodeIfNeeded    : False
RefreshFrequencyMins  : 30
RefreshMode           : Push
ReportManagers        : {}
ResourceModuleManagers : {}
PSComputerName        :

```

All diese Einstellungen können über die *Settings* und weitere LCM-spezifische Ressourcen beeinflusst werden. Bei Bedarf werde ich in den weiteren Folgen darauf näher eingehen.

## 1.5 Spielereien mit dem Einführungsbeispiel

Jetzt können wir mit dem vorstehenden Einführungsbeispiel ein bisschen spielen, z.B.

- Wenden Sie die Konfiguration mit *Start-DscConfiguration* an
  - ➔ Die Datei *C:\DscDemo\DemoFile.txt* wird mit dem angegebenen Inhalt erzeugt.
- Ändern Sie jetzt den Inhalt der Datei, z.B. mit Notepad
- Wenden Sie die Konfiguration erneut mit *Start-DscConfiguration* an
  - ➔ Die Datei *C:\DscDemo\DemoFile.txt* hat wieder den in der Konfiguration angegebenen Inhalt.

## 1.6 DSC in Projektteams – Partielle Konfigurationen

In unserem vorstehenden Einführungsbeispiel haben wir mit einer einzelnen Konfiguration für ein Zielsystem experimentiert, d.h. das Erstellen und auch Pflegen des Konfigurationsskripts wird von einer einzelnen Person durchgeführt. Spannend wird es bei einem Zielsystem, das von verschiedenen Gruppen konfiguriert und verwaltet wird. In einer Cloud-Umgebung wird es z.B. ein Team geben, das für die Netzwerk-Konfiguration zuständig ist, ein anderes kümmert sich um das Thema Storage oder um Management Tools wie System Center. Jedes Team erstellt seine eigenen Konfigurationsskripte. Brauchen wir jetzt einen "Chefredakteur", der für das Zusammenführen der verschiedenen Konfigurationen zuständig ist? Ja und nein.

Arbeitet man mit der PowerShell 5.0, können die Teams auf eine neue Funktionalität zurückgreifen, die sich "Partielle Konfigurationen" nennt. Dabei erstellt jede Gruppe ihr eigenes Konfigurationsskript und stellt es dem Local Configuration Manager einzeln zur Verfügung. Das kann sowohl im Push als auch im Pull Mode oder sogar gemischt geschehen. Entscheidend ist der LCM des Zielsystems. Bei ihm müssen die verschiedenen Einzelkonfigurationen als partielle Konfigurationen registriert werden und dabei der Betriebsmodus angegeben werden. Soll nun eine partielle Konfiguration im Push-Modus an den LCM übergeben werden, verwendet man statt des Cmdlet *Start-DscConfiguration* das Cmdlet *Publish-DscConfiguration*. Die Konfiguration wird dann zwischengespeichert. Sind alle Einzelkonfigurationen mit *Publish-DscConfiguration* dem LCM übergeben worden, kann die Gesamtkonfiguration auf dem Zielsystem mit *Start-DscConfiguration* initiiert werden, wobei als Parameter *-UseExisting* angegeben wird.

Im weiteren Verlauf unserer DSC-Experimente werde ich dies noch mit einem Beispiel verdeutlichen.

## 1.7 Nächste Schritte

Mit diesem Überblick haben wir uns ein bisschen die Idee von DSC und die damit verbundenen wichtigsten Arbeitsschritte angesehen. Ich denke, es ist nun an der Zeit, sich an konkreten Anwendungsszenarien zu versuchen. Im nächsten Kapitel werde ich Ihnen zeigen, wie man im Hyper-V mit DSC virtuelle Maschinen erzeugen und auf diesen dann spezielle Serverrollen wie z.B. einen DSC Web Pull Server konfigurieren kann.



## 2 VMs im Hyper-V, Partielle Konfigurationen und ein DSC Web Pull Server

Im vorstehenden [Teil 1](#) habe ich ein paar grundlegende Ideen und Vorgehensweisen der Desired State Configuration (DSC) beschrieben. Nun möchte ich Szenarien vorstellen, wie man in einer Lab-Umgebung mit DSC virtuelle Maschinen (VMs) im Hyper-V erzeugen und konfigurieren kann.

### 2.1 Die Lab Systemumgebung

Als physische Basis und Entwicklungsumgebung meiner virtuellen DSC-Lab Umgebung verwende ich eine Workstation, auf der ein deutsches Windows 10 Pro in der aktuellen Version 1607 läuft und das immer mit allen Patches aktuell gehalten wird. Alternativ kann natürlich auch Windows 10 Enterprise oder Windows Server 2016 verwendet werden. Windows 10 Home Edition scheidet aus, da für die nachfolgend beschriebenen Szenarien Hyper-V Komponente benötigt wird, die ja in den Home Editionen nicht enthalten ist.

#### 2.1.1 Die Physik

- i7 Prozessor
- 32 GB RAM
- Windows 10 Boot aus VHDX (= Laufwerk C:) mit aktiver Hyper-V Komponente
- Festplatte D: enthält Projektverzeichnis D:\DSC-Lab. In diesem Verzeichnis werden wir die DSC Konfigurationsskripte ablegen und ggf. in separaten Unterverzeichnissen zusätzliche Daten für die jeweiligen VMs.
- SSD-Laufwerk E: enthält Verzeichnis E:\Hyper-V, in dem die erzeugten VMs landen. Für jede VM wird darunter ein eigenes Verzeichnis mit dem Namen der VM angelegt, das dann alle Daten der VM enthält (Hyper-V Definitionen und virtuelle Laufwerke).

#### 2.1.2 Die Hyper-V Umgebung

Im Hyper-V habe ich bereits einen Switch für ein internes virtuelles Netzwerk mit dem Namen *NatSwitch80* angelegt. Dieses virtuelle Netzwerk werden wir als Management Netz für unsere VMs verwenden. Damit die mit diesem Netz verbundenen VMs auch Zugriff haben ins Internet, ist für dieses Netz zusätzlich ein NAT-Objekt im Hyper-V Host angelegt. Der IP-Bereich des Netzes ist 192.168.80.0/24, wobei über das NAT-Objekt die Adresse 192.168.80.1 als Gateway in die weite Welt definiert ist. Details über die NAT-Funktionalität von internen Hyper-V Netzwerken finden Sie in einem früheren [Blogpost](#) von mir. Hier nur nochmals kurz das PowerShell Skript zum Anlegen dieses NAT-Switch – Achtung: Mit Administratorrechten ausführen!:

```

Function New-NatSwitch (
    [Parameter(Mandatory)]
    [ValidateNotNullOrEmpty()]
    [String]$Name,

    [Parameter(Mandatory)]
    [ValidateNotNullOrEmpty()]
    [String]$InternalIPInterfaceAddressPrefix,

    [Int]$PrefixLength,

    [Parameter(Mandatory)]
    [ValidateNotNullOrEmpty()]
    [String]$GatewayIP
)
{
    New-VMSwitch -Name $Name -SwitchType Internal
    # find InterfaceIndex of the network adapter for the newly created internal switch
    $ifindex = (Get-NetAdapter | where {$_.Name -match $Name}).ifindex
    New-NetIPAddress -IPAddress $GatewayIP
        -PrefixLength $PrefixLength -InterfaceIndex $ifindex
    New-NetNat -Name $Name
        -InternalIPInterfaceAddressPrefix "$InternalIPInterfaceAddressPrefix/$PrefixLength"
}

New-NatSwitch -Name NatSwitch80
-InternalIPInterfaceAddressPrefix 192.168.80.0
-PrefixLength 24 -GatewayIP 192.168.80.1

```

### 2.1.3 Betriebssystem Images für die VMs

Wir werden unsere virtuellen Maschinen aus einem "jungfräulichen" Betriebssystem Image erzeugen, das wir aus der heruntergeladenen [ISO-Datei des Windows Server 2016](#) in einer .VHDX-Datei generieren. Dabei müssen wir unterscheiden zwischen Images, die alle zu einer bestimmten Server Edition gehörigen Komponenten vollständig enthalten, und Images für den Nano Server. Ich werde hier nur mit vollständigen Images arbeiten. DSC kann zwar auch im Nano Server genutzt werden, jedoch muss man bei der Auswahl von DSC-Ressourcen sorgfältig prüfen, ob die darin benötigten Systemfunktionen auch zur Verfügung stehen.

Zum Erstellen einer solchen Image-Datei aus der heruntergeladenen ISO-Datei verwenden wir das PowerShell Skript *Convert-WindowsImage.ps1*. Es existiert bereits seit Einführung von virtuellen Festplattendateien (Dateien des Typs .VHD bzw. .VHDX) – also seit Windows Vista bzw. Windows Server 2008 – in der [PowerShell Gallery](#). Das Skript wurde laufend weiterentwickelt und an die jeweiligen neuen Windows Versionen angepasst. Auf dem Windows Server 2016 Installationsmedium wird dieses Skript nun direkt in der aktuellsten Version bereitgestellt. Es befindet sich im Verzeichnis *NanoServer\NanoServerImageGenerator*

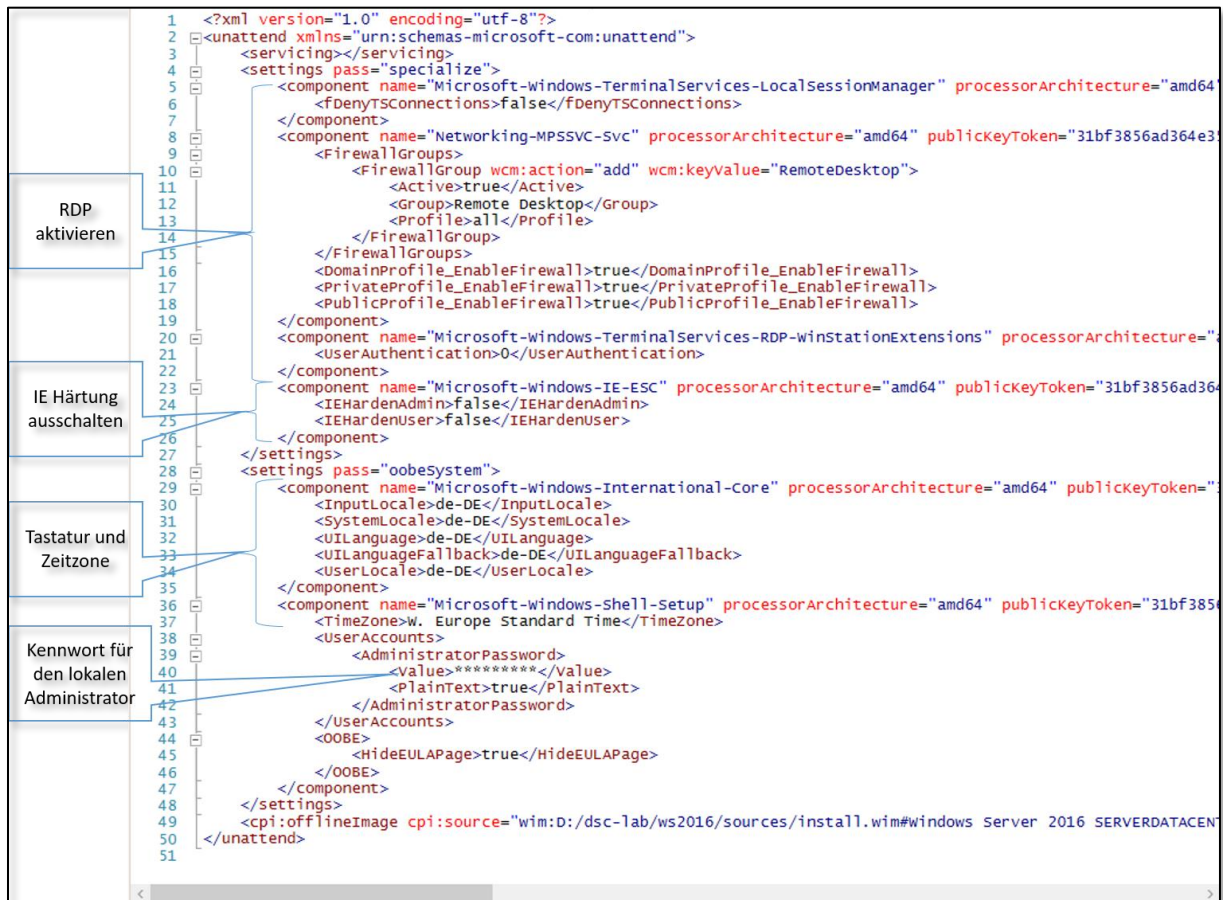
Startet man zum ersten Mal eine VM aus einem mit *Convert-WindowsImage.ps1* erzeugten Image, so wird zunächst der normale Windows Setup ausgeführt. Dabei wird man ein paar Dinge beobachten, die einen automatischen unbeaufsichtigten Start verhindern:

- Es werden die Lizenzbedingungen angezeigt, die man interaktiv bestätigen muss.
- Man muss Sprach- und Tastatureinstellungen interaktiv festlegen bzw. zumindest bestätigen.
- Man muss das initiale Passwort für den lokalen Administrator festlegen.

Abhilfe kann man schaffen, indem man vor dem Erzeugen des Images eine Antwortdatei mit dem Namen *Unattend.xml* erstellt und diese in das Image einfügt. Das Skript *Convert-WindowsImage.ps1* bietet einen Parameter, bei dem man den Pfadnamen der Antwortdatei angeben kann, die in das Image kopiert werden soll.

Eine Antwortdatei für den Windows Setup ermöglicht es zwar, fast alle Einstellungen eines Systems zu konfigurieren. Unser Ziel hier ist es aber, Konfigurationen mit *Desired State Configuration (DSC)* durchzuführen. Deshalb beschränken wir uns auf eine minimale Antwortdatei, mit der die beschriebenen Setup Stopper automatisch übersprungen werden. Kopieren Sie das folgende XML-Skript in eine PowerShell ISE Sitzung, legen Sie ein geeignetes Passwort für den lokalen Administrator fest

(Zeile 41) und speichern Sie dann die Datei auf Ihren Entwicklungsrechner (z.B. nach *D:\DSC-Lab\Unattend.xml*).



Anmerkung: Neben den beschriebenen Setup-Stoppfern habe ich in dieser XML-Datei noch ein paar zusätzliche Einstellungen definiert, die einem das Leben in einer Test- und Entwicklungsumgebung erleichtern, wie z.B. Aktivieren der Remote Desktop Funktion einschließlich der passenden Firewall-Regeln sowie das Ausschalten der IE-Härtung. Für eine produktive Umgebung sollten Sie sich genau überlegen, ob Sie diese sicherheitsrelevanten Einstellungen übernehmen wollen.

Um mit *Convert-WindowsImage.ps1* ein vollständiges Image mit dem Windows Server 2016 einschließlich der vorstehenden Datei *Unattend.xml* zu erzeugen, gehen Sie folgendermaßen vor:

1. Öffnen Sie die ISO-Datei von Windows Server 2016 durch einen Doppelklick im Windows Explorer. Der Inhalt erscheint in einem neuen virtuellen CD/DVD-Laufwerk.
2. Kopieren Sie den gesamten Inhalt der CD in ein Verzeichnis Ihrer Festplatte, z.B. *D:\DSC-Lab\WS2016*
3. Starten Sie jetzt eine PowerShell Konsolen- oder ISE-Sitzung mit Administratorrechten und wechseln Sie in das Verzeichnis *NanoServer\NanoServerImageGenerator* der Kopie des CD-Inhalts (also z.B. *D:\DSC-Lab\WS2016\NanoServer\NanoServerImageGenerator*). Dort finden Sie die aktuelle Version des PowerShell Skripts *Convert-WindowsImage.ps1*, mit dessen Hilfe Sie aus einer ISO- bzw. der darin enthaltenen .WIM-Datei ein Betriebssystem Image für VMs im Hyper-V erzeugen können. Die benötigte .WIM-Datei befindet sich im Verzeichnis *Sources* der CD-Kopie und hat den Namen *Install.wim*, für unser Beispiel also *D:\DSC-Lab\WS2016\Sources\install.wim*. Die Da-

tei enthält sowohl die Standard als auch die Datacenter Editionen von Windows Server 2016 jeweils mit und ohne Desktop Experience. Wir werden für unsere DSC-Experimente die *Windows Server 2016 Datacenter Evaluation (Desktop Experience)* verwenden, die in der .WIM-Datei den Index 4 trägt. Das Image werden wir als VHDX-Datei für Gen2 Hyper-V VMs erzeugen und direkt im Verzeichnis für die zu erzeugenden VMs – also *E:\Hyper-V* – unter dem Namen *WS2016\_GPT\_DCGUI.vhdx* ablegen.

Anmerkungen:

1. Falls Sie anstatt der Datacenter Edition von Windows Server 2016 eine andere Edition verwenden wollen, können Sie den entsprechenden Index mit dem DISM-Befehl ermitteln.

```
dism /Get-ImageInfo /Imagefile:D:\DSC-Lab\ws2016\sources\install.wim
Index: "1"
Name: "Windows Server 2016 Standard Evaluation"
Index: "2"
Name: "Windows Server 2016 Standard Evaluation (Desktop Experience)"
Index: "3"
Name: "Windows Server 2016 Datacenter Evaluation"
Index: "4"
Name: "Windows Server 2016 Datacenter Evaluation (Desktop Experience)"
```

2. *Convert-WindowsImage.ps1* ist eigentlich kein "echtes" PowerShell Skript, sondern enthält "nur" eine PowerShell Funktion. Um diese nutzen zu können, müssen wir das Skript zunächst per *Dot Sourcing* in den globalen Workspace der PowerShell laden und können dann erst die Funktion aufrufen.

Mit den folgenden PowerShell Befehlen erzeugen wir uns nun ein Basisimage für unsere weiteren Experimente, in das wir auch die zuvor erzeugte Datei *Unattend.xml* einfügen lassen:

```
cd 'D:\DSC-Lab\ws2016\nanoserver\NanoServerImageGenerator'
. .\Convert-WindowsImage.ps1
# Erzeuge dyn. VHDX für Gen2 Hyper-V VM mit Server 2016
Convert-WindowsImage
-SourcePath 'D:\DSC-Lab\w2016\sources\install.wim'
-Edition 4
-VHDPATH 'E:\Hyper-V\WS2016_GPT_DCGUI.vhdx'
-UnattendPath 'D:\DSC-Lab\unattend.xml'
-SizeBytes 40GB
-VHDFORMAT VHDX
-DiskLayout UEFI
```

Ergebnis: Die Image-Datei steht jetzt unter *E:\Hyper-V\WS2016\_GPT\_DCGUI.vhdx* zur Verfügung und wir können sie als Basisimage für VMs verwenden – sowohl als Parent Disk für differenzierende Laufwerke als auch als Quelle einer Kopie.

#### 2.1.4 Einsammeln und Installieren benötigter DSC-Ressourcen

Für unsere DSC-Experimente zum Erzeugen von VMs im Hyper-V und deren Konfiguration benötigen wir eine Reihe von DSC-Ressourcen (PowerShell Module), die nicht standardmäßig auf unserem Entwicklungs- und Testsystem vorhanden sind. Wir müssen sie vielmehr zuerst aus dem Internet von verschiedenen Quellen herunterladen und installieren. In der nachstehenden Tabelle habe ich die für die hier beschriebenen Beispiele verwendeten Module mit Versionsnummern und Hinweisen zum Download / Installieren zusammengefasst.

Anmerkung: Von einigen Modulen gibt es mittlerweile neuere Versionen, die Sie über andere URLs herunterladen können. Es spricht nichts dagegen, mit diesen neueren Versionen zu arbeiten. Beachten Sie jedoch, dass Sie dann in den Konfigurationsskripten teilweise die Versionsnummern anpassen müssen. Die hier genannten URLs sind in jedem Fall weiterhin gültig.

Modul	PSDesiredStateConfiguration
Version	
Quelle	lokal unter <i>C:\Windows\System32\WindowsPowerShell\v1.0\Modules\PSDesiredStateConfiguration</i>
Autor	Microsoft
Installation	nicht notwendig, da standardmäßig in Windows enthalten
Anmerkungen	Enthält die grundlegenden DSC-Ressourcen und Cmdlets; sollte in jedes Konfigurationskript importiert werden

Modul	xHyper-V
Version	3.6.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/xHyper-V/3.6.0.0">https://www.powershellgallery.com/packages/xHyper-V/3.6.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name xHyper-V -RequiredVersion 3.6.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Konfigurieren eines Hyper-V Hosts und zum Erzeugen von VMs

Modul	cHyper-V
Version	3.0.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/cHyper-V/3.0.0.0">https://www.powershellgallery.com/packages/cHyper-V/3.0.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name cHyper-V -RequiredVersion 3.0.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Erzeugen und Konfigurieren von Hyper-V- und VM-Netzwerkkomponenten

Modul	xComputerManagement
Version	1.9.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/xComputerManagement/1.9.0.0">https://www.powershellgallery.com/packages/xComputerManagement/1.9.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name xComputerManagement -RequiredVersion 1.9.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Konfigurieren systemspezifischer Eigenschaften wie Computernamen und Domain bzw. Workgroup Mitgliedschaft

Modul	xNetworking
Version	3.1.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/xNetworking/3.1.0.0">https://www.powershellgallery.com/packages/xNetworking/3.1.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name xNetworking -RequiredVersion 3.1.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Konfigurieren systemspezifischer IP-Eigenschaften wie IP-Adresse, DNS-Server-Adresse, Gateway IP-Adresse, usw.

Modul	xPSDesiredStateConfiguration
Version	5.1.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/xPSDesiredStateConfiguration/5.1.0.0">https://www.powershellgallery.com/packages/xPSDesiredStateConfiguration/5.1.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name xPSDesiredStateConfiguration -RequiredVersion 5.1.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Konfigurieren verschiedener Systemdienste, u.a. zum Erzeugen und Konfigurieren eines DSC Pull Servers

Modul	xWindowsUpdate
Version	2.5.0.0
Quelle	PowerShell Gallery - <a href="https://www.powershellgallery.com/packages/xWindowsUpdate/2.5.0.0">https://www.powershellgallery.com/packages/xWindowsUpdate/2.5.0.0</a>
Autor	PowerShell Team, Microsoft
Installation	<i>Install-Module -Name xWindowsUpdate -RequiredVersion 2.5.0.0 -Force -Verbose</i>
Anmerkungen	Enthält Ressourcen zum Erzeugen und Konfigurieren von Hyper-V- und VM-Netzwerkkomponenten

PowerShell Skript für den Download der Module:

```
Install-Module -Name xHyper-V -RequiredVersion 3.6.0.0 -Force -Verbose
Install-Module -Name cHyper-V -RequiredVersion 3.0.0.0 -Force -Verbose
Install-Module -Name xComputerManagement -RequiredVersion 1.9.0.0 -Force -Verbose
Install-Module -Name xNetworking -RequiredVersion 3.1.0.0 -Force -Verbose
Install-Module -Name xPSDesiredStateConfiguration -RequiredVersion 5.1.0.0 -Force -Verbose
Install-Module -Name xWindowsUpdate -RequiredVersion 2.5.0.0 -Force -Verbose
```

## 2.2 Ein DSC Konfigurationskript zum Erzeugen von VMs im Hyper-V

Nachdem wir jetzt alle Voraussetzungen in unserer DSC Lab-Umgebung geschaffen haben, können wir uns daranmachen, mit DSC eine VM im Hyper-V zu erzeugen. Laden Sie das nachstehende Konfigurationskript *DSCHyperV\_VMO.ps1* aus dem Lab-Verzeichnis *D:\DSC-Lab* in eine PowerShell ISE Sitzung, die mit Administratorrechten gestartet wurde.

```

1 Configuration DSCHyperV_VM {
2   # Optionale Parameterliste
3   param (...)
44
45 #region - Importieren benötigter DSC Ressourcen ...
50
51 #region - Variablendefinitionen ...
58
59   # mind. 1 Node Block für Zielsystem
60   Node $NodeName
61   {
62     #region - DSC-Ressourcenblöcke mit Konfigurationswerten...
175
176 }

```

Schauen wir uns die einzelnen Abschnitte dieses Konfigurationskripts etwas genauer an.

### 2.2.1 Parameterliste

Das Skript startet mit einer umfangreichen Parameterliste, über die fast alle Einstellungen zum Erzeugen einer VM im Hyper-V vorgegeben werden können, wobei ich für die meisten Parameter bereits Standardeinstellungen vorbelegt habe, die für unsere DSC-Experimente geeignet sind. Hierdurch vereinfacht sich der Aufruf der Konfiguration auf wenige Angaben wie Name der VM oder Pfadname der Datei mit dem Basisimage. Damit wird dann eine virtuelle Maschine auf unserem DSC Entwicklungsrechner ("localhost") mit diesen Einstellungen erzeugt:

- Anzahl Prozessoren: 2
- Dynamischer RAM-Speicher: 512 MB – 16 GB; Start mit 2GB
- Betriebssystem Disk: Differenzierend zur angegebenen Basisimage Datei
- Netzwerkadaptername: MGMT ("Management")
- verbunden mit virtuellem Switch: NatSwitch80

```

1 Configuration DSCHyperV_VM {
2   # Optionale Parameterliste
3   param (
4     # Zielsystem auf dem die VM erzeugt werden soll
5     [Parameter()]
6     [string]$NodeName = 'localhost',
7
8     # Pfad auf dem Zielsystem, unter dem die VM erzeugt werden soll
9     [Parameter()]
10    [string]$HyperVPath = 'E:\Hyper-V',
11
12    # Name der VM
13    [Parameter(Mandatory)]
14    [string]$VMName,
15
16    # Pfadname des Basis Image der VM
17    [Parameter(Mandatory)]
18    [string]$ParentVhdPath,
19
20    # Kopie des Basis Image oder Differencing Disk erstellen
21    [ValidateSet("Differencing", "Copy")]
22    [string]$OSDiskType = "Differencing",
23
24    # Hyper-V Switch, mit dem die VM verbunden werden soll
25    [Parameter()]
26    [string]$SwitchName = 'NatSwitch80',
27
28    # Startup RAM der VM
29    [ValidateRange(1,8)]
30    [UInt64]$ProcessorCount = 2,
31
32    # Startup RAM der VM
33    [ValidateRange(512MB,16GB)]
34    [UInt64]$StartupMemory = 2GB,
35
36    # Minimum RAM der VM. Hierdurch wird dynamic RAM aktiviert
37    [ValidateRange(512MB,16GB)]
38    [UInt64]$MinimumMemory = 1GB,
39
40    # Maximum RAM der VM. Hierdurch wird dynamic RAM aktiviert
41    [ValidateRange(2GB,16GB)]
42    [UInt64]$MaximumMemory = 16GB
43  )

```

Natürlich können Sie bei Bedarf die Vorbelegungen der verschiedenen Parameter beim Aufruf des Skripts durch Angabe eigener Werte überschreiben.

## 2.2.2 Importieren benötigter DSC-Ressourcen

Als erstes binden wir die in der *Configuration* verwendeten DSC-Ressourcen mit dem Befehl *Import-DscResource* ein.

```
44 #region - Importieren benötigter DSC Ressourcen
45     Import-DscResource -ModuleName 'PSDesiredStateConfiguration'
46     Import-DscResource -module xHyper-V -ModuleVersion 3.6.0.0
47     Import-DscResource -module xHyper-V -ModuleVersion 3.0.0.0
48 #endregion
```

## 2.2.3 Variablendefinitionen

An verschiedenen Stellen in den Ressourcen Blöcken des Skripts benötigen wir mehrfach die gleichen Pfadnamen. Deshalb definieren wir hier an zentraler Stelle Variable dafür, so dass eine spätere Wartung erleichtert wird.

```
50 #region - Variablendefinitionen
51     # Einige Variable
52     $VMPath = Join-Path -Path $HypervPath -ChildPath $VMName           # Verzeichnispfad für die VM
53     $VhdPath = Join-Path -Path $VMPath -ChildPath "Virtual Hard Disks" # Verzeichnispfad für die virt. Disks der VM
54     $VhdFilePath = Join-Path -Path $VhdPath -ChildPath "$VMName.vhdx" # Pfadname der Boot Disk
55     $VMSetupFilePath = Join-Path -Path (Get-Location).Path -ChildPath "$VMName\Setup\*" # Setup Dateien für die VM
56 #endregion
```

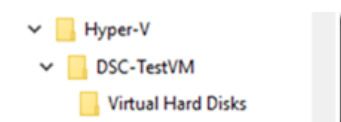
## 2.2.4 Die Konfiguration im Detail

Innerhalb des *Node* Blocks werden jetzt die einzelnen Aktionen für das Erzeugen einer VM festgelegt.

### 2.2.4.1 Verzeichnisse für die VM erzeugen

```
61     # Das Verzeichnis für die VM erzeugen
62     File VMPath
63     {
64         Ensure = 'Present'
65         Type = 'Directory'
66         DestinationPath = $VMPath
67     }
68     # Das Verzeichnis für die virtuellen Disks der VM erzeugen
69     File VHDPath
70     {
71         Ensure = 'Present'
72         Type = 'Directory'
73         DestinationPath = $VhdPath
74         DependsOn = '[File]VMPath'
75     }
```

Mit Hilfe der File Ressource stellen wir sicher, dass im Verzeichnis für die erzeugten VMs (in unserer Lab-Umgebung *E:\Hyper-V*) ein Unterverzeichnis für die neue VM existiert einschließlich eines darin enthaltenen Ordners für die virtuellen Laufwerke der VM. Wir erhalten dann also die nebenstehende Verzeichnisstruktur (der Name der VM ist hier *DSC-TestVM*).



### 2.2.4.2 Die Startdatei für die VM erzeugen

Jetzt müssen wir im gerade erzeugten Verzeichnis die Startdatei für die VM in Abhängigkeit des Parameters *\$OSDiskType* erzeugen. Ist der Parameterwert *'Copy'*, können wir mit der DSC Standard-Ressource *File* eine Kopie des im Parameter *\$ParentVhdPath* angegebenen Betriebssystem-Basisimage erstellen. Ist für *\$OSDiskType* als Wert *'Differencing'* angegeben, müssen wir auf eine Ressource *xVHD* aus dem PowerShell Modul *xHyper-V* zurückgreifen. Damit können wir eine differenzierende *.VHDX*-Datei zum angegebenen Betriebssystem-Basisimage erzeugen. Vorteil: Die ist kleiner als eine Kopie und nimmt nur die Änderungen auf, die im Betrieb der VM entstehen.

```

77 # Soll die VM aus einer Kopie oder aus einer Differential Disk starten
78 if ( $OSDiskType -eq "Copy")
79 {
80     # Kopie des Basis Image im Disk Verzeichnis erstellen
81     File BootVHD
82     {
83         SourcePath = $ParentVhdPath
84         DestinationPath = $VhdFilePath
85         Ensure = 'Present'
86         DependsOn = '[File]VHDPATH'
87     }
88 }
89 else
90 {
91     # Differential Disk erzeugen
92     xVHD BootVHD
93     {
94         Ensure = 'Present'
95         Name = $VMName
96         Path = $VhdPath
97         ParentPath = $ParentVhdPath
98         Generation = 'vhdx'
99         DependsOn = '[File]VHDPATH'
100     }
101 }

```

#### 2.2.4.3 Setup Dateien für die VM in die Startdatei kopieren

Mit der im nächsten Ressourcenblock angegebenen Ressource *xVHDFile* (ebenfalls aus dem Modul *xHyper-V*) können wir – falls vorhanden – VM-spezifische Konfigurationsdaten in das soeben erstellte Startlaufwerk der VM kopieren. Die Ressource erzeugt intern für die bei *VhdPath* angegebene VHDX-Datei temporär ein Laufwerk und kopiert anschließend mit Hilfe der *File* Ressource den Inhalt des als *SourcePath* angegebenen Verzeichnisses in den *DestinationPath*, der relativ zur Root des temporären Laufwerks anzugeben ist. Wie solche Konfigurationsdaten strukturiert sein können, werden wir uns später noch anschauen, wenn wir unseren DSC Web Pull Server konfigurieren.

```

103 # copy setup files for VM if available
104 if ( Test-Path $VMSetupFilePath )
105 {
106     xVHDFile SetupFiles
107     {
108         VhdPath = $VhdFilePath
109         FileDirectory = MSFT_xFileDirectory {
110             SourcePath = $VMSetupFilePath
111             DestinationPath = "."
112         }
113     }
114 }

```

#### 2.2.4.4 Die VM im Hyper-V erzeugen

Jetzt haben wir alle Informationen und Daten zusammengestellt, um mit Hilfe der *xVMHyperV* Ressource aus dem Modul *xHyper-V* eine neue VM anzulegen. Ich verwende bevorzugt Gen2 VMs, da sie flexibler zu handhaben sind, wenn man Änderungen an der ursprünglichen Konfiguration durchführen will (z.B. zusätzliche virtuelle Hardware im laufenden Betrieb hinzufügen).

Die VM wird zunächst nur angelegt, aber noch nicht gestartet (*State='Off'*). Wir haben dadurch die Möglichkeit, noch Konfigurationsänderungen festzulegen, die nur bei ausgeschalteten VMs möglich sind. Wie wir gleich sehen werden, werde ich die Netzwerkkonfiguration noch ändern.



```

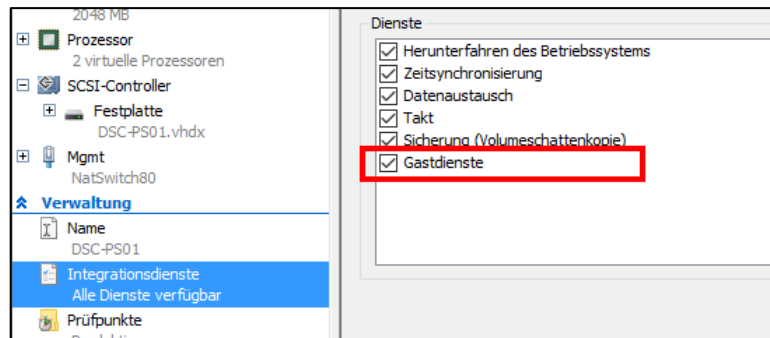
116 # Die VM im Hyper-V anlegen
117 xVMHyperV CreateVM
118 {
119     Name = $VMName
120     SwitchName = $SwitchName
121     VhdPath = Join-Path -Path $VhdPath -ChildPath "$VMname.vhdx"
122     ProcessorCount = 2
123     StartupMemory = $StartupMemory
124     MaximumMemory = $MaximumMemory
125     MinimumMemory = $MinimumMemory
126     State = 'Running'
127     Generation = 2
128     # EnableGuestService = $true
129     Path = $HypervPath
130 }

```

Vielleicht fragen Sie sich, warum ich die Zeile `EnableGuestService = $true` auskommentiert habe. Wenn wir in einer nicht-englischen Hyper-V Umgebung diese Ressourcen-Eigenschaft setzen, führt das beim Ausführen der Konfiguration zu einem Abbruchfehler. Hintergrund ist ein Bug im Ressourcen-Modul. Dort wird der Service hart codiert mit seiner US-englischen Bezeichnung *Guest Service Interface* angesprochen. In einer deutschen Windows Umgebung hat der Service hingegen z.B. die Bezeichnung *Gastdienstschnittstelle*, mit der Folge, dass die Ressource den Service nicht findet und einen Fehler ausgibt.

Wenn Sie in einer US-englischen Umgebung arbeiten, können Sie problemlos den Kommentar wegnehmen.

Bei nicht-englischen Betriebssystemen können Sie später den Service über die GUI des Hyper-V Managers aktivieren.



Ich bin mit dem Autor der Ressource [Daniel Scott-Raynsford](#) über seine [Issue Website bei Github](#) in Kontakt. Eine Korrektur wäre relativ simpel, allerdings mit einer Modifikation des Codes im Ressourcen Modul verbunden, was hier jetzt den Rahmen unserer DSC-Experimente sprengen würde. Es ist zu erwarten, dass dies in einer der nächsten Releases dieser Ressource korrigiert wird.

#### 2.2.4.5 Dem Netzwerkadapter der VM einen vernünftigen Namen geben

Eine durchaus vernünftige, für unsere Zwecke aber etwas unschöne Eigenschaft des Hyper-V PowerShell Kommandos `New-VM` – auf dem die DSC-Ressource `xVMHyperV` basiert – ist die Tatsache, dass grundsätzlich ein Netzwerkadapter für die VM erzeugt wird. Der Name dieses Adapters ist leider abhängig von der Sprache der Windows Version. In einer deutschen Hyper-V Umgebung heißt er z.B. *Netzwerkkarte*, in englischen Versionen *Network Adapter*. Eine Änderung des Namens wäre über die GUI im Hyper-V Manager möglich und auch in der PowerShell gibt es hierfür das Cmdlet `Rename-VMNetworkAdapter`. Leider habe ich aber keine DSC-Ressource gefunden, mit der dies machbar wäre – eine Herausforderung für Experten, eine entsprechende Ressource selbst zu schreiben. Um in meinen VMs den Namen des Netzwerkadapters in *Mgmt* umzuändern, werde ich ein bisschen tricksen:

Zunächst entferne ich mit Hilfe der DSC-Ressource `cVMNetworkAdapter` aus dem Modul `cHyper-V` die Netzwerkkarten mit den unerwünschten Namen durch die Angabe von `Absent` bei der Eigenschaft `Ensure`. Wenn es keine Netzwerkkarte mit dem entsprechenden Namen gibt, ist das kein Problem; die Ressource führt dann eben keine Aktion aus.

Jetzt können wir unserer VM ebenfalls mit `cVMNetworkAdapter` einen neuen Adapter mit dem gewünschten Namen `Mgmt` verpassen und diesen mit dem entsprechenden Switch `NatSwitch80` verbinden. Und damit dieser Name später auch in der VM genutzt werden kann, setze ich mit Hilfe der Ressource `cNetworkAdapterSettings` die Eigenschaft `DeviceNaming` des Netzwerkadapters auf `$true`. Details zu dieser neuen Eigenschaft von Windows Server 2016 bzw. Windows 10 hat Jan Kappen bereits in einem früheren [Blogpost](#) beschrieben. Ich werde später beim Konfigurieren von VMs darauf zurückgreifen.

```

131 # Entfernen des Standard Netzwerkadapters und Neuanlegen mit vernünftigen Namen
132 cVMNetworkAdapter StdNICen
133 {
134     Id = ([guid]::NewGuid()).guid
135     SwitchName = $SwitchName
136     Name = 'Network Adapter'           # Name des virt. Netzwerkadapters in englischen OS
137     VMName = $VMName
138     Ensure = 'Absent'
139     DependsOn = '[xVMHyperV]CreateVM'
140 }
141 cVMNetworkAdapter StdNICde
142 {
143     Id = ([guid]::NewGuid()).guid
144     SwitchName = $SwitchName
145     Name = 'Netzwerkkarte'             # Name des virt. Netzwerkadapters in deutschen OS
146     VMName = $VMName
147     Ensure = 'Absent'
148     DependsOn = '[xVMHyperV]CreateVM'
149 }
150 cVMNetworkAdapter Mgmt
151 {
152     Id = ([guid]::NewGuid()).guid
153     Name = 'Mgmt'
154     SwitchName = $SwitchName
155     VMName = $VMName
156     Ensure = 'Present'
157     DependsOn = '[xVMHyperV]CreateVM'
158 }
159 cVMNetworkAdapterSettings Mgmt
160 {
161     Id = ([guid]::NewGuid()).guid
162     Name = 'Mgmt'
163     VMName = $VMName
164     SwitchName = $SwitchName
165     DeviceNaming = 'On'               # Nur bei Server 2016 / Windows 10 v 1607
166     DependsOn = '[cVMNetworkAdapter]Mgmt'
167 }

```

Ich geb's ja zu, ein bisschen "von hinten durch die Brust ins Knie"; wenn jemand eine bessere Idee hat, lasst es mich wissen

### 2.2.5 Eine erste Test-VM

Jetzt können wir einen ersten Test mit dem vorstehenden DSC-Konfigurationsskript durchführen und eine VM mit dem Namen `DSC-TestVM` erzeugen Öffnen Sie das Skript in einer PowerShell ISE Sitzung, die mit Administratorrechten gestartet wurde und führen Sie folgende Schritte aus:

1. Wechseln Sie in das `DSC-Lab` Verzeichnis
2. Laden Sie das Konfigurationsskript per `Dot Sourcing` in den globalen Arbeitsbereich der PowerShell ISE Sitzung

```
PS D:\dsc-lab> . .\DSCHyperV_VM0.ps1
```

```
PS D:\dsc-lab>
```

3. Kompilieren Sie das Skript mit folgendem Befehl

```

DSCHyperV_VM -VMname DSC-TestVM `
  -ParentVhdPath E:\Hyper-V\WS2016_GPT_DCGUI.vhdx `
  -OutputPath .\DSC-TestVM `
  -Verbose

verzeichnis: D:\dsc-lab\DSC-TestVM

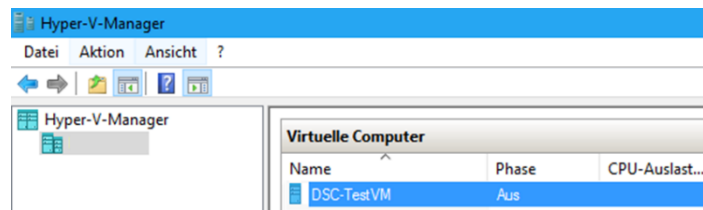
Mode                LastWriteTime         Length Name
----                -
-a----            05.06.2017   12:45           9990 localhost.mof

```

#### 4. Starten Sie die Konfiguration der VM

```
Start-DscConfiguration -wait -force -verbose -Path .\DSC-TestVM
```

Im Konsolenfenster der PowerShell ISE können wir den Ablauf mitverfolgen. Zum Abschluss sehen wir im Hyper-V Manager, dass eine neue VM mit dem Namen *DSC-TestVM* erzeugt wurde.



#### 5. Jetzt können wir die VM starten

```
Start-VM -Name DSC-TestVM
```

und uns über den Hyper-V Manager mit ihr verbinden.

### 2.3 Bereitstellen VM-spezifischer Konfigurationsdaten am Beispiel eines *Web Pull Servers*

Schön, jetzt haben wir also eine virtuelle Maschine im Hyper-V am Laufen. Diese beinhaltet aber keine speziellen Funktionen, da sie aus einem "neutralen" Betriebssystemimage erzeugt wurde. Wir müssen uns deshalb eine Vorgehensweise überlegen, wie man sie nach dem ersten Start für spezielle Funktionen konfigurieren kann – natürlich soweit möglich mit DSC. Dazu müssen die notwendigen DSC-Konfigurationsdaten wie Skripte und DSC-Ressourcen irgendwie in die VMs gebracht und dann der *Local Configuration Manager (LCM)* in der VM veranlasst werden, die Konfiguration abzuarbeiten. Als Beispiel für eine Möglichkeit, dies zu bewerkstelligen, soll die Konfiguration eines *DSC Web Pull Servers* dienen.

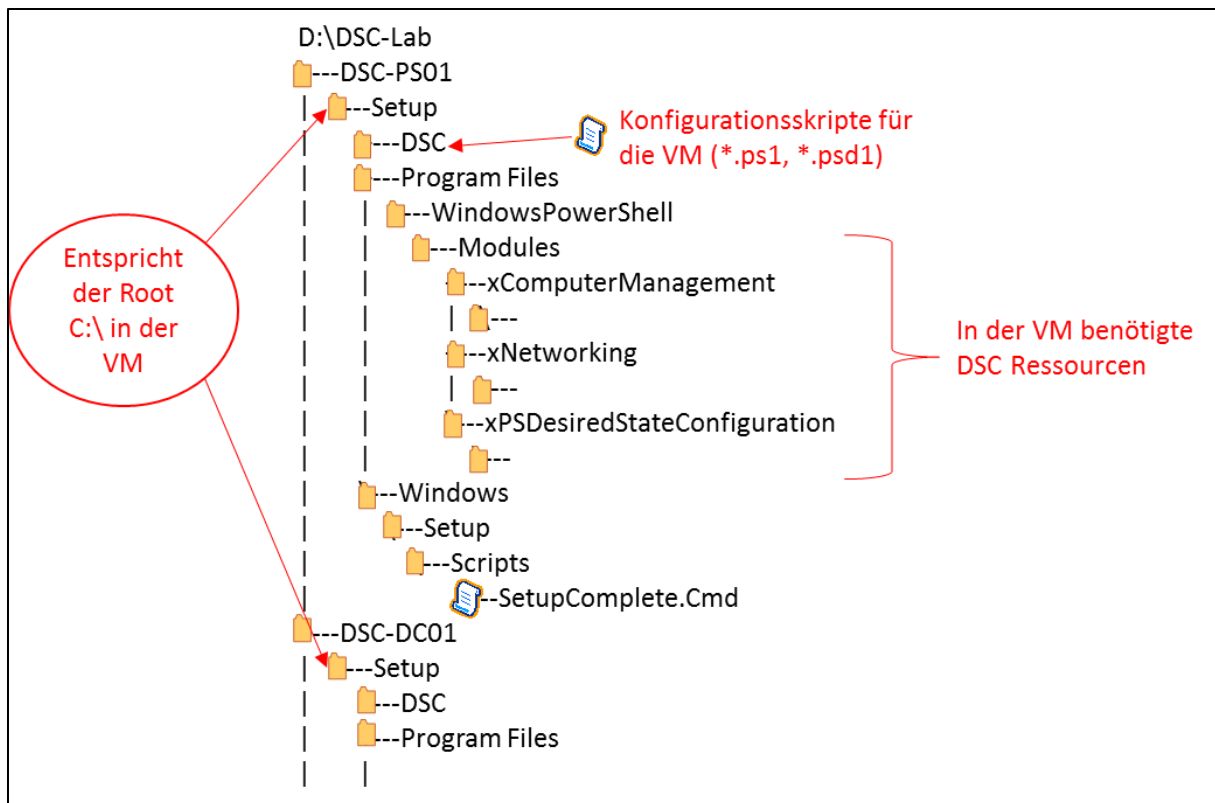
#### 2.3.1 Was ist ein DSC Web Pull Server?

Im Teil 1 dieses White Papers habe ich bereits kurz die beiden [Betriebsmodi Push und Pull](#) für DSC beschrieben. Für den Pull Betrieb benötigen wir (mindestens) ein Server System, das die Konfigurationsdaten bereitstellt. Dabei gibt es 2 Varianten: Entweder über eine SMB-Freigabe oder über einen Web Service. Im letzteren Fall sprechen wir von einem *DSC Web Pull Server*. Auf das Arbeiten mit einem solchen Web Pull Server will ich hier zunächst nicht näher eingehen. Dies wollen wir später nachholen.

#### 2.3.2 Bereitstellen der Konfigurationsdaten

Für meine DSC Lab-Umgebung habe ich folgende Vorgehensweise zum Bereitstellen von Konfigurationsdaten gewählt: Für jede VM wird ein eigenes Unterverzeichnis mit dem Namen der VM im Projektverzeichnis *D:\DSC-Lab* angelegt und dann in einem weiteren Unterverzeichnis eine Verzeichnisstruktur *Setup* erstellt. Diese enthält die DSC Konfigurationsdaten in exakt der gleichen Form wie sie in der Ziel-VM ausgehend von der Root des Startlaufwerks C: benötigt werden.

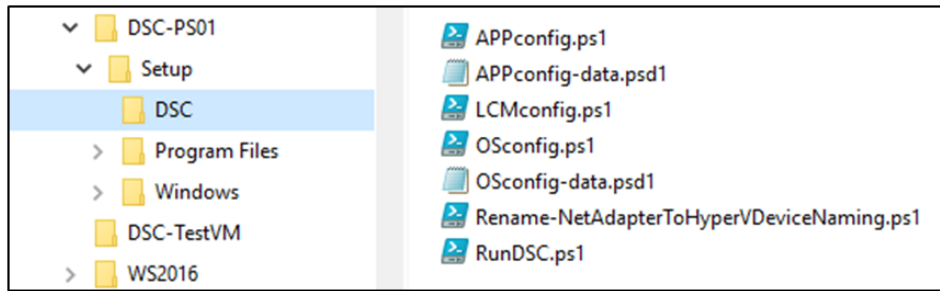
Für unseren Web Pull Server *DSC-PS01* würde dies wie folgt aussehen



- Das Verzeichnis *D:\DSC-Lab\DSC-PS01* enthält alle Daten für eine VM mit dem Namen *DSC-PS01* (unseren DSC Web Pull Server).
- Das Verzeichnis *D:\DSC-Lab\DSC-PS01\Setup* stellt die Root des Startlaufwerks C: der VM *DSC-PS01* dar.
- In das Unterverzeichnis *D:\DSC-Lab\DSC-PS01\Setup\Program Files\WindowsPowerShell\Modules* werden alle DSC Ressourcen Module kopiert, die für die DSC-Konfiguration der VM notwendig sind. Man kann sich diese aus dem Standardverzeichnis für DSC-Ressourcen auf unserem Hyper-V Test- und Entwicklungsrechner - also aus *C:\Program Files\WindowsPowerShell\Modules* - per Copy-Befehl holen.
- Im Unterverzeichnis *D:\DSC-Lab\DSC-PS01\Setup\Windows\Setup\Scripts* wird eine Datei mit dem Namen *SetupComplete.cmd* abgelegt, über die die DSC-Konfiguration in der VM angestoßen wird. Dies ist eine "klassische" Windows Batch Datei, die vom Windows Setup automatisch zum Abschluss einer erfolgreichen Installation – also noch vor der ersten Login Möglichkeit – aufgerufen wird. Es ist der ideale Zeitpunkt, um die DSC-Konfiguration einer neu erzeugten VM anzustoßen. Dazu rufen wir ein PowerShell Skript *RunDSC.ps1* aus dem DSC-Verzeichnis der VM auf, das dann die eigentliche DSC-Konfiguration ausführt:

```
@echo off
PowerShell.exe -ExecutionPolicy RemoteSigned -File C:\DSC\RunDSC.ps1
exit
```

- Das Unterverzeichnis *D:\DSC-Lab\DSC-PS01\Setup\DSC* enthält alle Skripte und Daten, die für die DSC-Konfiguration der VM erforderlich sind. Für unseren Web Pull Server *DSC-PS01* habe ich die folgenden Dateien hinterlegt, auf deren Inhalt ich gleich zu sprechen komme.



Ich habe mir PowerShell Skripte geschrieben, die diese Verzeichnis- und Dateistrukturen für eine VM erzeugen. Ein Beispiel für unseren Web Pull Server *DSC-PS01* finden Sie in der zum Download angebotenen Datei [DSC-Lab.zip](#) unter dem Namen *New-DSC-PS01\_SetupFiles.ps1*. Ich will hier nicht näher auf dieses Skript eingehen, sondern nur die Inhalte der damit erzeugten Dateien etwas genauer betrachten. Hält man sich an die hier beschriebene Vorgehensweise zum Bereitstellen von Konfigurationsdaten, so kann man dieses Skript *New-DSC-PS01\_SetupFiles.ps1* als Vorlage auch für andere VMs verwenden. Die einzelnen Abschnitte muss man dann natürlich an die Anforderungen der VM anpassen.

Existiert beim Erzeugen einer VM mit dem im vorherigen Kapitel beschriebenen Konfigurationsskript *DscHyperV\_VMO* eine solche *Setup*-Struktur mit Konfigurationsdaten, wird diese mit der Ressource *xVhdFile* in die Root des Startlaufwerks der VM kopiert (Zeile 103-114).

## 2.4 Die Konfiguration unseres DSC Web Pull Servers DSC-PS01 mit partiellen Konfigurationen

Im voranstehenden Teil 1 dieses White Papers habe ich das Thema [Partielle Konfigurationen](#) bereits kurz angeschnitten. Am Beispiel unseres Web Pull Servers will ich dies nun etwas genauer beleuchten.

Wenn wir eine neue VM erzeugt haben, müssen wir sie noch konfigurieren. Typischerweise sind dabei zumindest folgende Aktionen notwendig.

- Wir müssen das Betriebssystem konfigurieren, also z.B. IP-Adressen zuweisen und dem Betriebssystem in der VM einen vernünftigen Computernamen geben. Ich will diesen Schritt hier als *OSconfig* bezeichnen.
- Wir müssen die in der VM benötigten Anwendungskomponenten installieren und konfigurieren. Diesen Schritt will ich hier mit *APPconfig* bezeichnen. Die durchzuführenden Aktionen sind natürlich abhängig von den Funktionalitäten, die in der VM verfügbar sein sollen. Für einen DSC Web Pull Server müssen wir dazu die Windows Komponenten *DSC Service* und *IIS* sowie eine *Web Service Anwendung* - den *DSC Web Pull Server* - installieren und konfigurieren.
- Dann müssen wir noch den *Local Configuration Manager (LCM)* in der VM konfigurieren und ihm die notwendigen partiellen Konfigurationsschritte bekannt machen. Ich will dies hier als *LCMconfig* bezeichnen.

In der Praxis bietet es sich an, für jede der Aktionen ein eigenes Konfigurationsskript zu erstellen.

### 2.4.1 Die Betriebssystemkonfiguration (*OSconfig*)

Beginnen wir mit der Betriebssystemkonfiguration. Dazu habe ich ein Konfigurationsskript *OSconfig.ps1* sowie eine PowerShell Datendatei *OSconfig-data.psd1* erstellt. Beide Dateien liegen im Verzeichnis *C:\DSC* unserer VM:

*OSconfig.ps1*:

```
1 Configuration osconfig {
2
3     Import-DscResource -Module PSDesiredStateConfiguration
4     Import-DscResource -Module xComputerManagement -ModuleVersion 1.9.0.0
5     Import-DscResource -Module xNetworking -ModuleVersion 3.1.0.0
6
7     Node $AllNodes.NodeName {
8         xIPAddress SetIP {
9             IPAddress = $Node.IPAddress
10            InterfaceAlias = $Node.InterfaceAlias
11            PrefixLength = $Node.PrefixLength
12            AddressFamily = $Node.AddressFamily
13        }
14        xDefaultGatewayAddress SetDefaultGateway {
15            Address = $Node.DefaultGateway
16            InterfaceAlias = $Node.InterfaceAlias
17            AddressFamily = $Node.AddressFamily
18        }
19        xDNSServerAddress SetDNS {
20            Address = $Node.DNSAddress
21            InterfaceAlias = $Node.InterfaceAlias
22            AddressFamily = $Node.AddressFamily
23        }
24        xComputer SetName {
25            Name = $Node.MachineName
26        }
27    }
28 }
29 OSconfig -ConfigurationData osconfig-data.psd1
```

Zum Festlegen der IP-Konfiguration der VM verwenden wir 3 Ressourcen aus dem *xNetworking* Modul:

- *xIPAddress*: Legt die IP-Adresse des bei *InterfaceAlias* angegebenen Netzwerkadapters fest. *InterfaceAlias* ist der Name des Adapters, wie er in der VM erscheint. Über *PrefixLength* wird die Netzmaske festgelegt und über *AddressFamily* legen wir fest, ob es sich um eine IPv4 oder IPv6 Adresse handelt.
- *xDefaultGateway*: Hier wird die IP-Adresse für das Default Gateway festgelegt - in unserem Beispiel ist dies die Adresse unseres NAT-Switch.
- *xDNSServerAddress*: Hiermit werden die IP-Adressen unserer DNS-Server festgelegt. In einem späteren Beitrag werde ich noch auf das Bereitstellen eines DNS-Servers per DSC eingehen. Da dieser Server - er wird die IP-Adresse 192.168.80.10 erhalten - momentan in unserer Lab-Umgebung noch nicht existiert, definiere ich hier zusätzlich als alternativen DNS-Server meinen Internet Router (192.168.1.1), der dann automatisch verwendet wird, wenn der primäre DNS-Server nicht erreichbar ist.

Zum Festlegen des Computernamens verwenden wir eine Ressource aus dem Modul *xComputerManagement*:

- *xComputer*: Diese Ressource weist dem Betriebssystem in der VM den gewünschten Computernamen zu und kümmert sich dann um den danach notwendigen Neustart des Systems.

*OSconfig-data.psd1:*

```

1  @{
2  AllNodes = @(
3  @ {
4  NodeName = 'localhost'
5  MachineName = 'DSC-PS01'
6  InterfaceAlias = 'Mgmt'
7  AddressFamily = 'IPv4'
8  IPAddress = '192.168.80.250'
9  DefaultGateway = '192.168.80.1'
10 PrefixLength = '24'
11 DNSAddress = '192.168.80.10', '192.168.1.1'
12 }
13 )
14 }

```

Wozu diese Datendatei? Das Konfigurationsskript in *OSconfig.ps1* ist neutral und kann in dieser Form für jede VM verwendet werden. Die echten Daten werden beim Kompilieren der Konfiguration in Form einer PowerShell Datendatei mit dem Parameter *-ConfigurationData* übergeben (siehe Zeile 29). Für eine andere VM mit anderen Daten müssen dann nur die Werte in der Datendatei angepasst werden. In der [TechNet Library](#) finden Sie weitere Informationen zum Trennen von Konfigurationen und zugehörigen Daten.

2.4.2 Die Anwendungskonfiguration (*APPconfig*) für unseren Web Pull Server *DSC-PS01*  
Analog zur Betriebssystemkonfiguration bauen wir auch die Anwendungskonfiguration auf: Ein Konfigurationsskript mit dem Namen *APPconfig.ps1* und eine Datendatei *APPconfig-data.psd1*.

*APPconfig.ps1:*

```

1  Configuration APPconfig {
2
3      Import-DscResource -Module PSDesiredStateConfiguration
4      Import-DscResource -Module xPSDesiredStateConfiguration -Moduleversion 5.1.0.0
5
6      Node $AllNodes.NodeName {
7
8          # Konfigurationsblöcke für Anwendungen und Serverrollen
9
10         # Beispiel für DSC Pull Server
11
12         windowsFeature DSCServiceFeature
13         {
14             Ensure = "Present"
15             Name = "DSC-Service"
16         }
17         windowsFeature webServerManagement
18         {
19             Ensure = 'Present'
20             Name = "web-Mgmt-Tools"
21             DependsOn = "[windowsFeature]DSCServiceFeature"
22         }
23         xDscWebService PSDSCPullServer
24         {
25             Ensure = "Present"
26             EndpointName = "PSDSCPullServer"
27             Port = 8080
28             PhysicalPath = "$env:SystemDrive\inetpub\wwwroot\PSDSCPullServer"
29             AcceptSelfSignedCertificates = $false
30             CertificateThumbPrint = "AllowUnencryptedTraffic"
31             ModulePath = "$env:PROGRAMFILES\windowsPowerShell\DscService\Modules"
32             ConfigurationPath = "$env:PROGRAMFILES\windowsPowerShell\DscService\Configuration"
33             State = "Started"
34             UseSecurityBestPractices = $false
35             DependsOn = "[windowsFeature]DSCServiceFeature"
36         }
37         File RegistrationKeyFile
38         {
39             Ensure = 'Present'
40             Type = 'File'
41             DestinationPath = "$env:ProgramFiles\windowsPowerShell\DscService\RegistrationKeys.txt"
42             Contents = $Node.DSCRegistrationKey
43         }
44     }
45 }
46 APPconfig -ConfigurationData APPconfig-data.psd1

```

APPconfig-data.psd1:

```

1  @{
2      AllNodes = @(
3          @{
4              NodeName = 'localhost'
5              DSCRegistrationKey = '14fc8e72-5036-4e79-9f89-5382160053aa'
6          }
7      )
8  }

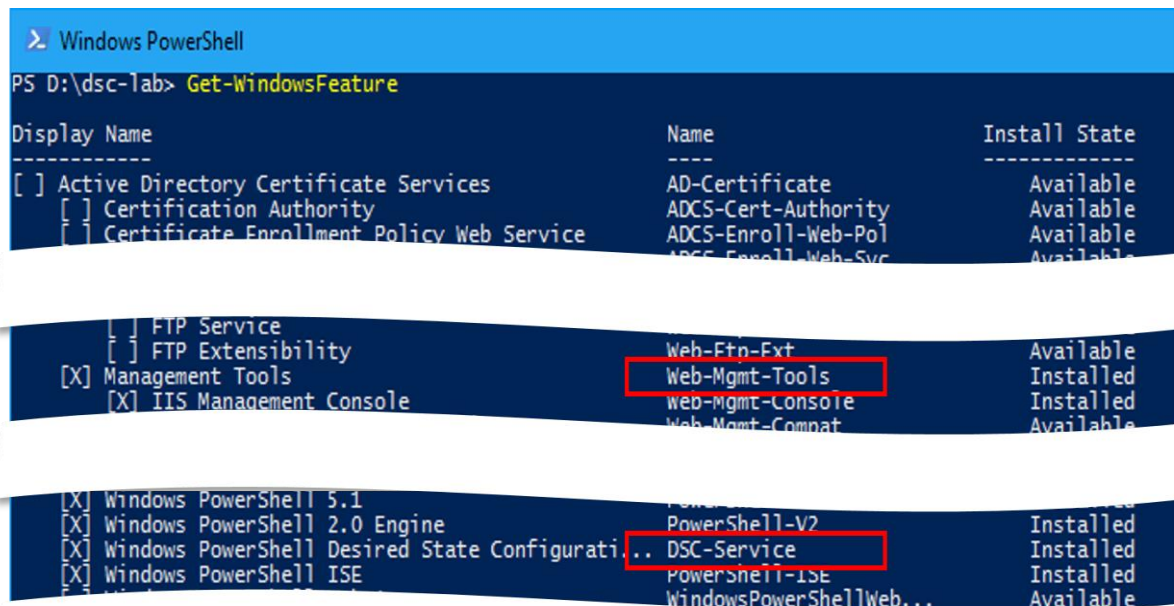
```

Um einen DSC Web Pull Server zu konfigurieren, müssen wir zunächst einige Windows Komponenten installieren. Hierzu steht im DSC Standardmodul *PSDesiredStateConfiguration* eine Ressource *WindowsFeature* zur Verfügung.

Die Ressource *WindowsFeature* erwartet den Namen der zu installierenden Windows Komponente als Wert bei der Eigenschaft *Name*. Doch wie ermittelt man diesen Namen? Geben Sie in einem PowerShell Konsolenfenster folgenden Befehl ein:

**Get-WindowsFeature**

Als Ergebnis erhalten Sie eine lange 3-spaltige Liste mit allen installierbaren Windows Komponenten. Die in der 2. Spalte mit der Überschrift "Name" ausgegebenen Bezeichnungen sind die Werte, die von der Ressource *WindowsFeature* erwartet werden.



Display Name	Name	Install State
[ ] Active Directory Certificate Services	AD-Certificate	Available
[ ] Certification Authority	ADCS-Cert-Authority	Available
[ ] Certificate Enrollment Policy Web Service	ADCS-Enroll-Web-Pol	Available
[ ] FTP Service	Web-Ftp-Ext	Available
[ ] FTP Extensibility	Web-Mgmt-Tools	Installed
[X] Management Tools	Web-Mgmt-Console	Installed
[X] IIS Management Console	Web-Mgmt-Compat	Available
[X] Windows PowerShell 5.1	PowerShell-V2	Installed
[X] Windows PowerShell 2.0 Engine	DSC-Service	Installed
[X] Windows PowerShell Desired State Configurati...	PowerShell-ISE	Installed
[X] Windows PowerShell ISE	WindowsPowerShellWeb...	Available

Die erste Komponente, die für einen DSC Web Pull Server installiert werden muss, ist der *DSC Service*. Die Ressource *WindowsFeature* sorgt dafür, dass alle weiteren dafür notwendigen Windows Komponenten ebenfalls installiert werden, insbesondere hier der *IIS*, jedoch ohne die GUI Tools mit dem Namen *Web-Mgmt-Tools*. Diese installieren wir mit einer weiteren *WindowsFeature* Instanz. Durch Angabe von *'Present'* bei der Eigenschaft *Ensure* wird die Komponente jeweils installiert, mit der Angabe *'Absent'* würde sie deinstalliert werden.



Anmerkung: Die Ressource *WindowsFeature* kann nur auf Server Betriebssystemen verwendet werden. Sie setzt auf dem *Server Manager* des Betriebssystems auf, der bei Client Betriebssystemen wie Windows 10 nicht vorhanden ist. Als Alternative steht dort eine Ressource *WindowsOptionalFeature* mit ähnlicher Funktionalität zur Verfügung.

Nun können wir mit der Ressource *xDSCWebService* aus dem Modul *xPSDesiredStateConfiguration* die eigentliche *Web Service Anwendung* für den Web Pull Server installieren.

Normalerweise sollte die Kommunikation zwischen einem DSC Web Pull Server und den Clients mit SSL verschlüsselt werden. Für unsere Lab-Umgebung verzichten wir darauf, um die Sache nicht zu kompliziert zu gestalten. Deshalb setzen wir die entsprechenden Eigenschaften wie folgt:

```
AcceptSelfSignedCertificates = $false
CertificateThumbPrint       = "AllowUnencryptedTraffic"
UseSecurityBestPractices    = $false
```

Weiterhin verlangt die Ressource, dass wir einen Namen festlegen, unter dem der Web Service erreichbar ist ("*Endpoint*") und den zugehörigen IP-Port.

Und dann müssen wir natürlich noch die Pfadnamen festlegen, wo die Web Service Anwendung installiert werden soll (*PhysicalPath* - hier in der Root des IIS) und wo Konfigurationsskripte und DSC-Ressourcen Module abgelegt werden sollen (*ConfigurationPath* und *ModulePath*).

Auch wenn wir bei der Kommunikation mit dem Pull Server auf Verschlüsselung verzichten, so muss sich ein Client trotzdem authentisieren. Der Authentisierungscode besteht aus einer GUID, die in einem *RegistrationKeyFile* auf dem Web Server hinterlegt ist. Jeder Client, der mit dem Pull Server kommunizieren will, muss diesen Authentisierungscode beim Verbindungsaufbau als "Ticket" vorzeigen. Eine GUID kann man sich in der PowerShell mit dem Befehl

```
New-Guid
```

erzeugen. Den angezeigten Wert hinterlegen wir in der Datendatei *APPconfig-data.psd1* und lassen ihn mit einer Instanz der *File* Ressource in die angegebene Datei schreiben.

### 2.4.3 Die Konfiguration des Local Configuration Managers (*LCMconfig*) für unseren DSC Web Pull Server DSC-PS01

Jetzt fehlt uns noch das Konfigurationsskript für den *Local Configuration Manager (LCM)* in unserer VM. Es ist in der Datei *LCMconfig.ps1* hinterlegt.

```
1  [DSCLocalConfigurationManager()]
2  configuration LCMconfig
3  {
4      Node localhost
5      {
6          Settings
7          {
8              RefreshMode = 'Push'
9              AllowModuleOverwrite = $true
10             DebugMode = 'All'
11             RebootNodeIfNeeded = $true
12             ActionAfterReboot = 'ContinueConfiguration'
13             RefreshFrequencyMins = 30
14             ConfigurationModeFrequencyMins = 60
15             ConfigurationMode = 'ApplyAndAutoCorrect'
16         }
17         PartialConfiguration OSconfig
18         {
19             Description = 'Configure OS with computername and IP address.'
20             RefreshMode = 'Push'
21         }
22         PartialConfiguration APPconfig
23         {
24             Description = 'Configure VM for additional services, e.g. DSC Pull Server.'
25             RefreshMode = 'Push'
26         }
27     }
28 }
29 LCMconfig
```

Im Ressourcenblock *Settings* legen wir die globalen Einstellungen für den LCM fest. Konfigurationen müssen im Push-Modus übergeben werden (*RefreshMode = 'Push'*), dabei ist das überschreiben vorhandener Module erlaubt (*AllowModuleOverwrite = \$true*) und wir arbeiten im Debug-Modus (*DebugMode = 'All'*). Weiterhin legen wir fest, dass ein automatischer Neustart des Systems erfolgen soll, falls dies notwendig ist - z.B. nach dem Ändern des Computernamens (*RebootNodeIfNeeded = \$true*) und dass danach die Konfiguration fortgesetzt werden soll (*ActionAfterReboot = 'ContinueConfiguration'*). Dann definieren wir noch die Zeitintervalle, nach denen der LCM jeweils aktiviert wird und dass er im Falle von Konfigurationsänderungen automatisch die ursprüngliche Konfiguration wieder herstellen soll (*ConfigurationMode = 'ApplyAndAutoCorrect'*).

Nach dem Festlegen dieser globalen Einstellungen müssen wir den LCM noch informieren über seine partiellen Konfigurationen. Dies geschieht jeweils mit einem Ressourcenblock *PartialConfiguration*. Der Name des jeweiligen Blocks muss exakt dem Namen der jeweiligen partiellen Konfiguration entsprechen.

#### 2.4.4 Das Initiieren der DSC-Konfiguration in der VM nach der Windows Installation

Weiter oben habe ich bereits erwähnt, dass wir im Betriebssystem der VM eine Batchdatei *SetupComplete.Cmd* hinterlegt haben, die vom Windows Setup automatisch als letzte Aktion aufgerufen wird. Aus dieser Batchdatei heraus starten wir die PowerShell, um ein Skript *RunDSC.ps1* auszuführen.

```
1 # RunDSC.ps1 - finish system setup with DSC
2
3 Set-Location -Path $PSScriptRoot
4
5 # 1. rename network adapters to their DisplayValues
6 #   given by Hyper-V device naming property
7 .\Rename-NetAdapterToHyperVDeviceNaming.ps1
8
9 # 2. configure LCM for the partial configurations
10 .\LCMconfig.ps1
11 Set-DscLocalConfigurationManager -path .\LCMconfig -Force
12
13 # 3. Apply partial DSC Configurations
14
15 # 3.1. define OS configuration (IP addresses and computername)
16 .\OSconfig.ps1 -ConfigurationData .\OSconfig-data.psd1 -OutputPath .\OSconfig
17 Publish-DscConfiguration -Force -Verbose -Path .\OSconfig
18
19 # 3.2. define APP configuration (e.g. DSC Pull Server)
20 .\APPconfig.ps1 -ConfigurationData .\APPconfig-data.psd1 -OutputPath .\APPconfig
21 Publish-DscConfiguration -Force -Verbose -Path .\APPconfig
22
23 # 3.3. start the configuration
24 Start-DscConfiguration -wait -Force -Verbose -UseExisting
```

Dieses Skript führt jetzt die einzelnen Schritte aus zum Initiieren der DSC-Konfiguration in der VM.

- Mit *Set-Location -Path \$PSScriptRoot* setzen wir das Arbeitsverzeichnis auf unser Skript-Verzeichnis *C:\DSC*
- Durch den Aufruf des Skripts *Rename-NetAdapterToHyperVDeviceNaming.ps1* - Sie finden es ebenfalls in der Download Library [DSC-Lab.zip](#) - werden die Namen der Netzwerkadapter innerhalb der VM (sie heißen normalerweise *Ethernet*, *Ethernet 2*, ...) auf die Namen geändert, die im Hyper-V festgelegt sind und für die die Eigenschaft *DeviceNaming* auf 'On' steht. In unserem Beispiel hat der Netzwerkadapter in der VM nun nicht mehr den Namen *Ethernet*, sondern heißt jetzt *Mgmt*.
- Nun kompilieren wir das Konfigurationsskript für den LCM und übergeben es ihm durch einen Aufruf des Cmdlets *Set-DscLocalConfigurationManager*. Der LCM wartet jetzt auf die Übergabe der partiellen Konfigurationen.
- Zunächst wird die Betriebssystemkonfiguration *OSconfig.ps1* kompiliert und dem LCM mit *Publish-DscConfiguration* bekannt gemacht.
- Analog verfahren wir mit der Anwendungskonfiguration *APPconfig.ps1*.
- Der LCM hat nun alle Angaben, um seine Arbeit aufzunehmen, d.h. wir können jetzt mit *Start-DscConfiguration* die Konfiguration starten. Durch den Parameter *-UseExisting* weisen wir ihn darauf hin, die zuvor übergebenen partiellen Konfigurationen zu verwenden.

## 2.5 Der DSC Web Pull Server DSC-PS01 entsteht

Jetzt ist endlich der Zeitpunkt gekommen, wo wir unseren DSC Web Pull Server DSC-PS01 als VM im Hyper-V erzeugen können. Nachdem alle Konfigurationsdaten in der zuvor beschriebenen Verzeichnisstruktur abgelegt sind, verwenden wir das eingangs beschriebene Konfigurationsskript *DSCHyperV-VM0.ps1* und führen in einer PowerShell (ISE) Sitzung folgende Befehle aus (bitte mit Administratorrechten starten!);

1. Wechseln Sie in das *DSC-Lab* Verzeichnis
2. Laden Sie das Konfigurationsskript per *Dot Sourcing* in den globalen Arbeitsbereich der PowerShell ISE Sitzung

```
PS D:\dsc-lab> . .\DSCHyperV_VM0.ps1
PS D:\dsc-lab>
```

3. Kompilieren Sie das Skript mit folgendem Befehl

```
DSCHyperV_VM0 -VMname DSC-TestVM
               -ParentVhdPath E:\Hyper-V\WS2016_GPT_DCGUI.vhdx
               -OutputPath .\DSC-PS01
               -Verbose

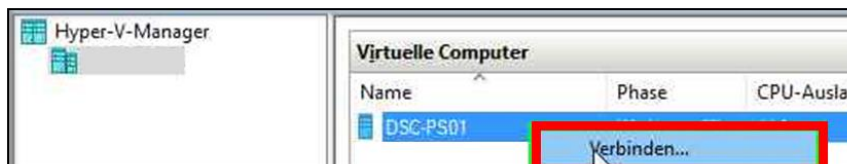
Verzeichnis: D:\dsc-lab\DSC-PS01

Mode                LastWriteTime         Length Name
----                -
-a----            05.06.2017   12:45           9990 localhost.mof
```

4. Starten Sie die Konfiguration zum Erzeugen der VM und dann die VM selbst

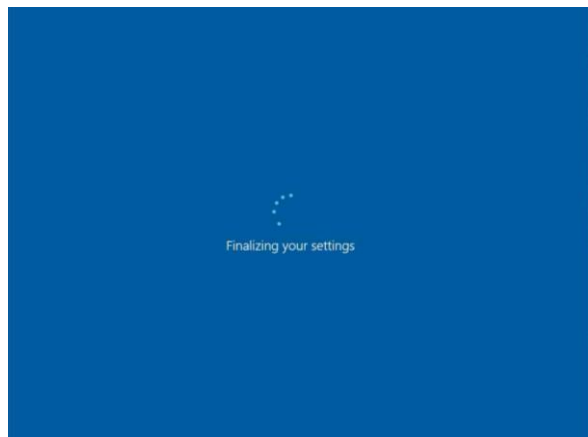
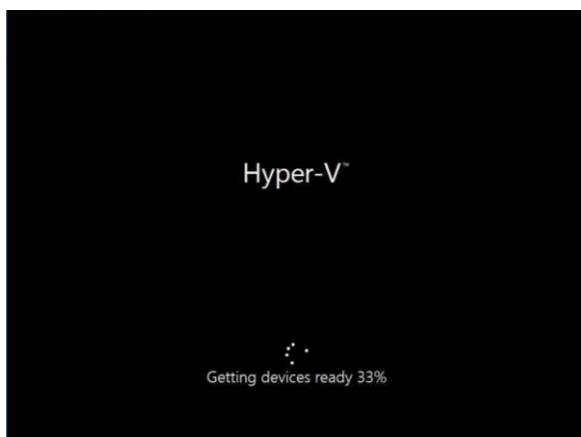
```
Start-DscConfiguration -wait -force -Verbose -Path .\DSC-PS01
Start-VM -Name DSC-PS01
```

Wir können uns jetzt mit der VM *DSC-PS01* über den Hyper-V Manager verbinden und den weiteren Ablauf mitverfolgen.



### 2.5.1 Start und Konfiguration

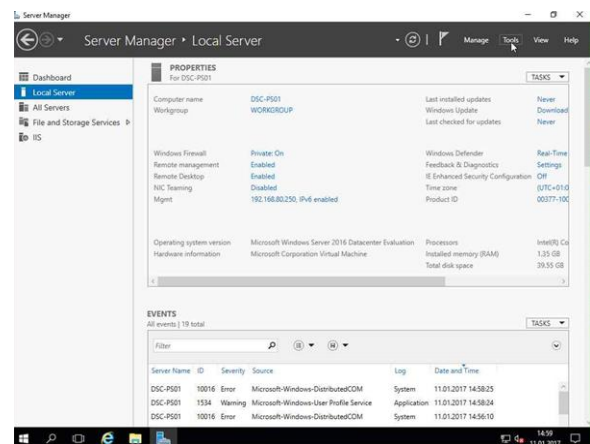
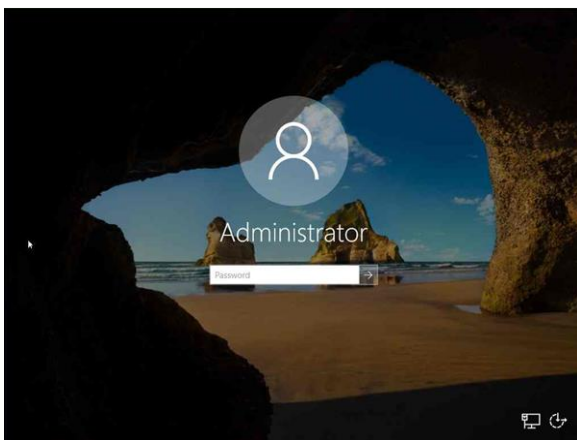
Der Windows Setup startet



Jetzt wird der LCM im Hintergrund aktiv. Es wird durch das Setzen des neuen Computernamens während der Betriebssystemkonfiguration ein Neustart durchgeführt ..



... und dann können wir uns als *Administrator* anmelden (das Kennwort haben wir ganz am Anfang in der *Unattend.xml* festgelegt!).



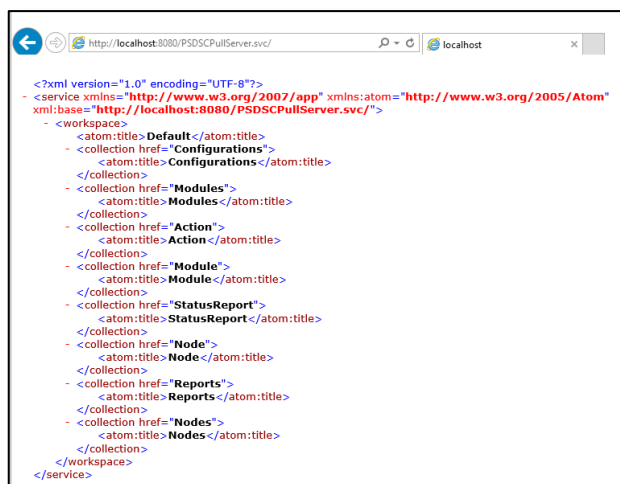
Im Server Manager können wir das System schon mal inspizieren, während im Hintergrund noch die Anwendungskonfiguration (APPconfig) läuft. Nicht verzweifeln, das kann einige Minuten dauern!

### 2.5.2 Funktionsprüfung

Starten Sie in der VM den Internet Explorer (oder einen anderen Browser) und geben Sie folgende URL in das Adressfeld ein:

<http://localhost:8080/PSDSCPullServer.svc>

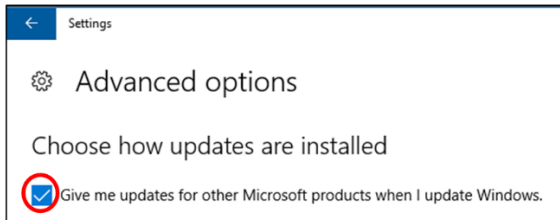
Wenn dann im Browser-Fenster XML-Code erscheint, der ähnlich aussieht wie in der nebenstehenden Abbildung, dann können Sie davon ausgehen, dass der Web Service mit dem Pull Server korrekt installiert und konfiguriert wurde. Die Bedeutung des XML-Codes soll uns hier nicht weiter interessieren.



## 2.6 Ein einfacher Pull Client

Nun haben wir also einen DSC Web Pull Server am Laufen. Aber wie kann ich ein solches System nutzen, um DSC-Konfigurationen an andere Systeme zu verteilen. Das wollen wir nun mit einem kleinen Beispiel aufzeigen: Wir werden eine VM erzeugen, die sich ihre Einstellung für den *Microsoft Update* als Pull Client von unserem Web Pull Server abholt.

Gemeint ist der Haken, den man bei Windows 10 bzw. beim Windows Server 2016 in den *Settings – Update & Security – Advanced Options* setzen kann. Standardmäßig ist er nicht gesetzt.



### 2.6.1 Konfigurationsskripte und DSC-Ressourcen erstellen und einsammeln

Als erstes müssen wir natürlich Konfigurationsskripte erstellen. Ich werde wieder mit partiellen Konfigurationen arbeiten, wie wir sie bereits für unseren Web Pull Server verwendet haben, also eine DSC-Konfiguration zum Konfigurieren des Betriebssystems (*OSconfig*) und weitere für die Anwendungs-komponenten (*APPconfig*), die auf dem Zielsystem installiert werden sollen.

Bisher haben wir in unserer DSC-Labumgebung (unter *D:\DSC-Lab*) für jedes Zielsystem ein eigenes Unterverzeichnis angelegt, in dem die jeweiligen Konfigurationen abgelegt wurden.

An der Vorgehensweise, wie wir das Betriebssystem konfigurieren, also z.B. Computernamen und IP-Konfiguration setzen, werden wir nichts ändern. *OSconfig* werden wir weiterhin für jedes System getrennt verwalten und im Push Modus übermitteln und abarbeiten lassen.

Eine Anwendungskonfiguration wie das Aktivieren des *Microsoft Updates*, die systemunabhängig für viele bzw. alle Systeme gelten soll, ist ein ideales Szenario, um sie über einen DSC Web Pull Server bereitzustellen.

Dazu werden wir die Anwendungskonfigurationen nicht mehr systemspezifisch speichern, sondern direkt im Lab-Verzeichnis *D:\DSC-Lab* erstellen. In einem eigenen neues Unterverzeichnis *PullSetup* werden wir dann alles sammeln, was auf den Pull Server bereitgestellt werden soll.

Zu allererst benötigen wir eine DSC-Ressource, mit der wir den *Microsoft Update* aktivieren können. Wir finden sie unter dem Namen [xWindowsUpdate](#) in der PowerShell Gallery. Diese Ressource installieren wir wie üblich zunächst auf unserem Entwicklungsrechner.

```
Install-Module -Name xwindowsupdate -Force
```

Nun müssen wir ein für den *Microsoft Update* passendes Konfigurationsskript erstellen und kompilieren. [xWindowsUpdate](#) enthält unter anderem die DSC-Ressource *xWindowsUpdateAgent*, mit der man die verschiedenen Einstellungen für den *Windows Update* setzen kann:

- *IsSingleInstance* muss immer mit 'Yes' definiert sein und bedeutet, dass die Ressource in einem Skript nur einmal vorkommen darf.
- Bei *Category* geben wir an, welche Arten von Updates berücksichtigt werden sollen. Es müssen nicht alle Kategorien (wie im Beispiel) angegeben werden. Will man z.B. nur die Sicherheitsupdates, genügt die Angabe von 'Security'.

- Bei *Source* legen wir fest, woher Updates geladen werden sollen. Mögliche andere Quellen wären '*Windows Update*' (Default) oder '*WSUS*'.
- Die Angabe bei *Notifications* legt fest, wie Updates behandelt werden sollen. '*ScheduledInstallation*' ist der Standardwert.
- Mit *UpdateNow* können wir festlegen, ob beim Abarbeiten der Konfiguration Updates sofort automatisch gesucht und installiert werden sollen. Mit *\$false* wird dies auf den nächsten regulären Zeitpunkt für den Windows Update zurückgestellt.

```

Configuration EnableMSUpdate
{
    Import-DscResource -Module xwindowsupdate

    Node localhost
    {
        xwindowsupdateAgent EnableMSUpdate
        {
            IsSingleInstance = 'Yes'
            Category = @( 'Security', 'Important', 'Optional' )
            Source = 'MicrosoftUpdate'
            Notifications = 'ScheduledInstallation'
            UpdateNow = $false
        }
    }
}
EnableMSUpdate

```

Diese Konfiguration müssen wir nun kompilieren durch einen Aufruf des Namens (letzte Zeile).

Durch das Kompilieren wird ein Unterverzeichnis mit dem Namen der Konfiguration erzeugt, das eine .MOF-Datei mit dem Namen des Nodes enthält, in unserem Beispiel also `.\EnableMSUpdate\localhost.mof`.

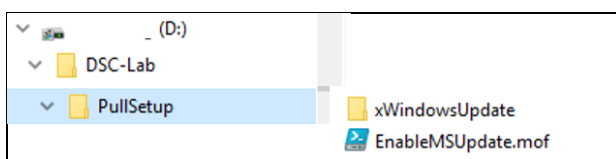
Diese .MOF-Datei (nur diese, nicht die Skriptdatei!) benötigen wir später auf dem Pull Server, d.h. wir kopieren sie in unser Sammelverzeichnis *PullSetup*. Wichtig dabei ist, der .MOF-Datei einen passenden Namen zu geben, sinnvollerweise `<Konfiguration>.mof`. Pull Clients rufen Konfigurationen über deren Namen beim Server ab. Für unser Beispiel werden wir die Datei *localhost.mof* also beim Kopieren in *EnableMSUpdate.mof* umbenennen.

```
Copy-Item .\EnableMSUpdate\localhost.mof .\PullSetup\EnableMSUpdate.mof -Force
```

Die für das Publizieren einer Konfiguration notwendigen DSC-Ressourcen müssen ebenfalls auf dem Pull Server verfügbar gemacht werden. Wir kopieren deshalb noch die für unser Beispiel benötigte DSC-Ressource *xWindowsUpdate* in das *PullSetup*-Verzeichnis.

```
Copy-Item "$env:ProgramFiles\windowsPowerShell\modules\xwindowsupdate" `
"D:\DSC-Lab\PullSetup" -Force -Recurse
```

Wir haben für unser Beispiel auf unserem Entwicklungsrechner also jetzt die folgende Verzeichnisstruktur:



Dieses Verzeichnis müssen wir jetzt auf unseren Pull Server kopieren und dort publizieren. Am einfachsten geht das in unserer Lab-Umgebung über eine VM Verbindung im *Erweiterten Modus (Enhanced Session)* vom lokalen Hyper-V Manager zum Pull Server *DSC-PS01*. Dort sehen wir sowohl unser *DSC-Lab* Verzeichnis auf unserem Hyper-V Host als auch das Laufwerk C: des Pull Servers und wir können Dateien hin- und herkopieren.

Tipp: Wenn Sie den Pull Server gemäß der Beschreibung im vorangegangenen Kapitel erzeugt haben, gibt es dort ein Verzeichnis `C:\DSC`, das noch die Daten von der Pull Server Konfiguration enthält. Kopieren Sie das *PullSetup*-Verzeichnis dorthin.

### 2.6.2 Konfigurationsskripte und DSC-Ressourcen auf dem Pull Server publizieren

Beim Installieren des Pull Servers haben wir in der DSC-Konfiguration zwei Pfade festgelegt:

- *ConfigurationPath*: In diesem Verzeichnis liegen die .MOF-Dateien der für Clients verfügbaren Konfigurationen; bei unserem *DSC-PS01* haben wir dafür festgelegt:  
`"C:\Program Files\WindowsPowerShell\DscService\Configuration"`
- *ModulePath*: Hier müssen die für die Konfigurationen notwendigen DSC-Ressourcen abgelegt werden. Für unseren *DSC-PS01* hatten wir festgelegt:  
`"C:\Program Files\WindowsPowerShell\DscService\Modules"`

Doch halt: Es genügt nicht, zum Publizieren die Skripte und Ressourcen einfach dorthin zu kopieren. Wir müssen sie vielmehr noch ein wenig aufbereiten.

#### 2.6.2.1 Aufbereiten der MOF-Konfigurationsdateien

Vor dem Kopieren einer MOF-Konfigurationsdatei in den *ConfigurationPath* müssen wir eine Prüfsummerdatei dafür erzeugen. Der *Local Configuration Manager (LCM)* kann damit die Konfiguration validieren. Zum Erzeugen einer Prüfsummendatei steht das CmdLet *New-DSCChecksum* zur Verfügung. Es erwartet als Parameter einen Pfad, in dem die MOF-Dateien liegen. Das CmdLet erzeugt für jede MOF-Datei in diesem Verzeichnis eine Prüfsummendatei mit dem Namen *ConfigurationMOFName.mof.checksum*, wobei *ConfigurationMOFName* der Name der Konfiguration ist. Die .MOF- und die zugehörige .checksum-Datei müssen nun zusammen in den *ConfigurationPath* des Pull Servers kopiert werden.

Wichtig: Wenn wir zu einem späteren Zeitpunkt eine Konfiguration ändern und damit eine neue .MOF-Datei erzeugen, müssen wir auch wieder eine neue .checksum-Datei dafür erzeugen!

#### 2.6.2.2 Aufbereiten der DSC Ressourcen Module

Jedes PowerShell Modul mit DSC-Ressourcen muss in eine .ZIP-Datei mit folgender Namenskonvention gepackt werden: *{Modulname}\_{Moduleversion}.zip*. Dummerweise kann man eine DSC-Ressource, wie man sie mit *Install-Module* aus der PowerShell Gallery heruntergeladen hat, nicht einfach in eine .ZIP-Datei packen, da mit PowerShell / WMF 5.0 durch die neu eingeführte Versionsverwaltung die interne Verzeichnisstruktur erweitert wurde. DSC-Ressourcen haben jetzt die Struktur:

```
'{Module Folder}\{Module Version}\DscResources\{DSC Resource Folder}'
```

wohingegen in der .ZIP-Datei die frühere Struktur

```
'{Module Folder}\DscResources\{DSC Resource Folder}'
```

(also ohne dem Verzeichnis *{Module Version}*) erwartet wird.

Wir müssten also temporär eine Kopie ohne das Verzeichnis *{Module Version}* erzeugen, diese dann packen und den Namen der .ZIP-Datei an die geforderte Namenskonvention anpassen.



Und wie nicht anders zu erwarten, müssen wir auch für die .ZIP-Datei eine Prüfsummendatei erzeugen und dann in das Verzeichnis *ModulePath* auf dem Pull Server kopieren.

### 2.6.2.3 Es geht einfacher...

Etwas mühsam, werden Sie sagen, insbesondere für die Aufbereitung der .ZIP-Dateien. Das müssen sich auch die Entwickler im PowerShell Team gedacht haben und liefern mit der DSC-Ressource *xPSDesiredStateConfiguration* noch gleich ein zusätzliches Modul mit dem Namen *PublishModulesAndMofsToPullServer.psm1*. Es enthält eine Funktion mit dem Namen *Publish-DSCModuleAndMof*, die das mühsame Erzeugen der Prüfsummendateien und das Packen der DSC-Ressourcen in .ZIP-Dateien auf einen Funktionsaufruf reduziert und auch gleich die Ergebnisse in den entsprechenden Pfaden auf dem Pull Server publiziert.

Für unser Beispiel zum Aktivieren des *Microsoft Updates* verbinden Sie sich jetzt mit unserem Pull Server *DSC-PS01* über eine VM-Verbindung aus dem Hyper-V Manager, starten dort eine PowerShell-Sitzung mit Administratorrechten, wechseln dort in das Verzeichnis *C:\DSC\PullSetup* und geben die folgenden Befehle ein:

```
Import-Module xPSDesiredStateConfiguration
Publish-DSCModuleAndMof -Source . -Force -Verbose
```

- Beim Parameter *-Source* geben Sie das Verzeichnis an, in dem sich die aufzubereitenden Konfigurationen und DSC-Ressourcen befinden (hier das aktuelle Verzeichnis).
- Durch die Angabe von *-Force* erzwingen Sie das Überschreiben bereits vorhandener älterer Versionen.
- Und mit *-Verbose* können Sie ein Protokoll im Konsolenfenster mitlaufen lassen.

Und als Ergebnis finden wir die benötigten Dateien im *ModulePath* bzw. *ConfigurationPath* auf dem Pull Server.

```
PS C:\DSC\PullSetup> dir

Directory: C:\DSC\PullSetup

Mode                LastWriteTime         Length Name
----                -
d-----          14.02.2017   18:56             xWindowsUpdate
-a-----          14.02.2017   18:54          2250 EnableMSUpdate.mof

PS C:\DSC\PullSetup> Publish-DSCModuleAndMof -Source . -Force -Verbose

PS C:\DSC\PullSetup> dir "$env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration"

Directory: C:\Program Files\WindowsPowerShell\DscService\Configuration

Mode                LastWriteTime         Length Name
----                -
-a-----                2250 EnableMSUpdate.mof
-a-----                64  EnableMSUpdate.mof.checksum

PS C:\DSC\PullSetup> dir "$env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"

Directory: C:\Program Files\WindowsPowerShell\DscService\Modules

Mode                LastWriteTime         Length Name
----                -
-a-----          20794 xWindowsUpdate_2.5.0.0.zip
-a-----                64  xWindowsUpdate_2.5.0.0.zip.checksum

PS C:\DSC\PullSetup> |
```

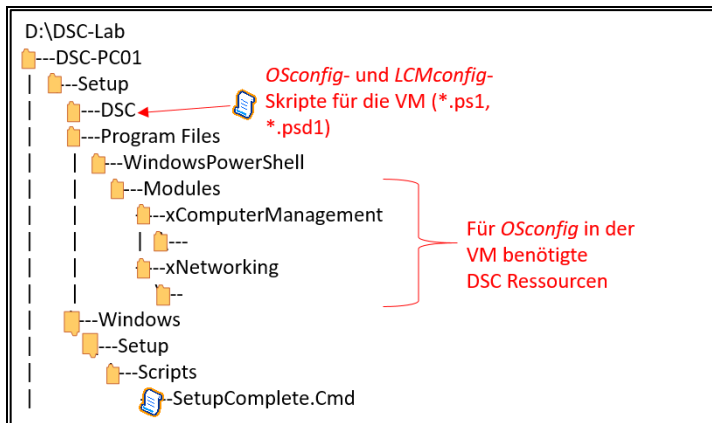
ConfigurationPath des Pull Servers

ModulePath des Pull Servers

Damit stehen die Konfigurationen und Ressourcen auf dem Pull Server zum Abrufen durch Pull Clients bereit.

### 2.6.3 Erstellen des Pull Clients

Wenn wir jetzt die Funktionalität unseres DSC Pull Servers testen wollen, benötigen wir einen DSC Pull Client. Ich erstelle mir dafür eine VM *DSC-PC01* mit Windows Server 2016 als Betriebssystem nach der im vorangegangenen Kapitel beschriebenen Methode. Ein entsprechendes PowerShell Skript *New-DSC-PC01\_SetupFiles.ps1* zum Erzeugen der Verzeichnisstruktur und der Dateien finden Sie in der zum Download beigefügten .ZIP-Datei.



Im *Setup*-Verzeichnis dieser VM finden Sie nur noch Skripte zum Konfigurieren des Betriebssystems (*OSconfig*), des Local Configuration Managers (*LCMconfig*) sowie zum Initiieren der DSC-Konfiguration (*SetupComplete.Cmd* und *RunDSC.ps1*).

Um nun weitere (Anwendungs-) Konfigurationen von unserem Pull Server abzurufen, müssen wir dem Local Configuration Manager (LCM) des Pull Clients entsprechende Informationen mitgeben. Das *LCMconfig*-Skript sieht dann folgendermaßen aus:

```

[DSCLocal]ConfigurationManager()
configuration LCMconfig
{
    Node localhost
    {
        settings
        {
            RefreshMode = 'Push'
            AllowModuleOverwrite = $true
            DebugMode = 'All'
            RebootNodeIfNeeded = $true
            ActionAfterReboot = 'ContinueConfiguration'
            RefreshFrequencyMins = 30
            ConfigurationModeFrequencyMins = 60
            ConfigurationMode = 'ApplyAndAutoCorrect'
        }
        ConfigurationRepositoryWeb DSC-PS01
        {
            ServerURL = 'http://DSC-PS01:8080/PSDSCPullServer.svc'
            RegistrationKey = '14fc8e72-5036-4e79-9f89-5382160053aa'
            AllowUnsecureConnection = $true
            ConfigurationNames = @('EnableMSUpdate')
        }
        ReportServerWeb DSC-PS01
        {
            ServerURL = 'http://DSC-PS01:8080/PSDSCPullServer.svc'
            AllowUnsecureConnection = $true
        }

        PartialConfiguration OSconfig
        {
            Description = 'Configure OS with computername and IP address.'
            RefreshMode = 'Push'
        }
        PartialConfiguration EnableMSUpdate
        {
            Description = 'Configure VM for NS Update.'
            RefreshMode = 'Pull'
            ConfigurationSource = @("[ConfigurationRepositoryWeb]DSC-PS01")
            Dependson = '[PartialConfiguration]OSconfig'
        }
    }
}
LCMconfig

```

- Im *Settings* Block legen wir allgemeine Einstellungen für den LCM fest, wie z.B. die Zeitintervalle, nach denen der LCM immer aktiv wird, um die Konfigurationen zu prüfen und gegebenenfalls automatisch zu korrigieren. Sie werden etwas verwundert sein über die Einstellung *RefreshMode = 'Push'*, wo wir doch über einen Pull Client sprechen. Diese Einstellung ist für unser Beispiel ohne Bedeutung, da wir im Folgenden mit partiellen Konfigurationen arbeiten; und bei jeder partiellen Konfiguration müssen wir angeben, wie sie der LCM erhält (*Push* oder *Pull*).
- im nächsten Block *ConfigurationRepositoryWeb* teilen wir dem LCM Informationen für den zu verwendenden Pull Server mit:
  - *ServerURL*: Dies ist die Endpunkt-Adresse einschließlich IP-Port des Pull Server Webdienstes, die wir beim Installieren des Web Pull Servers festgelegt haben (siehe "[Die Anwendungskonfiguration \(APPconfig\) für unseren Web Pull Server DSC-PS01](#)" weiter oben)
  - *RegistrationKey*: Beim Installieren des Pull Servers haben wir eine GUID festgelegt, mit der sich Clients authentisieren müssen. Diese muss hier wieder angegeben werden.
  - *AllowUnSecureConnection*: Normalerweise erfolgt die Kommunikation mit dem Pull Server über verschlüsselte Verbindungen. Wir haben bei unserem Pull Server der Einfachheit halber darauf verzichtet und müssen dies hier jetzt dem LCM mitteilen
  - Und dann müssen wir dem LCM über die Angabe *ConfigurationNames* noch mitteilen, welche der anschließend beschriebenen partiellen Konfigurationen er auf diesem Pull Server findet. Die Angabe ist ein String Array mit den Namen der abzurufenden Konfigurationen - in unserem Fall gibt es nur eine Konfiguration mit dem Namen *EnableMSUpdate*.

Falls wir Konfigurationen von verschiedenen Pull Servern abrufen wollen, können wir weitere *ConfigurationRepositoryWeb*-Blöcke hier definieren - natürlich jeweils mit unterschiedlichen Blocknamen.

- Mit dem nächsten Block *ReportServerWeb* können wir dem LCM mitteilen, wo er Status-Informationen zu durchgeführten Aktionen ablegen soll. Dies ist eine Zusatz-Funktionalität eines Web Pull Servers, auf die ich später noch zurückkommen werde.
- Jetzt folgen noch die partiellen Konfigurationen mit der Angabe, ob sie der LCM lokal vorfindet (*RefreshMode='Push'*) oder von einem Pull Server abrufen soll (*RefreshMode = 'Pull'*). Für Konfigurationen im Pull Modus ist dann noch ein Verweis auf den *ConfigurationRepositoryWeb* Block des zu verwendenden Pull Servers notwendig.
- Und damit die partielle Konfiguration für das Setzen des *Microsoft Updates* erst abgearbeitet wird, wenn das Betriebssystem erfolgreich konfiguriert wurde, definieren wir noch mit *DependsOn* diese Abhängigkeit von der partiellen Konfiguration *OSconfig*.

Jetzt haben wir alle Teile für unseren Pull Client *DSC-PC01* zusammen und können die VM unter Zuhilfenahme der schon weiter oben vorgestellten DSC-Konfiguration *DSCHyperV\_VM0* erstellen:

1. Wechseln Sie in das DSC-Lab Verzeichnis
2. Laden Sie das Konfigurationsskript per Dot Sourcing in den globalen Arbeitsbereich der PowerShell ISE Sitzung

```
. .\DSCHyperV_VM0.ps1
```

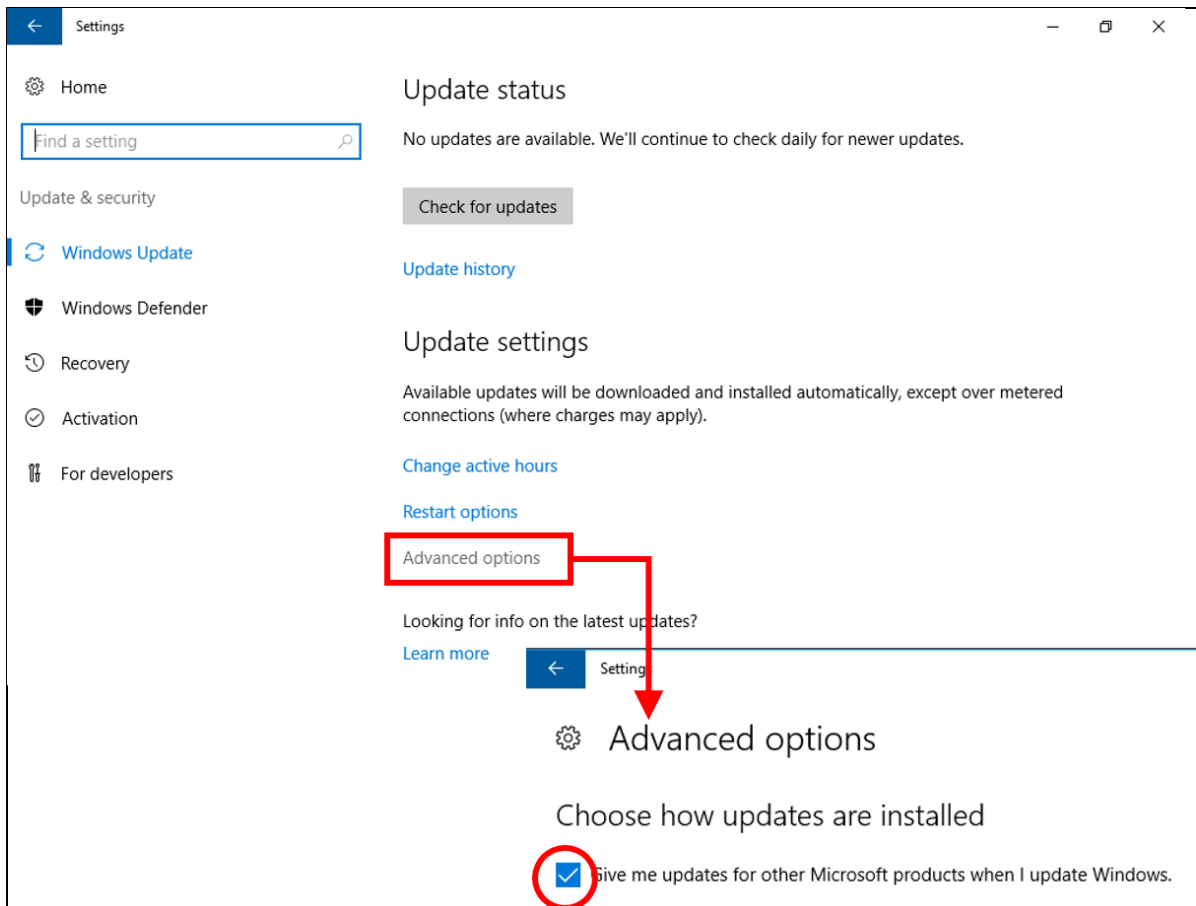
3. Kompilieren Sie das Skript mit folgendem Befehl

```
DSCHyperV_VM0 -VMname DSC-PC01 `
  -ParentVhdPath E:\Hyper-V\WS2016_GPT_DCGUI.vhdx `
  -OutputPath .\DSC-PC01 -Verbose
```

4. Starten Sie die Konfiguration der VM

```
Start-DscConfiguration -wait -force -verbose -Path .\DSC-PC01
```

Nachdem die VM gestartet wurde, können Sie über eine VM Verbindung vom lokalen Hyper-V Manager zum Pull Client *DSC-PC01* den Erstellungsvorgang mit verfolgen - wobei allerdings außer einigen automatischen Neustarts nichts Spektakuläres passiert. Melden Sie sich dann als lokaler Administrator an und rufen in den Settings die Kategorie *Update & Security* auf. Klicken Sie dort auf *Advanced Options*. Im nächsten Fenster sollte jetzt die Option *Give me updates for other Microsoft products when I update Windows* aktiviert sein.



Um zu überprüfen, ob der LCM auch regelmäßig seinen Dienst versieht, können Sie die Option für den *Microsoft Update* deaktivieren. Schließen Sie das Fenster mit den *Advanced Options* bzw. gehen zurück auf die Startseite von *Update & Security* und warten ... die Zeit ist abhängig von den Angaben im *Settings*-Block des LCM (in unserem Beispiel bis zu 60 Minuten).

Wenn Sie nicht so lange warten wollen, können Sie auf dem Pull Client auch einen Refresh erzwingen durch Eingabe des Befehls

```
Start-DscConfiguration -Wait -Force -Verbose -UseExisting
```

Wenn Sie danach die *Advanced Options* erneut aufrufen, werden Sie feststellen, dass die Option für den *Microsoft Update* wieder aktiviert ist.

## 2.7 Der Pull Server als Report Server

Ein DSC Web Pull Server stellt nicht nur DSC-Konfigurationen bereit zur Nutzung durch Pull Clients, er kann auch als Report Server für Pull Clients arbeiten. Diese Funktionalität ist immer aktiviert, es sind also keine besonderen Konfigurationen am Pull Server notwendig.

### 2.7.1 Senden von Report Daten

Ob ein Pull Client Status Meldungen an einen Report Server senden soll und an welchen, kann man in der LCM-Konfiguration durch den Ressourcen-Block *ReportServerWeb* festlegen.

```
ReportServerWeb DSCC-PS01
{
  ServerURL = 'http://DSC-PS01:8080/PSDSCPullServer.svc'
  AllowUnsecureConnection = $true
}
```

Darin ist eigentlich nur eine Angabe notwendig, nämlich die URL des Pull Servers, an den Statusdaten gesendet werden soll. Das muss nicht der gleiche Pull Server sein, von dem auch die Konfigurationen geholt werden (wie in unserem Beispiel).

Für den Fall, dass die Statusdaten an einen anderen Server gesendet werden sollen, muss außerdem noch dessen *RegistrationKey* hinterlegt werden. Bei unserem Beispiel können wir darauf verzichten, da hier Pull und Report Server identisch sind. Der LCM verwendet dann den *RegistrationKey* aus dem *ConfigurationRepositoryWeb*-Block.

In unserer Testumgebung haben wir beim Pull Server auch festgelegt, dass ohne Verschlüsselung gearbeitet werden soll. Deshalb müssen wir bei der Definition des Report Servers dies dem LCM durch die Angabe von *AllowUnsecureConnection = \$true* mitteilen

### 2.7.2 Auswerten der Report Daten

Die Statusdaten der Pull Clients werden in einer internen Datenbank auf dem Web Pull Server verwaltet. Sie können durch Funktionsaufrufe (*Web Requests*) an den Webservice abgerufen werden. Bei diesen Aufrufen muss die *AgentID* des Local Configuration Managers auf dem gewünschten Pull Client angegeben werden. Diese *AgentID* versteckt sich in den Eigenschaften des LCM auf dem jeweiligen Client und kann dort mit dem CmdLet *Get-DscLocalConfigurationManager* (Alias: *glcm*) ermittelt werden:

```
PS C:\dsc> Get-DscLocalConfigurationManager
ActionAfterReboot      : ContinueConfiguration
AgentId                : 238BD835-F471-11E6-8F7F-00155D011E48
AllowModuleOverwrite  : true
CertificateID          :
ConfigurationDownloadManagers : {[ConfigurationRepositoryWeb]DSC-PS01}
ConfigurationID        :
ConfigurationMode      : ApplyAndAutoCorrect
ConfigurationModeFrequencyMins : 60
Credential             :
DebugMode              : {All}
DownloadManagerCustomData :
DownloadManagerName    :
LCMCompatibleVersions  : {1.0, 2.0}
LCMState               : Idle
LCMStateDetail         :
LCMVersion             : 2.0
StatusRetentionTimeInDays : 10
SignatureValidationPolicy : NONE
SignatureValidations   : {}
MaximumDownloadSizeMB : 500
PartialConfigurations  : {[PartialConfiguration]OSConfig,
                           [PartialConfiguration]EnableMSUpdate}
RebootNodeIfNeeded     : True
RefreshFrequencyMins   : 30
RefreshMode            : Push
ReportManagers         : {[ReportServerWeb]DSCC-PS01}
ResourceModuleManagers : {}
PSComputerName         :
```

... oder direkt:

```
PS C:\dsc> (glcm).AgentId
238BD835-F471-11E6-8F7F-00155D011E48
```

Mit der nachstehenden PowerShell Funktion *GetReport* (Quelle: [MSDN](#)) kann man nun die Statusdaten eines Pull Clients abrufen:

```
function GetReport
{
    param (
        $AgentId = "$((glcm).AgentId)",
        $serviceURL = "http://DSC-PS01:8080/PSDSCPullServer.svc"
    )
    $requestUri = "$serviceURL/Nodes(AgentId= '$AgentId')/Reports"
    $request = Invoke-WebRequest -Uri $requestUri `
        -ContentType "application/json;odata=minimalmetadata;streaming=true;charset=utf-8" `
        -UseBasicParsing -Headers @{Accept = "application/json"; ProtocolVersion = "2.0"} `
        -ErrorAction SilentlyContinue -ErrorVariable ev
    $object = ConvertFrom-Json $request.content
    return $object.value
}
```

Die Funktion liefert ein Array von JSON-Objekten zurück, die man mit einigen weiteren Befehlen aufbereiten kann. Das folgende Skript zeigt, wie man z.B. von unserem Hyper-V Host aus die *AgentID* des Pull Clients *DSC-PC01* ermitteln kann, dann damit die Statusdaten vom Pull Server abrufen, sie nach dem Startdatum sortiert und dann das Ergebnis des letzten LCM-Laufs anzeigen kann.

```
function GetReport
{...}

#region - Ermitteln der AgentID des Pull Clients DSC-PC01

$VMname = "DSC-PC01"
# Zum Ermitteln der AgentID verbinden wir uns per Powershell direkt mit dem Pull Client
# ==> Dazu müssen wir uns dort als lokaler Administrator anmelden
$cred = Get-Credential -Message "Anmelden am Pull Client $VMname" -UserName "Administrator"
$AgentID = Invoke-Command -VMName DSC-PC01 -Credential $cred -ScriptBlock { (gilm).AgentId }

# Jetzt die Reüprtdaten abrufen
$reports = GetReport -AgentId $AgentID
# ... absteigend nach Datum sortieren
$reportsByStartTime = $reports | Sort-Object {$_. "StartTime" -as [DateTime] } -Descending
# ... die neuesten Daten herauspicken
$reportMostRecent = $reportsByStartTime[0]
# ... aus dem JSON-Format konvertieren
$statusData = $reportMostRecent.StatusData | ConvertFrom-Json
# ... und anzeigen
$statusData.ResourcesInDesiredState
$statusData.ResourcesNotInDesiredState # diese Liste sollte leer sein!

#endregion
```

Hinweis: Das vollständige Skript finden Sie in der zu diesem White Paper gehörenden .ZIP-Datei.

Wenn wir dieses Skript ausführen, erhalten wir etwa die folgende Anzeige:

```
PS D:\DSC-Lab> D:\DSC-Lab\Get-DSCReport.ps1


SourceInfo      : C:\DSC\OSconfig.ps1::8::9::xIPAddress
ModuleName      : xNetworking
DurationInSeconds : 1,906
InstanceName    : SetIP
StartDate       : 2017-02-17T17:22:51.2300000+01:00
ResourceName    : xIPAddress
ModuleVersion   : 3.1.0.0
RebootRequested : False
ResourceId      : [xIPAddress]SetIP
ConfigurationName : OSconfig
IndesiredState  : True

SourceInfo      : C:\DSC\OSconfig.ps1::14::9::xDefaultGatewayAddress
ModuleName      : xNetworking
DurationInSeconds : 0,094
InstanceName    : SetDefaultGateway
StartDate       : 2017-02-17T17:22:53.1360000+01:00
ResourceName    : xDefaultGatewayAddress
ModuleVersion   : 3.1.0.0
RebootRequested : False
ResourceId      : [xDefaultGatewayAddress]SetDefaultGateway
ConfigurationName : OSconfig
IndesiredState  : True

SourceInfo      : C:\DSC\OSconfig.ps1::19::9::xDNSServerAddress
ModuleName      : xNetworking
DurationInSeconds : 0,578
InstanceName    : SetDNS
StartDate       : 2017-02-17T17:22:53.2300000+01:00
ResourceName    : xDNSServerAddress
ModuleVersion   : 3.1.0.0
RebootRequested : False
ResourceId      : [xDNSServerAddress]SetDNS
ConfigurationName : OSconfig
IndesiredState  : True

SourceInfo      : C:\DSC\OSconfig.ps1::24::8::xComputer
ModuleName      : xComputerManagement
DurationInSeconds : 0,015
InstanceName    : SetName
StartDate       : 2017-02-17T17:22:53.8080000+01:00
ResourceName    : xComputer
ModuleVersion   : 1.9.0.0
RebootRequested : False
ResourceId      : [xComputer]SetName
ConfigurationName : OSconfig
IndesiredState  : True

SourceInfo      : D:\DSC-Lab\MSUpdate.ps1::10::9::xWindowsUpdateAgent
ModuleName      : xWindowsUpdate
DurationInSeconds : 10,902
InstanceName    : EnableMSUpdate
StartDate       : 2017-02-17T17:22:53.8240000+01:00
ResourceName    : xWindowsUpdateAgent
ModuleVersion   : 2.5.0.0
RebootRequested : False
ResourceId      : [xWindowsUpdateAgent]EnableMSUpdate
ConfigurationName : EnableMSUpdate
IndesiredState  : True
```



Damit können wir uns schnell einen Überblick über die letzten DSC-Aktionen auf einem Pull Client verschaffen. Bei Problemen ist dies natürlich nicht unbedingt ausreichend. Dann muss man auf weitergehende Methoden zurückgreifen, wie ich sie im nächsten Abschnitt "[Troubleshooting](#)" beschreiben werde.



## 2.8 Troubleshooting

Wenn Sie den Beschreibungen in diesem White Paper folgen und dabei auch gleich die Skripte aus der zugehörigen Download Library [DSC-Lab.zip](#) verwenden, sollten Sie eigentlich ohne Probleme die vorgestellten Szenarien nachvollziehen können. Doch was tun, wenn's mal nicht klappt?

Ich will Ihnen hier ein paar Tipps geben, wie Sie Probleme aufdecken können.

Solange wir interaktiv mit DSC Konfigurationsskripten arbeiten und dabei den Parameter *-Verbose* angeben, sehen wir im PowerShell Fenster immer ein ausführliches Protokoll über die von den DSC Ressourcen ausgeführten Aktionen. Bei einem dabei auftretenden Problem werden entsprechende PowerShell Fehlermeldungen mit protokolliert. So kann man in den Konfigurationen schon mal mit der Fehlersuche bei den Ressourcen Blöcken beginnen, bei denen der Fehler auftritt.

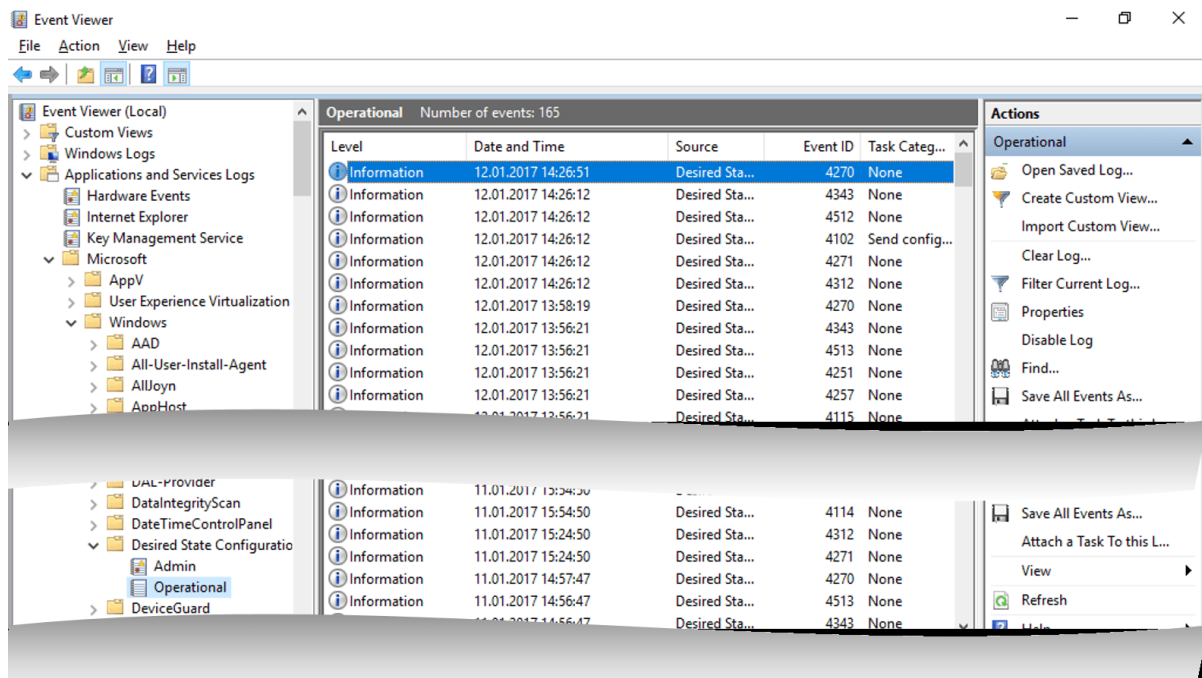
Spannender wird es, wenn man bei einem Szenario feststellt, dass eine DSC Konfiguration in einer erzeugten VM nicht so abläuft, wie man sich das vorgestellt hat. In diesem Fall sieht man interaktiv zunächst nichts, man merkt nur, dass die VM nicht richtig arbeitet.

In einem solchen Fall muss man in der VM selbst die Problemstelle suchen. Dazu gibt es folgende Hilfsmittel:

- Die Ereignisanzeige (Event Viewer) in der VM
- Protokolldateien in der VM

### 2.8.1 Die Ereignisanzeige (Event Viewer) in der VM

Der *Local Configuration Manager (LCM)* führt einen eigenen Kanal im Windows Ereignisprotokoll. In dem alle Aktionen aufgeführt sind. In der Windows Ereignisanzeige (Event Viewer) findet man dieses Protokoll in der Kategorie *Operational* unter *Application and Services Logs -> Microsoft -> Windows -> Desired State Configuration*. Es ist zwar mitunter etwas mühsam, in der Fülle der protokollierten Ereignisse geeignete Hinweise zu finden, aber im Fehlerfall sollte man hier einen Blick hineinwerfen.



### 2.8.2 Protokolldateien

Neben den Einträgen in der Ereignisanzeige erstellt der *LCM* auch ausführliche Protokolldateien, deren Inhalt den Ausgaben beim interaktiven Arbeiten entsprechen. Die Dateien werden im Verzeichnis

`C:\Windows\System32\Configuration\ConfigurationStatus` angelegt mit der Namenerweiterung `.JSON`. Leider bestehen die Dateinamen aus GUIDs, so dass eine Suche damit relativ sinnlos ist. Einfacher wird es, wenn man sie nach Datum / Uhrzeit sortiert.

Die `.JSON`-Dateien enthalten Text, der mit einem Editor wie Notepad angeschaut werden kann.

### 2.8.3 Praxistipp für unser Vorgehen

Zum Testen der in diesem White Paper vorgestellten Skripte hat sich folgendes Vorgehen bewährt: In der problembehafteten VM starte ich eine PowerShell ISE Sitzung mit Administratorrechten und lade das Skript `RunDSC.ps1`, das ja alle für die VM vorgesehenen DSC Aktionen sequentiell aufruft. Nun kann man mit den Debug-Funktionen der PowerShell ISE die Problemstelle bei der DSC Konfiguration ermitteln.

Wichtig: Korrigieren Sie Fehler nicht nur in der VM, sondern vor allem auch im `Setup`-Verzeichnis der VM bzw. im Skript, mit dem Sie die `Setup`-Verzeichnisstruktur auf Ihrem Test- und Entwicklungsrechner erzeugen (im Beispiel hier `New-DSC-PS01_SetupFiles.ps1`), damit bei einem erneuten Versuch, die VM zu generieren, Korrekturen dann auch in die neue VM übernommen werden.

## 3 Wie geht's weiter?

In diesem Dokument habe ich versucht, die Mythen um DSC etwas zu entschleiern und wie man diese Technologie nutzen kann. Nun sind Sie als Anwender bzw. Administrator gefordert, weiterführende Szenarien mit DSC zu entwerfen und umzusetzen. Ich werde natürlich auch am Ball bleiben und weiter über interessante DSC-Projekte berichten.

## Anhang A: Die Skripte zum Downloaden













In diesem Dokument finden Sie viele Code Schnipsel aus PowerShell Skripten in Form von Bildschirmfotos. Um Ihnen lästige Tipparbeit beim Nachvollziehen der Szenarien zu ersparen, habe ich die Skripte in eine Datei mit dem Namen *DSC-Lab.zip* gepackt, die Sie von der URL

<https://www.hyper-v-server.de/wp-content/uploads/2017/02/DSC-Lab.zip>

herunterladen können. Details zum Inhalt dieser .ZIP-Datei und deren Handhabung finden Sie in diesem Anhang.

Bitte beachten Sie, dass diese Skripte auf meine weiter oben beschriebene [Test- und Entwicklungs-umgebung](#) ausgerichtet sind. Falls Sie eine andere Systemumgebung verwenden, müssen Sie gegebenenfalls an der einen oder anderen Stelle Laufwerks- und / oder Pfadangaben anpassen.

Inhalt der .ZIP-Datei:

 Create-DSC-PC01.ps1	PS1-Datei
 Create-DSC-PS01.ps1	PS1-Datei
 Download-DscResources.ps1	PS1-Datei
 DSCHyperV_VM0.ps1	PS1-Datei
 Enable-MSUpdate.ps1	PS1-Datei
 Get-DSCReport.ps1	PS1-Datei
 New-DSC-PC01_SetupFiles.ps1	PS1-Datei
 New-DSC-PS01_SetupFiles.ps1	PS1-Datei
 New-NatSwitch80.ps1	PS1-Datei
 New-WS2016Image.ps1	PS1-Datei
 Readme.rtf	Rich-Text-Format
 unattend.xml	XML-Dokument

Gehen Sie nach dem Herunterladen der .ZIP-Datei folgendermaßen vor:

- Inhalt der .ZIP-Datei entpacken nach *D:\DSC-Lab*
- NAT-Switch erzeugen mit *New-NatSwitch80.ps1*
- Windows Server 2016 Basisimage aus der von Microsoft heruntergeladenen ISO erzeugen mit *New-WS2016Image.ps1*; Achtung: Zuvor Administrator Passwort in der *Unattend.xml* anpassen!
- Download der DSC-Ressourcen mit *Download-DscResources.ps1*
- Setup-Dateien für den Pull Server *DSC-PS01* erzeugen mit *New-DSC-PS01\_SetupFiles.ps1*
- VM *DSC-PS01* erzeugen und starten mit *Create-DSC-PS01.ps1*
- Setup-Dateien für den Pull Client *DSC-PC01* erzeugen mit *New-DSC-PC01\_SetupFiles.ps1*
- Pull Konfiguration für MS-Update erzeugen und im Unterverzeichnis *PullSetup* bereitstellen mit *Enable-MSUpdate.ps1*
- Unterverzeichnis *PullSetup* auf den Pull Server *DSC-PS01* ins Verzeichnis *C:\DSC* transferieren
- Publizieren der Konfiguration im *PullSetup* Verzeichnis mit *Publish-DSCModuleAndMof*
- VM *DSC-PC01* erzeugen und starten mit *Create-DSC-PC01.ps1*
- Jetzt kann man vom Hyper-V Host aus DSC-Statusdaten des Pull Clients mit *Get-DSCReport.ps1* abrufen.