Introductie Powershell

Januari 2013

Introduction Powershell

Krachtige interactie met .NET ,COM en WMI-Objecten.

Inleiding

Het doel van deze hand-out is om jullie een beknopte samenvatting te geven van de behandelde items. Het is bij Powershell belangrijk om te beginnen bij de basis. Als je de basis items begrijpt en weet toe te passen, zal je ontwikkeling in PowerShell en het begrijpen van de “command-line” en scripts gemakkelijker vergaan.

Ondanks het feit dat sommige misschien al ervaring hebben met Powershell kan het naar mijn mening geen kwaad om nog eens te beginnen bij de basis.

In deze hand-out zijn wat extra onderdelen toegevoegd. Objecten komen in avond 2 aanbod.”Functions” en “Profiles” worden niet behandeld in deze cursus maar in de praktijk zal je het zeker tegen gaan komen.

Wat is Powershell?

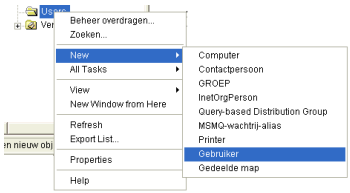
Powershell is een nieuwe interactieve shell en scripttaal. Het maakt gebruik van een complete nieuwe architectuur. Het is gebouwd op het .net framework en heeft een krachtige interactie met .NET, COM en WMI objecten.
Het emuleert vele legacy commando’s uit cmd.exe maar ook commando’s uit Linux en Unix shells.
PowerShell is een revolutionaire shell. Het ziet er in eerste instantie uit zoals iedere andere shell maar onder de oppervlakte is powershell zeer krachtig. De rede dat Powershell zo revolutionair is, is omdat het niet een tekst-based shell maar object-based shell is.
Je zult zien dat je direct interactie zult hebt met objecten, waardoor je geen complexe dingen hoeft te bouwen om taken te kunnen uitvoeren.
Op dit moment zegt dat misschien nog weinig, maar als we verder in de cursus zijn, zal dat op zijn plek vallen.

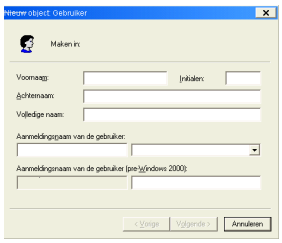
Old versus New

De GUI is uitermate krachtig. Zonder dat je kennis van alle functionaliteiten hebt, kun je zelf ontdekken wat allemaal mogelijk is. De wizards etc. helpen je flink op weg.

Als we bijvoorbeeld kijken naar de Active Directory dan is het aanmaken van een gebruiker niet zo ingewikkeld.

In de huidige vorm van werken is de GUI niet weg te denken maar Microsoft weet ook dat we dingen willen automatiseren.





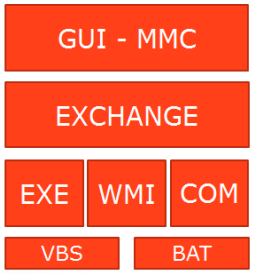
Je krijgt een wizard, deze vul je in, je drukt op OK en de gebruiker is aangemaakt. Je kunt de gebruiker daarna openen en eventueel de ontbrekende gegevens aanvullen. Stel dat je dit moet doen voor 10 gebruikers, voor 100, of misschien wel voor een complete migratie? Buiten het feit dat dat niet echt uitdagend werk is en je er RSI klachten aan zult overhouden is het erg tijd rovend en is de kans op fouten groot. Deze fouten kunnen misschien zelfs security issues met zich meebrengen.

In de huidige vorm van werken is de GUI niet weg te denken, maar Microsoft weet ook dat we dingen willen automatiseren. Hiervoor ontwikkelde Microsoft voor ons exe files , WMI en/of Com objecten.
In het bovenstaande voorbeeld heeft Microsoft voor ons bvijvoorbeeld LDIF.exe ontwikkeld. Hiermee kun je in bulk gebruikers aanmaken. Daarnaast kunnen we ook gebruik maken van LDAP en COM objecten zoals ADSI om d.m.v. VBScripts dit proces te automatiseren. Dit noemen we de "*Old way*".

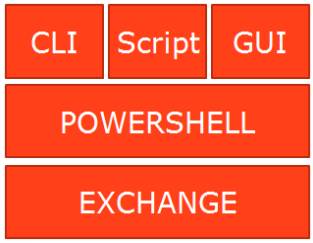
Powershell, een nieuwe manier van beheren.

De "*new way*" is de Powershell manier. Alle functionaliteiten zitten nu in powershell (i.p.v. in de MMC) en daarboven op wordt de GUI gemaakt. Dit houdt in dat we nu alles kunnen automatiseren. De GUI zal minder uitgebreid zijn en voornamelijk de basis elementen bevatten
Hieronder schematisch " the Old Way" versus " the New Way"

Old way



New Way

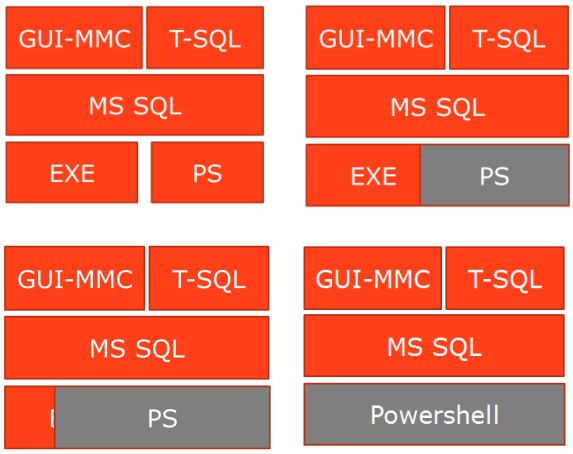


Fase 1 & 2 producten

Op dit moment onderscheiden we fase 1 en fase 2 powershell producten.

Fase 1 producten

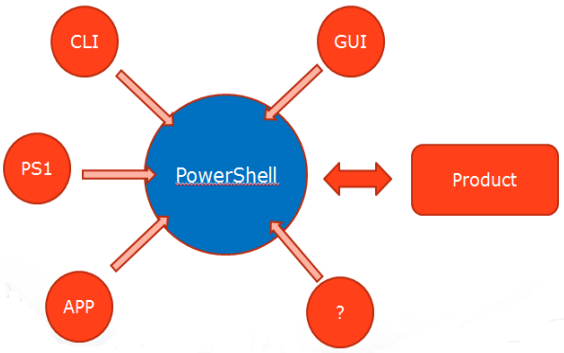
Fase 1 producten beschikken gedeeltelijk over Powershell.
Bv MS SQL 2008. MS SQL 2008 heeft een MMC gui die een gedeelte van de functionaliteit bedekt. Daarnaast natuurlijk de eigen scripttaal van SQL voor het maken van query's.
Ook zijn er exe files beschikbaar gemaakt door Microsoft en is er een stuk Powershell ingebouwd. Alle onderdelen bedienen maar een stuk van MS SQL.
Je zal echter zien dat bij iedere update de invloed van Powershell steeds groter wordt en de uiteindelijk volledige functionaliteit zal hebben. (Zie afbeelding)



Fase 2 Producten

In Fase 2 staat Powershell centraal. Powershell communiceert met het product en het product communiceert met Powershell.

Powershell kan vervolgens aangesproken worden vanuit een GUI, de "command-line" interface, vanuit een ps1 script, vanuit een applicaties en natuurlijk wat de toekomst gaat brengen.

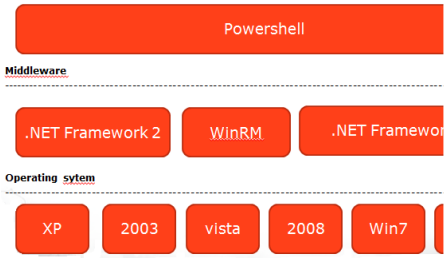


Basis Concepten van Powershell

PowerShell heeft nieuwe manier hoe je dingen kunt doen. Hieronder de basis tools en concepten.

System requirements
Powershell is beschikbaar voor de operating systems vanaf windows xp. Het is te downloaden voor de "oudere" operating systems maar vanaf windows 7 is powershell default geïnstalleerd
De minimum requirement is .Net framework 2 en heeft de meeste functionaliteiten maar niet alle features . .NET framework 3.5 en de WINRM service zullen moeten worden geïnstalleerd voor de volledige powershell en remoting functionaliteiten .

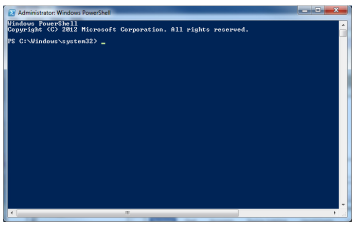
Powershell staat centraal. Powershell communiceert met het product en het product communiceert met Powershell .

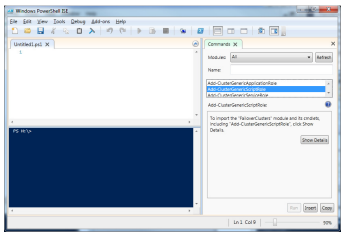


Console versus ISE

Er zijn 2 manieren hoe je default met powershell kunt werken.

De "command-line" Interface versus ISE (Interactive Scripting Environment





- **simpel**
- **Quick**
- **Niet echt**

- **Ontwikkelen van scripts**
- **Gebruiksvriendelijk**

CMDlets

Cmdlets zijn specifiek en speciaal ontworpen voor powershell.

Een cmdlet is 'de kleinste functionele eenheid in een PowerShell. Zo klein als ze zijn, zo handig zijn ze ook. De cmdlet, uit te spreken als 'command-let', voegt namelijk extra functionaliteit toe aan de Microsoft command-line. cmdlets zijn dus zeer gewild.
De naam van de commando's is over het algemeen zelf beschrijvend gekozen en bestaat uit een werkwoord gevolgd door een zelfstandig naamwoord. Met het commando set-date, is het dus bijvoorbeeld mogelijk om de huidige datum in te stellen, terwijl get-process een lijstje met draaiende processen teruggeeft. Met de opdracht get-command krijgt men een overzicht van de beschikbare Cmdlets.
Cmdlets zijn specifiek en speciaal voor powershell ontworpen. Wanneer je een parameter mee geeft aan een cmdlet zijn er 2 methodes die je kunt gebruiken.
Named parameters en Positionele parameters
Wanneer je een "Named parameter" gebruikt dan is het karakter dat aangeeft dat het om een named parameter gaat "-" Dit geldt voor alle cmdlets en zorgt voor consistentie in powershell. Bij een named parameter moet je de parameternaam verplicht invullen.

Voorbeelden	**Set-location –path c:\windows** **Get-service –name Netlogon**

Wanneer je positionele parameters gebruikt moeten de parameter waarden in de juiste volgorde staan maar de naam van de parameter kan weggelaten worden. Het reduceert het typen maar reduceert ook de leesbaarheid van het script.

Voorbeelden	**Set-location c:\windows** **Get-service Netlogon**

Tip: een cmdlet is altijd enkelvoud. Het is niet Get-Services maar Get-Service!!!

Essential CMDLETS

Get-Help

Get-help is absoluut een essentiële cmdlet en geeft gedetailleerde informatie en voorbeelden over cmdlets. Get-help accepteert wildcards en kan gebruikt worden voor het zoeken nar cmdlets op bepaalde tekst.

Voorbeeld	
Zoeken op cmdlets met wildcard (*)	Get-help *service*
Get-help van een specifiek cmdlet Get-help <cmdlet-name>	Get-help get-childItem
Get-help <cmdlet-name> -detailed Gedetaileerdere informatie dan alleen get-help Geeft ook voorbeelden	Get-help get-childItem -detailed
Get-help <cmdlet-name> -example Geeft alleen de voorbeelden weer	Get-help get-childItem -example
Get-help <cmdlet-name> -full Volledige info , examples, uitleg over parameters etc	Get-help get-childItem -full

Get-Member

Het Cmdlet Get-member is ook een essentieel commando. Met Get-member kun je namelijk alle eigenschappen (properties) en Methods van het object opvragen.

Voorbeelden	Get-childitem | get-member Get-ChildItem | GM (alias!!!) Get-service | Get-member

Get-Command

Met de opdracht get-command krijgt men een overzicht van de beschikbare Cmdlets.

Voorbeelden	Get-command Gcm Get-command –verb get Get-command –noun *service* Get-command –type cmdlet

OBJECTS

Om powershell goed te begrijpen moet je weten wat objecten zij. Objecten zijn simpel gezegd een digitale representatie van iets, dat bestaat uit onderdelen en waar je iets mee kunt doen.
Denk bijvoorbeeld aan een fiets.
Een fiets bestaat uit onderdelen, wielen, pedalen, zadel, frame , stuur, etc. We gebruiken deze collectie van onderdelen als een geheel. We kunnen trappen, linksaf sturen en bijvoorbeeld remmen. De "dingen" die we dus kunnen doen met de fiets, worden "methods" genoemd (acties).
In Powershell is bijna alles een object. Of we nu werken met mailboxen, gebruikers, processen, eventlogs, netwerkkaarten, etc., Alles wordt gerepresenteerd als een object en heeft dus "properties" en "methods".
Objecten in PowerShell zijn vergelijkbaar alleen in plaats van onderdelen hebben we "properties". In het voorbeeld van get-service is een property bijvoorbeeld de servicenaam of de status . Een "method" kan zijn stop of start. Eerder hebben we het al gehad over get-member. Met behulp van get-member kun je dus de "properties" en "methods" opvragen

Voor meer informatie : get-help about_objects

Voorbeelden Get-member van 1 service → Waardes van de properties →	Get-service | Get-Member get-service | where {$_.name –eq "winRM" }\| Get-Member get-service \| where {$_.name –eq "winRM" }| Format-list *

Objecten zijn simpel gezegd een digitale representatie van iets dat bestaat uit onderdelen en waar je iets mee kunt doen.

Onderdelen	Acties
Wielen	Vooruit
Stuur	rechtsaf
Rem	Remmen

Properties		Methods
name	Services	Stop/start
displayname		pause

The Pipeline

De pipeline is een erg krachtig component in PowerShell. De pipeline is het karakter "|". Het zorgt ervoor dat de output van een cmdlet wordt doorgezet als input naar het volgende cmdlet.
"|"word al jaren in vele Shells gebruikt. Maar PowerShell gebruikt het heel anders. De output van PowerShell zijn objecten i.p.v. tekst zoals bv in Linux/Unix

Voorbeelden

Get-service
Geeft een lijst met Service objecten terug.
Als we deze bv willen filteren dan kunnen we deze output pipen
Get-service | Where-object {$_.status –eq "running"}
Met dit commando filteren we uit alle services allen de services die running zijn. Met deze output kunnen we natuurlijk verder gaan.
Get-service | Where-object {$_.status –eq "running"} | stop-service –whatif
Wat we nu doen is dat we de running commando's pipen naar het cmdlet Stop-services. Als extra geven we de optie –whatif mee om te kijken wat er gebeurd als!!. Zonder –whatif krijg je namelijk een blue screen en crashed je pc ☺
Get-service | Where-object {$_.status –eq "stopped"} | format-list
Van alle services willen we alleen de services met de status stopped en deze willen in lijst geformateerd hebben

"The Pipeline" zorgt ervoor dat de output van een cmdlet wordt doorgezet als input naar het volgende cmdlet

Aliassen

Aliassen zijn verkorte namen (afkortingen) voor een commando/cmdlet en kan gebruikt worden i.p.v. de volledige naam. Als je niet goed kunt typen is dit een uitkomst. In scripts raden we het af om geen aliassen te gebruiken. Dit om de leesbaarheid van het script te bevorderen.
Tip: Als je de shell afsluit zullen de zelf aangemaakte aliassen verloren gaan

Voorbeelden	Get-help *alias* Get-alias get-command (-shows existing alias) New-alias gh Het-help (-aanmaken nieuwe alias)

Functions

Functies in de simpelste vorm, zijn vergelijkbaar met aliassen, maar kunnen complete script Blocks bevatten.

Voorbeelden	Get-help *function* Function scriptdir{set-location "c:\mystuff\ps scripts"} Function Get-BigFiles{dir –recurse | where-object {}$.Length –get 100mb}

"Functions" worden opgeslagen in de functie provider (zie provider) en zullen net zoals bij aliasen verloren gaan als je de shell afsluit.
Voor meer informatie : Get-help function
Get-Help about_function

Profiles

Profiles zijn simpel gezegd een "startup script" voor Powershell

Profiles zijn simpel gezegd een "startup script" voor Powershell. Elke keer als je Powershell start wordt dit gestart. Het is te vergelijken met bijvoorbeeld autoexec.bat. Een Powershell Profile bevat simple scriptregels, Functions, aliassen en soms erg complexe codes .
$profile is een speciale variabele die de locatie van het profile(s) bevat. Het bevat niet het profile zelf, maar de locatie.
$profile bestaat altijd, maar vanwege security redenen bestaat het actuele profile niet en zal je deze zelf aan moeten maken.

De snelste manier om je profile te maken, is door de volgende regel te runnen:
New-item –path $profile –type File –force

Profile	Path
Current User, Current Host	$Home\[My]Documents\WindowsPowerShell\Profile.ps1
Current User, All Hosts	$Home\[My]Documents\Profile.ps1
All Users, Current Host	$PsHome\Microsoft.PowerShell_profile.ps1
All Users, All Hosts	$PsHome\Profile.ps1

Providers

Een provider is een gemeenschappelijke interface naar verschillende datastores. Denk bij een datastore aan het "*file system*".
Een PowerShell provider is een adapter, deze zorgt er voor dat een bepaald data storage medium te zien is als een diskdrive

Voor meer informatie : get-help about_provider

Denk bij een datastore aan het file system