# Introductie Powershell

Juli 2016

## Introduction Powershell



## WMI

WMI staat voor Windows Management Instrumentation. WMI is een database die op iedere Windows computer aanwezig is (sinds Windows NT). In deze database staat informatie over de computer zelf, zoals de hoeveelheid geheugen, het type videokaart, de laatste updates en welke software is geïnstalleerd. Vrijwel alle gegevens over hard- en software is terug te vinden in deze database. Management Tools van Microsoft en andere partijen maken veelal gebruik van deze database.

**Geschiedenis**

In de midden jaren 90 was er een industriële groep genaamd de DMTF (Desktop Management Task Force). Tegenwoordig verwijst de D naar Distribited; Distributed Management Task Force. Microsoft was een lid van deze groep en het doel van deze groep was om te komen tot een leveransier-neutrale standaard voor het verkrijgen van managementinformatie, Distributed Management Informatie van computers uit je hele domein.

Het moet niet uitmaken wie de hardware of software leveransier is. Een van de dingen die ze moesten doen was het ontwikkelen van een manier om deze management informatie te tonen. Een standaard formaat waar iedereen het mee eens was. Ze kwamen met het Common Information Model. Tegenwoordig is het versie 2. CIMv2

Elke leveransier ging hier mee aan de slag. Microsoft kwam met Web Based Enterprise Management. Dit was in de tijd toen IIS nog het mooiste ooit was. IIS 5.0, voordat het uitdraaide op een mega beveilings probleem toen ze het op alle computers gingen installeren. Microsoft hakte het stuk web eraf en besloot om het Windows Management Instrumentaiton (WMI) te noemen.

*WMI was in eerste instantie ontworpen voor het lezen van Informatie.*

WMI is een middel voor het ophalen van management informatie van computers die deze gegevens opslaan in de cimv2 manier die in overeenstemming is met de afspraken beschreven door DMTF

WMI was in eerste instantie ontworpen voor het lezen van informatie. WMI wil informatie halen uit een Computer, Een servicepack nummer, Processor gegevens, Serienummer, over een applicatie die geïnstalleerd is .

Iets later ondekte microsoft dat dit ook een fantastische manier is om commando's te pushen voor bv configuratie veranderingen zoals het herconfigureren van een service, Het restarten of shutdownen van een computer etc. Dit zijn de dingen die we vandaag de dag kunnen doen met WMI.

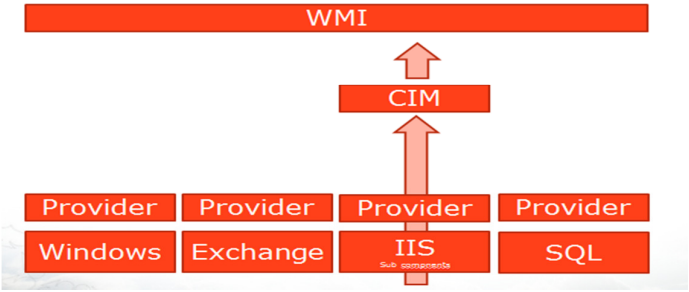
Microsoft is zoals je waarschijnlijk al weet verdeeld in meerdere verschillende productgroepen. We hebben een afdeling voor Windows, een ander team voor exchange, weer een ander team werkt aan subcomponenten zoals bv IIS, en we hebben een groep die werken aan SQL.

Elke groep is gevraagd om aan te gegeven wat voor hun product belangrijk/logisch is om in WMI te presenteren. De manier waarop ze dit doen is d.m.v. providers.

Het windows team schreef er een, Het exchange team kwam er met een, Het IIS team maakte zelfs een hele geweldige waar je veel mee kunt doen. Het SQL team bouwde er een. Enz.

*Het verschil tussen enkele en dubbele aanhalingstekens is dat PowerShell alles tussen enkele aanhalingstekens behandelt als letterlijke tekenreeks*

Deze providers zijn ontwikkeld om informatie uit het product te halen en deze beschikbaar te maken op de CIM manier. Microsofts heeft een standaard technologie daar boven op die deze info leest en beschikbaar maakt voor ons, voor scripts, commando's en voor Powershell




Je kunt gebruik maken van andere command utilities zoals Wmic.exe. Die runt onder cmd.exe, maar PowerShell heeft ook WMI mogelijkheden ingebouwd die redelijk vergelijkbaar zijn maar veel beter en veel gemakkelijker te gebruiken zijn dan WMIC.

Omdat al deze informatie je behoorlijk kan overspoelen was er een behoefte om dit te organiseren. En zo doende is WMI georganiseerd in namespaces. Elke namespace vertegenwoordigd een bepaald product. Bv IIS of DNS , AD, etc. Binnen deze namespaces zijn er 1 of meerder classes, Een class is een manageble Component. Bv een Diskdrive, of een cpu, een A record in DNS of misschien een Website in IIS. De classes zijn eigenlijk een beschrijving van wat de manageble compont is. Een disk heeft vrij schrijfruimte, een total size. De CPU heeft een serienummer, een Speed, Websites hebben allerlei configuraties en je kunt dit aanpassen of je kunt zelfs de website verwijderen.


*Powershell, een nieuwe manier van beheren.*

De volgende level is een instance. Voor een disk class zul je waarschijnlijk een instance hebben voor iedere diskdrive, Een dual processor zal een instance hebben voor iedere processor. Als je DNS server 100 A-records heeft dan zul je 100 instances hebben van het a-record class. Als IIS 2 websites heeft dan zul je 2 instances hebben van de website class

Wat de class en de namespace ook is wij zullen voornamelijk werken met de root\cimv2. Dit is namelijk de plek waar de meeste core windows OS spullen zich bevinden.

Omdat WMI zo groot is is het handig om hier een tool voor te gebruiken. Wmi-explorer bijvoorbeeld.

Help Get-wmiobject
Get-WmiObject [[-Class] <string>] verplicht

De -class parameter is een verplichte parameter die je mee moet geven. De root\cimv2 is de default namespace. De parameter namespace hoef je daarom alleen te gebruiken als je een andere namespace wilt bevragen.

**PS c:\> GWMI win32_service**
**PS c:\> GWMI win32_service |FL ***

Filteren in WMI. Dit is waar single en double quote's erg handig zijn. Plaats je Wmi filter altijd binnen double quote's. Dit is omdat WMI andere criteria hanteert dan Powershell voor filteren.
Dit kan bijvoorbeeld niet : **GWMI win32_service –filter "B*"**
Het zou moeten zijn "B%" De wildcard van WMI is % en niet * Dit is een beetje wennen.

*Het mooie aan WMI is dat er eigenlijk geen verschil is tussen werken op een remote computer of een lokale computer*

Hierbij plaats je de servicename in singel quote's en je sluit je de gehele filtering in Double quote's zodat het beter te lezen is.

**GWMI win32_operatingsystem**
**GWMI win32_operatingsystem | FL***

Het mooie aan WMI is dat er eigenlijk geen verschil is tussen werken op een remote computer of op een lokale computer.
Door gebruik te maken van de –computer parameter kun je een remote machine bevragen.

**GWMI Win32_operatingsystem –comp labsrv01**
**GWMI Win32_operatingsystem –comp " labsrv01","localhost"**
**GWMI Win32_operatingsystem –comp (type c:\demo\computers.txt)**

Doormiddel van het plaatsen van de computer.txt tussen haakjes forceer je dat dit eerst gestart wordt. Type (alias get-content) zal het tekstbestand lezen en de computernamen hieruit halen om deze vervolgens door te geven aan de –computername parameter. Dit is geen komma seperated list maar gewoon één computernaam per regel.

opgelet: Wildcard In WMI is % en niet *

# Remoting

Er zijn verschillende manieren om remote een computer te beheren.
We onderscheiden binnen PowerShell 3 manieren

**Windows management instrumentation (WMI)**
- Gebruikt RPC's voor communicatie
- Primair voor het ontvangen van management info

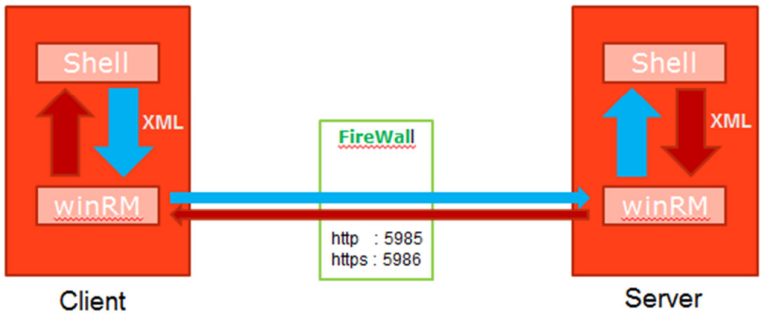
**De -computer parameter**
- bv Get-service and Get-process
- gebruikt onderliggende technologie voor communicatie
- Alleen beschikbaar op een paar cmdlets

**PowerShell remoting**
- generiek, goed voor alle commando's
- Gebruikt WinRM /WS-man voor communicatie
- De nieuwe standaard

# Powershell Remoting

*WSman is het protocol en WinRM de background service.*

**Het Idee achter Remote PowerShell**
Remote PowerShell werk eigenlijk een beetje hetzelfde als Telnet en ander ouderwetse remote control technologieën. Als je een commando runt dan run je die op de remote computer . Alleen de resultaten van dat commando komen terug naar jou computer. IPV Telnet of SSH gebruikt PowerShell een nieuw communicatie protocol. Web services for management ( WS-Man) . WSMAN opereert volledig over http en https en is daardoor makkelijk te routeren door firewalls. De implementatie van WS-man gebeurd volledig door een background services. Windows Remote Management. (winRM)
winRM wordt meegenomen sinds PS 2.0 en is default alleen gestart op servers (2008r2). Op Windows 7 is staat het standaard geïnstalleerd maar staat het op disabled.
WSman is het protocol, WinRM is de background services ( Niet alleen voor PowerShell) Veel; applicaties zullen straks endpoints krijgen via winRM en daardoor kun je alle management info binnenhalen via 1 poort dus dat maakt je firewall beheer een stuk gemakkelijker.




Powershell cmdlets genereren objecten als output. Wanneer je dus een remote commando uitvoert moet de output in een formulier geplaatst worden die makkelijk te transporteren is over het Netwerk via http (of https). XML is hier het ideale formaat voor dus Powershell zet automatisch de output objecten om naar XML. Deze worden getransporteerd over het netwerk naar jou computer en via WINRM weer in de shell geplaatst die hier weer prima mee overweg kan.
Waarom zouden we ons druk moeten maken over hoe de output terug gestuurd wordt? Nou omdat het snapshot's zijn. Ze updaten zich niet zelf. Dus als je remote de processen opvraagt dan zijn dat de processen van dat moment. Dit is een limitatie van Remoting maar laat je niet uit het veld slaan!!!.

1. Om remoting aan de praat te krijgen zijn er 2 basis voorwaarden
   Beide computers moeten minimaal PS v2 hebben (XP is het oudste windows os waar PSv2 op kan draaien)
2. Beide computers moeten lid zijn van het zelfde domein. (Of trusted/trusting domein)

**Ports:**
http → 5985
https → 5986

Als je default een andere port wilt definiëren dan kan dat. ( Winrm set ).
Nadeel is dat je altijd de poort moet specificeren bij een Remoting command.

GCM –noun Pssession
Enable-PSRemote

$session = New-PSSession -computername pc01,pc02 -name Domaincomputers
invoke-command {get-eventlog security -newest

## Remoting weetjes

- Staat standaard aan op Server 2008r2 en 2012. Niet op Clients.
- Aanzetten op de client:
  - Enable-PSRemoting
  - GPO
- Default authenticatie is met Kerberos
- Verbindingen worden gelogged als een network login
- Powershell profielen werken niet met Remoting
- Execution Policy van het remote systeem geldt
- Alleen rechten als je local admin op het remote systeem bent
- Minder resource gebruik op het systeem als met een Remote Desktop
  Output komt terug

*Welke WMI objecten er ook geproduceerd worden je zult hier ook wat mee moeten doen ( Dit is nesting)*

# Backgroundjobs

Deze feature is nieuw sinds ps 2.0.
D.m.v. background jobs kunnen we een reeks/aaneensluiting van commando's gestart worden en in de achtergrond gerund worden.
Er zijn vele manieren om background jobs te runnen en veel cmdlets staan toe om interactief gerund te worden . Hiervoor moet het cmdlet de parameter -asjob hebben

-Starting Jobs
-managing jobs
-Working with Job results

invoke-command {get-eventlog security -newest

Je zult zien dat bovenstaande commando even duurt en al die tijd kun je deze Commandline niet gebruiken. Voor 2 machines valt het uiteindelijk natuurlijk wel mee maar stel dat je dit voor 1000 computers moet doen !!!

D.m.v. de –asjob parameter kunnen we dit commando ook in de achtergrond laten draaien.

Het meegeven van de -jobname is niet noodzakelijk maar het maakt het wel overzichtelijk .
je ziet dat je commandline meteen terug hebt. We kunnen alles weer doen.
Als de job klaar is zal het op de achtergrond op ons wachten. (Niet de commandline afsluiten natuurlijk)

get-command –noun job

Get the results uit PS backgroundjobs
Je kunt op een paar manieren werken. Je kunt de job results opvragen door ze te pipen.
Als een job connect naar meerdere remote computers dan hebben we 1 masterjob dat alles coördineert en voor iedere computer een child Job maakt.


Als je receive job gebruikt dan zul je bij default alle resultaten voor de hele job (inclusief childjobs) ontvangen (→ recursion)
Als je dit niet wilt, gebruik je –norecursion
Bij Default zullen de resultaten niet gecashed blijven in het geheugen.
Als de job is afgerond zullen de resultaten in het geheugen opgeslagen worden. Wanneer je deze vervolgens opvraagt zullen ze uit het geheugen komen en gaan naar waar jij ze naar toe stuurt en zullen dan uit het geheugen zijn. Behalve als je ze naar een Variablen of bv een txt file exporteert.
Als je de –keep paramater gebruikt blijven de resultaten wel gewoon in het geheugen en kun je ze eventueel voor een 2e maal gebruiken.
Het is dus belangrijk om te onthouden dat als je niks doet de resultaten na het ontvangen verdwenen zijn.

| Cmdlet | |
|---|---|
| Get-Job | laat de jobs zien |
| Receive-Job | resultaten uit de job halen |
| Remove-Job | verwijderen van job |
| Start-Job | starten de job (op de local computer, als je op meerder computers iets wilt runnen dan moet je invoke command gebruiken) |
| Stop-Job | stopt de job |
| Wait-Job | freeze the shell (scripts) |

```
Get-job
Receive-job –name eventlogs –keep
$eventlogs = receive-job –name eventlogs –keep
$eventlogs | group-object pscomputername
$eventlogs | Where {$_.instanceid –eq 4634}
```

Door de resultaten in een variabelen te stoppen kun je er leuke dingen mee doen.

```
Get-job –name eventlogs | fl *
```

Je ziet dat de naam eventlogs die we gegeven hebben weer makkelijk te gebruiken is ipv job9.
Je ziet dat je meer properties kunt opvragen dan dat je via Get-job alleen zou krijgen.
Het childitem property is het gene dat we zoeken. Zoals we al eerder vertelde heb je een master job met child jobs voor iedere computer
Laten we het weer even in een variablen stoppen.

→ ziet er hetzelfde uit als get-job maar nu kun je de individuele properties bevragen.

```
$job.childjobs     → het is een array dus ik wil bv de eerste
$job.childjobs[0]
$job.childjobs[1]
Receice-job –name
```

Jobs gebruiken Ram geheugen en houdt deze hier dus als je er mee klaar bent dan is het goed om ze te verwijderen.

Oepss dat werkt niet. Dat klopt want je kunt geen childjob verwijderen. Je kunt alleen verwijderen d.m.v. het verwijderen van de parentjob.
Get-job –name eventlogs | remove-job.
Als de shell sluit ben je natuurlijk ook alles kwijt.

| The Basic lifcycle | |
|---|---|
| Start a job | (dmb start-job of de cmdlet parmeter –asjob) |
| Check a Job | of hij is afgerond. (dmv get-job of wait-job ; om speciefiek te wachten tot een job klaar is om met de resulaten verder te gaan.) |
| Receive-job | als de job klaar is wil je iets met de data doen ( -keep results / exporteer naar variablen of file |
| Remove-job | om de job uit het geheugen te verwijdern of gewoon de commandline sluiten |
| Stop-job | om een Background job te stoppen. Bv als hij al een uur bezig is terwijl hij er normaa 2 minuten over doet. DIT IS GEEN DELETE of REMOVE-JOB!!!!! |

# Links

### Microsoft Virtual Academy Powershell

https://mva.microsoft.com/en-us/training-courses/getting-started-with-powershell-3-0-jump-start-8276?l=r54lrOWy_2304984382

https://mva.microsoft.com/search/SearchResults.aspx#!q=powershell&lang=1033

### Don Jones

https://www.youtube.com/watch?v=6CRTahGYnws&list=PL6D474E721138865A

https://www.youtube.com/watch?v=-ERyfmOmyoI
https://www.youtube.com/watch?v=NI_8sJOu_fo

### Overig

http://powershell.org/

https://nl-nl.facebook.com/Dupsug