## Scripts maken en opslaan in Azure PowerShell

* 10 minuten

Complexe of repetitieve taken nemen vaak veel administratieve tijd in beslag. Organisaties geven er de voorkeur aan om deze taken te automatiseren om kosten te besparen en fouten te voorkomen.

Dit is belangrijk in het voorbeeld van Customer Relationship Management (CRM). Daar test u uw software op meerdere Virtuele Linux-machines (VM’s) die u voortdurend moet verwijderen en opnieuw moet maken. U wilt een PowerShell-script gebruiken om het maken van de VM’s te automatiseren in plaats van ze elke keer handmatig te maken.

Naast de kernbewerking van het maken van een VM, hebt u een paar aanvullende vereisten voor uw script:

* U maakt meerdere VM’s, dus u wilt de creatie in een lus plaatsen
* U moet VM’s maken in drie verschillende resource-groepen, dus de naam van de resource-groep moet als parameter aan het script worden doorgegeven

In deze sectie ziet u hoe u een Azure PowerShell-script schrijft en uitvoert dat aan deze vereisten voldoet.

## Wat is een PowerShell-script?

Een PowerShell-script is een tekstbestand met opdrachten en besturingsconstructies. De opdrachten zijn aanroepingen van cmdlets. De besturingsconstructies zijn programmeerfuncties zoals loops, variabelen, parameters, opmerkingen, enz., Geleverd door PowerShell.

PowerShell-scriptbestanden hebben de bestand-extensie **.ps1**. U kunt deze bestanden maken en opslaan met elke teksteditor.

Fooi

Als u PowerShell-scripts schrijft onder Windows, kunt u de Windows PowerShell Integrated Scripting Environment (ISE) gebruiken. Deze editor biedt functies zoals syntaxiskleuring en een lijst met beschikbare cmdlets.

In de volgende schermafbeelding ziet u de Windows PowerShell Integrated Scripting Environment (ISE) met een voorbeeldscript om verbinding te maken met Azure en een virtuele machine in Azure te maken.

Screenshot of the Windows PowerShell Integrated Scripting Environment with a script to create a virtual machine open in the editing window.

Nadat u het script hebt geschreven, voert u het uit vanaf de PowerShell-opdrachtregel door de naam van het bestand door te geven, voorafgegaan door een punt en een backslash:

.\myScript.ps1

## PowerShell-technieken

PowerShell heeft veel functies die te vinden zijn in typische programmeertalen. U kunt variabelen definiëren, vertakkingen en lussen gebruiken, opdrachtregelparameters vastleggen, functies schrijven, opmerkingen toevoegen, enzovoort. We hebben drie functies nodig voor ons script: variabelen, lussen en parameters.

### Variabelen

Zoals u in de vorige eenheid hebt gezien, ondersteunt PowerShell variabelen. Gebruik **$** om een variabele te declareren en **=** om een waarde toe te wijzen. Bijvoorbeeld:

$loc = "East US"  
$iterations = 3

Variabelen kunnen objecten bevatten. Met de volgende definitie wordt bijvoorbeeld de variabele **adminCredential** ingesteld op het object dat wordt geretourneerd door de cmdlet **Get-Credential**.

$adminCredential = Get-Credential

Als u de waarde wilt verkrijgen die is opgeslagen in een variabele, gebruikt u het voorvoegsel **$** en de naam ervan, zoals in het volgende geval:

$loc = "East US"  
New-AzResourceGroup -Name "MyResourceGroup" -Location $loc

### Loops

PowerShell has several loops: **For**, **Do…While**, **For…Each**, and so on. The **For** loop is the best match for our needs, because we will execute a cmdlet a fixed number of times.

The core syntax is shown below; the example runs for two iterations and prints the value of **i** each time. The comparison operators are written **-lt** for “less than”, **-le** for “less than or equal”, **-eq** for “equal”, **-ne** for “not equal”, etc.

For ($i = 1; $i -lt 3; $i++)  
{  
 $i  
}

### Parameters

When you execute a script, you can pass arguments on the command line. You can provide names for each parameter to help the script extract the values. For example:

.\setupEnvironment.ps1 -size 5 -location "East US"

Inside the script, you’ll capture the values into variables. In this example, the parameters are matched by name:

param([string]$location, [int]$size)

You can omit the names from the command line. For example:

.\setupEnvironment.ps1 5 "East US"

Inside the script, you’ll rely on position for matching when the parameters are unnamed:

param([int]$size, [string]$location)

We could take these parameters as input and use a loop to create a set of VMs from the given parameters. We’ll try that next.

The combination of PowerShell and Azure PowerShell gives you all the tools you need to automate Azure. In our CRM example, we’ll be able to create multiple Linux VMs using a parameter to keep the script generic and a loop to avoid repeated code. This means that we can execute a formerly complex operation in a single step.

## Next unit: Exercise - Create and save scripts in Azure PowerShell

[Continue](https://docs.microsoft.com/en-us/learn/modules/automate-azure-tasks-with-powershell/8-exercise-create-resource-using-script/)