探索 PowerShell 命令

跟所有程式語言一樣，PowerShell 也有自己的命令，亦即我們對於具名可執行運算式的一般稱呼。一個命令可以有各種形式，從舊有的 *ping.exe* 工具、到剛剛才介紹過的 `Get-Alias` 命令都是。甚至你還可以自行建置自己的命令。但是如果你嘗試使用一個不存在的命令，就會領教到惡名昭彰的大紅字警告，如清單 1-2 所示。

```
PS> foo
foo : The term 'foo' is not recognized as the name of a cmdlet, function,
script file, or operable program. Check the spelling of the name, or if a
path was included, verify that the path is correct and try again.
At line:1 char:1
+ foo
+ ~~~
    + CategoryInfo          : ObjectNotFound: (foo:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

清單 1-2：輸入無法辨識的命令後，就會顯示以上的錯誤訊息。

各位也可以試著執行 `Get-Command`，觀看 PowerShell 預設的所有命令清單。你也許會注意到一個共通的模式。大多數的命令名稱都有相同的格式：*動詞 - 名詞*。這是 PowerShell 獨有的特徵。為了讓這種語言儘量直覺化，微軟為命令的命名訂出了方針。雖說是否要遵循這個命名慣例全看自己，我們還是鄭重建議以這個慣例來創建你自己的命令。

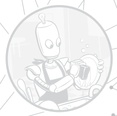
PowerShell 的命令分成這幾種：cmdlets、函式、別名、有時還有外來的指令碼。大部份來自微軟的內建命令都是 *cmdlets*，這些命令通常係以 C# 之類的其他語言撰寫而成。若是執行 `Get-Command` 命令，如清單 1-3 所示，就會看到 `CommandType` 這個欄位。

```
PS> Get-Command -Name Get-Alias

CommandType     Name                Version    Source
-----------     ----                -------    ------
Cmdlet          Get-Alias           3.1.0.0    Microsoft.PowerShell.Utility
```

清單 1-3：顯示 Get-Alias 命令的類型

碁峯 www.gotop.com.tw



PARSING STRUCTURED DATA

JSON 資料

如果你在過去五年中都從事資訊科技領域工作，必然對 JSON 不陌生。它誕生於 2000 年初期，全名為 *JavaScript Object Notation*（*JSON* 是縮寫），屬於一種可供機器閱讀、人類亦可理解的語言，用於呈現階層式的資料集。如同其名稱所示，JavaScript 應用程式對它的運用最多，亦即它在網頁開發中有極顯要的地位。

最近使用 *REST API*（這是一種可以在用戶端與伺服器之間傳送資料的技術）的線上服務數量有激增之勢，也連帶使得 JSON 的用量大增。如果你正以網頁處理一些事情，JSON 絕對是值得理解的格式，而且 PowerShell 也能輕易地管理它。

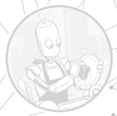
讀取 JSON

PowerShell 讀取 JSON 時有一點跟 CSV 相似，就是方式不只一種：要剖析、或不加剖析。由於 JSON 也是純文字格式，PowerShell 預設也將其視為字串。舉例來說，請從本章資源網頁取得 *Employees.json* 這個 JSON 檔案，內容如下：

```
{
    "Employees": [
        {
            "FirstName": "Adam",
            "LastName": "Bertram",
            "Department": "IT",
            "Title": "Awesome IT Professional"
        },
        {
            "FirstName": "Bob",
            "LastName": "Smith",
            "Department": "HR",
            "Title": "Crotchety HR guy"
        }
    ]
}
```

如果你只要看字串輸出，只需使用 `Get-Content -Path Employees.json -Raw` 就可以讀取檔案，並傳回字串內容。但字串能派上的用場不多。你需要的是結構的資訊。要取得這部份，必須要有可以理解 JSON 結構描述

碁峯 www.gotop.com.tw



執行以上命令後，應該會看到一長串的屬性輸出到畫面上；這些都是 web app 的各種相關設定。

部署一個 Azure 的 SQL 資料庫

另一種常見的 Azure 任務，就是部署一個 Azure 的 SQL 資料庫。要部署一套 Azure 的 SQL 資料庫，必須做三件事：首先是建立一個資料庫賴以運作的 Azure SQL 伺服器，其次是建立資料庫本身，最後則是建立一個 SQL 伺服器的防火牆規則，以便允許連接資料庫。

一如往常，各位必須先建立一個容納相關資源的資源群組。執行 `New-AzResourceGroup -Name 'PowerShellForSysAdmins-SQL' -Location 'East US'` 就可以做到。然後就要建立資料庫賴以運作的 SQL 伺服器。

建立一套 Azure 的 SQL 伺服器

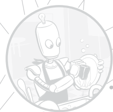
建立 Azure 的 SQL 伺服器一樣也只需要一道命令：`New-AzSqlServer`。但是也一樣必須提供資源群組名稱、SQL 伺服器本身的名稱、還有地域（但此處還須加上伺服器的 SQL 管理員的使用者名稱及其密碼）。這會多費點功夫。由於你必須建立一份認證（credential）以便傳給 `New-AzSqlServer`，我們先處理這個部份。筆者在 196 頁的「建立一個服務主體」小節中已經介紹過如何建立 `PSCredential` 物件了，所以這裡不再贅述。

```
PS> $userName = 'sqladmin'
PS> $plainTextPassword = 's3cretp@SSw0rd!'
PS> $secPassword = ConvertTo-SecureString -String $plainTextPassword -AsPlainText -Force
PS> $credential = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
$userName,$secPassword
```

一旦有了身分認證，剩下就只需把所有的參數放進一個雜湊表，再把雜湊表餵給 `New-AzSqlServer` 函式即可，如清單 12-20 所示。

```
PS> $parameters = @{
    ResourceGroupName = 'PowerShellForSysAdmins-SQL'
    ServerName = 'PowerShellForSysAdmins-SQLSrv'
```


碁峯 www.gotop.com.tw



先決條件

筆者假設各位已經有一個 AWS 帳戶，而且有權取用 root 使用者。你可以到 *https://aws.amazon.com/free/* 註冊一個 AWS 免費帳戶。並非所有的事都要靠 root 才能完成，但各位還是需要建立自己的第一個身分和存取管理（*identity and access management*（*IAM*））的使用者。你還得下載和安裝 `AWSPowerShell` 模組，如前所述。

AWS 認證

在 AWS 裡，認證是靠 IAM 服務來進行的，該服務負責處理 AWS 中的一切認證、授權、計量（accounting）、以及稽核（auditing）。要在 AWS 中進行認證，你的訂閱內容必須包括一個 IAM 使用者，而該使用者必須有權取用適當的資源。要處理 AWS 的第一步，就是建立一個 IAM 使用者。

一旦建立了 AWS 帳戶，就會自動隨之建立一個 root 使用者，於是就可以透過 root 使用者來建立你的 IAM 使用者。從技術上說，你確實可以只用 root 使用者在 AWS 裡從事任何行為，但我們鄭重建議不要如此。

以 Root 使用者認證

我們先來建立一個 IAM 使用者，以便在接下來的章節中運用。首先要先通過認證。如果還沒有另一個 IAM 使用者的身分可用，唯一能通過認證的方式就是利用 root 使用者。亦即這時你得把 PowerShell 放在一邊。先用 AWS 管理主控台（AWS Management Console）的圖形使用介面，取得 root 使用者的存取金鑰（access key）和私密存取金鑰（secret key）。

首先登入你的 AWS 帳戶。然後瀏覽至畫面右上角，點開帶有你的 AWS 帳戶名稱的下拉式選單，如圖 13-1 所示。

碁峰 www.gotop.com.tw

在 AWS 中建立一個 SQL 伺服器資料庫

身為 AWS 管理員，你會需要設置各種類型的關聯式資料庫。AWS 提供了 Amazon 關聯式資料庫服務（Amazon Relational Database Service，簡稱 Amazon RDS），允許管理員輕而易舉地開通數種資料庫。資料庫選項有好幾種，但目前我們只需專注在 SQL 上就好。

在這個小節中，各位要在 RDS 中建立一個空白的微軟 SQL 伺服器資料庫。主要的命令為 `New-RDSDBInstance`。就跟 `New-AzureRmSqlDatabase` 一樣，`New-RDSDBInstance` 也有大量的參數可以搭配，數目多到用這個小節也講不完。如果各位對於其他開通 RDS 執行個體的方式有興趣，筆者鼓勵大家去看看 `New-RDSDBInstance` 的說明內容。

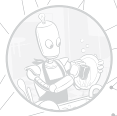
要達成目標，需要先準備以下資訊：

- 執行個體的名稱
- 資料庫引擎（SQL Server、MariaDB、MySQL 等等）
- 執行個體的類別，指定 SQL 伺服器運行所需的資源類型
- master 使用者名稱和密碼
- 資料庫的大小（單位為 GB）

此處有幾件事是各位可以輕易搞清楚的，包括：資料庫執行個體名稱、使用者名稱與密碼、以及資料庫大小等等。其他則需要進一步蒐集資訊。

我們先從資料庫引擎的版本著手。請利用 `Get-RDSDBEngineVersion` 命令取得所有現有的資料庫引擎及其版本清單。此一命令若不加上參數，就會傳回大量的資訊——遠遠超過我們調查所需的資訊。請加上 `Group-Object` 命令，按照資料庫引擎的類別，將輸出的物件分類，這樣就可以很容易地看出各種單一引擎的所有版本編號清單。如清單 13-26 所示，分類後的輸出顯然清爽許多，可以輕鬆地分辨出有哪些引擎可用。

碁峯 www.gotop.com.tw




```
PS> Get-Module -Name PowerLab -ListAvailable

    Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Script     1.0        PowerLab
```

萬一 PowerLab 模組沒有出現在輸出的底端，請回到前一步檢查。此外也請檢視 *C:\Program Files\WindowsPowerShell\Modules* 目錄下，是否有 *PowerLab* 資料夾存在、其中是否有 *PowerLab.psm1* 和 *PowerLab.psd1* 兩個檔案的蹤影。

自動化開通虛擬環境

現在我們已經做好模組的骨架，可以著手為其添加功能了。由於建立 SQL 或 IIS 伺服器之類的任務，都包含各種彼此休戚相關的步驟，因此應當首先著手準備以自動化方式建立一個虛擬交換器、虛擬機器、和虛擬磁碟。然後才能把日後為上述 VM 部署作業系統的動作也自動化，最後才能在這些 VM 上安裝 SQL 伺服器及 IIS。

虛擬交換器

要能自動化建立 VM 之前，必須先確保在 Hyper-V 主機上已有虛擬交換器的存在。有了*虛擬交換器*（*virtual switches*），VM 才能與用戶端機器、或是建置在相同宿主主機上的其他 VM 相互溝通。

手動建立一個虛擬交換器

第一個虛擬交換器必須要是一個 *external*（外部的）交換器，我們將其命名為 `PowerLab`。Hyper-V 主機中很可能還沒有這個交換器存在，但為謹慎起見，我們還是先把主機上既有的虛擬交換器列出來。小心點總不是壞事。

碁峰 www.gotop.com.tw

要把剛建立的 VHDX 設為第一個開機裝置，請執行 `Set-VMFirmware` 命令並修改 `FirstBootDevice` 參數：:

```
$vm | Set-VMFirmware -FirstBootDevice $vm.HardDrives[0]
```

到此應該有一部名為 LABDC 的 VM 存在、並裝上了虛擬磁碟，可以開機進入 Windows。請執行 **`Start-VM -Name LABDC`** 啟動該 VM，確認它確實進入 Windows。到此就算是成功了！

自動化部署 OS

到目前為止，各位已經成功地建立了名為 LABDC 的 VM、並開機進入 Windows。現在的重點是，各位必須了解，目前使用的指令碼是特別為單一 VM 訂製的。在現實世界中不太可能這麼有福氣。好的指令碼必須可以重複使用、也易於移植，亦即不需因為特定的輸入而需要更改指令碼，而是只需靠一組可以變化的參數就能運作。

我們來觀察一下 PowerLab 模組裡的 `Install-PowerLabOperatingSystem` 函式，本章的下載資源裡就有一份可以參考。該函式是改寫 *Install-LABDCOperatingSystem.ps1* 指令碼的最佳示範，改寫後就可以用來替不同的虛擬磁碟部署作業系統，只要改動一下參數就可以了。

筆者不會在這個小節中逐一詳述整段指令碼，因為其中大部份的功能都已在前一小節說明過了，但筆者必須指出其中的差異何在。首先請注意，改寫後版本的變數會更多。變數可以讓指令碼更富於彈性。它們可以做為替資料值預留的位置（placeholder），而不再是把資料值直接寫在程式碼當中。

此外也請注意指令碼當中的條件邏輯。請觀察清單 16-1 的程式碼。這是一個 `switch` 陳述，會根據作業系統名稱決定 ISO 檔案的路徑。前一小節的指令碼不需要這個寫法，是因為所有內容都已寫死在原有指令碼中的緣故。

由於 `Install-PowerLabOperatingSystem` 函式擁有 `Operating System` 這個參數，因而具備了可以安裝其他作業系統所需的彈性。只需找出可以因應

碁峯 www.gotop.com.tw

```
    [Parameter()]
    [string]$VhdPartitionStyle = 'GPT',

    [Parameter()]
    [string]$VhdBaseFolderPath = 'C:\PowerLab\VHDs',

    [Parameter()]
    [string]$IsoBaseFolderPath = 'C:\PowerLab\ISOs',

    [Parameter()]
    [string]$VhdPath
)
```

利用 Install-PowerLabOperatingSystem 函式，就可以把所有的內容整合在一行命令內，但同樣可以支援成打的各種組態。現在你手中有一個統合的程式碼單元，可以隨心所欲地呼叫，而完全不用改動到指令碼的任何一行內容！

把加密過的認證儲存在磁碟中

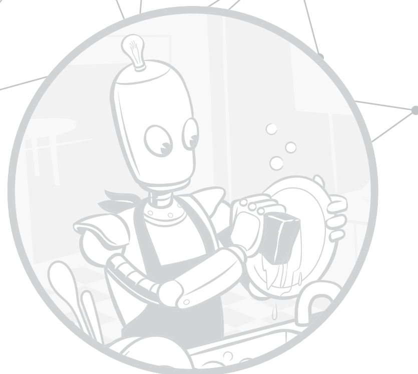
我們很快就可以結束這個階段的專案內容了，但在進入下個階段前，我們得繞一點路。這是因為我們要讓 PowerShell 從事一些需要身分認證（credential）的工作之故。指令碼中含有明文形式的敏感性資訊已經是常態（例如使用者名稱與密碼）。同樣地，一般也公認此舉在測試環境中見怪不怪（不過這是個壞習慣）。重點是，就算只是測試，也必須對安全措施有所警覺，這樣才能在測試內容轉移至正式環境時，仍能維持良好的安全性。

避免在指令碼中採用以明文儲存密碼，最簡單的方式就是將其加密為檔案。需要引用密碼時，指令碼就會將其解密、然後引用。幸好 PowerShell 提供了原生的處理方式：也就是 Windows Data Protection API。這個 API 暗藏在 Get-Credential 命令底下，而該命令會傳回 PSCredential 物件。

Get-Credential 會建立一個加密形式的密碼，亦即所謂的安全字串（*secure string*）。一旦變成安全字串格式，整個身分認證物件就可以安心地用 Export-CliXml 命令儲存到磁碟當中；反過來也可以用 Import-CliXml

碁峰 www.gotop.com.tw

DEPLOYING ACTIVE DIRECTORY



17

部署 Active Directory

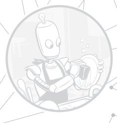
在這一章裡，各位要把過去在第二篇中所學的內容拿出來，開始在自己的虛擬機器上部署服務。由於有許多其他的服務必須仰賴 Active Directory，因此各位必須先部署一個 Active Directory 的樹系及網域。AD 樹系與網域將會支援以下其餘章節中所需的認證和授權。

假設各位已經依序讀到這裡，也在先前章節中開通了 LABDC VM，就可以利用該 VM，把 Active Directory 樹系的開通完全自動化，並填入若干測試用的使用者和群組。

先決條件

在本章中，各位會用到第 16 章時製作的內容，因此筆者假設各位手邊應該都已經設好 LABDC 這個 VM，而且也已經透過自動應答的 XML 檔案裝好 Windows Server 2016、並正常開機執行。這樣就可以繼續讀下去了！如果沒有，也還是可以讀完這一章的範例，了解如何將 Active Directory 自動化，但是請留意：你無法完全照著本章的內容作練習。

碁峰 www.gotop.com.tw



一如既往，請執行相關用於先決條件檢測的 Pester 測試，確保本章所需的所有先決條件都已經達到。

建立一個 Active Directory 樹系

好消息是，如果把所有的內容都考慮一遍，其實 PowerShell 建立一個 AD 樹系算是相當容易的。說到頭來，基本上只會用到兩個命令：`Install-WindowsFeature` 和 `Install-ADDSForest`。只需借助這兩道命令，就能建立一個單一樹系、建立一個網域、並開通一部作為網域控制站的 Windows 伺服器。

由於各位是在實驗環境中使用這個樹系，因此必須自行建立若干組織單位、使用者、以及群組。也由於這是測試環境之故，因此你不會處理到什麼正式環境的物件。不需煩惱正式環境與實驗環境間的 AD 物件同步問題，相反地，你還可以盡情地建立各種物件來模擬正式環境，讓自己有可以練習的物件。

建置一個樹系

建立全新的 AD 樹系時，首先要做的便是升級一部*網域控制站*，它是 Active Directory 中最根本的共同基礎。要讓 AD 環境運作，必須至少要有一部網域控制站存在。

由於這只是實驗環境，網域控制站只要有一部就夠了。但是在現實世界中，網域控制站至少要有兩部，以便達成容錯備援的效果。然而由於實驗環境中沒什麼資料，而且很快地就可以從無到有建立出一個新環境，因此一部網域控制站應該就足敷所需。在開始動手之前，必須先在 LABDC 伺服器上安裝 `AD-Domain-Services` 這個 Windows 功能。安裝該 Windows 功能的命令是 `Install-WindowsFeature`：

```
PS> $cred = Import-CliXml -Path C:\PowerLab\VMCredential.xml
PS> Invoke-Command -VMName 'LABDC' -Credential $cred -ScriptBlock
{ Install-windowsfeature -Name AD-Domain-Services }
PSComputerName : LABDC RunspaceId : 33d41d5e-50f3-475e-a624-4cc407858715
Success : True RestartNeeded : No FeatureResult : {Active Directory Domain
Services, Remote Server Administration Tools, Active Directory module for
Windows PowerShell, AD DS and AD LDS Tools...} ExitCode : Success ```
```


碁峯 www.gotop.com.tw

```
        $image = Mount-DiskImage -ImagePath 'C:\en_sql_server_2016_standard_x64_
dvd_8701871
        .iso' -PassThru ❶
        $installerPath = "$(($image | Get-Volume).DriveLetter):"
        $null = & "$installerPath\setup.exe" "/CONFIGURATIONFILE=C:\$($using:tempFile.
Name)" ❷
        $image | Dismount-DiskImage ❸
    }
}
Invoke-Command @icmParams
```

清單 18-5：利用 *Invoke-Command* 掛載、安裝和卸載映像檔

首先，我們把複製過來的 ISO 檔案掛載至遠端機器 ❶；然後執行安裝檔，同時把不需要的輸出訊息拋往 `$null` ❷；一旦全部完成，就把映像檔卸載 ❸。在清單 18-5 當中，我們以 `Invoke-Command` 和 PowerShell Direct 來遠端執行以上的命令。

當 SQL Server 安裝完畢，就必須進行若干清理工作，以便把先前所有複製到位的暫存檔都清除，如清單 18-6 所示。

```
$scriptBlock = { Remove-Item -Path 'C:\en_sql_server_2016_standard_x64_dvd
_8701871.iso', "C:\$($using:tempFile.Name)" -Recurse -ErrorAction Ignore }
Invoke-Command -ScriptBlock $scriptBlock -Session $session
$session | Remove-PSSession
```

清單 18-6：清理暫存檔

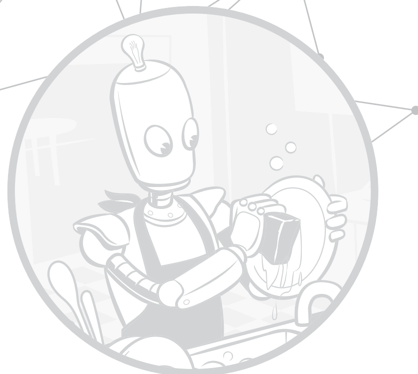
此時 SQL Server 已經設置完畢，可以使用了！只花了 64 行的 PowerShell 程式碼，就從無到有建立了一套微軟的 SQL Server，唯一預先存在的只有 Hyper-V 主機。這是一個了不起的過程，但是它還有改善空間。

SQL 伺服器自動化

苦工已經做得差不多了。目前各位手中已經有一段可以完成所有必要動作的指令碼。接下來要做的，就是把這一切功能變成 PowerLab 模組裡的各個函式：亦即 `New-PowerLabSqlServer` 和 `Install-PowerLabOperatingSystem` 這兩個函式。

碁峰 www.gotop.com.tw

REFACTORING YOUR CODE



19

重構程式碼

在前一章當中，各位只憑藉著一台既有的 hypervisor 主機、一個作業系統 ISO 檔案、再加上一點程式碼，就建構出了一台作為 SQL 伺服器的 VM。這個動作意味著我們必須把許多在先前章節中寫好的函式串在一起。但在本章裡，各位要做一點不一樣的事情：暫時不再為 PowerLab 模組添加新功能，而是深入重讀自己的程式碼，看看是否能把模組再改得更模組化一點。

當筆者提及模組化（*modular*）一詞時，意思是要把程式的功能分拆成可以重複使用的函式，以便因應各種情況。程式模組化的越徹底、其適用的彈性就越大。而它能適應的情況越多，就表示其可用性越好。若程式碼能模組化，意即我們可以隨意引用 `New-PowerLabVM` 或 `Install-PowerLabOperatingSystem` 等函式來安裝不同種類的伺服器（如下一章所述）。

碁峰
www.gotop.com.tw