Introducing PowerShell Desired State Configuration (DSC)

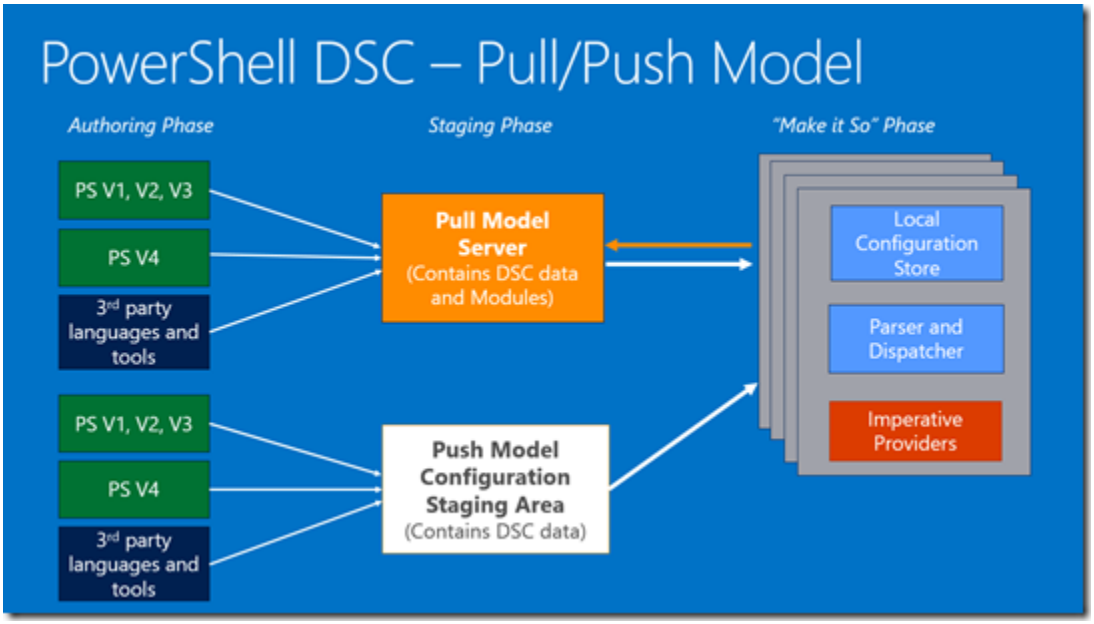
What is DSC?

DSC is a PowerShell extension and ships with Windows Server 2012 R2 and Windows 8.1. A couple of examples what DSC enables you to do are:

- Install or remove server roles and features
- Manage registry settings
- Manage files and directories
- Start, stop, and manage processes and services
- Manage local groups and user accounts
- Install and manage packages such as .msi and .exe
- Manage environment variables
- Run Windows PowerShell scripts
- Fix a configuration that has drifted away from the desired state
- Discover the actual configuration state on a given node

How does DSC work?

The following section describes the DSC flow, phases and the two different models: Push and Pull. In the illustration below both models are shown.



Let's dive into the different phases.

Authoring Phase

In this phase the DSC configuration is created through PowerShell or by third party languages and tools. The output from the Authoring Phase is one or more MOF (Management Object Format) files, the format which is consumable by DSC. MOF files can be created in multiple ways. You could use Notepad.exe if you wanted to. Using PowerShell v4 adds declarative syntax extensions and IntelliSense for making it easier to create a MOF file. It handles schema validations as well. The section below called *"Creating a DSC configuration"* will explain further how to create a DSC configuration.

Staging Phase

In this phase DSC data (MOF files) is staged. In case of adopting the Pull Model, which is likely the model which is going to be used the most, DSC data and custom providers are kept on the Pull Server. A Pull Server is an IIS webserver with a well defined OData interface. The target system contacts the Pull Server by passing an URI and an unique identifier to retrieve its configuration and verifies if all the custom providers are available. If not, those are downloaded to the target system. In case of using the Push Model, DSC data is being pushed to the target system. One catch though is that you need to make sure that your custom provider exists on the target system. Make sure that you put it here: *"%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\Modules\PSDesiredStateConfiguration\PSProviders"*

"Make it So" Phase

The final phase is to apply (enact) the configuration, to "make it so" as Jean-Luc Picard would say. DSC data is either pulled or pushed to the *"Local Configuration Store"* and contains the current, previous and the desired (DSC) state configuration. The configuration then gets parsed and the relevant (WMI) provider implements the change and *"makes it so".*

DSC comes with 12 out of the box providers. These providers enable you to configure roles and features, copy files, creating a registry entry, managing services, creating local users and groups, etc. See http://technet.microsoft.com/en-us/library/dn249921.aspx for a full listing. You can also create your own custom provider. How to do that can be found here: http://technet.microsoft.com/en-us/library/dn249927.aspx

Creating a DSC configuration

For a DSC configuration to be consumed, you need to declare it first. DSC consumes MOF (Management Object Format) files, a DMTF standard, which can actually be created with any text editor. PowerShell v4 makes it easier to create a MOF file by adding declarative syntax extensions and IntelliSense. It handles schema validations as well. The PowerShell v4 DSC

extension adds a new keyword called *"Configuration"*. With this keyword, which is actually a *"function"*, you define a DSC configuration like in the example below:

```
001 Configuration ContosoWebsite
002 {
003 }
```

Now you can go ahead and extend your configuration by adding for example "roles and features":

```
001  Configuration ContosoWebsite
002  {
003    param ($MachineName)
004
005
006    Node $MachineName
007    {
008      #Install the IIS Role
009      WindowsFeature IIS
010      {
011        Ensure = "Present"
012        Name = "Web-Server"
013      }
014
015      #Install ASP.NET 4.5
016      WindowsFeature ASP
017      {
018        Ensure = "Present"
019        Name = "Web-Asp-Net45"
020      }
021    }
   }
```

Please notice the usage of *"Node"* and *"WindowsFeature"* in the snippet above. With *"Node"* you can specify your target systems so that a specific node could have a specific configuration. The usage of *"WindowsFeature"* is an example of one of the DSC providers you can use.

Now that you have declared your DSC configuration you can create a DSC consumable MOF file by calling the function just as you would call any PowerShell function:

```
001 ContosoWebsite -MachineName "MyServer"
```

This will create a folder with the same name as your configuration name and will contain our MOF output file.

Applying a DSC configuration

Now that we have a MOF file which can be consumed by DSC, we can apply it by using the *"Start-DscConfiguration"* cmdlet:

```
001 Start-DscConfiguration –Path .\ContosoWebsite –Wait –
    Verbose
```

The *"Path"* parameter (the path where your MOF file is stored) can be a UNC or a local path.

How to detect configuration drift

If you want to compare the current and the actual configuration, you can use the cmdlet *"Test-DscConfiguration"* in the following way:

```
001 Test-DscConfiguration –CimSession $session
```

This will either return a *"True"* (when the current and actual configuration match) or *"False"* (if there's a mismatch).

Why is DSC so cool?

- First of all declaring a DSC configuration is PowerShell based. So you can leverage all your PowerShell skills to not only define a configuration, but also for troubleshooting.
- DSC is designed to support "continuous deployments" which means that you can deploy your configuration over and over without breaking anything
- When a DSC configuration is being applied only those settings which do not match will be set, the rest will be skipped which can result in a faster deployment time
- You can separate the configuration data from the logic of your configuration so that you can reuse your configuration data for different resources, nodes, and configurations, see http://technet.microsoft.com/en-us/library/dn249925.aspx
- DSC can be used on-premise, in a public or in a private Cloud environment. You just need either Windows Server 2012 R2 or Windows 8.1 and local administrator permissions to execute the DSC PowerShell scripts

- You can integrate DSC with any Microsoft or non-Microsoft solutions as long as you can execute a PowerShell script on the target system. Using DSC within the Windows Azure Pack portal in conjunction with SMA is a good example