Global Information Assurance Certification Paper

Configuration Management with Windows PowerShell Desired State Configuration (DSC)

GIAC (GSEC) Gold Certification

Author: Brian E. Quick, brian@brianequick.com
Advisor: Hamed Khiabani, Ph.D.
Accepted: Aug 15th 2015

Abstract

Keeping information system baselines consistent with a formal configuration management plan can be a very difficult task. Changes to server based systems and networking must be monitored in order to provide some measure of compliance. A new distributed configuration management platform by Microsoft® called Desired State Configuration (DSC) makes this task easier.

The objective of this paper is to describe in depth how PowerShell 4.0 can help to solve this common problem. DSC uses a declarative syntax that any skilled administrator can utilize to deploy software, monitor configuration drift and even report conformance. DSC is cross-platform compatible with hundreds of useful resources freely available. DSC leverages PowerShell 4.0 and gives administrators a useful way to automate configuration management.

1. Introduction

Every organization serious about information system security must be able to account for configuration changes. Most organizations create a formal configuration management (CM) plan but struggle to control configuration changes. Information systems are constantly changing to make services available to customers while balancing performance with adequate security. A recent Algosec network security survey concluded that poor change management poses the greatest challenge in managing risk due to poor processes and a lack of information system visibility. More than 80% of respondents experienced network or application outages resulting from out-of-process changes (Algosec, 2014). Organizations need more reliable automated mechanisms that help identify information system changes, control unauthorized changes and validate a formal change management process. Microsoft® has a relatively new feature called Desired State Configuration (DSC) released with Windows PowerShell 4.0. Windows PowerShell is also called Windows Management Framework because of its fundamental design. DSC can provide the reliability and extensibility needed to plan, deploy and monitor configuration changes. Any organization with a Microsoft® Windows network and administrators adept with PowerShell could use DSC to make configuration management goals a reality.

2. Why Adopt Desired State Configuration (DSC)

DSC offers some measurable benefits over group policy objects (GPO). DSC is capable of measuring if the configuration of specific nodes has drifted from an approved baseline. Measuring and communicating GPO effectiveness is often difficult in large enterprises. To measure GPO effectiveness, administrators frequently resort to the `gpresult` command commonly used to troubleshoot GPO conflicts and analyze the Resultant Set of Policy. Conflicts and errors in applying GPO's are also common with filtering, linking, blocking of inheritance or other GPO's controlling an object due to higher precedence. GPO's can also be relatively easy to defeat if the end user wishes to prevent a given GPO from being applied. GPO's require Windows Active Directory and

brian@brianequick.com

are cumulative in the application of configuration policy, but these are not requirements for DSC.

An organization wishing to more effectively monitor and control the configuration of critical nodes may have to consider acquiring a third party application to accomplish the task. However, what if this configuration management and reporting capability could be included in a free upgrade? This DSC capability comes with the Windows Management Framework (WMF) 4.0. Even greater DSC enhancements will come with WMF 5.0, which will be provided to eligible Windows desktop operating systems as a free upgrade built into Windows 10 to be released on 29 July 2015. An organization managing a Windows based network infrastructure could benefit tremendously from this capability to control and fix configuration drift.

A first step for any organization that primarily uses a Windows network is to conduct an inventory of operating system versions and PowerShell versions that may already be installed. DSC also requires the installation of the .NET Framework 4.5 as a pre-requisite to installing WMF 4.0. An organization could use the following information found in Table 1 to assess its current Windows infrastructure.

OS Version	Operating System Name	PS/WMF Version	DSC Capable	Requires PS/WMF 4.0 and .NET 4.5
10	Windows 10	5.0	Yes	No
6.3	Server 2012 R2	4.0	Yes	No
6.2	Server 2012	3.0	Yes	Yes
6.3	Windows 8.1	4.0	Yes	No
6.2	Windows 8	4.0	Yes	Yes
6.1	Windows 7 SP1	3.0	Yes	Yes
6.1	Server 2008 R2	2.0	Yes	Yes
6.0	Server 2008	1.0	Yes	Yes

Table 1 – Requirements per Operating System and PowerShell Version

DSC can also be integrated with System Center Operations Manager (SCOM) to receive configuration change alerts in order to validate critical nodes against an approved baseline.

2.1 DSC Is Built on PowerShell 4.0

Windows PowerShell is a task based command line shell and configuration management framework built on the Microsoft® .NET framework. DSC is essentially an extension of the PowerShell language and provides a declarative syntax to express a configuration for information systems. Declarative means when an administrator is writing a script using PowerShell they do not necessarily have to know how DSC will provide a specific feature or software installation because the declarative syntax is more like an INI type expression specifying what should be present on the node (Jones, & Siddaway, & Hicks, 2014). A person with basic PowerShell skills can understand the declarative syntax used in a DSC configuration script.

MakePullServer.ps1

```
configuration MakePullServer
{
  param
  (
    [string[]]$ComputerName = 'localhost'
  )

  Import-DSCResource -ModuleName xPSDesiredStateConfiguration

  Node $ComputerName
  {
    WindowsFeature DSCServiceFeature
    {
      Ensure = "Present"
      Name   = "DSC-Service"
    }

    xDscWebService PSDSCPullServer
    {
      Ensure         = "Present"
      EndpointName       = "PSDSCPullServer"
      Port           = 443
      PhysicalPath       = "$env:SystemDrive\inetpub\wwwroot\PSDSCPullServer"
      CertificateThumbPrint   = "7F96DB10512C97C25BE08C21DE8E574156360484"
      ModulePath         = "$env:PROGRAMFILES\WindowsPowerShell\Modules"
      ConfigurationPath      = "$env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration"
      State          = "Started"
      DependsOn          = "[WindowsFeature]DSCServiceFeature"
    }

    xDscWebService PSDSCComplianceServer
    {
      Ensure         = "Present"
      EndpointName       = "Compliance-Server"
      Port           = 9080
      PhysicalPath       = "$env:SystemDrive\inetpub\wwwroot\PSDSCComplianceServer"
      CertificateThumbPrint   = "AllowUnencryptedTraffic"
      State          = "Started"
      IsComplianceServer     = $true
      DependsOn          = ("[WindowsFeature]DSCServiceFeature","[xDSCWebService]PSDSCComplianceServer")
    }

  }
}
MakePullServer -ComputerName pull-server.dev.dsc
```

Figure 1 - Sample Code of Declarative Syntax

PowerShell 4.0 introduced a new scripting keyword named "`configuration`". This keyword enables the declaration of resources with an additional new dynamic keyword named "`node`". Other new commands introduced are as follows in Table 2.

Command Modules	Description
`Get-DscResource`	Gets desired state configuration resources present on the computer.
`Start-DscConfiguration`	Applies a configuration to a node.
`Stop-DscConfiguration`	Stops a configuration currently running a configuration job.
`Get-DscConfiguration`	Gets the current configuration of the node.
`Test-DscConfiguration`	Tests whether an actual configuration on a node matches the desired configuration.
`Restore-DscConfiguration`	Restores the previous configuration for a node.
`Update-DscConfiguration`	Runs the existing configuration on the computer.
`Get-DscLocalConfigurationManager`	Gets the local configuration manager (LCM) setting for a node.
`Set-DscLocalConfigurationManager`	Applies LCM settings to a node.
`New-DscCheckSum`	Creates checksum files for DSC documents and DSC resources.

Table 2 – DSC Commands Also Called DSC Cmdlets

brian@brianequick.com

2.1.1 DSC Has Cross-Platform Compatibility Standards

PowerShell is designed to be an object based scripting language and makes DSC possible. DSC is built on the Common Information Model (CIM) standard developed by the Desktop Management Task Force (DMTF) and provides cross platform compatibility with its language used to define managed elements in the Managed Object Format (MOF). DSC uses Windows Remote Management (WinRM) technology as a communication mechanism. WinRM is the Microsoft® implementation of web services for management called WS-Management (WSMan) (Chaganti, 2014). The MOF is the primary configuration language in defining how specific nodes should be configured. MOF files can be created by PowerShell to describe the classes and instance definitions of a configuration in textual form. MOF files can also be created by third party tools like Puppet or Chef, and DSC is capable of applying them. MOF files are used by the Local Configuration Manager (LCM) to enforce a precise configuration for each unique node whether the operating system is Windows or Linux as seen in Figure 2 (Greene, 2014).

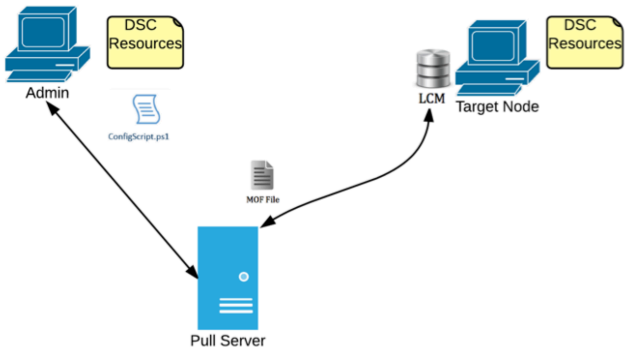

Figure 2 – DSC Configuration Concept Diagram

The LCM is the engine or agent of DSC and is installed when PowerShell 4.0 is installed. How this configuration data is communicated to nodes is explained next.

2.1.2 DSC Has Flexible Modes of Operation

The architecture of DSC can be described as a push or pull mode of operation. Push mode is best described as DSC being initiated manually from a server and the configuration data being pushed out to connected nodes. This paper will show an instance where a MOF file is pushed to another server. In contrast, the pull mode is described as

each node requesting its specific MOF configuration file from the pull server at a pre-defined refresh frequency in minutes. Communication between nodes can be configured using Server Message Block (SMB), as in a common file share or using WSMan. In the interest of security it is recommended as a best practice to configure WinRM communications using HTTPS with a Secure Socket Layer (SSL) certificate when using DSC in a production environment. PowerShell has built-in command modules called "cmdlets" that make it easy to check, validate and configure the necessary WinRM listeners enabling secure communication. Figure 3 provides a conceptual depiction of DSC components in pull mode.

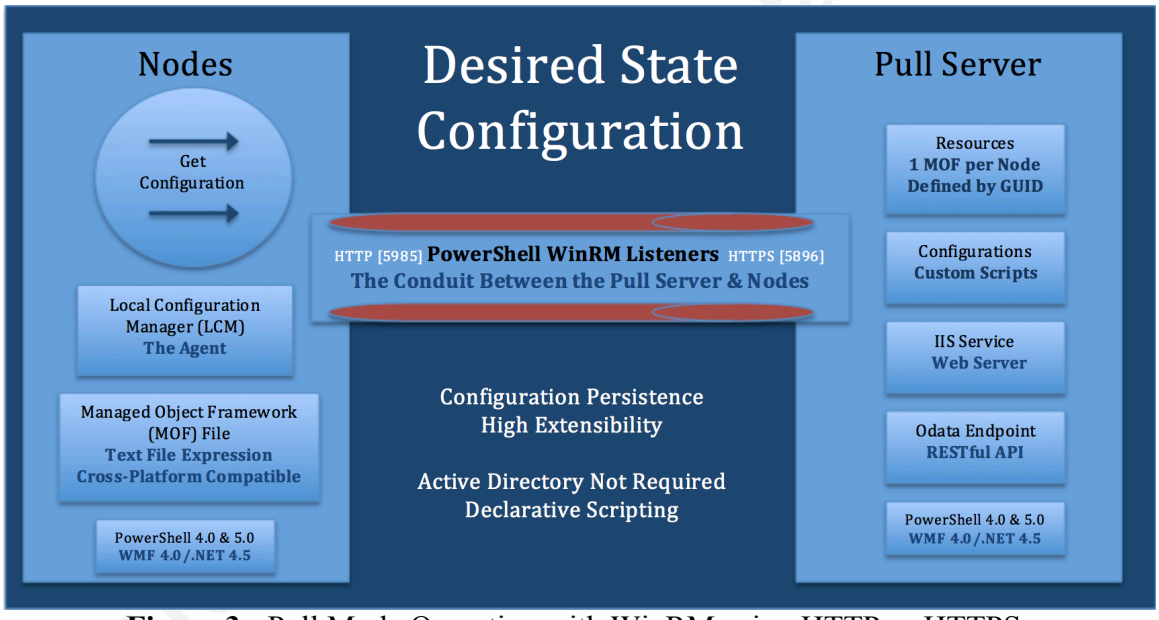

Figure 3 - Pull Mode Operation with WinRM using HTTP or HTTPS

Each node participating in DSC registers itself with the pull server using a global unique ID (GUID). This GUID is sensitive information that correlates to a specific MOF file designed uniquely for a specific node. The primary advantage of implementing DSC in pull mode is scalability. A single pull server can provide DSC configurations to many connected nodes with the additional benefit of specifying how often the LCM on each node should check back with the pull server enforcing a configuration. Configuration management procedures may dictate that general servers only need to have configuration drift checked once every 48 hours, but every 15 minutes for critical servers that host sensitive data where system changes could result in serious losses to the organization.

Once the organization has agreed upon a planned baseline system, administrators and developers can begin creating DSC scripts with resources necessary to deploy an approved baseline.

2.2 DSC Resources Offer Extensibility

2.2.1 Built-in DSC Resources

DSC comes with built-in resources also called resource providers, which are the building blocks required to write configuration scripts and deploy configuration management solutions. Twelve DSC resources are immediately available upon installation of the WMF 4.0. Some of these built-in resources are `"Archive", "Environment", "File", "Group", "Log", "Registry" and "WindowsFeature"`. These familiar names provide mechanisms to manage what each title implies. For example, an administrator can use the `"WindowsFeature"` resource to make certain that the IIS and ASP 4.5 roles are installed for multiple nodes. Administrators can open a PowerShell command prompt and type in **`PS C:\> Get-WindowsFeature`** to see the very same roles or features available in Server Manager that can be installed using DSC. The syntax allows administrators to specify this in a configuration script by setting the `"Ensure"` property equal to the value of `"Present"` demonstrating this easy to use declarative syntax in DSC. If administrators planned to install all the sub features for the "`WindowsFeature`" resource, they could simply insert the line `"IncludeAllSubFeature = $true"` under the `"Ensure"` property. This configuration script would create a MOF file used to enforce the configuration for the node or nodes specified after the `"Node"` element. A small sample configuration script is shown in Figure 4.

```
1   Configuration  IISWebservers
2   {
3       Node IIS-Area3
4           {
5             WindowsFeature IISA3
6             {
7                 Ensure = "Present"
8                 Name   = "Web-Server"
9                 IncludeAllSubFeature = $true
10            }
11
12          }
13  }
14
15    IISWebservers "."
```

Figure 4 – Code Snippet using the new Configuration Keyword

DSC resources that come with WMF 4.0, when installed on a given machine can be displayed by using the ***PS C:\> Get-DscResource*** cmdlet. The output below in Figure 5 shows the twelve built-in resources explained with the properties available with each named resource.

```
PS C:\> get-dscresource

ImplementedAs   Name                 Module                         Properties
-------------   ----                 ------                         ----------
Binary          File                                                {DestinationPath, Attributes, Checksum, Con...
PowerShell      Archive              PSDesiredStateConfiguration    {Destination, Path, Checksum, Credential...}
PowerShell      Environment          PSDesiredStateConfiguration    {Name, DependsOn, Ensure, Path...}
PowerShell      Group                PSDesiredStateConfiguration    {GroupName, Credential, DependsOn, Descript...
Binary          Log                  PSDesiredStateConfiguration    {Message, DependsOn}
PowerShell      Package              PSDesiredStateConfiguration    {Name, Path, ProductId, Arguments...}
PowerShell      Registry             PSDesiredStateConfiguration    {Key, ValueName, DependsOn, Ensure...}
PowerShell      Script               PSDesiredStateConfiguration    {GetScript, SetScript, TestScript, Credenti...
PowerShell      Service              PSDesiredStateConfiguration    {Name, BuiltInAccount, Credential, DependsO...
PowerShell      User                 PSDesiredStateConfiguration    {UserName, DependsOn, Description, Disabled...
PowerShell      WindowsFeature       PSDesiredStateConfiguration    {Name, Credential, DependsOn, Ensure...}
PowerShell      WindowsProcess       PSDesiredStateConfiguration    {Arguments, Path, Credential, DependsOn...}
```

Figure 5 – Example List of Get-DscResource Output

Once administrators identify a resource they want to utilize in their configuration script, they can further expand and analyze all the properties available. A compound command, like **`PS C:\> Get-DscResource WindowsFeature,`** can be used to obtain more specific property information available for each resource. Specific property and value information is shown below in Figure 6 for the `"WindowsFeature"`.

Properties for all DSC resources such as `"WindowsFeature",` are available, making it very simple to declare what features an administrator or developer may want installed on each node with specific help information available in PowerShell.

```
PS C:\> Get-DscResource WindowsFeature –Syntax
```

```
PS C:\Users\brian> get-dscresource Windowsfeature -syntax
WindowsFeature [string] #ResourceName
{
    Name = [string]
    [ Credential = [PSCredential] ]
    [ DependsOn = [string[]] ]
    [ Ensure = [string] { Absent | Present }  ]
    [ IncludeAllSubFeature = [bool] ]
    [ LogPath = [string] ]
    [ Source = [string] ]
}
```

Figure 6 – Properties for the WindowsFeature Resource

These properties can be shown for each DSC resource aiding developers in creating configuration scripts using the built-in resources provided by PowerShell 4.0. What if administrators need more resources or a unique capability that is not yet available in the built-in resources that came with PowerShell 4.0?

2.2.2 Experimental DSC Resources

Since the 2013 release of WMF 4.0, Microsoft® and the development community have collaborated to create many new DSC resource providers that are being released in waves. Wave 10 is currently available with over fifty resources for use to create DSC scripts for configuration management and many other deployment solutions (Microsoft, 2015).

The DSC Resource Kit can be obtained and downloaded here:

https://gallery.technet.microsoft.com/scriptcenter/DSC-Resource-Kit-All-c449312d

Microsoft has released DSC resources as beta versions; and although these resources may not be fully supported by a Microsoft standard support program, organizations, such as Amazon Web Services (AWS) and Rackspace, are using DSC because it is a powerful tool. Amazon uses DSC to deploy IT infrastructure services with predefined configurations, and Rackspace uses DSC to maintain and manage applications based on customer defined requirements (Barr, 2014). The list of DSC resources continues to grow due to the devops community discovering the benefits of DSC and making contributions of their time and talent to develop new useful resources. There are an estimated two-

hundred DSC resources when including the WMF 5.0 preview found at https://www.powershellgallery.com with many seen below in Figure 7.

Figure 7 - Built-in DSC Resources with Experimental Resources Included

2.2.3 Creating New DSC Resources

If currently released resources do not meet the needs of an organization, Microsoft® has made it possible for any developer to create new DSC resources. (Murawski, 2014) Developers can create resources with the three mandatory functions named `"Get-TargetResource,"` `"Set-TargetResource"` and `"Test-TargetResource"` that enable custom defined properties to be applied. Explaining how to author new custom resources is outside the scope of this paper; however, an excellent article written by Ritesh Modi can help explain in greater detail how to author your own DSC custom resources (Modi, 2015).

3. Deploying DSC in Pull Server Mode

3.1 Setting Up a Pull Server

A DSC pull server can be setup in the following steps:

- Setup three servers with Windows Server 2012 R2 fully updated.
- Use the MakePullServer.ps1 script in this paper to create a MOF file.
- Use the pull server MOF file to push the configuration.
- Obtain certificates for client/server authentication if using HTTPS.

A rudimentary scenario is used in this paper to explain how to create a simple DSC pull server with a MOF file. The purpose of the DSC pull server is to help keep a WSUS server consistent with an approved baseline documented in a signed system security plan. The Server 2012 R2 operating system is utilized as three domain joined servers. The IIS role and DSC service could be installed with the Add Roles and Features wizard built into Server 2012 R2, but DSC will install most of the configurations needed for the new pull server. This paper demonstrates implementing a basic pull server with three PowerShell scripts. The first script to create the MOF file for the pull server; the second script will create a MOF file for a WSUS server, and the third will be used to configure the LCM on the WSUS server to make it a pull client. Server 2012 R2 will need additional resources from the Wave 10 release. The administrator should place these resources in the `modules` directory at this path `"C:\Program Files\WindowsPowerShell\Modules"` on all the servers. These new resources can be seen by typing "`Get-DscResource`". The "`DSCServiceFeature`" is a mandatory feature declared in the script along with "`xPSDesiredStateConfiguration`" and "`xDSCWebService`". The administrator will need another domain-connected server with the hostname of "`server2012r2`" as seen on line 5 and line 52 of Figure 9. The script should be executed in a PowerShell console using "Run as Administrator".

```
PS C:\MakePullServer> .\MakePullServer.ps1
```

The MakePullServer.ps1 (Figure 9) script creates a MOF file. The instance definitions created by this script can be seen in the following MOF file snippet in Figure 8.

```
/*
@TargetNode='server2012r2'
@GeneratedBy=brian.quick.adm
@GenerationDate=07/16/2015 18:32:57
@GenerationHost=PULL-SERVER
*/

instance of MSFT_RoleResource as $MSFT_RoleResource1ref
{
ResourceID = "[WindowsFeature]DSCServiceFeature";
 Ensure = "Present";
 SourceInfo = "C:\\Users\\brian.quick.adm\\Documents\\MakePullServer\\MakePullServer.ps1::12::5::WindowsFeature";
 Name = "DSC-Service";
 ModuleName = "PSDesiredStateConfiguration";
 ModuleVersion = "1.0";

};

instance of MSFT_RoleResource as $MSFT_RoleResource2ref
{
ResourceID = "[WindowsFeature]WebWindowsAuth";
 DependsOn = {
    "[WindowsFeature]DSCServiceFeature"
};
 Ensure = "Present";
 SourceInfo = "C:\\Users\\brian.quick.adm\\Documents\\MakePullServer\\MakePullServer.ps1::18::5::WindowsFeature";
 Name = "web-windows-auth";
 ModuleName = "PSDesiredStateConfiguration";
 ModuleVersion = "1.0";
```

Figure 8 - Snippet of Instance Definitions From a MOF file

MakePullServer.ps1

```
configuration MakePullServer
{
  param
  (
    [string[]]$ComputerName = 'server2012r2'
  )

  Import-DSCResource -ModuleName xPSDesiredStateConfiguration

  Node $ComputerName
  {
    WindowsFeature DSCServiceFeature
    {
      Ensure = "Present"
      Name   = "DSC-Service"
    }

    WindowsFeature WebWindowsAuth  {
      Ensure = "Present"
      Name = "web-windows-auth"
      DependsOn = "[WindowsFeature]DSCServiceFeature"
    }

    xDscWebService PSDSCPullServer
    {
      Ensure          = "Present"
      EndpointName    = "PSDSCPullServer"
      Port            = 8080
      PhysicalPath    = "$env:SystemDrive\inetpub\wwwroot\PSDSCPullServer"
      CertificateThumbPrint  = "AllowUnencryptedTraffic"
      ModulePath      = "$env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"
      ConfigurationPath     = "$env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration"
      State           = "Started"

    }

    xDscWebService PSDSCComplianceServer
    {
      Ensure          = "Present"
      EndpointName    = "PSDSCComplianceServer"
      Port            = 9080
      PhysicalPath    = "$env:SystemDrive\inetpub\wwwroot\PSDSCComplianceServer"
      CertificateThumbPrint  = "AllowUnencryptedTraffic"
      State           = "Started"
      IsComplianceServer    = $true

    }

  }
}
MakePullServer -ComputerName server2012r2
```

Figure 9 - Configuration Script used by PowerShell to Create a Pull Server MOF file.

The "`Start-DSCConfiguration`" cmdlet can be used to invoke a CIM session and push the MOF file to "`server2012r2`" as seen in Figure 10 below (Hicks, 2015).

hash.ps1*

```
$MakePullServer = @{
Path = ".\MakePullServer"
ComputerName = "server2012r2"
wait = $True
verbose = $True
Force = $True
}

Start-DscConfiguration @MakePullServer
```

Figure 10 - A Simple Hash Table can now be used to Apply the MOF file to the Node

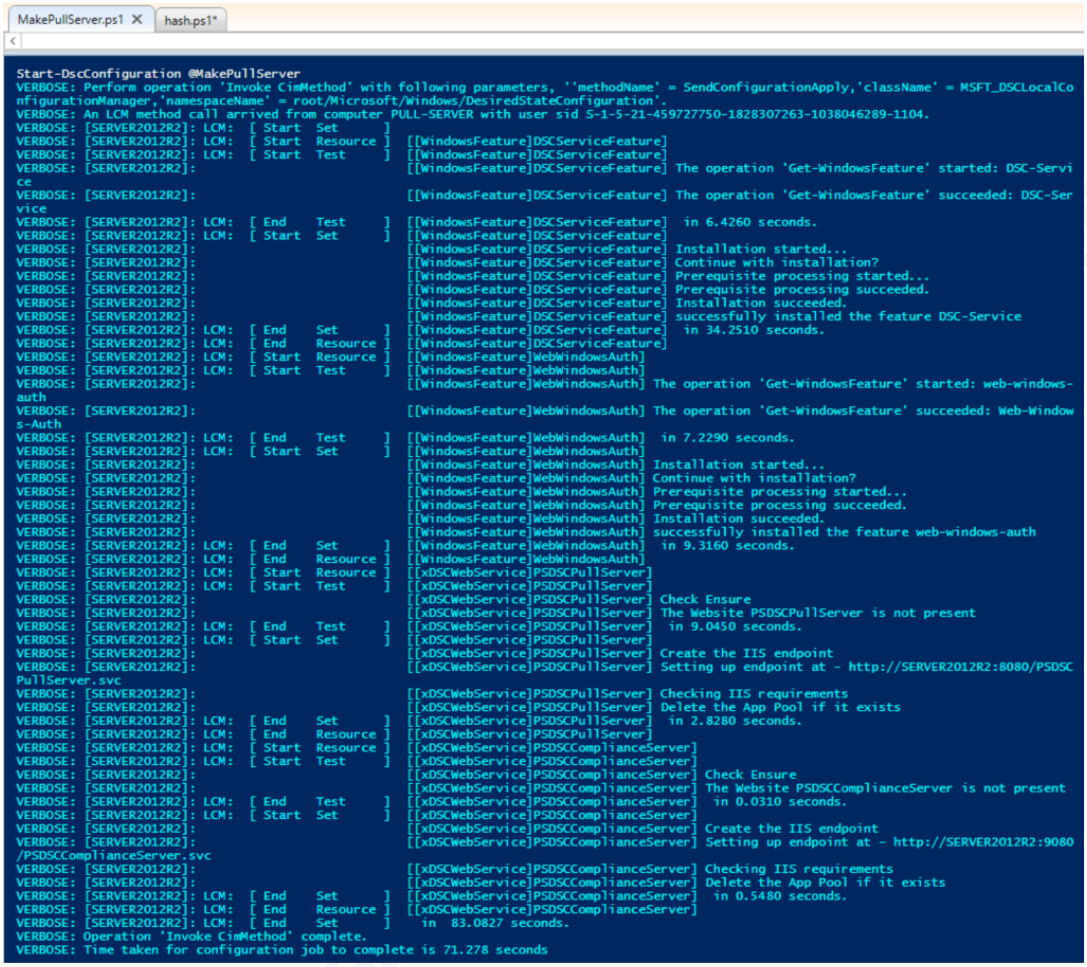

Figure 11 - Verbose Output as the Configuration is Applied to "server2012r2"

Figure 11 reveals verbose information as the push operation applies the configuration to "Server2012r2". "Server2012r2" should be renamed to "pullserver", and the administrator should now validate that the web services are functioning properly to operate in pull mode. Administrators first need a certificate that will provide server authentication. The certificate must be bound to the web service on the pull server. The administrator may wish to install and utilize "IIS Manager" using the Server certificates feature, add it there with “complete certificate request” and, when prompted, browse to the certificate file. When the MOF file configuration was pushed to the new pull server the following tasks were accomplished on the "pull server":

- Created a directory at "`c:\inetpub\wwwroot\PSDSCPullServer`"

- Copied five files from "`$pshome/Modules/PSDesiredStateConfiguration/PullServer`" Global.asax, PSDSCPullServer.mof, PSDSCPullServer.svc, PSDSCPullServer.xml and PSDSCPullServer.config to the following path "`c:\inetpub\wwwroot\PSDSCPullServer`"
- Renamed PSDSCPullServer.config to web.config.
- Created a new directory named "`c:\inetpub\wwwroot\bin`".
- Copied Devices.mdb at "`$pshome\modules\psdesiredstateconfiguration\pullserver\Devices.mdb`" and place it in "`$env:programfiles\WindowsPowerShell\DscService\Devices.mdb`"

Using the IIS web server manager, the administrator should verify that a new application pool named "PSWS" is running under the local system account. The final step in configuring web services is to add a database provider to the web.config configuration file by adding keys as seen in Figure 12 to the "appsettings" section of the web.config file at "`C:\inetpub\wwwroot\PSDSCPullServer\`" (2013, Murawski).

```
<add key="dbprovider" value="System.Data.OleDb" />
<add key="dbconnectionstr" value="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\Program Files\WindowsPowerShell\DscService\Devices.mdb;" />
<add key="ConfigurationPath" value="C:\Program Files\WindowsPowerShell\DscService\Configuration" />
<add key="ModulePath" value="C:\Program Files\WindowsPowerShell\DscService\Modules" />
```

Figure 12 - Database Provider Configuration

Finally, the administrator can verify that the pull server service is running by navigating to the "`PSDSCPullServer.svc`" service using a web browser on the pull server, as seen in Figure 13.

Figure 13 - Verification that the New Pull Server is Functioning

The second configuration script is designed to generate a MOF file for the WSUS server. The "`WindowsFeature`" and "`xFirewall`" resource providers are used in this example scenario to install Windows Update Services and a firewall exception adhering to a simple baseline as seen in Figure 14.

WSUS.ps1

```
1   configuration WSUS
2   {
3       param
4       (
5       [Parameter(Mandatory)]
6       $MachineName
7       )
8       Import-DscResource -ModuleName xNetworking
9
10      Node $MachineName
11      {
12          WindowsFeature UpdateServices
13          {
14                Ensure = "Present"
15                Name = "UpdateServices"
16                                    | }
17
18
19          xFirewall AllowWSUSRemoteDesktop
20          {
21              #Create a custom firewall rule.
22              Name = 'WSUS - Remote Desktop Connections'
23              DisplayGroup = "Remote Desktop"
24              Ensure = 'Present'
25              State = 'Enabled'
26              Access = "Allow"
27              Profile = ("Domain", "Private", "Public")
28          }
29
30       }
31  }
32  WSUS -MachineName WSUS01
```

Figure 14 - Script to Create the WSUS01 MOF File

The WSUS.ps1 script is also executed using "Run as Administrator" credentials.

```
PS C:\WSUS\> .\WSUS.ps1
```

The important difference in using pull mode is that the target nodes are identified by a Global Unique ID (GUID) rather than by a name. This method ensures that each target node gets the proper MOF file created for a specific node configuration. The "`New-CheckSum`" cmdlet is also used to generate a checksum of each MOF file to help protect the integrity of the MOF files on the pull server (Technet, 2013). A code sample is provided in Figure 15 providing a way to create these methods.

```
$Guid = [guid]::NewGuid()
$source = "WSUS01.mof"

$target = "\\pull-server\c$\program files\windowspowershell\dscservice\configuration\$Guid.mof"
copy $source $target

New-DSCChecksum $target
```

Figure 15 - GUID Creation

3.2 Setting Up Nodes to Communicate with the Pull Server

Since clients must be able to receive configuration data from the pull server administrators must validate that WinRM listeners are functioning properly for each node. This validation can be done with "`Test-WSMan`" and then enabling the HTTP or HTTPS listener as needed with the "`Set-WSManQuickConfig`" cmdlet (Hicks, 2013). A certificate must be added to the local machine store when using HTTPS. The LCM on "WSUS01" must be configured for pull mode, and the LCM.ps1 script as seen in Figure 16 will generate a special "meta.mof" file used for this purpose.

LCM.ps1

```
Configuration PullWSUSPull  {
    Node WSUS01  {
        LocalConfigurationManager  {
            ConfigurationID = "d5bc808a-7f73-4a63-8d99-b688d9a19cc4"
            ConfigurationMode = "ApplyAndAutoCorrect"
            RefreshMode = "Pull"
            ConfigurationModeFrequencyMins = "15"
            DownloadManagerName = "WebDownloadManager"
            DownloadManagerCustomData = @{
                ServerUrl = "http://PULLSERVER:8080/PSDSCPullServer.svc";
                AllowUnSecureConnection = "True"}

        }
    }

}

PullWSUSPull
```

Figure 16 - LCM.ps1 to Create LCM Meta MOF File for WSUS01

The "`Set-DscLocalConfigurationManager`" cmdlet can now be used to configure the LCM on "WSUS01" to use pull server mode and shown in Figure 17.

```
PS C:\Users\brian.quick.adm\Documents> Set-DscLocalConfigurationManager -ComputerName WSUS01 -path .\ -verbose
VERBOSE: Performing the operation "Start-DscConfiguration: SendMetaConfigurationApply" on target "MSFT_DSCLocalConfigurationMana
ger".
VERBOSE: Perform operation 'Invoke CimMethod' with following parameters, ''methodName' = SendMetaConfigurationApply,'className'
= MSFT_DSCLocalConfigurationManager,'namespaceName' = root/Microsoft/Windows/DesiredStateConfiguration'.
VERBOSE: An LCM method call arrived from computer PULLSERVER with user sid S-1-5-21-459727750-1828307263-1038046289-1104.
VERBOSE: [WSUS01]: LCM:  [ Start  Set      ]
VERBOSE: [WSUS01]: LCM:  [ Start  Resource ]  [MSFT_DSCMetaConfiguration]
VERBOSE: [WSUS01]: LCM:  [ Start  Set      ]  [MSFT_DSCMetaConfiguration]
VERBOSE: [WSUS01]: LCM:  [ End    Set      ]  [MSFT_DSCMetaConfiguration]  in 0.0330 seconds.
VERBOSE: [WSUS01]: LCM:  [ End    Resource ]  [MSFT_DSCMetaConfiguration]
VERBOSE: [WSUS01]: LCM:  [ End    Set      ]    in  0.1423 seconds.
VERBOSE: Operation 'Invoke CimMethod' complete.
VERBOSE: Set-DscLocalConfigurationManager finished in 0.242 seconds.
```

Figure 17 - "`Set-DscLocalConfigurationManager`" to Apply the Meta MOF

The default mode of operation in DSC is push mode; so after applying the LCM meta configuration, the new mode of operation on "WSUS01" should be pull mode as seen in Figure 18 by using "`Get-DscLocalConfigurationManager`". The new LCM Meta-Configuration shows "Pull" after the "RefreshMode" setting.

```
PS C:\> hostname
WSUS01
PS C:\> Get-DscLocalConfigurationManager


ActionAfterReboot              : ContinueConfiguration
AllowModuleOverwrite           : False
CertificateID                  :
ConfigurationID                : d5bc808a-7f73-4a63-8d99-b688d9a19cc4
ConfigurationMode              : ApplyAndAutoCorrect
ConfigurationModeFrequencyMins : 15
Credential                     :
DebugMode                      : {NONE}
DownloadManagerCustomData      : {MSFT_KeyValuePair (key = "ServerUrl"), MSFT_KeyValuePair (key =
                                 "AllowUnSecureConnection")}
DownloadManagerName            : WebDownloadManager
LCMCompatibleVersions          : {1.0}
LCMState                       : Idle
LCMVersion                     : 1.0
RebootNodeIfNeeded             : False
RefreshFrequencyMins           : 30
RefreshMode                    : Pull
PSComputerName                 :
```

Figure 18 - Local Configuration Manager Status on "WSUS01"

As the engine of DSC, the LCM will now check back with the pull server every 15 minutes as seen after the "ConfigurationModeFrequencyMins" setting. Configuration drift for "WSUS01" will now be corrected with the "ConfigurationMode" setting being defined as "ApplyAndAutoCorrect". How to troubleshoot LCM issues or communication problems with DSC logs is next.

3.3 Troubleshooting DSC with Logs

DSC records errors and events like most Windows machines in logs that can be viewed in "Event Viewer". These DSC logs are found under "Application and Service logs.", "Microsoft", "Windows", and then "Desired State Configuration". Writing configuration scripts can be challenging for beginners, and having logs will make problem solving easier if issues arise. DSC created three primary logs: Operational, Analytic, and Debug logs. Operational logs are turned on by default, but Analytic and Debug logging must be enabled in order to be utilized for troubleshooting. The wevtutil utility can be used to enable these logs (Technet, 2014).

```
PS C:\Users> wevtutil.exe set-log “Microsoft-Windows-Dsc/Analytic” /q:true /e:true

PS C:\Users> wevtutil.exe set-log “Microsoft-Windows-Dsc/Debug” /q:true /e:true
```

DSC also has two experimental resources that will help administrators analyze DSC logs; the "`xDscDiagnostics`" and "`Get-xDscOperation`". These resources have functions that help identify all local events from past DSC operations or DSC events on remote nodes and operations running on one or more nodes (Technet, 2014).

4. Measuring System Changes with Compliance Server

A formal configuration management plan seldom works well enough that all changes are accounted for in a timely and accurate way as to provide a measure of those changes against the approved baseline. Although relatively new, DSC offers an additional web service along with the pull server service "PSDSCPullServer.svc" called the compliance server "DSCComplianceServer.svc" service. The compliance server web service was created for the purpose of measuring the status of each node connected to a pull server. The pull operational status, configuration and node information is all stored in the database configured previously (Figure 12) and can be used by administrators to periodically check the status of the nodes to see if their configurations are in sync with the pull server or not. This node status querying capability can be enhanced with a lightweight data-interchange format called JavaScript Object Notation (JSON) to output the information to any website. The type query information that can be obtained from connected nodes is listed below.

- **NodeCompliant** - Information on the compliance of each node or nodes.
- **ServerCheckSum** - The checksum of the MOF on the pull server.
- **TargetCheckSum** - The checksum of the MOF on the target node.
- **LastComplianceTime** - The last successful node configuration.
- **LastHeartbeatTime** - The last successful node connection (Technet, 2013).

5. Conclusion

Configuration management for IT systems has always been a very challenging endeavor. Tracking and accounting for system changes is a daunting task but

PowerShell driven DSC provides a new and promising built-in resource that any organization using a Windows network infrastructure can utilize to monitor, control and report compliance. Uncontrolled changes to information systems introduces serious threats that often go undiscovered until more serious consequences occur. DSC brings developers and PowerShell savvy administrators a new capability making it even easier to automate change control and reporting.

brian@brianequick.com

References

Algosec. (2013). *The State of Network Security 2013*. Retrieved May 25, 2015, from http://www.algosec.com/resources/files/surveys/140406_algosec_state_of_network_security_2014.pdf

Barr, J. (2014, September 29). Windows PowerShell Desired State Configuration (DSC) on Amazon Web Services Website. Retrieved from https://aws.amazon.com/blogs/aws/ powershell-dsc-setup-quick-ref/

Chaganti, R. (2014). *Windows PowerShell Desired State Configuration Revealed*. New York: Apress.

Greene, M. (2014, May 19). *PowerShell DSC for Linux*, Step by Step - Building Clouds Blog - Site Home - TechNet Blogs. Retrieved from http://blogs.technet.com/b/privatecloud/archive/2014/05/19/powershell-dsc-for-linux-step-by-step.aspx

Hicks, J. (2015, March 13). Pluralsight. Retrieved from http://www.pluralsight.com/courses/advanced-powershell-dsc

Hicks, J. (2013). *PowerShell deep dives*. Shelter Island, NY: Manning.

Jones, D., Siddaway, R., & Hicks, J. (2013). *PowerShell in depth: An administrator's guide*. Shelter Island, NY: Manning Publications.

Koch, J. (2015). *Getting Started with PowerShell Desired State Configuration (DSC) Channel 9*. Retrieved May 24, 2015, from http://channel9.msdn.com/Series/Getting-Started-with-PowerShell-Desired-State-Configuration-DSC

Lowe, S. (2014, July 15). *Stateful Configuration with PowerShell Desired State Configuratio*n. Retrieved May 24, 2015, from http://www.windowsnetworking.com/articles-tutorials/netgeneral/stateful-configuration-powershell-desired-state-configuration.html

Microsoft. (2014, May 15). Desired State Configuration Pull Model Protocol. Retrieved from http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-DSCPM%5D.pdf

brian@brianequick.com

Microsoft. (2015, April 1). TechNet DSC Resource Kit (All Modules) Wave 10. Retrieved from https://gallery.technet.Microsoft.com/DSC-Resource-Kit-All-c449312d

Modi, Ritesh. *Authoring Desired State Configuration Custom Resources*. *Msdn* 30.4 (2015): 62-67. Web. 24 May 2015.

Murawski, S. (2013, October 3). Building a Desired State Configuration Pull Server » PowerShell.org. Retrieved from http://powershell.org/wp/2013/10/03/building-a-desired-state-configuration-pull-server/

Murawski, S. (2014). *Building Desired State Configuration Custom Resources*. Retrieved May 24, 2015, from http://powershell.org/wp/2014/03/13/building-desired-state-configuration-custom-resources/

Technet. (2013, June 24). Using the DSC Resource Designer tool. Retrieved June 6, 2015, from https://technet.microsoft.com/en-us/%5Clibrary/Dn249913.aspx

Technet. (2014, December 3). Set-WSManQuickConfig. Retrieved from https://technet.microsoft.com/en-us/library/hh849867(v=wps.630).aspx

Technet DSC Resource Kit. (2015, April 1). Retrieved from https://gallery.technet.microsoft.com/scriptcenter/DSC-Resource-Kit-All-c449312d

brian@brianequick.com