PowerShell Desired State Configuration for Linux
Release Notes

Version 1.1.0

Table of Contents

PowerShell Desired State Configuration for Linux Release Notes 1

Release Overview 1

Supported Linux operation system versions 1

Upgrading DSC for Linux from a Prior Version 2

New Scenarios Enabled in this Release 2

Issues Resolved in this Release 2

Known Limitations 2

New Scenarios in this Release 2

Separation of node and configuration IDs 2

Use Azure Automation as a DSC Pull Server 3

Additional Information 5

Performing DSC Operations from the Linux computer 5

Using PowerShell Desired State Configuration for Linux with a Pull Server 6

PowerShell Desired State Configuration for Linux Log Files 6

Release Overview

Supported Linux operation system versions

The following Linux operating system versions are supported for DSC for Linux.

- CentOS 5, 6, and 7 (x86/x64)
- Debian GNU/Linux 6, 7 and 8 (x86/x64)
- Oracle Linux 5, 6 and 7 (x86/x64)
- Red Hat Enterprise Linux Server 5, 6 and 7 (x86/x64)
- SUSE Linux Enterprise Server 10, 11 and 12 (x86/x64)
- Ubuntu Server 12.04 LTS and 14.04 LTS (x86/x64)

The following table describes the required package dependencies for DSC for Linux.

Required package	Description	Minumum version
glibc	GNU Library	2.4 – 31.30

python	Python	2.4 – 3.4
Omiserver*	Open Management Infrastructure	1.0.8-2
openssl*	OpenSSL Libraries	0.9.8e or 1.0
ctypes	Python CTypes library	Must match Python version
libcurl	cURL http client library	7.15.1

** These requirements have changed with this version of PowerShell Desired State Configuration for Linux.*

Upgrading DSC for Linux from a Prior Version

Upgrading PowerShell Desired State Configuration for Linux from version 1.0 to this version (1.1) is not supported. If you have version 1.0 currently installed, remove it prior to installing version 1.1 with: **`rpm -e dsc`** or **`dpkg -r dsc`**

New Scenarios Enabled in this Release

- **Separation of node and configuration IDs**
- **Use Azure Automation as a DSC Pull Server**

Issues Resolved in this Release

- **Restore-DSCConfiguration** does not restore the prior configuration
- **nxPackage/nxScript**: unreliable behavior is observed when the system locale is not UTF-8.
- **nxPackage**: On Ubuntu and Debian, packages that prompt for user input cause, such as MySQL server, prevent successful configuration
- **nxService**: in some cases, the running state of services is incorrectly detected
- **nxPackage**: when Ensure = Absent, installed packages fail to remove in some cases

Known Limitations

- In this release of PowerShell Desired State Configuration for Linux, only Pull Servers based on WMF 5.0 April Preview or later are supported. To use a prior version of the Pull Server, continue to use Desired State Configuration for Linux 1.0.
    - Partial Configurations are not supported in this release.

New Scenarios in this Release

Separation of node and configuration IDs

DSC currently uses a configuration ID to uniquely identify a single configuration for a single node. This feature separates the configuration ID into two distinct identifiers: Configuration Name and Agent ID. Configuration Name identifies the configuration for a computer; this ID can be shared by multiple nodes. Agent ID uniquely identifies a node; this ID must be unique for every node.

Metaconfig Updates for Seperation of Computer and Configuration IDs

Because the ConfigurationNames are no longer GUIDs (they are now friendly names), anyone can determine them. To mitigate this issue, we added an extra level of security by adding a registration step before a node can start requesting configurations from a server. A node registers itself with the pull server with a shared secret (which the node and the server both know already), and the name of the configuration it will request. This shared secret need not be unique for each computer. Assumption: the shared secret is a hard-to-guess identifier, like a GUID. We call this shared secret RegistrationKey in the metaconfig.

```
[DscLocalConfigurationManager()]
Configuration SampleLinuxMetaConfig
{
Node “mylinuxserver”{
    Settings
        {
            RefreshFrequencyMins = 30;
            RefreshMode = "PULL";
            ConfigurationMode =”ApplyAndMonitor“;
            AllowModuleOverwrite  = $true;
            RebootNodeIfNeeded = $true;
            ConfigurationModeFrequencyMins = 60;
        }

        ConfigurationRepositoryWeb ConfigurationManager
        {
            ServerURL = “https://PullServerMachine:8080/psdscpullserver.svc”
            RegistrationKey = "140a952b-b9d6-406b-b416-e0f759c9c0e4"
            ConfigurationNames = @(“MySQLRole”)
        }
      }
}
```

SampleMetaConfig

For more information on this feature and instructions for defining shared keys on the Pull Server, reference the WMF 5 Production Preview Release Notes.

Use Azure Automation as a DSC Pull Server

Note: For more information on Azure Automation’s DSC features, reference the documentation.

Linux computers can be onboarded to Azure Automation DSC, as long as they have outbound access to the internet, via a few simple steps:

1. Make sure version 1.1 or later of the DSC Linux agent is installed on the machines you want to onboard to Azure Automation DSC.

2. If the PowerShell DSC Local Configuration Manager defaults match your use case:

    - On each Linux machine to onboard to Azure Automation DSC, use Register.py to onboard using the PowerShell DSC Local Configuration Manager defaults:

      /opt/microsoft/dsc/Scripts/Register.py <Automation account registration key> <Automation account registration URL>

- To find the registration key and registration URL for your Automation account, see the Secure Registration section below.

Commented [JL1]: Todo, link

If the PowerShell DSC Local Configuration Manager defaults **do not** match your use case, follow steps 3 - 9. Otherwise, proceed directly to step 9.

3. Open the PowerShell console or PowerShell ISE as an administrator on a Windows machine in your local environment. This machine must have the latest version of WMF 5 installed.

4. Connect to Azure Resource Manager using the Azure PowerShell module:

   Add-AzureAccount

   Switch-AzureMode AzureResourceManager

5. Download, from the Automation account you want to onboard nodes to, the PowerShell DSC metaconfigurations for the machines you want to onboard:

   Get-AzureAutomationDscOnboardingMetaconfig -ResourceGroupName MyResourceGroup -AutomationAccountName MyAutomationAccount -ComputerName MyServer1, MyServer2 -OutputFolder C:\Users\joe\Desktop

6. Optionally, view and update the metaconfigurations in the output folder as needed to match the PowerShell DSC Local Configuration Manager fields and values you want, if the defaults do not match your use case.

7. Remotely apply the PowerShell DSC metaconfiguration to the machines you want to onboard:

```
$SecurePass = ConvertTo-SecureString -string "<root password>" -AsPlainText -Force

$Cred= New-Object System.Management.Automation.PSCredential "root", $SecurePass

$Opt = New-CimSessionOption -UseSsl:$true -SkipCACheck:$true -SkipCNCheck:$true -SkipRevocationCheck:$true

# need a CimSession for each Linux machine to onboard

$Session = New-CimSession -Credential:$Cred -ComputerName:<your Linux machine> -Port:5986 -Authentication:basic -SessionOption:$Opt

Set-DscLocalConfigurationManager -CimSession $Session –Path C:\Users\joe\Desktop\DscMetaConfigs
```

8. If you cannot apply the PowerShell DSC metaconfigurations remotely, for each Linux machine to onboard, copy the metaconfiguration corresponding to that machine from the folder in step 5 onto the Linux machine. Then call SetDscLocalConfigurationManager.py locally on each Linux machine to onboard to Azure Automation DSC:

   /opt/microsoft/dsc/Scripts/SetDscLocalConfigurationManager.py –configurationmof <path to metaconfiguration file>

9. Using the Azure portal or cmdlets, check that the machines to onboard now show up as DSC nodes registered in your Azure Automation account.

Additional Information

Performing DSC Operations from the Linux computer

DSC for Linux includes scripts to work with configuration from the local Linux computer. These scripts are located in /opt/microsoft/dsc/Scripts and include the following:

GetDscConfiguration.py

Returns the current configuration applied to the computer. Similar to the Windows PowerShell cmdlet Get-DscConfiguration cmdlet.

```
sudo ./GetConfiguration.py
```

GetDscLocalConfigurationManager.py

Returns the current meta-configuration applied to the computer. Similar to the Windows PowerShell cmdlet Get-DSCLocalConfigurationManager

```
# sudo ./GetLocalConfigurationManager.py
```

PerformRequiredConfigurationChecks.py

Immediately checks the configuration in accordance with the MetaConfiguration settings and applies the configuration if an update is available. Useful for immediately applying configuration changes on the pull server.

```
# sudo ./PerformRequiredConfigurationChecks.py
```

RestoreConfiguration.py

Applies the previous configuration known to DSC, a rollback.

```
# sudo ./RestoreConfiguration.py
```

SetDscLocalConfigurationManager.py

Applies a Meta Configuration MOF file to the computer. Similar to the Windows PowerShell cmdlet: Set-DSCLocalConfigurationManager. Requires the path to the Meta Configuration MOF to apply.

```
#sudo ./SendMetaConfiguration.py –configurationmof /tmp/localhost.meta.mof
```

StartDscLocalConfigurationManager.py

Applies a configuration MOF file to the computer. Similar to the Windows PowerShell cmdlet: Start-DscConfiguration. Requires the path to the configuration MOF to apply.

```
#sudo ./StartDscLocalConfigurationManager.py –configurationmof
/tmp/localhost.mof
```

TestDscConfiguration.py

Tests the current system configuration for compliance desired state. Similar to the Windows PowerShell cmdlet: Test-DscConfiguration.

```
# sudo ./TestDscConfiguration.py
```

InstallModule.py

Installs a custom DSC resource module. Requires the path to a .zip file containing the module shared object library and schema MOF files.

```
# sudo ./InstallModule.py /tmp/cnx_Resource.zip
```

RemoveModule.py

Removes a custom DSC resource module. Requires the name of the module to remove.

```
# sudo ./RemoveModule.py cnx_Resource
```

Using PowerShell Desired State Configuration for Linux with a Pull Server

Using HTTPS with the Pull Server

Though unencrypted HTTP is supported for communication with the Pull server, HTTPS (SSL/TLS) is recommended. When using HTTPS, the DSC Local Configuration Manager requires that the SSL certificate of the Pull server is verifiable (signed by a trusted authority, has a common name that matches the URL, etc.).

You can modify these HTTPS requirements as needed, by modifying the file `/etc/opt/omi/dsc/dsc.conf`. The supported properties defined in this file are:

- **NoSSLv3** set this to **true** to require the TLS protocol and set this to **false** to support SSLv3 or TLS. The default is **false.**
- **DoNotCheckCertificate** set this to **true** to ignore SSL certificate verification. The default is **false.**
- **CURL_CA_BUNDLE** an optional path to a curl-ca-bundle.crt file containing the CA certificates to trust for SSL/TLS. For more information, see: http://curl.haxx.se/docs/sslcerts.html
- **sslCipherSuite** Optionally set your preferred SSL cipher suite list. Only ciphers matching the rules defined by this list will be supported for HTTPS negotiation. The syntax and available ciphers on your computer depend on whether the cURL package is configured to use OpenSSL or NSS as its SSL library. To determine which SSL library cURL is using, run the following command and look for OpenSSL or NSS in the list of linked libraries:

```
# curl --version |head -n 1
curl 7.29.0 (x86_64-redhat-linux-gnu) libcurl/7.29.0 NSS/3.15.4
zlib/1.2.7 libidn/1.28 libssh2/1.4.3
```

  - For more information on configuring cipher support, see: http://curl.haxx.se/libcurl/c/CURLOPT_SSL_CIPHER_LIST.html

PowerShell Desired State Configuration for Linux Log Files

The following log files are generated for DSC for Linux messages.

Log file	Directory	Description
omiserver.log	/var/opt/omi/log	Messages relating to the operation of the OMI CIM server.
dsc.log	/var/opt/omi/log	Messages relating to the operation of the Local Configuration Manager and DSC resource operations.