Microsoft Centric Infrastructure

Desired State Configuration

Contents

A **walkthrough** is intended to bring you through a technical exercise. A walkthrough shows you how I completed a task in a particular context, on specific systems, at a point in time. The document is a good basic guide, but you should always confirm that the information is current and if a more recent best practice exists, you should make yourself aware of it.

Introduction

We can install features very easily in Windows Server by using Power Shell.

The command **Get-WindowsFeature** returns a list of available roles and features.

The command **Install-WindowsFeature** installs the required role or feature and **Uninstall-WindowsFeature** removes a feature.

However, maintaining a server can be more of a challenge, ensuring it is always in the correct state. This walkthrough is provided to introduce you to Desired State Configuration (DSC) in Windows Server 2019.

Managed Object Format (MOF) is the language used to describe Common Information Model (CIM) classes; we need to compile MOF files for using with DSC.

Deliverables

You will be required to keep a list of commands with notes as to why you are using them as well as results of any testing you have done. You may be asked to present these notes as evidence of work completed or as part of a lab book. Check with your lecturer!

Prerequisites

1. These notes are abbreviated, you should already understand some basic programming terminology. You have already covered an introduction to Python.
2. Ensure you have read the accompanying lecture notes before you begin. You should have already installed PowerShell 7.
3. I will carry out my exercises using
    a. A Windows 2019 VM Standard, Desktop Experience, running in Hyper-V.
    b. A Windows Core 2019 VM
4. You should have already installed Visual Studio Code as per my notes.
5. You have an existing GitHub account and a basic ability to use it.
6. You have created a directory to keep example files in and you are ready to code.
    a. On my VM, I am using **C:\Powershell**
    b. At the end of each session, I copy the files to **OneDrive\Powershell**

Aims

1. This walkthrough is intended to introduce the concept of Desired State Configuration (DSC) in Windows 2019.

Description

Setting up a server is one thing, keeping it with that configuration is another. Desired State Configuration (DSC) is a set of Power Shell features that allows us to configure and maintain a configuration of Windows Server 2019, to make sure every server is in the desired state. We could do this with scripts on a timed job, but this is a good way to automate this process as well.

Using DSC, we can

- Install and remove server roles and features
- Create registry settings
- Create files and folders and manage directories
- Start and stop processes
- Manage groups and users
- Run other Powershell scripts

Most importantly, we can examine a server to see if it is in its desired state and if not, fix it!

There are a range of built-in providers in DSC that enable this functionality.

```
PS C:\PowerShell\wt4\DscConfiguration> Get-DscResource

ImplementedAs   Name                    ModuleName                    Version    Properties
-------------   ----                    ----------                    -------    ----------
Binary          File                                                             {DestinationPath, Attributes, Checksum, Content...
Binary          SignatureValidation                                              {SignedItemType, TrustedStorePath}
PowerShell      PackageManagement       PackageManagement             1.4.8.1    {Name, AdditionalParameters, DependsOn, Ensure...}
PowerShell      PackageManagement       PackageManagement             1.0.0.1    {Name, AdditionalParameters, DependsOn, Ensure...}
PowerShell      PackageManagementSource PackageManagement             1.4.8.1    {Name, ProviderName, SourceLocation, DependsOn...}
PowerShell      PackageManagementSource PackageManagement             1.0.0.1    {Name, ProviderName, SourceUri, DependsOn...}
PowerShell      PSModule                PowerShellGet                 2.2.5      {Name, AllowClobber, DependsOn, Ensure...}
PowerShell      PSRepository            PowerShellGet                 2.2.5      {Name, DependsOn, Ensure, InstallationPolicy...}
PowerShell      Archive                 PSDesiredStateConfiguration   1.1        {Destination, Path, Checksum, Credential...}
PowerShell      Environment             PSDesiredStateConfiguration   1.1        {Name, DependsOn, Ensure, Path...}
PowerShell      Group                   PSDesiredStateConfiguration   1.1        {GroupName, Credential, DependsOn, Description...}
Composite       GroupSet                PSDesiredStateConfiguration   1.1        {DependsOn, PsDscRunAsCredential, GroupName, En...
Binary          Log                     PSDesiredStateConfiguration   1.1        {Message, DependsOn, PsDscRunAsCredential}
PowerShell      Package                 PSDesiredStateConfiguration   1.1        {Name, Path, ProductId, Arguments...}
Composite       ProcessSet              PSDesiredStateConfiguration   1.1        {DependsOn, PsDscRunAsCredential, Path, Credent...
PowerShell      Registry                PSDesiredStateConfiguration   1.1        {Key, ValueName, DependsOn, Ensure...}
PowerShell      Script                  PSDesiredStateConfiguration   1.1        {GetScript, SetScript, TestScript, Credential...}
PowerShell      Service                 PSDesiredStateConfiguration   1.1        {Name, BuiltInAccount, Credential, Dependencies...
Composite       ServiceSet              PSDesiredStateConfiguration   1.1        {DependsOn, PsDscRunAsCredential, Name, Startup...
PowerShell      User                    PSDesiredStateConfiguration   1.1        {UserName, DependsOn, Description, Disabled...}
PowerShell      WaitForAll              PSDesiredStateConfiguration   1.1        {NodeName, ResourceName, DependsOn, PsDscRunAsC...
PowerShell      WaitForAny              PSDesiredStateConfiguration   1.1        {NodeName, ResourceName, DependsOn, PsDscRunAsC...
PowerShell      WaitForSome             PSDesiredStateConfiguration   1.1        {NodeCount, NodeName, ResourceName, DependsOn...}
PowerShell      WindowsFeature          PSDesiredStateConfiguration   1.1        {Name, Credential, DependsOn, Ensure...}
Composite       WindowsFeatureSet       PSDesiredStateConfiguration   1.1        {DependsOn, PsDscRunAsCredential, Name, Ensure...}
PowerShell      WindowsOptionalFeature  PSDesiredStateConfiguration   1.1        {Name, DependsOn, Ensure, LogLevel...}
Composite       WindowsOptionalFeatureSet PSDesiredStateConfiguration 1.1        {DependsOn, PsDscRunAsCredential, Name, Ensure...}
PowerShell      WindowsPackageCab       PSDesiredStateConfiguration   1.1        {Ensure, Name, SourcePath, DependsOn...}
PowerShell      WindowsProcess          PSDesiredStateConfiguration   1.1        {Arguments, Path, Credential, DependsOn...}


PS C:\PowerShell\wt4\DscConfiguration> |
```

Note that one of the providers is called **File**, I can use this to automatically copy an important file to any server.

Also notice the provider **WindowsFeature.** To see which features I can add, I can use the command

Get-WindowsFeature

I am not going to show the output, it scrolls off the page!

Walkthrough

I create **wt4\dsc1.ps1** on **dc-1**, the script is on the VLE.

```
Configuration DscConfiguration
{

   param
   (
      [string[]]$ComputerName='localhost'
   )

   Import-DscResource -ModuleName PsDesiredStateConfiguration

   Node $ComputerName
   {
      WindowsFeature MyFeatureInstance
      {
         Ensure = 'Present'
         Name = 'RSAT'
      }

      WindowsFeature My2ndFeatureInstance
      {
         Ensure = 'Present'
         Name = 'DNS'
      }
      File HelloWorld {
         SourcePath = "C:\Users\Administrator\Documents\jor.txt"
         DestinationPath = "C:\Temp\HelloWorld.txt"
         Ensure = "Present"
         Contents   = "Hello World from DSC!"
      }
   }
}
DscConfiguration
```

Anatomy

The outer block is the configuration block, with the name **DscConfiguration**. We could add the node to run this against using

```
Node @('localhost', 'server-1')
```

In my script, I added a parameter so that the computers this will act against can be entered at the command line. Below I just use the ComputerName parameter "localhost".

```
PS C:\PowerShell\wt4> .\dsc1.ps1 -ComputerName "locahost"


    Directory: C:\PowerShell\wt4\DscConfiguration


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        01/12/2022     16:53           3582 localhost.mof
```

The node block will accept multiple node names.

Compile the MOF file

When I ran this, it resolved all variables and compiled a human readable MOF file. A folder gets created called whatever the script name is (**DscConfiguration).**

A MOF file gets created for each node and is saved in the script name directory. This is for the MOF files and any other supporting files which are created.

I only have a single example for localhost.

```
/*
@TargetNode='localhost'
@GeneratedBy=administrator
@GenerationDate=12/01/2022 16:16:45
@GenerationHost=DC-1
*/

instance of MSFT_RoleResource as $MSFT_RoleResource1ref
{
ResourceID = "[WindowsFeature]MyFeatureInstance";
 Ensure = "Present";
 SourceInfo = "C:\\PowerShell\\wt4\\dsc1.ps1::10::9::WindowsFeature";
 Name = "RSAT";
 ModuleName = "PsDesiredStateConfiguration";

ModuleVersion = "1.0";

 ConfigurationName = "DscConfiguration";

};
instance of MSFT_RoleResource as $MSFT_RoleResource2ref
{
ResourceID = "[WindowsFeature]My2ndFeatureInstance";
 Ensure = "Present";
 SourceInfo = "C:\\PowerShell\\wt4\\dsc1.ps1::16::9::WindowsFeature";
 Name = "DNS";
 ModuleName = "PsDesiredStateConfiguration";

ModuleVersion = "1.0";

 ConfigurationName = "DscConfiguration";

};
instance of OMI_ConfigurationDocument

                {
 Version="2.0.0";


                    MinimumCompatibleVersion = "1.0.0";
                    CompatibleVersionAdditionalProperties= {"Omi_BaseResource:ConfigurationName"};
                    Author="administrator";
                    GenerationDate="12/01/2022 16:16:45";
                    GenerationHost="DC-1";
                    Name="DscConfiguration";
                };
```

Apply the configuration

To apply the configuration, I run the command

Start-DscConfiguration -Path C:\PowerShell\WT4\DscConfiguration -Verbose -Wait -Force

```
PS C:\PowerShell\wt4> Start-DscConfiguration -Path C:\PowerShell\WT4\DscConfiguration -Verbose -Wait -Force
VERBOSE: Perform operation 'Invoke CimMethod' with following parameters, ''methodName' = SendConfigurationApply,'className' = MSFT_DSCLocalConfigurationManager
oot/Microsoft/Windows/DesiredStateConfiguration'.
VERBOSE: An LCM method call arrived from computer DC-1 with user sid S-1-5-21-1757429731-724326361-743102925-500.
VERBOSE: [DC-1]: LCM:  [ Start  Set      ]
VERBOSE: [DC-1]: LCM:  [ Start  Resource ]  [[WindowsFeature]MyFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ Start  Test     ]  [[WindowsFeature]MyFeatureInstance]
VERBOSE: [DC-1]:                            [[WindowsFeature]MyFeatureInstance] The operation 'Get-WindowsFeature' started: RSAT
VERBOSE: [DC-1]:                            [[WindowsFeature]MyFeatureInstance] The operation 'Get-WindowsFeature' succeeded: RSAT
VERBOSE: [DC-1]: LCM:  [ End    Test     ]  [[WindowsFeature]MyFeatureInstance]  in 0.5310 seconds.
VERBOSE: [DC-1]: LCM:  [ Skip   Set      ]  [[WindowsFeature]MyFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ End    Resource ]  [[WindowsFeature]MyFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ Start  Resource ]  [[WindowsFeature]My2ndFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ Start  Test     ]  [[WindowsFeature]My2ndFeatureInstance]
VERBOSE: [DC-1]:                            [[WindowsFeature]My2ndFeatureInstance] The operation 'Get-WindowsFeature' started: DNS
VERBOSE: [DC-1]:                            [[WindowsFeature]My2ndFeatureInstance] The operation 'Get-WindowsFeature' succeeded: DNS
VERBOSE: [DC-1]: LCM:  [ End    Test     ]  [[WindowsFeature]My2ndFeatureInstance]  in 0.3750 seconds.
VERBOSE: [DC-1]: LCM:  [ Skip   Set      ]  [[WindowsFeature]My2ndFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ End    Resource ]  [[WindowsFeature]My2ndFeatureInstance]
VERBOSE: [DC-1]: LCM:  [ Start  Resource ]  [[File]HelloWorld]
VERBOSE: [DC-1]: LCM:  [ Start  Test     ]  [[File]HelloWorld]
VERBOSE: [DC-1]:                            [[File]HelloWorld] The system cannot find the path specified.
VERBOSE: [DC-1]:                            [[File]HelloWorld] The related file/directory is: C:\Temp\HelloWorld.txt.
VERBOSE: [DC-1]: LCM:  [ End    Test     ]  [[File]HelloWorld]  in 0.0000 seconds.
VERBOSE: [DC-1]: LCM:  [ Start  Set      ]  [[File]HelloWorld]
VERBOSE: [DC-1]:                            [[File]HelloWorld] The system cannot find the path specified.
VERBOSE: [DC-1]:                            [[File]HelloWorld] The related file/directory is: C:\Temp\HelloWorld.txt.
VERBOSE: [DC-1]: LCM:  [ End    Set      ]  [[File]HelloWorld]  in 0.0150 seconds.
VERBOSE: [DC-1]: LCM:  [ End    Resource ]  [[File]HelloWorld]
VERBOSE: [DC-1]: LCM:  [ End    Set      ]
VERBOSE: [DC-1]: LCM:  [ End    Set      ]    in  1.1880 seconds.
VERBOSE: Operation 'Invoke CimMethod' complete.
VERBOSE: Time taken for configuration job to complete is 1.271 seconds

PS C:\PowerShell\wt4>
```

Testing

Despite the warning, PowerShell created the folder C:\Temp and the file HelloWorld.txt

To verify, run the command

Get-Content C:\Temp\HelloWorld.txt

To check the DSC Configuration Store, run the command

Get-DscConfiguration

Removing the configuration

To delete the configuration and stop using it, use the commands as shown below.

```
$Session = New-CimSession -ComputerName "localhost"
Remove-DscConfigurationDocument -Stage Current -CimSession $Session
```

After running this successfully, check the DSC Configuration Store again, it should be clear.

Maintaining DSC

We could use a timer to push this configuration or we could have a repository of MOF files on a file server and we could get the servers to pull configurations. We can use an SMB share [1] or web services; web ports tend to be more flexible in a large environment. We really need a certificate authority (CA) set up or we need a self-signed certificate to use this over the web.

References

Examples are based on documentation from Microsoft, DSC 1.1