Desired State Configuration

Topics

What is Desired State Configuration?

Desired State Configuration, or DSC, is a tool built into PowerShell that can be used to define a Windows host setup through code. The overall purpose of DSC is the same as Ansible, it is just executed in a different manner. Since Ansible 2.4, the `win_dsc` module has been added and can be used to take advantage of existing DSC resources when interacting with a Windows host.

More details on DSC can be viewed at DSC Overview.

Host Requirements

To use the `win_dsc` module, a Windows host must have PowerShell v5.0 or newer installed. All supported hosts can be upgraded to PowerShell v5.

Once the PowerShell requirements have been met, using DSC is as simple as creating a task with the `win_dsc` module.

Why Use DSC?

DSC and Ansible modules have a common goal which is to define and ensure the state of a resource. Because of this, resources like the DSC File resource and Ansible `win_file` can be used to achieve the same result. Deciding which to use depends on the scenario.

Reasons for using an Ansible module over a DSC resource:

- The host does not support PowerShell v5.0, or it cannot easily be upgraded
- The DSC resource does not offer a feature present in an Ansible module. For example win_regedit can manage the `REG_NONE` property type, while the DSC `Registry` resource cannot
- DSC resources have limited check mode support, while some Ansible modules have better checks
- DSC resources do not support diff mode, while some Ansible modules do
- Custom resources require further installation steps to be run on the host beforehand, while Ansible modules are built-in to Ansible
- There are bugs in a DSC resource where an Ansible module works

Reasons for using a DSC resource over an Ansible module:

- The Ansible module does not support a feature present in a DSC resource
- There is no Ansible module available
- There are bugs in an existing Ansible module

In the end, it doesn't matter whether the task is performed with DSC or an Ansible module; what matters is that the task is performed correctly and the playbooks are still readable. If you have more experience with DSC over Ansible and it does the job, just use DSC for that task.

How to Use DSC?

The `win_dsc` module takes in a free-form of options so that it changes according to the resource it is managing. A list of built in resources can be found at resources.

Using the Registry resource as an example, this is the DSC definition as documented by Microsoft:

```
Registry [string] #ResourceName
{
    Key = [string]
    ValueName = [string]
    [ Ensure = [string] { Enable | Disable }  ]
    [ Force =  [bool]   ]
    [ Hex = [bool] ]
    [ DependsOn = [string[]] ]
    [ ValueData = [string[]] ]
    [ ValueType = [string] { Binary | Dword | ExpandString | MultiString | Qword | String }  ]
}
```

When defining the task, `resource_name` must be set to the DSC resource being used - in this case the `resource_name` should be set to `Registry`. The `module_version` can refer to a specific version of the DSC resource installed; if left blank it will default to the latest version. The other options are parameters that are used to define the resource, such as `Key` and `ValueName`. While the options in the task are not case sensitive, keeping the case as-is is recommended because it makes it easier to distinguish DSC resource options from Ansible's `win_dsc` options.

This is what the Ansible task version of the above DSC Registry resource would look like:

```
- name: Use win_dsc module with the Registry DSC resource
  win dsc:
    resource name: Registry
    Ensure: Present
    Key: HKEY LOCAL MACHINE\SOFTWARE\ExampleKey
    ValueName: TestValue
    ValueData: TestData
```

Starting in Ansible 2.8, the `win_dsc` module automatically validates the input options from Ansible with the DSC definition. This means Ansible will fail if the option name is incorrect, a mandatory option is not set, or the value is not a valid choice. When running Ansible with a verbosity level of 3 or more (`-vvv`), the return value will contain the possible invocation options based on the `resource_name` specified. Here is an example of the invocation output for the above `Registry` task:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\[ansible-devel][docs][docsite][rst][user_guide]windows_dsc.rst, line 109)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
.. code-block:: ansible-output
```

```
        changed: [2016] => {
            "changed": true,
            "invocation": {
                "module_args": {
                    "DependsOn": null,
                    "Ensure": "Present",
                    "Force": null,
                    "Hex": null,
                    "Key": "HKEY_LOCAL_MACHINE\\SOFTWARE\\ExampleKey",
                    "PsDscRunAsCredential_password": null,
                    "PsDscRunAsCredential_username": null,
                    "ValueData": [
                        "TestData"
                    ],
                    "ValueName": "TestValue",
                    "ValueType": null,
                    "module_version": "latest",
                    "resource_name": "Registry"
                }
            },
            "module_version": "1.1",
            "reboot_required": false,
            "verbose_set": [
                "Perform operation 'Invoke CimMethod' with following parameters, ''methodName' = ResourceSet,'className' = MSFT_DSCLoca
                "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-3088887838-4058132883-1884671576-1105.",
                "[SERVER2016]: LCM:  [ Start  Set      ]  [[Registry]DirectResourceAccess]",
                "[SERVER2016]:                            [[Registry]DirectResourceAccess] (SET) Create registry key 'HKLM:\\SOFTWARE\\
                "[SERVER2016]:                            [[Registry]DirectResourceAccess] (SET) Set registry key value 'HKLM:\\SOFTWA
                "[SERVER2016]: LCM:  [ End    Set      ]  [[Registry]DirectResourceAccess]  in 0.1930 seconds.",
                "[SERVER2016]: LCM:  [ End    Set      ]    in  0.2720 seconds.",
                "Operation 'Invoke CimMethod' complete.",
                "Time taken for configuration job to complete is 0.402 seconds"
            ],
            "verbose_test": [
                "Perform operation 'Invoke CimMethod' with following parameters, ''methodName' = ResourceTest,'className' = MSFT_DSCLoc
                "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-3088887838-4058132883-1884671576-1105.",
                "[SERVER2016]: LCM:  [ Start  Test     ]  [[Registry]DirectResourceAccess]",
                "[SERVER2016]:                            [[Registry]DirectResourceAccess] Registry key 'HKLM:\\SOFTWARE\\ExampleKey' d
                "[SERVER2016]: LCM:  [ End    Test     ]  [[Registry]DirectResourceAccess] False in 0.2510 seconds.",
                "[SERVER2016]: LCM:  [ End    Set      ]    in  0.3310 seconds.",
                "Operation 'Invoke CimMethod' complete.",
                "Time taken for configuration job to complete is 0.475 seconds"
            ]
        }
```


The `invocation.module_args` key shows the actual values that were set as well as other possible values that were not set. Unfortunately this will not show the default value for a DSC property, only what was set from the Ansible task. Any `*_password` option will be masked in the output for security reasons, if there are any other sensitive module options, set `no_log: True` on the task to stop all task output from being logged.

Property Types

Each DSC resource property has a type that is associated with it. Ansible will try to convert the defined options to the correct type during execution. For simple types like `[string]` and `[bool]` this is a simple operation, but complex types like `[PSCredential]` or arrays (like `[string[]]`) this require certain rules.

PSCredential

A `[PSCredential]` object is used to store credentials in a secure way, but Ansible has no way to serialize this over JSON. To set a DSC PSCredential property, the definition of that parameter should have two entries that are suffixed with `_username` and `_password` for the username and password respectively. For example:

```
PsDscRunAsCredential_username: '{{ ansible_user }}'
PsDscRunAsCredential_password: '{{ ansible_password }}'

SourceCredential_username: AdminUser
SourceCredential_password: PasswordForAdminUser
```

> **Note**
>
> On versions of Ansible older than 2.8, you should set `no_log: yes` on the task definition in Ansible to ensure any credentials used are not stored in any log file or console output.

A `[PSCredential]` is defined with `EmbeddedInstance("MSFT_Credential")` in a DSC resource MOF definition.

CimInstance Type

A `[CimInstance]` object is used by DSC to store a dictionary object based on a custom class defined by that resource. Defining a value that takes in a `[CimInstance]` in YAML is the same as defining a dictionary in YAML. For example, to define a `[CimInstance]` value in Ansible:

```
# [CimInstance]AuthenticationInfo == MSFT_xWebAuthenticationInformation
AuthenticationInfo:
  Anonymous: no
  Basic: yes
  Digest: no
  Windows: yes
```

In the above example, the CIM instance is a representation of the class MSFT_xWebAuthenticationInformation. This class accepts four boolean variables, `Anonymous`, `Basic`, `Digest`, and `Windows`. The keys to use in a `[CimInstance]` depend on the class it represents. Please read through the documentation of the resource to determine the keys that can be used and the types of each key value. The class definition is typically located in the `<resource name>.schema.mof`.

HashTable Type

A `[HashTable]` object is also a dictionary but does not have a strict set of keys that can/need to be defined. Like a `[CimInstance]`, define it like a normal dictionary value in YAML. A `[HashTable]]` is defined with `EmbeddedInstance("MSFT_KeyValuePair")` in a DSC resource MOF definition.

Arrays

Simple type arrays like `[string[]]` or `[UInt32[]]` are defined as a list or as a comma separated string which are then cast to their type. Using a list is recommended because the values are not manually parsed by the `win_dsc` module before being passed to the DSC engine. For example, to define a simple type array in Ansible:

```
# [string[]]
ValueData: entry1, entry2, entry3
ValueData:
- entry1
- entry2
- entry3

# [UInt32[]]
ReturnCode: 0,3010
ReturnCode:
- 0
- 3010
```

Complex type arrays like `[CimInstance[]]` (array of dicts), can be defined like this example:

```
# [CimInstance[]]BindingInfo == MSFT_xWebBindingInformation
```

```
BindingInfo:
- Protocol: https
  Port: 443
  CertificateStoreName: My
  CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
  HostName: DSCTest
  IPAddress: '*'
  SSLFlags: 1
- Protocol: http
  Port: 80
  IPAddress: '*'
```

The above example, is an array with two values of the class MSFT_xWebBindingInformation. When defining a `[CimInstance[]]`, be sure to read the resource documentation to find out what keys to use in the definition.

DateTime

A `[DateTime]` object is a DateTime string representing the date and time in the ISO 8601 date time format. The value for a `[DateTime]` field should be quoted in YAML to ensure the string is properly serialized to the Windows host. Here is an example of how to define a `[DateTime]` value in Ansible:

```
# As UTC-0 (No timezone)
DateTime: '2019-02-22T13:57:31.2311892+00:00'

# As UTC+4
DateTime: '2019-02-22T17:57:31.2311892+04:00'

# As UTC-4
DateTime: '2019-02-22T09:57:31.2311892-04:00'
```

All the values above are equal to a UTC date time of February 22nd 2019 at 1:57pm with 31 seconds and 2311892 milliseconds.

Run As Another User

By default, DSC runs each resource as the SYSTEM account and not the account that Ansible use to run the module. This means that resources that are dynamically loaded based on a user profile, like the `HKEY_CURRENT_USER` registry hive, will be loaded under the `SYSTEM` profile. The parameter `PsDscRunAsCredential` is a parameter that can be set for every DSC resource force the DSC engine to run under a different account. As `PsDscRunAsCredential` has a type of `PSCredential`, it is defined with the `_username` and `_password` suffix.

Using the Registry resource type as an example, this is how to define a task to access the `HKEY_CURRENT_USER` hive of the Ansible user:

```
- name: Use win_dsc with PsDscRunAsCredential to run as a different user
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_CURRENT_USER\ExampleKey
    ValueName: TestValue
    ValueData: TestData
    PsDscRunAsCredential_username: '{{ ansible_user }}'
    PsDscRunAsCredential_password: '{{ ansible_password }}'
  no_log: yes
```

Custom DSC Resources

DSC resources are not limited to the built-in options from Microsoft. Custom modules can be installed to manage other resources that are not usually available.

Finding Custom DSC Resources

You can use the PSGallery to find custom resources, along with documentation on how to install them on a Windows host.

The `Find-DscResource` cmdlet can also be used to find custom resources. For example:

```
# Find all DSC resources in the configured repositories
Find-DscResource

# Find all DSC resources that relate to SQL
Find-DscResource -ModuleName "*sql*"
```

> **Note**
>
> DSC resources developed by Microsoft that start with `x`, means the resource is experimental and comes with no support.

Installing a Custom Resource

There are three ways that a DSC resource can be installed on a host:

- Manually with the `Install-Module` cmdlet
- Using the `win_psmodule` Ansible module
- Saving the module manually and copying it another host

This is an example of installing the `xWebAdministration` resources using `win_psmodule`:

```
- name: Install xWebAdministration DSC resource
  win_psmodule:
    name: xWebAdministration
    state: present
```

Once installed, the win_dsc module will be able to use the resource by referencing it with the `resource_name` option.

The first two methods above only work when the host has access to the internet. When a host does not have internet access, the module must first be installed using the methods above on another host with internet access and then copied across. To save a module to a local filepath, the following PowerShell cmdlet can be run:

```
Save-Module -Name xWebAdministration -Path C:\temp
```

This will create a folder called `xWebAdministration` in `C:\temp` which can be copied to any host. For PowerShell to see this offline resource, it must be copied to a directory set in the `PSModulePath` environment variable. In most cases the path `C:\Program Files\WindowsPowerShell\Module` is set through this variable, but the `win_path` module can be used to add different paths.

Examples

Extract a zip file

```
- name: Extract a zip file
  win_dsc:
    resource name: Archive
    Destination: C:\temp\output
    Path: C:\temp\zip.zip
    Ensure: Present
```

Create a directory

```
- name: Create file with some text
  win_dsc:
    resource name: File
    DestinationPath: C:\temp\file
    Contents: |
```

```
        Hello
        World
    Ensure: Present
    Type: File

- name: Create directory that is hidden is set with the System attribute
  win_dsc:
    resource_name: File
    DestinationPath: C:\temp\hidden-directory
    Attributes: Hidden,System
    Ensure: Present
    Type: Directory
```

Interact with Azure

```
- name: Install xAzure DSC resources
  win_psmodule:
    name: xAzure
    state: present

- name: Create virtual machine in Azure
  win_dsc:
    resource_name: xAzureVM
    ImageName: a699494373c04fc0bc8f2bb1389d6106__Windows-Server-2012-R2-201409.01-en.us-127GB.vhd
    Name: DSCHOST01
    ServiceName: ServiceName
    StorageAccountName: StorageAccountName
    InstanceSize: Medium
    Windows: yes
    Ensure: Present
    Credential_username: '{{ ansible_user }}'
    Credential_password: '{{ ansible_password }}'
```

Setup IIS Website

```
- name: Install xWebAdministration module
  win_psmodule:
    name: xWebAdministration
    state: present

- name: Install IIS features that are required
  win_dsc:
    resource_name: WindowsFeature
    Name: '{{ item }}'
    Ensure: Present
  loop:
  - Web-Server
  - Web-Asp-Net45

- name: Setup web content
  win_dsc:
    resource_name: File
    DestinationPath: C:\inetpub\IISSite\index.html
    Type: File
    Contents: |
      <html>
      <head><title>IIS Site</title></head>
      <body>This is the body</body>
      </html>
    Ensure: present

- name: Create new website
  win_dsc:
    resource_name: xWebsite
    Name: NewIISSite
    State: Started
    PhysicalPath: C:\inetpub\IISSite\index.html
    BindingInfo:
    - Protocol: https
      Port: 8443
      CertificateStoreName: My
      CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
      HostName: DSCTest
      IPAddress: '*'
      SSLFlags: 1
    - Protocol: http
      Port: 8080
      IPAddress: '*'
    AuthenticationInfo:
      Anonymous: no
      Basic: yes
      Digest: no
      Windows: yes
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\[ansible-devel][docs][docsite][rst][user_guide]windows_dsc.rst`, line 495)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`playbooks_intro`
       An introduction to playbooks
   :ref:`playbooks_best_practices`
       Tips and tricks for playbooks
   :ref:`List of Windows Modules <windows_modules>`
       Windows specific module list, all implemented in PowerShell
   `User Mailing List <https://groups.google.com/group/ansible-project>`_
       Have a question?  Stop by the google group!
   :ref:`communication_irc`
       How to join Ansible chat channels
```