SSM PowerShell Modules and Scripts – Part 3 of 5

Sivaprasad Padisetty (October 2015)

*Third in the series of SSM blogs, previous* [*one*](http://www.padisetty.com/2015/10/ssm-getting-started-and-msi.html) *covers the application task, this covers the task ‘aws:psModule’. This configuration element can be used in three ways a) Deploying PowerShell modules b) Executing an idempotent PowerShell script and c) Deploy PS module and then execute the script.*

# Schema for aws:psModule

"aws:psModule": {

"properties": [

{

"runCommand": "command",

"source": "url",

"sourceHash": "hash"

},

{

"runCommand": [

"command1",

"command2",

"command3"

],

"sourceHash": "hash"

},

...

]

}

# ExecutePsModule Helper function

The function ExecutePsModule can either deploy the PS module and/or execute PS script. This makes it flexible to use. The attribute “source” points to a zip file which contains one or more PS Modules. Each PS Module is a folder with appropriate psd1 file. The attribute “runCommand” defines the script to be executed. It can either be a string or a JSON array, where each line corresponds to one line in PS and is executed one after the other. All the lines are executed in the same session. Therefore, variables defined can be accessed by subsequent lines. The function is slightly more complicated as it dynamically constructs $properties. It adds “source” if $ModulePath is defined. Likewise, it adds “runCommand” if $RunCommand is defined. When both $ModulePath and $RunCommand are defined a comma separator is added.

function ExecutePSModule ($Instance, $ModulePath, $RunCommand)

{

$properties = ''

if ($ModulePath)

{

$properties += @"

"source": "$ModulePath"

"@

}

if ($RunCommand)

{

if ($properties.Length -gt 0)

{

$properties += ",`n"

}

$properties += @"

"runCommand": $RunCommand

"@

}

$doc = @"

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:psModule": {

"description": "Install and run ps modules.",

"properties": [

{

$properties

}

]

}

}

}

"@

SSMAssociate $instance $doc -RetrySeconds 600 -Credential $cred

}

# Deploying PowerShell Module

Step one is to package the PowerShell module to be deployed as a zip file. The zip file can contain more than one module. The easiest way is to copy all the PS modules into a folder, with one sub folder per PS module. Then, zip the folder (and its subfolders) into a single zipfile. Once the zip file is created, upload it to a S3 location with appropriate access. While S3 is not a requirement, it is convenient. The role created in part 1 of the [blog](http://www.padisetty.com/2015/08/ssm-prerequisites-setup-and-cleanup.html) does not grant additional S3 access. If the zip file is uploaded into S3 with restricted access, then the role permissions should be adjusted appropriately.

#Helper function to create the zip file

function PSUtilZipFolder(

$SourceFolder,

$ZipFileName,

$IncludeBaseDirectory = $true)

{

del $ZipFileName -ErrorAction 0

Add-Type -Assembly System.IO.Compression.FileSystem

[System.IO.Compression.ZipFile]::CreateFromDirectory($SourceFolder,

$ZipFileName, [System.IO.Compression.CompressionLevel]::Optimal,

$IncludeBaseDirectory)

}

#Zip the .\PSDemo to PSDemo.zip

$dir = pwd

PSUtilZipFolder -SourceFolder "$dir\PSDemo" `

-ZipFileName "$dir\PSDemo.zip" -IncludeBaseDirectory $false

#Upload PSDemo.zip into S3 bucket, with public read access

write-S3Object -BucketName 'sivabuckets3' -key 'public/PSDemo.zip' `

-File .\PSDemo.zip -PublicReadOnly

#delete the temporary zip file created above

del .\PSDemo.zip

#Deploy PS Module

ExecutePSModule -Instance $instance `

-ModulePath 'https://s3.amazonaws.com/sivabuckets3/public/PSDemo.zip'

The doc produced:

PS C:\temp\ssm> $doc

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:psModule": {

"description": "Install and run ps modules.",

"properties": [

{

"source": "https://s3.amazonaws.com/sivabuckets3/public/PSDemo.zip"

}

]

}

}

}

# Executing Idempotent PowerShell Script

An idempotent script is one that can be executed any number of times, but will alter the state only if the system is not already in the target state. Chocolatey (<https://chocolatey.org/>) is a package manager like apt-get, but designed for Windows. This repository has 1000s of packages that can be installed and updated with one line of code. Below example first installs chocolatey and then install chrome and 7-zip.

$runCmd = @'

[

"Set-ExecutionPolicy RemoteSigned -Force",

"$url = 'https://chocolatey.org/install.ps1'",

"iex ((new-object net.webclient).DownloadString($url)); $Error.Clear();",

"choco install googlechrome -y",

"choco install 7zip -y"

]

'@

ExecutePSModule -Instance $instance `

-RunCommand $runCmd

The doc produced:

PS C:\temp\ssm> $doc

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:psModule": {

"description": "Install and run ps modules.",

"properties": [

{

"runCommand": [

"Set-ExecutionPolicy RemoteSigned -Force",

"$url = 'https://chocolatey.org/install.ps1'",

"iex ((new-object net.webclient).DownloadString($url)); $Error.Clear()",

"choco install googlechrome -y",

"choco install 7zip -y"

]

}

]

}

}

}

You can find the code under “AWS/SSM” folder at <https://github.com/padisetty/Samples>.

Explore & Enjoy!

/Siva