SSM Getting Started and MSI Installations – Part 2 of 5

Sivaprasad Padisetty (October 2015)

*Amazon EC2 Simple Systems Manager (SSM) enables you to configure and manage your EC2 instances. Previous* [*blog*](http://www.padisetty.com/2015/08/ssm-prerequisites-setup-and-cleanup.html)*, covered launching an instance with the right prerequisites. This blog will get into the details of SSM. The basic flow is: create a JSON SSM configuration document and then associate with one or more instances. The agent (ec2config) that is running inside the Windows instance applies the configuration.*

*SSM associate is similar to user data, but has few distinct advantages a) unlike user data, SSM JSON document is a structured document b) User data can be defined only at the instance launch and cannot be modified during run time c) SSM JSON document is desired state based and is idempotent. This means the document can be applied multiple times periodically to prevent the system from diverging.*

# 

# JSON based Configuration Document

JSON document describes one or more tasks. Currently four types of tasks are supported, they are a) Install MSI packages b) PowerShell module c) CloudWatch configuration and d) Automated domain join. All the four tasks are idempotent. The document’s overall structure is:

{

"schemaVersion": "version",

"description": "description",

"runtimeConfig": {

"aws:applications": {...},

"aws:cloudWatch": {...},

"aws:domainJoin": {...},

"aws:psModule": {...}

}

}

This blog covers the application task where the action can be Install, Uninstall or Repair. The “source” points to the MSI package. The parameters map to the parameters passed to the installer. The “sourceHash” can be used to verify the integrity of the source package and is an optional attribute. As seen below, it can support multiple packages.

"aws:applications": {

"properties": [

{

"action": "action",

"source": "url",

"sourceHash": "hash"

},

{

"action": "action",

"source": "url",

"parameters": "string"

},

...

]

}

# SSM Basic Flow

SSM document defines the state of the instance that it is associated with. Currently, only one document can be associated with an instance. This means before associating a new document, current association has to be deleted. Since the document can be associated with multiple instances, by deleting an association does not delete the underlying document. The helper function below does the following: a) Deletes the association if present and optionally deletes the document b) Creates a new document and associates with the instance c) Waits for convergence d) Finally, throws an exception if convergence fails.

Another neat trick this helper function does is: it invokes the ec2config-cli to trigger the application of the document as soon as it is associated. This triggers immediate application instead of waiting for the next poll interval in ec2config. Likewise, periodic reapply can be achieved by creating a recurring schedule task.

function SSMAssociate (

$instance,

[string]$doc,

[PSCredential] $Credential,

[int]$RetrySeconds = 150,

[boolean]$ClearEventLog = $true,

[boolean]$DeleteDocument = $true)

{

#Only one association is support per instance at this time

#Delete the association if it exists.

$association = Get-SSMAssociationList -AssociationFilterList `

@{Key='InstanceId'; Value=$instance.instanceid}

if ($association)

{

Remove-SSMAssociation -InstanceId $association.InstanceId `

-Name $association.Name -Force

if ($DeleteDocument)

{

Remove-SSMDocument -Name $association.Name -Force

}

}

$instanceId = $instance.InstanceId

$ipaddress = $instance.PublicIpAddress

if ($ClearEventLog)

{

icm $ipaddress {Clear-EventLog -LogName Ec2ConfigService} `

-Credential $Credential -Port 80

}

#generate a new document with unique name

$name = 'doc-' + [Guid]::NewGuid()

Write-Verbose "Document Name=$name"

$null = New-SSMDocument -Content $doc -name $name

#assocate the document to the instance

$null = New-SSMAssociation -InstanceId $instance.InstanceId -Name $name

#apply config

$cmd = {& "$env:ProgramFiles\Amazon\Ec2ConfigService\ec2config-cli.exe" -a}

$null = icm $ipaddress $cmd -Credential $Credential -Port 80

#Wait for convergence

$cmd = {

$status = (Get-SSMAssociation -InstanceId $instanceid -Name $name).Status

$status.Name -eq 'Success' -or $status.Name -eq 'Failed'

}

$null = SSMWait $cmd -Message 'Converge Association' `

-RetrySeconds $RetrySeconds

#Output Status

$status = (Get-SSMAssociation -InstanceId $instanceid -Name $name).Status

Write-Verbose "Status=$($status.Name), Message=$($status.Message)"

if ($status.Name -ne 'Success')

{

throw 'SSM Failed'

}

}

# Installing 7-zip

InstallMSI is a helper function that takes the URL to MSI file and construct the JSON doc. The template for the document is defined as a multiline here-string, $doc. Once the $doc is constructed, it is passed as the input to the SSMAssociate helper shown above. After convergence, it validates if the app is installed by querying the WMI repository.

function InstallMSI ($instance, $Source, $ProductName)

{

$doc = @"

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:applications": {

"description": "Install $ProductName and PS module for networking scripts",

"properties": [

{

"action": "Install",

"extension": "MSI",

"source": "$Source"

}

]

}

}

}

"@

Write-Verbose "Install $ProductName instanceid=$($instance.InstanceId)"

SSMAssociate $instance $doc -Credential $cred

$cmd = {

$cmd1 = {gwmi win32\_product | where { $\_.Name -like $using:ProductName} | select Name }

$programs = icm $instance.PublicIpAddress $cmd1 -Credential $cred -Port 80

$programs -ne $null

}

$null = SSMWait $cmd -Message "$ProductName Install" -RetrySeconds 15

}

$source = 'http://downloads.sourceforge.net/sevenzip/7z938.msi'

$productName = '7-Zip 9.38'

InstallMSI $instance $source $productName

Sample doc produced by above function.

PS C:\temp\ssm> $doc

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:applications": {

"description": "Install 7-Zip 9.38 and PS module for networking scripts",

"properties": [

{

"action": "Install",

"extension": "MSI",

"source": "http://downloads.sourceforge.net/sevenzip/7z938.msi"

}

]

}

}

}

# Uninstalling 7-zip

Uninstalling the app is similar.

function UninstallMSI ($instance, $Source, $ProductName)

{

$doc = @"

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:applications": {

"description": "UnInstall $ProductName and PS module for networking scripts",

"properties": [

{

"action": "UnInstall",

"extension": "MSI",

"source": "$source"

}

]

}

}

}

"@

Write-Verbose "UnInstall $ProductName instanceid=$($instance.InstanceId)"

SSMAssociate $instance $doc -Credential $cred

$cmd = {

$cmd1 = {gwmi win32\_product | where { $\_.Name -like $using:ProductName} | select Name }

$programs = icm $instance.PublicIpAddress $cmd1 -Credential $cred -Port 80

$programs -eq $null

}

$null = SSMWait $cmd -Message "$ProductName Uninstall" -RetrySeconds 15

}

$source = 'http://downloads.sourceforge.net/sevenzip/7z938.msi'

$productName = '7-Zip 9.38'

UnInstallMSI $instance $source $productName

Sample doc produced by above function.

PS C:\temp\ssm> $doc

{

"schemaVersion": "1.0",

"description": "MSI Test Example",

"runtimeConfig": {

"aws:applications": {

"description": "UnInstall 7-Zip 9.38 and PS module for networking scripts",

"properties": [

{

"action": "UnInstall",

"extension": "MSI",

"source": "http://downloads.sourceforge.net/sevenzip/7z938.msi"

}

]

}

}

}

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

Explore & Enjoy!

/Siva