SSM Prerequisites Setup and Cleanup – Part 1 of 5

Sivaprasad Padisetty (May 2015)

*Amazon EC2 Simple Systems Manager (SSM) enables you to configure and manage your EC2 instances at scale. The design is light weight, declarative, idempotent and is in line with DevOps principles like simplicity, repeatable at scale, configuration as code. It is not reinventing a new desired state system, instead it integrates with existing one like PowerShell DSC. The basic flow is: define the configuration as a JSON document, then associate the document with one or more instances. The agent (ec2config) that is running inside the Windows instance retrieves and applies the document. Modern DevOps thinking is agile and on demand. In other words, EC2 instances are built on demand as needed, and does not hesitate to throw away in favor of rebuilding again from scratch. This ensures the state of the instance is predictable and can be rebuilt anytime. The source for configuration is typically version controlled along with other source code. AWS maintains 100s of AMIs (OS Images) and keeps them updated with latest patches like a clockwork. By having the automation to rebuild the instance from the stock, AMI avoids the overhead of maintaining custom AMIs. SSM helps with that automation. SSM has few requirements, this blog will cover automating these requirements.*

# Create IAM Role

IAM role is a secure mechanism to delegate access to users, applications, or services that don't normally have access to AWS resources. The IAM role defines who and what access to specific resource. These are defined as two separate JSON policy documents. When an IAM role is associated with the EC2 instance, EC2 makes temporary credentials available to the instance, and periodically rotates them. The applications running inside the instance can retrieve and use these temporary credentials. The role creation for EC2 is a multistep process a) Create the role with policy document that defines who can assume this role b) Write the permission policy document to grant specific permissions c) Create the instance profile d) Add role to the instance profile e) Associate instance profile to the instance as part of instance launch. Role cannot be directly associated with the instance, but only through instance profile. The profile container provides extra level of indirection. That is why a profile is created first and then the role is added to it. As a convention, profile name is the same as the role name. AWS Console follows the same convention as well.

Policy document defines “who” has access. In this case permissions are granted to the agent running inside the EC2 instance (ec2config).

#Define “who” has access

$assumePolicy = @"

{

"Version":"2012-10-17",

"Statement":[

{

"Sid":"",

"Effect":"Allow",

"Principal":{"Service":"ec2.amazonaws.com"},

"Action":"sts:AssumeRole"

}

]

}

"@

Below JSON document defines which API actions and resources, the application or user is allowed after assuming the role. The ssm:\* APIs listed below are needed for ec2config to retrieve the SSM document and to update the status after execution. The ds:CreateComputer is needed for auto domain join, logs:\* and cloudwatch:\* are needed for the cloudwatch plugin. Follow the security best practices in granting minimal permissions needed for all tasks (auto domain join, cloudwatch, PowerShell module, and MSI application install).

#Define “what” access (specific APIs and resource)

$policy = @"

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowAccessToSSM",

"Effect": "Allow",

"Action": [

"ssm:DescribeAssociations",

"ssm:ListAssociations",

"ssm:GetDocument",

"ssm:UpdateAssociationStatus",

"ds:CreateComputer",

"ec2:DescribeInstanceStatus",

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:DescribeLogGroups",

"logs:DescribeLogStreams",

"logs:PutLogEvents",

"cloudwatch:PutMetricData"

],

"Resource": [

"\*"

]

}

]

}

"@

Below script performs the above defined four steps to create a role and add it to the instance profile.

#step a - Create the role and specify who can assume

$role = 'role'

New-IAMRole -RoleName $role -AssumeRolePolicyDocument $assumePolicy

#step b - write the role policy

Write-IAMRolePolicy -RoleName $role -PolicyDocument $policy -PolicyName 'ssm'

#step c - Create instance profile

New-IAMInstanceProfile -InstanceProfileName $role

#step d - Add the role to the profile

Add-IAMRoleToInstanceProfile -InstanceProfileName $role -RoleName $role

# Create Firewall Security Group

A security group acts as a virtual firewall that controls the traffic for one or more instances. AWS offers layers of security and security group is one among them. Security best practice is to allow only the expected traffic and from known sources. The script below allows network traffic only from one source, where the script is executed. <http://checkip.amazonaws.com/> is an easy way to get the source IP address as seen by AWS. The source IP has to be a public IP address. Private IP address behind the NAT or proxy server will not work. The code below retrieves the source IP address and creates a security group to allow RDP, PS remoting, port 80, ICMP traffic from this source IP.

#pick appropriate subnet. Below line simply picks the first subnet

$subnet = Get-EC2Subnet | select -First 1

$securityGroupName = 'ssm-demo-sg' + $index

$securityGroupId = New-EC2SecurityGroup $securityGroupName -Description "SSM Demo" -VpcId $subnet.VpcId

$bytes = (Invoke-WebRequest 'http://checkip.amazonaws.com/').Content

$SourceIPRange = @(([System.Text.Encoding]::Ascii.GetString($bytes).Trim() + "/32"))

Write-Verbose "$sourceIPRange retreived from checkip.amazonaws.com"

$fireWallPermissions = @(

@{IpProtocol = 'tcp'; FromPort = 3389; ToPort = 3389; IpRanges = $SourceIPRange},

@{IpProtocol = 'tcp'; FromPort = 5985; ToPort = 5986; IpRanges = $SourceIPRange},

@{IpProtocol = 'tcp'; FromPort = 80; ToPort = 80; IpRanges = $SourceIPRange},

@{IpProtocol = 'icmp'; FromPort = -1; ToPort = -1; IpRanges = $SourceIPRange}

)

Grant-EC2SecurityGroupIngress -GroupId $securityGroupId `

-IpPermissions $fireWallPermissions

# Create Private KeyPair

EC2 key pair is based on private key and public key. The key pair is used to encrypt password in Windows and ssh access to Linux. EC2 generates random password and stores in an encrypted fashion, using the public key associated with key name defined during the instance launch. EC2 does not save the unencrypted password and also does not have access to private key. Since only the EC2 customer has access to the private key, only the customer can decrypt. Creating a key pair is simple. The private key associated with the key pair can only be retrieved at the time of creating. If the private key is lost, then the password cannot be retrieved. Script to create a key pair and retrieve the associated private key is shown below. The retrieved private key is saved in the current directory.

$keyName = 'ssm-demo-key'

$keypair = New-EC2KeyPair -KeyName $keyName

$dir = pwd

$keyfile = "$dir\$keyName.pem"

"$($keypair.KeyMaterial)" | Out-File -encoding ascii -filepath $keyfile

# Create Instances with an IAM Role and Security Group

Now that we have created all the dependent resources like IAM role, security group, and key pair, the instance can be created.

#Get the latest R2 base image

$image = Get-EC2ImageByName WINDOWS\_2012R2\_BASE

#User Data to enable PowerShell remoting on port 80

#User data must be passed in as 64bit encoding.

$userdata = @"

<powershell>

Enable-NetFirewallRule FPS-ICMP4-ERQ-In

Set-NetFirewallRule -Name WINRM-HTTP-In-TCP-PUBLIC -RemoteAddress Any

New-NetFirewallRule -Name "WinRM80" -DisplayName "WinRM80" -Protocol TCP -LocalPort 80

Set-Item WSMan:\localhost\Service\EnableCompatibilityHttpListener -Value true

</powershell>

"@

$utf8 = [System.Text.Encoding]::UTF8.GetBytes($userdata)

$userdataBase64Encoded = [System.Convert]::ToBase64String($utf8)

#Launch EC2 Instance with the role, firewall group

# and on the right subnet

$instance = (New-EC2Instance -ImageId $image.ImageId `

-InstanceProfile\_Id $role `

-AssociatePublicIp $true `

-SecurityGroupId $securityGroupId `

-SubnetId $subnet.SubnetId `

-KeyName $keyName `

-UserData $userdataBase64Encoded `

-InstanceType 'c3.large').Instances[0]

Instance creation is an async operation (i.e.) New-EC2Instance API just initiates the instance creation process, to wait for completion periodically poll in a loop. We will use “Wait” function explained in a previous blog located [here](http://www.padisetty.com/2014/07/generic-powershell-function-wait-to.html). Waiting for instance creation is achieved in two steps. First one is to poll for password generation. Once the password is generated, keep trying until the remote connection is established successfully.

#Wait function

function SSMWait (

[ScriptBlock] $Cmd,

[string] $Message,

[int] $RetrySeconds)

{

$msg = "Waiting for $Message to succeed"

$t1 = Get-Date

Write-Verbose "$msg in $RetrySeconds seconds"

while ($true)

{

$t2 = Get-Date

try

{

$result = & $cmd 2>$null | select -Last 1

if ($? -and $result)

{

Write-Verbose "Succeeded $Message in " +

"$([int]($t2-$t1).TotalSeconds) Seconds, Result=$result"

break;

}

}

catch

{

}

$t = [int]($t2 - $t1).TotalSeconds

if ($t -gt $RetrySeconds)

{

throw "Timeout - $Message after $RetrySeconds seconds, " +

"Current result=$result"

break

}

Write-Verbose "$msg ($t/$RetrySeconds) Seconds."

Sleep -Seconds 5

}

}

#Wait to retrieve password

$cmd = {

$password = Get-EC2PasswordData -InstanceId $instance.InstanceId `

-PemFile $keyfile -Decrypt

$password -ne $null

}

SSMWait $cmd 'Password Generation' 600

$password = Get-EC2PasswordData -InstanceId $instance.InstanceId `

-PemFile $keyfile -Decrypt

$securepassword = ConvertTo-SecureString $Password -AsPlainText -Force

$creds = New-Object System.Management.Automation.PSCredential ("Administrator", $securepassword)

#update the instance to get the public IP Address

$instance = (Get-EC2Instance $instance.InstanceId).Instances[0]

#Wait for remote PS connection

$cmd = {

icm $instance.PublicIpAddress {dir c:\} -Credential $creds -Port 80

}

SSMWait $cmd 'Remote Connection' 450

# Cleanup

The code below shows how to clean up the resources created above. The code also includes the deletion of the private key that is saved locally and the associated key pair.

#Cleanup

#Terminate the instance

$null = Stop-EC2Instance -Instance $instance.InstanceId -Force -Terminate

#Remove the instance role and IAM Role

Remove-IAMRoleFromInstanceProfile -InstanceProfileName $role `

-RoleName $role -Force

Remove-IAMInstanceProfile $role -Force

Remove-IAMRolePolicy $role ssm -Force

Remove-IAMRole $role -Force

#delete keypair

del $keyfile -ea 0

Remove-EC2KeyPair -KeyName $keyName -Force

#To deal with timing, SSMWait is used.

SSMWait {(Remove-EC2SecurityGroup $securityGroupId -Force) -eq $null} `

'Delete Security Group' 150

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

Explore & Enjoy!

/Siva