PowerShell Script to launch and remotely connect to an EC2 Instance

Sivaprasad Padisetty (February 2014)

*AWS offers rich command line interfaces for automation. Pretty much anything you do from the AWS console can be done from the command line. In this blog, I will cover an end to end EC2 sample that will launch an instance, set the firewall rules, create key pairs, enables PS remoting, connect to the instance remotely and finally terminate it. This script is intended to cover the concepts as opposed to using it in production. Please be warned that there is no error checking in this script.*

*AWS offers multiple flavors of command line interface (CLI) for Windows. They are,*

* *PowerShell based* [*http://aws.amazon.com/powershell/*](http://aws.amazon.com/powershell/)*.*
* *AWS CLI* [*http://aws.amazon.com/cli/*](http://aws.amazon.com/cli/)*. One executable that drives all the services. This one can run both in Windows and Linux.*
* *Amazon EC2 CLI is a community contributed Java based command line tool* [*http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/InstallEC2CommandLineTools.html*](http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/InstallEC2CommandLineTools.html)

*I will cover the PS based script in detail, as Windows users will likely use this model. For other flavors, I will give a pointer to equivalent script, should be easy to follow.*

# 

# Prerequisites and Helper function

* Sign up for AWS and get the AccessKey & SecretKey. You can find info about AWS Account and Access Keys at [link](http://docs.aws.amazon.com/powershell/latest/userguide/pstools-appendix-signup.html).
* Install PowerShell module from [link](http://aws.amazon.com/powershell/). All the AWS tools for Windows are nicely bundled as a single MSI. Setup instructions can be found at [link](http://docs.aws.amazon.com/powershell/latest/userguide/pstools-getting-set-up.html)
* First thing first, to ensure that AWS PS module is loaded correctly, do the following

# PSModule should be autoloaded in PS v3.0 and above

# setup adds AWS module location to the PSModulePath

# To manually load PSModule

# import-module "C:\Program Files (x86)\AWS Tools\PowerShell\AWSPowerShell\AWSPowerShell.psd1"

# To check if AWS PS module is accessible.

Get-AWSPowerShellVersion -ListServices

* Call Initialize-AWSDefaults to setup the default AccessKey and SecretKey to use. This way, AccessKey and SecretKey need not be provided for every cmdlet invocation. This is very handy. Replace AccessKey and SecretKey with yours!

Initialize-AWSDefaults -AccessKey AKIAIOSFODNN7EXAMPLE -SecretKey wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY -Region us-east-1

* Since the EC2 instance that we are going to create is not a domain joined machine, it has to be added to the trusted hosts. The computers in the TrustedHosts list are not authenticated. (i.e.) There is no way for the client to know if it is talking to the right machine. The client may send credential information to these computers. Either add the specific DNSName or “\*” to trust any machine. Since it is for the testing purpose, I chose to add “\*”

Set-Item WSMan:\localhost\Client\TrustedHosts "\*" -Force

* Helper function to wait for the desired EC2 instance state.

function WaitForState ($instanceid, $desiredstate)

{

while ($true)

{

$a = Get-EC2Instance -Filter @{Name = "instance-id"; Values = $instanceid}

$state = $a.Instances[0].State.Name

if ($state -eq $desiredstate)

{

break;

}

"$(Get-Date) Current State = $state, Waiting for Desired State=$desiredstate"

Sleep -Seconds 5

}

}

# Firewall Security Groups

A security group acts as a firewall that controls the traffic for one or more instances. When you launch an instance, you associate one or more security groups with the instance. You add rules to each security group that allow traffic to or from its associated instances. You can modify the rules for a security group at any time; the new rules are automatically applied to all instances that are associated with the security group. When we decide whether to allow traffic to reach an instance, we evaluate all the rules from all the security groups that are associated with the instance. More info [here](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html)

The script below creates a security group named “sg1”, adds firewall exceptions for PowerShell Remoting, Remote Desktop and ICMP. Allowing ICMP enables “ping” to function, helps with debugging. The example below opens up to any IpRange, which means that the EC2 instance can be contacted from anywhere in the world.

$groupid = New-EC2SecurityGroup sg1 -Description "Security group 1"

Get-EC2SecurityGroup -GroupNames sg1

Grant-EC2SecurityGroupIngress -GroupName sg1 -IpPermissions @{IpProtocol = "icmp"; FromPort = -1; ToPort = -1; IpRanges = @("0.0.0.0/0")}

$ipPermissions = New-Object Amazon.EC2.Model.IpPermission

$ipPermissions.IpProtocol = "tcp"

$ipPermissions.FromPort = 3389

$ipPermissions.ToPort = 3389

$ipPermissions.IpRanges.Add("0.0.0.0/0")

Grant-EC2SecurityGroupIngress -GroupName sg1 -IpPermissions $ipPermissions

Grant-EC2SecurityGroupIngress -GroupName sg1 -IpPermissions @{IpProtocol = "udp"; FromPort = 3389; ToPort = 3389; IpRanges = @("0.0.0.0/0")}

Grant-EC2SecurityGroupIngress -GroupName sg1 -IpPermissions @{IpProtocol = "tcp"; FromPort = 5985; ToPort = 5986; IpRanges = @("0.0.0.0/0")}

# EC2 Key Pairs

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt a piece of data, such as a password, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair.

To log in to your instance, you must create a key pair, specify the name of the key pair when you launch the instance, and provide the private key when you connect to the instance. You use a key pair to obtain the administrator password and then log in using RDP. More info [here](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html).

The script below creates a new key pair and saves the private key under $folder. Note you have to use ASCII encoding instead of the default encoding. The key name and fingerprint are stored in the same file, which help to correlate. The private key is saved locally, if you lose it there is no way to retrieve it. AWS does not store the private key.

#create a KeyPair, this is used to encrypt the Administrator password.

$keypair1 = New-EC2KeyPair -KeyName keypair1

"$($keypair1.KeyMaterial)" | out-file -encoding ascii -filepath $folder\keypair1.pem

"KeyName: $($keypair1.KeyName)" | out-file -encoding ascii -filepath $folder\keypair1.pem -Append

"KeyFingerprint: $($keypair1.KeyFingerprint)" | out-file -encoding ascii -filepath $folder\keypair1.pem -Append

# Find the image

When you launch an Amazon EC2 instance, you need to specify an Amazon Machine Image (AMI) for the instance configuration in which you are interested. However, the IDs for the AMIs change periodically because AWS updates these images with new features and security enhancements. Get-EC2Image and Get-EC2ImageByName cmdlets are used to find the current ID.

$a = Get-EC2Image -Filters @{Name = "name"; Values = "Windows\_Server-2012-RTM-English-64Bit-Base\*"}

$imageid = $a.ImageId

Alternatively,

#use Get-EC2ImageByName to the list of images

Get-EC2ImageByName

#get the specific image

$a = Get-EC2ImageByName -Names WINDOWS\_2012\_BASE

$imageid = $a.ImageId

# Launch EC2 instance

Now that the security group and Key Pairs are ready, we are ready to launch the EC2 instance using NewEC2Intance.

EC2 lets you define userdata to run custom scripts during the launch. This metadata is made available from within the instance at http://169.254.169.254/latest/user-data/. This information remains static for the life of the instance, persisting when the instance is stopped and started, until it is terminated. For EC2Config to execute userdata, you must enclose the lines of the script within one of the following special tags:

* <script></script> - Run any command that you can run at the cmd.exe prompt.
* <powershell></powershell> - Run any command that you can run at the Windows PowerShell prompt.

By default, Windows Firewall restricts PS remoting to local subnet only. Set-NetFirewallRule is executed via userdata section to open this up to any address.

$userdata = @"

<powershell>

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

</powershell>

"@

#Userdata has to be base64 encoded

$userdataBase64Encoded = [System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($userdata))

$a = New-EC2Instance -ImageId $imageid -MinCount 1 -MaxCount 1 -InstanceType t1.micro -KeyName keypair1 -SecurityGroups sg1 -UserData $userdataBase64Encoded

$instanceid = $a.Instances[0].InstanceId

#Wait for the running state

WaitForState $instanceid "Running"

$a = Get-EC2Instance -Filter @{Name = "instance-id"; Values = $instanceid}

$publicDNS = $a.Instances[0].PublicDnsName

#Wait for ping to succeed

while ($true)

{

ping $publicDNS

if ($LASTEXITCODE -eq 0)

{

break

}

"$(Get-Date) Waiting for ping to succeed"

Sleep -Seconds 10

}

# Extract the password

Get-EC2PasswordData is used to extract the password by passing the private key file created earlier. This is tried in a loop until the password is ready.

$password = $null

#Wait until the password is available

#blindsly eats all the exceptions, bad idea for a production code.

while ($password -eq $null)

{

try

{

$password = Get-EC2PasswordData -InstanceId $instanceid -PemFile $folder\keypair1.pem -Decrypt

}

catch

{

"$(Get-Date) Waiting for PasswordData to be available"

Sleep -Seconds 10

}

}

# Remote PowerShell

The script below establishes a remote session, invokes a remote command (using Invoke-Command), then cleans up the session. The remote command executed is “Invoke-WebRequest” to obtain the userdata. Since the instance might not be fully initialized, this is tried in a loop.

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

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

#Wait until PSSession is available

while ($true)

{

$s = New-PSSession $publicDNS -Credential $creds 2>$null

if ($s -ne $null)

{

break

}

"$(Get-Date) Waiting for remote PS connection"

Sleep -Seconds 10

}

Invoke-Command -Session $s {(Invoke-WebRequest http://169.254.169.254/latest/user-data).RawContent}

Remove-PSSession $s

# Terminate the instance and cleanup

Script below terminates the instance and then cleans up the security group and key pair created.

#Terminate the Instance

Get-EC2Instance -Filter @{Name = "instance-id"; Values = $instanceid} | Stop-EC2Instance -Force -Terminate

#Wait for the instance state to be terminated.

WaitForState $instanceid "Terminated"

Remove-EC2KeyPair -KeyName keypair1 -Force

#There is a timing thing, so has to retry.

$err = $true

while ($err)

{

$err = $false

try

{

Remove-EC2SecurityGroup -GroupName sg1 -Force

}

catch

{

$err = $true

}

}

# Alternate CLI

1. AWS CLI based script can be found [here](https://github.com/padisetty/Samples/blob/master/AWS/01%20EC2%20Launch%20AWS%20CLI.ps1).
2. Amazon EC2 CLI based script can be found [here](https://github.com/padisetty/Samples/blob/master/AWS/01%20EC2%20Launch%20Amazon%20EC2%20CLI.ps1).

You can find all the scripts under AWS folder at <https://github.com/padisetty/Samples>.

Explore & Enjoy!

/Siva