Generic PowerShell Function “Wait” to Convert Asynchronous APIs to Synchronous

Sivaprasad Padisetty (July 2014)

*While Asynchronous APIs have several benefits it can increase the complexity and design of the code. This is especially true when writing simple scripts. To convert async API to synchronous, you need to check the status if done, sleep for a few seconds and check again. Do this until the operation is complete or times out. Given this pattern is frequently used, a generic function “wait” is presented in this blog. This pattern is applied for Start-EC2Instance and Stop-EC2Instance. If you are not already familiar with ScriptBlock in PowerShell, please read* [*All About PowerShell ScriptBlock*](http://www.padisetty.com/2014/05/all-about-powershell-scriptblock.html)*.*

# Wait Usage

The proposed function “Wait” function takes three parameters

* $Cmd – This is the command (ScriptBlock) is executed in a loop again and again until it succeeds or timeout occurs. A success means no errors or exception occurred in the execution of ScriptBlock and the return value has to be logically true.
* $Message – A text message to identify the operation
* $RetrySeconds – Number of seconds to retry before failing.

# Wait Function

The idea of the wait function is simple. It takes $cmd (ScriptBlock) as input, it executes the ScriptBlock in a loop, until it returns true or times out.

# local variables has \_wait\_ prefix to avoid potential conflict in ScriptBlock

# Retry the scriptblock $cmd until no error and return true

function wait ([ScriptBlock] $Cmd, [string] $Message, [int] $RetrySeconds)

{

$\_wait\_activity = "Waiting for $Message to succeed"

$\_wait\_t1 = Get-Date

$\_wait\_timeout = $false

while ($true)

{

try

{

$\_wait\_success = false

$\_wait\_result = & $cmd 2>$null | select -Last 1

if ($? -and $\_wait\_result)

{

$\_wait\_success = $true

}

}

catch

{

}

$\_wait\_t2 = Get-Date

if ($\_wait\_success)

{

$\_wait\_result

break;

}

if (($\_wait\_t2 - $\_wait\_t1).TotalSeconds -gt $RetrySeconds)

{

$\_wait\_timeout = $true

break

}

$\_wait\_seconds = [int]($\_wait\_t2 - $\_wait\_t1).TotalSeconds

Write-Progress -Activity $\_wait\_activity -PercentComplete (100.0\*$\_wait\_seconds/$RetrySeconds) `

-Status "$\_wait\_seconds Seconds, will try for $RetrySeconds seconds before timeout"

Sleep -Seconds 15

}

Write-Progress -Activity $\_wait\_activity -Completed

if ($\_wait\_timeout)

{

Write-Verbose "$\_wait\_t2 $Message [$([int]($\_wait\_t2-$\_wait\_t1).TotalSeconds) Seconds - Timeout]"

throw "Timeout - $Message after $RetrySeconds seconds"

}

else

{

Write-Verbose "$\_wait\_t2 Succeeded $Message in $([int]($\_wait\_t2-$\_wait\_t1).TotalSeconds) Seconds."

}

}

# Stop-WinEC2Instance

In AWS, Stop-EC2Instance is asynchronous function (i.e.) it initiates the stop but returns right away. To make it synchronous Wait is used, which checks the state in a loop.

“(Get-EC2Instance -Instance $InstanceId).Instances[0].State.Name” returns the current state of the instance. $cmd will return true as soon as the state changes to ‘stopped’. Code for Stop-WinEC2Instance is given below:

function Stop-WinEC2Instance (

[Parameter (Position=1, Mandatory=$true)][string]$InstanceId,

[Parameter(Position=2)][string]$Region=$DefaultRegion

)

{

trap { break }

$ErrorActionPreference = 'Stop'

Set-DefaultAWSRegion $Region

Write-Verbose "Stop-WinEC2Instance - InstanceId=$InstanceId, Region=$Region"

$result = Get-EC2Instance $InstanceId

$InstanceId = $result.instances.InstanceId

$a = Stop-EC2Instance -Instance $InstanceId -Force

$cmd = { (Get-EC2Instance -Instance $InstanceId).Instances[0].State.Name -eq "Stopped" }

$a = Wait $cmd "Stop-WinEC2Instance InstanceId=$InstanceId- Stopped state" 450

}

Sample usage:

PS C:\temp> Stop-WinEC2Instance i-08cdb523

VERBOSE: Stop-WinEC2Instance - InstanceId=i-08cdb523, Region=us-east-1

VERBOSE: 07/05/2014 20:42:06 Succeeded Stop-WinEC2Instance InstanceId=i-08cdb523-

Stopped state in 77 Seconds.

# Start-WinEC2Instance

In AWS, Start-EC2Instance is asynchronous function (i.e.) it initiates the start but returns right away. To make it synchronous Wait is called, which checks the state in a loop. The function also waits until a remote PowerShell connection is established. Optionally it waits until the reachability check succeeds.

Given the Wait function polls until it returns true, the trick to convert the result of ping to bool is to check the $LASTEXITCODE. This is achieved by “{ping $publicDNS; $LASTEXITCODE -eq 0}”

“{New-PSSession $publicDNS -Credential $Cred -Port 80}” is used to poll for remote PowerShell connection. Note: In this case it is assumed that PowerShell on the remote machine is running on port 80 instead of the default port of 5985.

“{$(Get-EC2InstanceStatus $InstanceId).Status.Status -eq 'ok'}” is used for polling reachability check.

The code for Start-WinEC2Instance is:

function Start-WinEC2Instance (

[Parameter (Position=1, Mandatory=$true)]$InstanceId,

[System.Management.Automation.PSCredential][Parameter(Mandatory=$true, Position=2)]$Cred,

[switch]$IsReachabilityCheck,

[Parameter(Position=3)]$Region=$DefaultRegion

)

{

trap { break }

$ErrorActionPreference = 'Stop'

Set-DefaultAWSRegion $Region

Write-Verbose "Start-WinEC2Instance - InstanceId=$InstanceId, Region=$Region"

$result = Get-EC2Instance $InstanceId

$InstanceId = $result.instances.InstanceId

$startTime = Get-Date

$a = Start-EC2Instance -Instance $InstanceId

$cmd = { $(Get-EC2Instance -Instance $InstanceId).Instances[0].State.Name -eq "running" }

$a = Wait $cmd "Start-WinEC2Instance - running state" 450

#Wait for ping to succeed

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

$publicDNS = $instance.PublicDnsName

Write-Verbose "publicDNS = $($instance.PublicDnsName)"

$cmd = { ping $publicDNS; $LASTEXITCODE -eq 0}

$a = Wait $cmd "Start-WinEC2Instance - ping" 450

$cmd = {New-PSSession $publicDNS -Credential $Cred -Port 80}

$s = Wait $cmd "Start-WinEC2Instance - Remote connection" 300

Remove-PSSession $s

if ($IsReachabilityCheck)

{

$cmd = { $(Get-EC2InstanceStatus $InstanceId).Status.Status -eq 'ok'}

$a = Wait $cmd "Start-WinEC2Instance - Reachabilitycheck" 600

}

Write-Verbose ('Start-WinEC2Instance - {0:mm}:{0:ss} - to start' -f ((Get-Date) - $startTime))

}

Sample usage:

PS C:\temp> $cred = Get-Credential

cmdlet Get-Credential at command pipeline position 1

Supply values for the following parameters:

PS C:\temp> Start-WinEC2Instance i-08cdb523 $cred

VERBOSE: Start-WinEC2Instance - InstanceId=i-08cdb523, Region=us-east-1

VERBOSE: 07/05/2014 20:44:11 Succeeded Start-WinEC2Instance - running state in 77

Seconds.

VERBOSE: publicDNS = ec2-54-88-243-187.compute-1.amazonaws.com

VERBOSE: 07/05/2014 20:44:48 Succeeded Start-WinEC2Instance - ping in 37 Seconds.

VERBOSE: 07/05/2014 20:44:53 Succeeded Start-WinEC2Instance - Remote connection i

n 5 Seconds.

VERBOSE: Start-WinEC2Instance - 01:59 - to start

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

Explore & Enjoy!

/Siva