Install epel release,ansible,puthon-pip and pywinrm

yum install epel-release

yum install ansible

yum install python-pip

pip install pywinrm

Make sure Ansible can connect to windows by DNS name

cat /etc/hosts

192.168.1.59 winserver

Add Windows to Ansible hosts file

cat /etc/ansible/hosts

[windows]

winserver

On windows. open powershell and execute following command, it will create self-signed certificate and will Turn on and configure WinRM

#Requires -Version 3.0

# Configure a Windows host for remote management with Ansible

# -----------------------------------------------------------

#

# This script checks the current WinRM (PS Remoting) configuration and makes

# the necessary changes to allow Ansible to connect, authenticate and

# execute PowerShell commands.

#

# All events are logged to the Windows EventLog, useful for unattended runs.

#

# Use option -Verbose in order to see the verbose output messages.

#

# Use option -CertValidityDays to specify how long this certificate is valid

# starting from today. So you would specify -CertValidityDays 3650 to get

# a 10-year valid certificate.

#

# Use option -ForceNewSSLCert if the system has been SysPreped and a new

# SSL Certificate must be forced on the WinRM Listener when re-running this

# script. This is necessary when a new SID and CN name is created.

#

# Use option -EnableCredSSP to enable CredSSP as an authentication option.

#

# Use option -DisableBasicAuth to disable basic authentication.

#

# Use option -SkipNetworkProfileCheck to skip the network profile check.

# Without specifying this the script will only run if the device's interfaces

# are in DOMAIN or PRIVATE zones.  Provide this switch if you want to enable

# WinRM on a device with an interface in PUBLIC zone.

#

# Use option -SubjectName to specify the CN name of the certificate. This

# defaults to the system's hostname and generally should not be specified.

# Written by Trond Hindenes <trond@[hindenes.com](https://docs.corp.hentsu.com/display/ENG/hindenes.com)>

# Updated by Chris Church <cchurch@[ansible.com](https://docs.corp.hentsu.com/display/ENG/ansible.com)>

# Updated by Michael Crilly <mike@[autologic.cm](https://docs.corp.hentsu.com/display/ENG/autologic.cm)>

# Updated by Anton Ouzounov <[Anton.Ouzounov@careerbuilder.com](https://docs.corp.hentsu.com/display/ENG/Anton.Ouzounov@careerbuilder.com)>

# Updated by Nicolas Simond <contact@[nicolas-simond.com](https://docs.corp.hentsu.com/display/ENG/nicolas-simond.com)>

# Updated by Dag Wieërs <dag@[wieers.com](https://docs.corp.hentsu.com/display/ENG/wieers.com)>

# Updated by Jordan Borean <jborean93@[gmail.com](https://docs.corp.hentsu.com/display/ENG/gmail.com)>

# Updated by Erwan Quélin <[erwan.quelin@gmail.com](https://docs.corp.hentsu.com/display/ENG/erwan.quelin@gmail.com)>

# Updated by David Norman <david@[dkn.email](https://docs.corp.hentsu.com/display/ENG/dkn.email)>

#

# Version 1.0 - 2014-07-06

# Version 1.1 - 2014-11-11

# Version 1.2 - 2015-05-15

# Version 1.3 - 2016-04-04

# Version 1.4 - 2017-01-05

# Version 1.5 - 2017-02-09

# Version 1.6 - 2017-04-18

# Version 1.7 - 2017-11-23

# Version 1.8 - 2018-02-23

# Version 1.9 - 2018-09-21

# Support -Verbose option

[CmdletBinding()]

Param (

    [string]$SubjectName = $env:COMPUTERNAME,

    [int]$CertValidityDays = 1095,

    [switch]$SkipNetworkProfileCheck,

    $CreateSelfSignedCert = $true,

    [switch]$ForceNewSSLCert,

    [switch]$GlobalHttpFirewallAccess,

    [switch]$DisableBasicAuth = $false,

    [switch]$EnableCredSSP

)

Function Write-Log

{

    $Message = $args[0]

    Write-EventLog -LogName Application -Source $EventSource -EntryType Information -EventId 1 -Message $Message

}

Function Write-VerboseLog

{

    $Message = $args[0]

    Write-Verbose $Message

    Write-Log $Message

}

Function Write-HostLog

{

    $Message = $args[0]

    Write-Output $Message

    Write-Log $Message

}

Function New-LegacySelfSignedCert

{

    Param (

        [string]$SubjectName,

        [int]$ValidDays = 1095

    )

    $hostnonFQDN = $env:computerName

    $hostFQDN = [[System.Net.Dns]::GetHostByName((](https://docs.corp.hentsu.com/display/ENG/System.Net.Dns%5d::GetHostByName(()$env:computerName)).Hostname

    $SignatureAlgorithm = "SHA256"

    $name = New-Object -COM "[X509Enrollment.CX500DistinguishedName.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX500DistinguishedName.1)"

    $name.Encode("CN=$SubjectName", 0)

    $key = New-Object -COM "[X509Enrollment.CX509PrivateKey.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509PrivateKey.1)"

    $key.ProviderName = "Microsoft Enhanced RSA and AES Cryptographic Provider"

    $key.KeySpec = 1

    $key.Length = 4096

    $key.SecurityDescriptor = "D:PAI(A;;0xd01f01ff;;;SY)(A;;0xd01f01ff;;;BA)(A;;0x80120089;;;NS)"

    $key.MachineContext = 1

    $key.Create()

    $serverauthoid = New-Object -COM "[X509Enrollment.CObjectId.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CObjectId.1)"

    $serverauthoid.InitializeFromValue("[1.3.6.1.5.5.7.3.1](https://docs.corp.hentsu.com/display/ENG/1.3.6.1.5.5.7.3.1)")

    $ekuoids = New-Object -COM "[X509Enrollment.CObjectIds.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CObjectIds.1)"

    $ekuoids.Add($serverauthoid)

    $ekuext = New-Object -COM "[X509Enrollment.CX509ExtensionEnhancedKeyUsage.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509ExtensionEnhancedKeyUsage.1)"

    $ekuext.InitializeEncode($ekuoids)

    $cert = New-Object -COM "[X509Enrollment.CX509CertificateRequestCertificate.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509CertificateRequestCertificate.1)"

    $cert.InitializeFromPrivateKey(2, $key, "")

    $cert.Subject = $name

    $cert.Issuer = $cert.Subject

    $cert.NotBefore = (Get-Date).AddDays(-1)

    $cert.NotAfter = $cert[.NotBefore.AddDays(](https://docs.corp.hentsu.com/display/ENG/.NotBefore.AddDays()$ValidDays)

    $SigOID = New-Object -ComObject [X509Enrollment.CObjectId](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CObjectId)

    $SigOID.InitializeFromValue(([[Security.Cryptography.Oid]](https://docs.corp.hentsu.com/display/ENG/Security.Cryptography.Oid%5d)$SignatureAlgorithm).Value)

    [string[]] $AlternativeName  += $hostnonFQDN

    $AlternativeName += $hostFQDN

    $IAlternativeNames = New-Object -ComObject [X509Enrollment.CAlternativeNames](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CAlternativeNames)

    foreach ($AN in $AlternativeName)

    {

        $AltName = New-Object -ComObject [X509Enrollment.CAlternativeName](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CAlternativeName)

        $AltName.InitializeFromString(0x3,$AN)

        $IAlternativeNames.Add($AltName)

    }

    $SubjectAlternativeName = New-Object -ComObject[X509Enrollment.CX509ExtensionAlternativeNames](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509ExtensionAlternativeNames)

    $SubjectAlternativeName.InitializeEncode($IAlternativeNames)

    [String[]]$KeyUsage = ("DigitalSignature", "KeyEncipherment")

    $KeyUsageObj = New-Object -ComObject [X509Enrollment.CX509ExtensionKeyUsage](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509ExtensionKeyUsage)

    $KeyUsageObj.InitializeEncode([int][[Security.Cryptography.X509Certificates.X509KeyUsageFlags](](https://docs.corp.hentsu.com/display/ENG/Security.Cryptography.X509Certificates.X509KeyUsageFlags%5d()$KeyUsage))

    $KeyUsageObj.Critical = $true

    $cert[.X509Extensions.Add(](https://docs.corp.hentsu.com/display/ENG/.X509Extensions.Add()$KeyUsageObj)

    $cert[.X509Extensions.Add(](https://docs.corp.hentsu.com/display/ENG/.X509Extensions.Add()$ekuext)

    $cert[.SignatureInformation.HashAlgorithm](https://docs.corp.hentsu.com/display/ENG/.SignatureInformation.HashAlgorithm) = $SigOID

    $CERT[.X509Extensions.Add(](https://docs.corp.hentsu.com/display/ENG/.X509Extensions.Add()$SubjectAlternativeName)

    $cert.Encode()

    $enrollment = New-Object -COM "[X509Enrollment.CX509Enrollment.1](https://docs.corp.hentsu.com/display/ENG/X509Enrollment.CX509Enrollment.1)"

    $enrollment.InitializeFromRequest($cert)

    $certdata = $enrollment.CreateRequest(0)

    $enrollment.InstallResponse(2, $certdata, 0, "")

    # extract/return the thumbprint from the generated cert

    $parsed\_cert = New-Object[System.Security.Cryptography.X509Certificates.X509Certificate2](https://docs.corp.hentsu.com/display/ENG/System.Security.Cryptography.X509Certificates.X509Certificate2)

    $[parsed\_cert.Import([System.Text.Encoding]::UTF8.GetBytes(](https://docs.corp.hentsu.com/display/ENG/parsed_cert.Import(%5bSystem.Text.Encoding%5d::UTF8.GetBytes()$certdata))

    return $[parsed\_cert.Thumbprint](https://docs.corp.hentsu.com/display/ENG/parsed_cert.Thumbprint)

}

Function Enable-GlobalHttpFirewallAccess

{

    Write-Verbose "Forcing global HTTP firewall access"

    # this is a fairly naive implementation; could be more sophisticated about rule matching/collapsing

    $fw = New-Object -ComObject [HNetCfg.FWPolicy2](https://docs.corp.hentsu.com/display/ENG/HNetCfg.FWPolicy2)

    # try to find/enable the default rule first

    $add\_rule = $false

    $matching\_rules = $fw.Rules | ? { $[\_.Name](https://docs.corp.hentsu.com/display/ENG/_.Name) -eq "Windows Remote Management (HTTP-In)" }

    $rule = $null

    If ($matching\_rules) {

        If ($matching\_rules -isnot [Array]) {

            Write-Verbose "Editing existing single HTTP firewall rule"

            $rule = $matching\_rules

        }

        Else {

            # try to find one with the All or Public profile first

            Write-Verbose "Found multiple existing HTTP firewall [rules...](https://docs.corp.hentsu.com/display/ENG/rules...)"

            $rule = $matching\_rules | % { $[\_.Profiles](https://docs.corp.hentsu.com/display/ENG/_.Profiles) -band 4 }[0]

            If (-not $rule -or $rule -is [Array]) {

                Write-Verbose "Editing an arbitrary single HTTP firewall rule (multiple existed)"

                # oh well, just pick the first one

                $rule = $matching\_rules[0]

            }

        }

    }

    If (-not $rule) {

        Write-Verbose "Creating a new HTTP firewall rule"

        $rule = New-Object -ComObject [HNetCfg.FWRule](https://docs.corp.hentsu.com/display/ENG/HNetCfg.FWRule)

        $rule.Name = "Windows Remote Management (HTTP-In)"

        $rule.Description = "Inbound rule for Windows Remote Management via WS-Management. [TCP 5985]"

        $add\_rule = $true

    }

    $rule.Profiles = 0x7FFFFFFF

    $rule.Protocol = 6

    $rule.LocalPorts = 5985

    $rule.RemotePorts = "\*"

    $rule.LocalAddresses = "\*"

    $rule.RemoteAddresses = "\*"

    $rule.Enabled = $true

    $rule.Direction = 1

    $rule.Action = 1

    $rule.Grouping = "Windows Remote Management"

    If ($add\_rule) {

        $fw[.Rules.Add(](https://docs.corp.hentsu.com/display/ENG/.Rules.Add()$rule)

    }

    Write-Verbose "HTTP firewall rule $($[rule.Name)](https://docs.corp.hentsu.com/display/ENG/rule.Name)) updated"

}

# Setup error handling.

Trap

{

    $\_

    Exit 1

}

$ErrorActionPreference = "Stop"

# Get the ID and security principal of the current user account

$myWindowsID=[[System.Security.Principal.WindowsIdentity]::GetCurrent()](https://docs.corp.hentsu.com/display/ENG/System.Security.Principal.WindowsIdentity%5d::GetCurrent())

$myWindowsPrincipal=new-object[System.Security.Principal.WindowsPrincipal(](https://docs.corp.hentsu.com/display/ENG/System.Security.Principal.WindowsPrincipal()$myWindowsID)

# Get the security principal for the Administrator role

$adminRole=[[System.Security.Principal.WindowsBuiltInRole]::Administrator](https://docs.corp.hentsu.com/display/ENG/System.Security.Principal.WindowsBuiltInRole%5d::Administrator)

# Check to see if we are currently running "as Administrator"

if (-Not $myWindowsPrincipal.IsInRole($adminRole))

{

    Write-Output "ERROR: You need elevated Administrator privileges in order to run this script."

    Write-Output "       Start Windows PowerShell by using the Run as Administrator option."

    Exit 2

}

$EventSource = $MyInvocation[.MyCommand.Name](https://docs.corp.hentsu.com/display/ENG/.MyCommand.Name)

If (-Not $EventSource)

{

    $EventSource = "Powershell CLI"

}

If ([[System.Diagnostics.EventLog]::Exists(](https://docs.corp.hentsu.com/display/ENG/System.Diagnostics.EventLog%5d::Exists()'Application') -eq $False -or[[System.Diagnostics.EventLog]::SourceExists(](https://docs.corp.hentsu.com/display/ENG/System.Diagnostics.EventLog%5d::SourceExists()$EventSource) -eq $False)

{

    New-EventLog -LogName Application -Source $EventSource

}

# Detect PowerShell version.

If ($PSVersionTable[.PSVersion.Major](https://docs.corp.hentsu.com/display/ENG/.PSVersion.Major) -lt 3)

{

    Write-Log "PowerShell version 3 or higher is required."

    Throw "PowerShell version 3 or higher is required."

}

# Find and start the WinRM service.

Write-Verbose "Verifying WinRM service."

If (!(Get-Service "WinRM"))

{

    Write-Log "Unable to find the WinRM service."

    Throw "Unable to find the WinRM service."

}

ElseIf ((Get-Service "WinRM").Status -ne "Running")

{

    Write-Verbose "Setting WinRM service to start automatically on boot."

    Set-Service -Name "WinRM" -StartupType Automatic

    Write-Log "Set WinRM service to start automatically on boot."

    Write-Verbose "Starting WinRM service."

    Start-Service -Name "WinRM" -ErrorAction Stop

    Write-Log "Started WinRM service."

}

# WinRM should be running; check that we have a PS session config.

If (!(Get-PSSessionConfiguration -Verbose:$false) -or (!(Get-ChildItemWSMan:\localhost\Listener)))

{

  If ($SkipNetworkProfileCheck) {

    Write-Verbose "Enabling PS Remoting without checking Network profile."

    Enable-PSRemoting -SkipNetworkProfileCheck -Force -ErrorAction Stop

    Write-Log "Enabled PS Remoting without checking Network profile."

  }

  Else {

    Write-Verbose "Enabling PS Remoting."

    Enable-PSRemoting -Force -ErrorAction Stop

    Write-Log "Enabled PS Remoting."

  }

}

Else

{

    Write-Verbose "PS Remoting is already enabled."

}

# Ensure LocalAccountTokenFilterPolicy is set to 1

# <https://github.com/ansible/ansible/issues/42978>

$token\_path = "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"

$token\_prop\_name = "LocalAccountTokenFilterPolicy"

$token\_key = Get-Item -Path $token\_path

$token\_value = $[token\_key.GetValue($token\_prop\_name,](https://docs.corp.hentsu.com/display/ENG/token_key.GetValue($token_prop_name,) $null)

if ($token\_value -ne 1) {

    Write-Verbose "Setting LocalAccountTOkenFilterPolicy to 1"

    if ($null -ne $token\_value) {

        Remove-ItemProperty -Path $token\_path -Name $token\_prop\_name

    }

    New-ItemProperty -Path $token\_path -Name $token\_prop\_name -Value 1 -PropertyType DWORD > $null

}

# Make sure there is a SSL listener.

$listeners = Get-ChildItem WSMan:\localhost\Listener

If (!($listeners | Where {$[\_.Keys](https://docs.corp.hentsu.com/display/ENG/_.Keys) -like "TRANSPORT=HTTPS"}))

{

    # We cannot use New-SelfSignedCertificate on 2012R2 and earlier

    $thumbprint = New-LegacySelfSignedCert -SubjectName $SubjectName -ValidDays$CertValidityDays

    Write-HostLog "Self-signed SSL certificate generated; thumbprint: $thumbprint"

    # Create the hashtables of settings to be used.

    $valueset = @{

        Hostname = $SubjectName

        CertificateThumbprint = $thumbprint

    }

    $selectorset = @{

        Transport = "HTTPS"

        Address = "\*"

    }

    Write-Verbose "Enabling SSL listener."

    New-WSManInstance -ResourceURI 'winrm/config/Listener' -SelectorSet$selectorset -ValueSet $valueset

    Write-Log "Enabled SSL listener."

}

Else

{

    Write-Verbose "SSL listener is already active."

    # Force a new SSL cert on Listener if the $ForceNewSSLCert

    If ($ForceNewSSLCert)

    {

        # We cannot use New-SelfSignedCertificate on 2012R2 and earlier

        $thumbprint = New-LegacySelfSignedCert -SubjectName $SubjectName -ValidDays$CertValidityDays

        Write-HostLog "Self-signed SSL certificate generated; thumbprint: $thumbprint"

        $valueset = @{

            CertificateThumbprint = $thumbprint

            Hostname = $SubjectName

        }

        # Delete the listener for SSL

        $selectorset = @{

            Address = "\*"

            Transport = "HTTPS"

        }

        Remove-WSManInstance -ResourceURI 'winrm/config/Listener' -SelectorSet$selectorset

        # Add new Listener with new SSL cert

        New-WSManInstance -ResourceURI 'winrm/config/Listener' -SelectorSet$selectorset -ValueSet $valueset

    }

}

# Check for basic authentication.

$basicAuthSetting = Get-ChildItem WSMan:\localhost\Service\Auth | Where-Object{$[\_.Name](https://docs.corp.hentsu.com/display/ENG/_.Name) -eq "Basic"}

If ($DisableBasicAuth)

{

    If (($basicAuthSetting.Value) -eq $true)

    {

        Write-Verbose "Disabling basic auth support."

        Set-Item -Path "WSMan:\localhost\Service\Auth\Basic" -Value $false

        Write-Log "Disabled basic auth support."

    }

    Else

    {

        Write-Verbose "Basic auth is already disabled."

    }

}

Else

{

    If (($basicAuthSetting.Value) -eq $false)

    {

        Write-Verbose "Enabling basic auth support."

        Set-Item -Path "WSMan:\localhost\Service\Auth\Basic" -Value $true

        Write-Log "Enabled basic auth support."

    }

    Else

    {

        Write-Verbose "Basic auth is already enabled."

    }

}

# If EnableCredSSP if set to true

If ($EnableCredSSP)

{

    # Check for CredSSP authentication

    $credsspAuthSetting = Get-ChildItem WSMan:\localhost\Service\Auth | Where{$[\_.Name](https://docs.corp.hentsu.com/display/ENG/_.Name) -eq "CredSSP"}

    If (($credsspAuthSetting.Value) -eq $false)

    {

        Write-Verbose "Enabling CredSSP auth support."

        Enable-WSManCredSSP -role server -Force

        Write-Log "Enabled CredSSP auth support."

    }

}

If ($GlobalHttpFirewallAccess) {

    Enable-GlobalHttpFirewallAccess

}

# Configure firewall to allow WinRM HTTPS connections.

$fwtest1 = netsh advfirewall firewall show rule name="Allow WinRM HTTPS"

$fwtest2 = netsh advfirewall firewall show rule name="Allow WinRM HTTPS"profile=any

If ($fwtest1.count -lt 5)

{

    Write-Verbose "Adding firewall rule to allow WinRM HTTPS."

    netsh advfirewall firewall add rule profile=any name="Allow WinRM HTTPS"dir=in localport=5986 protocol=TCP action=allow

    Write-Log "Added firewall rule to allow WinRM HTTPS."

}

ElseIf (($fwtest1.count -ge 5) -and ($fwtest2.count -lt 5))

{

    Write-Verbose "Updating firewall rule to allow WinRM HTTPS for any profile."

    netsh advfirewall firewall set rule name="Allow WinRM HTTPS" new profile=any

    Write-Log "Updated firewall rule to allow WinRM HTTPS for any profile."

}

Else

{

    Write-Verbose "Firewall rule already exists to allow WinRM HTTPS."

}

# Test a remoting connection to localhost, which should work.

$httpResult = Invoke-Command -ComputerName "localhost" -ScriptBlock{$env:COMPUTERNAME} -ErrorVariable httpError -ErrorAction SilentlyContinue

$httpsOptions = New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck

$httpsResult = New-PSSession -UseSSL -ComputerName "localhost" -SessionOption$httpsOptions -ErrorVariable httpsError -ErrorAction SilentlyContinue

If ($httpResult -and $httpsResult)

{

    Write-Verbose "HTTP: Enabled | HTTPS: Enabled"

}

ElseIf ($httpsResult -and !$httpResult)

{

    Write-Verbose "HTTP: Disabled | HTTPS: Enabled"

}

ElseIf ($httpResult -and !$httpsResult)

{

    Write-Verbose "HTTP: Enabled | HTTPS: Disabled"

}

Else

{

    Write-Log "Unable to establish an HTTP or HTTPS remoting session."

    Throw "Unable to establish an HTTP or HTTPS remoting session."

}

Write-VerboseLog "PS Remoting has been successfully configured for Ansible."

On Ansible create file /etc/ansible/group\_vars/windows

---

ansible\_user: Administrator

ansible\_password: Pass

ansible\_connection: winrm

ansible\_port: 5986

ansible\_winrm\_server\_cert\_validation: ignore

This file can be encrypted, exporting ansible vault password file location to environmental variable will enable automatic decryption without typing password each time when running ansible

ansible-vault encrypt windows

echo 'Pass'>/opt/ansiblepass/pass

vi /etc/environment

export ANSIBLE\_VAULT\_PASSWORD\_FILE=/opt/ansiblepass/pass

Test connection:

ansible winserver -m win\_ping

