# Setting up a Windows Host

For Ansible to communicate to a Windows host and use Windows modules, the Windows host must meet these requirements:

- Ansible can generally manage Windows versions under current and extended support from Microsoft. Ansible can manage desktop OSs including Windows 7, 8.1, and 10, and server OSs including Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, and 2019.
- Ansible requires PowerShell 3.0 or newer and at least .NET 4.0 to be installed on the Windows host.
- A WinRM listener should be created and activated. More details for this can be found below.

Note

While these are the base requirements for Ansible connectivity, some Ansible modules have additional requirements, such as a newer OS or PowerShell version. Please consult the module's documentation page to determine whether a host meets those requirements.

### Upgrading PowerShell and .NET Framework

Ansible requires PowerShell version 3.0 and .NET Framework 4.0 or newer to function on older operating systems like Server 2008 and Windows 7. The base image does not meet this requirement. You can use the Upgrade-PowerShell.ps1 script to update these.

This is an example of how to run this script from PowerShell:

```
$url =
"https://raw.githubusercontent.com/jborean93/ansible-windows/master/scri
pts/Upgrade-PowerShell.ps1"
$file = "$env:temp\Upgrade-PowerShell.ps1"
$username = "Administrator"
$password = "Password"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force

# Version can be 3.0, 4.0 or 5.1
&$file -Version 5.1 -Username $username -Password $password -Verbose
```

Once completed, you will need to remove auto logon and set the execution policy back to the default of `Restricted`. You can do this with the following PowerShell commands:

```
# This isn't needed but is a good security practice to complete
Set-ExecutionPolicy -ExecutionPolicy Restricted -Force

$reg_winlogon_path = "HKLM:\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon"
Set-ItemProperty -Path $reg_winlogon_path -Name AutoAdminLogon -Value 0
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultUserName
-ErrorAction SilentlyContinue
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultPassword
-ErrorAction SilentlyContinue
```

The script works by checking to see what programs need to be installed (such as .NET Framework 4.5.2) and what PowerShell version is required. If a reboot is required and the `username` and `password` parameters are set, the script will automatically reboot and logon when it comes back up from the reboot. The script will continue until no more actions are required and the PowerShell version matches the target version. If the `username` and `password` parameters are not set, the script will prompt the user to manually reboot and logon when required. When the user is next logged in, the script will continue where it left off and the process continues until no more actions are required.

**Note**

If running on Server 2008, then SP2 must be installed. If running on Server 2008 R2 or Windows 7, then SP1 must be installed.

**Note**

Windows Server 2008 can only install PowerShell 3.0; specifying a newer version will result in the script failing.

**Note**

The `username` and `password` parameters are stored in plain text in the registry. Make sure the cleanup commands are run after the script finishes to ensure no credentials are still stored on the host.

### WinRM Memory Hotfix

When running on PowerShell v3.0, there is a bug with the WinRM service that limits the amount of memory available to WinRM. Without this hotfix installed, Ansible will fail to execute certain commands on the Windows host. These hotfixes should installed as part of the system bootstrapping or imaging process. The script Install-WMF3Hotfix.ps1 can be used to install the hotfix on affected hosts.

The following PowerShell command will install the hotfix:

```
$url =
"https://raw.githubusercontent.com/jborean93/ansible-windows/master/scri
pts/Install-WMF3Hotfix.ps1"
$file = "$env:temp\Install-WMF3Hotfix.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
powershell.exe -ExecutionPolicy ByPass -File $file -Verbose
```

### WinRM Setup

Once Powershell has been upgraded to at least version 3.0, the final step is for the WinRM service to be configured so that Ansible can connect to it. There are two main components of the WinRM service that governs how Ansible can interface with the Windows host: the `listener` and the `service` configuration settings.

Details about each component can be read below, but the script ConfigureRemotingForAnsible.ps1 can be used to set up the basics. This script sets up both HTTP and HTTPS listeners with a self-signed certificate and enables the `Basic` authentication option on the service.

To use this script, run the following in PowerShell:

```
$url =
"https://raw.githubusercontent.com/ansible/ansible/devel/examples/script
s/ConfigureRemotingForAnsible.ps1"
$file = "$env:temp\ConfigureRemotingForAnsible.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)

powershell.exe -ExecutionPolicy ByPass -File $file
```

There are different switches and parameters (like

- EnableCredSSP

and

- ForceNewSSLCert

) that can be set alongside this script. The documentation for these options are located at the top of the script itself.

Note

The ConfigureRemotingForAnsible.ps1 script is intended for training and development purposes only and should not be used in a production environment, since it enables settings (like `Basic` authentication) that can be inherently insecure.

### WinRM Listener

The WinRM services listens for requests on one or more ports. Each of these ports must have a listener created and configured.

To view the current listeners that are running on the WinRM service, run the following command:

```
winrm enumerate winrm/config/Listener
```

This will output something like:

```
Listener
    Address = *
    Transport = HTTP
    Port = 5985
    Hostname
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1,
fe80::5efe:10.0.2.15%6, fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3,
fe80::e8ea:d765:2c69:7756%7

Listener
    Address = *
    Transport = HTTPS
    Port = 5986
    Hostname = SERVER2016
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint = E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1,
fe80::5efe:10.0.2.15%6, fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3,
fe80::e8ea:d765:2c69:7756%7
```

In the example above there are two listeners activated; one is listening on port 5985 over HTTP and the other is listening on port 5986 over HTTPS. Some of the key options that are useful to understand are:

- `Transport`: Whether the listener is run over HTTP or HTTPS, it is recommended to use a listener over HTTPS as the data is encrypted without any further changes required.
- `Port`: The port the listener runs on, by default it is `5985` for HTTP and `5986` for HTTPS. This port can be changed to whatever is required and corresponds to the host var `ansible_port`.
- `URLPrefix`: The URL prefix to listen on, by default it is `wsman`. If this is changed, the host var `ansible_winrm_path` must be set to the same value.
- `CertificateThumbprint`: If running over an HTTPS listener, this is the thumbprint of the certificate in the Windows Certificate Store that is used in the connection. To get the details of the certificate itself, run this command with the relevant certificate thumbprint in PowerShell:

```
$thumbprint = "E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
Get-ChildItem -Path cert:\LocalMachine\My -Recurse | Where-Object {
$_.Thumbprint -eq $thumbprint } | Select-Object *
```

## Setup WinRM Listener

There are three ways to set up a WinRM listener:

- Using `winrm quickconfig` for HTTP or `winrm quickconfig -transport:https` for HTTPS. This is the easiest option to use when running outside of a domain environment and a simple listener is required. Unlike the other options, this process also has the added benefit of opening up the Firewall for the ports required and starts the WinRM service.
- Using Group Policy Objects. This is the best way to create a listener when the host is a member of a domain because the configuration is done automatically without any user input. For more information on group policy objects, see the Group Policy Objects documentation.
- Using PowerShell to create the listener with a specific configuration. This can be done by running the following PowerShell commands:

```
$selector_set = @{
    Address = "*"
    Transport = "HTTPS"
}
$value_set = @{
    CertificateThumbprint =
"E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
}

New-WSManInstance -ResourceURI "winrm/config/Listener" -SelectorSet
$selector_set -ValueSet $value_set
```

To see the other options with this PowerShell cmdlet, see New-WSManInstance.

**Note**

When creating an HTTPS listener, an existing certificate needs to be created and stored in the `LocalMachine\My` certificate store. Without a certificate being present in this store, most commands will fail.

## Delete WinRM Listener

To remove a WinRM listener:

```
# Remove all listeners
Remove-Item -Path WSMan:\localhost\Listener\* -Recurse -Force

# Only remove listeners that are run over HTTPS
Get-ChildItem -Path WSMan:\localhost\Listener | Where-Object { $_.Keys
-contains "Transport=HTTPS" } | Remove-Item -Recurse -Force
```

Note

The `Keys` object is an array of strings, so it can contain different values. By default it contains a key for `Transport=` and `Address=` which correspond to the values from winrm enumerate winrm/config/Listeners.

### WinRM Service Options

There are a number of options that can be set to control the behavior of the WinRM service component, including authentication options and memory settings.

To get an output of the current service configuration options, run the following command:

```
winrm get winrm/config/Service
winrm get winrm/config/Winrs
```

This will output something like:

```
Service
    RootSDDL =
O:NSG:BAD:P(A;;GA;;;BA)(A;;GR;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)
    MaxConcurrentOperations = 4294967295
    MaxConcurrentOperationsPerUser = 1500
    EnumerationTimeoutms = 240000
    MaxConnections = 300
    MaxPacketRetrievalTimeSeconds = 120
    AllowUnencrypted = false
    Auth
        Basic = true
        Kerberos = true
        Negotiate = true
        Certificate = true
        CredSSP = true
        CbtHardeningLevel = Relaxed
    DefaultPorts
        HTTP = 5985
        HTTPS = 5986
    IPv4Filter = *
    IPv6Filter = *
    EnableCompatibilityHttpListener = false
    EnableCompatibilityHttpsListener = false
    CertificateThumbprint
    AllowRemoteAccess = true

Winrs
    AllowRemoteShellAccess = true
    IdleTimeout = 7200000
    MaxConcurrentUsers = 2147483647
    MaxShellRunTime = 2147483647
    MaxProcessesPerShell = 2147483647
    MaxMemoryPerShellMB = 2147483647
    MaxShellsPerUser = 2147483647
```

While many of these options should rarely be changed, a few can easily impact the operations over WinRM and are useful to understand. Some of the important options are:

- `Service\AllowUnencrypted`: This option defines whether WinRM will allow traffic that is run over HTTP without message encryption. Message level encryption is only possible when `ansible_winrm_transport` is `ntlm`, `kerberos` or `credssp`. By default this is `false` and should only be set to `true` when debugging WinRM messages.
- `Service\Auth\*`: These flags define what authentication options are allowed with the WinRM service. By default, `Negotiate (NTLM)` and `Kerberos` are enabled.
- `Service\Auth\CbtHardeningLevel`: Specifies whether channel binding tokens are not verified (None), verified but not required (Relaxed), or verified and required (Strict). CBT is only used when connecting with NTLM or Kerberos over HTTPS.
- `Service\CertificateThumbprint`: This is the thumbprint of the certificate used to encrypt the TLS channel used with CredSSP authentication. By default this is empty; a self-signed certificate is generated when the WinRM service starts and is used in the TLS process.
- `Winrs\MaxShellRunTime`: This is the maximum time, in milliseconds, that a remote command is allowed to execute.
- `Winrs\MaxMemoryPerShellMB`: This is the maximum amount of memory allocated per shell, including the shell's child processes.

To modify a setting under the `Service` key in PowerShell:

```
# substitute {path} with the path to the option after
winrm/config/Service
Set-Item -Path WSMan:\localhost\Service\{path} -Value "value here"

# for example, to change Service\Auth\CbtHardeningLevel run
Set-Item -Path WSMan:\localhost\Service\Auth\CbtHardeningLevel -Value
Strict
```

To modify a setting under the `Winrs` key in PowerShell:

```
# Substitute {path} with the path to the option after winrm/config/Winrs
Set-Item -Path WSMan:\localhost\Shell\{path} -Value "value here"

# For example, to change Winrs\MaxShellRunTime run
Set-Item -Path WSMan:\localhost\Shell\MaxShellRunTime -Value 2147483647
```

Note

If running in a domain environment, some of these options are set by GPO and cannot be changed on the host itself. When a key has been configured with GPO, it contains the text `[Source="GPO"]` next to the value.

### Common WinRM Issues

Because WinRM has a wide range of configuration options, it can be difficult to setup and configure. Because of this complexity, issues that are shown by Ansible could in fact be issues with the host setup instead.

One easy way to determine whether a problem is a host issue is to run the following command from another Windows host to connect to the target Windows host:

```
# Test out HTTP
winrs -r:http://server:5985/wsman -u:Username -p:Password ipconfig

# Test out HTTPS (will fail if the cert is not verifiable)
winrs -r:https://server:5986/wsman -u:Username -p:Password -ssl ipconfig

# Test out HTTPS, ignoring certificate verification
$username = "Username"
$password = ConvertTo-SecureString -String "Password" -AsPlainText
-Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential
-ArgumentList $username, $password

$session_option = New-PSSessionOption -SkipCACheck -SkipCNCheck
-SkipRevocationCheck
Invoke-Command -ComputerName server -UseSSL -ScriptBlock { ipconfig }
-Credential $cred -SessionOption $session_option
```

If this fails, the issue is probably related to the WinRM setup. If it works, the issue may not be related to the WinRM setup; please continue reading for more troubleshooting suggestions.

### HTTP 401/Credentials Rejected

A HTTP 401 error indicates the authentication process failed during the initial connection. Some things to check for this are:

- Verify that the credentials are correct and set properly in your inventory with `ansible_user` and `ansible_password`
- Ensure that the user is a member of the local Administrators group or has been explicitly granted access (a connection test with the `winrs` command can be used to rule this out).
- Make sure that the authentication option set by `ansible_winrm_transport` is enabled under `Service\Auth\*`
- If running over HTTP and not HTTPS, use `ntlm`, `kerberos` or `credssp` with `ansible_winrm_message_encryption: auto` to enable message encryption. If using another authentication option or if the installed pywinrm version cannot be upgraded, the `Service\AllowUnencrypted` can be set to `true` but this is only recommended for troubleshooting
- Ensure the downstream packages `pywinrm`, `requests-ntlm`, `requests-kerberos`, and/or `requests-credssp` are up to date using `pip`.
- If using Kerberos authentication, ensure that `Service\Auth\CbtHardeningLevel` is not set to `Strict`.
- When using Basic or Certificate authentication, make sure that the user is a local account and not a domain account. Domain accounts do not work with Basic and Certificate authentication.

### HTTP 500 Error

These indicate an error has occurred with the WinRM service. Some things to check for include:

- Verify that the number of current open shells has not exceeded either `WinRsMaxShellsPerUser` or any of the other Winrs quotas haven't been exceeded.

### Timeout Errors

These usually indicate an error with the network connection where Ansible is unable to reach the host. Some things to check for include:

- Make sure the firewall is not set to block the configured WinRM listener ports
- Ensure that a WinRM listener is enabled on the port and path set by the host vars
- Ensure that the `winrm` service is running on the Windows host and configured for automatic start

### Connection Refused Errors

These usually indicate an error when trying to communicate with the WinRM service on the host. Some things to check for:

- Ensure that the WinRM service is up and running on the host. Use `(Get-Service -Name winrm).Status` to get the status of the

service.
- Check that the host firewall is allowing traffic over the WinRM port. By default this is `5985` for HTTP and `5986` for HTTPS.

Sometimes an installer may restart the WinRM or HTTP service and cause this error. The best way to deal with this is to use `win_psexec` from another Windows host.

## Windows SSH Setup

Ansible 2.8 has added an experimental SSH connection for Windows managed nodes.

Warning

Use this feature at your own risk! Using SSH with Windows is experimental, the implementation may make backwards incompatible changes in feature releases. The server side components can be unreliable depending on the version that is installed.

### Installing Win32-OpenSSH

The first step to using SSH with Windows is to install the Win32-OpenSSH service on the Windows host. Microsoft offers a way to install `Win32-OpenSSH` through a Windows capability but currently the version that is installed through this process is too old to work with Ansible. To install `Win32-OpenSSH` for use with Ansible, select one of these three installation options:

- Manually install the service, following the install instructions from Microsoft.
- Use `win_chocolatey` to install the service:

```
- name: install the Win32-OpenSSH service
  win_chocolatey:
    name: openssh
    package_params: /SSHServerFeature
    state: present
```

- Use an existing Ansible Galaxy role like jborean93.win_openssh:

```
# Make sure the role has been downloaded first
ansible-galaxy install jborean93.win_openssh

# main.yml
- name: install Win32-OpenSSH service
  hosts: windows
  gather_facts: no
  roles:
  - role: jborean93.win_openssh
    opt_openssh_setup_service: True
```

**Note**

`Win32-OpenSSH` is still a beta product and is constantly being updated to include new features and bugfixes. If you are using SSH as a connection option for Windows, it is highly recommend you install the latest release from one of the 3 methods above.

### Configuring the Win32-OpenSSH shell

By default `Win32-OpenSSH` will use `cmd.exe` as a shell. To configure a different shell, use an Ansible task to define the registry setting:

```
- name: set the default shell to PowerShell
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
    name: DefaultShell
    data: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
    type: string
    state: present

# Or revert the settings back to the default, cmd
- name: set the default shell to cmd
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
    name: DefaultShell
    state: absent
```

#### Win32-OpenSSH Authentication

Win32-OpenSSH authentication with Windows is similar to SSH authentication on Unix/Linux hosts. You can use a plaintext password or SSH public key authentication, add public keys to an `authorized_key` file in the `.ssh` folder of the user's profile directory, and configure the service using the `sshd_config` file used by the SSH service as you would on a Unix/Linux host.

When using SSH key authentication with Ansible, the remote session won't have access to the user's credentials and will fail when attempting to access a network resource. This is also known as the double-hop or credential delegation issue. There are two ways to work around this issue:

- Use plaintext password auth by setting `ansible_password`
- Use `become` on the task with the credentials of the user that needs access to the remote resource

#### Configuring Ansible for SSH on Windows

To configure Ansible to use SSH for Windows hosts, you must set two connection variables:

- set `ansible_connection` to `ssh`
- set `ansible_shell_type` to `cmd` or `powershell`

The `ansible_shell_type` variable should reflect the `DefaultShell` configured on the Windows host. Set to `cmd` for the default shell or set to `powershell` if the `DefaultShell` has been changed to PowerShell.

#### Known issues with SSH on Windows

Using SSH with Windows is experimental, and we expect to uncover more issues. Here are the known ones:

- Win32-OpenSSH versions older than `v7.9.0.0p1-Beta` do not work when `powershell` is the shell type
- While SCP should work, SFTP is the recommended SSH file transfer mechanism to use when copying or fetching a file