Windows Ansible Connection by Openssh

Using OpenSSH over WinRM for managing Windows servers offers several advantages:

1. **Unified Management:** With OpenSSH, you can manage both Windows and Unix-like systems using the same set of tools and protocols.
2. **Standardization:** OpenSSH is a widely-used and well-established protocol for secure remote access. By using OpenSSH for Windows management, you adhere to a widely accepted standard, which can simplify security auditing and compliance requirements.
3. **Portability:** OpenSSH is available on a wide range of platforms, including Linux, macOS, and Windows.
4. **Security:** OpenSSH provides strong encryption and authentication mechanisms, ensuring secure communication between the management system and Windows servers.

In this Document we will explore,

1. Install OpenSSH in Windows Server 2022
2. Configure Ansible to connect to Windows Server using SSH (port 22)
3. Test the connectivity from Ansible host using adhoc commands
4. Create an Ansible playbook to run few commands in Windows Server

Pre-Requisites:

1. AWS free tier account
2. Command prompt / iTerm in your local machine

Step1: Create a Amazon EC2 instance (name it as Ansible-Server) selecting Ubuntu OS with instance type as t2.micro.

Select a keypair if you have it already or create a new key pair. Configure security group with inbound rules accepting traffic from anywhere to port SSH 22. With the rest of the default details, launch the instance and name it as “Ansible-Server”.

1) **Start Linux server with putty and execute below commands.**

1)Update Linux server

$ sudo apt-get update

2) upgrades all installed packages

$ sudo apt-get upgrade –y

3) Create one project directory

$ mkdir windows_ansible

4) Install Prerequisites for ansible installation

$ sudo apt install software-properties-common

5) Add the Ansible Personal Package Archive (PPA)

$ sudo add-apt-repository --yes --update ppa:ansible/ansible

6) Install Ansible

$ sudo apt install ansible

7) Verify the Installation

$ ansible --version

8) Install ‘python3-venv’

$ sudo apt install python3-venv

9)Create a Virtual Environment

$ python3 -m venv myenv

10) Activate the Virtual Environment

$ source myenv/bin/activate

11) Install pywinrm within the Virtual Environment

$ pip install pywinrm

12) Deactivate the Virtual Environment

```
$ deactivate
```

13) Create host file inside directory

```
$ vi hostfile
```

Note: Enter below host contain in this hostfile

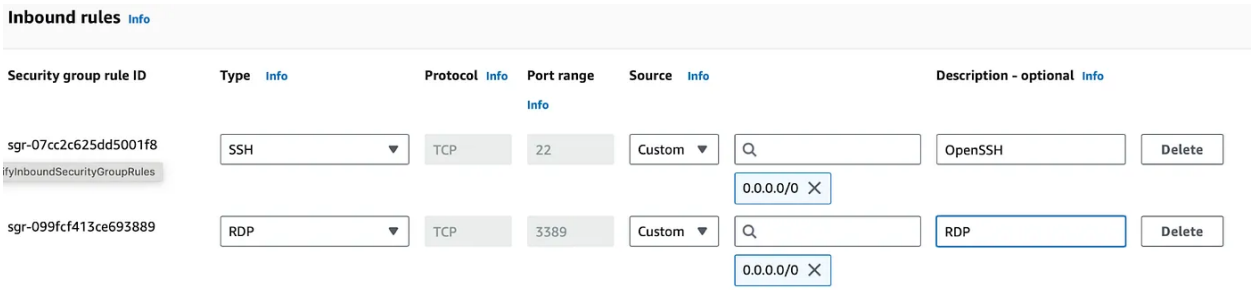
```
[win]
13.201.185.247

[win:vars]
ansible_user=Administrator
ansible_password=GCqlrOZL@@Z9JvWk9ggGv9eNX@aIvxVY
ansible_connection=ssh
ansible_shell_type=cmd
ansible_ssh_common_args=-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
ansible_ssh_retries=3
ansible_become_method=runas
```

Note: below Changes need in the host file.

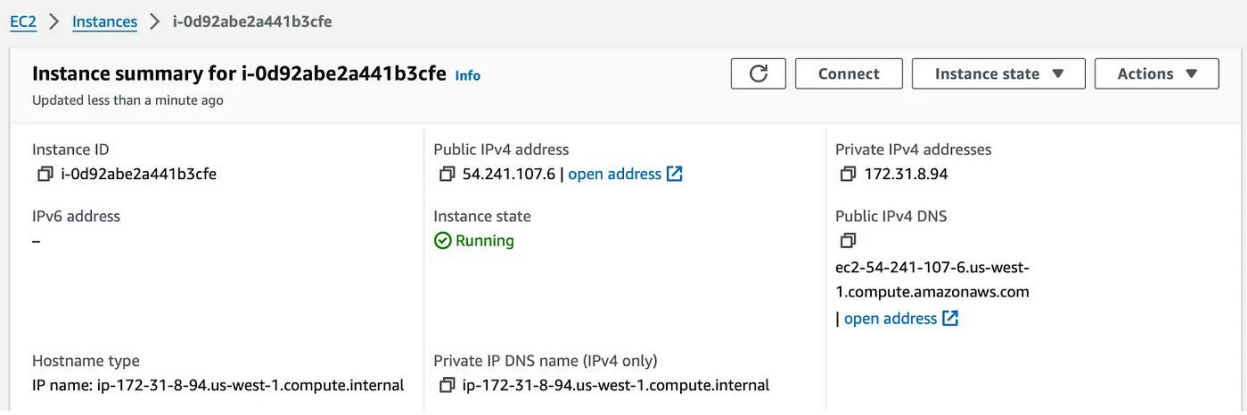
Replace ‘windows_host ansible_host’ with your windows server host Ip.

Replace ‘ansible_password’ with your windows RDP password.

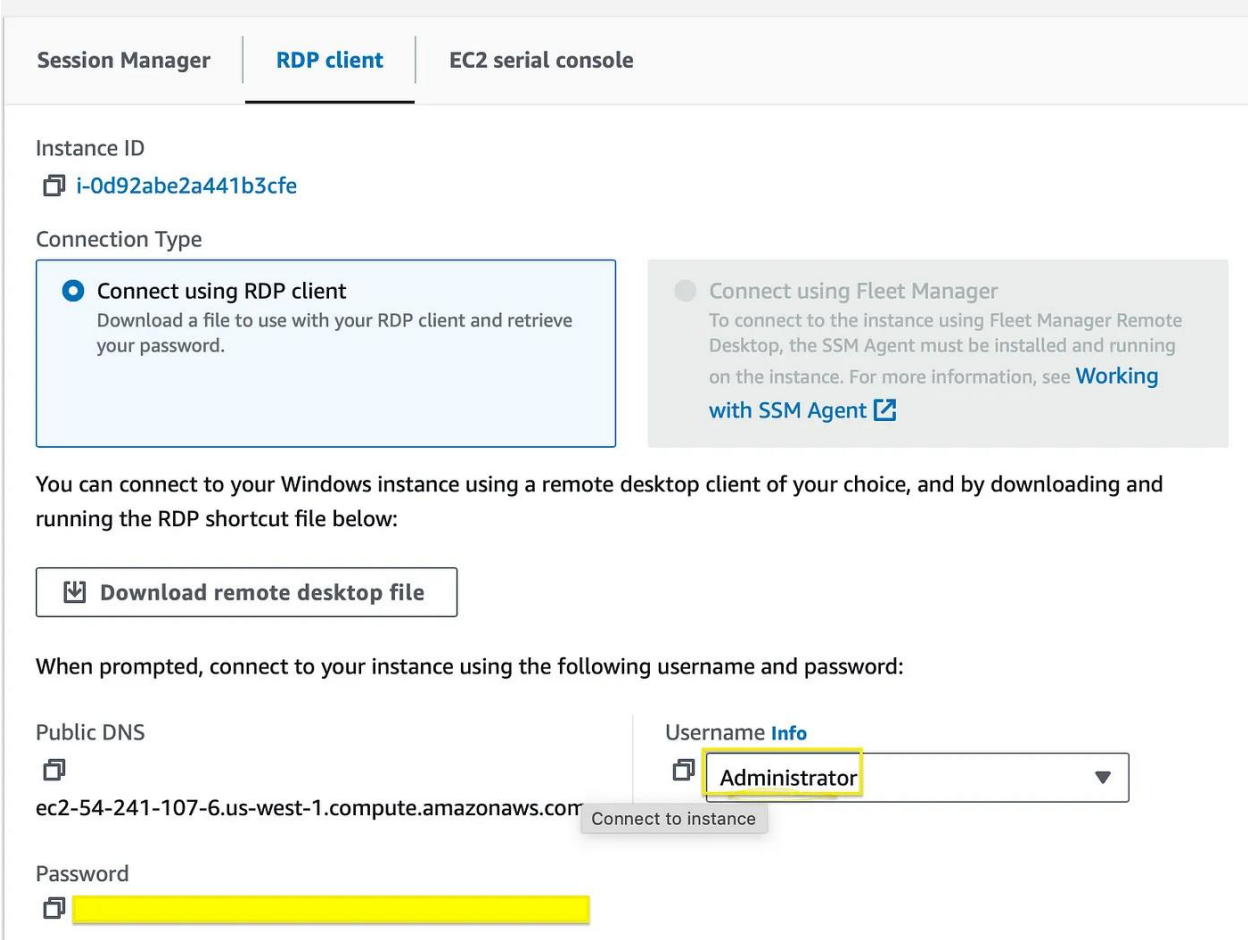
Step2. Create another Amazon EC2 instance with OS as Windows Server 2022 with instance type as t2.micro. Select the keypair and configure security group with inbound rules as shown below.



RDP allows you to login to windows server from a remote machine. Using this we will install OpenSSH
SSH enables connectivity from Ansbile host to windows server.
Launch the instance and name it as “Windows-Server”. Allow some time for the instance to start running.
Select the windows Server instance and click on Connect in the console



In the RDP client tab, with the default selection of connection type as “Connect using RDP client”, download the remote desktop file. Click on Get Password and note the password by supplying the keypair file. User to connect to this instance is “Administrator”.



Step 3: Using the Desktop file you downloaded, launch the Remote Desktop and provide the username & password information gathered in Step2. You would be able to successfully log into the Windows Server.

Step 4: Once you logged into Windows Server, open “ Windows powershell” with elevated privilege.

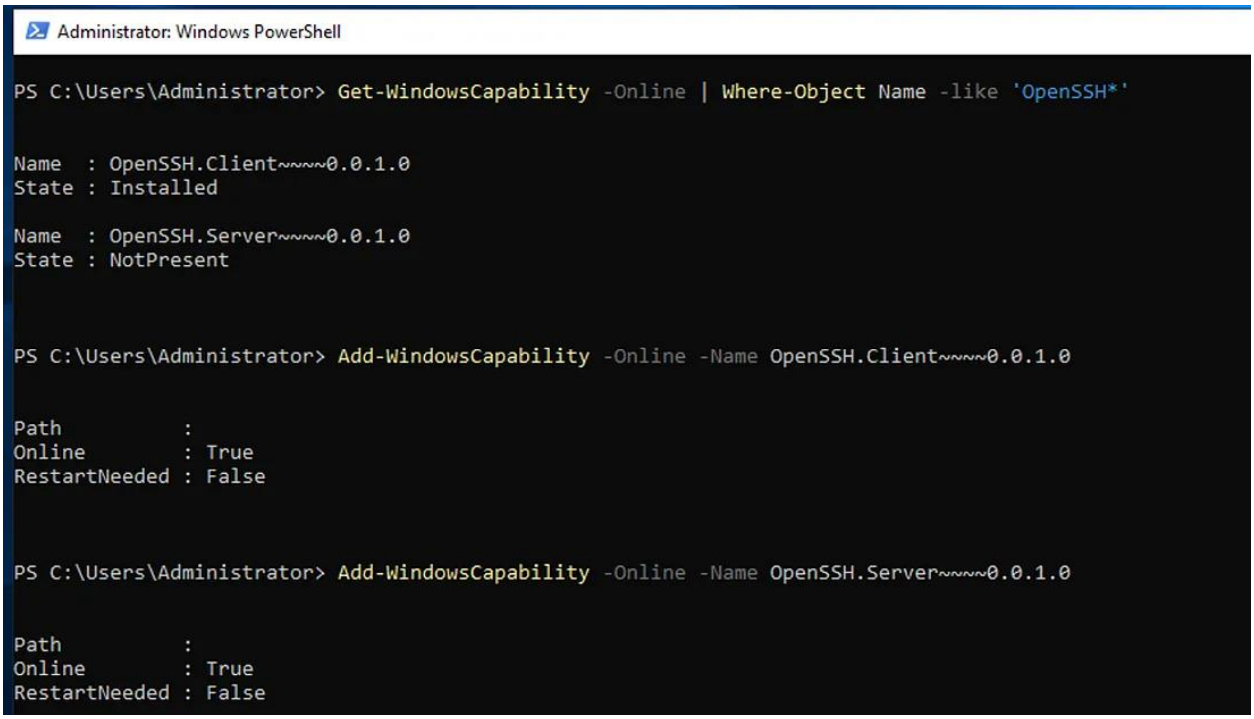
Step 5: Run the following commands in the powershell.

1. List all the OpenSSH-related capabilities
   $ Get-WindowsCapability -Online | Where-Object Name -like ‘OpenSSH*’

2. Installs the OpenSSH Client feature on the currently running Windows operating system.
   $ Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

3. Installs the OpenSSH Server feature on the currently running Windows operating system.
   $ Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0

The above commands install OpenSSH and enable OpenSSH Client and OpenSSH Server



4. Start sshd Service.
   $ Start-Service sshd

5. Set Openssh service startup type to automatic
   $ Set-Service -Name sshd -StartupType 'Automatic'

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> start-service sshd
PS C:\Users\Administrator> Set-Service -Name sshd -StartupType 'Automatic'
PS C:\Users\Administrator>
```

6. Below command helps setting the firewall for port 22.

$ if (!(Get-NetFirewallRule -Name "OpenSSH-Server-In-TCP" -ErrorAction SilentlyContinue | Select-Object Name, Enabled)) {
Write-Output "Firewall Rule 'OpenSSH-Server-In-TCP' does not exist, creating it..."
New-NetFirewallRule -Name 'OpenSSH-Server-In-TCP' -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22
} else {
Write-Output "Firewall rule 'OpenSSH-Server-In-TCP' has been created and exists."
}

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> if (!(Get-NetFirewallRule -Name "OpenSSH-Server-In-TCP" -ErrorAction SilentlyContinue | Selec
t-Object Name, Enabled)) {
>>     Write-Output "Firewall Rule 'OpenSSH-Server-In-TCP' does not exist, creating it..."
>>     New-NetFirewallRule -Name 'OpenSSH-Server-In-TCP' -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction I
nbound -Protocol TCP -Action Allow -LocalPort 22
>> } else {
>>     Write-Output "Firewall rule 'OpenSSH-Server-In-TCP' has been created and exists."
>> }
Firewall rule 'OpenSSH-Server-In-TCP' has been created and exists.
```

Step 6: Open the Linux server and execute below command for Generate SSH Key Pair (if not already done).

$ ssh-keygen

Note: After generating the SSH key pair, copy the **'`id_rsa.pub`'** file contains from the Linux server. On the Windows server, create a **'`.ssh`'** directory, then create an **'`authorized_keys`'** file inside this directory, and paste the contents of the copied public key file into the `authorized_keys` file.

Step 7: Create ansible inventory file **'hosts.ini'** in the /home/ubuntu directory as below.

```
[win]
13.201.185.247

[win:vars]
ansible_user=Administrator
ansible_password=GCqlrOZL@@Z9JvWk9ggGv9eNX@aIvxVY
ansible_connection=ssh
ansible_shell_type=cmd
ansible_ssh_common_args=-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
ansible_ssh_retries=3
ansible_become_method=runas
```

Note: `below Changes needed in the host file.`

Replace 'windows_host ansible_host' with your windows server host Ip.

Replace 'ansible_password' with your windows RDP password.

Step 8: Now it's the time to check the connectivity to windows server from Ansible host. Enter the below ansible ad hoc ping module command.

$ ansible win –i host.ini -m win_ping

```
root@ip-172-31-8-208:/home/ubuntu/Windows_Ansible_Openssh# ansible win -i hosts.ini -m win_ping
13.201.185.247 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Great ! we are now able to communicate with Windows Server.

Step9: Let's create a Ansible playbook (ansible-playbook.yml) to perform a few tasks in the Windows server from Ansible host.

A) Create an Ansible playbook on a Linux server to **create** a directory and file on windows server, then execute that playbook by using the command below.

1) Ansible playbook

$ vi test_playbook.yml

```
---
- name: Create folder and file on Windows desktop
  hosts: windows_host
  gather_facts: no
  tasks:
    - name: Create directory on desktop
      win_shell: New-Item -Path "\Desktop\NewFolder" -ItemType Directory
      become: yes
      become_user: Administrator

    - name: Create file inside the folder
      win_shell: Out-File -FilePath "\Desktop\NewFolder\example.txt" -InputObject "Hello, World!"
      become: yes
      become_user: Administrator
```

2) Execute ansible playbook

$ ansible-playbook –i hostfile test_playbook.yml

Note: After executing the playbook successfully, you can check the results on your Windows server at the specified location.

```
(myenv) root@ip-172-31-8-208:/home/ubuntu/Windows_Ansible# ansible-playbook -i hostfile test_playbook.yml

PLAY [Create folder and file on Windows Server] ***********************************************************************

TASK [Create directory] ***********************************************************************************************
changed: [windows_host]

TASK [Create file] ****************************************************************************************************
changed: [windows_host]

PLAY RECAP ************************************************************************************************************
windows_host               : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

B) Create an Ansible playbook on a Linux server to **remove** a directory and file from windows server desktop, then execute that playbook by using the command below.

1) Ansible playbook

$ vi remove_folder_file.yml

```
---
- name: Delete folder and file on Windows desktop
  hosts: 13.201.185.247
  gather_facts: no

  tasks:
    - name: Delete file inside the folder
      win_shell: Remove-Item -Path
"C:\Users\Administrator\Desktop\NewFolder\example.txt" -Force
      become: yes
      become_user: Administrator
      ignore_errors: yes  # Ignore errors if the file does not exist

    - name: Delete directory on desktop
      win_shell: Remove-Item -Path "C:\Users\Administrator\Desktop\NewFolder" -Recurse -
Force
      become: yes
      become_user: Administrator
      ignore_errors: yes  # Ignore errors if the folder does not exist
```

2) Execute ansible playbook

   $ ansible-playbook –i hostfile remove_folder_file.yml

   Note: After executing the playbook successfully, you can check the results on your Windows server at the specified location.