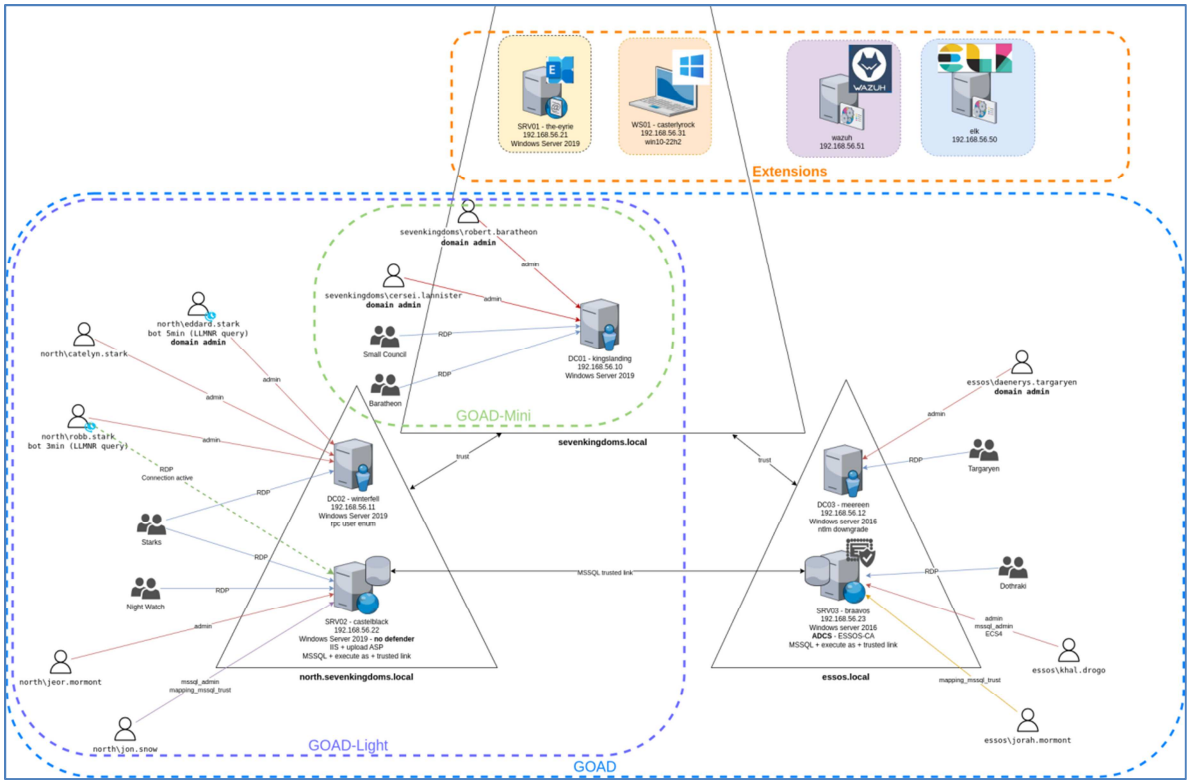
GOAD Active Directory Lab Setup from a Windows host

JJKirn 10.1.2025

This write up is to document the install of GOAD – Game of Active Directory v3 using **VMware Workstation Pro** on a **Windows 11 OS PC (host)**.



GOAD: 5 VMs, 2 forests, 3 domains (full GOAD lab)

This install requires that you have a **Windows (10 or 11) OS PC host** with **VMware Workstation Pro** (VMware-workstation-17.6.3.24583834.exe) installed and you have downloaded the following ISOs:

- **Ubuntu 20.04 Desktop**: ubuntu-22.04.5-desktop-amd64.iso
- **Kali Linux**: kali-linux-2025.2-installer-amd64.iso

SECTION OVERVIEW:

1. **Section A - Create the 5 GOAD Windows VMs.** The details in creating the 5 GOAD VMs using only vagrant are covered in this section.

2. **Section B – Provisioning.** The details in creating the Ubuntu 20.04 VM used for provisioning for all 5 GOAD VMs using only ansible are covered in this section.

3. **Section C - Attack Techniques.** The details in creating the kali VM and running our first attack command are covered in this section.

4. **Section D - Extra Information.** This section provides helpful in answering questions about the installs not covered elsewhere.

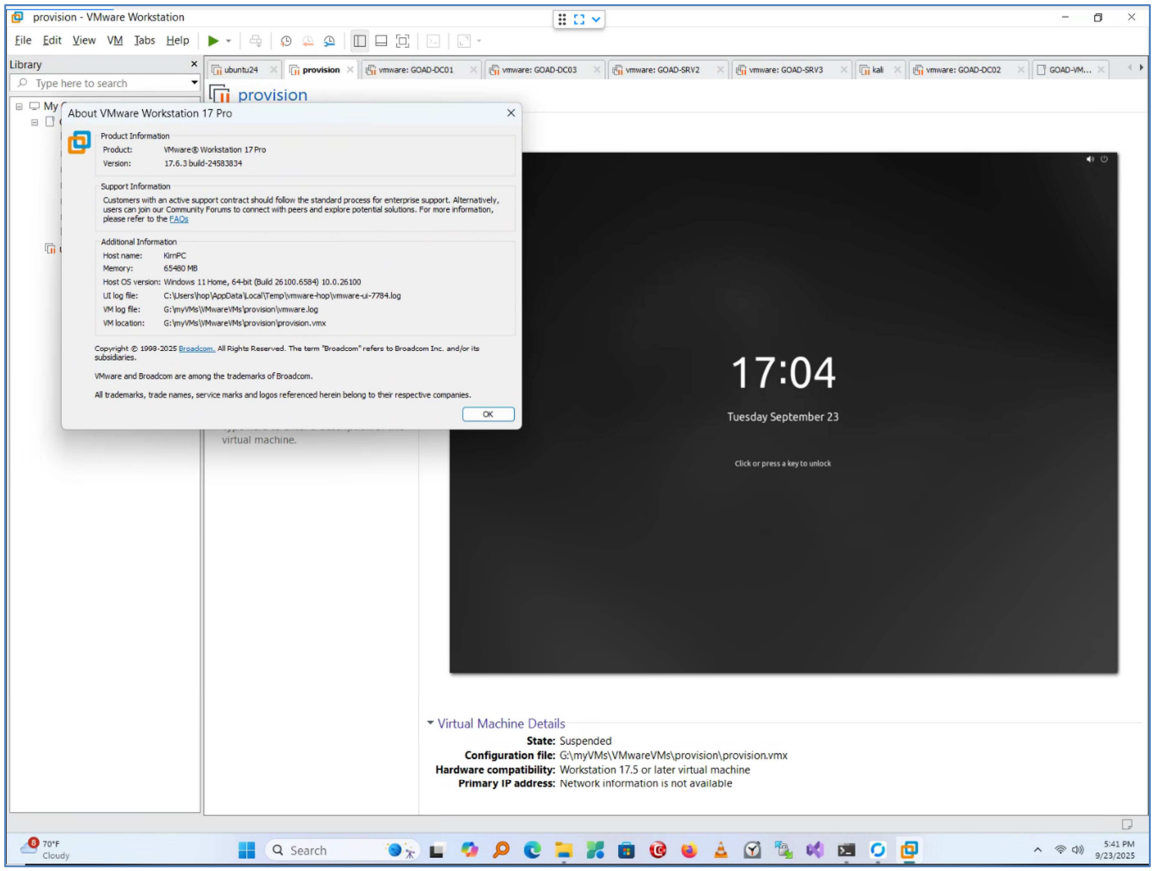
GENERAL NOTES:

1. You will use your **Window s 11 OS host** with the installed **VMware Workstation Pro** together with an install of **GOAD** (**https://github.com/Orange-Cyberdefense/GOAD.git**) to create the first environment. In this environment you will only need to run **vagrant** to create the 5 GOAD VMs. See **Section A. Create the 5 GOAD Windows VMs** below for details.

2. The **Ubuntu 22.04 Desktop VM** you create will only be used for "**provisioning**". You will also need to install **GOAD** (**https://github.com/Orange-Cyberdefense/GOAD.git**) on that VM. You will only need to run **ansible** in this environment for the **provisioning**. See section **B. Provisioning** below for details.

A. Create the 5 GOAD Windows VMs for VMware Workstation Pro

1. First step is to make sure you have **VMware Workstation Pro installed:**

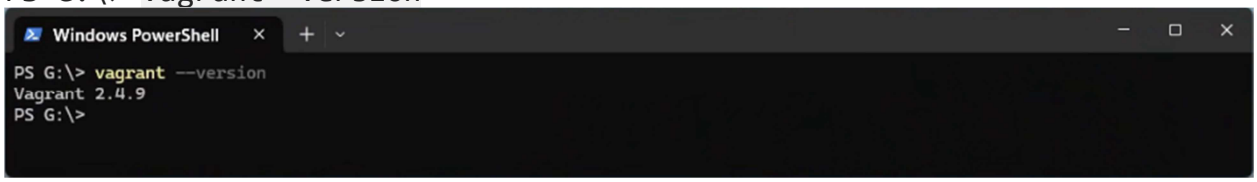


To download and install VMware Workstation Pro 17 for Windows, go to the Broadcom support portal, create an account if you don't have one, and log in and download the latest version.

2. Next you need to download and install the “Vagrant WMware Utility” for Windows.

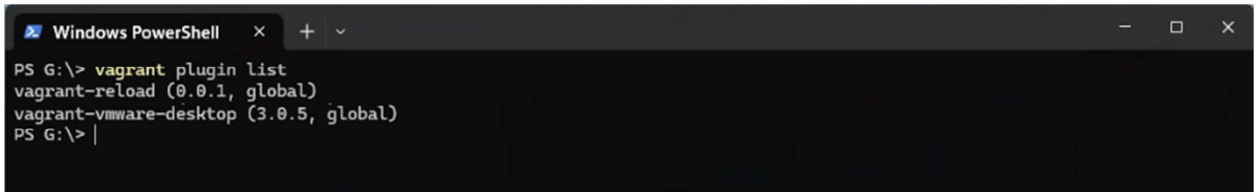
When you complete the Vagrant installation, you can check whether the installation was successful via powershell (reminder this in on the Windows 11 host):

```
PS G:\> vagrant -version
```



You also need two vagrant plugins:

```
PS G:\> vagrant plugin list
```



3. Now you need to download and install the GOAD lab and use vagrant to create the 5 Window VMs (Again, this part is done on Windows 11 host directly)

Open powershell and go to the location where you want to install GOAD:

```
PS G:\> git clone https://github.com/Orange-Cyberdefense/GOAD.git
```

Start by fixing the "Vagrantfile" to be used for the install:

```
PS G:\GOAD\ad\GOAD\providers\vmware> cat .\Vagrantfile

#--- JJK 9/22/25, https://github.com/Orange-Cyberdefense/GOAD
Vagrant.configure("2") do |config|

  ENV['VAGRANT_DEFAULT_PROVIDER'] = 'vmware_desktop'

    config.vm.boot_timeout = 600   # this stuff is from template
    config.vm.graceful_halt_timeout = 600
    config.winrm.retry_limit = 30
    config.winrm.retry_delay = 10

  boxes = [
    {
      name: "GOAD-DC01",
      ip: "192.168.56.10",
      box: "StefanScherer/windows_2019",
      box_version: "2021.05.15",
      os: "windows",
      cpus: 2,
      mem: 3000
    },
      {
      name: "GOAD-DC02",
      ip: "192.168.56.11",
      box: "StefanScherer/windows_2019",
      box_version: "2021.05.15",
      os: "windows",
      cpus: 2,
      mem: 3000
    },
      {
      name: "GOAD-DC03",
```

```
      ip: "192.168.56.12",
      box: "StefanScherer/windows_2016",
      box_version: "2017.12.14",
      os: "windows",
      cpus: 2,
      mem: 3000
    },
      {
      name: "GOAD-SRV2",
      ip: "192.168.56.22",
      box: "StefanScherer/windows_2019",
      box_version: "2021.05.15",
      os: "windows",
      cpus: 2,
      mem: 6000
    },
      {
      name: "GOAD-SRV3",
      ip: "192.168.56.23",
      box: "StefanScherer/windows_2016",
      box_version: "2019.02.14",
      os: "windows",
      cpus: 2,
      mem: 5000
#    },
#    {
#      name: "GOAD-BUILDER",  # This is if you wanted to Ubuntu server box
#      ip: "192.168.56.100",
#
#      box: "bento/ubuntu-22.04",
#      cpus: 2,
#      mem: 2048,
#      os: "linux"
    }

  ]

  boxes.each do |opts|
    config.vm.define opts[:name] do |vm|
        vm.vm.box = opts[:box]
      vm.vm.box_version = opts[:box_version]
      vm.vm.hostname = opts[:name] # This sets the VM hostname inside the OS
      # issues/49
      vm.vm.synced_folder '.', '/vagrant', disabled: true
      vm.vm.network "private_network", ip: opts[:ip]

      vm.vm.provider :vmware_desktop do |v|
         v.vmx["memsize"] = opts[:mem].to_s
         v.cpus = opts[:cpus]
         # jjk 9.18.25
```

```
         # v.force_vm_license = "workstation" #force the license for some
vagrant plugin issues
         v.vmx["ethernet0.pcislotnumber"] = "192"  # Fix warning
         v.vmx["ethernet1.pcislotnumber"] = "224"  # Fix warning
      end

      # Optional shell provisioner - JJK, from template
      if opts[:os] == "windows"
        vm.vm.guest = :windows
        vm.vm.communicator = "winrm"
        vm.vm.provision :shell, :path => "G:/GOAD/vagrant/Install-
WMF3Hotfix.ps1", privileged: false
        vm.vm.provision :shell, :path =>
"G:/GOAD/vagrant/ConfigureRemotingForAnsible.ps1", privileged: false

        # fix for vmware static ip
        vm.vm.provision :shell, :path => "G:/GOAD/vagrant/fix_ip.ps1",
privileged: true, args: opts[:ip]
        # also enable ping response
        vm.vm.provision :shell, :path => "G:/GOAD/vagrant/fix_ping.ps1",
privileged: true

      else
        vm.vm.communicator = "ssh"
      end

    end
  end
end
```

Next you need to fix the IP addresses in the "inventory" file located at the same location:

```
PS G:\GOAD\ad\GOAD\providers\vmware> cat inventory

[default]
; Note: ansible_host *MUST* be an IPv4 address or setting things like DNS
; servers will break.
; ------------------------------------------------
; sevenkingdoms.local
; ------------------------------------------------
dc01 ansible_host=192.168.10 dns_domain=dc01 dict_key=dc01
; ------------------------------------------------
; north.sevenkingdoms.local
; ------------------------------------------------
dc02 ansible_host=192.168.56.11 dns_domain=dc01 dict_key=dc02
srv02 ansible_host=192.168.56.22 dns_domain=dc02 dict_key=srv02
; ------------------------------------------------
; essos.local
```

```
; ----------------------------------------------
dc03 ansible_host=192.168.56.12 dns_domain=dc03 dict_key=dc03
srv03 ansible_host=192.168.56.23 dns_domain=dc03 dict_key=srv03
```

We are now ready to create all the 5 GOAD Windows VMs using the following powershell command at the location indicated:

```
PS G:\GOAD\ad\GOAD\providers\vmware> vagrant up
```

4. You need to check and update the IPs of all the Windows Machines:

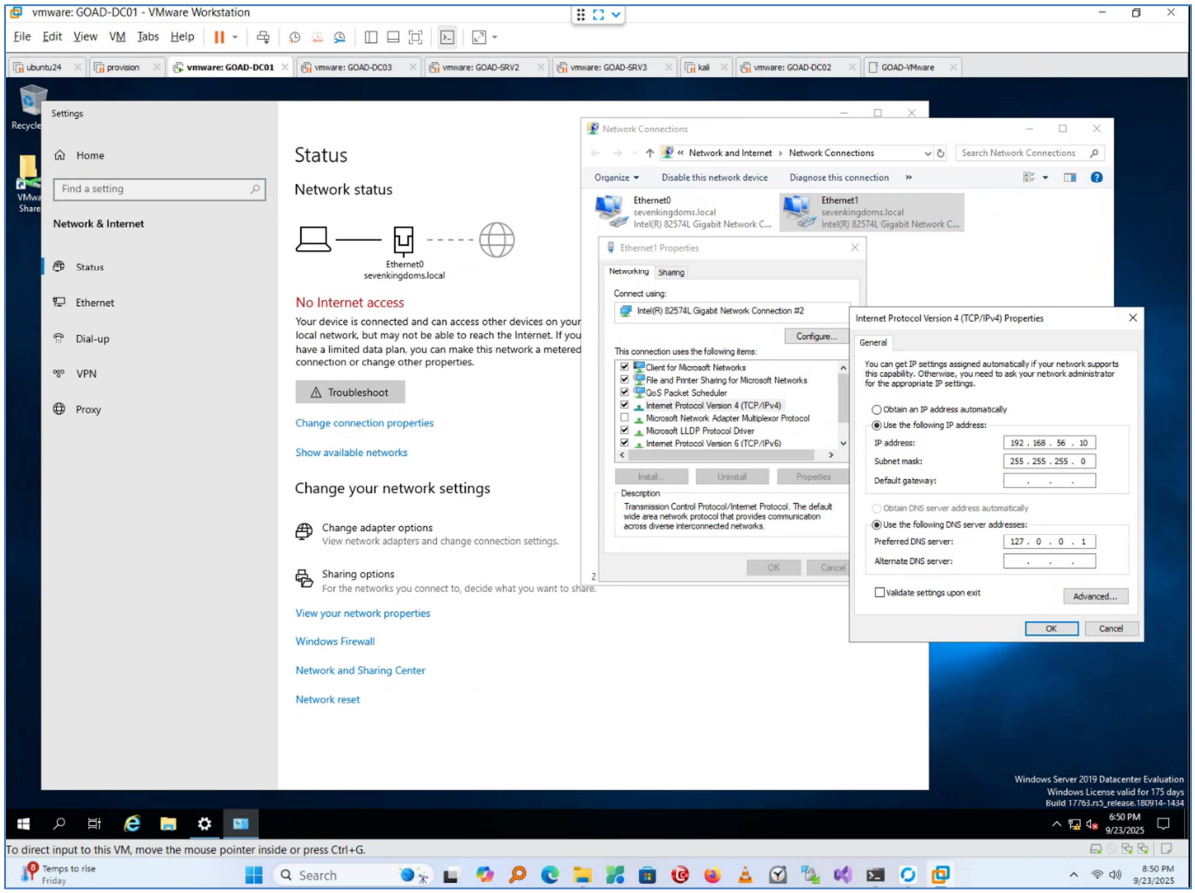
The Vagrant script is supposed to correctly "fix the IP" on Windows VMs. However for some reason it is not always successful. I observed that although the VMs were created, they didn't have the right IPs.

Several times had to manually set the IPs as follows:

Server	IP	MASK
DC1	192.168.56.10	255.255.255.0
DC2	192.168.56.11	255.255.255.0
DC3	192.168.56.12	255.255.255.0
SRV02	192.168.56.22	255.255.255.0
SRV03	192.168.56.23	255.255.255.0

I logging into each of the 5 GOAD VMs as vagrant:vagrant and made changes if necessary:

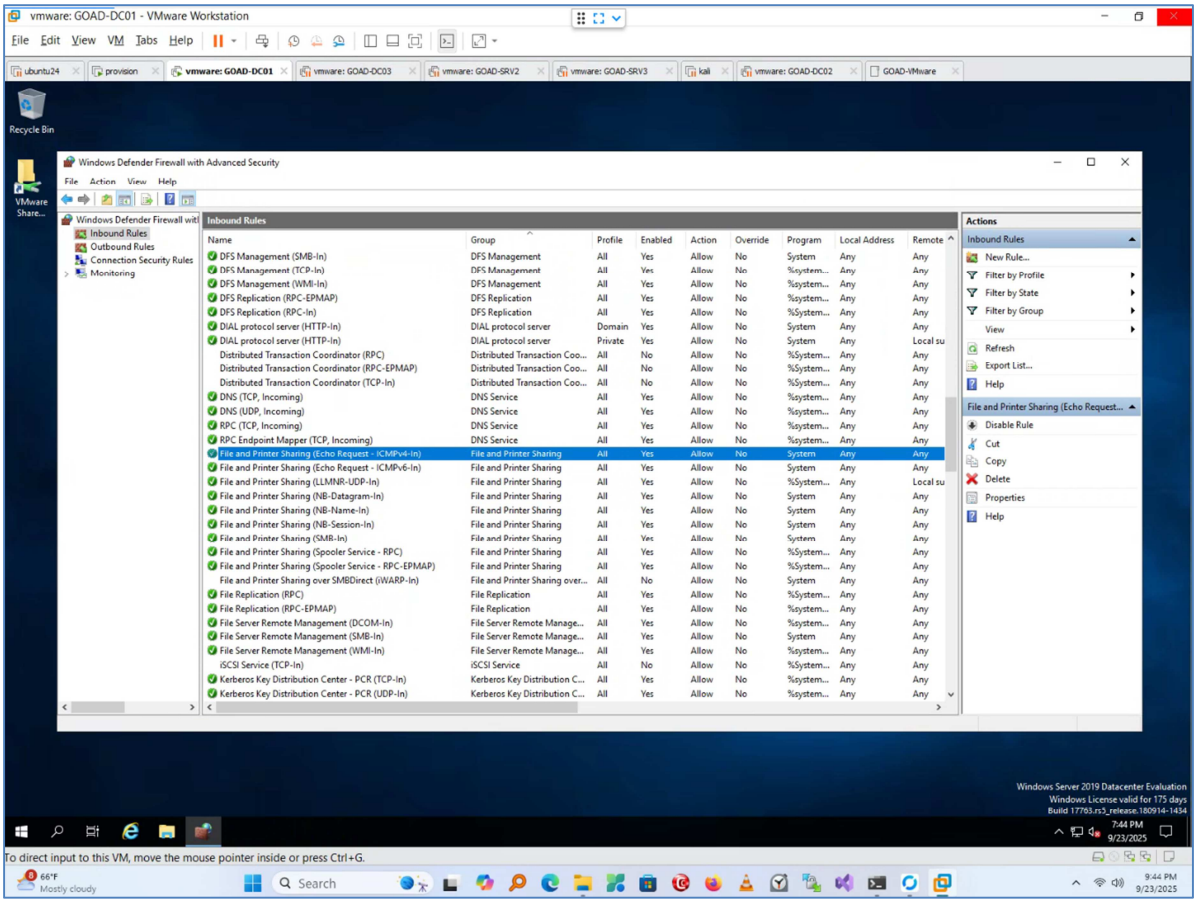
Example of DC01:



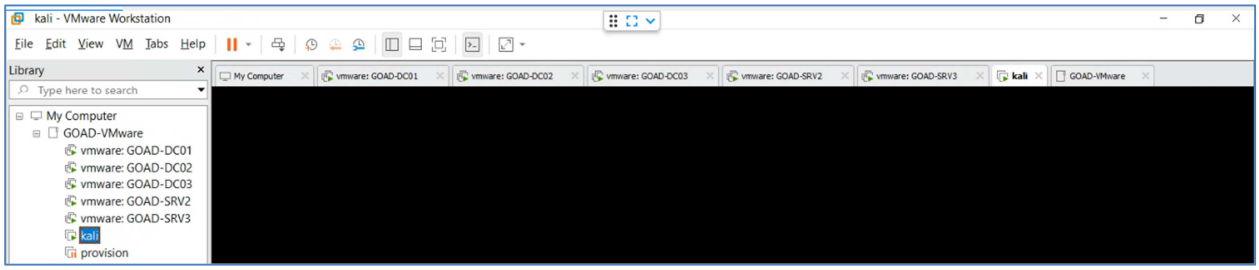
5. You also need to verify “enable ping” on all servers:

The Vagrant script is supposed to “enable ping” response on Windows VMs. However for some reason it is not always successful. To verify “enable ping” was performed correctly, open the Windows Defender Firewall with Advanced Security console, select Inbound Rules, and enable the pre-existing rule named "File and Printer Sharing (Echo Request - ICMPv4-In)" if necessary. This needs to be done on each of the 5 GOAD Windows VMs.

Example for DC01:



Assuming all goes well with `vagrant up`, you should now see the 5 newly created **GOAD VMs**:



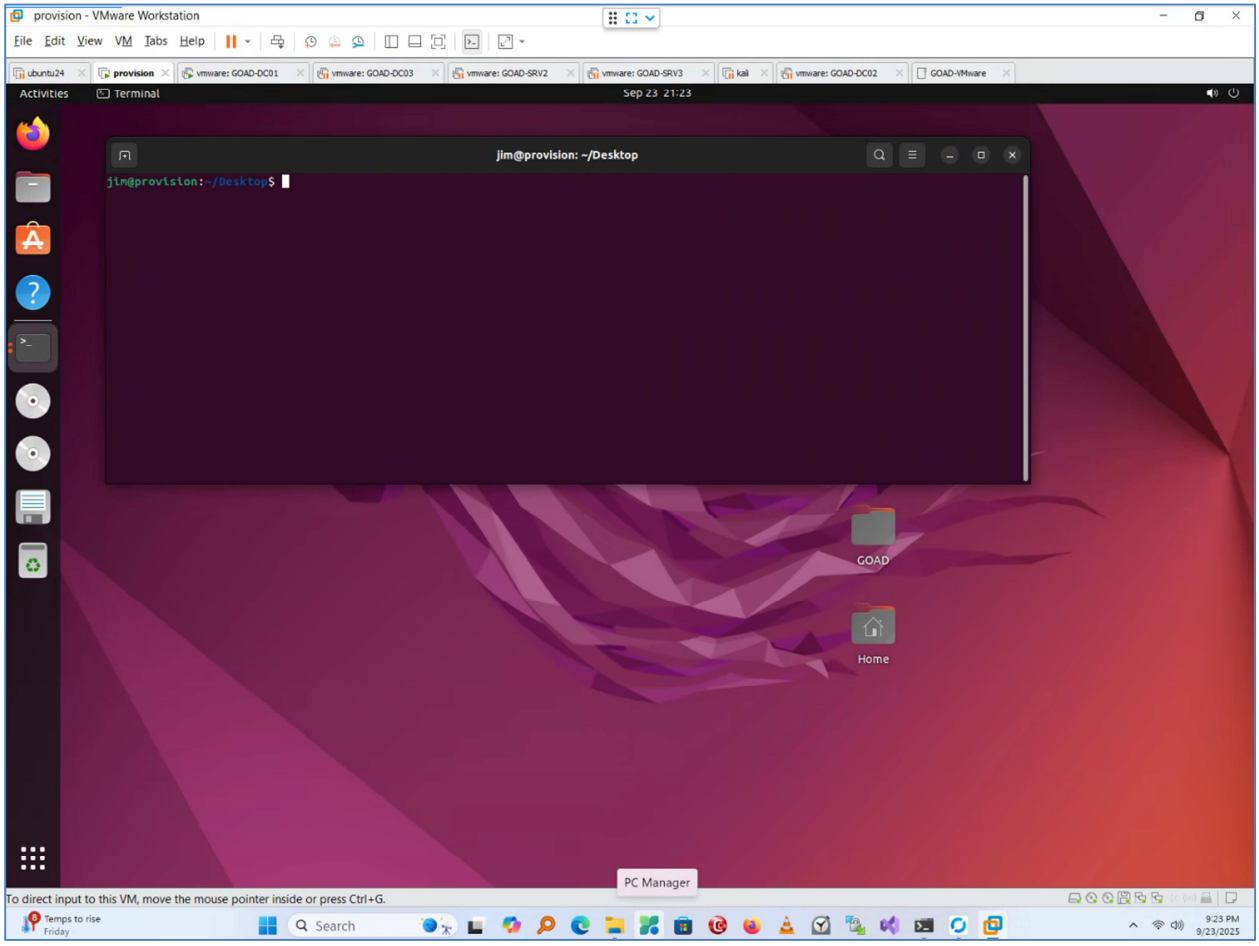
B. Provisioning (done via the Ubuntu 22.04 Desktop VM)

Great! We have created all the 5 Windows GOAD virtual machines, now we need to have a vulnerable Active Directory environment before practicing the attack techniques!

Let's switch to the Ubuntu "provision" VM that I manually created from ISO:

- RAM: 4 GB
- Processors: 2
- HD: 60GB
- VM: Ubuntu 22.04 Desktop

- Network Adapter 1: NAT
- Network Adapter 2: VMnet2
    a. Static IP 192.168.56.100



I'm not sure you need this step, but it won't hurt and it helps with ping verifications. Update the /etc/hosts file to include the IP addresses for our GOAD VMs:

```
jim@provision:~/Desktop$ cat /etc/hosts:

127.0.0.1       localhost
127.0.1.1       provision

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# GOAD IPs
192.168.56.10   dc01   DC01
192.168.56.11   dc02   DC02
```

```
192.168.56.12  dc03  DC03
192.168.56.22  srv02  SRV02
192.168.56.23  srv03  SRV03
```

Next I started the provisioning part by installing all requirements before running ansible to perform the provisioning.

To install python virtual environment:

```
sudo apt update
sudo apt upgrade

sudo apt install git
git clone http://github.com/Orange-Cyberdefense/GOAD.git

cd GOAD/ansible
```

NOTE:
Ansible is currently stable with Python versions 3.8 through 3.11. To resolve any potential issues use python3.10

```
sudo apt install python3.10-venv
python3.10 -m venv .venv

source .venv/bin/activate
```

To install ansible pywinrm in the .venv:

```
python3 -m pip install --upgrade pip
python3 -m pip install ansible-core==2.12.6
python3 -m pip install pywinrm
```

The following command is outside of the GOAD directory, so if you are copy and pasting make sure to cd to GOAD. Install all the ansible-galaxy requirements:

```
ansible-galaxy install -r ansible/requirements.yml
```

We will use the following command to run ansible playbooks for GOAD on VMware Ubuntu based provision VM. (from ./GOAD/ansible)

```
ansible-playbook -i ../ad/GOAD/data/inventory -i
../ad/GOAD/providers/vmware/inventory main.yml
```

However, we first need to modify the `../ad/GOAD/providers/vmware/inventory` file and change the default IP addresses and replace the IPs we've set at the beginning.

(JJK 9/22/25)

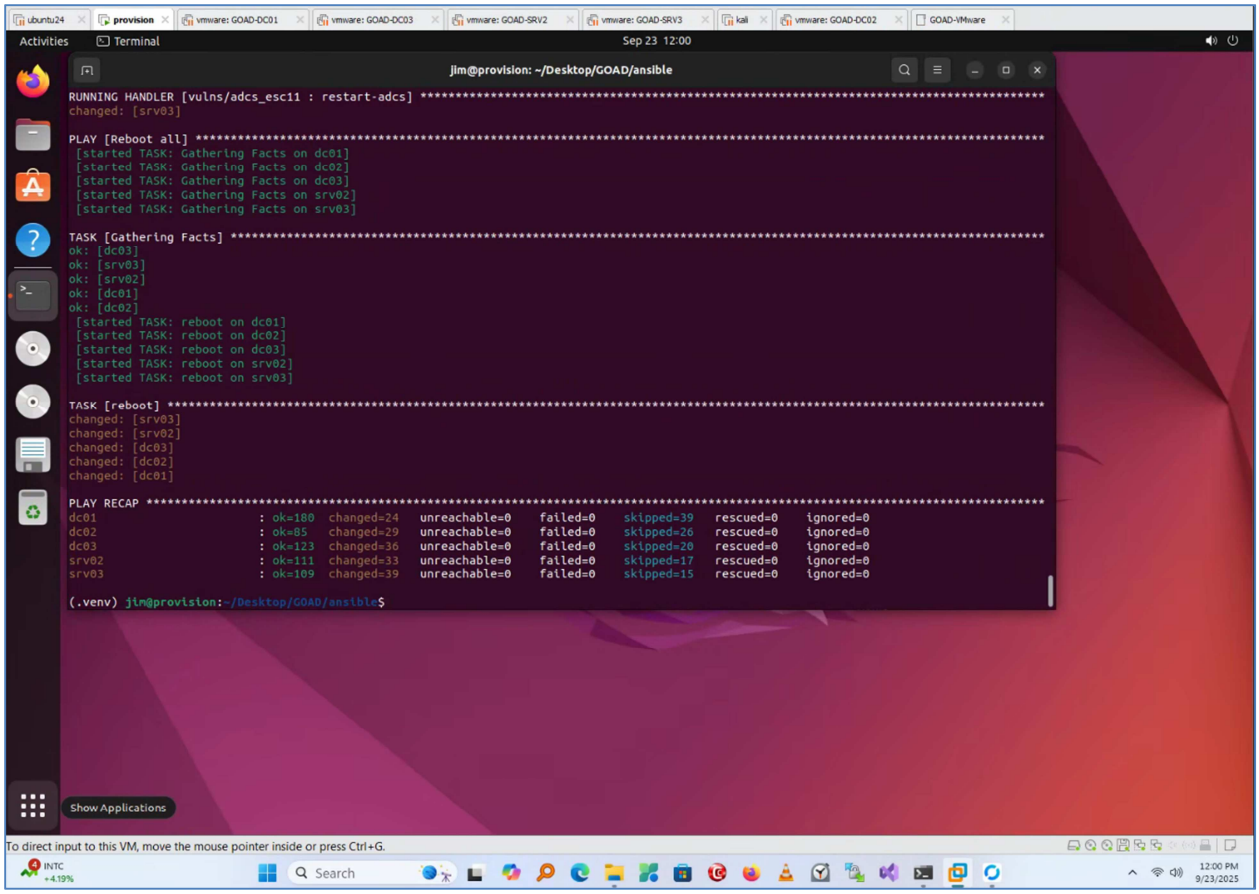
[inventory]

```
[default]
; Note: ansible_host *MUST* be an IPv4 address or setting things like
DNS
; servers will break.
; ----------------------------------------------
; sevenkingdoms.local
; ----------------------------------------------
dc01 ansible_host=192.168.56.10 dns_domain=dc01 dict_key=dc01
; ----------------------------------------------
; north.sevenkingdoms.local
; ----------------------------------------------
dc02 ansible_host=192.168.56.11 dns_domain=dc01 dict_key=dc02
srv02 ansible_host=192.168.56.22 dns_domain=dc02 dict_key=srv02
; ----------------------------------------------
; essos.local
; ----------------------------------------------
dc03 ansible_host=192.168.56.12 dns_domain=dc03 dict_key=dc03
srv03 ansible_host=192.168.56.23 dns_domain=dc03 dict_key=srv03
```

Time to run the provision:

```
ansible-playbook -i ../ad/GOAD/data/inventory -i
../ad/GOAD/providers/vmware/inventory main.yml
```

This by far took the longest to complete. It took approximately 20 minutes on my computer.



RUNNING HANDLER [vulns/adcs_esc11 : restart-adcs] ********
changed: [srv03]

PLAY [Reboot all] ********
[started TASK: Gathering Facts on dc01]
[started TASK: Gathering Facts on dc02]
[started TASK: Gathering Facts on dc03]
[started TASK: Gathering Facts on srv02]
[started TASK: Gathering Facts on srv03]

TASK [Gathering Facts] ********
ok: [dc03]
ok: [srv03]
ok: [srv02]
ok: [dc01]
ok: [dc02]
[started TASK: reboot on dc01]
[started TASK: reboot on dc02]
[started TASK: reboot on dc03]
[started TASK: reboot on srv02]
[started TASK: reboot on srv03]

TASK [reboot] ********
changed: [srv03]
changed: [srv02]
changed: [dc03]
changed: [dc02]
changed: [dc01]

PLAY RECAP ********
dc01 : ok=180 changed=24 unreachable=0 failed=0 skipped=39 rescued=0 ignored=0
dc02 : ok=85 changed=29 unreachable=0 failed=0 skipped=26 rescued=0 ignored=0
dc03 : ok=123 changed=36 unreachable=0 failed=0 skipped=20 rescued=0 ignored=0
srv02 : ok=111 changed=33 unreachable=0 failed=0 skipped=17 rescued=0 ignored=0
srv03 : ok=109 changed=39 unreachable=0 failed=0 skipped=15 rescued=0 ignored=0

(.venv) jim@provision:~/Desktop/GOAD/ansible$

Ansible – Zoomed

If you get an error, run ansible-playbook again just in case there was a VM that was in the process of re-booting:

```
ansible-playbook -i ../ad/GOAD/data/inventory -i
../ad/GOAD/providers/vmware/inventory main.yml
```

C. Attack Techniques (using Kali VM)

Let's create a **Kali VM** from ISO:

- RAM: 4 GB
- Processors: 2
- HD: 60GB
- VM: kali-linux-2025-2-installer-amd64.iso
- Network Adapter 1: NAT
- Network Adapter 2: VMnet2
    a. Static IP 192.168.56.50

Add a 2nd interface:
`sudo nano /etc/network/interfaces`

Add the following to create a static IP on the second interface:

```
auto eth1
iface eth1 inet static
    address 192.168.56.50
    netmask 255.255.255.0
    dns-nameserver 8.8.8.8
```

Make change to `/etc/host`:

```
127.0.0.1       localhost
127.0.1.1       kali

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

#GOAD
192.168.56.10   sevenkingdoms.local kingslanding.sevenkingdoms.local
kingslanding
192.168.56.11   winterfell.north.sevenkingdoms.local
north.sevenkingdoms.local winterfell
192.168.56.12   essos.local meereen.essos.local meereen
192.168.56.22   castelblack.north.sevenkingdoms.local castelblack
```

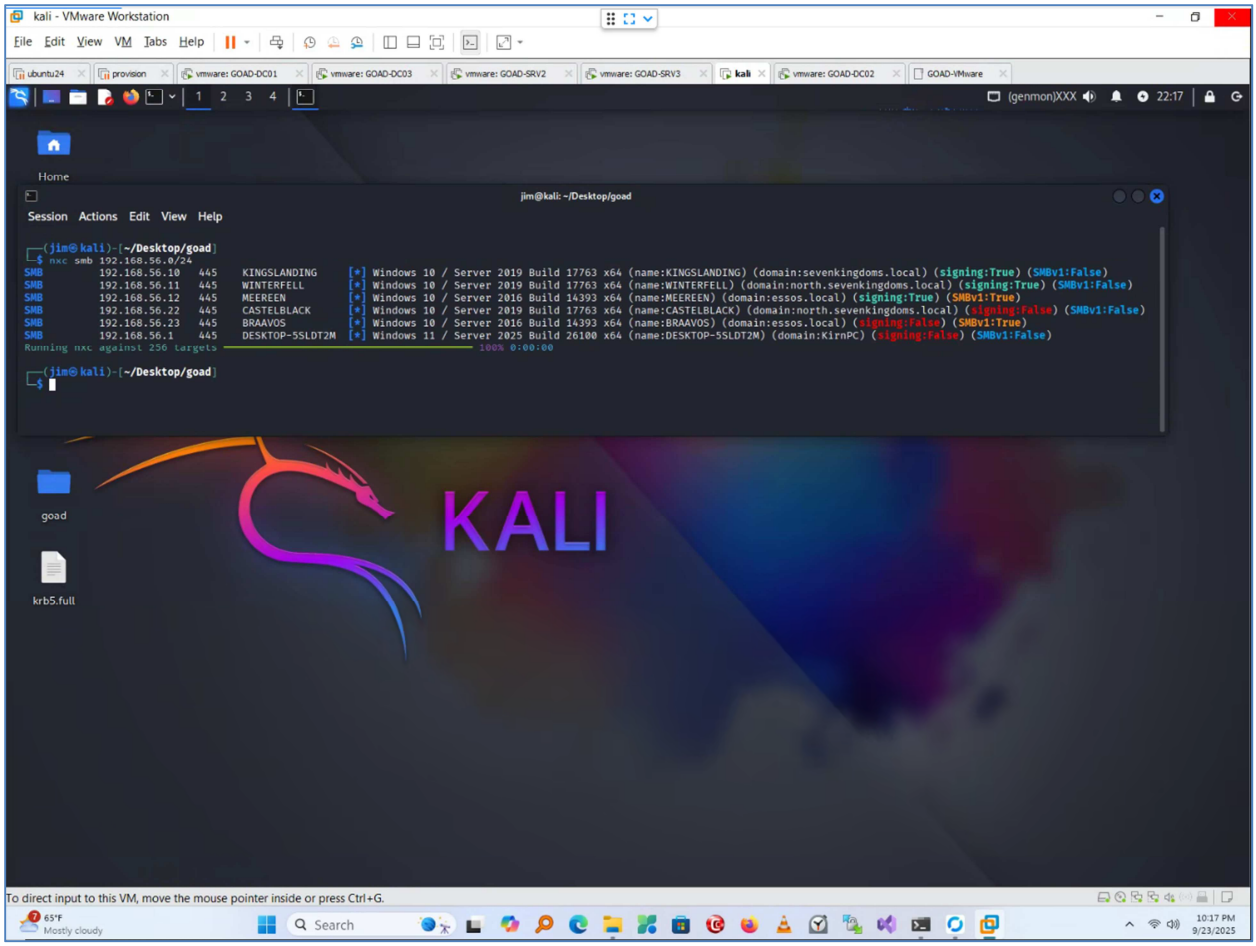
```
192.168.56.23   braavos.essos.local braavos
```

Install krb5 user file:
```
sudo apt install krb5-user
```

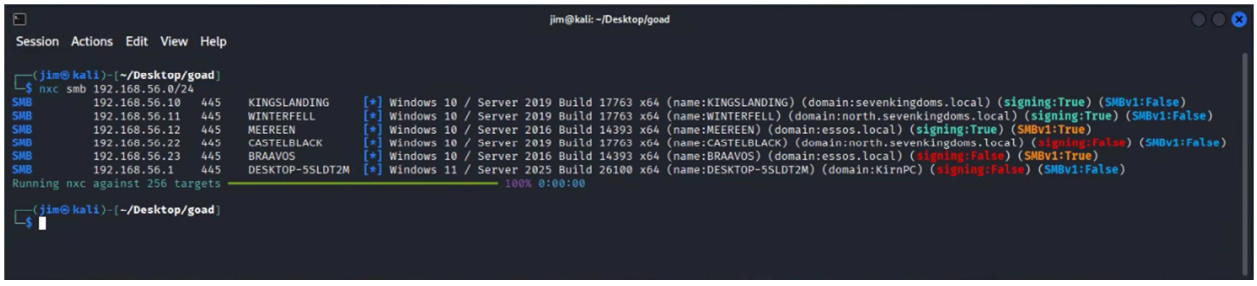
Replace the `/etc/krb5.conf` file with:

```
[libdefaults]
        default_realm = essos.local
        kdc_timesync = 1
        ccache_type = 4
        forwardable = true
        fcc-mit-ticketflags = true

[realms]
        north.sevenkigdoms.local = {
            kdc = winterfell.north.sevenkigdoms.local
            admin_server = winterfell.north.sevenkigdoms.local
        }
        sevenkigdoms.local = {
            kdc = kingslaning.sevenkigdoms.local
            admin_server = kingslaning.sevenkigdoms.local
        }
        essos.local = {
            kdc = meereen.essos.local
            admin_server = meereen.essos.local
        }


[domain_realm]
        .sevenkingdoms.local = SEVENKINGDOMS.LOCAL
        sevenkingdoms.local = SEVENKINGDOMS.LOCAL

        .north.sevenkingdoms.local = NORTH.SEVENKINGDOMS.LOCAL
        north.sevenkingdoms.local = NORTH.SEVENKINGDOMS.LOCAL

        .meereen.essos.local = MEEREEN.ESSOS.LOCAL
        meereen.essos.local = MEEREEN.ESSOS.LOCAL
```

Run our first command:



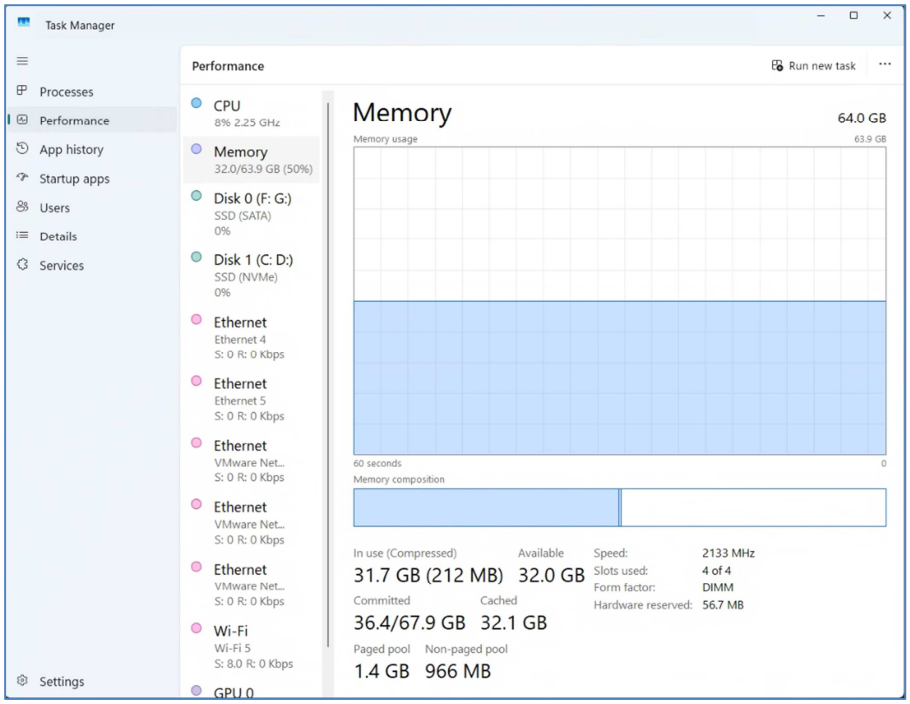
Zoomed In



Now ready to perform our attacks!!!

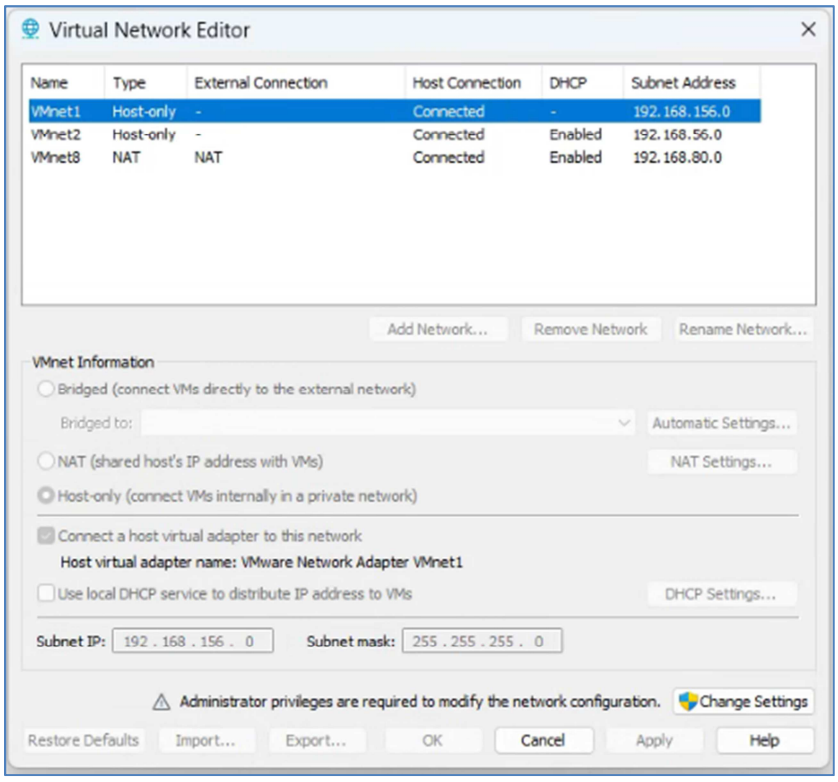
D. Extra Information

1. FYI – Memory Usage for all 5 Windows VMs and Kali VM



Just under 32 GB !

2. VMware Virtual Networking Editor:



3. Add Powershell script to G:\GOAD\vagrant to "Enable Ping" response on all VMs (Windows Host Only):

fix_ping.ps1

```
# Ensure script runs with admin privileges
if (-not ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.Wind
owsBuiltInRole]::Administrator)) {
    Write-Host "This script must be run as Administrator."
    exit 1
}

# Define rule name
$ruleName = "ICMP Allow incoming V4 echo request"

# Check if rule already exists
$existingRule = Get-NetFirewallRule -DisplayName $ruleName -ErrorAction
SilentlyContinue

if ($existingRule) {
    Write-Host "Firewall rule '$ruleName' already exists."
} else {
    Write-Host "Adding firewall rule '$ruleName'..."

    try {
        New-NetFirewallRule `
            -DisplayName $ruleName `
            -Direction Inbound `
            -Protocol ICMPv4 `
            -IcmpType 8 `
            -Action Allow `
            -Profile Any `
            -Enabled True `
            -Group "Custom ICMP Rules" `
            -Description "Allow incoming ICMPv4 echo requests (ping)" `
            -Program "System"

        Write-Host "Firewall rule added successfully."
    } catch {
        Write-Host "Failed to add firewall rule: $_"
        exit 1
    }
}
```

4. Revise the Powershell script "fix_ip.ps1" located at G:\GOAD\vagrant (Windows Host Only):

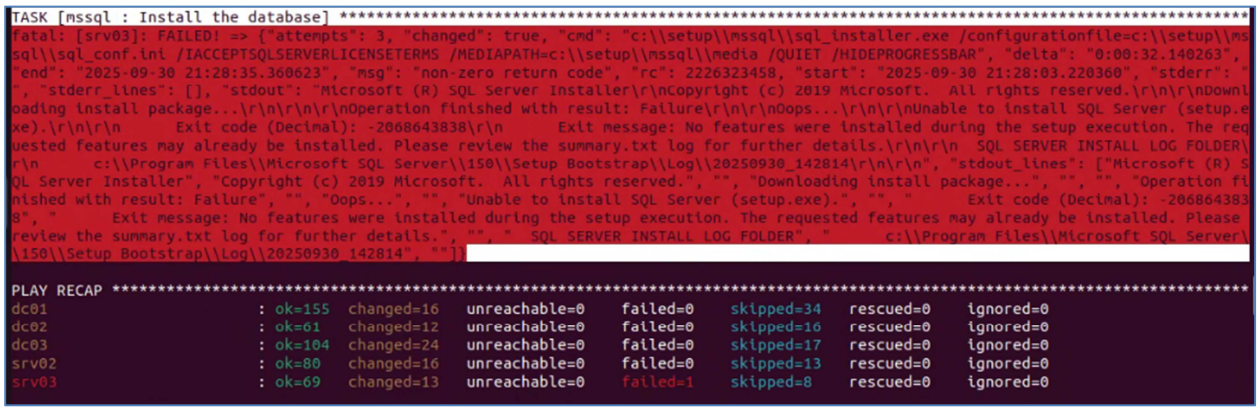
fix_ip.ps1

```
param (
```

```
    [string]$ip
)

$interfaceName = "Ethernet1"
$subnetPrefixLength = 24

Write-Host "Using interface: $interfaceName"

# Validate IP format
if (-not $ip -or $ip -notmatch '^(?:\d{1,3}\.){3}\d{1,3}$') {
    Write-Host "Invalid or missing IP address. Usage: fix_ip.ps1 <IPv4 address>"
    exit 1
}

# Confirm the interface exists and is up
$interface = Get-NetAdapter | Where-Object { $_.Name -eq $interfaceName -and
$_.Status -eq "Up" }
if (-not $interface) {
    Write-Host "Network interface '$interfaceName' not found or not active."
    exit 1
}

# Disable DHCP
try {
    Set-NetIPInterface -InterfaceAlias $interfaceName -Dhcp Disabled -ErrorAction
Stop
    Write-Host "DHCP disabled on $interfaceName"
} catch {
    Write-Host "Warning: Failed to disable DHCP. Continuing..."
}

# Remove existing IPv4 addresses (not sure this is needed)
try {
    Remove-NetIPAddress -InterfaceAlias $interfaceName -IPAddress $ipEntry.IPAddress
-Confirm:$false -ErrorAction SilentlyContinue
    Write-Host "Removed existing IP addresses from $interfaceName"
} catch {
    Write-Host "Warning: Failed to remove existing IP address. Continuing..."
}

# Set the new static IP
try {
    New-NetIPAddress -InterfaceAlias $interfaceName -IPAddress $ip -PrefixLength
$subnetPrefixLength -ErrorAction Stop
    Write-Host "IP address successfully set to $ip on '$interfaceName'."
} catch {
    Write-Host "Failed to set IP address: $_"
    exit 1
}
```

5. **./GOAD/ansible/roles/tasks/main.yml (**on Ubuntu 20.04 Desktop VM only**) for "provision" change.**

This change takes care of an error where the ansible output indicates there is an error although everything completed correctly. Here is an example:

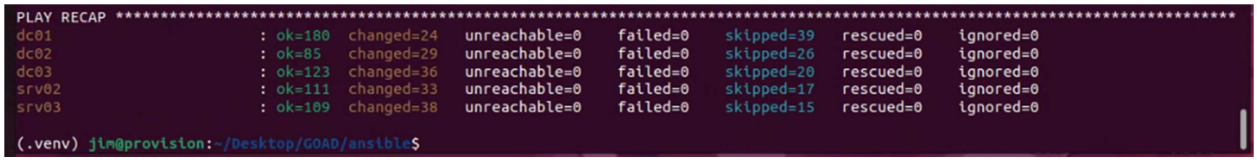


(starting at line #88 - just a simple one line addition as highlighed)

```
# Install the database with a domain admin user
- name: Install the database
  win_command: c:\setup\mssql\sql_installer.exe
/configurationfile=c:\setup\mssql\sql_conf.ini /IACCEPTSQLSERVERLICENSETERMS
/MEDIAPATH=c:\setup\mssql\media /QUIET /HIDEPROGRESSBAR
  args:
    chdir: c:\setup
  vars:
    ansible_become: yes
    ansible_become_method: runas
    ansible_become_user: "{{domain_admin}}"
    ansible_become_password: "{{domain_admin_password}}"
    ansible_become_flags: logon_type=new_credentials logon_flags=netcredentials_only
  register: mssqlinstall
  failed_when: sql_install.rc not in [0, -2068643838]
  until: "mssqlinstall is not failed"
  retries: 3
  delay: 120
  when: mssql_install_already_done.state is not defined or
mssql_install_already_done.name is not defined
#  when: not mssql_install_already_done.stat.exists
```

Re-run after the “fix”:

```
(.venv) jim@provision:~/Desktop/GOAD/ansible$ ansible-playbook -i
../ad/GOAD/data/inventory -i ../ad/GOAD/providers/vmware/inventory main.yml
```



END !