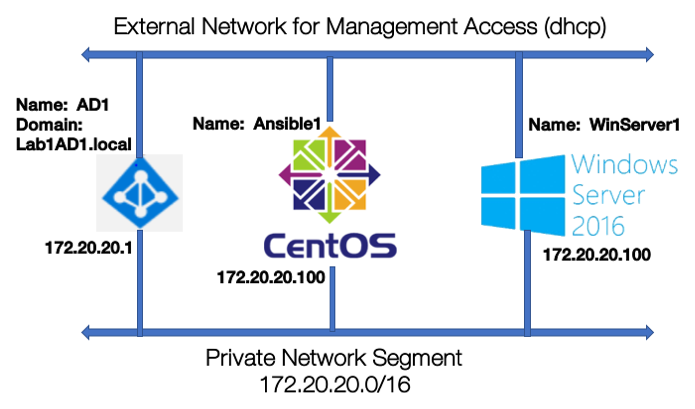
This lab is built on three VMs running on Hyper-V on a Windows 10 desktop. Here is an article describing a similar a similar scenario, [How to Build Windows Storage Spaces Direct on a Virtual Lab](https://argonsys.com/microsoft-cloud/articles/how-to-build-windows-storage-spaces-direct-virtual-lab/).

Below is a diagram of the example lab environment.



All communications between systems runs on a private network segment for simplicity.

**Install Lab Servers**

Build the following servers in your environment.

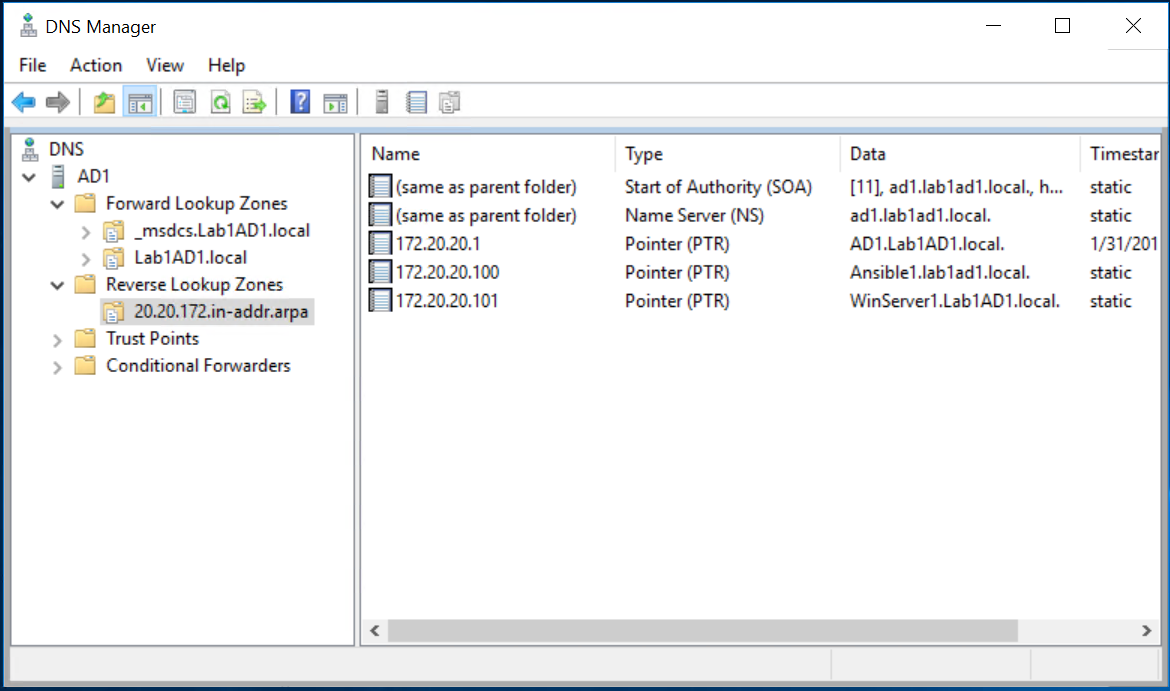
| **Name** | **Notes** |
| --- | --- |
| **Ansible1** | This will be a CentOS version 7 system with the minimal software selection. Make sure you can access this system via SSH from a client such as Putty. |
| **WinServer1** | This is a Windows Server 2016.   * After installing Windows Server 2016, apply all the latest Microsoft Updates * Rename the server to WinServer1 (or whatever you like) |
| **AD1** | This is a Windows Server 2016 with the Active Directory Domain Services role configured. This system is not required for domain services for the examples below. This node only provides DNS services for this environment.   * Apply all the latest Microsoft Updates * Rename the server to AD1 (or whatever) * Install the Active Directory Domain Services role * Configure the Active Directory * Domain: Lab1AD1.local |

**Configure Lab Network**

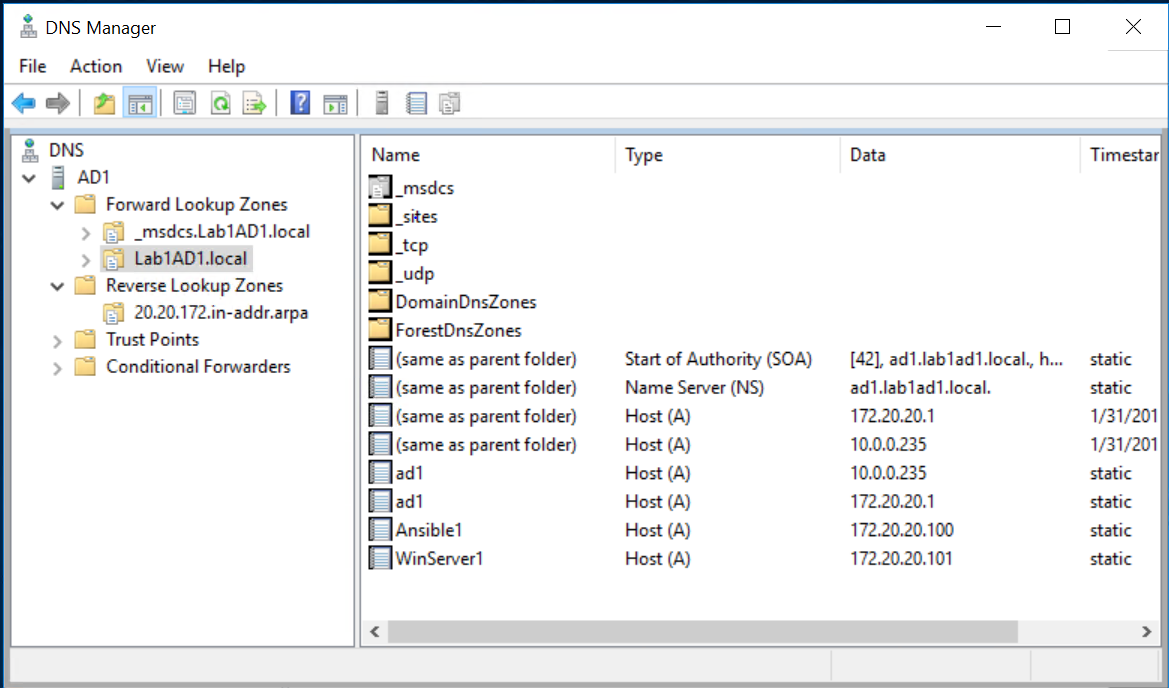
**Configure AD1 DNS services**

Configure DNS Forward and Reverse Lookups, Kerberos requires both forward and reverse DNS lookup to resolve correctly.

Configure the DNS Reverse Lookup Zone. This will look like the screenshot below.

[](https://argonsys.com/wp-content/uploads/2018/02/kb32-dns-reverse-lookup-zone.png)

The DNS Forward Lookup Zone will be like the screenshot below

[](https://argonsys.com/wp-content/uploads/2018/02/kb32-dns-forward-lookup-zone.png)

Add **Ansible1** and **WinServer1** A records:

* Ansible1: 172.20.20.100 – Select to create the PTR record automatically
* WinServer1: 172.20.20.101 – Select to create the PTR record automatically

Review the Reverse Look Zone 20.20.172.in-addr.arpa. You may need to select **Right-Click** and **REFRESH** to see the new records.

**Configure Ansible1 CentOS System Network**

Update the following files. Be sure to modify as appropriate for your own environment.

The eth0 interface in this example uses DHCP addresses to access the Internet and provide SSH access for administration.

1. Edit: /etc/sysconfig/network-scripts/ifcfg-eth0



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | TYPE="[Ethernet](https://argonsys.com/microsoft-cloud/glossary/ethernet/)"  PROXY\_METHOD="none"  BROWSER\_ONLY="no"  BOOTPROTO="dhcp"  DEFROUTE="yes"  PEERDNS="no"  IPV4\_FAILURE\_FATAL="no"  NAME="eth0"  DEVICE="eth0"  ONBOOT="yes" |

PEERDNS is set to “no” to prevent DHCP from inserting external DNS records. The only DNS records should come from AD1.

The eth1 interface is the private network and is shared only with the other servers in this lab.

1. Edit /etc/sysconfig/network-scripts/ifcfg-eth1



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | TYPE=Ethernet  PROXY\_METHOD=none  BROWSER\_ONLY=no  BOOTPROTO=static  DEFROUTE=yes  IPADDR=172.20.20.100  NETMAST=255.255.0.0  DNS1=172.20.20.1  IPV4\_FAILURE\_FATAL=no  NAME=eth1  DEVICE=eth1  ONBOOT=yes |

The resolv.conf configures the Linux DNS client.

1. Edit /etc/resolv/conf



|  |  |
| --- | --- |
| **1**  **2** | search Lab1AD1.local  nameserver 172.20.20.1 |

If Network Manager is running (the default configuration), it will overwrite this configuration. You can disable the Network Manager with the command **systemctl disable NetworkManager**

Set the hostname

1. Edit: /etc/hostname



|  |  |
| --- | --- |
| **1** | Ansible1 |

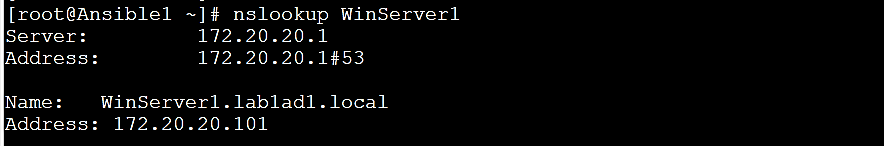
1. Reboot the server

Type: **reboot**

1. Test your network. Make sure you can resolve DNS correctly.

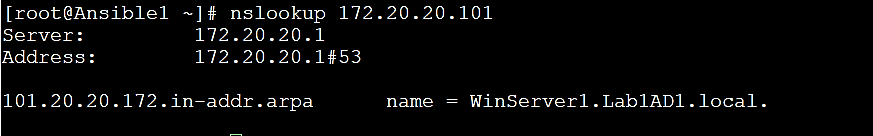
Type: **nslookup WinServer1**

This command will query the AD1 DNS server for **WinServer1** without a fully qualified name.

[](https://argonsys.com/wp-content/uploads/2018/02/kb32-ansible-nslookup-winserver.png)

Type: **nslookup 172.20.20.101**

This command will do a reverse lookup on the IP address. This should return WinServer1 as the name.

[](https://argonsys.com/wp-content/uploads/2018/02/kb32-ansible-nslookup.png)

**Configure Ansible Environment**

Other versions of Linux will work equally well. The configuration commands will have to be adjusted for each version of Linux.

**Install Prerequisite Packages**

Use Yum to install the following packages.

1. Install GCC required for Kerberos



|  |  |
| --- | --- |
| **1** | yum -y group install “Development Tools” |

1. Install EPEL



|  |  |
| --- | --- |
| **1** | yum -y install epel-release |

1. Install Ansible



|  |  |
| --- | --- |
| **1** | yum -y install ansible |

1. Install Kerberos



|  |  |
| --- | --- |
| **1** | yum -y install python-devel krb5-devel krb5-libs krb5-workstation |

1. Install Python PIP



|  |  |
| --- | --- |
| **1** | yum -y install python-pip |

1. Install BIND utilities for nslookup



|  |  |
| --- | --- |
| **1** | yum -y install bind-utils |

1. Bring all packages up to the latest version



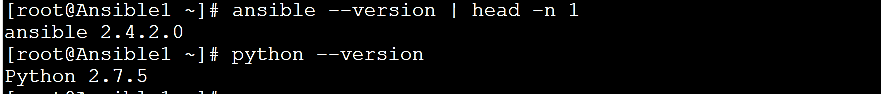
|  |  |
| --- | --- |
| **1** | yum -y update |

**Check Ansible and Python is Installed**

1. Run the commands:



|  |  |
| --- | --- |
| **1**  **2** | ansible - - version | head -l 1  python - - version |



The versions of Ansible and Python here are 2.4.2 and 2.7.5. Ansible is developing extremely rapidly so these instructions will likely change in the near future.

**Configure Kerberos**

There are other options than Kerberos, but Kerberos is generally the best option, though not the simplest.

1. Install the Kerberos wrapper:



|  |  |
| --- | --- |
| **1** | pip install pywinrm[Kerberos] |

Kerberos packages were installed previously which will have created /etc/krb5.conf

1. Edit /etc/krb5.conf

Add:



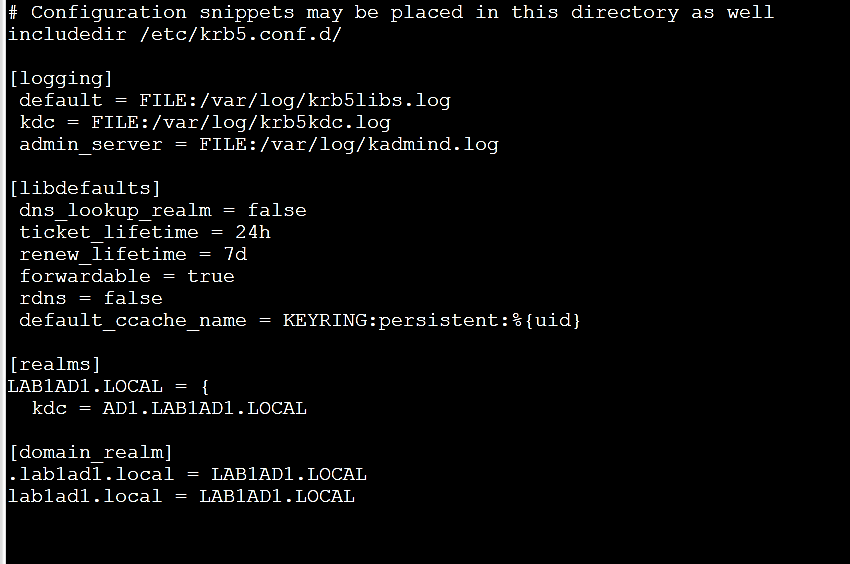
|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | [realms]  LAB1AD1.LOCAL = {    kdc = AD1.LAB1AD1.LOCAL  } |

Add:



|  |  |
| --- | --- |
| **1**  **2**  **3** | [domain\_realm]  .lab1ad1.local = LAB1AD1.LOCAL  lab1ad1.local = LAB1AD1.LOCAL |

The /etc/krb5.conf file when complete will be similar to:



Only the [realms] and [domain\_realm] were updated manually.

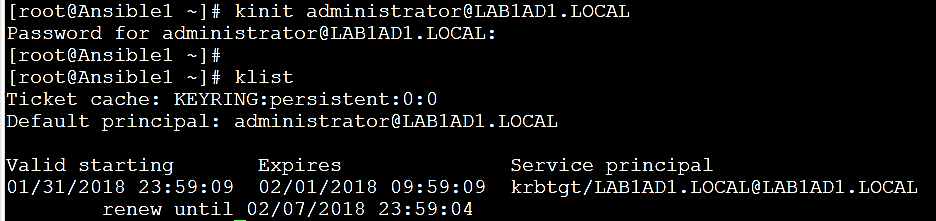
**Test Kerberos**

1. Run the following commands to test Kerberos:



|  |  |
| --- | --- |
| **1** | kinit administrator@LAB1AD1.LOCAL |

You will be prompted for the administrator password **klist**  
You should see a Kerberos KEYRING record.



**Configure Ansible**

Ansible is complex and is sensitive to the environment. Troubleshooting an environment which has never initially worked is complex and confusing. We are going to configure Ansible with the least complex possible configuration. Once you have a working environment, you can make extensions and enhancements in small steps.

The core configuration of Ansible resides at /etc/ansible

https://argonsys.com/wp-content/uploads/2018/02/kb32-ansible-configuration-file.png

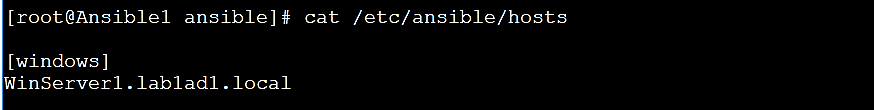
We are only going to update two files for this exercise.

**Update the Ansible Inventory file**

1. Edit /etc/ansible/hosts and add:



|  |  |
| --- | --- |
| **1**  **2** | [windows]  WinServer1.lab1ad1.local |



“[windows]” is a created group of servers called “windows”. In reality this should be named something more appropriate for a group which would have similar configurations, such as “Active Directory Servers”, or “Production Floor Windows 10 PCs”, etc.

**Update the Ansible Group Variables for Windows**

Ansible Group Variables are variable settings for a specific inventory group. In this case, we will create the group variables for the “windows” servers created in the /etc/ansible/hosts file.

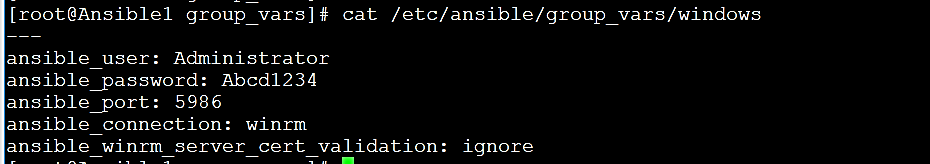
1. Create /etc/ansible/group\_vars/windows and add:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6** | ---  ansible\_user: Administrator  ansible\_password: Abcd1234  ansible\_port: 5986  ansible\_connection: winrm  ansible\_winrm\_server\_cert\_validation: ignore |

This is a YAML configuration file, so make sure the first line is three dashes “‐‐‐”

Naturally change the Administrator password to the password for **WinServer1**.



For best practices, Ansible can encrypt this file into the Ansible Vault. This would prevent the password from being stored here in clear text. For this lab, we are attempting to keep the configuration as simple as possible. Naturally in production this would not be appropriate.

**Configure Windows Servers to Manage**

To configure the Windows Server for remote management by Ansible requires a bit of work. Luckily the Ansible team has created a PowerShell script for this. Download this script from [here] to each Windows Server to manage and run this script as Administrator.

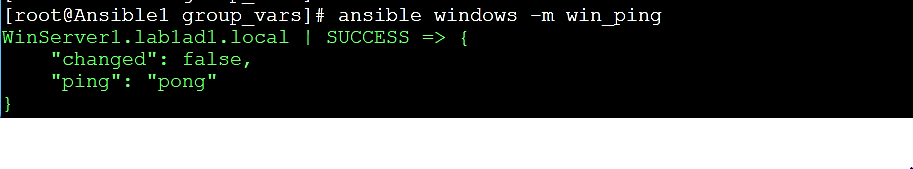
Log into **WinServer1** as Administrator, download [ConfigureRemotingForAnsible.ps1](https://github.com/ansible/ansible/blob/devel/examples/scripts/ConfigureRemotingForAnsible.ps1)and run this PowerShell script without any parameters.

Once this command has been run on the **WinServer1**, return to the Ansible1 Controller host.

**Test Connectivity to the Windows Server**

If all has gone well, we should be able to perform an Ansible PING test command. This command will simply connect to the remote **WinServer1** server and report success or failure.

1. Type: **ansible windows -m win\_ping**



This command runs the Ansible module “win\_ping” on every server in the “windows” inventory group.

1. Type: **ansible windows -m setup** to retrieve a complete configuration of Ansible environmental settings.
2. Type: **ansible windows -c ipconfig**

If this command is successful, the next steps will be to build Ansible playbooks to manage Windows Servers.

**Managing Windows Servers with Playbooks**

Let’s create some playbooks and test Ansible for real on Windows systems.

1. Create a folder on **Ansible1** for the playbooks, YAML files, modules, scripts, etc. For these exercises we created a folder under /root called win\_playbooks.

https://argonsys.com/wp-content/uploads/2018/02/kb32-ansible-playbooks-folder.png

Ansible has some expectations on the directory structure where playbooks reside. Create the library and scripts folders for use later in this exercise.

Commands:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | cd /root  mkdir win\_playbooks  mkdir win\_playbooks/library  mkdir win\_playbooks/scripts |

1. Create the first playbook example “netstate.yml”

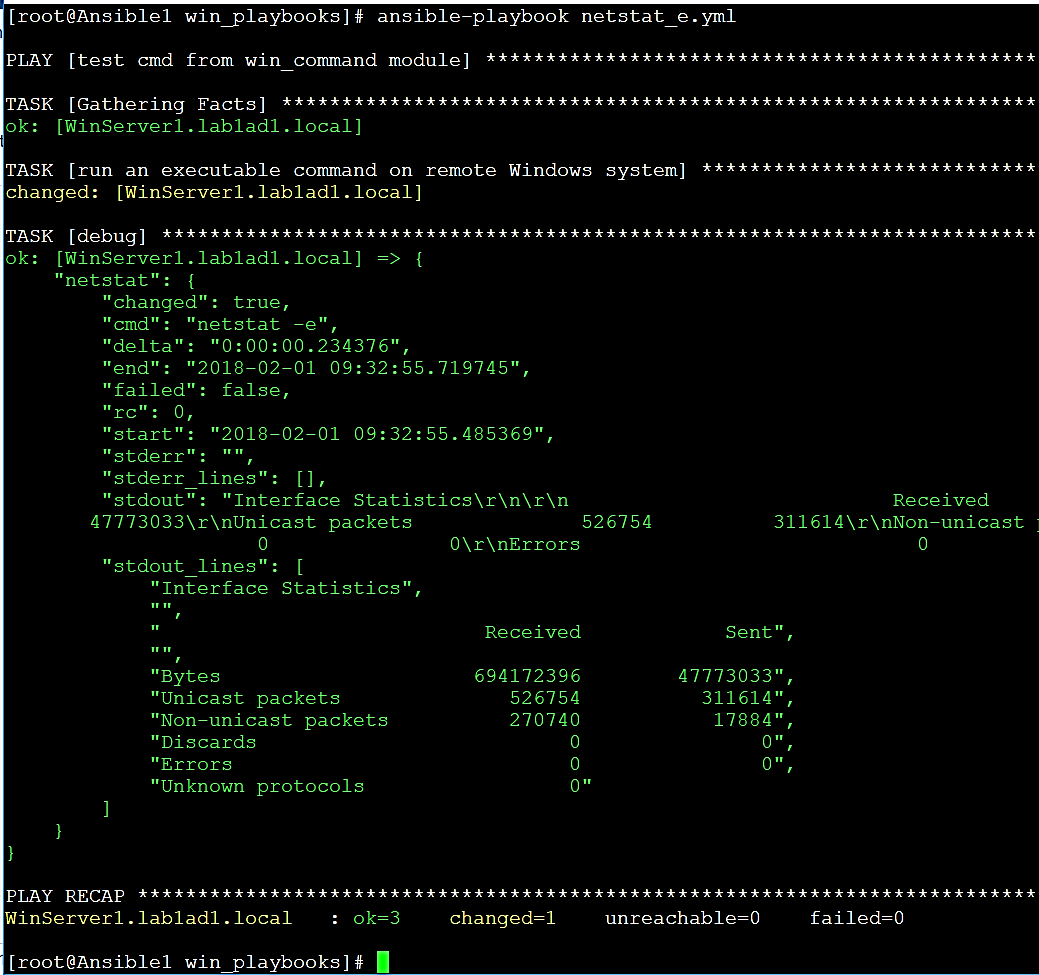
The contents are:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | - name: test cmd from win\_command module    hosts: windows    tasks:      - name: run netstat and return Ethernet stats        win\_command: netstat -e        register: netstat      - debug: var=netstat |

This playbook does only one task, to connect to the servers in the Ansible inventory group “windows” and run the command **netstat.exe -a** and return the results.

To run this playbook, run this command on **Ansible1**: **ansible-playbook netstat\_e.yml**



OK, not exciting, but it did run, just not very friendly.

Also, notice that the “changed” flag is set. As of the time this article was created, all of the Windows commands return the “changed” flag as true. This is the same with running PowerShell scripts remotely. This makes much of the value of Ansible difficult as a configuration and deployment tool.

Interestingly enough, Ansible modules created with PowerShell do work correctly and return the “changed” flag correctly. PowerShell scripts and direct commands always return the “changed” flag as true. This is problematic for managing systems with Ansible. The rest of this article will focus on PowerShell modules which can perform complex management functions as well as integrate with other non-Windows systems.

**Creating Ansible Modules with PowerShell**

Ansible modules are plugin programs which are:

* 1. Loaded by Ansible when running a playbook test
  2. Ansible generates module input parameters in the [JSON](https://argonsys.com/microsoft-cloud/glossary/javascript-object-notation/) format
  3. Modifies the module into a generalized script and command
  4. Copies the modified script to the remote system(s)
  5. Executes the modified module on the remote system
  6. The module generates a response in JSON and this response is returned
  7. The returned JSON is parsed and values are saved or use by other tests
  8. The module returns a flag called “changed” which is important to maintain “Desired State Configurations”. A return of “changed = True” will signal other tests to run to achieve “Desired State”

Modules are stored in several locations where Ansible will find them. One location is a folder named “library” located in the folder where the playbook is run. Modules stored in our library are first in the module search path and will override modules of the same name.

This example lab has a folder win\_playbooks/library/

https://argonsys.com/wp-content/uploads/2018/02/kb32-ls-win_playbookslibrary.png

PowerShell modules consist of two programs, a Python program and a PowerShell program. The Python program configures the local Ansible environment and the PowerShell program does the actual processing on the remote systems.

Create a file called “win\_playbooks/library/get\_version.py” and add the following contents:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39**  **40**  **41**  **42**  **43**  **44**  **45**  **46**  **47**  **48**  **49**  **50** | #!/usr/bin/python  # -\*- coding: utf-8 -\*-  #License: Public Domain  ANSIBLE\_METADATA = {'metadata\_version': '1.1',                      'status': ['stableinterface'],                      'supported\_by': 'core'}  DOCUMENTATION = r'''  ---  module: get\_version  short\_description: Windows get OS verison and return changed flag if below min  description:    - Checks Windows OS version and returns the version data in JSON    - Checks that the min version against the passed "major,minor,build" parameters    - Returns "changed" if the remote system's OS version is below a minimum    - The "changed" flag is t notify operations or to trigger another task  options:    major:      description:        - Windows major version [MAJOR.minor.build]      default: 5    minor:      description:        - Windows minor version [major.MINOR.build]      default: 1    build:      description:        - Windows build version [major.minor.BUILD]      default: None  author:  - Robert Keith (Argon Systems)  '''    EXAMPLES = r'''  # Example Ansible Playbook  - name: test remote Windows OS version    hosts: windows    tasks:      - name: PowerShell module example to get Windows OS version        get\_version:          major: "5"          minor: "1"          build: "13493"    '''    RETURN = '''      description: Output of (Get-Host).Version powershell in JSON format      returned: success      type: string  ''' |

Next create a file “win\_playbooks/library/get\_version.ps1” and add the following contents:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25** | #!powershell  # WANT\_JSON  # POWERSHELL\_COMMON  $ErrorActionPreference = "Stop"  $params = Parse-Args $args -supports\_check\_mode $true  $MajorVer = Get-AnsibleParam -obj $params -name "major" -type "str"  $MinorVer = Get-AnsibleParam -obj $params -name "minor" -type "str"  $BuildVer = Get-AnsibleParam -obj $params -name "build" -type "str"  #Should validate input parameters here naturally  $Vers = (Get-Host).Version  $Message = $Message = "Windows OS is at desired version"  $BelowMin = $false  if ($Vers.Major -lt $MajorVer) { $BelowMin = $true }  if ($Vers.Minor -lt $MinorVer) { $BelowMin = $true }  if ($Vers.Build -lt $BuildVer) { $BelowMin = $true }  if ($BelowMin) { $Message = "Windows OS below desired version" }  $result = @{    changed = $BelowMin    version = $Vers    message = $Message    desired\_major\_ver = $MajorVer    desired\_minor\_ver = $MinorVer    desired\_build\_ver = $BuildVer  }  Exit-Json $result |

Next, create the playbook YAML file to setup and run the new module.

From the win\_playbooks folder, create a file with the following contents:

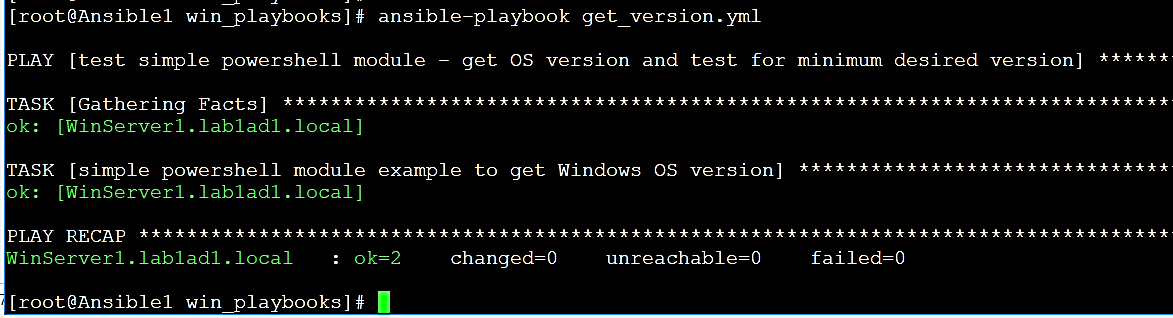


|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | - name: test simple powershell module - get OS version and test for minimum desired version    hosts: windows    tasks:      - name: simple powershell module example to get Windows OS version        get\_version:          major: "5"          minor: "1"          build: "14393" |

Execute this playbook with the command:



|  |  |
| --- | --- |
| **1** | ansible-playbook get\_version.yml |

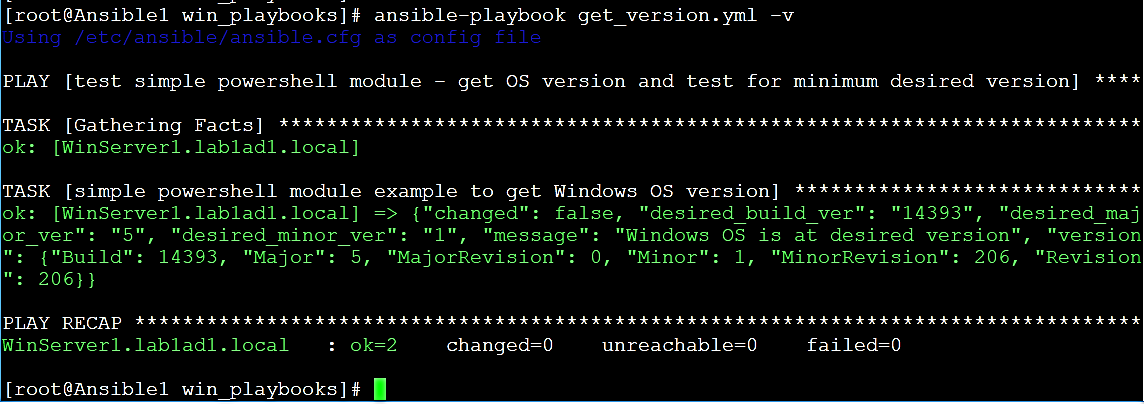


Notice the “changed” flag is set to False. In this case, this means the remote operating system matched the values in the get\_version.yml playbook.

If we set the Verbose flag, we see the data returned from the PowerShell module in the JSON format.



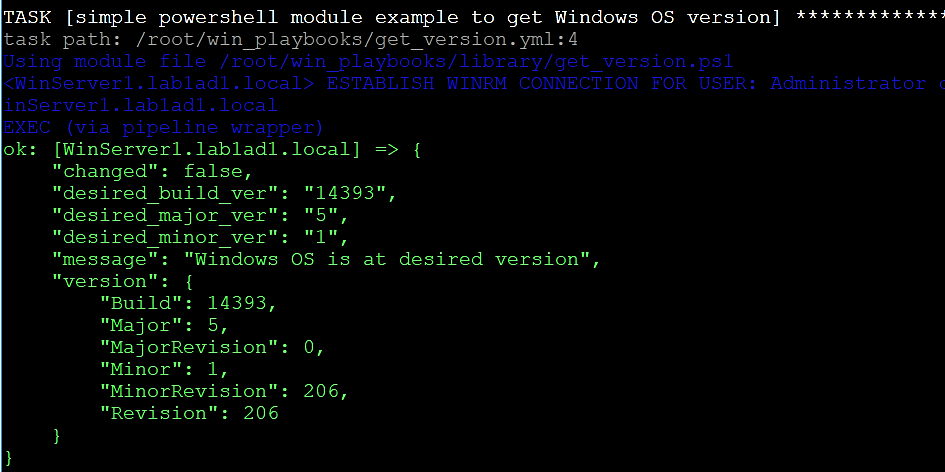
|  |  |
| --- | --- |
| **1** | ansible-playbook get\_version.yml -v |



Setting the Very Very Verbose flag give more information, and oddly enough formats the JSON into a more readable format.



|  |  |
| --- | --- |
| **1** | ansible-playbook get\_version.yml -vvv |



Now if we modify the playbook expecting a different version. We set the major version from 5 to 9 (which does not exist).

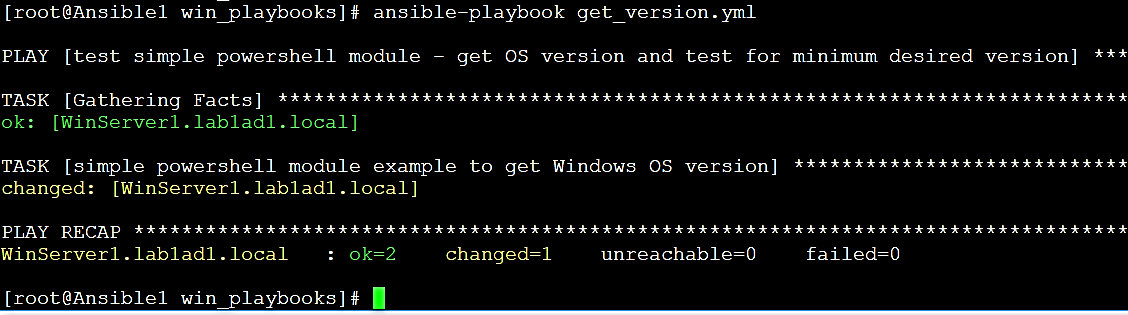


|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | - name: test simple powershell module - get OS version and test for minimum desired version    hosts: windows    tasks:      - name: simple powershell module example to get Windows OS version        get\_version:          major: "9"          minor: "1"          build: "14393" |

And run the playbook again



|  |  |
| --- | --- |
| **1** | ansible-playbook get\_version.yml |



The “changed” flag is set.

We could extend this playbook with other PowerShell modules, for example:



|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | - name: test simple powershell module - get OS version and test for minimum desired version    hosts: windows    tasks:      - name: simple powershell module example to get Windows OS version        get\_version:          major: "9"          minor: "1"          build: "14393"        register: get\_version\_output      - debug:          var: get\_version\_output.message      - enable\_win\_updates: Enable unattended Windows Updates        when get\_version\_output.changed |

This playbook could check the Windows version, and if it is below a version threshold, the next task could then enable updates, manually install updates, or practically any possible action imaginable to maintain the Windows servers to a “Desired State”.