Using Ansible and Windows

- Use Cases
  - Installing Software
  - Installing Updates
  - Set Up Users and Groups
    - Local
    - Domain
  - Running Commands
    - Choosing Command or Shell
    - Argument Rules
  - Creating and Running a Scheduled Task
- Path Formatting for Windows
  - YAML Style
  - Legacy key=value Style
- Limitations
- Developing Windows Modules

When using Ansible to manage Windows, many of the syntax and rules that apply for Unix/Linux hosts also apply to Windows, but there are still some differences when it comes to components like path separators and OS-specific tasks. This document covers details specific to using Ansible for Windows.

Topics

Use Cases

Ansible can be used to orchestrate a multitude of tasks on Windows servers. Below are some examples and info about common tasks.

Installing Software

There are three main ways that Ansible can be used to install software:

- Using the `win_chocolatey` module. This sources the program data from the default public Chocolatey repository. Internal repositories can be used instead by setting the `source` option.
- Using the `win_package` module. This installs software using an MSI or .exe installer from a local/network path or URL.
- Using the `win_command` or `win_shell` module to run an installer manually.

The `win_chocolatey` module is recommended since it has the most complete logic for checking to see if a package has already been installed and is up-to-date.

Below are some examples of using all three options to install 7-Zip:

```
# Install/uninstall with chocolatey
- name: Ensure 7-Zip is installed through Chocolatey
  win_chocolatey:
    name: 7zip
    state: present

- name: Ensure 7-Zip is not installed through Chocolatey
  win_chocolatey:
    name: 7zip
    state: absent

# Install/uninstall with win_package
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Ensure 7-Zip is installed through win_package
  win_package:
    path: C:\temp\7z.msi
    state: present

- name: Ensure 7-Zip is not installed through win_package
  win_package:
    path: C:\temp\7z.msi
    state: absent

# Install/uninstall with win_command
```

```
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Check if 7-Zip is already installed
  win_reg_stat:
    name: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{23170F69-40C1-2702-1701-000001000000}
  register: 7zip_installed

- name: Ensure 7-Zip is installed through win_command
  win_command: C:\Windows\System32\msiexec.exe /i C:\temp\7z.msi /qn /norestart
  when: 7zip_installed.exists == false

- name: Ensure 7-Zip is uninstalled through win_command
  win_command: C:\Windows\System32\msiexec.exe /x {23170F69-40C1-2702-1701-000001000000} /qn /norestart
  when: 7zip_installed.exists == true
```

Some installers like Microsoft Office or SQL Server require credential delegation or access to components restricted by WinRM. The best method to bypass these issues is to use `become` with the task. With `become`, Ansible will run the installer as if it were run interactively on the host.

> Many installers do not properly pass back error information over WinRM. In these cases, if the install has been verified to work locally the recommended method is to use become.
>
> Some installers restart the WinRM or HTTP services, or cause them to become temporarily unavailable, making Ansible assume the system is unreachable.

Installing Updates

The `win_updates` and `win_hotfix` modules can be used to install updates or hotfixes on a host. The module `win_updates` is used to install multiple updates by category, while `win_hotfix` can be used to install a single update or hotfix file that has been downloaded locally.

> The `win_hotfix` module has a requirement that the DISM PowerShell cmdlets are present. These cmdlets were only added by default on Windows Server 2012 and newer and must be installed on older Windows hosts.

The following example shows how `win_updates` can be used:

```
- name: Install all critical and security updates
  win_updates:
    category_names:
    - CriticalUpdates
    - SecurityUpdates
    state: installed
  register: update_result

- name: Reboot host if required
  win_reboot:
  when: update_result.reboot_required
```

The following example show how `win_hotfix` can be used to install a single update or hotfix:

```
- name: Download KB3172729 for Server 2012 R2
  win_get_url:
    url:
http://download.windowsupdate.com/d/msdownload/update/software/secu/2016/07/windows8.1-
kb3172729-x64_e8003822a7ef4705cbb65623b72fd3cec73fe222.msu
    dest: C:\temp\KB3172729.msu

- name: Install hotfix
  win_hotfix:
    hotfix_kb: KB3172729
    source: C:\temp\KB3172729.msu
    state: present
  register: hotfix_result

- name: Reboot host if required
  win_reboot:
  when: hotfix_result.reboot_required
```

Set Up Users and Groups

Ansible can be used to create Windows users and groups both locally and on a domain.

Local

The modules `win_user`, `win_group` and `win_group_membership` manage Windows users, groups and group memberships locally.

The following is an example of creating local accounts and groups that can access a folder on the same host:

```
- name: Create local group to contain new users
  win_group:
    name: LocalGroup
    description: Allow access to C:\Development folder

- name: Create local user
  win_user:
    name: '{{ item.name }}'
    password: '{{ item.password }}'
    groups: LocalGroup
    update_password: false
    password_never_expires: true
  loop:
  - name: User1
    password: Password1
  - name: User2
    password: Password2

- name: Create Development folder
  win_file:
    path: C:\Development
    state: directory

- name: Set ACL of Development folder
  win_acl:
    path: C:\Development
    rights: FullControl
    state: present
    type: allow
    user: LocalGroup
```

```
- name: Remove parent inheritance of Development folder
  win_acl_inheritance:
    path: C:\Development
    reorganize: true
    state: absent
```

Domain

The modules `win_domain_user` and `win_domain_group` manages users and groups in a domain. The below is an example of ensuring a batch of domain users are created:

```
- name: Ensure each account is created
  win_domain_user:
    name: '{{ item.name }}'
    upn: '{{ item.name }}@MY.DOMAIN.COM'
    password: '{{ item.password }}'
    password_never_expires: false
    groups:
    - Test User
    - Application
    company: Ansible
    update_password: on_create
  loop:
  - name: Test User
    password: Password
  - name: Admin User
    password: SuperSecretPass01
  - name: Dev User
    password: '@fvr3IbFBujSRh!3hBg%wgFucD8^x8W5'
```

Running Commands

In cases where there is no appropriate module available for a task, a command or script can be run using the `win_shell`, `win_command`, `raw`, and `script` modules.

The `raw` module simply executes a Powershell command remotely. Since `raw` has none of the wrappers that Ansible typically uses, `become`, `async` and environment variables do not work.

The `script` module executes a script from the Ansible control node on one or more Windows hosts. Like `raw`, `script` currently does not support `become`, `async`, or environment variables.

The `win_command` module is used to execute a command which is either an executable or batch file, while the `win_shell` module is used to execute commands within a shell.

Choosing Command or Shell

The `win_shell` and `win_command` modules can both be used to execute a command or commands. The `win_shell` module is run within a shell-like process like `PowerShell` or `cmd`, so it has access to shell operators like `<`, `>`, `|`, `;`, `&&`, and `||`. Multi-lined commands can also be run in `win_shell`.

The `win_command` module simply runs a process outside of a shell. It can still run a shell command like `mkdir` or `New-Item` by passing the shell commands to a shell executable like `cmd.exe` or `PowerShell.exe`.

Here are some examples of using `win_command` and `win_shell`:

```
- name: Run a command under PowerShell
  win_shell: Get-Service -Name service | Stop-Service

- name: Run a command under cmd
  win_shell: mkdir C:\temp
  args:
    executable: cmd.exe

- name: Run multiple shell commands
  win_shell: |
    New-Item -Path C:\temp -ItemType Directory
    Remove-Item -Path C:\temp -Force -Recurse
    $path_info = Get-Item -Path C:\temp
    $path_info.FullName

- name: Run an executable using win_command
  win_command: whoami.exe

- name: Run a cmd command
  win_command: cmd.exe /c mkdir C:\temp

- name: Run a vbs script
  win_command: cscript.exe script.vbs
```

> Some commands like `mkdir`, `del`, and `copy` only exist in the CMD shell. To run them with `win_command` they must be prefixed with `cmd.exe /c`.

Argument Rules

When running a command through `win_command`, the standard Windows argument rules apply:

- Each argument is delimited by a white space, which can either be a space or a tab.
- An argument can be surrounded by double quotes `"`. Anything inside these quotes is interpreted as a single argument even if it contains whitespace.
- A double quote preceded by a backslash `\` is interpreted as just a double quote `"` and not as an argument delimiter.
- Backslashes are interpreted literally unless it immediately precedes double quotes; for example `\` == `\` and `\"` == `"`
- If an even number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is used as a string delimiter for the argument.
- If an odd number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is escaped and made a literal double quote in the argument.

With those rules in mind, here are some examples of quoting:

```
- win_command: C:\temp\executable.exe argument1 "argument 2" "C:\path\with space" "double \"quoted\""

argv[0] = C:\temp\executable.exe
argv[1] = argument1
argv[2] = argument 2
argv[3] = C:\path\with space
argv[4] = double "quoted"

- win_command: '"C:\Program Files\Program\program.exe" "escaped \\\" backslash" unquoted-end-backslash\'

argv[0] = C:\Program Files\Program\program.exe
argv[1] = escaped \" backslash
argv[2] = unquoted-end-backslash\

# Due to YAML and Ansible parsing '\"' must be written as '{% raw %}\\{% endraw %}"'
- win_command: C:\temp\executable.exe C:\no\space\path "arg with end \ before end quote{% raw %}\\{% endraw %}"
```

```
argv[0] = C:\temp\executable.exe
argv[1] = C:\no\space\path
argv[2] = arg with end \ before end quote\"
```

For more information, see escaping arguments.

Creating and Running a Scheduled Task

WinRM has some restrictions in place that cause errors when running certain commands. One way to bypass these restrictions is to run a command through a scheduled task. A scheduled task is a Windows component that provides the ability to run an executable on a schedule and under a different account.

Ansible version 2.5 added modules that make it easier to work with scheduled tasks in Windows. The following is an example of running a script as a scheduled task that deletes itself after running:

```
- name: Create scheduled task to run a process
  win_scheduled_task:
    name: adhoc-task
    username: SYSTEM
    actions:
    - path: PowerShell.exe
      arguments: |
        Start-Sleep -Seconds 30  # This isn't required, just here as a demonstration
        New-Item -Path C:\temp\test -ItemType Directory
    # Remove this action if the task shouldn't be deleted on completion
    - path: cmd.exe
      arguments: /c schtasks.exe /Delete /TN "adhoc-task" /F
    triggers:
    - type: registration

- name: Wait for the scheduled task to complete
  win_scheduled_task_stat:
    name: adhoc-task
  register: task_stat
  until: (task_stat.state is defined and task_stat.state.status != "TASK_STATE_RUNNING") or
(task_stat.task_exists == False)
  retries: 12
  delay: 10
```

> The modules used in the above example were updated/added in Ansible version 2.5.

Path Formatting for Windows

Windows differs from a traditional POSIX operating system in many ways. One of the major changes is the shift from `/` as the path separator to `\`. This can cause major issues with how playbooks are written, since `\` is often used as an escape character on POSIX systems.

Ansible allows two different styles of syntax; each deals with path separators for Windows differently:

YAML Style

When using the YAML syntax for tasks, the rules are well-defined by the YAML standard:

- When using a normal string (without quotes), YAML will not consider the backslash an escape character.
- When using single quotes `'`, YAML will not consider the backslash an escape character.
- When using double quotes `"`, the backslash is considered an escape character and needs to escaped with another backslash.

> You should only quote strings when it is absolutely necessary or required by YAML, and then use single quotes.

The YAML specification considers the following escape sequences:

- `\0`, `\\`, `\"`, `\_`, `\a`, `\b`, `\e`, `\f`, `\n`, `\r`, `\t`, `\v`, `\L`, `\N` and `\P` -- Single character escape
- `<TAB>`, `<SPACE>`, `<NBSP>`, `<LNSP>`, `<PSP>` -- Special characters
- `\x..` -- 2-digit hex escape
- `\u....` -- 4-digit hex escape
- `\U........` -- 8-digit hex escape

Here are some examples on how to write Windows paths:

```
# GOOD
tempdir: C:\Windows\Temp


# WORKS
```

```
tempdir: 'C:\Windows\Temp'
tempdir: "C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir: C:\\Windows\\Temp
tempdir: 'C:\\Windows\\Temp'
tempdir: C:/Windows/Temp
```

This is an example which will fail:

```
# FAILS
tempdir: "C:\Windows\Temp"
```

This example shows the use of single quotes when they are required:

```
---
- name: Copy tomcat config
  win_copy:
    src: log4j.xml
    dest: '{{tc_home}}\lib\log4j.xml'
```

Legacy key=value Style

The legacy `key=value` syntax is used on the command line for ad hoc commands, or inside playbooks. The use of this style is discouraged within playbooks because backslash characters need to be escaped, making playbooks harder to read. The legacy syntax depends on the specific implementation in Ansible, and quoting (both single and double) does not have any effect on how it is parsed by Ansible.

The Ansible key=value parser parse_kv() considers the following escape sequences:

- `\`, `'`, `"`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t` and `\v` -- Single character escape
- `\x..` -- 2-digit hex escape
- `\u....` -- 4-digit hex escape
- `\U........` -- 8-digit hex escape
- `\N{...}` -- Unicode character by name

This means that the backslash is an escape character for some sequences, and it is usually safer to escape a backslash when in this form.

Here are some examples of using Windows paths with the key=value style:

```
# GOOD
tempdir=C:\\Windows\\Temp

# WORKS
tempdir='C:\\Windows\\Temp'
tempdir="C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir=C:\Windows\Temp
tempdir='C:\Windows\Temp'
tempdir="C:\Windows\Temp"
tempdir=C:/Windows/Temp

# FAILS
tempdir=C:\Windows\temp
tempdir='C:\Windows\temp'
tempdir="C:\Windows\temp"
```

The failing examples don't fail outright but will substitute `\t` with the `<TAB>` character resulting in `tempdir` being `C:\Windows<TAB>emp`.

Limitations

Some things you cannot do with Ansible and Windows are:

- Upgrade PowerShell
- Interact with the WinRM listeners

Because WinRM is reliant on the services being online and running during normal operations, you cannot upgrade PowerShell or interact with WinRM listeners with Ansible. Both of these actions will cause the connection to fail. This can technically be avoided by using `async` or a scheduled task, but those methods are fragile if the process it runs breaks the underlying connection Ansible uses, and are best left to the bootstrapping process or before an image is created.

Developing Windows Modules

Because Ansible modules for Windows are written in PowerShell, the development guides for Windows modules differ substantially from those for standard standard modules. Please see `developing_modules_general_windows` for more information.

`playbooks_intro`
An introduction to playbooks

`playbooks_best_practices`
Tips and tricks for playbooks

`List of Windows Modules <windows_modules>`
Windows specific module list, all implemented in PowerShell

`Communication<communication>`
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide

---

□□□ #2
□ □□□ □□□ 2026-10-31 14:40:00 UTC
□ □□□ □□□ 2026-10-31 14:40:01 UTC