Troubleshooting Ansible Playbook Execution: Common Issues and Solutions

Ansible is a robust automation tool, but users may encounter problems during playbook execution. This document details common issues and provides their solutions.

1. Error: Failed to Import PyVmomi

Description:

This error occurs when Ansible attempts to interact with a **VMware vSphere infrastructure** but cannot import the required **PyVmomi Python library**. PyVmomi is the Python SDK for the VMware vSphere API, and it must be installed on the Ansible Controller node. The issue is generally related to **environment configuration** rather than the playbook itself.

Symptoms:

- Task execution fails with a traceback ending in:

```
None
ModuleNotFoundError: No module named 'pyVim'
```

- Ansible error message:

```
None
Failed to import the required Python library (PyVmomi)...
Please read module documentation and install in the appropriate
location.
```

- Execution summary shows the playbook failing on VMware-related tasks.

Resolution:

- **Install PyVmomi using pip.**
    - Recommended (user-level install):

```
Shell
pip3 install --user PyVmomi
```

    - As root (less recommended):

```
Shell
sudo pip3 install PyVmomi
```

- **Verify installation.**
    - Run:

```
Shell
pip3 show PyVmomi
```

    - Ensure the library is visible in the Python environment used by Ansible.
- **Check Python interpreter.**
    - Confirm Ansible is using the correct Python version:

```
Shell
ansible --version
```

    - If needed, set `ansible_python_interpreter` in your inventory to match the environment where PyVmomi is installed.

Code

(Execution Failure → Fixed):

```
None
# Playbook snippet: VMware info gathering
```

```
- name: info vm Playbook
  hosts: all
  tasks:
    - name: get VM info
      community.vmware.vmware_vm_info:
        hostname: "vmware.example.com"
        username: "admin"
        password: "password"
        validate_certs: no
```

Failure Example (before installing PyVmomi):

```
None
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Failed
to import the required Python library (PyVmomi)..."}
```

Success Example (after installing PyVmomi):

```
None
TASK [get VM info]
**********************************************************************
***************
ok: [localhost]
```

Benefits of Installing PyVmomi Correctly:

- Enables seamless integration between Ansible and VMware vSphere.
- Ensures tasks like VM info retrieval, provisioning, and lifecycle management work as expected.
- Prevents runtime errors related to missing Python dependencies.
- Improves reliability of Ansible playbooks for VMware automation projects.

2. VMware Unknown error while connecting to vCenter or ESXi API, Name or service not known

Description:
Ansible fails to reach the VMware endpoint because the **hostname is wrong or cannot be resolved**, or there's a **network path issue** (VPN/firewall/DNS). This typically happens when a VMware playbook references a misspelled vCenter/ESXi host in variables (e.g., `vars.yml`) or when outbound connectivity to port **443** is blocked.

Symptoms:

- Fatal task failure when calling VMware modules (e.g., `vmware_guest_info`):

```
None
Unknown error while connecting to vCenter or ESXi API at
vm-ware.example.com:443 : [Errno -2] Name or service not known
```

- Play recap shows `failed=1` and no changes made.
- DNS lookup for the configured hostname fails from the Ansible controller.

Resolution:

- **Fix the hostname in variables/inventory.**
  Ensure `vcenter_hostname` exactly matches the real FQDN (no typos or extra characters).
- **Verify DNS resolution and reachability from the controller.**
  - `getent hosts <hostname>` or `nslookup <hostname>` should return an address.
  - Test TCP port 443: `nc -vz <hostname> 443` (or `telnet <hostname> 443`).
- **Check network path conditions.**
  - Connect your **VPN** if the environment requires it.

    - Confirm firewalls/security groups allow outbound 443 from the controller to vCenter/ESXi.

- **Keep TLS settings accurate.**
    - If using self-signed certs, `validate_certs: false` may be required (as in your example).

- **Re-run the playbook** after correcting the hostname and connectivity.

Code

(Incorrect → Correct):

```
None
# Incorrect vars.yml (typo in hostname)
---
vcenter_hostname: "vm-ware.example.com"
vcenter_datacenter: "vmwaredatacenter"
vcenter_validate_certs: false
vcenter_username: "username@vsphere.local"
vcenter_password: "MySecretPassword123"
vm_name: "myvm"
vcenter_destination_folder: "myvm"
vm_template: "mytemplate"
```

```
None
# Correct vars.yml (fixed hostname)
---
vcenter_hostname: "vmware.example.com"
vcenter_datacenter: "vmwaredatacenter"
vcenter_validate_certs: false
vcenter_username: "username@vsphere.local"
vcenter_password: "MySecretPassword123"
vm_name: "myvm"
vcenter_destination_folder: "myvm"
vm_template: "mytemplate"
```

None

```
# Playbook excerpt calling VMware API (vm_info.yml)
---
- name: info vm Playbook
  hosts: localhost
  gather_facts: false
  collections:
    - community.vmware
  pre_tasks:
    - include_vars: vars.yml
  tasks:
    - name: get VM info
      vmware_guest_info:
        hostname: "{{ vcenter_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        datacenter: "{{ vcenter_datacenter }}"
        validate_certs: "{{ vcenter_validate_certs }}"
        name: "{{ vm_name }}"
      register: detailed_vm_info

    - name: print VM info
      ansible.builtin.debug:
        var: detailed_vm_info
```

Shell

```
# Helpful controller-side checks
getent hosts vmware.example.com
nslookup vmware.example.com
nc -vz vmware.example.com 443   # or: telnet vmware.example.com
443
```

Benefits of this fix:

- Predictable connectivity to vCenter/ESXi and reliable module behavior.

- Faster troubleshooting by validating DNS and network before re-running playbooks.
- Clear separation of configuration (in `vars.yml`) from play logic, making typos easy to spot.

3. VMware: SSL `CERTIFICATE_VERIFY_FAILED` when connecting to vCenter/ESXi

Description:

When running VMware modules (for example, `vmware_guest_info`) Ansible may fail to connect to vCenter/ESXi with:
`[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed`.
This usually means the Ansible controller cannot validate the remote server's TLS certificate—commonly due to a self-signed cert or an incomplete chain of trust.

Symptoms:

- Task fails with a message similar to:

```
None
Unable to connect to vCenter or ESXi API at vmware.example.com on
TCP/443:
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
(_ssl.c:897)
```

- Play recap shows one failed task and no changes made.

Resolution:

- Preferable approach: **trust the VMware CA chain on the controller**
    - Obtain the vCenter/ESXi root and intermediate CA certificates.
    - Add them to the controller's trust store, then reload CA trust:
        - Debian/Ubuntu: place `.crt` in `/usr/local/share/ca-certificates/`, run `sudo update-ca-certificates`.

        - RHEL/CentOS/Rocky: place `.crt` in `/etc/pki/ca-trust/source/anchors/`, run `sudo update-ca-trust`.

    - Re-run the playbook to validate successfully with `validate_certs: true` (default).

- Pragmatic workaround (demo or lab only): **disable certificate validation**

    - Set `validate_certs: false` in the VMware task (or via a variable).

    - Note: This reduces security and should not be used in production.

Code

(Incorrect → Correct):

```
None
# Incorrect: no validate_certs provided, failing when the cert is
self-signed
- name: info vm Playbook
  hosts: localhost
  gather_facts: false
  collections:
    - community.vmware
  pre_tasks:
    - include_vars: vars.yml
  tasks:
    - name: get VM info
      vmware_guest_info:
        hostname: "{{ vcenter_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        datacenter: "{{ vcenter_datacenter }}"
        name: "{{ vm_name }}"
      register: detailed_vm_info
```

None

```
# Correct (workaround): disable TLS verification using a variable
flag
# vm_info.yml
- name: info vm Playbook
  hosts: localhost
  gather_facts: false
  collections:
    - community.vmware
  pre_tasks:
    - include_vars: vars.yml
  tasks:
    - name: get VM info
      vmware_guest_info:
        hostname: "{{ vcenter_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        datacenter: "{{ vcenter_datacenter }}"
        validate_certs: "{{ vcenter_validate_certs }}"
        name: "{{ vm_name }}"
      register: detailed_vm_info

# vars.yml
vcenter_hostname: "vmware.example.com"
vcenter_datacenter: "vmwaredatacenter"
vcenter_username: "username@vsphere.local"
vcenter_password: "MySecretPassword123"
vcenter_validate_certs: false
```

None

```
# Correct (preferred): keep validation enabled after trusting the
CA
# vars.yml (after installing the proper CA chain on the
controller)
vcenter_validate_certs: true
```

Why this works:

- Installing the correct CA chain lets Python's SSL stack verify vCenter/ESXi, so connections succeed securely.
- Setting `validate_certs: false` tells the module to skip verification, allowing connections to succeed at the cost of security.

Verification:

- Re-run the playbook and confirm the VMware task returns successfully.
- Optionally, print the gathered info to ensure data retrieval is working:

```
None
- name: print VM info
  ansible.builtin.debug:
    var: detailed_vm_info
```

Notes and best practices:

- Use `validate_certs: false` only for testing; restore verification for production.
- Keep vCenter/ESXi certificates and intermediary CAs up to date on the controller.
- Store sensitive credentials in Ansible Vault rather than plaintext variables.

4. Error: Windows 10 WSL Error 0x80370102

Description:
Error `0x80370102` appears when attempting to install or run **Windows Subsystem for Linux (WSL)** on Windows 10. It indicates that **the virtual machine could not be started because a required feature is not installed**. This typically happens when virtualization features (Hyper-V and WSL2 dependencies) are disabled or not supported by the hardware.

Symptoms:

- Installation of a Linux distribution fails with:

```
None
WslRegisterDistribution failed with error: 0x80370102
Error: 0x80370102 The virtual machine could not be started
because a required feature is not installed.
```

- WSL2 cannot launch distributions.
- Virtualization support (CPU/BIOS) may be missing or turned off.

Resolution:

- **Enable required Windows features for WSL2:**
    - Open PowerShell as Administrator and run:

```
None
dism.exe /online /enable-feature
/featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature
/featurename:VirtualMachinePlatform /all /norestart
dism.exe /online /enable-feature
/featurename:Microsoft-Hyper-V-All /all /norestart
```

    - Reboot the system after enabling features.
- **Verify CPU virtualization support:**
    - Ensure virtualization is enabled in BIOS/UEFI (Intel VT-x or AMD-V).
    - Check in Task Manager → Performance → CPU → "Virtualization: Enabled".
- **Workaround for unsupported hardware:**
    - If your system cannot support WSL2, use WSL1 instead:

```
None
wsl --set-default-version 1
```

- **Update WSL components:**
    - Run:

```
None
wsl --update
```

- **Install or re-install a Linux distribution:**
    - List available distributions:

```
None
wsl --list -o
```

    - Install Ubuntu (or another distro):

```
None
wsl --install -d Ubuntu
```

Code

(Problematic → Correct):

```
None
# Problematic: WSL2 install without required features
PS C:\Users\user> wsl --install
Installing: Virtual Machine Platform
Installing: Windows Subsystem for Linux
Installing: WSL Kernel
```

```
Downloading: Ubuntu
WslRegisterDistribution failed with error: 0x80370102
```

```
None
# Correct: Enable features, then set WSL1 if needed
PS C:\Users\user> wsl --set-default-version 1
The operation completed successfully.

PS C:\Users\user> wsl --update
Checking for updates...
```

Benefits of Fixing Error 0x80370102:

- Ensures smooth installation and use of WSL/WSL2.
- Provides the ability to run Linux distributions natively on Windows 10.
- Offers compatibility with Ansible and other automation tools in Windows environments.
- Workaround ensures functionality even on hardware without full virtualization support.

5. Error: Windows Subsystem for Linux – 0x80370102

Description:
The error `0x80370102` occurs when attempting to install or launch a **Windows Subsystem for Linux (WSL) distribution** on Windows 11. The full message is:

```
None
Error: 0x80370102 The virtual machine could not be started
because a required feature is not installed.
```

This error indicates that **required Windows features or virtualization support are missing**. It often arises when trying to use WSL version 2, which requires Hyper-V and virtualization capabilities that may not be supported by your CPU or system configuration.

Symptoms:

- Error appears when running `wsl --install` or launching a Linux distribution (e.g., Ubuntu).
- Message includes:

```
None
WslRegisterDistribution failed with error: 0x80370102
The virtual machine could not be started because a required
feature is not installed.
```

- On unsupported CPUs, an additional message may appear:

```
None
Hyper-V cannot be installed: The processor does not support
second level address translation (SLAT)
```

Resolution:

- **Verify required Windows features for WSL2 are enabled:**
  - Windows Subsystem for Linux
  - Virtual Machine Platform
  - Hyper-V Platform and Hyper-V Hypervisor
- These can be enabled via *Windows Features* or PowerShell.
- **If CPU or virtualization is not supported, use WSL1 as a workaround:**
  - Update WSL:

```
None
wsl --update
```

- Set WSL1 as the default:

```
None
wsl --set-default-version 1
```

- List available distributions:

```
None
wsl --list -o
```

- Install a distribution (e.g., Ubuntu):

```
None
wsl --install -d Ubuntu
```

Code

(Problematic → Correct):

```
None
# Problematic: Attempting WSL2 without required features
PS C:\Users\user> wsl --install
Installing: Ubuntu
WslRegisterDistribution failed with error: 0x80370102
Error: 0x80370102 The virtual machine could not be started
because a required feature is not installed.
```

```
None
# Correct: Workaround with WSL1
PS C:\Users\user> wsl --update
PS C:\Users\user> wsl --set-default-version 1
```

```
PS C:\Users\user> wsl --install -d Ubuntu
Ubuntu is already installed.
Launching Ubuntu...
Installation successful!
```

Benefits of Applying Fixes:

- Ensures WSL can be installed and launched successfully.
- Provides compatibility with older CPUs or environments lacking virtualization support.
- Offers flexibility: use WSL2 where supported, or fallback to WSL1 when needed.
- Improves developer productivity by enabling Linux workflows on Windows.

6. Error: chgrp failed

Description:
The `chgrp failed` error occurs in Ansible when a task tries to change the group ownership of a file or directory, but the operation fails. This typically happens because the specified group does not exist, the user running Ansible lacks sufficient privileges, or group membership is misconfigured.

Symptoms:

- Ansible task execution fails with an error like:

```
None
fatal: [host]: FAILED! => {"changed": false, "msg": "chgrp
failed: failed to change group of /path/to/file"}
```

- The user or group intended to be applied does not exist on the system.
- Group membership inconsistencies are revealed when checking with `groups <username>` or `id <username>`.

Resolution:

- **Verify the target group exists.**
    - Run `getent group <groupname>` or `cat /etc/group` to confirm the group.
- **Ensure the user belongs to the correct groups.**
    - Use:

```shell
groups devops
id devops
```

    - If necessary, add the user to a group:

```shell
usermod -aG users devops
```

- **Check playbook tasks.**
    - Confirm that the `group` parameter in modules such as `ansible.builtin.file` references a valid group.
- **Use privilege escalation if required.**
    - Add `become: true` at the play or task level to allow Ansible to apply ownership changes.

Code

(Incorrect → Correct):

```none
# Incorrect: Group might not exist or user not assigned
- name: Ensure file has correct group
```

```
  ansible.builtin.file:
    path: /home/devops/report.txt
    owner: devops
    group: users
    mode: '0644'
```

```
None
# Correct: Ensure group exists and user is assigned before file
task
- name: Ensure users group exists
  ansible.builtin.group:
    name: users
    state: present

- name: Ensure devops user is in users group
  ansible.builtin.user:
    name: devops
    groups: users
    append: yes

- name: Ensure file has correct group
  ansible.builtin.file:
    path: /home/devops/report.txt
    owner: devops
    group: users
    mode: '0644'
  become: true
```

Benefits of Fixing chgrp failed Errors:

- Prevents failures in file permission or ownership tasks.
- Ensures user and group configurations are consistent across environments.
- Increases reliability of automation by validating groups before applying changes.

- Improves security by enforcing correct group-based access controls.

7. Error: Destination Does Not Exist

Description:
The **"destination does not exist"** error occurs when the Ansible `get_url` module (or other file-related modules) attempts to download or copy a file to a location that does not exist on the target system. The error typically means that the destination path is missing or incorrectly defined.

Symptoms:

- Playbook execution fails with:

```
None
fatal: [demo.example.com]: FAILED! => {
  "msg": "Destination  does not exist"
}
```

- The task references a file in the home directory or relative path without properly specifying the directory.
- The download or copy never completes successfully.

Resolution:

- **Specify the full or relative path correctly.**
    - Use `./{{ variable }}` if saving in the current directory.
    - Provide an absolute path like `/home/devops/ansible-2.9.25.tar.gz`.
- **Ensure the destination directory exists.**
    - If needed, create the directory first with `ansible.builtin.file`.
- **Verify permissions.**

- Confirm the user running the playbook has write access to the target directory.

Code

(Incorrect → Correct):

```
None
# Incorrect: Missing directory in destination path
- name: Get_url module Playbook
  hosts: all
  vars:
    myurl:
"https://releases.ansible.com/ansible/ansible-2.9.25.tar.gz"
    mycrc:
"sha256:https://releases.ansible.com/ansible/ansible-2.9.25.tar.g
z.sha"
    mydest: "ansible-2.9.25.tar.gz"
  tasks:
    - name: Download file
      ansible.builtin.get_url:
        url: "{{ myurl }}"
        dest: "{{ mydest }}"
        checksum: "{{ mycrc }}"
        mode: '0644'
```

```
None
# Correct: Destination path fixed with current directory
- name: Get_url module Playbook
  hosts: all
  vars:
    myurl:
"https://releases.ansible.com/ansible/ansible-2.9.25.tar.gz"
    mycrc:
"sha256:https://releases.ansible.com/ansible/ansible-2.9.25.tar.g
z.sha"
    mydest: "ansible-2.9.25.tar.gz"
```

```
tasks:
  - name: Download file
    ansible.builtin.get_url:
      url: "{{ myurl }}"
      dest: "./{{ mydest }}"
      checksum: "{{ mycrc }}"
      mode: '0644'
```

Verification:

After running the fixed playbook, log in to the target host and confirm the file is present:

```
Shell
$ ssh devops@demo.example.com
[devops@demo ~]$ ls -al
-rw-r--r--. 1 devops wheel 14280306 Dec 26 21:47
ansible-2.9.25.tar.gz
```

Benefits of Fixing the Error:

- Ensures files are downloaded or copied successfully.
- Prevents wasted retries or failed automation steps.
- Improves reliability of playbooks dealing with remote file transfers.
- Clarifies exactly where files are placed on target systems.

8. Error: Failure Downloading

Description:
The **failure downloading** error occurs when Ansible cannot fetch content from a given URL. This is most often caused by an **incorrect or misspelled URL** or when the requested resource has been moved or deleted. Frequently, the error corresponds to an **HTTP 404 (Not Found)** response, meaning the file no longer exists at the specified location.

Symptoms:

- Playbook execution stops with a message indicating **failure downloading**.
- Associated with HTTP error codes (commonly `404 Not Found`).
- Occurs when modules like `unarchive` or `get_url` attempt to download from an invalid or outdated URL.

Resolution:

- **Double-check the URL:**
  - Verify the URL in a browser before using it in Ansible.
  - Ensure correct path segments (e.g., `refs/heads/master.zip` instead of `refs/master.zip`).
- **Update the playbook variable with the correct URL.**
- **Test URL reachability** using `curl` or `wget` before executing the playbook.
- **Validate certificate settings** if using `validate_certs: true` with HTTPS resources.

Code

(Incorrect → Correct):

```
None
# Incorrect: Invalid URL (missing 'heads/')
- name: unarchive module Playbook
  hosts: all
  become: false
  vars:
    myurl:
"https://github.com/lucab85/ansible-pilot/archive/refs/master.zip
"
  tasks:
    - name: extract archive
      ansible.builtin.unarchive:
        src: "{{ myurl }}"
```

```
        dest: "/home/devops/"
        remote_src: true
        validate_certs: true
```

```
None
# Correct: Valid URL including 'heads/master.zip'
- name: unarchive module Playbook
  hosts: all
  become: false
  vars:
    myurl:
"https://github.com/lucab85/ansible-pilot/archive/refs/heads/mast
er.zip"
  tasks:
    - name: extract archive
      ansible.builtin.unarchive:
        src: "{{ myurl }}"
        dest: "/home/devops/"
        remote_src: true
        validate_certs: true
```

Benefits of Fixing Failure Downloading Errors:

- Ensures reliable playbook execution by fetching the intended files.
- Prevents interruptions caused by HTTP 404 or broken links.
- Improves automation resilience by validating URLs before deployment.
- Enhances team collaboration by maintaining correct, working references to external resources.

9. Error: Fatal template error while templating string

Description:
This runtime error occurs when Ansible encounters **invalid syntax in a Jinja2 expression**. A

common cause is using unsupported characters (like ~ for the home directory) directly inside a variable definition. Ansible treats the tilde as a literal, leading to a **templating failure**.

Symptoms:

- Execution fails with a fatal error similar to:

```
None
fatal: [demo.example.com]: FAILED! => {"msg": "An unhandled
exception occurred while templating '{{ ~/example.txt }}'.
Error was a <class 'ansible.errors.AnsibleError'>, original
message: template error while templating string: unexpected '~'.
String: {{ ~/example.txt }}"}
```

- The error clearly points to the offending Jinja2 expression, typically involving improper use of ~.
- Tasks referencing the misformatted variable fail to execute.

Resolution:

- Do not use Jinja2 braces around paths with ~.
- Instead, assign the home path as a string:

```
None
myfile: "~/example.txt"
```

- Reference the variable correctly inside tasks:

```
None
path: "{{ myfile }}"
```

- Alternatively, expand paths dynamically using `ansible_env.HOME` or `expanduser`.

Code

(Incorrect → Correct):

```
None
# Incorrect: Invalid Jinja2 expression with '~'
- name: file module demo
  hosts: all
  vars:
    myfile: "{{ ~/example.txt }}"
  tasks:
    - name: Creating an empty file
      ansible.builtin.file:
        path: "{{ myfile }}"
        state: touch
```

```
None
# Correct: Properly formatted variable assignment
- name: file module demo
  hosts: all
  vars:
    myfile: "~/example.txt"
  tasks:
    - name: Creating an empty file
      ansible.builtin.file:
        path: "{{ myfile }}"
        state: touch
```

Benefits of Fixing the Error:

- Prevents runtime crashes during playbook execution.
- Ensures file paths expand correctly in all environments.
- Improves readability and correctness of variable definitions.
- Aligns with Ansible best practices for handling paths and Jinja2 templates.

10. Error: Invalid Argument

Description:
The **“Invalid Argument”** error occurs when using the **Ansible `file` module** to create a symbolic link but providing incorrect or incomplete parameters. Specifically, the module requires both a **`src`** (source file to link from) and a **`dest`** (destination path to link to). If `src` is missing or parameters are misused, the task fails with this error.

Symptoms:

- Playbook execution fails with:

```
None
OSError: [Errno 22] Invalid argument: b'/proc/cpuinfo'
```

- Occurs when creating symbolic links using `state: link` in the `file` module.
- Verbose execution (`-vvv`) shows traceback pointing to missing or invalid parameters in the file module.

Resolution:

- Ensure both `src` (source file) and `dest` (destination symlink path) are provided.
    - Correct:

```
None
src: "/proc/cpuinfo"
dest: "~/example"
state: link
```

    - Incorrect:

```
None
path: "~/example"
```

```
dest: "/proc/cpuinfo"
state: link
```

- Check parameter names — use `src` for the source and `dest` for the symlink target.
- Verify that the source path exists on the managed host.
- Consult the **`ansible.builtin.file`** documentation for required arguments when creating symlinks.

Code

(Incorrect → Correct):

None

```
# Incorrect: Missing src parameter
- name: file module demo
  hosts: all
  vars:
    mylink: "~/example"
    mysrc: "/proc/cpuinfo"
  tasks:
    - name: Creating a symlink
      ansible.builtin.file:
        path: "{{ mylink }}"
        dest: "{{ mysrc }}"
        state: link
```

None

```
# Correct: Proper src and dest parameters
- name: file module demo
  hosts: all
  vars:
    mylink: "~/example"
```

```
    mysrc: "/proc/cpuinfo"
  tasks:
    - name: Creating a symlink
      ansible.builtin.file:
        src: "{{ mysrc }}"
        dest: "{{ mylink }}"
        state: link
```

Benefits of Fixing Invalid Argument Errors:

- Predictable behavior when creating symlinks.
- Avoids runtime errors due to misused parameters.
- Aligns with module documentation and best practices.
- Enhances playbook maintainability and clarity for future readers.

11. Error: Missing Module Parameter

Description:
The “Missing Module Parameter” error occurs when a required parameter in an Ansible module is misspelled, omitted, or incorrectly defined. Since Ansible modules expect specific parameter names, any mismatch results in task failure. This often happens due to **typos in parameter names** or forgetting mandatory parameters.

Symptoms:

- Playbook execution fails with messages such as:

```
None
fatal: [example.com]: FAILED! => {"msg": "An unhandled exception
occurred while templating '{{ nme }}'... unexpected 'n'."}
```

- The error points to an undefined or invalid variable where the module expected a parameter.

- Task fails immediately and does not apply changes to the target system.

Resolution:

- Double-check **module documentation** (`ansible-doc <module_name>`) to confirm required parameters.
- Ensure parameter names are spelled correctly (e.g., `name` instead of `nme`).
- Add all mandatory parameters explicitly.
- Validate your playbook with `ansible-playbook --syntax-check` before execution to catch such issues early.

Code

(Incorrect → Correct):

None

```
# Incorrect: Missing/typo in parameter name
- name: Service module Playbook
  hosts: all
  become: true
  tasks:
    - name: Sshd restart
      ansible.builtin.service:
        nme: sshd      # Typo: should be "name"
        state: restarted
        enabled: true
```

None

```
# Correct: Proper parameter name
- name: Service module Playbook
  hosts: all
  become: true
  tasks:
    - name: Sshd restart
```

```
    ansible.builtin.service:
      name: sshd     # Correct spelling
      state: restarted
      enabled: true
```

Benefits of Fixing Missing Module Parameters:

- Ensures Ansible modules run with the expected configuration.
- Prevents task execution failures caused by typos or missing arguments.
- Improves reliability of automation by validating inputs.
- Enhances readability and maintainability of playbooks.

12. Error: Missing or Incorrect Sudo Password

Description:
Ansible requires privilege escalation (`become: true`) to perform tasks as another user (commonly `root`). If the sudo password is not provided, you will see a **"Missing sudo password"** error. If the password is provided but incorrect, you will see an **"Incorrect sudo password"** error. These issues occur when **Ansible cannot authenticate with sudo** during privilege escalation.

Symptoms:

- Missing password error:

```
None
fatal: [demo.example.com]: FAILED! => {"msg": "Missing sudo
password"}
```

- Incorrect password error:

```
None
fatal: [demo.example.com]: FAILED! => {"msg": "Incorrect sudo
password"}
```

- Tasks requiring elevated privileges fail, while normal user tasks may succeed.

Resolution:

- **Provide the sudo password at runtime**:
    - Use the `-K` or `--ask-become-pass` flag with `ansible-playbook`:

```
Shell
ansible-playbook -i inventory playbook.yml -bK
```

    - Ansible will prompt you for the sudo password.
- **Configure passwordless sudo for the Ansible user (recommended in automation):**
    - Edit `/etc/sudoers.d/<username>` or `/etc/sudoers` and add:

```
None
devops ALL=(ALL) NOPASSWD: ALL
```

    - This allows the specified user to run sudo commands without a password.
- **Verify sudo privileges manually:**
    - Log into the target host:

```
Shell
ssh devops@demo.example.com
sudo su
```

- Confirm whether the password is required and works correctly.

Code

(Incorrect → Correct):

```
None
# Incorrect: Task fails because no sudo password is provided
- name: debug module Playbook
  hosts: all
  become: true
  tasks:
    - name: root test
      ansible.builtin.debug:
        msg: "privilege escalation successful"
```

```
None
# Correct: With passwordless sudo configured
# /etc/sudoers.d/devops contains:
# devops ALL=(ALL) NOPASSWD: ALL

- name: debug module Playbook
  hosts: all
  become: true
  tasks:
    - name: root test
      ansible.builtin.debug:
        msg: "privilege escalation successful"
```

Benefits of Fixing Sudo Password Issues:

- Ensures privilege escalation works reliably.
- Allows automation to run unattended (when using NOPASSWD).
- Eliminates repeated prompts for sudo passwords.

- Reduces task failures caused by incorrect authentication.

13. Error: not a valid attribute for a Play

Description:
The “not a valid attribute for a Play” error occurs when an **invalid keyword** is used in a playbook. Ansible playbooks have a strict structure, and only specific attributes are allowed at the play level (e.g., `hosts`, `vars`, `tasks`, `roles`). A common mistake is using `task` instead of `tasks`, or introducing a misspelled attribute.

Symptoms:

- Execution fails immediately with:

```
None
ERROR! 'task' is not a valid attribute for a Play
```

- The error message points to the line where the invalid attribute is used.
- Playbook fails to start, and no tasks are executed.

Resolution:

- Review the playbook for incorrect or misspelled attributes.
    - Use `tasks` (plural), not `task`.
- Confirm valid play-level attributes such as `hosts`, `vars`, `tasks`, `roles`, `gather_facts`, etc.
- Run `ansible-playbook --syntax-check` to detect structural issues early.

Code

(Incorrect → Correct):

None

```
# Incorrect: Using 'task' instead of 'tasks'
- name: file module demo
  hosts: all
  vars:
    myfile: "~/example.txt"
  task:
    - name: Creating an empty
      ansible.builtin.file:
        path: "{{ myfile }}"
        state: touch
```

None

```
# Correct: Using 'tasks'
- name: file module demo
  hosts: all
  vars:
    myfile: "~/example.txt"
  tasks:
    - name: Creating an empty
      ansible.builtin.file:
        path: "{{ myfile }}"
        state: touch
```

Benefits of Fixing Invalid Attributes:

- Ensures the playbook structure is compliant with Ansible syntax.
- Prevents execution from halting at the start due to schema validation errors.
- Improves playbook readability and maintainability.
- Reduces the risk of similar errors when reusing playbook templates.