Common Issues in Ansible Playbook Execution

Ansible is a powerful automation tool, but users can face challenges during playbook execution. This document outlines frequent problems and offers solutions to address them.

1. Error: Passwordless Account

Description:
This error occurs when attempting to unlock a user account in Ansible without setting a password. By default, Linux prevents unlocking accounts without a password because it would create a **security risk** (a passwordless login). The error message typically appears when using the `ansible.builtin.user` module with `password_lock: false` but no `password` parameter defined.

Symptoms:

- Playbook execution fails with an error like:

```
None
usermod: unlocking the user's password would result in a
passwordless account.
You should set a password with usermod -p to unlock this user's
password.
```

- The targeted user account cannot be enabled.
- The task fails immediately and no password is set.

Resolution:

- When unlocking a user account (`password_lock: false`), ensure you also provide a password.
- Use the `password` parameter with a hashed value, generated via Ansible's `password_hash` filter.
- Always hash passwords instead of providing plaintext for security reasons.

Code

(Incorrect → Correct):

None

```
# Incorrect: Unlocking user without setting password
- name: user module Playbook
  hosts: all
  become: true
  vars:
    myuser: "example"
  tasks:
    - name: create a disabled user
      ansible.builtin.user:
        name: "{{ myuser }}"
        state: present
        password_lock: true

    - name: enable user
      ansible.builtin.user:
        name: "{{ myuser }}"
        state: present
        password_lock: false   # No password provided → error
```

None

```
# Correct: Unlocking user with password set
- name: user module Playbook
  hosts: all
  become: true
  vars:
    myuser: "example"
    mypassword: "password"
  tasks:
    - name: create a disabled user
      ansible.builtin.user:
        name: "{{ myuser }}"
        state: present
        password_lock: true
```

```
- name: enable user
  ansible.builtin.user:
    name: "{{ myuser }}"
    password: "{{ mypassword | password_hash('sha512') }}"
    state: present
    password_lock: false
```

Benefits of Fixing Passwordless Account Error:

- Security — prevents creation of accounts without a password.
- Reliability — ensures user accounts can be unlocked properly.
- Compliance — aligns with best practices for Linux account management.
- Maintainability — playbooks behave predictably when managing users.

2. Error: “The PowerShell shell family is incompatible with the sudo become plugin”

Description:
This runtime error occurs when a Windows play (which uses the **PowerShell** shell via WinRM) attempts privilege escalation with the default **sudo** become plugin. The sudo plugin is for Unix-like systems and is not supported on Windows. For Windows hosts, either avoid become entirely or use the Windows-supported **runas** become method.

Symptoms:

- Playbook fails on Windows hosts with:

```
None
fatal: [WindowsServer]: FAILED! => {"msg": "The PowerShell shell family is incompatible with the sudo become plugin"}
```

- Failure happens as soon as Ansible gathers facts or runs the first task with `become: true` under PowerShell.

Resolution:

Choose one of the two correct approaches:

Do not use become on Windows tasks

- Set `become: false` at play or task level when administrative elevation is not required (many Windows modules already perform the needed action under the authenticated user context).

Use the Windows-supported runas become method

- If elevation is required, explicitly select `become_method: runas` (and optionally `become_user`, e.g., `Administrator`).
- Provide credentials appropriately (inventory, vars, or prompt), and avoid sudo on Windows.

Code

(Incorrect → Correct):

```
None
# Incorrect: Using sudo-style become with a Windows play
- name: win_reboot module Playbook
  hosts: all
  become: true
  tasks:
    - name: reboot host(s)
      ansible.windows.win_reboot:
```

```
None
# Correct option 1: No become on Windows
- name: win_reboot module Playbook
  hosts: all
  become: false
  tasks:
    - name: reboot host(s)
```

```
      ansible.windows.win_reboot:
```

```
None
# Correct option 2: Use runas for Windows elevation
- name: Elevated Windows task with runas
  hosts: all
  vars:
    ansible_become_method: runas
    ansible_become_user: Administrator   # or another admin user
    # ansible_become_password: "{{ vault_admin_password }}"  #
supply securely
  tasks:
    - name: Reboot host(s) with elevation
      become: true
      ansible.windows.win_reboot:
```

Benefits of Applying the Fix:

- Prevents immediate task failure on Windows hosts using PowerShell.
- Uses the correct privilege escalation mechanism for Windows (`runas`) when elevation is required.
- Improves clarity by separating Linux sudo usage from Windows runas usage.
- Ensures predictable behavior across mixed OS environments in the same automation codebase.

3. Error: Role Not Found

Description:
This error occurs when a play references a role that Ansible cannot locate in any of the configured role paths. Typical causes include the role not being installed locally, a misspelled role name, or incorrect role path configuration.

Symptoms:

- Playbook fails with a message similar to:

```
None
ERROR! the role 'lucab85.ansible_role_log4shell' was not found in
/project/roles:~/.ansible/roles:/usr/share/ansible/roles:/etc/ans
ible/roles:...
```

- The failure points to the `roles:` section of your playbook.

Resolution:

- **Declare the role in `requirements.yml` and install it with Ansible Galaxy.**
- **Re-run the playbook** after installation to confirm the role is resolved.
- (Optional) **Verify or set role paths** via `ANSIBLE_ROLES_PATH` or `ansible.cfg` if you use a custom directory structure.
- **Double-check the role name spelling** exactly matches the Galaxy name.

Code

(Incorrect → Correct):

```
None
# Incorrect: Playbook references a role that isn't installed
locally
# role.yml
---
- name: role Playbook
  hosts: all
  become: true
  roles:
    - role: lucab85.ansible_role_log4shell
      detector_path: "/var"
```

```
None
# Correct Step 1: Declare the role dependency
# requirements.yml
---
roles:
  - name: lucab85.ansible_role_log4shell
```

```
Shell
# Correct Step 2: Install the role, then run the playbook
ansible-galaxy install -r troubleshooting/role/requirements.yml
ansible-playbook -i virtualmachines/demo/inventory
troubleshooting/role/role.yml
```

Benefits of Fixing Role Resolution:

- Ensures reusable role code is correctly fetched and available at runtime.
- Eliminates “role not found” failures stemming from missing dependencies.
- Encourages consistent, documented dependency management across teams.

4. Error: Undefined Variable

Description:
An Ansible task references a variable that has not been defined in the play, inventory, group/host vars, included vars, facts, or extra vars. This is commonly a scoping or precedence issue, or simply a missing `vars` definition.

Symptoms:

- Playbook fails with an error similar to:

```
None
The task includes an option with an undefined variable. The error
was: 'fruit' is undefined
```

- Ansible points to the task and line where the variable was used.

Resolution:

- Define the variable in an appropriate place (play `vars`, `group_vars/`, `host_vars/`, inventory, role defaults/vars, or via `--extra-vars`).
- If a variable may be absent, provide a safe fallback using Jinja's `default` filter.
- For debugging, print candidate sources with `debug` and verify variable precedence.
- If the value must exist, fail early and clearly using `assert` or `fail`.

Code

(Incorrect → Correct):

```
None
# Incorrect: variable 'fruit' never defined
- name: debug module Playbook
  hosts: all
  tasks:
    - name: debug message
      ansible.builtin.debug:
        msg: "{{ fruit }}"
```

```
None
# Correct: define the variable in the play
- name: debug module Playbook
  hosts: all
  vars:
    fruit: "apple"
  tasks:
    - name: debug message
      ansible.builtin.debug:
        msg: "{{ fruit }}"
```

Alternative Safe Patterns:

```
None
# Safe fallback with default
- name: safe debug with default
  hosts: all
  tasks:
    - name: debug message with fallback
      ansible.builtin.debug:
        msg: "{{ fruit | default('unknown') }}"
```

```
None
# Define from inventory or group_vars/host_vars (example
inventory.ini)
[all]
demo.example.com fruit=apple
```

```
None
# Enforce presence with an assertion
- name: require fruit to be defined
  hosts: all
  tasks:
    - name: fruit must be provided
      ansible.builtin.assert:
        that:
          - fruit is defined
        fail_msg: "Variable 'fruit' is required but not defined."
```

Benefits of Fixing Undefined Variables:

- Predictable execution with clear variable sources and precedence.
- Reduced runtime failures by supplying defaults or assertions.
- Easier troubleshooting due to explicit definitions and checks.

5. Error: url open error

Description:
This error appears when the `ansible.builtin.uri` module cannot reach the target URL. Common causes include a misspelled domain, DNS resolution issues, network/proxy problems, SSL verification failures, or timeouts. In the example, the domain was mistyped (`reqres.it` instead of `reqres.in`), causing DNS to fail.

Symptoms:

- Task fails with status code `-1` and a message similar to:

```
None
Request failed: <urlopen error [Errno -2] Name or service not known>
```

- `ansible.builtin.uri` task does not receive the expected HTTP status (e.g., 200).
- `elapsed` shows time spent before failure; no content returned in `result`.

Resolution:

- **Verify the URL:** Check domain, scheme (http/https), path, and query string.
- **Test connectivity from the managed host:**
    - Use `getent hosts <domain>` or `nslookup`/`dig` to confirm DNS resolution.
    - Use `curl -I <url>` or `wget --spider <url>` if available.
- **Check proxies and firewalls:** Ensure outbound access is allowed and proxy env vars are set correctly if needed (`http_proxy`/`https_proxy`).
- **Validate TLS/SSL settings:** If SSL verification fails, either provide proper CA certs or set `validate_certs: true` with the correct CA bundle (avoid disabling cert checks in production).
- **Adjust timeouts and expected status codes** as appropriate.

- **Re-run the playbook** after correcting the URL or environment issue.

Code

(Incorrect → Correct):

```
None
# Incorrect: Misspelled domain causes DNS failure
- name: uri module Playbook
  hosts: all
  become: false
  vars:
    server: "https://reqres.it"
    endpoint: "/api/users?page=2"
  tasks:
    - name: list users
      ansible.builtin.uri:
        url: "{{ server }}{{ endpoint }}"
        method: GET
        status_code: 200
        timeout: 30
      register: result

    - name: debug
      ansible.builtin.debug:
        var: result.json.data
```

```
None
# Correct: Fixed domain; request succeeds
- name: uri module Playbook
  hosts: all
  become: false
  vars:
    server: "https://reqres.in"
    endpoint: "/api/users?page=2"
  tasks:
```

```
    - name: list users
      ansible.builtin.uri:
        url: "{{ server }}{{ endpoint }}"
        method: GET
        status_code: 200
        timeout: 30
      register: result

    - name: debug
      ansible.builtin.debug:
        var: result.json.data
```

Helpful diagnostics (optional tasks you can add):

```
None
- name: Check DNS resolution from target
  ansible.builtin.command: "getent hosts reqres.in"
  register: dns_result
  changed_when: false

- name: Show DNS resolution
  ansible.builtin.debug:
    var: dns_result.stdout

- name: Simple HEAD request for quick connectivity check
  ansible.builtin.uri:
    url: "https://reqres.in"
    method: HEAD
    status_code: 200
  register: head_check
```

Benefits of fixing urlopen errors:

- Reliable HTTP interactions for APIs and web services.

- Faster troubleshooting with clear URL and connectivity checks.
- More predictable automation runs that fail fast and informatively.

6. Error: SSH with Passwords Requires sshpass

Description:
This error occurs when using the **SSH connection type** in Ansible with either a password or `pkcs11_provider` authentication. Ansible relies on the `sshpass` utility to handle non-interactive password passing. If `sshpass` is missing, playbook execution fails with:

```
None
to use the ssh connection type with passwords or pkcs11_provider,
you must install the sshpass program
```

Symptoms:

- Playbook execution stops immediately with the above error.
- Ansible cannot authenticate to the target host using password-based SSH.
- The error occurs even if the inventory contains `ansible_ssh_pass`.

Resolution:

- **Check if sshpass is installed:**

```
Shell
sshpass -V
```

- If the command is not found, `sshpass` is missing.
- **Install sshpass:**
    - On Debian/Ubuntu:

```shell
sudo apt-get install sshpass
```

- On RHEL/CentOS/Fedora:

```shell
sudo yum install sshpass
```

- On macOS (via Homebrew):

```shell
brew install hudochenkov/sshpass/sshpass
```

- **Specify the password in your Ansible inventory:**
  Use the `ansible_ssh_pass` variable for password authentication.

- **Set custom path to sshpass (if installed in non-standard location):**
  Use the `ansible_ssh_executable` variable to define the correct path.

Code

(Inventory Example):

```
[servers]
host1 ansible_ssh_user=myuser ansible_ssh_pass=mypassword
ansible_ssh_executable=/usr/local/bin/sshpass
```

In this example:

- `ansible_ssh_user` specifies the remote user.

- `ansible_ssh_pass` provides the password for SSH authentication.

- `ansible_ssh_executable` ensures Ansible finds `sshpass` if it's not in the default path.

Benefits of Fixing sshpass Requirement:

- Enables smooth execution of playbooks using password-based SSH.
- Ensures compatibility with systems that cannot use SSH keys.
- Provides flexibility by supporting `pkcs11_provider` authentication.
- Avoids manual password prompts, making automation fully non-interactive.

7. Error: user Module `password_expire_min` Bug

Description:
A bug in Ansible's **`user` module** prevents the correct handling of the `password_expire_min` parameter when set alongside `password_expire_max`. While the task appears to succeed, the minimum number of days between password changes defaults to **0 instead of the specified value**.

Symptoms:

- Playbook runs without errors, but the system settings do not reflect the expected values.
- Example output from `chage -l <user>` shows:

```
None
Minimum number of days between password change  : 0
Maximum number of days between password change  : 90
```

- Linter/playbook execution reports tasks as `ok` or `changed`, masking the underlying misconfiguration.

Resolution:

- Split the configuration into **two separate tasks** — one for `password_expire_min`, another for `password_expire_max`.
- Apply them in sequence to ensure values are written correctly.
- Validate results with `chage -l <user>` on the target host.

Code

(Problematic → Workaround):

```
None
# Problematic: password_expire_min ignored when combined with
password_expire_max
- name: user module Playbook
  hosts: all
  become: true
  vars:
    myuser: "example"
  tasks:
    - name: password expiration
      ansible.builtin.user:
        name: "{{ myuser }}"
        password_expire_min: 7
        password_expire_max: 90
```

```
None
# Workaround: Separate tasks for min and max expiration
- name: user module Playbook
  hosts: all
  become: true
  vars:
    myuser: "example"
  tasks:
    - name: password min expiration
      ansible.builtin.user:
```

```
        name: "{{ myuser }}"
        password_expire_min: 7

    - name: password max expiration
      ansible.builtin.user:
        name: "{{ myuser }}"
        password_expire_max: 90
```

Benefits of Workaround:

- Ensures both `password_expire_min` and `password_expire_max` are applied correctly.
- Provides predictable results (`chage` shows expected values).
- Avoids silent misconfigurations that could lead to weaker security policies.
- Aligns with bug report [#75017] and fix proposal [#75390], pending upstream resolution.

8. Error: SSH Connection Failure

Description:
Ansible relies on **SSH** to connect to target machines. A connection failure occurs when Ansible cannot establish an SSH session, typically due to **network issues, incorrect hostnames, firewall restrictions, or unreachable ports**. This prevents tasks from running on the remote host.

Symptoms:

- Playbook execution fails with messages such as:

```
None
Failed to connect to the host via ssh: ssh: connect to host
hostname port 22: Operation timed out
```

- Manual SSH attempts also fail:

```shell
ssh username@hostname
ssh: connect to host hostname port 22: Operation timed out
```

- Target host is unreachable or does not respond on port 22.

Resolution:

- **Verify network connectivity:**
    - Ping the host:

```shell
ping hostname
```

    - Check if port 22 is reachable:

```shell
nc -zv hostname 22
```

- **Test SSH manually:**

```shell
ssh username@hostname
```

    - If it fails, resolve DNS, firewall, or routing issues.
    - If it succeeds, Ansible should also connect once inventory variables are correct.
- **Ensure correct inventory configuration:**
    - Verify hostnames and IP addresses in your `inventory`.

    - Specify the correct user with `ansible_user`.

- **Check firewall and security group rules:**
    - Ensure SSH (port 22) is open between the control node and the managed host.
- **Confirm the target machine is running and accessible.**

Code

(Manual Test Example):

```
Shell
# Incorrect: Network issue prevents SSH
$ ssh username@hostname
ssh: connect to host hostname port 22: Operation timed out
```

```
Shell
# Correct: Connection works after network fix
$ ssh username@hostname
username@hostname:~$
```

Benefits of Fixing SSH Connection Failures:

- Ensures Ansible can reach and manage target hosts.
- Prevents wasted time debugging playbooks when the root cause is network-related.
- Improves reliability of automation by confirming infrastructure connectivity.
- Enables successful task execution across distributed systems.

9. Error: Indentation Error

Description:
Indentation errors are among the most common issues in Ansible playbooks. Since playbooks are written in YAML, **whitespace and indentation matter**. An incorrect number of spaces or

misplaced dashes (`-`) can cause parsing failures. Even a single extra or missing space may lead to errors when executing tasks.

Symptoms:

- Playbook execution fails immediately with YAML or syntax parsing errors.
- Ansible may report messages like:

```
None
ERROR! We were unable to read either as JSON nor YAML
found character that cannot start any token
found unexpected key
```

- Tasks appear “detached” from their parent sections (e.g., a task not under `tasks:`).

Resolution:

- Ensure **consistent indentation** throughout the playbook. YAML requires spaces, not tabs.
- Verify that **tasks under `tasks:` are indented two spaces** from the parent key.
- Use a YAML linter (e.g., `yamllint`) or `ansible-playbook --syntax-check` to validate before running.
- Always align modules and their parameters consistently under task names.

Code

(Incorrect → Correct):

```
None
# Incorrect: Misaligned indentation
- name: blockinfile module demo
  hosts: all
  become: true
```

```
  tasks:
- name: Generate /etc/hosts file
  ansible.builtin.blockinfile:
    state: present
    dest: /etc/hosts
    content: |
      192.168.0.200 Playbook demo.example.com
```

```
None
# Correct: Proper indentation with tasks indented under parent
- name: blockinfile module demo
  hosts: all
  become: true
  tasks:
    - name: Generate /etc/hosts file
      ansible.builtin.blockinfile:
        state: present
        dest: /etc/hosts
        content: |
          192.168.0.200 Playbook demo.example.com
```

Benefits of Fixing Indentation Errors:

- Playbooks execute successfully without YAML parsing issues.
- Code becomes easier to read and maintain.
- Prevents confusion when tasks are misplaced under the wrong hierarchy.
- Ensures compatibility with tools like `ansible-lint` and `yamllint`.

10. Error: Privilege Escalation Errors

Description:
These errors occur when the SSH connection user does not have sufficient permissions to perform an operation (e.g., installing packages, editing system files, managing services).

Ansible must **escalate privileges** to an administrative user by enabling `become`. The default method is `sudo`, but others exist (e.g., `su`, `runas` on Windows, `pfexec`, `doas`, `pbrun`, `dzdo`, `ksu`, `machinectl`, Centrify, and more).

Symptoms:

- Task failures with messages like:
    - `FAILED! => {"msg": "You need to be root to perform this command"}`
    - `permission denied`, `access denied`, or module-specific privilege errors
- Package, service, file, or template tasks fail when targeting system paths or privileged operations
- Works when run manually with `sudo`, but fails via Ansible without `become`

Resolution:

- **Enable privilege escalation** where needed:
    - At the play level with `become: true`
    - Or at the task level for specific privileged actions
- **Optionally specify method and user**:
    - `become_method: sudo` (default on most Unix-like targets)
    - `become_user: root` (or another admin user)
- **If prompted for a password** and sudo requires one:
    - Run with `--ask-become-pass` (or configure `ansible_become_password` securely)
- **Ensure the remote user is allowed to escalate**:
    - Confirm sudoers policy (`/etc/sudoers` or included files) allows the user to run the required commands, preferably without a TTY if not needed

- **Windows targets** use `become_method: runas` (privilege model differs from sudo)

Code

(Incorrect → Correct):

```
None
# Incorrect: No privilege escalation for a privileged operation
- name: yum module Playbook
  hosts: all
  become: false
  tasks:
    - name: install package
      yum:
        name: git
        state: present
```

```
None
# Correct: Play-level privilege escalation
- name: yum module Playbook
  hosts: all
  become: true
  tasks:
    - name: install package
      yum:
        name: git
        state: present
```

```
None
# Correct: Task-level privilege escalation with explicit
method/user
- name: Install package with sudo as root
  hosts: all
  tasks:
```

```
    - name: install package
      yum:
        name: git
        state: present
      become: true
      become_method: sudo
      become_user: root
```

Shell

```
# Running playbook when sudo requires a password
ansible-playbook -i inventory play.yml --ask-become-pass
```

Benefits of Fixing Privilege Escalation:

- Security and predictability — privileged tasks run under the correct account
- Clarity — explicit `become` usage documents intent for reviewers
- Fewer failures — avoids permission-related task errors
- Flexibility — choose per-task or play-wide escalation and methods appropriate to the OS and policy

11. Error: macOS fork error (`OBJC` initialize during `fork()`)

Description:
On macOS, Ansible (via Python) may load Objective-C frameworks that aren't fork-safe. When a task triggers a `fork()` while an Objective-C class is initializing, macOS aborts the child process and prints an `objc` error. This manifests as a crash during playbook runs on macOS controllers.

Symptoms:

- Terminal shows messages like:

```
None
objc[22868]: +[__NSCFConstantString initialize] may have been in
progress in another thread when fork() was called.
objc[22868]: ... We cannot safely call it or ignore it in the
fork() child process. Crashing instead.
Set a breakpoint on objc_initializeAfterForkError to debug.
```

- Ansible playbook stops unexpectedly on macOS.

Resolution:

- Set the environment variable to disable the fork-safety check **for the current session**.
- Persist the environment variable for **future sessions** (shell startup file).
- Verify the variable is set before running Ansible.

Code

(Fix — current session):

```shell
# Enable workaround for this terminal session only
export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES

# Run your playbook
ansible-playbook -i inventory site.yml
```

(Fix — all future sessions):

```shell
# If you use zsh (default on modern macOS)
echo 'export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES' >> ~/.zshrc

# If you use bash
```

```
echo 'export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES' >>
~/.bash_profile
```

(Verify):

```shell
env | grep OBJC_DISABLE_INITIALIZE_FORK_SAFETY
# Expected:
# OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES
```

Notes:

- Set the variable in the **shell that launches Ansible** (Terminal, iTerm, CI runner, etc.).
- After persisting, **restart** your terminal or `source` your startup file (e.g., `source ~/.zshrc`).
- Consider scoping the variable narrowly (e.g., only when calling Ansible) if you prefer minimal global changes.

Benefits of Applying This Fix:

- Prevents Objective-C fork-safety crashes on macOS during Ansible runs.
- Restores predictable playbook execution on macOS controllers.
- Works for both one-off sessions and persistent developer environments.