Troubleshooting Ansible Playbook Execution

Ansible is an effective automation tool, though users may encounter challenges. This document details common issues and their resolutions.

1. Pod Information Retrieval Timeout

Description:
Ansible fails to collect Pod information within the configured timeout period. This typically occurs when the cluster API is slow or unresponsive, the cluster is under heavy load, or there are a large number of Pods to process.

Symptoms:

- Operation fails with a **timeout** after 601 seconds
- Logs show messages like *"Failed to gather Pod information within 601 seconds"*
- Tasks that query Pods hang before aborting
- Cluster API responses are slow or intermittent

Resolution:

- Verify the Kubernetes API server is healthy and responsive
- Check cluster load and reduce pressure if possible
- Narrow the Pod query scope (namespace/label selectors)
- Increase the timeout value when querying large clusters
- Retry after cluster performance stabilizes

Code

```shell
oc get pods --all-namespaces
oc get --raw /healthz
```

2. Missing Host Variable Attribute

Description:
An Ansible task references a host variable attribute that is not defined in hostvars. This usually happens when a variable was never set, is conditionally created but not present for all hosts, or is referenced with an incorrect name or scope.

Symptoms:

- Playbook fails with an **attribute missing** or **undefined variable** error
- Error message points to a specific variable attribute in a task or template
- Failure occurs during variable access from `hostvars`
- Task stops execution in the affected playbook or role

Resolution:

- Ensure the variable attribute is defined for all relevant hosts
- Check `host_vars/`, `group_vars/`, inventory, and play-level variables
- Verify the variable name and attribute path are correct
- Provide a safe fallback using Jinja `default` filter if the value may be absent
- Add an assertion to fail early if the attribute is required

Code

```
None
# Safe access with fallback

{{ hostvars[inventory_hostname].student_password | default('') }}

# Enforce presence

- ansible.builtin.assert:

    that:

      - hostvars[inventory_hostname].student_password is defined
```

3. Cluster API Connection Refused

Description:
Ansible fails to connect to the cluster API endpoint because the connection is repeatedly refused. This usually indicates that the API server is not running, not yet ready, or is unreachable due to networking, DNS, or firewall issues.

Symptoms:

- Connection attempts to the cluster API fail with **“connection refused”**
- Retries are exhausted without a successful connection
- Tasks interacting with the cluster API abort early
- Manual access to the API endpoint on port **6443** also fails

Resolution:

- Verify the cluster API server is running and healthy
- Check DNS resolution for the API endpoint
- Ensure network routes, firewalls, and load balancers allow access to port **6443**
- Confirm the cluster is fully initialized and not still starting up
- Retry after connectivity to the API endpoint is restored

Code

```
None
nc -vz <api-endpoint> 6443

curl -k https://<api-endpoint>:6443/healthz
```

4. Pod Not Found Causing Application Failure

Description:
An operation fails because a required Kubernetes Pod cannot be found. This usually happens

when the Pod has not been created yet, was deleted or restarted, or exists in a different namespace than expected.

Symptoms:

- Error indicating a specific Pod (for example, `vault-0`) was not found
- Dependent tasks or applications fail when trying to access the Pod
- `kubectl`/`oc` queries for the Pod return **NotFound**
- The workload remains unavailable or in a failed state

Resolution:

- Verify the Pod exists and is running in the correct namespace
- Check whether the controlling resource (Deployment/StatefulSet) is healthy
- Review recent restarts, deletions, or scaling events
- Ensure the application has finished initializing before retrying the operation
- Retry once the Pod is present and ready

Code

```
None
oc get pods -n <namespace>

oc describe pod <pod-name> -n <namespace>

oc get statefulset,deployment -n <namespace>
```

5. Helm Download Network Timeout

Description:
A Helm binary download fails because the connection to the download server times out. This is

usually caused by network connectivity issues, blocked outbound access (proxy/firewall), DNS problems, or a slow/unreliable mirror.

Symptoms:

- Download fails with a **timeout** while fetching Helm
- Logs mention inability to connect or stalled download progress
- The same URL may work intermittently or fail consistently from the environment
- Other outbound HTTP/HTTPS requests may also be slow or blocked

Resolution:

- Verify outbound internet access from the host/container (DNS + routing)
- Check proxy/firewall rules and configure `HTTP_PROXY/HTTPS_PROXY/NO_PROXY` if required
- Retry using an alternate mirror or internal artifact repository
- Increase download timeout/retry settings if supported
- Pre-download Helm and ship it with the image (best for air-gapped / CI stability)

Code

```shell
curl -I
http://mirror.openshift.com/pub/openshift-v4/clients/helm/3.6.2/h
elm-linux-amd64

export HTTPS_PROXY=http://<proxy>:<port>
```

6. VM Destroy Failed Due to Invalid VM State

Description:
A virtual machine destroy/delete operation fails because the VM is in a state that does not allow

deletion (for example, powering on/off in progress, suspended, locked by a task, or already being deleted). This is usually a platform-side state/lock issue rather than an Ansible task syntax problem.

Symptoms:

- VM delete/destroy task fails with **"invalid virtual machine state"**
- VM appears **busy/locked** or stuck in a transition state
- Similar operations (power off, snapshot, migrate) also fail
- Re-running immediately fails with the same state error

Resolution:

- Check the VM's current power/task state and wait for ongoing operations to complete
- Cancel/clear stuck tasks or locks if the platform supports it
- Power off the VM (if required) before attempting deletion
- Ensure no dependent operations exist (snapshots, backups, attached ISO, pending migrations)
- Retry the destroy operation after the VM reaches a stable state

7. Let's Encrypt Certificate Issuance Failure

Description:
A certificate request fails while using Let's Encrypt. This typically happens when ACME validation cannot complete due to DNS issues, unreachable endpoints, misconfigured challenge solvers, or rate limits.

Symptoms:

- Certificate workload reports a **failure** or remains pending
- Logs show ACME/Let's Encrypt challenge errors
- HTTPS endpoints remain untrusted or unavailable

- Repeated retries do not result in a valid certificate

Resolution:

- Verify DNS records point to the correct public endpoint
- Ensure the ACME challenge path (`/.well-known/acme-challenge`) is reachable
- Check Issuer/ClusterIssuer configuration and credentials
- Confirm outbound access to Let's Encrypt endpoints
- Retry after fixing DNS/networking or use the staging environment to test

Code

```shell
kubectl get certificate,certificaterequest,challenge

kubectl describe challenge <name>
```

8. AWX API Request Connection Failure

Description:
Ansible fails to reach the AWX/Tower API endpoint after multiple retries while monitoring a job that is still reported as running. This usually indicates a temporary connectivity issue to the AWX API (network/DNS/proxy), AWX service instability, or an incorrect API endpoint.

Symptoms:

- API requests to AWX fail repeatedly while the job status remains **running**
- Logs show connection errors, timeouts, or retries exhausted when calling the AWX API
- Manual access to the AWX API endpoint is slow or unreachable
- Job ID is known, but status cannot be refreshed from the API

Resolution:

- Verify AWX API endpoint URL, DNS resolution, and network connectivity
- Check AWX service health (controller pods/services/ingress) and API responsiveness
- Confirm authentication/token is valid and not expired (if errors indicate auth)
- Ensure proxies/firewalls allow outbound access to the AWX endpoint
- Increase retry/backoff intervals and retry once AWX/API connectivity is restored

Code

```shell
curl -k https://<awx-host>/api/v2/jobs/<job_id>/

kubectl get pods -n <awx-namespace>
```

9. List Index Error (No Element 0)

Description:
A task fails because it tries to access the first element of a list (index 0 or | first) but the list is empty or undefined in workload.yml.

Symptoms:

- Error message like **"list object has no element 0"**
- Failure points to a line using `[0]`, `| first`, or a loop result assumed to be non-empty
- The referenced list variable is empty due to filtering, missing input, or no matching resources

Resolution:

- Ensure the list variable is defined and contains at least one element
- Add safe fallbacks using `default([])` and `first | default(...)`

- Guard the task with a condition (`when`) or fail early with `assert` for clearer errors
- Verify filters/selectors producing the list actually match expected items

10. MultiClusterHub Deployments Have Unknown Status

Description:
Multiple deployments under MultiClusterHub report an unknown status because no status conditions are available even after repeated checks. This usually indicates delayed reconciliation, API/cluster instability, or controllers failing to update deployment status.

Symptoms:

- Deployment status remains **Unknown** after **x attempts**
- No `status.conditions` are present for affected deployments
- Deployments appear stuck (not progressing to Available/Degraded)
- Controller/operator logs show reconciliation delays or errors

Resolution:

- Verify MultiClusterHub and related operators/controllers are running and healthy
- Check affected deployments and events to see why conditions aren't being set
- Confirm the cluster API is responsive and not under heavy load
- Ensure required CRDs/resources are installed and not stuck in reconciliation
- Increase retry/timeout if this is expected during initial install, then re-check

11. Undefined Variable in Module Defaults

Description:
A playbook fails because a variable referenced in module_defaults is not defined. This typically happens when a required configuration value is missing from play, inventory, group/host vars, or external inputs.

Symptoms:

- Error indicating a variable is **undefined** in `module_defaults`
- Failure points to a specific YAML file and task (e.g., `delete_project.yaml`)
- Tasks using the affected module fail immediately
- Re-running without changes results in the same error

Resolution:

- Define the variable in an appropriate scope (play vars, group_vars, host_vars, inventory, or extra vars)
- Provide a safe fallback using Jinja `default()` if the value may be optional
- Validate required variables early using `assert` for clearer failures
- Review variable precedence to ensure the value is not overridden or missing

12. Workspace Operation Failed Despite HTTP 200

Description:
An API request succeeds at the transport level (HTTP 200 OK), but the returned payload indicates the workspace/job is in a failed state. This typically happens with asynchronous automation platforms where the API call returns successfully while the underlying workspace execution reports an error.

Symptoms:

- API response returns **status code 200**, but workspace status is **FAILED/ERROR**
- Logs show a mismatch like *"request succeeded but workspace indicates failure"*
- Subsequent polling continues to report a failed status
- Workspace run output contains an underlying provisioning or validation error

Resolution:

- Treat HTTP 200 as “request processed” and use the **workspace status field** as the real success signal
- Fetch detailed workspace/job logs to identify the first failing step
- Verify inputs/variables, credentials, permissions, quotas, and target resource settings
- Add logic to stop polling immediately on **FAILED/ERROR** and surface the failure reason
- Retry only after fixing the reported root cause in the workspace run logs

Code

```shell
# Example: fetch workspace details and logs (API/CLI depends on platform)

curl -s https://<api>/workspaces/<id>

curl -s https://<api>/workspaces/<id>/logs
```

13. Kubernetes API Connection Refused

Description:
Ansible fails to reach the Kubernetes API endpoint after multiple retries because the connection is refused. This usually means the API server is not reachable or not accepting connections (service down, cluster still starting, DNS/LB misrouting, or firewall rules blocking the API port).

Symptoms:

- Requests to the Kubernetes API fail with **“connection refused”**
- Retries are exhausted without a successful API response
- Kubernetes-related tasks fail immediately (listing resources, applying manifests, RBAC queries)
- Manual checks to the API endpoint on port **6443** also fail

Resolution:

- Verify the API endpoint resolves to the correct address (DNS / load balancer)
- Ensure network routes/firewalls/security groups allow access to port **6443**
- Confirm the API server is running and healthy (cluster not still bootstrapping)
- Check control-plane health and logs if you have node access
- Retry once API connectivity is restored

Code

```shell
nc -vz <api-endpoint> 6443

curl -k https://<api-endpoint>:6443/healthz
```

14. EC2 Instance Start Failed After Max Retries

Description:
An attempt to start an EC2 instance fails after repeated retries. This usually happens when AWS cannot transition the instance to running due to capacity limits, invalid instance settings, blocked dependencies (network/IAM), or transient AWS API issues.

Symptoms:

- Instance start operation retries and eventually fails with **max retries reached**
- Instance remains in **pending**, **stopping**, or returns to **stopped**
- AWS reports errors related to capacity, limits, or instance state transitions
- Subsequent start attempts fail in the same way

Resolution:

- Check the instance state and AWS event messages for the failure reason
- Verify region/AZ capacity and try a different instance type or availability zone

- Confirm quotas and limits (instances, vCPU, Elastic IPs, spot limits if relevant)
- Validate networking and dependencies (subnets, security groups, ENIs)
- Retry with increased backoff, or after capacity/limit issues are resolved

Code

```shell
aws ec2 describe-instances --instance-ids <instance-id>

aws ec2 describe-instance-status --instance-ids <instance-id>
--include-all-instances
```