Bonus Exercises

Ansible for Proxmox VE

credativ GmbH

Table of contents

For participants who finish the exercises early. Each one stands on its own and assumes the state after exercise 05: a quorate cluster, the storage `pve-backup`, the service account `automation@pve` with its token, and VM 9001 `web01` on `pve02`.

They are ordered roughly by effort, not by importance.

> **i** Documentation
>
> `community.proxmox` collection · `ansible.builtin.uri` · Conditionals · API browser · offline: `ansible-doc -l community.proxmox`

B1 — Make the backup idempotent

Objective: the backup task in `playbooks/30-operations.yml` reports `changed` on every run, because a backup is an action rather than a state. Guard it so that a second run within the same day does nothing.

Read the existing backups with `community.proxmox.proxmox_backup_info` and skip the backup task when one already exists for this VMID.

Check: run the playbook twice. The second run reports `changed=0` for the whole play.

B2 — Scope the token to a pool

Objective: `automation@pve` currently holds `PVEVMAdmin` on `/`, which is every guest in the cluster. Narrow it to one resource pool, and make the token itself the subject of the permission.

1. Create a pool and put VM 9001 in it (`proxmox_pool`, `proxmox_pool_member`).
2. Give the **token** `PVEVMAdmin` on `/pool/<id>` (`proxmox_access_acl` with `type: token`).
3. Switch the token to `privsep: true` and remove the datacenter-wide grant.

Check: the token can still stop and start 9001, and can no longer create a guest outside the pool.

B3 — A scheduled backup job without a module

Objective: no module creates a scheduled backup job. Build one with `ansible.builtin.uri` against `/cluster/backup`, and give it the idempotency the module would have provided.

Query the existing jobs first, and create yours only when no job with your comment exists.

Check: run it twice; the second run creates nothing. The job appears under *Datacenter → Backup*.

B4 — Template and clone

Objective: turn `web01` into a template and deploy two guests from it.

`proxmox_kvm` converts an existing VM with `template: true`, and clones with `clone:` plus `newid:`. A template cannot be started, and it cannot be converted back.

Check: `qm list` on `pve02` shows the template and two clones. Running the playbook again creates nothing.

B5 — Let tags drive a play

Objective: close the loop between the dynamic inventory and a playbook. Tag a guest in the GUI, and have a play act on the group that the tag produces.

Write a play against the group `webservers` from exercise 06 that prints the VMID and node of every member. Then add the tag to a second guest and re-run without touching any file.

Check: the second guest appears in the output, and you changed nothing but a tag.

B6 — A health check playbook

Objective: write `playbooks/90-check.yml` that only reads, changes nothing, and fails loudly when the cluster is not healthy.

Assert that the cluster is quorate, that every node in the inventory is a member and online, and that `pve-backup` is active. Use the `*_info` modules, not `command`.

Check: the playbook reports `changed=0` and passes. Stop `corosync` on one node and it fails with a message that says what is wrong.

Solutions

B1 — idempotent backup

```yaml
- name: Read existing backups
  community.proxmox.proxmox_backup_info:
    vmid: "{{ pve_vm_id }}"
  register: existing

- name: Back up the VM
  community.proxmox.proxmox_backup:
    mode: include
    vmids: ["{{ pve_vm_id }}"]
    storage: "{{ pve_backup_storage }}"
    compress: zstd
    wait: true
    wait_timeout: 600
  when: existing.backups | default([]) | length == 0
```

The guard above is deliberately crude: any backup at all is enough to skip. A more honest version compares the timestamp, for example only backing up when the newest archive is older than a day. Both are worth discussing, because the question "what counts as already done" is the one that decides whether an action can be made idempotent at all.

Point out that this is the same problem as `creates:` on a `command` task, one layer up: the module cannot know, so you tell it.

B2 — token scoped to a pool

```yaml
- name: Create the pool
  community.proxmox.proxmox_pool:
    poolid: web
    comment: Web tier

- name: Put the VM in the pool
  community.proxmox.proxmox_pool_member:
    poolid: web
    member: "{{ pve_vm_id }}"
    type: vm

- name: Grant the token rights on the pool only
  community.proxmox.proxmox_access_acl:
    path: /pool/web
    type: token
    ugid: "automation@pve!ansible"
    roleid: PVEVMAdmin
    propagate: 1

- name: Remove the datacenter-wide grant from the group
```

```
  community.proxmox.proxmox_access_acl:
    path: /
    type: group
    ugid: automation
    roleid: PVEVMAdmin
    state: absent
```

Then set `privsep: true` on the token in the `proxmox_user` task and recreate it.

The teaching point is the intersection rule from module 05: with privilege separation the token holds the *intersection* of the user's rights and its own ACL. The user must therefore still be allowed on the pool, otherwise the intersection is empty and the token can do nothing, no matter what you granted it.

Expect a participant to grant the token `Administrator` on `/` and be surprised that it changes nothing. That is the rule working as designed.

B3 — backup job with `uri`

```
- name: Read the existing backup jobs
  ansible.builtin.uri:
    url: "https://{{ pve_api_host }}:8006/api2/json/cluster/backup"
    method: GET
    validate_certs: false
    headers:
      Authorization: "PVEAPIToken={{ pve_api_token }}"
  register: jobs

- name: Create the nightly job
  ansible.builtin.uri:
    url: "https://{{ pve_api_host }}:8006/api2/json/cluster/backup"
    method: POST
    validate_certs: false
    headers:
      Authorization: "PVEAPIToken={{ pve_api_token }}"
    body_format: form-urlencoded
    body:
      schedule: "02:30"
      storage: "{{ pve_backup_storage }}"
      mode: snapshot
      all: 1
      comment: managed-by-ansible
    status_code: [200]
  when: jobs.json.data
        | selectattr('comment', 'defined')
        | selectattr('comment', 'equalto', 'managed-by-ansible')
        | list | count == 0
```

Two things to draw out. First, the guard is the whole exercise: `uri` has no idea what it is doing, so idempotency is something you build, exactly as with `command`. Second, the comment is being used as an identifier. That is a common pattern with APIs that hand out generated ids, and it is fragile in a way worth naming: anyone who edits the comment in the GUI creates a duplicate job on the next run.

B4 — template and clone

```
- name: Convert the VM into a template
  community.proxmox.proxmox_kvm:
    node: "{{ pve_vm_node }}"
    vmid: "{{ pve_vm_id }}"
    template: true

- name: Clone two guests from it
  community.proxmox.proxmox_kvm:
    node: "{{ pve_vm_node }}"
    clone: "{{ pve_vm_name }}"
    vmid: "{{ pve_vm_id }}"
    newid: "{{ item }}"
    name: "web{{ item }}"
    storage: "{{ pve_vm_storage }}"
    full: true
    timeout: 300
  loop: [9101, 9102]
```

Note what this costs: the template can no longer be started, and there is no way back. Participants who want `web01` as a running guest afterwards have to clone it and delete the template. Say so before they run it, or expect the question.

Clone is idempotent through `newid`: the second run finds 9101 and 9102 and reports `ok`.

B5 — tags driving a play

```
- name: Report the tagged guests
  hosts: webservers
  gather_facts: false
  tasks:
    - name: Show what the inventory knows
      ansible.builtin.debug:
            msg: "{{ inventory_hostname }} = vmid {{ proxmox_vmid }} on
{{ proxmox_node }}"
```

```
export PVE_TOKEN_SECRET='...'
ansible-playbook -i inventory/lab.proxmox.yml playbooks/report.yml
```

The point is operational, not technical: the person who creates the guest decides what it is, with a tag, in the interface they already use. No file changes hands. This is the argument that sells dynamic inventories to an operations team.

Needs `want_facts: true` in the inventory configuration, otherwise `proxmox_vmid` and `proxmox_node` are not defined.

B6 — health check

```
---
- name: Check the cluster
  hosts: localhost
  connection: local
  gather_facts: false
  module_defaults:
    group/community.proxmox.proxmox:
      api_host: "{{ hostvars['pve01'].ansible_host }}"
      api_user: root@pam
      api_password: "{{ hostvars['pve01'].pve_root_password }}"
      validate_certs: false

  tasks:
    - name: Read the cluster status
      community.proxmox.proxmox_cluster_status_info:
      register: cl
      changed_when: false

    - name: The cluster is quorate
      ansible.builtin.assert:
        that: cl.cluster_status | selectattr('type', 'equalto', 'cluster')
              | selectattr('quorate', 'equalto', true) | list | count == 1
        fail_msg: "cluster is not quorate"

    - name: Every inventory node is a member and online
      vars:
       members: "{{ cl.cluster_status | selectattr('type', 'equalto', 'node')
| list }}"
      ansible.builtin.assert:
        that:
          - members | map(attribute='name') | sort | list == groups['pve']
| sort
          - members | selectattr('online', 'equalto', 1) | list | count ==
groups['pve'] | count
            fail_msg: "expected {{ groups['pve'] }}, found {{ members |
map(attribute='name') | list }}"

    - name: Read the storage configuration
      community.proxmox.proxmox_storage_info:
```

```
      register: st
      changed_when: false

    - name: The backup storage exists
      ansible.builtin.assert:
        that: st.proxmox_storages | default([])
              | selectattr('storage', 'equalto', pve_backup_storage)
              | list | count == 1
        fail_msg: "storage {{ pve_backup_storage }} is missing"
```

This is the shape of the verification suite in `code/tests/`, which is worth showing afterwards: the same assertions, one directory up. A playbook that only reads is the cheapest monitoring you will ever write, and it runs in CI.