

Forming the Cluster — Participant Handout

Ansible for Proxmox VE · Module 03

credativ GmbH

Table of contents

1	What a Proxmox VE cluster is	1
2	The collection	1
3	Where API modules run	2
4	Handling credentials: lab vs. production	2
5	The playbook	3
5.1	Play 1: create the cluster	3
5.2	Play 2: join the remaining nodes	3
5.3	Play 2, continued: wait for the node to actually arrive	4
6	The same thing with curl	5
6.1	Creating the cluster	6
6.2	Reading the join information	6
6.3	Joining a node	6
6.4	Checking the status	6
7	The CLI fallback	7
8	Verifying the result	7
9	Quorum, in one table	8
10	Further reading	8

1 What a Proxmox VE cluster is

Three things happen when nodes form a cluster:

- **corosync** provides cluster communication and decides who is in the membership.
- **pmxcfs**, the Proxmox cluster file system, mounts `/etc/pve` and replicates every configuration file to all members. Editing a guest configuration on one node makes it visible on all of them.
- **Quorum**, a majority of votes, is required before a node may change anything. With three nodes the quorum is two.

None of this is optional knowledge for the automation: quorum is the reason we join nodes sequentially, and pmxcfs is the reason a single API call changes the state of the whole cluster.

2 The collection

The Proxmox modules live in **community.proxmox**. They previously lived in **community.general** and were removed there in version 11, which is why many examples found online still use the old names.

```
1 ---
2 collections:
3   - name: community.proxmox
4     version: ">=2.0.0"
```

```
1 ansible-galaxy collection install -r requirements.yml
2 ansible-doc -l community.proxmox | head -20 # what you just installed
```

Requirements worth checking before the first run:

Requirement	Where	Check
ansible-core $\geq$ 2.17	control node	<code>ansible --version</code>
proxmoxer $\geq$ 2.3, requests	control node	<code>python3 -c 'import proxmoxer'</code>
API reachable on port 8006	from control node	<code>curl -k https://192.168.0.1:8006/</code>

3 Where API modules run

An SSH module runs *on* the managed node. An API module does not need to: it opens an HTTPS connection to port 8006 and talks to the Proxmox VE API. It therefore runs wherever proxmoxer is installed, which here is the control node.

```
1 - name: Ensure the cluster exists
2   community.proxmox.proxmox_cluster:
3     api_host: "{{ ansible_host }}"
4     ...
5   delegate_to: localhost
```

The play still iterates over the Proxmox hosts, so `ansible_host`, `hostvars` and `per-host` variables all behave as usual. Only the execution location changes to the control node.

i Note

`validate_certs: false` accepts the self-signed certificate that every fresh Proxmox VE node ships with. In production you install a certificate from your own CA (or use the ACME integration) and leave validation switched on.

4 Handling credentials: lab vs. production

Credentials never belong directly in task definitions; they belong in variables. In `inventory/group_vars/pve.yml`:

```
1 pve_cluster_name: training
2 pve_root_password: "password123"
```

i Note

In production, credentials belong in Ansible Vault or your CI/CD secret store. In this lab we keep them plain to avoid typing overhead.

5 The playbook

5.1 Play 1: create the cluster

```

1  ---
2  - name: Create the cluster on the primary node
3    hosts: pve_primary
4    gather_facts: false
5    tasks:
6      - name: Ensure the cluster exists
7        community.proxmox.proxmox_cluster:
8          state: present
9          api_host: "{{ ansible_host }}"
10         api_user: root@pam
11         api_password: "{{ pve_root_password }}"
12         validate_certs: false
13         cluster_name: "{{ pve_cluster_name }}"
14         link0: "{{ ansible_host }}"
15         delegate_to: localhost

```

link0 is the address corosync uses for its first ring. Setting it explicitly means the cluster communicates over the network you chose, not over whichever address the node happened to resolve first. A second ring for redundancy would be link1.

The module is idempotent: it reads the cluster status first, reports ok if a cluster with that name exists, and fails clearly if the node belongs to a *different* cluster.

5.2 Play 2: join the remaining nodes

```

1  - name: Join the remaining nodes
2    hosts: pve_secondary
3    gather_facts: false
4    serial: 1
5    vars:
6      primary: "{{ groups['pve_primary'][0] }}"
7    tasks:
8      - name: Read the join information from the primary
9        community.proxmox.proxmox_cluster_join_info:
10         api_host: "{{ hostvars[primary].ansible_host }}"
11         api_user: root@pam
12         api_password: "{{ pve_root_password }}"

```

```

13     validate_certs: false
14     delegate_to: localhost
15     register: join_info
16
17     - name: Join this node to the cluster
18       community.proxmox.proxmox_cluster:
19         state: present
20         api_host: "{{ ansible_host }}"
21         api_user: root@pam
22         api_password: "{{ pve_root_password }}"
23         validate_certs: false
24         master_ip: "{{ hostvars[primary].ansible_host }}"
25         fingerprint: "{{ (join_info.cluster_join.nodelist
26                           | selectattr('name', 'equalto', primary)
27                           | first).pve_fp }}"
28         link0: "{{ ansible_host }}"
29         delegate_to: localhost

```

Three details deserve attention:

serial: 1. Without it, Ansible would run the play for pve02 and pve03 in parallel and both would try to join at the same moment. corosync rewrites its configuration for every new member, and the joining node restarts its own services mid-way. Sequential joins are not a style preference; they are the supported path.

The fingerprint. `proxmox_cluster_join_info` returns the current cluster configuration, including a `nodelist` with one entry per member. We filter that list for the primary node by name and take its `pve_fp` value. This is why your inventory host names must match the real Proxmox VE node names.

master_ip. The address of a node that is already in the cluster. The module also uses it to recognise that a node is already a member, in which case it reports ok and does nothing.

5.3 Play 2, continued: wait for the node to actually arrive

The API accepts the join request and returns immediately; the work happens in the background, and the joining node's web service restarts in the process.

```

1     - name: Wait until this node is a quorate cluster member
2       community.proxmox.proxmox_cluster_status_info:
3         api_host: "{{ ansible_host }}"
4         api_user: root@pam
5         api_password: "{{ pve_root_password }}"
6         validate_certs: false
7         delegate_to: localhost
8         register: cl
9         until: cl.cluster_status | default([])
10              | selectattr('type', 'equalto', 'cluster')

```

```

11         | selectattr('quorate', 'equalto', true) | list | count > 0
12     retries: 30
13     delay: 5

```

Two details in that condition are not decoration:

quorate, not just “a cluster exists”. POST /cluster/config returns before corosync has reached a quorum. A node that joins in that window is rejected with cluster not ready - no quorum?, and the join task reports changed anyway because the API accepted the request. Waiting for port 8006 does not help either, because pveproxy never went away.

default([]). The joining node restarts its own web service during the join, so the status call fails for a few seconds. Without the default, the until expression raises on an undefined key and the task aborts instead of retrying.

Skipping this is the classic way to build a playbook that works when you run it by hand and fails in a pipeline: the next play starts before the previous change has landed.

6 The same thing with curl

Every module in this collection is a wrapper around one or two REST calls. Seeing the call underneath tells you what the module can possibly do, makes error messages readable, and gives you a fallback when a module lacks an option.

Set up the credentials once. An API token goes in as an HTTP header, with no login round-trip:

```

1  TOKEN='PVEAPIToken=root@pam!ansible=xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxxxx'
2  PVE=https://192.168.0.1:8006/api2/json
3  curl -sk -H "Authorization: $TOKEN" "$PVE/version" | jq

```

-k accepts the self-signed lab certificate, jq formats the JSON envelope, which always wraps the payload in a data key.

i Where that token comes from

Nothing in this workshop creates a token for root@pam, so make one on a node before you try these calls:

```
pveum user token add root@pam ansible --privsep 0
```

The secret is printed once. If you use the automation@pve token from module 05 instead, the cluster-wide endpoints come back **empty rather than failing**: that account holds PVEVMAdmin, which does not include Sys.Audit. An empty result from the Proxmox VE API usually means missing privileges, not missing data.

Ansible module	HTTP call	API viewer
proxmox_cluster (create)	POST /cluster/config	/cluster/config

Ansible module	HTTP call	API viewer
proxmox_cluster (join)	POST /cluster/config/ join	/cluster/config/join
proxmox_cluster_join_info	GET /cluster/config/join	same page
proxmox_cluster_status_info	GET /cluster/status	/cluster/status

6.1 Creating the cluster

```
1 # Ansible: community.proxmox.proxmox_cluster (state: present,
  cluster_name, link0)
2 curl -sk -H "Authorization: $TOKEN" \
3     -X POST "$PVE/cluster/config" \
4     -d clustername=training \
5     -d link0=192.168.0.1 | jq
```

6.2 Reading the join information

```
1 # Ansible: community.proxmox.proxmox_cluster_join_info
2 curl -sk -H "Authorization: $TOKEN" "$PVE/cluster/config/join" \
3     | jq '.data.nodelist[] | {name, pve_addr, pve_fp}'
```

That pve_fp is exactly the value the playbook extracts with selectattr.

6.3 Joining a node

Note the host: this call goes to the **joining** node, not to the primary.

```
1 # Ansible: community.proxmox.proxmox_cluster (master_ip + fingerprint)
2 curl -sk -H "Authorization: $TOKEN2" \
3     -X POST "https://192.168.0.2:8006/api2/json/cluster/config/join" \
4     -d hostname=192.168.0.1 \
5     -d fingerprint="08:B5:B2:F9:..." \
6     -d password="$PVE_ROOT_PASSWORD" \
7     -d link0=192.168.0.2
```

The response is a **UPID**, a task id rather than a result. The join runs in the background, which is precisely why the playbook waits afterwards. You can follow the task:

```
curl -sk -H "Authorization: $TOKEN2" "$PVE2/nodes/pve02/tasks/$UPID/status"
| jq
```

6.4 Checking the status

```
1 # Ansible: community.proxmox.proxmox_cluster_status_info
2 curl -sk -H "Authorization: $TOKEN" "$PVE/cluster/status" \
3     | jq '.data[] | {type, name, online, quorate}'
```

i Note

The complete API of **your** cluster is browsable on every node at `https://<node>:8006/pve-docs/api-viewer/`, including the exact parameters of your version. The public copy is at <https://pve.proxmox.com/pve-docs/api-viewer/>.

7 The CLI fallback

If the collection is unavailable (an air-gapped environment, an older control node), the same result is reachable over SSH. You then own the idempotency:

```
1  - name: Create the cluster
2    ansible.builtin.command:
3      cmd: "pvecm create {{ pve_cluster_name }} --link0 {{ ansible_host }}"
4    args:
5      creates: /etc/pve/corosync.conf          # the guard
6    when: inventory_hostname in groups['pve_primary']
```

Joining over the CLI additionally needs passwordless root SSH from the joining node to the primary, because `pvecm add --use_ssh` authenticates that way. That is extra moving parts for the same outcome, so prefer the module when you can.

8 Verifying the result

```
1  ansible pve01 -m ansible.builtin.command -a 'pvecm status'
2  ansible pve01 -m ansible.builtin.command -a 'pvecm nodes'
3  ansible pve   -m ansible.builtin.command -a 'ls /etc/pve/nodes'
```

`pvecm status` should show Quorum information with Quorate: Yes, three votes, and the expected cluster name. `ls /etc/pve/nodes` returns the same three directories on every node, which is `pmxcfs` replication at work.

Then run the playbook again. Everything reports ok. The cluster has stopped being something that happened and has become something you describe.

i What the join does to your SSH keys

`/root/.ssh/authorized_keys` on a Proxmox VE node is a symlink into `pmxcfs`:

```
/root/.ssh/authorized_keys -> /etc/pve/priv/authorized_keys
```

So it is not a per-node file, it is cluster-wide. A node that joins gets the primary's `/etc/pve`, and with it the primary's copy of that file. Keys that existed only on the joining node are gone afterwards.

`deploy.sh` installs your key on all three nodes before you form anything, so the primary's copy already contains it and the join changes nothing for you. Worth knowing before you distribute keys to a cluster you are about to build: put them on the primary, or put them everywhere.

9 Quorum, in one table

Nodes online (of 3)	Quorate	What still works
3	yes	everything
2	yes	everything, without further redundancy
1	no	<code>/etc/pve</code> read-only, no guest starts, no configuration changes

For two-node clusters a **QDevice** provides the third vote. That is a topic of the *Clustering & Shared Storage* course.

! Important

The Proxmox VE API can create and join clusters, but it cannot make a node *leave* one. Removing a node is a deliberate, partly manual procedure (`pvecm delnode`, followed by reinstalling the removed node before any reuse). Plan cluster membership before you automate it.

10 Further reading

The collection

- `community.proxmox` [collection index](#) (all modules, the inventory plugin, and the authentication guide)
- `proxmox_cluster`
- `proxmox_cluster_join_info`
- `proxmox_cluster_status_info`
- [Source and issue tracker on GitHub](#) (when a module does not do what you need, the source is short and readable, and the issue tracker usually already has your question)

- [Installing collections](#)

Ansible concepts from this module

- [Controlling where tasks run: delegation](#) (`delegate_to`, `run_once`)
- [Controlling playbook execution: strategies and more](#) (`serial`, and why it exists)
- [Retrying a task until a condition is met](#) (`until`, `retries`, `delay`)
- `ansible.builtin.wait_for`
- [Jinja filters](#): `selectattr`, `map`, `first`

Proxmox VE

- [Cluster Manager \(pvecm\)](#): read the sections on requirements, adding nodes, and separating the cluster network
- [Cluster Manager wiki page](#)
- [API viewer](#): look up `/cluster/config` and `/cluster/config/join`, which is exactly what the module calls
- `man pvecm` on any node

Offline

```
ansible-doc community.proxmox.proxmox_cluster
ansible-doc -l community.proxmox          # everything the collection offers
```