

Preparing the Nodes — Participant Handout

Ansible for Proxmox VE · Module 02

credativ GmbH

Table of contents

1 What a cluster needs before it exists	1
2 Playbook structure	1
3 Variables and facts	2
3.1 Your own variables	2
3.2 { }: Jinja, and where it runs	2
3.3 loop: and item	3
3.4 Facts, gathered from the machine	3
4 Job 1: repositories	3
5 Handlers	4
5.1 Is there an API equivalent?	5
6 Job 2: name resolution, verified	5
7 Jobs 3 and 4: packages and time	6
8 Running it	7
8.1 Flags you will use every day	7
9 Further reading	8

1 What a cluster needs before it exists

Forming a Proxmox VE cluster fails in predictable ways when the groundwork is missing. This module removes all four causes:

Requirement	Consequence if missing
Usable package sources	apt fails; you cannot install or update anything
Mutual name resolution	pvecm and corosync cannot address the other nodes
Synchronised clocks	corosync membership becomes unstable
Identical base state	problems appear on one node only and are hard to find

Each requirement becomes one group of tasks in a single playbook.

2 Playbook structure

A playbook is a list of **plays**. A play binds a **host pattern** to a list of **tasks**.

```

1  ---
2  - name: Prepare the Proxmox VE nodes
3    hosts: pve
4    gather_facts: true
5    tasks:
6      - name: Install the admin tools we want everywhere
7        ansible.builtin.apt:
8          name: "{{ pve_packages }}"
9          state: present

```

Conventions worth adopting from the start:

- Give **every** task a name:. It is what appears in the output, and it is what --start-at-task and reviewers rely on.
- Use the **fully qualified** module name (ansible.builtin.apt, not apt). It removes any doubt about which collection a module comes from.
- Keep the playbook free of values. Values belong in variables.

3 Variables and facts

3.1 Your own variables

```

1  pve_packages:
2    - vim
3    - curl
4    - jq
5    - tmux

```

Files under inventory/group_vars/ are loaded automatically for the group named by the file. inventory/group_vars/all.yml applies to every host.

3.2 { }: Jinja, and where it runs

Everything between {{ and }} is a **Jinja** expression. Ansible evaluates it on the **control node**, before the task is sent anywhere — which is why a fact gathered from the node can end up in a task argument in the first place.

```

name: "{{ pve_packages }}" # a variable
suites: "{{ ansible_distribution_release }}" # a fact
loop: "{{ groups['pve'] | difference([inventory_hostname]) }}" # a filter

```

| pipes a value through a **filter**. difference here returns the group minus this host — the list of peers each node has to resolve. There are several hundred filters; the two pages worth bookmarking are **templating** and the **filter list**.

Quote the value whenever it begins with {{. Unquoted, YAML reads the opening brace as the start of a dictionary and the playbook fails to parse.

3.3 loop: and item

```
- name: Disable the enterprise repositories
  ansible.builtin.file:
    path: "{{ item }}"
    state: absent
  loop:
    - /etc/apt/sources.list.d/pve-enterprise.sources
    - /etc/apt/sources.list.d/ceph.sources
```

The task runs once per entry, and the current entry is always called `item`. When the entries are long or are dictionaries, `loop_control: {label: "..."}` decides what the output shows instead of the whole value — the cluster playbook in module 03 uses that.

Older material writes `with_items:`. It still works and means the same thing; `loop:` is what current documentation uses.

3.4 Facts, gathered from the machine

With `gather_facts: true`, Ansible collects a few hundred variables per host before the first task runs:

```
ansible_distribution      = "Debian"
ansible_distribution_release = "trixie"          # Debian 13 = Proxmox VE 9
ansible_default_ipv4.address = "192.168.0.1"
ansible_hostname          = "pve01"
ansible_memtotal_mb        = 16384
```

Your lab nodes run **Proxmox VE 9**, based on Debian 13 (“trixie”). Using `{ ansible_distribution_release }` instead of hardcoding `trixie` keeps the playbook working when the next Debian release arrives. Inspect everything a node reports with:

```
ansible pve01 -m ansible.builtin.setup
```

4 Job 1: repositories

A fresh Proxmox VE installation has the **enterprise** repository enabled, which requires a subscription. Without one, every `apt update` ends in an error. In the lab we disable it and enable the no-subscription repository instead.

```
1  - name: Disable the enterprise repositories
2    ansible.builtin.file:
3      path: "{{ item }}"
4      state: absent
5    loop:
6      - /etc/apt/sources.list.d/pve-enterprise.sources
7      - /etc/apt/sources.list.d/ceph.sources
8    notify: Update apt cache
```

```

9
10 - name: Enable the no-subscription repository
11   ansible.builtin.deb822_repository:
12     name: pve-no-subscription
13     types: [deb]
14     uris: http://download.proxmox.com/debian/pve
15     suites: "{{ ansible_distribution_release }}"
16     components: [pve-no-subscription]
17     signed_by: /usr/share/keyrings/proxmox-archive-keyring.gpg
18     notify: Update apt cache

```

deb822_repository writes the deb822 .sources format that Debian 13 and Proxmox VE 9 use natively.

Note

The no-subscription repository is intended for testing and evaluation. Production systems belong on the enterprise repository with a valid subscription. We use it here because the lab has no subscription, and because it is a realistic first automation task.

Verify the keyring path

The file name of the Proxmox signing key has changed between releases. Check what your nodes actually have before you run the playbook:

```
ansible pve01 -m ansible.builtin.command -a 'ls /usr/share/keyrings/'
```

Put the correct path into group_vars instead of repeating it in the playbook.

5 Handlers

A handler is a task that runs **only when notified**, and **only once** per play no matter how many tasks notified it.

```

1   handlers:
2     - name: Update apt cache
3       ansible.builtin.apt:
4         update_cache: true

```

By default handlers run at the *end* of the play. Here we need the refreshed cache *before* installing packages, so we flush them explicitly:

```

1     - name: Apply pending handlers now
2       ansible.builtin.meta: flush_handlers

```

This is the standard pattern whenever a later task depends on a handler's effect.

5.1 Is there an API equivalent?

For this module: mostly no, and that is the point. Package sources, installed packages and /etc/hosts are **Debian** concerns, and the Proxmox VE API does not model them. The one exception is the subscription status, which is why the enterprise repository exists at all:

```
1  TOKEN='PVEAPIToken=root@pam!ansible=xxxxxxxx-xxxx-xxxx-xxxx-
  xxxxxxxxxxxx' # pveum user token add root@pam ansible --privsep 0
2  PVE=https://192.168.0.1:8006/api2/json
3
4  # what the node thinks about its subscription
5  curl -sk -H "Authorization: $TOKEN" "$PVE/nodes/pve01/subscription" | jq
6
7  # available updates, as the GUI shows them
8  curl -sk -H "Authorization: $TOKEN" "$PVE/nodes/pve01/apt/update" | jq
  '.data[] | {Package, Version}'
9
10 # the repository status the GUI warns about
11 curl -sk -H "Authorization: $TOKEN" "$PVE/nodes/pve01/apt/repositories"
  | jq '.data.files[].path'
```

API reference: /nodes/{node}/apt and /nodes/{node}/subscription.

Useful for *reading* state; the actual base configuration happens in files on the node. From module 03 onwards, cluster and workload management moves to the API side.

6 Job 2: name resolution, verified

Every cluster node has to resolve the others, but that is not the playbook's job here: DNS and /etc/hosts come from the provisioning, including the real domain of your lab. What the playbook does is confirm the assumption before the cluster depends on it.

```
1  - name: Verify that every node resolves the other cluster nodes
2    ansible.builtin.command: "getent hosts {{ item }}"
3    register: pve_resolve
4    changed_when: false
5    failed_when: pve_resolve.rc != 0
6    loop: "{{ groups['pve'] | difference([inventory_hostname]) }}"
7    loop_control: {label: "{{ item }}"}
```

i Why not `ansible.utils.resolvable`?

There is a Jinja test for this, and it reads better:

```
that: "{{ item is ansible.utils.resolvable }}"
```

It is the wrong tool **here**, for the reason the Jinja section above gives: a test is evaluated by the templating engine, on the control node. That would check whether your desktop resolves the node names, which proves nothing about the nodes — and it is `corosync` on the node that needs to resolve its peers.

`getent hosts` runs as a task, on the node, which is where the question is. The test is a good fit for a pre-flight check *about the control node*, and it needs the `ansible.utils` collection, which this lab's offline bundle does not carry.

Three ideas here return later:

- **groups['pve']** is the inventory as data. The playbook does not contain a list of nodes; it asks the inventory.
- **difference([inventory_hostname])** removes the node itself from the loop, so each node only checks its peers.
- **changed_when: false** marks the task as read-only, which keeps a clean run at `changed=0`.

Add a fourth node to `inventory/hosts.yml` and the check covers it automatically.

⚠ Do not let Ansible own `/etc/hosts` here

It is tempting to generate that file from the inventory, and in a platform you build from scratch it is a reasonable thing to do. In this lab it is not: the provisioning already writes it, together with the node's real fully qualified domain name. Proxmox VE resolves its **own** node name through it, so a template that replaces the FQDN with an invented domain changes the node's identity, quietly and on all three nodes at once.

The rule generalises beyond this workshop: before you automate a file, find out whether something else already owns it.

7 Jobs 3 and 4: packages and time

```
1  - name: Install the admin tools
2    ansible.builtin.apt:
3      name: "{{ pve_packages }}"
4      state: present
5
6  - name: Check whether the clock is synchronised
7    ansible.builtin.command: timedatectl show -p NTPSynchronized --value
8    register: ntp_state
9    changed_when: false
```

```

10     check_mode: false
11
12 - name: Fail early if time is not in sync
13   ansible.builtin.assert:
14     that: ntp_state.stdout == 'yes'
15     fail_msg: "Time is not synchronised - corosync will be unhappy"

```

changed_when: false tells Ansible that this command only reads. Without it, every run would report a change and your changed=0 proof would be gone.

assert turns an assumption into a check. A playbook that fails here, loudly and early, is much friendlier than a cluster that misbehaves an hour later.

8 Running it

```

1  ansible-playbook playbooks/10-node-prep.yml --check --diff # dry run
2  ansible-playbook playbooks/10-node-prep.yml                # for real
3  ansible-playbook playbooks/10-node-prep.yml                # again

```

The third run should end in:

```

PLAY RECAP
pve01 : ok=7  changed=0  unreachable=0  failed=0  skipped=0
pve02 : ok=7  changed=0  unreachable=0  failed=0  skipped=0
pve03 : ok=7  changed=0  unreachable=0  failed=0  skipped=0

```

changed=0 is the evidence that your description matches reality. From here on, the playbook is documentation you can execute.

8.1 Flags you will use every day

Flag	Effect
--check	dry run: report what would change, change nothing
--diff	show the actual difference for file-based modules
--limit pve02	restrict the run to one host or pattern
--start-at-task "..."	resume at a named task
-v, -vvv	more detail; -vvv includes the SSH conversation

i Note

--check cannot predict everything. A task whose input depends on an earlier change may report incorrectly, and command/shell tasks are skipped entirely unless they declare check_mode: false. It still catches the majority of mistakes before they reach all three nodes at once.

9 Further reading

Playbook concepts from this module

- [Intro to playbooks](#)
- [Handlers: running operations on change](#) (including `flush_handlers`)
- [Using variables](#) and [Discovering variables: facts](#)
- [Loops](#)
- [Conditionals](#) (`when:`, `changed_when:`, `failed_when:`)
- [Validating tasks: check mode and diff mode](#)

The modules used here

- `ansible.builtin.apt`
- `ansible.builtin.deb822_repository`
- `ansible.builtin.command`
- `ansible.builtin.assert`
- `ansible.builtin.file`

Proxmox VE

- [Package Repositories](#), the wiki page that tells you which repository belongs on which release
- [Proxmox VE Administration Guide](#), the complete documentation, also installed on every node under `https://<node>:8006/pve-docs/`

Finding things offline

```
ansible-doc ansible.builtin.deb822_repository
ansible-doc -s ansible.builtin.deb822_repository # task skeleton with all
options
```