

Ansible in 30 Minutes — Participant Handout

Ansible for Proxmox VE · Module 01

credativ GmbH

Table of contents

1	Architecture & Execution Model	1
1.1	Proxmox REST API	2
2	Inventory, Modules, and Tasks	2
2.1	Inventory	2
2.2	Configuration	3
3	Ad-hoc commands	3
3.1	Reading the result	4
4	Idempotency	5
4.1	The exception: command and shell	5
5	Command reference	6
6	Documentation links	6

i Workshop scope

Starting point: three freshly installed Proxmox VE nodes, pve01 to pve03.

Automated during the workshop, in this order: package repositories, packages, /etc/hosts and time sync; cluster formation; storage; service accounts and API tokens; a VM and a backup; and finally the inventory itself.

1 Architecture & Execution Model

Ansible is agentless. It is installed on the *control node* (desktop). From there, it manages nodes via two mechanisms:

Path	Purpose	Execution location
SSH	Debian host-level configuration: packages, repositories, /etc/hosts, time sync	on pve01–pve03
HTTPS API (:8006)	Proxmox cluster state: cluster creation, storage, users/tokens, VMs, backups	on the control node (delegate_to: localhost), talking to the PVE API

Module 02 uses SSH, module 05 uses the API, and module 03 uses both.

1.1 Proxmox REST API

The Proxmox modules in the `community.proxmox` collection interact directly with the Proxmox VE REST API:

```
1 PVE=https://192.168.0.1:8006/api2/json
2
3 # the API has no HTTP basic auth: exchange the password for a ticket first
4 TICKET=$(curl -sk -d 'username=root@pam&password=THE-PASSWORD' \
5             "$PVE/access/ticket" | jq -r .data.ticket)
6
7 curl -sk -H "Cookie: PVEAuthCookie=$TICKET" "$PVE/version" | jq -c .data
8 # {"version":"9.2.2","release":"9.2","repoid":"b9984c6d90a4bd80"}
```

i Note

`curl -u user:password` does **not** work against this API, it answers 401. Either exchange the password for a ticket as above, or use an API token, which is what the later modules do.

The full API schema of your node is available at <https://192.168.0.1:8006/pve-docs/api-viewer/> (public copy at <https://pve.proxmox.com/pve-docs/api-viewer/>).

2 Inventory, Modules, and Tasks

- **Inventory:** defines target hosts and groups (`inventory/hosts.yml`).
- **Modules:** execution units (`ansible.builtin apt`, `community.proxmox.proxmox_kvm`, ...).
- **Tasks and playbooks:** maps modules and arguments to target host groups.

2.1 Inventory

```
1 all:
2   children:
3     pve:
4       hosts:
5         pve01:
6           ansible_host: 192.168.0.1
7         pve02:
8           ansible_host: 192.168.0.2
9         pve03:
10          ansible_host: 192.168.0.3
11      vars:
12        ansible_user: root
13    pve_primary:
14      hosts:
15        pve01:
16    pve_secondary:
17      hosts:
```

```
18     pve02:
19     pve03:
```

pve01 is a member of both pve and pve_primary. That is intentional and normal: groups are labels, not folders. Verify what Ansible actually parsed with:

```
ansible-inventory --list --yaml
ansible-inventory --graph
```

2.2 Configuration

Ansible reads `ansible.cfg` from the **current working directory** (among other places). Put it next to your inventory and always run Ansible from that directory.

```
1  [defaults]
2  inventory          = inventory/hosts.yml
3  host_key_checking  = False
4  interpreter_python = auto_silent
5
6  [ssh_connection]
7  pipelining         = True
```

⚠ host_key_checking = False is a lab shortcut

It disables SSH host key verification and therefore any protection against a man-in-the-middle. It is fine for lab machines that are thrown away after the workshop. In production you keep verification on and distribute `known_hosts` instead.

3 Ad-hoc commands

An ad-hoc command runs a single module once, without writing a file. Use it to look around, to test connectivity, and to try a module before you put it in a playbook.

```
1  cd ~/ansible-proxmox # see the
warning below
2  ansible pve -m ping # reachability
3  ansible pve -m ansible.builtin.setup \
4      -a 'filter=ansible_distribution*' # gathered facts
5  ansible pve -m ansible.builtin.command -a 'pveversion' # run a command
6  ansible pve01 -m ansible.builtin.command -a 'pvecm status' # single host
```

The host pattern (pve, pve01, all, pve*) selects the targets; -m names the module; -a passes its arguments.

! Run these from ~/ansible-proxmox

Ansible reads `ansible.cfg` from the **current working directory**, not from the location of the playbook and not from your home directory. That file is what points at `inventory/hosts.yml`. Run an ad-hoc command anywhere else and you get:

```
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available.
Note that
the implicit localhost does not match 'all'
[WARNING]: Could not match supplied host pattern, ignoring: pve
```

Three warnings, exit code 0, nothing executed. There is no error, because an empty inventory is a legitimate state. When a command silently does nothing, check `pwd` first.

3.1 Reading the result

Ad-hoc output is not the PLAY RECAP you get from `ansible-playbook`. There are two shapes, and which one you see depends on the module.

Modules that return data print JSON:

```
pve01 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.13"
  },
  "changed": false,
  "ping": "pong"
}
```

`ansible_facts.discovered_interpreter_python` shows up in almost every result. It is Ansible telling you which Python it picked on the target, not something the module did.

command and **shell** print a header line and then the raw output of the program, unparsed:

```
pve01 | CHANGED | rc=0 >>
pve-manager/9.2.2/b9984c6d90a4bd80 (running kernel: 7.0.2-6-pve)
```

Note CHANGED on a command that only read a version number. `command` cannot know whether what it ran changed anything, so it reports changed every time. Module 03 comes back to this.

A failure carries the return code and the program's own error:

```
pve01 | FAILED | rc=2 >>
Error: Corosync config '/etc/pve/corosync.conf' does not exist - is this node
part of a cluster?
```

The hosts answer in whatever order they finish, not in inventory order: ad-hoc runs them in parallel, five at a time by default (`forks` in `ansible.cfg`).

Word	Meaning
SUCCESS	the host is already in the desired state; nothing was done
CHANGED	Ansible changed something on the host
FAILED	the task ran and did not succeed
UNREACHABLE	Ansible could not connect at all

FAILED and UNREACHABLE are different problems: the first is your task, the second is your network, SSH or credentials.

In a playbook the same states appear as ok, changed, failed, unreachable and skipped in the PLAY RECAP. Same concepts, lower case, counted per host.

4 Idempotency

Ansible modules describe a **target state** and do only what is missing to reach it.

```
ansible pve -m ansible.builtin.file -a 'path=/root/demo state=directory'
```

Run it once: changed. Run it again: ok. Nothing was recreated, nothing was reset. This is what makes it safe to run the same playbook every day, and it is why you can use your playbook as documentation of the target state.

4.1 The exception: command and shell

```
1  - name: Read the version          # BAD: reports changed on every single
   execution
2  ansible.builtin.command: pveversion
```

Ansible cannot know what an arbitrary command does, so it reports changed every time, and worse, it *executes* every time. Exercise 01 shows this on pveversion, which changes nothing at all and still reports CHANGED.

Reach for a module before you reach for command. Nearly everything you would type on a Proxmox VE node has one, including the cluster commands: module 03 forms the cluster with community.proxmox.proxmox_cluster rather than pvecm. command is what is left when nothing else fits, and then you supply the guard yourself:

```
1  - name: Fetch the Debian 13 container template
2      ansible.builtin.command: pveam download local debian-13-
   standard_13.6-1_amd64.tar.zst
3      args:
4          # skip the task if this path already exists
5          creates: /var/lib/vz/template/cache/debian-13-
   standard_13.6-1_amd64.tar.zst
```

Other guards you will meet: `when:` (a condition), `changed_when:` (decide yourself what counts as a change), and `failed_when:`. The command [module page](#) lists all of them, and `ansible-doc ansible.builtin.command` is the same content offline.

💡 Tip

Prefer dedicated modules over `command` or `shell`. If a shell command is unavoidable, always add state guards (`creates`, `removes`, `changed_when`).

5 Command reference

```
ansible --version           # version of ansible-core
ansible-inventory --graph   # display parsed inventory hierarchy
ansible <pattern> -m ping   # reachability check
ansible <pattern> -m <module> -a '<args>' # ad-hoc run
ansible-doc <module>        # show module documentation offline
ansible-doc -l community.proxmox # list collection modules
```

6 Documentation links

- [Ansible Documentation](#)
- [Inventory Guide](#)
- [Ad-hoc Commands](#)
- [Configuration Settings](#) (`ansible.cfg`)
- [Return Values](#)

Offline inspection:

```
ansible-doc ansible.builtin.file           # documentation for installed
module version
ansible-doc -s ansible.builtin.file        # concise parameter snippet
ansible-doc -l community.proxmox           # list all Proxmox modules
```