Exercise 01 — Ansible in 30 Minutes

Ansible for Proxmox VE · Module 01

credativ GmbH

~10 min · on `desktop`, in `~/ansible-proxmox`

> **Objective:** talk to all three nodes from a project you understand, and watch a second run do nothing.

> ! Where the sheets are
>
> All exercise sheets, handouts, the slides and the cheat sheet are on the share that is mounted on your desktop. Open it once and keep the tab:
>
> ```
> xdg-open ~/Desktop/workshop/index.html
> ```
>
> Every module has a handout (the reference, with more detail than the slides) and an exercise sheet like this one. You do not need to print or copy anything.

> i Where to look
>
> Everything here was on the slides. When you want more than that: Inventory · Ad-hoc commands · `ansible.cfg` · `file` module · offline: `ansible-doc <module>` and the handout for this module.

1 — Know what you are running

`~/ansible-proxmox` is ready. Read `ansible.cfg` and `inventory/hosts.yml` before you use them, then check that Ansible parses the inventory the way you just read it:

```
cd ~/ansible-proxmox
ansible-inventory --graph
```

Done when: `pve` has three hosts, `pve_primary` one, `pve_secondary` two — and you can say why `pve01` appears twice.

2 — Reach the nodes

```
ansible pve -m ping
```

Done when: three `SUCCESS` blocks come back. Nothing asked for a password: `deploy.sh` put your SSH key on the nodes. Now see the unprepared case, which is what you will meet everywhere else:

```
ansible pve -m ping --ask-pass      # password123
```

3 — Aim

Say the number out loud *before* you press enter, three times:

```
ansible pve_primary -m ping
ansible pve_secondary -m ping
ansible 'pve0*' -m ping
```

Done when: one host answered, then two, then three. Groups and patterns select hosts; they are labels, not folders.

Ansible patterns are not shell globs: * works, `pve0[23]` matches nothing at all — and matching nothing is not an error, it is an empty run.

4 — Run the same thing twice

Create the directory `/root/demo` on all three nodes with the `file` module, then run your command a second time, unchanged.

The command is not written out here, and `file` is a module you have not used yet. Looking it up is half the exercise:

```
ansible-doc ansible.builtin.file        # OPTIONS near the top, EXAMPLES
further down
ansible-doc -s ansible.builtin.file   # every option as a paste-ready snippet
```

`ansible-doc` is the same content as the module page online, offline and for the exact version you are running. Two options are enough here: `path` and `state`.

Done when: the first run reports `CHANGED`, the second `SUCCESS`, and nothing else in the result differs. If you can say why that matters, you have the point of this module.

If you finish early

- Do task 4 again with `ansible.builtin.command` and `mkdir -p /root/demo`. What does it report the second time, and why?
- `ansible pve01 -m ansible.builtin.setup` with no filter. How many facts are there? Find `ansible_memtotal_mb`.
- `ansible-doc ansible.builtin.file` — which other values does `state` accept?
- Remove `/root/demo` again with a single ad-hoc command.
- `ansible pve01 -m ansible.builtin.command -a 'pvecm status'`. Read the error. It is true, and it is what module 03 is about.
- Write `ansible.cfg` and the inventory from scratch in an empty directory.

Solutions

Solution — Task 1

```
cd ~/ansible-proxmox
ansible-inventory --graph
```

```
@all:
  |--@ungrouped:
  |--@pve:
  |  |--pve01
  |  |--pve02
  |  |--pve03
  |--@pve_primary:
  |  |--pve01
  |--@pve_secondary:
  |  |--pve02
  |  |--pve03
```

`pve01` appears twice because groups are labels, not folders. Module 03 depends on exactly that split.

Check that they read the two files rather than skipping ahead: ask one participant what `host_key_checking = False` does, and another which file tells Ansible where the inventory is. `ansible --version | grep "config file"` settles the second question and prints the path of the `ansible.cfg` actually in use.

If it fails: they are not in `~/ansible-proxmox`. Ansible reads `ansible.cfg` from the current working directory, not from the playbook directory.

Solution — Task 2

```
ansible pve -m ping            # key-based, no prompt
ansible pve -m ping --ask-pass # password: password123
```

Expected either way: three `SUCCESS` blocks with `"ping": "pong"`, in whatever order the hosts finish.

If it fails:

- Three `[WARNING]` lines and nothing else → you are in the wrong directory. The single most common one; check it first.
- `Permission denied` → the key never reached the node, or `ansible_user: root` is missing from the inventory. Rerunning `~/Desktop/workshop/deploy.sh` installs the key again.
- `UNREACHABLE` → the node is down or the address in the inventory is wrong.
- `sshpass is required` → only `--ask-pass` needs it; install it on the desktop.

Solution — Task 3

```
ansible pve_primary -m ping     # 1 host
ansible pve_secondary -m ping   # 2 hosts
ansible 'pve0*' -m ping         # 3 hosts
```

The prediction is the exercise. Ask two or three people for their number before the first run — a wrong answer here is worth more than a right one, because it shows the inventory has not landed yet.

Quote the wildcard, or the shell expands it before Ansible sees it.

Ansible patterns are not shell globs. Measured on the lab:

Pattern	Hosts
`pve_primary` / `pve_secondary`	1 / 2
`pve0*`, `pve*`	3
`pve02,pve03`	2
`!pve01`	2
`pve0[23]`	**0**
`pve0[2:3]`	**0**

A character class matches nothing, and the slice syntax `[2:3]` applies to groups, not to host names. Neither is an error — you get an empty run and a warning, which is the same thing participants see when they are in the wrong directory.

Solution — Task 4

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

The command is not on the sheet on purpose: it was on the idempotency slide, and reconstructing it is the only place in this module where they have to produce something rather than copy it. Expect a few people to need a nudge towards `ansible-doc ansible.builtin.file`; that is a good outcome, not a failure.

The `file` module stats the path, finds a directory with matching attributes, and reports `SUCCESS`. For contrast:

```
ansible pve -m ansible.builtin.command -a 'mkdir -p /root/demo'   # CHANGED, twice
```

Same result on the machine, different report. This is the clearest demonstration in the workshop of why modules beat commands, so spend the extra minute on it.

Solution — Optional tasks

```
ansible pve01 -m ansible.builtin.setup | wc -l                    # a few
hundred lines
ansible pve01 -m ansible.builtin.setup -a 'filter=ansible_mem*'
ansible-doc ansible.builtin.file                                  # state:
absent, directory,
                                                                   # file,
hard, link, touch
ansible pve -m ansible.builtin.file -a 'path=/root/demo state=absent'
```

`pvecm status` **fails on purpose**: `pve01 | FAILED | rc=2 >>` and `Error: Corosync config '/etc/pve/corosync.conf' does not exist - is this node part of a cluster?` The node is not in a cluster, so there is no cluster status to report. Say so explicitly: the error is information about the lab, not a broken environment, and module 03 is the fix. Worth demonstrating from the projector even if nobody gets that far.

Automated verification

```
./tests/run.sh 01
```

See `tests/README.md`. The checks for this module are environment-independent and already usable; they confirm the config file location, the parsed inventory groups, and reachability of all three nodes.