Playbook Reference (Preflight, Verify, Teardown)

Playbooks Overview

The repository contains four playbooks under `playbooks/`:

Playbook	Purpose	Changes target?
`preflight.yml`	Verify readiness; no changes	No
`ssh-baseline.yml`	Run preflight then apply the baseline role	Yes
`verify.yml`	Post-deployment validation	No
`teardown.yml`	Reverse the role (testing only)	Yes — destructive

All four playbooks share common properties: `serial: 1` (one host at a time), `any_errors_fatal: true` (stop the whole rollout on first failure), and `gather_facts: true` (need facts for virtualization detection, OS version checks, etc.).

preflight.yml, ssh-baseline.yml, and verify.yml reference `hosts: targets` — the deployment scope group. **teardown.yml uses** `hosts: all` deliberately, because teardown may need to operate on hosts that have been removed from `targets` for cleanup purposes.

---

preflight.yml

Verification-only playbook. Makes zero changes to target hosts.

```
---
# Run preflight verification only. Makes no changes to target hosts.
# Usage: ansible-playbook playbooks/preflight.yml -l pbr-uisp-kl1


- name: Preflight verification
```

```
hosts: targets
gather_facts: true
serial: 1
any_errors_fatal: true
roles:
  - role: preflight
```

Delegates entirely to the `preflight` role. That role imports five task files:

Task file	Tags	Scope
`local.yml`	`preflight, local`	Target host: OS, hostname, NTP, users, APT Universe, sudoers
`ad.yml`	`preflight, ad`	Target host: AD DC reachability on TCP 88 and 389
`scepman.yml`	`preflight, scepman`	Target host: SCEPman /ca endpoint reachability and CA validity
`schema.yml`	`preflight, schema`	Control node (delegate_to: localhost): AD schema has sshPublicKey
`control-node.yml`	`preflight, control`	Control node: vault password file, vault decryption, collections

Local checks (local.yml)

1. **OS is Ubuntu** — `ansible_distribution == "Ubuntu"`
2. **Ubuntu major >= 22** — configurable via `preflight_min_ubuntu_major`
3. **Hostname is real** — not `localhost`, `ubuntu`, or empty
4. **Hostname resolves** — `getent hosts <ansible_hostname>`
5. **NTP synchronised** — `timedatectl show -p NTPSynchronized --value` returns `yes`
6. **Required local users exist** — `ansible` and `pbr_admin` (configurable via `preflight_required_users`)
7. **APT Universe enabled** — `oddjob` and `oddjob-mkhomedir` have candidate versions. Hardened images sometimes disable Universe; fail fast.
8. **Sudoers validates** — `visudo -c` passes (with one specific exception, see below)

ThreatLocker sudoers exception

ThreatLocker's agent installs `/etc/sudoers.d/threatlocker_sudoers_general` with incorrect permissions. The file cannot be fixed because ThreatLocker enforces immutability on its own files. The preflight task ignores this specific failure:

```
- name: Validate sudoers (ignoring known ThreatLocker permission issue)
  ansible.builtin.command: visudo -c
  register: visudo_check
  changed_when: false
  failed_when:
    - visudo_check.rc != 0
    - visudo_check.stderr_lines | reject('search', 'threatlocker_sudoers_general') | list |
length &gt; 0

- name: Warn when ThreatLocker sudoers workaround is active
  ansible.builtin.debug:
    msg: &gt;-
      KNOWN ISSUE: /etc/sudoers.d/threatlocker_sudoers_general has incorrect
      permissions and cannot be modified due to ThreatLocker enforcement.
      sudo is NOT honouring that file. Raise with ThreatLocker support.
      Preflight is treating this as a known exception only.
  when:
    - visudo_check.rc != 0
    - "'threatlocker_sudoers_general' in visudo_check.stderr"
```

The `failed_when` filter: `stderr_lines | reject('search', 'threatlocker_sudoers_general')` removes any line mentioning that file, and only fails if there's still error output after the rejection. Any other sudoers error still fails the task.

When the workaround fires, a clear warning is printed so the operator knows it's been hit. The intent is to surface it for ongoing visibility, not to silently ignore it.

AD checks (ad.yml)

1. **Resolve AD domain** — `getent hosts pbr.org.au`. Parses output into a list of discovered DC IPs.
2. **Probe Kerberos/LDAP ports** — `wait_for` on each DC IP × each port in `preflight_ad_ports` ([88, 389]). 5-second timeout per probe.
3. **Check existing realm membership** — informational only. If the host is already joined, preflight does not fail; the baseline role's `realm join` task will skip if already joined.

SCEPman check (scepman.yml)

1. **Extract hostname** from `scepman_ca_url` via `urlsplit('hostname')`
2. **Resolve hostname** — `getent hosts pki.pbr.org.au`

3. **GET /ca** — downloads the CA cert to `/tmp/preflight-scepman-ca.der` with status_code 200, timeout 10s
4. **Parse with openssl** — `openssl x509 -inform DER -text -noout`. Verifies output contains `CA:TRUE` (the cert is genuinely a CA cert, not just any cert).
5. **Clean up** — remove the temp cert file.

Schema check (schema.yml)

Runs from the control node via `delegate_to: localhost`, `become: false`, `run_once: true`. Requires `python3-ldap` on the controller and the `community.general.ldap_search` module. Searches the AD Schema container for an entry with `cn=sshPublicKey`. Fails if not found.

Can be skipped (set `preflight_skip_schema_check: true`) if python3-ldap is unavailable and you've verified schema manually via another tool.

Control-node checks (control-node.yml)

1. **Vault password file exists** — `~/.ansible_vault_pass` present
2. **Mode 0600 or 0400** — not readable by anyone but the owner
3. **Vault decrypts to non-empty values** — `ad_join_user` and `ad_join_password` exist after vault decryption (asserted with `no_log: true`)
4. **Required collections installed** — `community.general` and `ansible.posix` are present

---

ssh-baseline.yml

The main deployment playbook. Two plays in sequence:

```
---
# Preflight verification followed by baseline application.
# serial: 1 ensures one host completes (or fails) before others are touched.
# any_errors_fatal stops the entire rollout if any host fails.

- name: Preflight verification
  hosts: targets
  gather_facts: true
  serial: 1
  any_errors_fatal: true
  roles:
```

```
    - role: preflight

- name: Apply SSH baseline
  hosts: targets
  gather_facts: true
  serial: 1
  any_errors_fatal: true
  roles:
    - role: ssh-baseline
```

The first play runs preflight (defence in depth — even if an operator just runs `ssh-baseline.yml` directly, preflight executes first). The second play applies the baseline.

Because `serial: 1` and `any_errors_fatal: true` are set on both plays, a host that fails preflight in play 1 stops the entire rollout before play 2 begins. A host that fails the baseline in play 2 stops further hosts from being processed.

The `ssh-baseline` role's `tasks/main.yml` orchestrates the work:

```
---
- name: Verify preconditions
  ansible.builtin.import_tasks: preconditions.yml
- name: Install SCEPman root CA
  ansible.builtin.import_tasks: ca-trust.yml
- name: Install required packages
  ansible.builtin.import_tasks: packages.yml
- name: Configure system timezone
  ansible.builtin.import_tasks: timezone.yml
- name: Join Active Directory and configure SSSD
  ansible.builtin.import_tasks: ad-join.yml
- name: Configure sudo
  ansible.builtin.import_tasks: sudo.yml
- name: Configure Duo MFA
  ansible.builtin.import_tasks: duo.yml
- name: Harden sshd
  ansible.builtin.import_tasks: sshd.yml
- name: Configure fail2ban
  ansible.builtin.import_tasks: fail2ban.yml
```

The order matters: CA trust before package install (the package metadata is over HTTPS); AD join before sudo (sudoers references the AD sudo group); Duo before sshd (sshd hardening references

the Duo PAM stack); fail2ban last (no dependencies, but jail.local references the final sshd port).

The auditd auto-detection in packages.yml

The packages task installs `auditd` and `audispd-plugins` unconditionally (they're harmless on LXC). The conditional logic decides whether to **enable and start** the auditd service:

```
- name: Determine whether to manage auditd on this host
  ansible.builtin.set_fact:
    _manage_auditd: &gt;-
      {{
        (manage_auditd | bool)
        if (manage_auditd is boolean
            or manage_auditd | string | lower in ['true', 'false', 'yes', 'no'])
        else (ansible_virtualization_type | default('') != 'lxc')
      }}

- name: Report auditd management decision
  ansible.builtin.debug:
    msg: &gt;-
      auditd on {{ inventory_hostname }}:
      {{ 'will be managed' if _manage_auditd else 'SKIPPED (LXC container or explicit
override)' }}
      [virtualization_type={{ ansible_virtualization_type | default('unknown') }},
       manage_auditd={{ manage_auditd }}]

- name: Enable auditd
  ansible.builtin.service:
    name: auditd
    state: started
    enabled: true
  when: _manage_auditd | bool
```

The expression: if `manage_auditd` is set to a boolean-like value (`true`, `false`, `yes`, `no`), use that. Otherwise (e.g. when set to the string `'auto'`), evaluate `ansible_virtualization_type != 'lxc'` — manage on KVM/bare metal, skip on LXC.

The debug task logs the decision and the inputs that produced it. This is visible in every playbook run, making the auditd state explicit per host.

---

verify.yml

Post-deployment validation. Requires the `verify_test_user` extra variable.

```
ansible-playbook playbooks/verify.yml -l pbr-uisp-kl1 \
    -e verify_test_user=a.mfraser \
    --vault-password-file ~/.ansible_vault_pass
```

The first task asserts the variable was supplied with a clear error message if not. Then the validation steps:

Check	Mechanism
Realm membership	`realm list --name-only` contains `{{ ad_domain }}`
AD user resolves via SSSD	`getent passwd {{ verify_test_user }}` rc == 0
SSH key retrievable	`/usr/bin/sss_ssh_authorizedkeys {{ verify_test_user }}` returns non-empty stdout
sshd config valid	`sshd -t` against the live combined config
auditd managed correctly	`_manage_auditd` recomputed; if true, `auditd.service` state == running
Critical services	`ssh.service`, `sssd.service`, `fail2ban.service` all running
fail2ban sshd jail	`fail2ban-client status sshd` rc == 0
Duo in sudo PAM stack	`grep -E "^auth.*pam_duo.so" /etc/pam.d/sudo`
sudo timestamp_timeout drop-in	`/etc/sudoers.d/sudo_timestamp_timeout` exists
ansible NOPASSWD sudo	`sudo -n true` as the `ansible` user succeeds
pbr_admin not in sg_sudo	If `pbr_admin` were in `sg_sudo`, it would hit Duo on sudo — defeating break-glass

The auditd recomputation in verify.yml

verify.yml duplicates the auditd auto-detection logic from packages.yml. This is intentional: verify.yml runs independently and may be invoked without re-running the role. It needs to know whether auditd should be running on this host:

```
- name: Determine whether auditd should be running on this host
  ansible.builtin.set_fact:
    _manage_auditd: &gt;-
      {{
```

```
        (manage_auditd | bool)
        if (manage_auditd is defined
            and (manage_auditd is boolean
                 or manage_auditd | string | lower in ['true', 'false', 'yes', 'no']))
        else (ansible_virtualization_type | default('') != 'lxc')
      }}

- name: Verify auditd running (where managed)
  ansible.builtin.assert:
    that:
      - ansible_facts.services["auditd.service"].state == "running"
    fail_msg: "auditd should be running but is not"
  when: _manage_auditd | bool
```

The auditd assertion is conditional on `_manage_auditd`. On LXC hosts (`pbr-graylog-kl1`, `pbr-thingsboard-kl1`), verify.yml does not check that auditd is running because the role didn't enable it. Documented as a known compliance gap in **Known Limitations**.

verify.yml summary output

At the end, verify.yml prints a multi-line summary:

```
TASK [Verification summary] *********
ok: [pbr-uisp-kl1] =&gt;
  msg:
  - '==================== VERIFICATION PASSED ===================='
  - 'Joined to realm:        pbr.org.au'
  - 'AD user resolves:       a.mfraser (1234:5678)'
  - 'SSH key retrieved:      ssh-ed25519 AAAA...'
  - 'sshd config valid:      yes'
  - 'All services running:   ssh, sssd, fail2ban, auditd'
  - ''
  - 'Next: SSH from your workstation as a.mfraser@pbr-uisp-kl1'
  - 'Expect: key auth + Duo push for SSH; Duo push + AD password for sudo'
```

On LXC, the services line reads: `ssh, sssd, fail2ban (auditd skipped: LXC)`.

---

teardown.yml

WARNING: This playbook is destructive. It is intended for testing — specifically, for restoring a host to a ~clean Ubuntu state before re-running `ssh-baseline` from scratch. It is **not** a production rollback.

From the playbook header:

> This will sever SSH access for AD users on the target host. Keep your `pbr_admin` and `ansible` (publickey) sessions open. After teardown, AD computer object must be deleted from AD before re-join.

Survival pattern

After teardown, the only paths into the host are:

- `pbr_admin` session that was open before teardown (still active)
- `ansible` publickey from the control node — survives because the local `ansible` account isn't touched
- Console / out-of-band access (Proxmox console, ScreenConnect, etc.)

AD users **cannot** log in until the role is re-applied. New `pbr_admin` SSH sessions **cannot** log in either, because teardown reverts `/etc/ssh/sshd_config.d/10-pbr-hardening.conf` and the `Match User pbr_admin` block goes with it.

What teardown removes

Listed in order of execution:

1. **fail2ban** — stop, disable, remove `jail.local`
2. **sshd hardening** — remove `/etc/ssh/sshd_config.d/10-pbr-hardening.conf`, remove `/etc/issue.net` (note: this also deletes the Include directive's effect, since there are no other drop-ins)
3. **Duo PAM** — restore `/etc/pam.d/sshd` from dpkg-dist (or reinstall openssh-server), remove sudo timestamp drop-in, reinstall sudo package to restore `/etc/pam.d/sudo`
4. **Duo packages** — purge `duo-unix`, purge legacy `libpam-duo`/`libduo3`, remove Duo APT source, remove Duo GPG keys, remove `/etc/duo` directory
5. **sudoers drop-ins** — remove `/etc/sudoers.d/ad_sudo` and `/etc/sudoers.d/pbr_admin`
6. **AD / SSSD** — `realm leave` if joined, stop and disable SSSD, remove keytab, clear SSSD caches and DB, remove `/etc/sssd/sssd.conf`, restore minimal `/etc/krb5.conf`
7. **SCEPman CA** — remove `/usr/local/share/ca-certificates/scepman-root-ca.crt`, run `update-ca-certificates --fresh`

What teardown deliberately does NOT do

The closing comment in teardown.yml:

> Note: leaving installed packages alone. The following are installed by the role but harmless to leave: sssd, sssd-tools, libnss-sss, libpam-sss, adcli, realmd, samba-common-bin, krb5-user, oddjob, oddjob-mkhomedir, auditd, unattended-upgrades, libpam-modules, fail2ban. Re-running the role finds them present and proceeds normally.

So teardown is "config-only" — package state isn't reversed. This makes the playbook faster and keeps re-deployment idempotent.

The `failed_when: false` pattern

Many teardown tasks have `failed_when: false` — the playbook is intentionally tolerant of partial prior state. If `realm leave` errors because the host is already de-realmed, that's fine. If `systemd` can't stop fail2ban because it's already stopped, that's fine. Teardown's job is to reach a known end state, not to enforce that all prior state was as expected.

After teardown

To re-deploy:

1. Delete the AD computer object in ADUC (`realm leave` doesn't always remove it cleanly; even if it did, replication lag can leave stale references)
2. Re-run `ansible-playbook playbooks/ssh-baseline.yml -l <host> --vault-password-file ~/.ansible_vault_pass`

If you skip step 1, the first realm join attempt almost certainly fails with "Computer object already exists".

Usage

```
ansible-playbook playbooks/teardown.yml -l pbr-test-kl1 \
    --vault-password-file ~/.ansible_vault_pass
```

The playbook uses `hosts: all` — the `-l` limit pattern is the only thing keeping it from running everywhere. **Always use `-l` with teardown.** Forgetting `-l` would attempt to tear down every

host in inventory.

Common Operational Patterns

Run preflight against multiple hosts before a wave

```
ansible-playbook playbooks/preflight.yml -l 'pbr-host1-kl1,pbr-host2-kl1,pbr-host3-kl1'
```

preflight is read-only, so running it against a wave of hosts before starting the actual baseline rollout is the standard "are we ready?" check.

Re-run baseline after a config change

The role is idempotent. Running it against an already-baselined host re-applies any drifted config and confirms current state. Useful after editing role defaults or vault entries.

```
ansible-playbook playbooks/ssh-baseline.yml -l pbr-uisp-kl1 \
    --vault-password-file ~/.ansible_vault_pass
```

Run verify after a host's package update window

If unattended-upgrades patches OpenSSH or libpam-* packages overnight, run verify to confirm no regression:

```
ansible-playbook playbooks/verify.yml -l pbr-uisp-kl1 \
    -e verify_test_user=a.mfraser \
    --vault-password-file ~/.ansible_vault_pass
```

Where to Read Next

- **Deployment Runbook — New Host** — the standard sequence of preflight → ssh-baseline → verify
- **Known Limitations, Troubleshooting & Version History** — what to do when preflight or verify fails
- **Architecture & Design Decisions** — why preflight is a separate role, why `serial: 1`

Revision #1
Created 2026-05-13 05:33:15 UTC by PBR_Documentation
Updated 2026-05-13 05:33:15 UTC by PBR_Documentation