Ansible

Ansible-based configuration management for PBR Linux infrastructure. Hosts the ssh-baseline role and related playbooks. Source: github.com/Puffing-Billy-Railway/pbr-infra

Overview & Repository Layout

Purpose

This book documents PBR's Ansible-based configuration management for Linux infrastructure. It covers the `ssh-baseline` role, supporting playbooks, design rationale, deployment procedure, and operational reference.

The `ssh-baseline` role establishes a hardened, AD-integrated SSH access baseline on Ubuntu servers. It joins each host to Active Directory via SSSD, retrieves SSH public keys from AD (via the `sshPublicKey` schema extension), enforces Duo MFA on both SSH login and sudo, applies CIS-aligned sshd hardening, and configures fail2ban.

Source Repository

GitHub: `git@github.com:Puffing-Billy-Railway/pbr-infra.git`

Branch: `main` — all production-ready changes commit here. There are no other long-lived branches.

Tags: Semantic version tags mark each baseline release (`v2.3`, `v2.4`, `v2.4.1`, `v2.4.2`). The current production release is **v2.4.2**.

Cloning the repo

```
git clone git@github.com:Puffing-Billy-Railway/pbr-infra.git
cd pbr-infra
```

Vault

The repo contains an encrypted Ansible Vault file at `inventory/group_vars/all/vault.yml`. The vault password lives at `~/.ansible_vault_pass` on the control node (mode 0600). Vault contents include:

- `vault_ad_join_user` — AD service account UPN for realm join
- `vault_ad_join_password` — that account's password
- `vault_duo_ikey`, `vault_duo_skey`, `vault_duo_api_host` — Duo Auth API credentials

The vault is never decrypted to disk; `ansible-playbook` reads `--vault-password-file ~/.ansible_vault_pass` at runtime.

Current Deployment State

All hosts run **ssh-baseline v2.4.2**:

Host	IP	Virtualization	auditd	Notes
`pbr-uisp-kl1`	10.1.8.23	KVM	Managed	Canary — deploy and verify here first
`pbr-docker-kl1`	10.1.8.55	KVM (Ubuntu 24.04)	Managed	Docker host
`pbr-graylog-kl1`	10.1.8.26	LXC	Skipped	auditd auto-skipped on LXC (see Known Limitations)
`pbr-lme-kl1`	10.1.8.35	KVM	Managed	Logging Made Easy
`pbr-thingsboard-kl1`	10.1.8.25	LXC	Skipped	ThingsBoard for level crossing telemetry

---

Control Node

Hostname: `pbr-ansible-kl1`

Working directory: `~/pbr-infra` (under `pbr_admin`)

The control node is explicitly excluded from inventory targets — playbooks reference `hosts: targets` rather than `all`, so the control node cannot be accidentally hit by a baseline run. The relevant comment in `inventory/hosts.yml`:

```
# Control node - excluded from automation.
# Uncomment only if you intentionally need ansible-kl1 in inventory
# (e.g., for monitoring or facts gathering) - never as an ssh-baseline target.
```

```
# pbr-ansible-kl1:
#   ansible_host: 127.0.0.1
```

The ansible service account on the control node uses an ed25519 private key (`~/.ssh/ansible_svc`). Public key:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBMc7IDlr/IZ5M/2HcXU7cGCKZ03SLjpr5cbmiHnokdP ansible-svc@pbr-ansible-kl1
```

This public key is installed on every target host by the bootstrap script (see Deployment Runbook).

Repository Layout

```
pbr-infra/
├── ansible.cfg                     # Inventory path, become config, vault password file
├── requirements.yml                # Collection dependencies
├── inventory/
│   ├── hosts.yml                   # Host definitions and `targets` group
│   └── group_vars/all/
│       ├── main.yml                # AD domain config (non-secret)
│       └── vault.yml               # Encrypted secrets (vault)
├── playbooks/
│   ├── preflight.yml               # Verification only (no changes)
│   ├── ssh-baseline.yml            # Preflight + apply baseline
│   ├── verify.yml                  # Post-deployment validation
│   └── teardown.yml                # Reverse the role (testing)
├── roles/
│   ├── preflight/                  # Preflight checks (separate role)
│   │   ├── defaults/main.yml
│   │   ├── meta/main.yml
│   │   └── tasks/
│   │       ├── main.yml
│   │       ├── local.yml           # OS, hostname, NTP, users, sudoers
│   │       ├── ad.yml              # AD DC reachability
│   │       ├── scepman.yml         # SCEPman CA reachability
│   │       ├── schema.yml          # sshPublicKey schema check
│   │       └── control-node.yml    # Vault password file, collections
│   └── ssh-baseline/               # Main role
```

```
│       ├── defaults/main.yml           # All tunable variables
│       ├── handlers/main.yml           # sshd, sssd, fail2ban, ca-cert restarts
│       ├── meta/main.yml
│       ├── tasks/
│       │   ├── main.yml                # Task orchestration
│       │   ├── preconditions.yml       # Ansible account local sudo group
│       │   ├── ca-trust.yml            # SCEPman root CA installation
│       │   ├── packages.yml            # apt installs, auditd auto-detect
│       │   ├── timezone.yml            # Australia/Melbourne
│       │   ├── ad-join.yml             # realm join, SSSD config
│       │   ├── sudo.yml                # AD sudo + pbr_admin sudoers drop-ins
│       │   ├── duo.yml                 # duo-unix install, PAM stacks
│       │   ├── sshd.yml                # Hardening drop-in, banner, validate
│       │   └── fail2ban.yml            # jail.local
│       └── templates/
│           ├── krb5.conf.j2            # Minimal client config; SRV discovery
│           ├── sssd.conf.j2            # AD provider, GPO disabled, access filter
│           ├── sshd_hardening.conf.j2  # 10-pbr-hardening.conf
│           ├── pam_sshd.j2             # /etc/pam.d/sshd with Duo + break-glass
│           ├── pam_sudo.j2             # /etc/pam.d/sudo with Duo + carve-outs
│           └── pam_duo.conf.j2         # ikey/skey/host, group restriction
└── scripts/
    └── bootstrap-ansible-user.sh       # Idempotent ansible-account bootstrap
```

Version Tags Overview

See the Known Limitations & Version History page for the full changelog. Quick reference:

Tag	Description
`v2.4.2`	Current release. Auto-skip auditd on LXC containers.
`v2.4.1`	Ensure ansible automation account is in local `sudo` group (post-hardening connectivity fix).
`v2.4`	Duo MFA on sudo for AD sudo group with carve-outs.
`v2.3`	Duo MFA on SSH via `duo-unix` from Duo's official repo (replacing Ubuntu universe `libpam-duo`).

Quick Reference: Standard Workflow

1. Bootstrap the ansible automation account on a fresh host (`scripts/bootstrap-ansible-user.sh`).
2. Pre-clean any stale AD computer object in AD Users & Computers.
3. Add the host to `inventory/hosts.yml` (both the `linux` children and the `targets` group).
4. Run preflight: `ansible-playbook playbooks/preflight.yml -l <host>`
5. Run baseline: `ansible-playbook playbooks/ssh-baseline.yml -l <host> --vault-password-file ~/.ansible_vault_pass`
6. Run verify: `ansible-playbook playbooks/verify.yml -l <host> -e verify_test_user=a.mfraser --vault-password-file ~/.ansible_vault_pass`
7. Manual SSH test from workstation as AD user and as `pbr_admin`.

See the **Deployment Runbook** page for the full procedure including known retry behaviour.

Where to Read Next

- **Architecture & Design Decisions** — the "why" behind each major choice in the role
- **Deployment Runbook — New Host** — step-by-step for adding a new host to the baseline
- **Configuration Reference** — every variable in `defaults/main.yml` explained

Architecture & Design Decisions

Purpose of this Page

This page captures the rationale behind every non-obvious design choice in the `ssh-baseline` role. Each entry follows the pattern: **What we did → Why → Trade-off accepted**.

Where possible, comments inside the role itself reference these decisions; this page consolidates them in one place.

Identity & Access

AD is the source of truth for SSH public keys

What we did: AD user accounts have their SSH public key stored in the `sshPublicKey` attribute (OpenSSH-LPK schema extension). On Linux, sshd retrieves keys via `AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys %u` (run as `nobody`), which queries SSSD which queries AD.

Why: Centralised key lifecycle — offboarding an AD user revokes their SSH access across every host immediately, without touching each server. Users cannot bypass revocation by maintaining their own `~/.ssh/authorized_keys` because `AuthorizedKeysFile` is globally set to `none`.

Trade-off: AD/SSSD must be available for AD users to log in. The `pbr_admin` break-glass account exists precisely for the case where AD/SSSD is unavailable.

Group membership is the sole gate; no per-user allow lists

What we did: SSSD is configured with `ad_access_filter` restricting login to members of `SG_ServerAccess` or `SG_Sudo`. `realm permit --groups` mirrors the same gate at the realmd layer. sshd's `AllowGroups` enforces it again at the SSH protocol layer.

Why: Three independent layers of group-based access control means a misconfiguration in any one layer cannot accidentally grant broader access. Group changes in AD propagate to every host without any local change.

Trade-off: Defence in depth at the cost of slightly more configuration to keep in sync. The role generates all three from the same variables (`ad_server_access_group`, `ad_sudo_group`), so drift is unlikely.

Break-glass: local pbr_admin account, password auth, source-IP restricted

What we did: The `pbr_admin` local account uses password authentication (only), restricted by sshd `Match` block to source IPs in `10.1.0.0/16,192.168.0.0/16`. It has full sudo with the local password (not AD password, no Duo).

Why: If AD, SSSD, or Duo is unavailable, an administrator can still access every host. Password-only is acceptable here because the account is gated by source-IP and protected by fail2ban.

Trade-off: A local password to manage on each host. Mitigation: the password is in 1Password, rotated on demand, and ssh access is source-IP-restricted to PBR admin networks (default `pbr_admin_allowed_sources`).

Ansible automation account: local user, key-only, NOPASSWD sudo

What we did: The `ansible` account is a local Unix user (not in AD). It authenticates by SSH key only and has `NOPASSWD ALL` in sudoers via `/etc/sudoers.d/ansible`.

Why: Ansible needs deterministic, non-interactive access. Tying it to AD or Duo would block automation during AD/Duo outages and require interactive MFA for every play.

Trade-off: A local account with passwordless sudo is a privileged credential. Mitigations: (1) account password is locked (`passwd -l`) — key authentication only, (2) the public key is unique to the control node, (3) the private key on `pbr-ansible-kl1` is owned by `pbr_admin` mode 0600.

SSH & PAM

AuthenticationMethods publickey,keyboard-interactive

What we did: sshd is configured to require both an SSH publickey *and* a keyboard-interactive PAM challenge. PAM is configured so that Duo is the keyboard-interactive challenge for AD users.

Why: This is Duo's documented Ubuntu integration pattern. PAM rather than `ForceCommand` means the MFA happens at the auth phase before the user's shell starts — including any failure path is logged and rate-limited consistently.

Trade-off: Royal TS's Rebex SSH library cannot do `AuthenticationMethods publickey,keyboard-interactive` directly — it supports one auth method per session. Workaround: set Royal TS authentication method to "Any" in Advanced/Security settings. Native OpenSSH clients (including PowerShell `ssh.exe`) handle it correctly.

AllowGroups includes the local sudo group

What we did: `sshd_config`'s `AllowGroups` directive lists `sudo sg_serveraccess sg_sudo`. The local `sudo` group entry is what permits the local accounts (`ansible`, `pbr_admin`) to log in — they are not AD users and have no AD group membership.

Why: A single `AllowGroups` directive is simpler than multiple `Match User` exceptions. Local accounts qualify via local `sudo`; AD users qualify via either AD group.

Trade-off (and the v2.4.1 fix): Any account that needs SSH access must be in the local `sudo` group. Initially the role assumed the bootstrap had handled this for the `ansible` account, but it had been done manually on the canary and not on later hosts. v2.4.1 added an idempotent task to `preconditions.yml` to enforce it.

AuthorizedKeysFile is globally "none"

What we did: Set `AuthorizedKeysFile none` globally, then re-enable `.ssh/authorized_keys` only inside the `Match User ansible` block.

Why: If `AuthorizedKeysFile` were enabled globally, an AD user could drop their own keys into `~/.ssh/authorized_keys` and bypass the AD-side key revocation that's central to the design. The `ansible` account is local and has no AD-side key, so its `Match` block specifically re-enables local key

file lookup.

Trade-off: Slightly non-obvious sshd config. Documented inline in the template.

PAM stack uses pam_succeed_if for break-glass carve-outs

What we did: Both `/etc/pam.d/sshd` and `/etc/pam.d/sudo` use `pam_succeed_if` at the top to detect the break-glass account (`pbr_admin`) and the AD sudo group, branching execution accordingly.

Why: This puts the auth policy in PAM where it can be uniformly logged and audited, rather than depending on multiple sudoers/sshd config layers. It also makes the policy explicit and reviewable in a single file per service.

Trade-off: PAM jump arithmetic (`success=1`, `success=2`, `success=done`) is non-obvious. See the PAM Stack section in the **Duo MFA Integration** page for full explanation.

pam_duo.so referenced by absolute path

What we did: PAM stacks reference `/usr/lib64/security/pam_duo.so` by absolute path rather than relying on PAM's module search path.

Why: Duo's `duo-unix` Debian package installs the module to `/usr/lib64/security/` which is not in Ubuntu's default PAM module search path (Ubuntu expects `/lib/x86_64-linux-gnu/security/`). This is Duo's documented approach for Ubuntu. See https://duo.com/docs/duounix#pam-configuration.

Trade-off: Absolute path is less portable across distributions, but the role only supports Ubuntu so this is acceptable.

Duo MFA

duo-unix from Duo's official APT repo (not Ubuntu universe libpam-duo)

What we did: Install `duo-unix` from Duo's official APT repository (`https://pkg.duosecurity.com/Ubuntu`) and explicitly remove `libpam-duo` / `libduo3` if present.

Why: Inline comment in `roles/ssh-baseline/tasks/duo.yml`:

> 1. Ubuntu universe ships 1.11.3 (2022) which has incompatibilities with current Duo Auth API and returns HTTP 403 in some scenarios.
> 2. Duo's 2.1.0+ is required for the April 2026 CA bundle rotation.
> 3. Duo's docs explicitly target the duo-unix package on Ubuntu 22.04.

Trade-off: An extra APT repository to manage. The role handles GPG key import, repo addition, and legacy package removal automatically.

failmode = safe (not secure)

What we did: `/etc/duo/pam_duo.conf` has `failmode = safe`, meaning if Duo's cloud is unreachable, authentication falls through to single-factor (publickey for SSH, password for sudo).

Why: A Duo cloud outage should not lock administrators out of every Linux host simultaneously. Single-factor publickey is still strong — AD-managed keys with key revocation in effect, plus source-IP restrictions on break-glass.

Trade-off: During a Duo outage, MFA is not enforced. Acceptable because (a) publickey alone is already a strong factor, (b) AD password is still required for sudo, (c) Duo outages are rare and visible.

Duo group restriction limits MFA to AD users

What we did: `pam_duo.conf` has `groups = sg_serveraccess,sg_sudo` (lowercased — SSSD normalises AD group names). pam_duo.so only prompts users in those groups.

Why: Local accounts (`pbr_admin`, `ansible`) should never hit Duo — `pbr_admin` is break-glass (Duo unavailability is exactly when you need it), and `ansible` is automation. The group filter cleanly excludes them.

Trade-off: AD groups must be membered manually. This matches PBR's existing AD-group-driven access management.

sudo timestamp_timeout extended to 30 minutes

What we did: A drop-in at `/etc/sudoers.d/sudo_timestamp_timeout` sets `Defaults timestamp_timeout=30` (default Ubuntu is 15).

Why: Reduces Duo prompt frequency for AD sudo users during typical maintenance sessions. The session-hijack window remains unchanged because the credential cache is per-tty.

Trade-off: Slightly longer interactive sudo grant window. Considered acceptable given the surrounding controls (Duo, AD password, source-IP restriction, fail2ban).

Active Directory / SSSD

ad_gpo_access_control = disabled

What we did: `sssd.conf` sets `ad_gpo_access_control = disabled`.

Why: Per `sssd-ad(5)`, the default is `enforcing`, which evaluates Windows GPO `RemoteInteractiveLogonRight` settings on every SSH login. Any GPO at any parent OU that sets this right (intentionally for Windows servers, or inherited from an ancestor container) would silently deny SSH access to Linux hosts. We use `ad_access_filter` as the sole access control scheme; the sssd-ad(5) manpage explicitly directs disabling GPO control when doing so.

Trade-off: Cannot use Windows GPO to manage Linux SSH access. Acceptable — AD group membership achieves the same control with less surprise.

Explicit DN references in ad_access_filter

What we did: `ad_access_filter` uses full DN references rather than just group names:

```
(|(memberOf=CN=SG_ServerAccess,OU=Security,OU=Groups,DC=pbr,DC=org,DC=au)(memberOf=CN=SG_Sudo,
OU=Security,OU=Groups,DC=pbr,DC=org,DC=au))
```

Why: Direct DN references make the filter unambiguous regardless of LDAP search base. If two groups with the same name existed in different OUs, a name-only filter could match the wrong one.

Trade-off: The filter is bound to the current AD structure. If the security groups move OUs, the filter must be updated.

krb5.conf uses DNS SRV discovery (not static KDC list)

What we did: `/etc/krb5.conf` has `dns_lookup_kdc = true` and no static KDC list. SSSD also writes dynamic snippets to `/var/lib/sss/pubconf/krb5.include.d/`.

Why: Resilient to DC topology changes — new DCs are discovered automatically. PBR has 4 DCs across two sites; SRV records let Kerberos route requests appropriately.

Trade-off: DNS must resolve `_kerberos._tcp.pbr.org.au` SRV records correctly. This is the standard AD-integrated DNS pattern, validated during preflight.

PKI

SCEPman as root CA, distributed via the role

What we did: The role downloads the SCEPman root CA from `https://pki.pbr.org.au/ca`, converts DER to PEM, drops it into `/usr/local/share/ca-certificates/pbr-root-ca.crt`, and runs `update-ca-certificates`.

Why: SCEPman is PBR's chosen ADCS replacement. Distributing the root CA via Ansible means every host trusts the internal PKI — including for Palo Alto IPSec tunnels, Proxmox node TLS, AOS-CX switch EST enrollment, and infrastructure-issued certificates.

Trade-off: SCEPman becomes a dependency for the role to complete. Preflight validates the endpoint reachability before the main role runs.

SCEPman /ca quirk: check mode uses uri+GET, real mode uses get_url

What we did: The CA download task is split: in check mode, it validates reachability via `ansible.builtin.uri` with method GET; in real mode it downloads via `ansible.builtin.get_url`.

Why: SCEPman's `/ca` endpoint returns 404 to HEAD requests (ASP.NET Core/Kestrel quirk). `get_url` does a HEAD pre-check in check mode, which would falsely fail.

Trade-off: Slightly more complex task logic. Documented inline in `ca-trust.yml`.

Operational Behaviour

Preflight is a separate role, importable as a standalone playbook

What we did: `roles/preflight/` is independent from `roles/ssh-baseline/`. The `preflight.yml` playbook runs only preflight; `ssh-baseline.yml` runs preflight first, then the baseline. Both playbooks reference `hosts: targets`.

Why: Operators can validate readiness without making changes. The baseline playbook still runs preflight to ensure it never proceeds against an unverified host. Separating the role makes both phases independently testable.

Trade-off: Two roles to maintain. The preflight role is small and changes infrequently.

serial: 1 and any_errors_fatal: true

What we did: Both playbooks run with `serial: 1` (one host at a time) and `any_errors_fatal: true`.

Why: A failed host stops the whole rollout, preventing fleet-wide breakage from a regression. `serial: 1` means at most one host is in a transient state at any time.

Trade-off: Slower rollouts. Acceptable at PBR's scale (currently 5 hosts; expected ceiling ~10-15).

targets group decouples deployment scope from inventory membership

What we did: Inventory has two groups: `linux` (all known Linux hosts) and `targets` (hosts opted-in to baseline deployment). Playbooks use `hosts: targets` exclusively.

Why: Hosts can be in inventory (for fact-gathering, ad-hoc commands, monitoring) without being in the deployment scope. Most importantly, the control node `pbr-ansible-kl1` can be referenced

but never targeted by a baseline run.

Trade-off: Two places to add a host. Mitigated by the deployment runbook checklist.

auditd: auto-detect LXC and skip (v2.4.2)

What we did: `manage_auditd: auto` is the default. The role evaluates `ansible_virtualization_type` at runtime: if `lxc`, auditd is skipped. The decision is reported via debug task. `manage_auditd: true` or `false` forces the decision explicitly.

Why: auditd cannot run inside LXC containers — the kernel audit netlink interface is isolated from container namespaces, and AppArmor's `lxc-default-cgns` profile blocks the mount operations auditd needs. Even root in the container cannot bind as primary audit consumer. Forcing auditd would fail with EPERM at the systemd start.

Trade-off: LXC hosts have no local audit log capture. Currently `pbr-graylog-kl1` and `pbr-thingsboard-kl1` are affected. Compliance evidence for those hosts depends on remote logging (Graylog SIEM). Documented in **Known Limitations**.

Bootstrap script lives outside the role

What we did: `scripts/bootstrap-ansible-user.sh` is a 13-line bash script run manually as root on a fresh host, before the host enters Ansible inventory.

Why: Ansible needs a working `ansible` account to run the role; the role establishes that account's *environment* (sudo group membership, etc.) but cannot create the account because there's no way in. The bootstrap solves the chicken-and-egg.

Trade-off: A small manual step. Easier than alternatives like cloud-init or pre-baked images.

no_log on the realm join task (and other secret-handling tasks)

What we did: The `realm join` task in `ad-join.yml` has `no_log: true`. The Duo PAM config task has `no_log: true`. The AD schema check has `no_log: true`.

Why: These tasks handle vault-decrypted secrets (AD service account password, Duo secret key). Logging them would leak credentials into stdout, `tee`'d log files, and CI output.

Trade-off: Failure diagnosis is harder because the actual error message is hidden. Temporary workaround during diagnosis: comment out `no_log`, repro, then restore (with cleanup of tee'd logs).

What We Considered but Didn't Do

retries on realm join (deferred to v2.5)

Three of five hosts deployed needed two attempts to complete realm join, despite proper AD pre-clean. Root cause: AD multi-master replication lag — the join hits a DC that hasn't replicated the deletion of the pre-cleaned computer object. Adding `retries: 2, delay: 30` would mask this transparently. Currently the role remains visible about the behaviour and operators retry manually. To be revisited as a v2.5 enhancement.

Per-VM Windows Server licensing analysis

Out of scope for this role — covered in separate licensing analysis. Mentioned here only because the question came up during baseline rollout planning.

SSH on a non-standard port

Ubuntu 22.10+ and 24.04 LTS use systemd socket activation for OpenSSH by default. Changing `ssh_port` from 22 requires also managing socket overrides under `/etc/systemd/system/ssh.socket.d/`. Avoided complexity for marginal security benefit (port-knocking is security theatre; fail2ban handles the brute-force noise). Documented as a comment in `defaults/main.yml`.

Where to Read Next

- **Deployment Runbook — New Host** — how to execute these design choices in practice
- **PAM Stack Design** (in the Duo MFA Integration page) — the carve-out arithmetic explained line by line
- **Known Limitations, Troubleshooting & Version History** — what we accept, what we plan to address

Deployment Runbook — New Host

When to Use This Runbook

Follow this runbook when adding a new Ubuntu host to the SSH baseline. The procedure assumes:

- The host runs Ubuntu 22.04 or 24.04 LTS (the role's supported versions)
- The host has a real hostname (not `ubuntu` or `localhost`)
- The host can reach AD DCs on TCP 88 (Kerberos) and 389 (LDAP)
- The host can reach `https://pki.pbr.org.au/ca` (SCEPman root CA)
- The host has NTP synchronisation working (`timedatectl status` shows `NTPSynchronized=yes`)

Preflight will validate all of these before any changes are made.

Step 1: Bootstrap the ansible automation account

On the **target host**, as root (e.g. via console, ScreenConnect, or your initial admin SSH session):

```
# Copy the bootstrap script to the host. Easiest: paste via SSH session or
# fetch from the repo.
curl -fsSL https://raw.githubusercontent.com/Puffing-Billy-Railway/pbr-
infra/main/scripts/bootstrap-ansible-user.sh \
    -o /tmp/bootstrap-ansible-user.sh

# Inspect it before running
less /tmp/bootstrap-ansible-user.sh

# Run as root
```

```
sudo bash /tmp/bootstrap-ansible-user.sh
```

The script is idempotent. It creates the local `ansible` account, adds it to the `sudo` group, locks the password (key auth only), installs the control node's public key at `~ansible/.ssh/authorized_keys`, and writes `/etc/sudoers.d/ansible` with NOPASSWD.

Full source:

```
#!/bin/bash
# Run as root on a fresh host before adding to ssh-baseline inventory.
# Creates the local ansible automation user with sudo group membership,
# key-only auth, and NOPASSWD sudoers. Idempotent.
set -e

PUBKEY="ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBMc7IDlr/IZ5M/2HcXU7cGCKZ03SLjpr5cbmiHnokdP
ansible-svc@pbr-ansible-kl1"

useradd -m -s /bin/bash -c "Ansible automation" ansible 2>/dev/null || true
usermod -aG sudo ansible
passwd -l ansible

install -d -m 0700 -o ansible -g ansible /home/ansible/.ssh
grep -qxF "$PUBKEY" /home/ansible/.ssh/authorized_keys 2>/dev/null \
    || echo "$PUBKEY" >> /home/ansible/.ssh/authorized_keys
chmod 0600 /home/ansible/.ssh/authorized_keys
chown ansible:ansible /home/ansible/.ssh/authorized_keys

echo "ansible ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/ansible
chmod 0440 /etc/sudoers.d/ansible
visudo -c -f /etc/sudoers.d/ansible

id ansible
```

Verify bootstrap success from the control node:

```
ansible -i 'NEW_HOST_IP,' all -m ping \
    -u ansible -e ansible_user=ansible \
    --private-key ~/.ssh/ansible_svc
```

Expected: `NEW_HOST_IP | SUCCESS => {"ping": "pong"}`. If this fails, fix bootstrap first — do not proceed.

Step 2: Create local pbr_admin break-glass account

On the **target host**, as root:

```
useradd -m -s /bin/bash -c "PBR break-glass admin" pbr_admin
passwd pbr_admin
# Set the password from 1Password (PBR &gt; Linux &gt; pbr_admin)
usermod -aG sudo pbr_admin
id pbr_admin
```

This account must exist before the baseline role runs; `preflight` verifies it.

Step 3: Pre-clean AD (PowerShell, on a domain-joined Windows host with AD module)

If the host has ever been joined to AD — even an aborted attempt — the AD computer object must be deleted before re-joining. Always check, even for fresh hosts (the name may collide with a decommissioned host).

```
# Check whether the computer object exists
Get-ADComputer NEW-HOSTNAME -ErrorAction SilentlyContinue

# If it exists and you're sure it's safe to delete
Get-ADComputer NEW-HOSTNAME -ErrorAction SilentlyContinue | Remove-ADComputer -Confirm:$false

# Confirm gone
Get-ADComputer NEW-HOSTNAME -ErrorAction SilentlyContinue
```

Note: Even with proper pre-clean, the first `realm join` attempt may fail due to AD multi-master replication lag. See Step 6 for the expected retry behaviour.

Step 4: Add host to inventory

On **pbr-ansible-kl1**, edit `~/pbr-infra/inventory/hosts.yml`. The host must be added in **two places**:

1. Under `all.children.linux.hosts` (with `ansible_host: <IP>`)
2. Under `all.children.targets.hosts` (no `ansible_host` — inherited)

```
---
all:
  children:
    linux:
      hosts:
        # ... existing hosts ...
        pbr-NEWHOST-kl1:
          ansible_host: 10.1.X.Y          # &lt;-- add here

    targets:
      hosts:
        # ... existing hosts ...
        pbr-NEWHOST-kl1:                  # &lt;-- and here
```

Why two places: The `linux` group lists known hosts (used for ad-hoc commands, monitoring, fact-gathering). The `targets` group is the deployment scope — playbooks use `hosts: targets` to ensure the control node and any informational-only hosts cannot be hit accidentally.

Commit and push the inventory change:

```
cd ~/pbr-infra
git add inventory/hosts.yml
git commit -m "inventory: add pbr-NEWHOST-kl1"
git push origin main
```

Step 5: Run preflight (no-changes verification)

```
cd ~/pbr-infra
ansible-playbook playbooks/preflight.yml -l pbr-NEWHOST-kl1 \
    --vault-password-file ~/.ansible_vault_pass
```

Preflight is read-only — it makes zero changes to the host. It validates:

- OS is Ubuntu 22.04 or 24.04
- Hostname is set to a real value and resolves
- System clock is NTP-synchronised
- Required local users (`ansible`, `pbr_admin`) exist
- APT Universe component is enabled (for `oddjob`, `oddjob-mkhomedir`)
- `visudo -c` passes (ignoring the known ThreatLocker drop-in permission issue)
- AD DCs are reachable on TCP 88 and 389
- No existing realm membership conflicts
- SCEPman `/ca` endpoint returns a valid CA cert
- AD schema has the `sshPublicKey` attribute
- Vault password file exists with correct permissions
- Required collections are installed on the control node

If preflight fails, fix the cause and re-run. Do not proceed to the baseline step until preflight is clean.

Step 6: Run the baseline role

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

The playbook runs preflight again (defence in depth) then applies the role. Expected duration: ~3-5 minutes per host on a typical KVM VM.

Expected behaviour: realm join may fail on first attempt

Despite a clean AD pre-clean, the first `realm join` attempt sometimes fails. This is a known pattern caused by AD multi-master replication lag — the join hits a DC that hasn't yet seen the deletion of the pre-cleaned computer object. The output looks like this (with `no_log: true` hiding the actual error):

```
TASK [ssh-baseline : Join Active Directory domain] ****************************
fatal: [pbr-NEWHOST-kl1]: FAILED! =&gt; changed=true
  censored: 'the output has been hidden due to the fact that no_log: true was specified for
this result'
```

Fix: Just re-run the playbook. The role is idempotent and the second attempt almost always succeeds:

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

If the second attempt also fails, dig deeper (see Troubleshooting in the **Known Limitations** page). The most common diagnostic is to read the host's `journalctl` for adcli/realmd/Kerberos errors:

```
ansible pbr-NEWHOST-kl1 -m shell -a '
    journalctl --since "10 minutes ago" --no-pager 2&gt;&amp;1 \
        | grep -iE "realm|adcli|krb5|sssd|kerberos" | tail -40
    timedatectl status | head -8
' --become --vault-password-file ~/.ansible_vault_pass
```

Step 7: Run post-deployment verification

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

Replace `a.mfraser` with any AD username that is a member of `SG_ServerAccess` or `SG_Sudo` and has an `sshPublicKey` populated.

Verify checks:

- Realm membership reports the correct domain
- The test AD user resolves via SSSD (`getent passwd`)
- The test user's SSH public key is retrievable via `sss_ssh_authorizedkeys`
- `sshd -t` passes (full config validates)
- Services `ssh`, `sssd`, `fail2ban` are running

- `auditd` is running on managed hosts (skipped on LXC)
- fail2ban sshd jail is active
- `pam_duo.so` is referenced in `/etc/pam.d/sudo`
- The sudo timestamp_timeout drop-in exists
- The ansible NOPASSWD sudo path still works (proves PAM stack didn't break automation)
- `pbr_admin` is not in `sg_sudo` (would force Duo on break-glass account)

The verification summary at the end looks like:

```
TASK [Verification summary] ****************************************************
ok: [pbr-NEWHOST-kl1] =&gt;
  msg:
  - '=================== VERIFICATION PASSED ====================='
  - 'Joined to realm:        pbr.org.au'
  - 'AD user resolves:       a.mfraser (1234:5678)'
  - 'SSH key retrieved:      ssh-ed25519 AAAAC3...'
  - 'sshd config valid:      yes'
  - 'All services running:   ssh, sssd, fail2ban, auditd'
  - ''
  - 'Next: SSH from your workstation as a.mfraser@pbr-NEWHOST-kl1'
  - 'Expect: key auth + Duo push for SSH; Duo push + AD password for sudo'
```

Step 8: Manual SSH validation from your workstation

This step proves the end-user experience actually works. From your workstation:

Test 1: AD user via SSH

```
ssh a.mfraser@pbr-NEWHOST-kl1.pbr.org.au
```

Expected: SSH key auth completes (no password prompt), then a Duo push to your phone. Approve the push, you land in a shell as your AD user.

Test 2: sudo as AD user

```
sudo whoami
```

Expected: Duo push prompt (auto-pushed), then AD password prompt, then `root`. Within the 30-minute timestamp window, subsequent `sudo` commands skip both prompts.

Test 3: pbr_admin break-glass

```
ssh pbr_admin@pbr-NEWHOST-kl1.pbr.org.au
```

Expected: Password-only prompt (no key, no Duo) — local password from 1Password.

```
sudo whoami
```

Expected: Local password prompt only (no Duo). Returns `root`.

Test 4: Ansible NOPASSWD path still works

From the control node (already validated by verify.yml but worth a manual check):

```
ansible pbr-NEWHOST-kl1 -m shell -a 'sudo -n true' --become
```

Expected: Success. Confirms PAM stack hasn't broken automation.

Step 9: Clean up tee'd log files (if any)

If you piped playbook output to a log file during deployment:

```
# Check whether any log contains the AD service account password
grep -l "MDT_JD\|--login-user" /tmp/*.log 2&gt;/dev/null

# Shred any logs created during this deployment
shred -u /tmp/NEWHOST-*.log 2&gt;/dev/null
```

Even with `no_log: true` restored, transient diagnostic logs from troubleshooting may contain sensitive material. Always scrub.

Royal TS Connection Notes

Royal TS 7's Rebex SSH library has a constraint: it does not support OpenSSH's `AuthenticationMethods publickey,keyboard-interactive` directive natively. Without configuration, Royal TS will fail to connect to baselined hosts.

Workaround: set Authentication Method to "Any"

1. Open the host's Royal TS connection properties
2. Navigate to **Advanced** > **Security**
3. Set **Authentication method** to `Any`
4. Save and reconnect

This lets Rebex negotiate either method per the server's policy, and the server's `AuthenticationMethods` directive will require both.

Auto-push approval

Royal TS's keyboard-interactive UI does not support pre-filling the Duo response. You will press Enter once at the Duo prompt to confirm the push. This is acceptable for a single round-trip MFA.

Alternative: External Application launching Windows OpenSSH

If Rebex limitations bite, configure Royal TS to launch Windows' native `ssh.exe` as an External Application connection instead. PowerShell `ssh.exe` handles `AuthenticationMethods publickey,keyboard-interactive` correctly and integrates with the 1Password SSH agent via the OpenSSH named pipe (`\\.\pipe\openssh-ssh-agent`).

Where to Read Next

- **Known Limitations, Troubleshooting & Version History** — detailed troubleshooting if deployment fails
- **Configuration Reference** — per-host overrides via `host_vars/` if a host needs non-default settings
- **Playbook Reference** — details on preflight, verify, and teardown

Configuration Reference

Variable Source Hierarchy

Variables resolve in standard Ansible precedence order. The role uses three layers:

1. **Role defaults** — `roles/ssh-baseline/defaults/main.yml` (lowest precedence; the safe baseline)
2. **Group vars** — `inventory/group_vars/all/main.yml` (organisation-wide overrides, including vault-sourced secrets)
3. **Host vars** — `inventory/host_vars/<hostname>.yml` (per-host overrides; not currently used in this repo but supported)

The `group_vars/all/main.yml` file overrides the most security-sensitive defaults (AD domain, OUs, groups, SCEPman URL) so they cannot drift even if a role default is accidentally edited.

Group Vars (Organisation-Wide)

File: `inventory/group_vars/all/main.yml`

```
---
# AD join credentials - sourced from vault.yml (encrypted)
ad_join_user: "{{ vault_ad_join_user }}"
ad_join_password: "{{ vault_ad_join_password }}"

# Domain configuration
ad_domain: "pbr.org.au"
ad_computer_ou: "OU=Linux,OU=Servers,OU=Computers,OU=PBR,DC=pbr,DC=org,DC=au"

# Access control via AD security groups (must exist in AD)
ad_server_access_group: "SG_ServerAccess"
ad_sudo_group: "SG_Sudo"

# SCEPman PKI - root CA distribution endpoint
```

```
scepman_ca_url: "https://pki.pbr.org.au/ca"
```

Vault-Sourced Variables

Group var	Vault key	Purpose
`ad_join_user`	`vault_ad_join_user`	UPN of the AD service account used by `realm join`. Must have create-computer rights in the target OU.
`ad_join_password`	`vault_ad_join_password`	Password for the join service account.

The Duo credentials are also vault-sourced and referenced in `roles/ssh-baseline/templates/pam_duo.conf.j2`:

Template var	Vault key	Purpose
`duo_ikey`	`vault_duo_ikey`	Duo Auth API integration key
`duo_skey`	`vault_duo_skey`	Duo Auth API secret key
`duo_api_host`	`vault_duo_api_host`	Duo API hostname (e.g. `api-XXXXXXXX.duosecurity.com`)

To edit the vault:

```
cd ~/pbr-infra
ansible-vault edit inventory/group_vars/all/vault.yml \
    --vault-password-file ~/.ansible_vault_pass
```

Role Defaults: AD & Access

File: `roles/ssh-baseline/defaults/main.yml` (referenced; group_vars override these)

Variable	Default	Purpose
`ad_domain`	`pbr.org.au`	AD DNS domain. Used for realm membership, krb5.conf, SSSD.
`ad_computer_ou`	Linux servers OU	OU where computer objects are created by `realm join`.
`ad_server_access_group`	`SG_ServerAccess`	AD security group for read-only SSH access (no sudo).
`ad_sudo_group`	`SG_Sudo`	AD security group for sudo-enabled users. Members trigger Duo on sudo.

Variable	Default	Purpose
`pbr_admin_allowed_sources`	`10.0.0.0/8,192.168.0.0/16`	Source-IP allow-list (CIDR, comma-separated, no spaces) for the `pbr_admin` break-glass `Match` block.
`ad_access_filter`	*See below*	LDAP filter applied by SSSD for access control. Default is `memberOf=<ServerAccess DN>` OR `memberOf=<Sudo DN>`, both fully qualified.

`ad_access_filter` default:

```
(|(memberOf=CN=SG_ServerAccess,OU=Security,OU=Groups,DC=pbr,DC=org,DC=au)(memberOf=CN=SG_Sudo,
OU=Security,OU=Groups,DC=pbr,DC=org,DC=au))
```

Role Defaults: PKI (SCEPman)

Variable	Default	Purpose
`scepman_ca_url`	`https://pki.pbr.org.au/ca`	Endpoint that returns the SCEPman root CA in DER format.
`scepman_ca_cert`	`/usr/local/share/ca-certificates/pbr-root-ca.crt`	PEM-format location of the trusted root CA (added to system trust store).
`scepman_ca_der`	`/etc/ssl/certs/pbr-root-ca.der`	DER-format location of the root CA (kept for reference; PEM is what's trusted).

Role Defaults: System

Variable	Default	Purpose
`timezone`	`Australia/Melbourne`	System timezone applied via `community.general.timezone`.
`manage_auditd`	`auto`	Whether to enable auditd. `auto` = skip on LXC (kernel audit netlink isolated), enable elsewhere. Accepts `true`, `false`, or `auto`.

Role Defaults: SSH Hardening

These map directly to `sshd_config` directives in `10-pbr-hardening.conf`.

Variable	Default	sshd_config directive	Notes
`ssh_port`	`22`	`Port`	Changing this requires systemd ssh.socket overrides on Ubuntu 22.10+.
`ssh_banner`	`/etc/issue.net`	`Banner`	Path to legal banner file.
`ssh_log_level`	`VERBOSE`	`LogLevel`	CIS Ubuntu 22.04 recommendation.
`ssh_login_grace_time`	`60`	`LoginGraceTime`	Seconds before unauthenticated connection drops.
`ssh_max_auth_tries`	`3`	`MaxAuthTries`	Per-connection auth attempt cap.
`ssh_max_sessions`	`4`	`MaxSessions`	Concurrent multiplexed sessions per connection.
`ssh_max_startups`	`10:30:60`	`MaxStartups`	Concurrent unauthenticated connections (start:rate:full).
`ssh_client_alive_interval`	`300`	`ClientAliveInterval`	Seconds between keepalive probes.
`ssh_client_alive_count_max`	`2`	`ClientAliveCountMax`	Idle connections drop after `interval × count_max` seconds.
`ssh_permit_root_login`	`no`	`PermitRootLogin`	Hard no.
`ssh_password_authentication`	`no`	`PasswordAuthentication`	Disabled globally; re-enabled for `pbr_admin` via `Match` block.
`ssh_pubkey_authentication`	`yes`	`PubkeyAuthentication`	Required by all flows.
`ssh_kbdint`	`yes`	`KbdInteractiveAuthentication`	Required for Duo PAM keyboard-interactive.
`ssh_allow_tcp_forwarding`	`no`	`AllowTcpForwarding`	Disabled.
`ssh_x11_forwarding`	`no`	`X11Forwarding`	Disabled.
`ssh_allow_agent_forwarding`	`no`	`AllowAgentForwarding`	Disabled.
`ssh_compression`	`no`	`Compression`	Defence against compression-side-channel attacks.

Variable	Default	sshd_config directive	Notes
`ssh_tcp_keep_alive`	`no`	`TCPKeepAlive`	Use SSH-level keep-alive instead.
`ssh_authentication_methods`	`publickey,keyboard-interactive`	`AuthenticationMethods`	Both required; keyboard-interactive is Duo via PAM.

Modern Crypto

Algorithm lists prepended with the post-quantum hybrid KEX where available:

Variable	Default
`ssh_ciphers`	`chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr`
`ssh_macs`	`hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com`
`ssh_kex_algorithms`	`sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256`

Role Defaults: fail2ban

Variable	Default	Purpose
`fail2ban_bantime_default`	`3600`	Default ban duration in seconds (1 hour) for non-sshd jails.
`fail2ban_findtime`	`600`	Window in seconds during which `maxretry` failures trigger a ban.
`fail2ban_maxretry_default`	`5`	Failures within `findtime` before ban (default for non-sshd jails).
`fail2ban_sshd_maxretry`	`3`	Tighter setting for the sshd jail.
`fail2ban_sshd_bantime`	`86400`	24-hour ban for sshd failures.
`fail2ban_ignoreip`	*list, see below*	CIDRs exempt from banning.

Default `fail2ban_ignoreip`:

```
fail2ban_ignoreip:
  - "127.0.0.1/8"
  - "::1"
```

```
  - "10.1.0.0/16"        # PBR server LAN
  - "10.1.8.80/32"       # pbr-ansible-kl1 control node (explicit)
  - "192.168.0.0/16"     # Admin workstation VLANs supernet (TEMPORARY)
```

The 192.168.0.0/16 entry is annotated **TEMPORARY** in the role — intended to be removed when VLAN segmentation completes and admin workstations land on a single, well-defined CIDR.

Role Defaults: Duo MFA

Variable	Default	Purpose
`duo_failmode`	`safe`	`safe` = allow login if Duo cloud unreachable (fall through to single-factor publickey); `secure` = deny login during outage.
`duo_pushinfo`	`yes`	Include hostname and command in the Duo push notification.
`duo_prompts`	`3`	Max retries at the Duo prompt before failure.
`duo_autopush`	`yes`	Auto-send push to user's primary device.
`break_glass_user`	`pbr_admin`	Username carved out of the Duo PAM flow.
`duo_sudo_enabled`	`true`	Toggle Duo MFA on sudo (v2.4+).
`sudo_timestamp_timeout`	`30`	Minutes the sudo credential cache lasts; reduces Duo prompts during a session.

Preflight Role Defaults

File: `roles/preflight/defaults/main.yml`

Variable	Default	Purpose
`preflight_min_ubuntu_major`	`22`	Minimum Ubuntu major version. 22.04 LTS is the floor.
`preflight_required_users`	`[ansible, pbr_admin]`	Local accounts that must exist before baseline.

Variable	Default	Purpose
`preflight_ad_ports`	`[88, 389]`	Ports tested for AD DC reachability. 88 = Kerberos, 389 = LDAP.
`preflight_skip_schema_check`	`false`	Set true to bypass the AD schema check if `python3-ldap` is unavailable on the control node and you've verified schema manually.

Override Patterns

Per-host override via host_vars

Create `inventory/host_vars/<hostname>.yml`. Example: a host that requires a tighter source-IP allow-list:

```
---
# inventory/host_vars/pbr-pos-belgrave.yml
pbr_admin_allowed_sources: "10.1.8.0/24"  # POS LAN only
fail2ban_sshd_bantime: 604800              # 7 days for POS hosts
```

Forcing auditd on/off per host

```
---
# inventory/host_vars/pbr-graylog-kl1.yml
# Force-skip auditd even if the host migrates from LXC to KVM
manage_auditd: false
```

Adding a CIDR to fail2ban ignoreip

Override the full list (Ansible doesn't merge list defaults by default):

```
fail2ban_ignoreip:
  - "127.0.0.1/8"
  - "::1"
  - "10.1.0.0/16"
  - "10.1.8.80/32"
```

```
  - "192.168.0.0/16"
  - "203.0.113.42/32"   # NEW: external admin static IP
```

---

ansible.cfg Settings

The runtime configuration on `pbr-ansible-kl1` is fixed by `ansible.cfg` in the repo root:

```
[defaults]
inventory           = inventory/hosts.yml
remote_user         = ansible
private_key_file    = ~/.ssh/ansible_svc
host_key_checking   = True
retry_files_enabled = False
stdout_callback     = yaml
interpreter_python  = auto_silent
vault_password_file = ~/.ansible_vault_pass
roles_path          = roles
collections_path    = collections
forks               = 5

[privilege_escalation]
become              = True
become_method       = sudo
become_user         = root
become_ask_pass     = False

[ssh_connection]
pipelining          = True
ssh_args            = -o ControlMaster=auto -o ControlPersist=60s
```

Notable settings:

- `host_key_checking = True` — rejects connection to hosts with unknown SSH host keys. Adding a new host requires accepting its host key once (the bootstrap step naturally surfaces this).
- `vault_password_file` is set in `ansible.cfg`, so the `--vault-password-file` flag is technically redundant on the command line. It's included explicitly in this book's runbooks for portability if the config changes.

- `forks = 5` caps concurrency. Combined with `serial: 1` in playbooks, the effective concurrency is 1 host at a time.
- `pipelining = True` reduces task overhead by skipping the SCP/SFTP transfer of small modules.

Collection Requirements

File: `requirements.yml`

```
---
collections:
  - name: ansible.posix
    version: ">=2.1.0"
  - name: community.general
    version: ">=12.0.0"
  - name: paloaltonetworks.panos
    version: ">=2.20"
  - name: arubanetworks.aoscx
    version: ">=10.0"
```

Used by ssh-baseline: `ansible.posix` (assorted modules), `community.general` (`timezone` module, `ldap_search` for schema check).

Other collections: `paloaltonetworks.panos` and `arubanetworks.aoscx` are listed for future use cases (Palo Alto NGFW automation, AOS-CX switch config) but are not used by the ssh-baseline role.

Install/update collections:

```
cd ~/pbr-infra
ansible-galaxy collection install -r requirements.yml --upgrade
```

Where to Read Next

- **AD Integration & SSSD** — how the AD variables map to SSSD config
- **Duo MFA Integration** — how the Duo variables map to `pam_duo.conf` and the PAM stacks
- **SSH Hardening Reference** — how each SSH variable lands in the deployed config

AD Integration & SSSD

Overview

The role integrates Ubuntu hosts with Active Directory via SSSD using `realm join`. Once joined, AD users authenticate via Kerberos (with their AD password), are authorised via AD group membership, and have their SSH public keys retrieved from the `sshPublicKey` attribute.

This page documents the integration's moving parts: krb5.conf, SSSD config, realm membership, schema requirements, and the access-control filter.

Realm Join Flow

From `roles/ssh-baseline/tasks/ad-join.yml`:

1. **Verify AD domain is resolvable** — `getent hosts pbr.org.au` returns at least one DC IP.
2. **Configure `/etc/krb5.conf`** — from the `krb5.conf.j2` template (minimal, SRV-discovery based).
3. **Check current AD join status** — `realm list --name-only`. If the host is already joined, the join task is skipped.
4. **Join AD** — `realm join --user=<svc account> --computer-ou=<OU> --os-name="Ubuntu Server" --os-version=<detected> <domain>`. Password is supplied via stdin from the vault. Task has `no_log: true`.
5. **Verify Kerberos keytab** exists at `/etc/krb5.keytab`.
6. **Configure realm access** — `realm deny --all`, then `realm permit --groups <ServerAccess> <Sudo>`. This is the realmd layer of the group gate (defence-in-depth alongside SSSD's `ad_access_filter` and sshd's `AllowGroups`).
7. **Enable SSS and mkhomedir PAM profiles** — `pam-auth-update --enable sss --enable mkhomedir`.
8. **Verify `pam_sss` in common-auth** with correct flow control (sanity check — if pam-auth-update silently failed, we catch it).
9. **Deploy `/etc/sssd/sssd.conf`** — from the `sssd.conf.j2` template.
10. **Validate SSSD config** — `sssctl config-check`.
11. **Enable and start SSSD**.

---

krb5.conf Template

Source: `roles/ssh-baseline/templates/krb5.conf.j2`

```
# Managed by Ansible - do not edit manually
# Minimal Kerberos client config; KDC/realm discovery via DNS SRV records.
# SSSD writes dynamic snippets under /var/lib/sss/pubconf/krb5.include.d/

includedir /var/lib/sss/pubconf/krb5.include.d/

[libdefaults]
default_realm = {{ ad_domain | upper }}
rdns = false
dns_lookup_realm = false
dns_lookup_kdc = true
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
udp_preference_limit = 0
```

Notable settings

Setting	Value	Why
`includedir`	`/var/lib/sss/pubconf/krb5.include.d/`	SSSD writes dynamic snippets here (realm mappings, KDC lists). Including this directory lets SSSD update krb5 behaviour without touching our template.
`rdns`	`false`	Don't reverse-resolve hostnames into principal names. Avoids principal-mismatch errors when reverse DNS is incomplete.
`dns_lookup_realm`	`false`	The realm is fixed (we know it's `PBR.ORG.AU`). Don't waste time on DNS lookups for the realm itself.
`dns_lookup_kdc`	`true`	Use SRV records to find KDCs. PBR has 4 DCs; SRV-based discovery is more resilient than static KDC lists.

Setting	Value	Why
`udp_preference_limit`	`0`	Always use TCP. UDP is unreliable for Kerberos tickets that exceed the default UDP packet size (large PAC for users in many groups).
`ticket_lifetime`	`24h`	How long a TGT is valid before requiring re-auth. Default for AD-integrated Linux.
`renew_lifetime`	`7d`	How long a TGT can be renewed before requiring full re-auth.

SSSD Configuration

Source: `roles/ssh-baseline/templates/sssd.conf.j2` — rendered with the variables from `defaults/main.yml` and `group_vars/all/main.yml`.

```
[sssd]
# Explicit services list (alternative to systemd socket activation).
# Includes ssh responder so sss_ssh_authorizedkeys works for sshd.
services = nss, pam, ssh
domains = {{ ad_domain }}
config_file_version = 2

[domain/{{ ad_domain }}]
id_provider = ad
access_provider = ad
ad_domain = {{ ad_domain }}
krb5_realm = {{ ad_domain | upper }}
krb5_store_password_if_offline = True
cache_credentials = True
default_shell = /bin/bash
override_homedir = /home/%u
use_fully_qualified_names = False
ldap_id_mapping = True
realmd_tags = manages-system joined-with-adcli

# Disable GPO-based access control.
ad_gpo_access_control = disabled
```

```
ad_access_filter = {{ ad_access_filter }}

# Retrieve SSH public keys from AD via the sshPublicKey attribute
# (OpenSSH-LPK schema extension applied via openssh-lpk.ldif).
ldap_user_extra_attrs = sshPublicKey
ldap_user_ssh_public_key = sshPublicKey
```

Service responders

`services = nss, pam, ssh` — SSSD runs three responder daemons:

- **nss** — serves user/group name resolution. `getent passwd a.mfraser` hits this.
- **pam** — handles PAM authentication. `pam_sss.so` talks to it.
- **ssh** — serves SSH public key lookups for `/usr/bin/sss_ssh_authorizedkeys`. Without this, sshd cannot retrieve keys from AD.

The explicit list is the alternative to systemd socket activation. Both work, but explicit listing makes the service set inspectable and removes a layer of indirection during troubleshooting.

Identity & access providers

Setting	Value	Purpose
`id_provider`	`ad`	Identity lookups go to AD via LDAP.
`access_provider`	`ad`	Access decisions go to AD — we use `ad_access_filter`.
`ad_domain` / `krb5_realm`	Per group_vars	Define the AD domain and Kerberos realm.
`krb5_store_password_if_offline`	`True`	Cache the user's Kerberos password if SSSD is offline. Enables offline login.
`cache_credentials`	`True`	Cache user credentials. Required for offline auth.
`default_shell`	`/bin/bash`	Default shell when AD doesn't supply one.
`override_homedir`	`/home/%u`	Force homedir to `/home/<username>` regardless of what AD has.
`use_fully_qualified_names`	`False`	Users are referenced as `a.mfraser`, not `a.mfraser@pbr.org.au`.
`ldap_id_mapping`	`True`	Generate POSIX UIDs/GIDs algorithmically from AD SIDs. No POSIX attributes in AD required.

Setting	Value	Purpose
`realmd_tags`	`manages-system joined-with-adcli`	Standard tags written by `realm join` — preserved by Ansible to avoid realmd discarding our config.

ad_gpo_access_control = disabled

This is the single most consequential SSSD setting in the file. Inline comment in the template:

> Per `sssd-ad(5)`, the default is `enforcing`, which evaluates Windows GPO `RemoteInteractiveLogonRight` settings on every SSH login. Any GPO at any parent OU that sets this right (intentionally for Windows servers, or inherited from an ancestor container) would silently deny SSH access. We use `ad_access_filter` as the sole access control scheme; the sssd-ad(5) manpage explicitly directs disabling GPO control when doing so.

This is documented behaviour, not a workaround. The default exists to make SSSD respect Windows server access policy when AD admins want it; for Linux servers managed independently, disabling it is the canonical approach.

ad_access_filter

The filter is supplied from `defaults/main.yml`:

```
ad_access_filter: >-
  (|(memberOf=CN={{ ad_server_access_group
}},OU=Security,OU=Groups,DC=pbr,DC=org,DC=au)(memberOf=CN={{ ad_sudo_group
}},OU=Security,OU=Groups,DC=pbr,DC=org,DC=au))
```

Rendered:

```
(|(memberOf=CN=SG_ServerAccess,OU=Security,OU=Groups,DC=pbr,DC=org,DC=au)(memberOf=CN=SG_Sudo,
OU=Security,OU=Groups,DC=pbr,DC=org,DC=au))
```

The filter uses full DN references because it makes the match unambiguous regardless of LDAP search base. If two groups with the same name existed in different OUs, a name-only filter could match the wrong one.

If the security groups move OUs, `defaults/main.yml` must be updated.

SSH public key retrieval

The bottom two lines of the SSSD config are the magic:

```
ldap_user_extra_attrs = sshPublicKey
ldap_user_ssh_public_key = sshPublicKey
```

`ldap_user_extra_attrs` tells SSSD to fetch the `sshPublicKey` attribute alongside the standard user attributes during user lookups. `ldap_user_ssh_public_key` tells the SSH responder to expose that attribute via `sss_ssh_authorizedkeys`.

sshd is configured to call `/usr/bin/sss_ssh_authorizedkeys %u` as the user `nobody` (see **SSH Hardening Reference**). The flow:

1. User connects to sshd with publickey auth, presenting their public key
2. sshd invokes `sss_ssh_authorizedkeys a.mfraser` as `nobody`
3. `sss_ssh_authorizedkeys` asks the SSSD ssh responder for the user's keys
4. The SSSD ssh responder queries AD via LDAP for the `sshPublicKey` attribute on the user object
5. The keys are returned to sshd, which compares against the presented public key
6. If a match, publickey auth succeeds — sshd then proceeds to the keyboard-interactive challenge (Duo)

AD Schema Requirements

sshPublicKey attribute

AD does not include the `sshPublicKey` attribute in its default schema. It must be added via the OpenSSH-LPK schema extension before the role can work.

The schema is applied once, against the AD Schema Master, using an LDIF file (`openssh-lpk.ldif`). PBR has applied this; preflight verifies it remains present:

```
# From roles/preflight/tasks/schema.yml
- name: Check sshPublicKey attribute exists in AD schema
  community.general.ldap_search:
    server_uri: "ldaps://{{ ad_domain }}"
    bind_dn: "{{ ad_join_user }}"
    bind_pw: "{{ ad_join_password }}"
```

```
    dn: "CN=Schema,CN=Configuration,DC={{ ad_domain | replace('.', ',DC=') }}"
    scope: onelevel
    filter: "(cn=sshPublicKey)"
    attrs:
      - cn
      - attributeID
  register: schema_check
  delegate_to: localhost
  become: false
  run_once: true
  no_log: true
```

If the schema check fails, preflight aborts with:

```
sshPublicKey attribute not found in AD schema at pbr.org.au.
Apply openssh-lpk.ldif against the Schema Master before continuing.
```

Populating sshPublicKey on user objects

End users have their SSH public key populated on their AD user object. This is done manually or via a self-service script — not by this role. The attribute is multi-valued; a user can have multiple keys.

To set programmatically (PowerShell, on a domain-joined Windows host):

```
Set-ADUser a.mfraser -Replace @{
    sshPublicKey = "ssh-ed25519 AAAA... user@workstation"
}
```

Service Account: ad_join_user

The role uses an AD service account stored in vault as `vault_ad_join_user` / `vault_ad_join_password`. Required AD permissions:

- **Create computer objects** in the target OU (`OU=Linux,OU=Servers,OU=Computers,OU=PBR,DC=pbr,DC=org,DC=au`)
- **Read access** to the Schema container (used by the preflight schema check)

It does **not** need Domain Admin rights. Best practice: a dedicated service account with delegated rights only.

The account password is rotated via a separate process (not by this role) and the vault updated via `ansible-vault edit`.

Realm Permit (realmd-layer Access Control)

After joining, the role runs:

```
realm deny --all
realm permit --groups SG_ServerAccess
realm permit --groups SG_Sudo
```

This adds entries to `/etc/sssd/sssd.conf` under `simple_allow_groups`. However, because we set `access_provider = ad` and use `ad_access_filter` instead, `simple_allow_groups` is not the effective gate — the AD access filter is.

The realmd commands are kept for two reasons:

1. **realmd-managed metadata.** `realm list` reflects what realmd thinks the access policy is. Keeping it consistent with the SSSD config avoids confusion when troubleshooting.
2. **Defence in depth.** If `access_provider` were ever changed to `simple`, `simple_allow_groups` becomes the gate, and the realmd-issued permits keep enforcement consistent.

PAM Wiring (Authentication Side)

The role enables the SSS and mkhomedir profiles via `pam-auth-update`:

```
pam-auth-update --enable sss --enable mkhomedir
```

This modifies the Ubuntu-managed `common-auth` / `common-account` / `common-password` / `common-session` stacks to include `pam_sss.so` and `pam_mkhomedir.so` (or equivalent).

The role then verifies the result is what we expected:

```
- name: Verify pam_sss is in common-auth with correct flow control
  ansible.builtin.shell: |
```

```
    set -o pipefail
    grep -E '^auth\s+\[success=1 default=ignore\]\s+pam_sss' /etc/pam.d/common-auth
```

This sanity check catches the (rare) case where `pam-auth-update` succeeds at the exit code level but doesn't actually add what we need.

How the Duo PAM stacks consume this: `/etc/pam.d/sshd` and `/etc/pam.d/sudo` are custom files (templated by the role). The sudo stack uses `@include common-auth` after Duo, which lets `pam_sss` validate the AD password as the post-Duo factor. See **Duo MFA Integration** for the full flow.

Troubleshooting AD/SSSD

User doesn't resolve via getent

```
getent passwd a.mfraser
# (no output)
```

Causes:

- User not in `SG_ServerAccess` or `SG_Sudo` (access filter excludes them — SSSD won't surface them via NSS)
- SSSD service not running — `systemctl status sssd`
- Stale SSSD cache — `sudo sss_cache -E` to invalidate
- LDAP connectivity to DCs broken — `sssctl domain-status pbr.org.au` shows ONLINE / OFFLINE

SSH key not found

```
sudo -u nobody /usr/bin/sss_ssh_authorizedkeys a.mfraser
# (no output or error)
```

Causes:

- `sshPublicKey` attribute not populated on the user's AD object — check in ADUC
- SSS ssh responder not running — `services = nss, pam, ssh` in sssd.conf? Restart SSSD.
- SSSD service account can't read user attributes — LDAP bind ACL issue (not the join account; SSSD uses the host's keytab)

sssctl config-check fails

This is caught by the role itself — the deploy halts if SSSD config doesn't validate. Inspect output:

```
sudo sssctl config-check
```

Usually a typo in `ad_access_filter` after a manual edit. Re-run the role to restore the template.

Where to Read Next

- **Duo MFA Integration** — how PAM connects AD authentication with Duo MFA
- **SSH Hardening Reference** — `AuthorizedKeysCommand` and the sshd-side of key retrieval
- **Known Limitations, Troubleshooting & Version History** — the realm join retry pattern caused by AD replication lag

Duo MFA Integration

Scope

Duo MFA is enforced in two places:

1. **SSH login** (v2.3+) — via PAM keyboard-interactive after publickey auth
2. **sudo** (v2.4+) — via PAM at the auth phase, with AD password as the post-Duo factor

The role uses Duo Security's official `duo-unix` package, not Ubuntu universe's `libpam-duo` (which is outdated and has Duo API incompatibilities).

Package Installation

Source: `roles/ssh-baseline/tasks/duo.yml`. The flow:

1. Download Duo's GPG signing key from `https://duo.com/DUO-GPG-PUBLIC-KEY.asc`
2. Convert to a dearmored keyring at `/etc/apt/trusted.gpg.d/duo.gpg`
3. Add APT repository: `deb [arch=amd64] https://pkg.duosecurity.com/Ubuntu {{ ansible_distribution_release }} main`
4. Purge any legacy `libpam-duo` / `libduo3` from Ubuntu universe
5. Install `duo-unix` package

Inline comment from the role explaining why we don't use Ubuntu universe:

> 1. Ubuntu universe ships 1.11.3 (2022) which has incompatibilities with current Duo Auth API and returns HTTP 403 in some scenarios.
> 2. Duo's 2.1.0+ is required for the April 2026 CA bundle rotation.
> 3. Duo's docs explicitly target the duo-unix package on Ubuntu 22.04.

The package installs `pam_duo.so` at `/usr/lib64/security/` — not in Ubuntu's default PAM module search path. Both PAM stack templates reference the module by absolute path for this reason.

Duo PAM Configuration File

Template: `roles/ssh-baseline/templates/pam_duo.conf.j2`. Deployed to `/etc/duo/pam_duo.conf` with mode 0600 (contains `skey`). The task that writes it has `no_log: true`.

```
# Managed by Ansible - PBR ssh-baseline role
# Source: roles/ssh-baseline/templates/pam_duo.conf.j2
#
# pam_duo.conf - configuration for Duo Security PAM module
# Permissions MUST be 0600 owned by root (contains skey).

[duo]
ikey = {{ duo_ikey }}
skey = {{ duo_skey }}
host = {{ duo_api_host }}

# failmode controls behaviour when Duo cloud is unreachable:
#   safe   = allow login (single-factor publickey fallback)
#   secure = deny login (locks out during Duo outage)
failmode = {{ duo_failmode }}

# Include hostname + command in push notification
pushinfo = {{ duo_pushinfo }}

# Max retries at the Duo prompt
prompts = {{ duo_prompts }}

# Auto-push to user's primary device (true) vs prompt for factor (false)
autopush = {{ duo_autopush }}

# Restrict Duo to AD server-access group members.
# Users not in this group (e.g. {{ break_glass_user }} break-glass) bypass Duo automatically.
groups = {{ ad_server_access_group | lower }},{{ ad_sudo_group | lower }}
```

The `groups` directive is the key Duo-level filter: `pam_duo.so` only challenges users in the listed groups. Local accounts (`pbr_admin`, `ansible`) are not in those groups, so they bypass Duo entirely — even before our pam_succeed_if carve-outs fire.

Group names are lowercased because SSSD normalises AD group names to lowercase when surfacing them via NSS.

SSH PAM Stack (pam_sshd.j2)

Deployed to `/etc/pam.d/sshd`. This is a custom file (not `@include common-auth` at the top) so we can control the order of Duo vs. password validation precisely.

```
# Managed by Ansible - PBR ssh-baseline role
# === Auth section ===
auth    [success=2 default=ignore]  pam_succeed_if.so user = pbr_admin quiet

# AD users: Duo MFA is required, failure terminates the stack
auth    requisite                   /usr/lib64/security/pam_duo.so

# Duo succeeded → exit stack with success (do not fall through to pam_unix)
auth    [success=done default=die]  pam_permit.so

# pbr_admin lands here (jumped past pam_duo + pam_permit)
auth    required                    pam_unix.so try_first_pass nullok_secure

# === Account section ===
account    required     pam_nologin.so
@include common-account

# === Session section ===
session    [success=ok ignore=ignore module_unknown=ignore default=bad]   pam_selinux.so close
session    required     pam_loginuid.so
session    optional     pam_keyinit.so force revoke
@include common-session
session    optional     pam_motd.so motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate
session    optional     pam_mail.so standard noenv
session    required     pam_limits.so
session    required     pam_env.so
session    required     pam_env.so user_readenv=1 envfile=/etc/default/locale
session    [success=ok ignore=ignore module_unknown=ignore default=bad]   pam_selinux.so open

# === Password section ===
@include common-password
```

Auth section dissection

Four lines of auth, each with deliberate control flow. Reading from the top:

Line 1: pbr_admin detection & branching

```
auth    [success=2 default=ignore]  pam_succeed_if.so user = pbr_admin quiet
```

- `pam_succeed_if.so user = pbr_admin` returns success if the authenticating user is `pbr_admin` .
- `success=2` means: on success, skip the next 2 modules (pam_duo and pam_permit).
- `default=ignore` means: for any other return value (the user is NOT pbr_admin), continue to the next module.

Effect: If you're `pbr_admin`, jump straight to the `pam_unix.so` line. If you're not, continue to pam_duo.

Line 2: Duo MFA

```
auth    requisite                   /usr/lib64/security/pam_duo.so
```

- `requisite` means: if this module fails, terminate the auth stack immediately with that failure code. Do not try further modules.
- This is for AD users (who reach this line because Line 1's pam_succeed_if didn't match).
- Inside `pam_duo.so`, the `groups` filter in `pam_duo.conf` applies — if the user is not in `sg_serveraccess` or `sg_sudo`, Duo skips them and returns success without prompting. (In practice, sshd's `AllowGroups` would have rejected them earlier, so this is defence-in-depth.)

Line 3: success exits the stack

```
auth    [success=done default=die]  pam_permit.so
```

- `pam_permit.so` always returns success.
- `success=done` means: terminate the auth stack with overall success. Do not run later auth modules.
- Reached only after pam_duo passes. AD users land here on success and exit the stack cleanly.

Line 4: pbr_admin's destination

```
auth    required                    pam_unix.so try_first_pass nullok_secure
```

- Reached only by `pbr_admin` (who jumped here via Line 1's `success=2`).
- `pam_unix.so` validates the local password against `/etc/shadow`.
- `try_first_pass` uses the password already supplied (sshd passes it via the keyboard-interactive PAM conversation).
- `required` means: failure makes the stack fail, but later modules still run (none in this stack).

The full sshd authentication picture

Putting sshd's `AuthenticationMethods publickey,keyboard-interactive` together with the PAM stack:

User	sshd Step 1: publickey	sshd Step 2: keyboard-interactive (PAM)
AD user (e.g. a.mfraser)	Validates against AD-stored `sshPublicKey` via SSSD	pam_succeed_if doesn't match → pam_duo prompts → success exits stack
pbr_admin	(see below)	pam_succeed_if matches → jump to pam_unix → validates local password
ansible	Local `~/.ssh/authorized_keys`; `AuthenticationMethods publickey` in Match block bypasses keyboard-interactive	Never enters PAM auth

Wait: how does pbr_admin authenticate at all if sshd requires publickey first?

The `Match User pbr_admin Address ...` block in `sshd_hardening.conf.j2` overrides `AuthenticationMethods` for that user to `password` only:

```
Match User pbr_admin Address {{ pbr_admin_allowed_sources }}
    PasswordAuthentication yes
    AuthenticationMethods password
```

So `pbr_admin` enters PAM via password auth (not keyboard-interactive), but the PAM stack handles both flows the same way — `pam_succeed_if` matches and jumps to `pam_unix` which validates the password.

---

sudo PAM Stack (pam_sudo.j2)

Deployed to `/etc/pam.d/sudo`. Replaces the Ubuntu-default file.

```
#%PAM-1.0
# Managed by Ansible - PBR ssh-baseline role v2.4

# Standard Ubuntu sudo session environment setup
session    required   pam_env.so readenv=1 user_readenv=0
session    required   pam_env.so readenv=1 envfile=/etc/default/locale user_readenv=0

# Skip Duo for users not in the AD sudo group (covers ansible, pbr_admin,
# and any local user with sudo rights).
auth       [success=1 default=ignore]  pam_succeed_if.so quiet user notingroup sg_sudo

# Require Duo MFA for AD users in the sudo group.
auth       requisite                   /usr/lib64/security/pam_duo.so

# Validate the user's password (AD via pam_sss for AD users, local via
# pam_unix for break-glass account). NOPASSWD entries in sudoers bypass
# this entire auth phase regardless.
@include common-auth
@include common-account
@include common-session-noninteractive
```

Auth section dissection

Line 1: AD sudo group check

```
auth       [success=1 default=ignore]  pam_succeed_if.so quiet user notingroup sg_sudo
```

- `pam_succeed_if user notingroup sg_sudo` returns success if the user is **not** in `sg_sudo`.
- `success=1` jumps over the next module (pam_duo).
- `default=ignore` continues to pam_duo for users IN `sg_sudo`.

Group name is lowercase because SSSD normalises AD group names. The template uses `{{ ad_sudo_group | lower }}` for safety.

Line 2: Duo for AD sudo users

```
auth       requisite                   /usr/lib64/security/pam_duo.so
```

- Reached only by users in `sg_sudo`.
- `requisite` aborts the stack on Duo failure (denied push, timeout, etc.).

- On success, falls through to common-auth.

Line 3: Password validation

```
@include common-auth
```

- `common-auth` runs `pam_sss.so` for AD users (validates AD password) or `pam_unix.so` for local users.
- NOPASSWD entries in sudoers bypass this entire auth phase — `ansible` sudo never reaches PAM auth at all.

The full sudo authentication picture

User	PAM flow	Effective auth
AD user in sg_sudo	pam_succeed_if doesn't match → pam_duo prompts → common-auth → pam_sss	Duo push + AD password
pbr_admin (NOT in sg_sudo)	pam_succeed_if matches → jump past pam_duo → common-auth → pam_unix	Local password
ansible (NOPASSWD sudoers)	sudoers NOPASSWD bypasses PAM auth entirely	None

---

sudo Credential Cache Extension

The role drops `/etc/sudoers.d/sudo_timestamp_timeout`:

```
# Managed by Ansible - PBR ssh-baseline role v2.4
# Extends sudo credential cache from default 15min to {{ sudo_timestamp_timeout }}min
# to reduce Duo MFA push frequency for AD sudo users without significantly
# weakening the control (session hijack window unchanged).
Defaults timestamp_timeout={{ sudo_timestamp_timeout }}
```

Default value: `sudo_timestamp_timeout: 30` (minutes). Ubuntu's default is 15.

The drop-in is validated with `visudo -cf` before being written. The file is mode 0440 (per sudoers convention).

Why extend: A typical maintenance session involves many sudo invocations. With the default 15-minute cache, an AD user gets repeated Duo pushes. Extending to 30 minutes reduces noise

without meaningfully changing the security envelope — the session-hijack window is per-tty and the underlying authentication is unchanged.

Failure Mode (failmode = safe)

If Duo's cloud is unreachable (DNS broken, Duo outage, firewall change), pam_duo returns success and the stack proceeds. For SSH this means single-factor publickey is sufficient; for sudo, common-auth still requires a password.

The trade-off:

- **With failmode = safe (chosen):** Duo outages don't lock administrators out. Single-factor publickey is still strong — AD-managed keys with revocation in effect.
- **With failmode = secure:** Stronger MFA guarantee but Duo outages cause fleet-wide lockout. `pbr_admin` break-glass would be the only path in.

Chosen: `safe`. PBR has acceptable compensating controls (key-based auth, AD password for sudo, source-IP-restricted break-glass) such that single-factor degradation during a Duo outage is acceptable.

Validation Tasks in the Role

After deploying both PAM stacks and pam_duo.conf, the role runs validation checks to fail fast if something is wrong:

```
- name: Validate Duo module is referenced in sudo PAM stack
  ansible.builtin.command: grep -c "pam_duo.so" /etc/pam.d/sudo
  failed_when: sudo_pam_duo_check.stdout | int &lt; 1

- name: Sanity check - sudo still works for non-Duo automation accounts
  ansible.builtin.command: sudo -n true
  become: false
  # Runs as the ansible_user (ansible). ansible has NOPASSWD in sudoers
  # and is not in sg_sudo, so it should bypass Duo entirely. If this fails,
  # the new PAM stack has broken local sudo - red flag, terminate deploy.

- name: Validate Duo module is referenced in sshd PAM stack
  ansible.builtin.command: grep -E "pam_duo\.so" /etc/pam.d/sshd
```

```
- name: Validate pam_duo.so exists at the absolute path used by PAM stack
  ansible.builtin.stat: path: /usr/lib64/security/pam_duo.so
  failed_when: not pam_duo_stat.stat.exists
```

The sanity sudo check is particularly important: it runs as the `ansible` user (non-Duo automation) and verifies that sudo still works. If the new PAM stack broke local sudo, the deploy halts immediately rather than continuing through subsequent tasks that depend on sudo working.

Compliance Note

From the inline comment in `defaults/main.yml`:

> Duo MFA on sudo (v2.4)
> Essential Eight ML2: MFA for privileged users performing privileged actions.

This is the only Essential Eight reference in the role's source. Broader compliance mappings (VPDSS, VG-CISO) are out of scope for this documentation — refer to PBR's separate compliance documentation if needed.

Troubleshooting Duo

"Permission denied" without a Duo prompt

Most likely the user is not in `SG_ServerAccess` or `SG_Sudo` — sshd's `AllowGroups` rejected them before PAM ran. Verify:

```
ssh -vvv a.mfraser@host.pbr.org.au 2&gt;&amp;1 | grep -i 'permission denied\|allowgroups'
```

Duo prompt arrives but auth fails

Check the host's Duo PAM logs:

```
sudo journalctl -u sshd --since "5 minutes ago" | grep -i duo
```

Common causes: Duo Auth API `ikey`/`skey`/`host` wrong in `/etc/duo/pam_duo.conf` (vault credentials mismatch), system clock drift (Duo requires NTP), user disabled in Duo admin console.

sudo asks for password but never prompts for Duo

Indicates the user is not in `sg_sudo`, so the pam_succeed_if branch skipped pam_duo. Verify:

```
id a.mfraser | tr ',' '\n' | grep -i sg_sudo
```

If empty, either the user isn't in the AD group (intended) or SSSD cache is stale (`sudo sss_cache -E`).

Where to Read Next

- **SSH Hardening Reference** — how sshd's Match blocks interact with the PAM stack
- **AD Integration & SSSD** — how pam_sss validates AD passwords post-Duo
- **Known Limitations, Troubleshooting & Version History** — Royal TS Rebex authentication caveats

SSH Hardening Reference

What This Page Covers

This page walks through every directive in `roles/ssh-baseline/templates/sshd_hardening.conf.j2` and explains how it lands on the target host. The deployed file is `/etc/ssh/sshd_config.d/10-pbr-hardening.conf`.

The hardening is aligned with CIS Ubuntu Linux 22.04 Benchmark v2.0.0. Where we deviate, it's documented inline and below.

How the Config Reaches sshd

Drop-in directory pattern

Ubuntu's `sshd_config` reads drop-in files from `/etc/ssh/sshd_config.d/` via an `Include` directive. Cloud-init images have this by default; some ISO installs do not. The role ensures the include is present:

```
- name: Ensure sshd_config has Include directive for drop-ins
  ansible.builtin.lineinfile:
    path: /etc/ssh/sshd_config
    line: "Include /etc/ssh/sshd_config.d/*.conf"
    insertbefore: BOF
    state: present
    validate: "/usr/sbin/sshd -t -f %s"
  notify: Restart sshd
```

Why insert at BOF (beginning of file): sshd uses first-match-wins semantics for most directives. Placing the Include directive at the top of `sshd_config` means drop-ins are evaluated first — our hardening directives win over any conflicting directive later in the base config.

Filename prefix: 10-

The deployed file is named `10-pbr-hardening.conf`. Drop-ins are loaded in lexicographic order. The `10-` prefix ensures our file loads before Ubuntu's default `50-cloud-init.conf`, which sets `PasswordAuthentication yes`. Without the `10-` prefix and first-match-wins, cloud-init's value could win.

Validation gating

Both the Include line and the hardening file are written with `validate: "/usr/sbin/sshd -t -f %s"`. Ansible writes to a temp file, runs `sshd -t -f <tempfile>` against it, and only moves the temp file into place if validation passes. After the file is in place, the role also runs a final `sshd -t` against the live combined config (defence in depth).

The Hardening File: Full Source

Template: `roles/ssh-baseline/templates/sshd_hardening.conf.j2`. Rendered output (all variables substituted with their defaults):

```
# PBR SSH Hardening - Managed by Ansible, do not edit manually
# CIS Ubuntu Linux 22.04 Benchmark v2.0.0 aligned

Port 22
LogLevel VERBOSE
LoginGraceTime 60

# === Authentication ===
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
KbdInteractiveAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
MaxAuthTries 3
GSSAPIAuthentication no
UsePAM yes
UseDNS no

# === Compliance affirmations (defaults made explicit for audit evidence) ===
IgnoreRhosts yes
HostbasedAuthentication no
```

```
PermitEmptyPasswords no
PermitUserEnvironment no

# === Session management ===
MaxSessions 4
MaxStartups 10:30:60
ClientAliveInterval 300
ClientAliveCountMax 2

# === Forwarding ===
AllowTcpForwarding no
X11Forwarding no
AllowAgentForwarding no

# === Other hardening ===
Compression no
TCPKeepAlive no

# === Modern crypto ===
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-
ctr,aes192-ctr,aes128-ctr
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com
KexAlgorithms sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-
sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256

# === Legal banner ===
Banner /etc/issue.net

# === Access control ===
AllowGroups sudo sg_serveraccess sg_sudo

# === SSH key retrieval ===
AuthorizedKeysFile none
AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys %u
AuthorizedKeysCommandUser nobody

# === Break-glass: pbr_admin ===
Match User pbr_admin Address 10.0.0.0/8,192.168.0.0/16
    PasswordAuthentication yes
    AuthenticationMethods password
```

```
# === Ansible automation account ===
Match User ansible
    AuthorizedKeysFile .ssh/authorized_keys
    AuthenticationMethods publickey
    KbdInteractiveAuthentication no
```

Directive Walkthrough

Authentication block

Directive	Value	Notes
`PermitRootLogin`	`no`	Root never logs in directly. Use `pbr_admin` + sudo or AD user + sudo.
`PasswordAuthentication`	`no`	Disabled globally. Re-enabled only inside the `pbr_admin` Match block.
`PubkeyAuthentication`	`yes`	Required by all flows except `pbr_admin`.
`KbdInteractiveAuthentication`	`yes`	Required for Duo PAM keyboard-interactive challenge. Disabled in `ansible` Match block.
`AuthenticationMethods`	`publickey,keyboard-interactive`	Both required. Overridden per-user in Match blocks for `pbr_admin` (password) and `ansible` (publickey only).
`MaxAuthTries`	`3`	Per-connection auth attempt limit.
`GSSAPIAuthentication`	`no`	We don't use GSSAPI/Kerberos for SSH auth. AD password validation happens via PAM/SSSD, not via Kerberos ticket forwarding.
`UsePAM`	`yes`	Required — Duo and pam_sss live in PAM.
`UseDNS`	`no`	Don't reverse-resolve client IPs into hostnames. Eliminates a slow DNS lookup on every connection and avoids confusion when client reverse-DNS is broken.

Compliance affirmations

These four directives are defaults in OpenSSH but stated explicitly for audit evidence:

Directive	Value	What it prevents
`IgnoreRhosts`	`yes`	.rhosts / .shosts trust files cannot be used for auth.
`HostbasedAuthentication`	`no`	Trust-by-host-key auth disabled.
`PermitEmptyPasswords`	`no`	Empty passwords cannot authenticate. (Belt-and-braces; `PasswordAuthentication no` already disallows.)
`PermitUserEnvironment`	`no`	Users cannot inject environment vars via `~/.ssh/environment` — prevents PATH/LD_PRELOAD-style attacks.

Session management

Directive	Value	Notes
`MaxSessions`	`4`	Concurrent multiplexed sessions per SSH connection. CIS recommendation.
`MaxStartups`	`10:30:60`	Up to 10 unauth'd connections; from 10-60, drop 30% randomly; reject at 60. Mitigates connection-exhaustion DoS.
`ClientAliveInterval`	`300`	Send keepalive probes every 5 minutes.
`ClientAliveCountMax`	`2`	Drop the connection after 2 missed keepalives. Idle sessions die after 10 minutes.

Forwarding (all disabled)

Directive	Value	What it prevents
`AllowTcpForwarding`	`no`	Local/remote port forwarding. No tunnel-the-DB-over-ssh patterns.
`X11Forwarding`	`no`	Graphical apps via X over SSH. Unused at PBR.

Directive	Value	What it prevents
`AllowAgentForwarding`	`no`	Forwarding ssh-agent to the remote host (would let a malicious admin on the remote pivot using your keys).

Other hardening

Directive	Value	Notes
`Compression`	`no`	Compression has historically been a source of side-channel attacks (CRIME-style).
`TCPKeepAlive`	`no`	Use SSH-level keep-alive (ClientAliveInterval) instead. TCPKeepAlive is unauthenticated and spoofable.

Modern Crypto

Ciphers

```
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
```

- AEAD ciphers preferred (chacha20-poly1305, aes-gcm) — encryption and integrity combined.
- aes-ctr modes retained for client compatibility with older OpenSSH releases (paired with hmac-sha2 in MACs).
- CBC modes and legacy 3DES/RC4/Blowfish/arcfour are all excluded.

MACs

```
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com
```

- `-etm` (Encrypt-then-MAC) only. Authenticates the ciphertext, preventing oracle attacks on the plaintext.
- SHA-2 family or umac-128. SHA-1 MACs are excluded.

Key Exchange (with post-quantum hybrid)

```
KexAlgorithms sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256
```

- `sntrup761x25519-sha512@openssh.com` — post-quantum hybrid KEX. Combines NTRU Prime (PQ) with X25519 (classical) so the resulting key is secure unless both are broken. Available in OpenSSH 9.0+.
- curve25519 fallbacks for clients without PQ support.
- ECDH (P-521, P-384, P-256) as classical fallbacks.
- SHA-1-based KEX, RSA-based KEX, and DH group 1/14 are all excluded.

Access Control: AllowGroups

```
AllowGroups sudo sg_serveraccess sg_sudo
```

sshd's `AllowGroups` is a hard allow-list checked early in the connection. A user must be in **at least one** listed group to even reach the authentication phase. Users not in any listed group get rejected with "User <user> from <ip> not allowed because none of user's groups are listed in AllowGroups".

The three groups:

Group	Origin	Members
`sudo`	Local Unix group	`ansible` (added by role preconditions), `pbr_admin` (added by manual bootstrap)
`sg_serveraccess`	AD group (SSSD-mapped)	AD users with SSH access (no sudo)
`sg_sudo`	AD group (SSSD-mapped)	AD users with sudo

Group names from AD are lowercased by SSSD when mapped to local POSIX groups, so the lowercase form is what sshd matches against.

Why include local `sudo` rather than special-casing `ansible` and `pbr_admin` via Match blocks: Match blocks override settings; they don't bypass `AllowGroups`. The user must qualify at the global level first. Listing `sudo` in `AllowGroups` is the simplest way to permit the two local accounts.

v2.4.1 corollary: Because `AllowGroups sudo` is what permits the `ansible` account to connect, the role must ensure `ansible` is in the local `sudo` group before the hardening config takes effect.

That's done idempotently in `preconditions.yml`.

Access Control: AuthorizedKeysCommand

```
AuthorizedKeysFile none
AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys %u
AuthorizedKeysCommandUser nobody
```

Three lines that change the default sshd key retrieval flow entirely:

- **`AuthorizedKeysFile none`** — disable the default file-based lookup (`~/.ssh/authorized_keys`). Critical: prevents AD users from bypassing AD-managed key revocation by writing their own key files.
- **`AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys %u`** — for each connection, sshd runs this command with the username, expects valid `authorized_keys`-format output on stdout.
- **`AuthorizedKeysCommandUser nobody`** — run the command as `nobody`. This is OpenSSH and SSSD's documented recommendation: the command should run as a low-privilege user.

The `sss_ssh_authorizedkeys` binary queries the SSSD ssh responder, which queries AD via LDAP for the user's `sshPublicKey` attribute. See **AD Integration & SSSD** for the full flow.

Match Block: pbr_admin (break-glass)

```
Match User pbr_admin Address 10.0.0.0/8,192.168.0.0/16
    PasswordAuthentication yes
    AuthenticationMethods password
```

Match conditions are AND-ed: the user must be `pbr_admin` AND connecting from one of the listed CIDRs. If both match, the block's directives override the global config for this connection only.

The overrides:

- `PasswordAuthentication yes` — re-enable password auth (globally `no`).
- `AuthenticationMethods password` — this user authenticates with password only (globally `publickey,keyboard-interactive`).

The source address list is templated from `pbr_admin_allowed_sources` in defaults. CIDR list, comma-separated, no spaces — per sshd_config(5) syntax.

Current scope (2026-06-17): widened to `10.0.0.0/8,192.168.0.0/16` — the break-glass account is reachable from anywhere on the internal 10/8 estate. As `pbr_admin` is password-only (carved out of the Duo PAM flow), this is a deliberately broad source scope; see the Version History on the *Known Limitations, Troubleshooting & Version History* page for the rationale and the plan to narrow it to per-site management CIDRs.

Important: this Match block does *not* bypass `AllowGroups`. `pbr_admin` must still be in `sudo` (handled by manual bootstrap, verified by preflight).

Match Block: ansible (automation)

```
Match User ansible
    AuthorizedKeysFile .ssh/authorized_keys
    AuthenticationMethods publickey
    KbdInteractiveAuthentication no
```

The `ansible` account is local-only and has no AD-side key. The overrides:

- **`AuthorizedKeysFile .ssh/authorized_keys`** — re-enable file-based key lookup (overrides global `none`). Bootstrap script installs the control node's public key here.
- **`AuthenticationMethods publickey`** — publickey is sufficient (overrides global `publickey,keyboard-interactive`). The ansible account skips PAM entirely on auth.
- **`KbdInteractiveAuthentication no`** — explicitly disable the keyboard-interactive flow for this user. Belt-and-braces with `AuthenticationMethods publickey`.

This is what lets Ansible run non-interactively, without Duo prompts, against every host.

Banner

```
Banner /etc/issue.net
```

The banner file is deployed by `roles/ssh-baseline/tasks/sshd.yml` from `roles/ssh-baseline/files/issue.net`. The banner displays before authentication — useful for legal notice and unauthorised-access deterrence.

Note: the banner content is in `files/issue.net` — not templated and not currently in the code dump. To inspect the deployed banner: `cat /etc/issue.net` on any baselined host.

Validation Flow

The role validates SSH config three times during deployment:

1. **During the Include directive write:** `lineinfile` validates via `sshd -t -f <tempfile>`. Catches a broken include line.
2. **During the hardening file write:** `template` validates via `sshd -t -f <tempfile>`. Catches a broken hardening directive before the file lands.
3. **After both files are in place:** `sshd -t` against the live combined config. Catches conflicts between the two files (which the per-file validation can't see).

Only after all three pass does the handler restart sshd.

Notes on Port 22 vs Custom Ports

From the inline comment in `defaults/main.yml`:

> ssh_port stays at 22. On Ubuntu 22.10+ and 24.04 LTS, OpenSSH uses systemd socket activation by default. If `ssh_port` is changed, `/etc/systemd/system/ssh.socket.d/` overrides must also be managed, or `ssh.socket` disabled in favour of `ssh.service`.

The role does not currently manage `ssh.socket` overrides. Changing `ssh_port` from 22 would require additional task work and is intentionally not supported until needed.

Where to Read Next

- **Configuration Reference** — the full list of SSH-related variables and how to override them
- **Duo MFA Integration** — the keyboard-interactive challenge that this hardening enables
- **AD Integration & SSSD** — how `sss_ssh_authorizedkeys` retrieves AD-stored keys

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`

Known Limitations, Troubleshooting & Version History

Known Limitations & Accepted Risks

LXC auditd compliance gap

Affected hosts: `pbr-graylog-kl1`, `pbr-thingsboard-kl1`

Issue: auditd cannot run inside LXC containers. The kernel audit netlink interface is isolated from container namespaces. Forcing auditd to start would fail with EPERM at the systemd start.

v2.4.2 introduced auto-detection: hosts with `ansible_virtualization_type == 'lxc'` have auditd installation but no service enablement. The `verify.yml` auditd assertion is skipped on these hosts.

Compliance implication: No local audit log capture on those two hosts. Compliance evidence for them depends entirely on remote logging via Graylog SIEM (system journal forwarding, application-level logs).

Mitigations in place:

- Both LXC hosts forward system events to Graylog
- Both run a limited service set with constrained external exposure
- Operating system logs are still captured via journald and forwarded

Future options to close the gap:

1. Migrate the affected workloads to KVM VMs (decouples from container constraints, restores local audit log capture)
2. Investigate Proxmox VE 9's enhanced container support for the audit subsystem (may not be available)

3. Formally accept the residual risk in PBR's risk register, citing the SIEM-based compensating control

---

Realm join multi-master replication retry pattern

Observed: During the v2.4.2 rollout, 3 of 5 hosts needed two attempts to complete `realm join` despite proper AD pre-clean.

Root cause: AD multi-master replication lag across PBR's 4 DCs. The `realm join` command picks a DC (via SRV record lookup), but that DC may not have replicated the deletion of the previously-cleaned-up computer object yet. The join then fails because "the object already exists."

Mitigation: Re-run the playbook. The role is idempotent, and by the time the second attempt runs, replication has usually caught up. The second attempt almost always succeeds.

Why we haven't added automatic retries: A `retries: 2, delay: 30` on the join task would mask the behaviour from operators. While that's convenient, it also hides a real symptom that's worth observing. Deferred to v2.5 with the intent to add retries plus a debug message about the replication-lag pattern.

ThreatLocker sudoers permission issue

Observed on: All hosts with ThreatLocker installed.

Issue: ThreatLocker's agent installs `/etc/sudoers.d/threatlocker_sudoers_general` with incorrect permissions. The file should be mode 0440 but is set to something `visudo -c` rejects. ThreatLocker enforces file immutability on its own files, so the permissions cannot be corrected.

Effect: `sudo` on the host does not honour the contents of that drop-in (it's rejected during sudoers parsing). Whatever rules ThreatLocker intended to install via that file are inactive.

Workaround in the role: preflight's `visudo -c` task ignores stderr lines mentioning `threatlocker_sudoers_general`. Any other sudoers error still fails preflight.

Action item: Raise with ThreatLocker support. Preflight emits a clear debug message when the workaround fires, so the operator is reminded each run.

Royal TS Rebex SSH library cannot do AuthenticationMethods publickey,keyboard-interactive

Issue: Royal TS 7's bundled Rebex SSH library does not support OpenSSH's `AuthenticationMethods publickey,keyboard-interactive` directive natively — it only handles one authentication method per session.

Symptoms: Royal TS fails to connect to baselined hosts with errors about authentication negotiation, or completes publickey auth and then disconnects without prompting for Duo.

Workaround: Set Royal TS's authentication method to `Any` under the connection's **Advanced → Security** properties. This lets Rebex negotiate either method, and the server-side `AuthenticationMethods` directive still requires both. The Duo keyboard-interactive prompt is then handled by the connection's interactive shell.

Alternative: Configure Royal TS to launch Windows OpenSSH (`ssh.exe`) as an External Application connection. Native OpenSSH handles `AuthenticationMethods` correctly and integrates with the 1Password SSH agent via the named pipe.

Hardcoded bootstrap SSH public key

Observed in: `scripts/bootstrap-ansible-user.sh`

The bootstrap script contains the control node's public key as a string literal:

```
PUBKEY="ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBMc7IDlr/IZ5M/2HcXU7cGCKZ03SLjpr5cbmiHnokdP
ansible-svc@pbr-ansible-kl1"
```

If the control node is rebuilt with a new ed25519 keypair, this script must be updated. The provenance comment in the script's banner explains the source.

This is a known trade-off: the script must work in isolation (run on a fresh host before any Ansible config is in place), so a hardcoded key is simplest. The alternative — templating the key into the script — would require a different deployment mechanism for the bootstrap step.

Banner file (issue.net) source not currently in repo dump

The role deploys `/etc/issue.net` from `roles/ssh-baseline/files/issue.net` via the `Deploy SSH login banner` task in `sshd.yml`. The banner file itself was not present in the v2.4.2 code dump used to author this documentation. To inspect the live banner, check any baselined host:

```
cat /etc/issue.net
```

Troubleshooting Reference

"User <user> from <ip> not allowed because none of user's groups are listed in AllowGroups"

Symptom: SSH connection rejected before authentication. Visible in the client with `ssh -vvv` and in `journalctl -u ssh` on the host.

Cause: The user is not a member of any group listed in sshd's `AllowGroups` directive (`sudo`, `sg_serveraccess`, `sg_sudo`).

For local accounts (ansible, pbr_admin): Verify membership in the local `sudo` group:

```
id ansible | tr ',' '\n' | grep -i sudo
id pbr_admin | tr ',' '\n' | grep -i sudo
```

If `ansible` isn't in `sudo`, re-run the role — v2.4.1's `preconditions.yml` adds it idempotently. This was the v2.4 → v2.4.1 fix.

For AD users: Verify SSSD resolves their group membership:

```
id a.mfraser
# Expected: a member of sg_serveraccess and/or sg_sudo (lowercased)
```

If the AD group memberships don't show, SSSD cache may be stale: `sudo sss_cache -E`.

realm join fails with no_log censored output

Symptom:

```
TASK [ssh-baseline : Join Active Directory domain] ****************************
fatal: [pbr-NEWHOST-kl1]: FAILED! =&gt; changed=true
  censored: 'the output has been hidden due to the fact that no_log: true was specified for
this result'
```

Most common cause: AD multi-master replication lag (the host being joined hits a DC that hasn't seen the previous computer object's deletion). Fix: re-run the playbook.

If second attempt also fails, dig deeper:

```
ansible pbr-NEWHOST-kl1 -m shell -a '
    journalctl --since "10 minutes ago" --no-pager 2&gt;&amp;1 \
        | grep -iE "realm|adcli|krb5|sssd|kerberos" | tail -40
    timedatectl status | head -8
' --become --vault-password-file ~/.ansible_vault_pass
```

Look for: clock skew (Kerberos requires <5 min skew with KDC), DNS resolution failures, computer object already exists messages, "krbtgt" related errors (KDC contact failures).

Last resort — temporarily remove no_log: Edit `roles/ssh-baseline/tasks/ad-join.yml`, comment out the `no_log: true` on the realm join task, re-run with output going to stdout (not `tee`'d to disk). Restore `no_log: true` immediately after. Scrub any tee'd diagnostic logs with `shred -u`.

SSSD user doesn't resolve via getent

```
getent passwd a.mfraser
# (no output, rc=2)
```

Possible causes (test in order):

1. **User not in SG_ServerAccess or SG_Sudo** — The `ad_access_filter` in SSSD excludes them. Check group membership in ADUC.

2. **SSSD service not running** — `systemctl status sssd`. If down, `systemctl start sssd` and check journalctl for the failure reason.
3. **SSSD cache stale** — `sudo sss_cache -E` invalidates the cache; SSSD re-queries AD on next lookup.
4. **SSSD offline** — `sssctl domain-status pbr.org.au`. ONLINE means LDAP is reachable; OFFLINE means SSSD has lost contact with DCs.
5. **LDAP connectivity broken** — verify DC reachability: `nc -zv 10.1.8.90 389; nc -zv 10.1.8.90 88`.

SSH key not retrieved from AD

Symptom: sshd publickey auth fails for an AD user whose `sshPublicKey` attribute is populated.

Diagnostic: Run the same lookup sshd does:

```
sudo -u nobody /usr/bin/sss_ssh_authorizedkeys a.mfraser
```

Expected: The user's public key on stdout.

If empty:

- Verify `sshPublicKey` populated on the AD user object (in ADUC or via PowerShell `Get-ADUser a.mfraser -Properties sshPublicKey`)
- Verify SSSD's ssh responder is running: `services = nss, pam, ssh` in `/etc/sssd/sssd.conf`. Re-deploy the role to restore if drifted.
- Verify SSSD is online: `sssctl domain-status pbr.org.au`
- Clear cache: `sudo sss_cache -E` and retry

Duo: "Permission denied" without a Duo prompt

Cause: Auth rejected before PAM ran. Most likely `AllowGroups` rejected the user.

```
ssh -vvv a.mfraser@host.pbr.org.au 2&gt;&amp;1 | grep -iE 'permission
denied|allowgroups|publickey'
```

Also possible: publickey auth failed (no matching key in AD) and the connection terminated before keyboard-interactive.

Duo: prompt arrives but authentication fails

Check the host's Duo logs:

```
sudo journalctl -u ssh --since "5 minutes ago" | grep -iE 'duo|pam'
```

Common causes:

- Duo API credentials wrong in `/etc/duo/pam_duo.conf` — vault credentials mismatch. Re-run the role to refresh.
- System clock drift — Duo's API requires close NTP sync. `timedatectl status`.
- User disabled in Duo admin console.
- User's primary device unreachable (no network on phone, app not installed).

---

sudo asks for password but never prompts for Duo

Cause: User is not in `sg_sudo`, so the `pam_succeed_if user notingroup sg_sudo` branch fired and skipped pam_duo. By design.

```
id a.mfraser | tr ',' '\n' | grep -i sg_sudo
```

If the user should be in sg_sudo but isn't showing: stale SSSD cache. `sudo sss_cache -E`.

Local sudo broken after role run

Caught by the role itself — the validation task `Sanity check - sudo still works for non-Duo automation accounts` runs `sudo -n true` as the `ansible` user during deployment. If this fails, the playbook aborts with a clear error, before later tasks that depend on working sudo.

If it does break (e.g. a manual edit to `/etc/pam.d/sudo` went wrong):

```
# As pbr_admin (break-glass, password auth):
ssh pbr_admin@&lt;host&gt;
sudo -i
```

```
# Restore Ubuntu default:
DEBIAN_FRONTEND=noninteractive apt-get install --reinstall -y \
    -o Dpkg::Options::="--force-confmiss" sudo

# Then re-run the role to restore the Duo-aware /etc/pam.d/sudo properly
```

SSH times out from one source only — fail2ban ban (incl. self-lockout)

Symptom: SSH to a baselined host *times out* — not "Connection refused", not "Permission denied" — and only from your source IP, while the host is up and other sources can still connect. fail2ban's drop action discards packets silently, so a ban presents as a connection *timeout* rather than a refusal. Repeated failed logins (or repeated attempts that reached the auth stage) can ban your own admin workstation.

Confirm the ban — needs another route onto the box (Proxmox/iKVM console, an unbanned source, or the Ansible control node, which is in `ignoreip`):

```
fail2ban-client status                # list active jails
fail2ban-client status sshd           # jail detail: currently banned + Banned IP list
fail2ban-client get sshd banned       # banned IPs for the sshd jail (fail2ban 0.11+)
grep "Ban " /var/log/fail2ban.log     # ban history  (or: journalctl -u fail2ban)
```

Release a ban:

```
fail2ban-client set sshd unbanip <ip>   # unban a specific IP in the sshd jail (always
available)
fail2ban-client unban <ip>              # unban an IP across all jails (fail2ban 0.10+)
fail2ban-client unban --all           # clear every ban in every jail
```

Prevent recurrence: add the trusted admin/control source to the fail2ban `ignoreip` allow-list in `roles/ssh-baseline/defaults/main.yml` (the same list that already exempts `127.0.0.1/8`, `::1`, the server LAN, and the `10.1.8.80/32` control node). IPs in `ignoreip` are never banned. Note POS hosts override `fail2ban_sshd_bantime` to `604800` (7 days), so a self-ban there persists far longer than the default.

Version History

2026-06-17 — pbr_admin source allow-list widened to 10.0.0.0/8 (role version tag TBC)

- Changed default `pbr_admin_allowed_sources` from `10.1.0.0/16,192.168.0.0/16` to `10.0.0.0/8,192.168.0.0/16`. The explicit `10.1.0.0/16` was dropped as it is a subset of `10.0.0.0/8`; `192.168.0.0/16` (TEMPORARY) retained.
- Effect: the `pbr_admin` break-glass Match block now accepts password auth from any host on the internal 10/8 estate, not just the server LAN. Per-host overrides (e.g. `pbr-pos-belgrave.yml` pinned to `10.1.8.0/24`) are unaffected.
- Trade-off noted: `pbr_admin` is password-only (carved out of the Duo PAM flow), so this widens the source scope of the one MFA-exempt account. Accepted as an interim measure; revisit narrowing to per-site management CIDRs once VLAN segmentation consolidates admin sources (tracked with the `192.168.0.0/16` TEMPORARY entry).
- Docs updated: SSH Hardening Reference (Match block + rendered source + scope note) and Configuration Reference (variable default). Role version tag to be assigned on commit.

v2.4.2 (current)

Title: Auto-skip auditd on LXC containers

Commit: `6286698` (with companion commits `296ab08`, `52befaf`, `56c0f73`)

Changes:

- `packages.yml`: added `set_fact: _manage_auditd` with auto-detection logic. Conditional `Enable auditd` service task.
- `verify.yml`: duplicate auto-detection added so verify works independently of packages.yml. Auditd assertion gated on `_manage_auditd`.
- `defaults/main.yml`: `manage_auditd: auto` default with explanatory comment.
- Companion: `scripts/bootstrap-ansible-user.sh` added to the repo (was previously informal).
- Companion: `296ab08` restored `no_log: true` on the realm join task (a temporary removal during diagnostic work).
- Companion: `52befaf` added `pbr-thingsboard-kl1` to inventory.

Rolled out: All 5 hosts — `pbr-uisp-kl1`, `pbr-docker-kl1`, `pbr-graylog-kl1`, `pbr-lme-kl1`, `pbr-thingsboard-kl1`.

---

v2.4.1

Title: Ensure ansible automation account is in sudo group

Commit: `4eb86b4`

Problem: After v2.4's `AllowGroups sudo sg_serveraccess sg_sudo` took effect on hosts where the `ansible` account had been bootstrapped historically without sudo group membership, sshd rejected the ansible connection with "User not allowed because none of user's groups are listed in AllowGroups."

Why it surfaced: The canary host (`pbr-uisp-kl1`) had had `ansible` added to `sudo` by an earlier manual bootstrap. `pbr-docker-kl1` did not. When v2.4 rolled to docker-kl1 with the hardened `AllowGroups`, the ansible session was severed mid-deployment.

Fix: `preconditions.yml` now runs as the first task of the role:

```
- name: Ensure ansible automation account is in local sudo group
  ansible.builtin.user:
    name: ansible
    groups: sudo
    append: true
```

Idempotent: if already a member, no-op. The role owns this prerequisite rather than depending on bootstrap variations.

---

v2.4

Title: Duo MFA on sudo for AD sudo group

Commit: `7eaf35a`

Changes:

- New template: `pam_sudo.j2` — PAM stack for `/etc/pam.d/sudo` with pam_duo, `pam_succeed_if user notingroup sg_sudo` carve-out, common-auth/account/session-noninteractive includes.
- New sudoers drop-in: `/etc/sudoers.d/sudo_timestamp_timeout` setting `Defaults timestamp_timeout=30`.
- New tag: `sudo-mfa` on the sudo PAM tasks.
- New defaults: `duo_sudo_enabled: true`, `sudo_timestamp_timeout: 30`.

- Validation: `grep -c "pam_duo.so" /etc/pam.d/sudo` and a runtime `sudo -n true` as the ansible user.

Compliance reference: Essential Eight ML2 — MFA for privileged users performing privileged actions. The only compliance reference in the role source code.

---

v2.3

Title: Duo MFA via duo-unix from Duo's official repo

Commit: `9d11756` (initial: `e02e4ac`)

Changes:

- New task file: `duo.yml` — GPG key fetch, APT repo add, legacy `libpam-duo`/`libduo3` purge, `duo-unix` install.
- New templates: `pam_duo.conf.j2` (with vault credentials), `pam_sshd.j2` (PAM stack for sshd).
- SSH `AuthenticationMethods` default changed to `publickey,keyboard-interactive`.
- New defaults: `duo_failmode: safe`, `duo_pushinfo: yes`, `duo_prompts: 3`, `duo_autopush: yes`, `break_glass_user: pbr_admin`.
- Why not Ubuntu universe `libpam-duo`: outdated 1.11.3 (2022) version, incompatible with current Duo Auth API, doesn't support April 2026 CA bundle rotation.

---

v2.2.1

Title: Remove invalid `core_dumpable` from sssd.conf.j2

Commit: `016259c`

Changes: Removed the `core_dumpable = false` directive from the SSSD config template — not a valid sssd.conf option, was silently being ignored.

---

v2.2

Title: krb5 udp_preference_limit, explicit ldap_id_mapping

Commits: `43a1aa5`, `4032534`

Changes (canary learnings from pbr-uisp-kl1):

- krb5.conf: added `udp_preference_limit = 0` to force TCP for Kerberos — addresses UDP packet size issues with large PAC (users in many groups).
- sssd.conf: explicit `ldap_id_mapping = True` — was implicit, made explicit for reviewability.
- General SSSD/PAM/sshd alignment tweaks discovered during canary deployment.

---

v2.1

Title: Drop ssh_local_access group; sudo group is the local gate

Commit: `0bdccfa`

Changes: Earlier versions referenced a custom `ssh_local_access` group for the local-account allow path. Simplified to use the standard local `sudo` group instead — one fewer thing to manage during bootstrap.

---

v2.0

Title: Baseline pre-canary-deploy

Commit: `f681246`

Description: The first version considered complete enough for canary deployment. v1 series was scaffolding (`96c3f79` initial structure, `11e8ee9` inventory, `44bf79e` vault + group_vars).

Deferred Items (Planned for v2.5)

These items have been identified during the v2.4 → v2.4.2 development cycle but deferred to keep the immediate release focused:

Item	Rationale to defer
CIS audit rules baseline (auditd rule file deployment)	Need to scope which CIS Linux Workstation/Server Profile applies. Useful but not blocking baseline operation.
Audit log forwarding to Graylog (auditd → audisp-remote)	Closes the LXC compliance gap if combined with auditd-on-KVM. Requires Graylog input config and a forwarder package decision.
`verify.yml` `vars_files` import for defaults inheritance	Currently verify.yml duplicates the `manage_auditd` logic from packages.yml. Cleaner via shared defaults file, but works correctly as-is.

Item	Rationale to defer
`retries: 2, delay: 30` on the realm join task	Would mask the multi-master replication lag pattern from operator view. Tension between operator visibility and automation smoothness.
Refactor `manage_auditd: 'auto'` sentinel	The string sentinel mixed into a boolean variable is awkward. Could be split into `manage_auditd: true|false` with a separate `manage_auditd_auto_skip_lxc: true` guard. Cosmetic; current logic is correct.

Where to Read Next

- **Overview & Repository Layout** — if you've reached this page first, start here
- **Deployment Runbook — New Host** — the standard procedure
- **Architecture & Design Decisions** — the "why" behind everything in the role

Adding SSH Key via 1password

1. Create SSH key in 1Pass.
2. Copy Public Key
3. Go to AD Server, open Powershell as admin.
4. Edit command

   ```
   Set-ADUser -Identity samaccountname -Add @{sshPublicKey="ssh-ed25519 AAAA...
   user@host"}
   ```

   Example would be Set-ADUser -Identity **a.dhealey**-Add @{sshPublicKey="**ssh-ed25519 AAAA... a.dhealey@1pass**"}

5. Run command. This should eddit the SSHPublickKEy attribute for the AD user.

6. Add fingerprint to whatever you are using for.