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

---

Revision #1
Created 2026-05-13 05:24:26 UTC by PBR_AI
Updated 2026-05-13 05:24:26 UTC by PBR_AI