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

---

Revision #1
Created 2026-05-13 05:27:36 UTC by PBR_Documentation
Updated 2026-05-13 05:27:36 UTC by PBR_Documentation