

Forward

One identity provider, every service, and the configuration that makes it work.

This book is a working reference for putting every service in a self-hosted environment behind a single identity provider. It opens with the case for why you should centralize your identities and the reasoning behind the tooling, then gives annotated, reproducible configuration for three representative integration patterns: an identity-only app (Discourse), a group-aware app (BookStack), and an app you write yourself (a Flask service). Each configuration page calls out the pattern it represents so it can be adapted to similar applications.

Why centralize identity

Every application that manages its own logins is a small identity system you now have to run. It stores passwords, it decides how (or whether) to do multi-factor authentication, it has its own idea of what a session is, and it keeps its own list of who is allowed in. Run a handful of apps that way and you have a handful of password stores, a handful of MFA configurations of varying quality, and a handful of places you have to remember to visit when someone should no longer have access.

A single sign-on identity provider (IdP) collapses all of that into one place. Users authenticate once, against one system, and every application trusts that system instead of holding credentials of its own. The practical consequences are the whole point:

- **One place to authenticate.** One login, one session, one password users actually have to remember, which means it can be a strong one, backed by MFA, instead of ten mediocre ones.
- **One place to enforce policy.** MFA, password rules, and session lifetimes are configured once and apply everywhere. No app can quietly be the weak link.
- **Apps stop storing passwords.** Each application holds a client secret and a trust relationship, not user credentials. The blast radius of any single app being compromised shrinks accordingly.
- ***One place to revoke.** Offboarding is a single action. Disable the account at the IdP and every downstream application is closed at once, with no hunting through ten admin panels hoping you didn't miss one.

This matters *more* at small scale, not less. A large organization has an IAM team to manage accounts. One person running an environment for friends and family will find it difficult to keep account state consistent and secure across every service by hand. Centralizing identity is what makes right-sized, genuinely secure operation achievable by one person. It is the single highest-

leverage security decision in the whole environment.

There is a catch: for any of this to hold, *every* service has to sit behind the IdP. A single application with its own local-login escape hatch defeats the revocation guarantee, the MFA guarantee, and the "apps don't store passwords" guarantee all at once. Universality isn't a nice-to-have here. It is the property that makes the model worth anything.

“***SSO centralizes authentication, not sessions.** Each application maintains its own local session once a user has logged in, and disabling an account at the identity provider does not automatically end those sessions. What actually happens on revocation depends entirely on the application:

- Applications that revalidate against the IdP, or that support back-channel logout and have it configured, close promptly.
- Applications with fully self-contained session management keep the user signed in until their local session expires, even with no active IdP sessions.
- Already-issued access tokens remain valid until they expire, because applications validate them by signature rather than by asking the IdP whether they are still good. Shortening token lifetime shrinks that window but does not eliminate it.

There is also a quirk worth knowing: in Keycloak, signing out an individual session fires back-channel logout to clients that support it, while the bulk "sign out all sessions" action does not.

The practical consequence is that full offboarding is a sequence, not a single click.

- Disable the account at the IdP first so nothing can re-authenticate
- Revoke refresh and offline tokens and push a not-before policy
- Then evict each application's local session

In my environment that last step is a script, because the applications expose different eviction surfaces, from clean per-user admin APIs to session tables that have to be cleared directly to filesystem session stores with no user index at all, where the only option is logging everyone out of that app.

None of this undermines the case for centralizing identity. Doing this without an IdP means hunting through every application's admin panel with no single gate to close first, which is undeniably worse. But "one action closes everything" oversells it, and the gap between stopping new access and ending existing access is the kind of detail that matters during an incident.

Why Keycloak

The IdP I chose for my environment is [Keycloak](#). The reasons it won out:

- **Open-source and self-hostable.** It runs on infrastructure I control, with no per-seat SaaS pricing and no user identities living in someone else's cloud. For a privacy-focused, self-hosted environment, that alignment matters; the identity system shouldn't be the one component you rent from a third party.
- **Standards-based.** It speaks OIDC, OAuth 2.0, and SAML, universal protocols of which any application that supports SSO will utilize at least one, and the knowledge transfers directly to enterprise systems. Every integration below is a standard OIDC flow.
- **Realms give real isolation.** Keycloak's realm model lets me run entirely separate identity domains. Production services authenticate against one realm; development instances authenticate against a second. Dev accounts and prod accounts never mingle, and I can experiment with client and mapper configuration in dev without risking the identity system real people depend on.
- **Enough authorization power without the bloat.** Groups, roles, client scopes, and protocol mappers are enough to drive real access control in every downstream app, without dragging in a heavyweight enterprise IAM suite that would be absurd at this scale.

The honest tradeoff: Keycloak has a real learning curve, and it does not shy away from breaking changes between major versions. Upgrades need planned maintenance windows with database backups taken first, release notes read carefully, and realm and client configuration re-verified afterward, not a casual `docker compose pull`. That is a cost taken on with eyes open. The centralization it buys is worth the operational care it demands.

Keycloak has a good startup guide to get you started here: [Keycloak Startup Guide](#)

How it fits the architecture

My IdP lives in the environment's infrastructure and trust zone, on its own host, and like everything else, the host and config structure is reachable only from within the VPC. Applications never handle credentials themselves. When a user hits an application without a valid session, the app redirects the browser to Keycloak; the user authenticates there (with MFA when required); Keycloak returns an authorization code; and the application exchanges that code for tokens on a back channel it never exposes to the browser. Authorization is then decided by the application from the claims in the token it receives.

Every integration uses the OIDC authorization-code flow with PKCE (S256), confidential clients, and standard flow only. That uniformity is deliberate: a new service isn't a fresh design problem, it is the same pattern applied again, which is what keeps the whole thing maintainable by one person.

Revision #18

Created 2026-07-20 03:25:52 UTC by Chris Landis

Updated 2026-07-27 20:19:25 UTC by Chris Landis