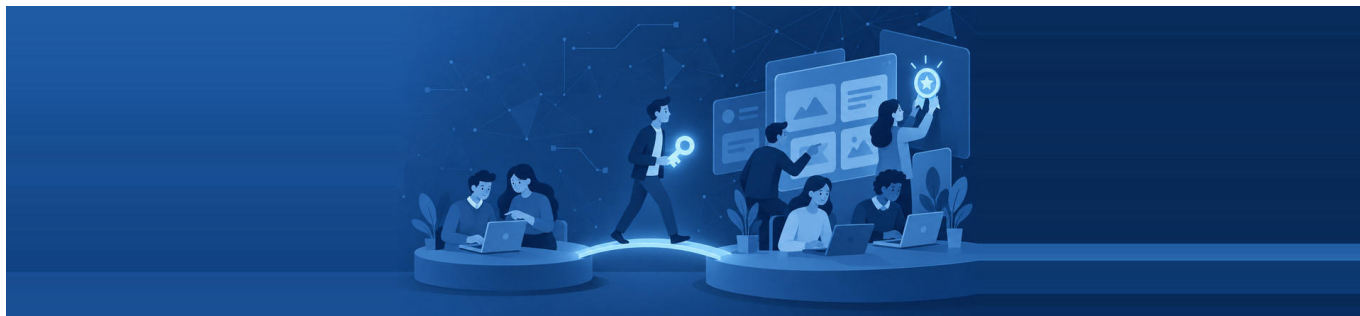


Blog » One login for staff and customers: how identity brokering saved us from duplicate accounts

One login for staff and customers: how identity brokering saved us from duplicate accounts

We run sites that mix paid content with internal material, and our apps talk to one Keycloak realm only. Here is how two realms can share one login without anyone creating a second account.



Derafu · 09/10/2026 · #security, #keycloak, #authentication

One login for staff and customers: how identity brokering saved us from duplicate accounts

We run several sites, and they do not all have the same audience. Some content is paid for (a course, a set of tools), some is internal and free but limited to the people on the team, and some is for any logged-in visitor. One site can have all three: a catalog of courses for everybody with an account, and a manual that only the team should read.

Access to all of it is decided by roles, in Keycloak. That part was easy. The part that was not was **who has an account where**.

The wall we ran into

A Keycloak realm is a boundary: its own users, its own policies, its own administrators. And an application that logs people in does it against **one** realm. Ours is no exception.

So the team and the customers had two options, and neither was good:

- **Everybody in one realm.** It works until you want different password policies, a different administration for the team's accounts, or a smaller blast radius if something goes wrong with the customers' side. The people who run the company should not live in the same pool as the people who buy from it.

- **Two realms, two accounts per person.** The team keeps an internal account, and gets another one to use the public sites. Two passwords, two places to forget to disable someone, and a sure source of “which one was I using?”.

We wanted a third thing: **two realms, one login.**

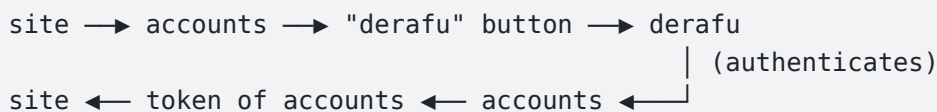
The idea: let one realm trust the other

Keycloak calls it *identity brokering*. A realm can accept the login of another realm, so the person authenticates where their account really lives and the first realm only receives the result.

We have two realms:

- **derafu**, for the team.
- **accounts**, for everybody who uses our sites: customers and team alike. We named it after what the accounts are, not after who pays: whether somebody pays is a matter of roles, not of identity.

The sites only ever talk to **accounts**. This is what happens when a member of the team opens one:



1. The site sends the person to **accounts**, as it always did.
2. On the login page of **accounts** there is a button for **derafu**. The person clicks it and logs in there, with the password they already have.
3. **derafu** tells **accounts** who this is.
4. **accounts** creates (the first time) an account linked to that identity, and issues its **own** token, with **its own** roles.

The site sees a token of **accounts**, like any other. It does not know that a second realm exists, and nothing in its configuration changes.

How we set it up

It takes a few minutes, in the administration console of Keycloak.

In **derafu**, the realm that provides the identity:

1. Create a client of type OpenID Connect, for example **accounts-broker**, with *Client authentication* turned on.
2. Set its *Valid redirect URIs* to the URI that **accounts** gives you when you create the provider (next section). It looks like `https://<host>/realms/accounts/broker/derafu/endpoint`.

3. Copy its client secret, from the *Credentials* tab.

In `accounts`, the realm that receives it:

1. Go to *Identity providers* and add a *Keycloak OpenID Connect* provider.
2. The alias is `derafu` (it is part of the redirect URI above).
3. The discovery endpoint is `https://<host>/realms/derafu/.well-known/openid-configuration`.
4. The client ID is `accounts-broker`, and the secret is the one you copied.
5. In the *Mappers* of the provider, add a *Hardcoded Role* mapper that gives the realm role `staff` to everyone who arrives through it. Everybody in `derafu` is on the team, so the role marks them by itself. If you make `staff` a composite role that includes the roles for the internal areas, they also get access to those with no manual work.
6. Try it with someone from the team.

That is all the sites need. A person who arrives this way is, for the application, one more user with some roles.

What we did not expect

Three things worth knowing

The first two cost us a minute of thinking each; the third could be a real problem if you miss it.

- **The first login creates an account in `accounts`.** Keycloak runs its *first broker login* flow, which may ask the person to review their profile or verify their email. Marking the provider as *Trust email* skips that when the realm that provides the identity is yours.
- **The login page shows a button, not the form of `derafu`.** Keycloak can jump straight to a provider with the `kc_idp_hint` parameter, but our authentication library does not send it today, so the team member picks the button. It is a small thing, and we have not needed more.
- **Disabling someone in `derafu` does not disable their account in `accounts`.** They cannot log in through the button anymore, but the linked account and any session that already exists there are not touched. If your offboarding only says “disable the internal account”, there is a hole in it. Disable the person in both realms, or remove the `staff` role in `accounts`, and write it down in the checklist.

What we would tell someone starting

- **Name realms for who the accounts are, not for what they buy.** `accounts` and `derafu` still make sense next year. `customers` stopped making sense the day the team needed to log in too.
- **Put the permissions in roles of each site's client**, not in the realm: one client per site, and roles named for what they give access to (`course-dte`, `docs-internal`). Realm roles are only for what means the same everywhere, like `staff`.
- **Start without the broker if you want.** You can create the team's accounts in `accounts` by

hand and add the broker later, without touching roles, clients or the configuration of any site.

In short

When the people who work on a product are also users of it, the question is not “which realm do they belong to” but “where should their password live”. Identity brokering lets it live in one place and be accepted in the other, and the applications stay oblivious to it.

How does Derafu Auth fit in?

The sites talk to Keycloak through Derafu Auth, which keeps authentication (who is asking) apart from authorization (may they enter), and checks roles by path.

[Read how it works](#)

Last updated on 09/10/2026