

API

Authenticate the clients of an API with a header: Bearer tokens (Keycloak) and Basic (database, httpasswd)

Anonymous · 08/10/2026

API

A browser keeps a session: it logs in once and then sends a cookie. A program that calls an API does not: it sends its credentials **in every request**, in the `Authorization` header, and it can not follow a redirect to a login page. Derafu Auth does both with the same providers and the same [protected paths](#); what changes is **how the client is identified**.

Two settings, two questions

Variable	Question	Default
<code>AUTH_API_PATHS</code>	How is a client authenticated on this path? With the credentials of the header, with no session, and a failure is a 401 in JSON (never a redirect).	<code>["/api"]</code>
<code>AUTH_PROTECTED_PATHS</code>	What needs a user (and which roles)? See Route Protection .	<code>[]</code>

They are independent, so each combination has a meaning:

```
# 1. The usual: /api is for programs and it needs a user; the pages need a session.  
AUTH_PROTECTED_PATHS='["/api", "/dashboard"]'
```

```
# 2. A public API: it is read with the header if it comes, and anybody can call it.  
AUTH_PROTECTED_PATHS='["/dashboard"]'
```

```
# 3. An API with roles, outside /api.  
AUTH_API_PATHS='["/api", "/v2"]'  
AUTH_PROTECTED_PATHS='{"/api": [], "/v2/admin": ["admin"]}'
```

```
# 4. A path for browsers and programs at once (it uses the session if there is  
#    no header, and the header if there is one).  
AUTH_API_PATHS='["/api", "/reports"]'  
AUTH_PROTECTED_PATHS='["/reports"]'
```

```
# 5. Nothing is an API: only sessions.  
AUTH_API_PATHS='[]'
```

On a path of `AUTH_API_PATHS`:

- **Header with valid credentials:** the user is the one of the credentials. **No session is created or read:** no cookie is sent and nothing is stored.
- **Header with invalid credentials:** `401`. It does **not** fall back to the session: a client that sent credentials gets an answer about those credentials.
- **No header:** the session still works (the case 4 above), and if the path is protected the answer is the `401`.
- **A header with another scheme** (Basic where the provider reads Bearer) is ignored, as if it was not there.

The header is read **only** on those paths. On any other path it is ignored, so a page can not be reached by a client that is not meant to be there.

Which credentials

One scheme for each provider, the one that reflects what it can verify:

Provider	Scheme	What the client sends
Keycloak	<code>Bearer</code>	An access token of the realm. A service (client credentials) and a person (a login of any flow) send the same thing.
Database	<code>Basic</code>	The identity and the password of a user of the table.
Htpasswd	<code>Basic</code>	The user and the password of the file.

The `401` and the `WWW-Authenticate` header

Every `401` must say how to authenticate (RFC 7235), and Derafu Auth does (with the one exception of [scripts of a page](#)):

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="API", error="invalid_token"
Content-Type: application/json

{"status": 401, "title": "...", "detail": "...}"
```

- `Bearer realm="API"` (Keycloak) or `Basic realm="API", charset="UTF-8"` (database and htpasswd) is the **challenge**.
- `error="invalid_token"` is added to `Bearer` only when a token **was sent and is not valid** (RFC 6750). Without credentials it is not there.
- The **realm** of this header (`AUTH_API_REALM`, `API` by default) is only a label of the protection space: a name for the clients. It is **not** the realm of Keycloak (`AUTH_KEYCLOAK_REALM`). It can not have quotes, backslashes or control characters.

A user that is authenticated but is not granted by the roles gets a 403, as in the pages (see [Authorization](#)).

Scripts of a page and the window of the browser

A browser answers a 401 with WWW-Authenticate: Basic by opening **its own window** to ask for a user and a password, whatever the body of the answer is. A page that calls the API with its session, when the session has expired, wants the 401 in JSON, not that window. So the challenge of Basic is **left out** when the request has the header X-Requested-With: XMLHttpRequest, as Rails and Spring do: the answer is still a 401 with the JSON, only without the header. Send that header from the scripts of your pages (fetch(url, {headers: {'X-Requested-With': 'XMLHttpRequest'}}); most libraries, like jQuery, add it for you).

A client that is a program (curl, a service) does not send it, and gets the challenge. Bearer is never answered with a window by a browser, so its challenge is always sent.

Keycloak: access tokens

The token is verified **without calling Keycloak** (signature with the keys of the realm, which are cached; iss, exp and nbf if they are in the token), and then:

1. **typ must be Bearer**. An ID token (ID), a refresh token (Refresh) or an offline token (Offline) are not access tokens, even when they are signed by the realm.
2. **The audience (aud) must be this API** (AUTH_KEYCLOAK_API_AUDIENCE, the client of the API by default: see [a client for the API](#)). A token that Keycloak gave for another service is not valid here.
3. **Introspection** (see below): Keycloak is asked whether the token is still active.

The roles are the ones of the realm and the ones of the client of the audience, as in the login. The exp is not required: it is checked only if the token has it (with 60 seconds of leeway).

Setting up the audience in Keycloak

A token only has the audience that Keycloak is told to put in it. The default access token of a client has aud with account, not with the client, so **without this step every token is refused** (The audience of the token is not this API). Step by step, in the admin console of Keycloak:

1. **Clients** → the client of the API (the one of AUTH_KEYCLOAK_CLIENT_ID). If a service calls the API, that service has its own client, with **Client authentication** on and **Service accounts roles** checked in its *Capability config*.
2. Open the client that **asks for the token** (the service, or the application of the people) → **Client scopes** tab → the scope <client>-dedicated → **Configure a new mapper** → **Audience**.
3. Fill the mapper:
 - **Name:** api-audience.

- **Included Client Audience:** the client of the API (the same value as `AUTH_KEYCLOAK_API_AUDIENCE`; by default, `AUTH_KEYCLOAK_CLIENT_ID`).
 - **Add to access token:** on. **Add to ID token:** off.
4. Save. Ask for a new token (the old ones do not change) and look at it in jwt.io: "aud": "<the client of the API>" (or a list that has it) and "typ": "Bearer".

To ask for a token as a service:

```
curl -s -X POST "$AUTH_KEYCLOAK_URL/realms/$AUTH_KEYCLOAK_REALM/protocol/openid-connect/token" \
  -d grant_type=client_credentials \
  -d client_id=my-service -d client_secret=the-secret \
  | jq -r .access_token
```

And to call the API:

```
curl -H "Authorization: Bearer $TOKEN" https://app.example.com/api/items
```

Introspection

A token that is signed and has not expired can still be **revoked**: the user was disabled, its session ended. Only Keycloak knows, so by default the API asks it in every request (token introspection, RFC 7662) and refuses a token that is not `active`, or that is the token of another user. It costs one request to Keycloak for each call of the API, and **if Keycloak does not answer, the answer is a 401**: a token that can not be checked is not accepted.

To not ask, and trust the lifespan of the token:

```
AUTH_KEYCLOAK_API_INTROSPECTION=false
```

Then a token is valid until it expires, so make its lifespan short (in the realm: *Realm settings* → *Tokens* → *Access Token Lifespan*; or only for a client, in its *Advanced* tab). A service that needs a long-lived credential should ask for a new token when the old one expires (client credentials costs nothing and no one has to renew it by hand), instead of a long lifespan. An *offline token* is a refresh token: it is used to **get** access tokens, it is not a `Bearer` for the API.

Keycloak lets a client introspect only the tokens that have that client in their audience. So when the introspection is on, the audience **must be the client that asks**; a different `AUTH_KEYCLOAK_API_AUDIENCE` is an error of the configuration that says so. If the API is another audience, turn the introspection off.

A client for the API

By default the client that asks Keycloak is the one of the application (`AUTH_KEYCLOAK_CLIENT_ID`), the

same of the login, and the audience is that client. If the API is a client of its own (in Keycloak, a client with **Client authentication** on and no login flows, that can only introspect), give its ID and its secret, **both**:

```
AUTH_KEYCLOAK_API_CLIENT_ID=billing-api
AUTH_KEYCLOAK_API_CLIENT_SECRET=the-secret-of-billing-api
```

Then Keycloak is asked with it, and the audience (the one of the mapper, see above) is `billing-api`. The login keeps using its own client.

Variable	Default	Description
<code>AUTH_KEYCLOAK_API_AUDIENCE</code>	the client of the API	The audience that the token must have.
<code>AUTH_KEYCLOAK_API_CLIENT_ID</code>	the client of the application	The client of the API, that asks Keycloak about the tokens.
<code>AUTH_KEYCLOAK_API_CLIENT_SECRET</code>	the secret of the client of the application	Its secret. It is required if the ID is given, and the other way around.
<code>AUTH_KEYCLOAK_API_INTROSPECTION</code>	<code>true</code>	Whether Keycloak is asked if the token is active.

Database and httpasswd: Basic

The client sends `Authorization: Basic base64(identity:password)`. The user is verified like in the login (the password is checked against its hash, and a user that is inactive or does not exist is refused), and the failures are counted by the `throttle` of the login, by identity and by the network of the client, so the API can not be used to guess passwords.

```
curl -u admin:the-password https://app.example.com/api/items
```

Two things to know:

- **Every request checks the password**, and with `bcrypt` that is deliberately slow (tens of milliseconds). There is no cache of verified passwords. If the API has a lot of traffic, use the Keycloak provider.
- The credentials go in **every** request, so use HTTPS.

Last updated on 08/10/2026