Enable single sign-on (SSO) with JWT

Build your help center

This article walks you through setting up JWT SSO on a HelpCenter.io help center. About 10 minutes end-to-end if you already have a backend that can sign HS256 tokens. If you're new to the concept first, read How Single Sign-On with JWT works.

Before you start

- You're on the **Catalyst, or Enterprise** plan. JWT SSO is included in all three; no support ticket is needed to unlock it.
- Your help center is set to **Private** (or **Password**) visibility. SSO is what gates access for non-public sites; it does nothing on a public help center.
- You have a backend that can sign HS256 JWTs. Most application stacks have this in their standard library or one npm install / composer require away.

Step 1 — Open the SSO settings

In the dashboard, go to **Settings → Security → SSO** and toggle SSO on.

You'll see a card grid with the providers we support:

- **JWT (HS256)** — Live. Use this. The rest of this guide covers it.

- SAML 2.0, OpenID Connect, Google Workspace, Okta, Microsoft Entra ID — placeholders for upcoming providers. Click "Notify me when this ships" on whichever you want; we'll email you the moment it goes live.

Click **Configure** on the JWT card.

Step 2 — Configure the JWT form

The form has three required fields and several optional ones.

Required

- **Login URL.** The URL on your side we should redirect visitors to when they need to authenticate. Example: https://app.yourcompany.com/auth/helpcenter. Your endpoint must accept a ?subdomain=...&key=... redirect from us, authenticate the user, mint a JWT, and 302 back to https://<your-help-center>/sso/jwt?jwt=<token>.

- **Shared Secret.** Click **Generate** to mint a 64-character base64 secret, or paste your own. **This signs every JWT you mint.** Treat it like an API key — never ship it to the browser. If you ever need to rotate it, click **Regenerate** (we'll prompt to confirm — generating a new secret immediately invalidates any token signed with the old one).

Optional

- **Logout URL.** Where to send visitors after they log out of the help center. Defaults to your help center root.

- **Issuer (**iss **claim).** If set, JWTs whose iss doesn't match are rejected. Useful when one secret is shared across multiple sites and you want to make sure tokens from one can't be accepted by another.

- **Audience (**aud **claim).** Same idea, different claim. Useful when you want one issuer to sign tokens for multiple downstream consumers and the help center should accept only its own.

- **Token TTL (seconds).** Hard cap on now - iat applied by the widget surface, even if exp in your token is more generous. Defaults to **300 seconds**. We recommend keeping this short — it's defense in depth against a leaked token.

Click **Apply JWT Settings**.

Step 3 — Mint a JWT from your backend

Required claims:

Claim	Type	Notes
jti	string	Unique per token. Used for replay protection.
iss	string	Issuer. Must match the Issuer you configured if you set one.
iat	int	Issued-at, Unix seconds.
exp	int	Expiry, Unix seconds. **Now required.** Keep short.
email	string	The user's email.

name	string	The user's display name.

Optional: external_id, lang, avatar_url, role (viewer / editor / admin), custom_fields, aud.

Node example

```
const jwt = require('jsonwebtoken');

function mintHelpcenterJwt(user) {
    return jwt.sign({
        jti: `${user.id}.${Date.now()}.${Math.random().toString(36).slice(2)}`,
        iss: 'app.yourcompany.com',
        iat: Math.floor(Date.now() / 1000),
        exp: Math.floor(Date.now() / 1000) + 300,
        email: user.email,
        name: user.fullName,
        external_id: String(user.id),
        role: 'viewer',
    }, process.env.HELPCENTER_SHARED_SECRET, { algorithm: 'HS256' });
}
```

PHP example

```
use Firebase\JWT\JWT;

function mint_helpcenter_jwt(User $user): string
{
    return JWT::encode([
        'jti' => $user->id . '.' . microtime(true),
        'iss' => 'app.yourcompany.com',
        'iat' => time(),
        'exp' => time() + 300,
        'email' => $user->email,
        'name' => $user->name,
```

```
        'external_id' => (string) $user->id,
        'role' => 'viewer',
    ], config('helpcenter.shared_secret'), 'HS256');
}
```

Ruby example

```
require 'jwt'

def mint_helpcenter_jwt(user)
  payload = {
    jti: "#{user.id}.#{Time.now.to_f}",
    iss: 'app.yourcompany.com',
    iat: Time.now.to_i,
    exp: Time.now.to_i + 300,
    email: user.email,
    name: user.name,
    external_id: user.id.to_s,
    role: 'viewer',
  }
  JWT.encode(payload, ENV['HELPCENTER_SHARED_SECRET'], 'HS256')
end
```

Step 4 — Set up the redirect endpoint (for the direct help-center flow)

Your **Login URL** must be a route on your app that:

1. Reads the subdomain and key query params we pass.
2. Authenticates the visitor (your existing login flow).

3. Mints a JWT using the function from Step 3, with jti set to the key we passed (this is the bit that ties together the redirect-and-return).

4. 302s the visitor to https://<subdomain>.helpcenter.io/sso/jwt?jwt=<token> (or https://<your-custom-domain>/sso/jwt?jwt=<token> if you've set up a custom domain).

Node + Express example

```
app.get('/auth/helpcenter', requireLogin, (req, res) => {
    const subdomain = req.query.subdomain;
    const key = req.query.key;

    const token = jwt.sign({
        jti: key,                          // critical — must match the redir
        iss: 'app.yourcompany.com',
        iat: Math.floor(Date.now() / 1000),
        exp: Math.floor(Date.now() / 1000) + 300,
        email: req.user.email,
        name: req.user.fullName,
        external_id: String(req.user.id),
    }, process.env.HELPCENTER_SHARED_SECRET, { algorithm: 'HS256' });

    res.redirect(`https://${subdomain}.helpcenter.io/sso/jwt?jwt=${token}`);
});
```

Step 5 — Set up the embedded widget (for the Smart Widget flow)

If you also embed the widget inside your app, the host snippet picks up the JWT directly — no redirect dance.

```
<script>
    window.hcOptions = {
        app_id: 'YOUR_WIDGET_ID',
        jwt: '<JWT minted server-side and templated in here>',
        onAuthExpired: async function () {
            const resp = await fetch('/helpcenter-token', { credentials: 'inc
            const json = await resp.json();
            return json.jwt;
        },
    };
</script>
<script src="https://helpcenter.io/js/init.js" async></script>
```

The widget sends the JWT on every API call as Authorization: Bearer <jwt>. When the token nears expiry, the widget calls your onAuthExpired and you return a fresh one — no page reload.

Full integration walkthrough: JWT SSO for the Embedded Widget.

Step 6 — Verify

Open your help center in an incognito window. You should be redirected to your Login URL, authenticated on your side, and returned to the help center logged in — no HelpCenter.io login screen at any point.

If something fails, the most informative diagnostic is the widget_jwt.rejected / sso.redirect_flow.rejected log events on our side (visible in your Settings → Logs view). The reason field maps to one of:

Reason	What's wrong
jwt_invalid_signature	Wrong shared secret on either side.

jwt_missing_required_claim	One of jti / iss / iat / exp / email / name is missing or empty.
jwt_expired	exp is in the past beyond the 30-second skew window.
jwt_iat_in_future	iat is more than 30 seconds in the future. Server clock drift.
jwt_too_old	iat is older than the configured Token TTL.
jwt_issuer_mismatch	iss claim doesn't match the configured Issuer.
jwt_audience_mismatch	aud claim doesn't match the configured Audience.
jwt_replayed	This jti has already been used. Mint a fresh one.
sso_not_configured	We hit the route but no shared secret is saved. Open the SSO settings and complete Step 2.

user_banned	The provisioned local user is suspended in our system. Contact support if this is unexpected.

Frequently asked questions