Help Center

What is Atlas?

Atlas is a complete authentication platform you embed in your application in minutes and then stop thinking about. It covers the entire surface of identity:

- Sign-in and sessions — email/password, magic links, passkeys, and 56 social and enterprise SSO connections, all flowing into one identity, backed by rotating, timing-safe session tokens and a signed JWKS your services can verify.
- Multi-factor auth — TOTP, recovery codes and passkey second factors, with step-up when a sensitive path demands it.
- Organizations and RBAC — multi-tenant organizations with roles, granular permissions and invitations, enforced at the data layer.
- Enterprise readiness — SAML single sign-on, SCIM directory sync, custom domains, signed webhooks and a tamper-evident audit log.

Unlike hosted-only incumbents, Atlas can run on your own infrastructure against your own database, and every instance ships with the same security posture on day one — encryption at rest and tenant isolation are never an upsell.

Read more in the documentation.

Atlas is a complete, self-hostable authentication and identity platform. It handles sign-in, sessions, multi-factor auth, organizations, roles and enterprise SSO so you don't have to build any of it yourself.

- How does Atlas compare to Clerk and Auth0?
- How do I add Atlas to my app?
- Is there a free tier?

How does Atlas compare to Clerk and Auth0?

Atlas is designed as an open alternative to Clerk and Auth0. The headline differences:

- You can run it yourself. Clerk and Auth0 are hosted-only. Atlas runs against your own Postgres, on your own domain and infrastructure, so your identity data never has to leave systems you control.
- Security is not tiered. Encryption at rest, tenant isolation, rotating tokens and a full audit trail ship on every instance from day one, rather than being reserved for enterprise plans.
- B2B from your first commit. Multi-tenant organizations, roles and invitations are core, not a late-stage add-on.
- Drop-in developer experience. Typed SDKs and prebuilt components (<SignIn/>, <AtlasProvider/>, hooks) get you to a working sign-in in minutes, the same ergonomics you'd expect from Clerk.
- Enterprise features without a rebuild. SAML, SCIM and enforced SSO are available when a customer asks, without re-architecting.

If you already use Clerk or Auth0, Atlas provides an import path — see the migration guide in the docs.

Atlas gives you Clerk- and Auth0-class capabilities — drop-in UI components, organizations, MFA, SSO — but adds true self-hosting and ships enterprise-grade security on every tier instead of paywalling it.

- How do I migrate from Clerk or Auth0?
- Can I self-host Atlas?
- What is Atlas?

Can I self-host Atlas?

Self-hosting is a first-class deployment mode, not an afterthought. You run Atlas against your own Postgres database, on your own domain, on infrastructure you control. Every secret — password verifiers, provider credentials, session tokens — is wrapped with per-instance envelope keys and encrypted at rest, exactly as in the managed offering.

The same security posture ships to every instance on day one; there is no tier where isolation or encryption becomes an upsell. This is the key difference from Clerk and Auth0, which are hosted-only.

See the self-hosting guide in the docs for deployment topologies (Docker Compose and container orchestration are both supported).

Yes. Atlas is self-hostable by design — point it at your own Postgres, bring your own domain, and run it on your own infrastructure. Your identity data never has to leave systems you control.

- What are the requirements to self-host Atlas?
- Who owns my data?
- How does Atlas compare to Clerk and Auth0?

What are the requirements to self-host Atlas?

A self-hosted Atlas deployment consists of a few services:

- API service — the core authentication and management API.
- Background worker — handles asynchronous jobs: webhook delivery, SCIM provisioning forwarding, data-subject-request processing, log-stream forwarding and maintenance tasks.
- Web frontend — the admin dashboard and hosted sign-in pages.

Requirements:

- PostgreSQL for primary storage.
- A container runtime — Docker Compose works out of the box; any container orchestrator is fine.
- A domain and TLS certificate for your instance's endpoints.
- Secrets for the envelope-encryption master key and provider credentials.

Deploys do not auto-migrate — you run migrations as an explicit step during upgrades. Full environment reference and a launch runbook are in the docs.

You need a PostgreSQL database, a container runtime (Docker/Compose or a Kubernetes-style orchestrator), and a domain with TLS. Atlas runs as an API service, a background worker, and the web/hosted-page frontends.

- Can I self-host Atlas?
- How do updates and upgrades work when self-hosting?
- How is my data encrypted at rest?

Who owns my data with Atlas?

Your data is yours. When you self-host, all identity data lives in your own PostgreSQL database — Atlas never phones it home. In the managed offering, your data lives in an instance isolated from every other customer.

Either way:

- Export any time. Users and configuration can be exported through the API and the Import/Export tools in the dashboard.
- Encrypted at rest. Secrets are wrapped with per-instance envelope keys, so they aren't readable even at the database layer.
- Deletable. You can delete users and honor data-subject requests through built-in tooling.

For contractual guarantees, a Data Processing Agreement (DPA) is available on Enterprise. See the docs and the DPA.

You do. Your users, sessions and configuration live in your database (self-hosted) or your isolated instance (managed). You can export it at any time, and secrets are encrypted so even at the storage layer they aren't readable.

- Where is my data stored, and can I choose the region?
- Is Atlas GDPR compliant?
- Can I self-host Atlas?

How do updates and upgrades work when self-hosting?

Self-hosted upgrades are deliberate and under your control:

- Pull the new images for the API, worker and web services.
- Run migrations explicitly. Deploys do not auto-migrate — this is intentional so schema changes never happen behind your back during a rolling restart.
- Restart the services.

Because envelope keys are per-instance, key rotation and re-encryption are supported as maintenance operations run by the worker. Follow the release notes and the upgrade runbook in the docs for any version-specific migration steps, and watch the changelog.

You pull the new container images and run database migrations as an explicit step. Upgrades don't auto-migrate, so you stay in control of when schema changes are applied.

- What are the requirements to self-host Atlas?
- How is my data encrypted at rest?
- Can I self-host Atlas?

Is there a free tier, and how much does Atlas cost?

Atlas is free while you build and fair when you grow. There are three plans:

- Free — $0. 10,000 monthly active users (MAU), all 56 SSO providers, passkeys, MFA and magic links, hosted pages and every SDK. The security that matters is never behind a paywall.
- Pro — $25/mo + usage. Everything in Free, plus organizations and RBAC, custom JWT claims and templates, webhooks, audit log and custom domains. Comes with a 14-day trial.
- Enterprise — custom. SAML and enforced SSO, a 99.99% uptime SLA, data residency and a signed DPA, and dedicated support.

You only upgrade when your user base does. See current details on the pricing page.

Yes. The Free plan is $0 and includes 10,000 monthly active users, all 56 SSO providers, passkeys, MFA and magic links, hosted pages and every SDK. Pro is $25/mo plus usage; Enterprise is custom.

- What happens if I exceed my free MAU limit?
- How do I upgrade or downgrade my plan?
- How does per-app billing work?

How do I sign up and create my first instance?

- Create an account at atlasauth.net — no credit card required for the Free plan.
- Create an instance. Each application you build gets its own Atlas instance, which isolates its users, settings and keys from every other instance.
- Copy your API keys from the dashboard. Every instance has a publishable key (pk_live_…, safe to ship in a browser) and a secret key (sk_live_…, server-only).
- Configure sign-in methods — enable the social providers, passkeys, magic links or password sign-in you want. All 56 providers are available on every plan.
- Drop the SDK into your app and you have a working sign-in.

Full walkthrough in the quickstart.

Sign up at atlasauth.net, create an instance (each app gets its own), and grab your publishable and secret API keys from the dashboard. You're ready to integrate in a couple of minutes.

- How do I add Atlas to my app?
- What's the difference between publishable and secret keys?
- Which sign-in methods does Atlas support?

How do I add Atlas to my app?

On the frontend, the entire integration is a provider plus a component. In React:

```
import { AtlasProvider, SignIn, useUser } from '@atlas/react';

function App() {
  const { user, isLoaded } = useUser();
  if (!isLoaded) return <Spinner />;
  if (!user) return <SignIn />;
  return <Dashboard user={user} />;
}

<AtlasProvider publishableKey="pk_live_…">
  <App />
</AtlasProvider>
```

That gives you sign-in, sign-up, sessions and the current user with no token plumbing of your own. On the backend, verify the session token your frontend sends with your secret key and you get the authenticated user, organization and permissions.

The same model is available for Next.js, Vue, Nuxt, Angular, Svelte, React Native and vanilla JS. See the framework guides at https://developers.atlasauth.net.

Wrap your app in <AtlasProvider> with your publishable key, then use the prebuilt <SignIn/> component and the useUser() hook. That's the whole frontend integration — usually under 15 minutes.

- Which languages and frameworks have SDKs?
- How do I verify a session on my backend?
- How do I sign up and create my first instance?

How is my data encrypted at rest?

Atlas uses envelope encryption with per-instance keys. Each instance has its own data-encryption keys, and every sensitive value — password verifiers, OAuth/social provider client secrets, session and refresh tokens, and other credentials — is encrypted with them before it is stored.

The practical consequences:

- A leak of the database alone does not expose secrets, because they are ciphertext at rest.
- Keys are scoped per instance, so one instance's keys can never decrypt another's data.
- Key rotation and re-encryption are supported as maintenance operations.

This posture is identical on every plan and in self-hosted deployments — encryption is the substrate, not a feature tier. More in the security overview.

Every secret — password verifiers, provider credentials, session tokens — is wrapped with per-instance envelope keys and encrypted at rest, so it is never readable in plaintext at the storage layer.

- How does Atlas isolate one tenant's data from another?
- How secure are sessions and tokens?
- Who owns my data?

How does Atlas isolate one tenant's data from another?

Tenant isolation in Atlas is enforced at the repository layer — the lowest data-access layer — rather than relying on every route to remember to filter correctly.

- Every query is scoped to its instance, and within an instance, organization-scoped data is scoped to its organization.
- Because the check lives beneath the application logic, there is no route that can accidentally read across a customer or tenant boundary.
- This is why Atlas advertises 100% tenant isolation: it is a structural guarantee, not a per-endpoint convention.

For B2B apps this means organizations are safely multi-tenant from your first commit. See the organizations guide.

Instance and tenant scoping is enforced at the repository (data-access) layer, not just in application logic — so no request can ever read across a customer or organization boundary, by construction.

- How do organizations (multi-tenancy) work?
- How is my data encrypted at rest?
- Is Atlas SOC 2 compliant?

Where is my data stored, and can I choose the region?

It depends on your deployment:

- Self-hosted: your data lives entirely in your own PostgreSQL database, in whatever region and jurisdiction you choose. Atlas never transmits it elsewhere.
- Managed: your data lives in an instance isolated from all other customers. Data residency — pinning storage to a specific region — is offered on the Enterprise plan, alongside a signed DPA.

If you have specific residency or sovereignty requirements, contact sales to confirm the current regions and guarantees. See the data residency notes and the DPA.

Self-hosted, your data lives wherever you run Postgres. On the managed platform, data lives in your isolated instance; region pinning and data-residency guarantees are available on Enterprise.

- Who owns my data?
- Is Atlas GDPR compliant?
- Can I self-host Atlas?

Is Atlas SOC 2 and GDPR compliant?

Atlas is engineered to support your compliance obligations:

- GDPR: built-in data-subject-request (DSR) tooling for access and erasure, user export/delete, encryption at rest, a full audit trail, and a Data Processing Agreement (DPA) available on Enterprise. Data residency can be pinned by region on Enterprise.
- SOC 2 / general controls: encryption at rest, enforced tenant isolation, a partitioned tamper-evident audit log, granular access control (RBAC), rate limiting and account lockout give you the technical controls auditors look for. When you self-host, the operational controls are yours to attest to.

For the current list of formal attestations and to request our compliance documentation and DPA, please contact sales. We describe our security controls in detail in the security overview.

Atlas is built with the controls compliance programs require — encryption at rest, strict tenant isolation, a tamper-evident audit log, access controls and data-subject-request tooling. A DPA and data residency are available on Enterprise; contact us for current attestations.

- Where is my data stored, and can I choose the region?
- What is your security disclosure and breach policy?
- How is my data encrypted at rest?

How secure are sessions and tokens?

Session security is one of the hardest parts of auth, and Atlas handles it at the core:

- Rotating tokens with reuse detection — a replayed or stolen refresh token is caught and the session is invalidated.
- Constant-time (timing-safe) verification to prevent timing side-channels.
- A signed JWKS endpoint so your backend services can verify session JWTs cryptographically without a network round-trip.
- No tokens in URLs — sign-in and verification use one-time tickets, so tokens don't leak into browser history, logs or referrer headers.
- Exact-origin redirect checks and CSRF-bound OAuth state to close open-redirect and CSRF holes.

See session and token security for the full model.

Session tokens are rotating and timing-safe, with reuse detection and constant-time verification, and Atlas publishes a signed JWKS so your services can verify tokens cryptographically. Tokens never travel in URLs.

- My JWT/session token fails to verify — what's wrong?
- Does Atlas support MFA?
- How is my data encrypted at rest?

Does Atlas support multi-factor authentication (MFA)?

Atlas has full MFA support:

- TOTP — standard authenticator-app codes (Google Authenticator, 1Password, Authy, etc.).
- Recovery / backup codes — one-time codes for when a device is lost.
- Passkey second factors — WebAuthn as a second factor.
- Per-session freeze — repeated failed factor attempts freeze that session rather than letting an attacker grind.
- Step-up authentication — Atlas can require re-verification before a sensitive action, enforced by the server on every path, including OAuth.
- Adaptive MFA — challenge only when signals (new device, unusual location) indicate risk, so low-risk sign-ins stay frictionless.

Configure MFA policy in the dashboard; see the MFA guide.

Yes — TOTP authenticator apps, recovery/backup codes and passkey second factors, with per-session freeze on repeated failures and step-up on sensitive paths. Adaptive MFA can require a factor only when risk warrants it.

- Do you support passkeys / WebAuthn?
- How secure are sessions and tokens?
- Which sign-in methods does Atlas support?

What is your security disclosure and breach policy?

Responsible disclosure. If you believe you've found a security vulnerability, please report it privately to our security team rather than disclosing publicly. Email the address listed on our security page; we aim to acknowledge reports promptly and will keep you updated through remediation.

What Atlas gives you for your own incident response:

- A partitioned, tamper-evident audit log of security-relevant events (sign-ins, factor changes, role and permission changes, admin actions).
- Signed webhooks for every event, so you can stream security events into your own SIEM in real time via log streams.

Breach notification. For managed and Enterprise customers, notification timelines and responsibilities are set out in the DPA. Self-hosted operators control their own detection and notification, using the audit log and log streams as the evidentiary record.

Contact sales for the DPA and our security documentation.

We welcome responsible disclosure of vulnerabilities and aim to acknowledge reports promptly. Atlas keeps a tamper-evident audit log and signed webhooks so you have a verifiable record of security-relevant events for your own incident response.

- Is Atlas SOC 2 and GDPR compliant?
- My webhooks aren't arriving — how do I debug?
- How secure are sessions and tokens?

Which sign-in methods does Atlas support?

Atlas supports the full range of sign-in methods, all feeding one unified identity:

- Password — with breach-resistant verifiers stored encrypted at rest.
- Magic links — passwordless email sign-in.
- Passkeys / WebAuthn — phishing-resistant biometric sign-in.
- One-time codes — email (and SMS where configured).
- Social & enterprise SSO — 56 connections including Google, GitHub, Microsoft, Apple, Slack, Discord, GitLab, LinkedIn, Facebook, Twitch, X, Spotify, Steam and many more.
- SAML — enterprise single sign-on for your B2B customers.

You choose which methods to enable per instance in the dashboard. See the authentication methods guide.

Email/password, magic links, passkeys, one-time codes, 56 social and enterprise SSO connections, and SAML. You enable whichever methods you want per instance; they all resolve to a single identity.

- Which social login providers are supported?
- Do you support passkeys / WebAuthn?
- Do you support SSO and SAML?

Which social login providers are supported?

Atlas ships 56 social and enterprise SSO connections, and all of them are available on every plan — including Free. The lineup includes:

Google, GitHub, Microsoft, Apple, Slack, Discord, GitLab, LinkedIn, Facebook, Twitch, X (Twitter), Spotify, Notion, Figma, Salesforce, Shopify, Steam, and dozens more.

Each provider is an OAuth 2.0 / OIDC connection you enable and configure (client ID and secret) in the dashboard. Once enabled, the provider appears automatically in your hosted sign-in and in the SDK components — no code change. All provider credentials are encrypted at rest.

See the full connections catalog.

56 social and enterprise connections, including Google, GitHub, Microsoft, Apple, Slack, Discord, GitLab, LinkedIn, Facebook, Twitch, X, Spotify, Steam and Salesforce — all available on every plan, including Free.

- Which sign-in methods does Atlas support?
- A social login redirect fails with a redirect_uri error — how do I fix it?
- Do you support SSO and SAML?

Do you support passkeys and WebAuthn?

Passkeys (WebAuthn) are fully supported, both as a primary sign-in method and as a second factor for MFA. Passkeys are phishing-resistant because the credential is bound to your origin and never leaves the user's device.

- Available in the hosted sign-in pages and every frontend SDK component out of the box.
- Users can register multiple passkeys (phone, laptop, hardware key).
- No extra infrastructure required — Atlas handles the WebAuthn ceremony, challenge and credential storage.

See the passkeys guide.

Yes. Passkeys are a first-class, phishing-resistant sign-in method and can also be used as a second factor. They work across the SDKs and hosted pages with no extra infrastructure.

- Does Atlas support MFA?
- Do you support magic links?
- Which sign-in methods does Atlas support?

Do you support magic links?

Magic links are supported as a passwordless sign-in method. A user enters their email, receives a one-time link, and clicking it signs them in.

Security details worth knowing:

- Links use one-time tickets — they can't be replayed after use.
- No session token ever travels in the URL, so it can't leak through browser history, referrer headers or server logs.
- Links are time-limited and origin-checked.

You can customize the email template and sender in the dashboard's messaging settings. See the magic links guide.

Yes. Magic links let users sign in by clicking a one-time link sent to their email — no password. The links use one-time tickets and never carry a session token in the URL.

- Do you support passkeys / WebAuthn?
- Do you support SMS / phone sign-in?
- Which sign-in methods does Atlas support?

Do you support SMS or phone-based sign-in?

Atlas supports SMS one-time codes for sign-in and verification. Because SMS requires a delivery gateway, you configure an SMS provider in the dashboard's messaging settings; once connected, SMS codes are available in your sign-in flows and can serve as a delivery channel for verification.

If you'd rather avoid SMS (cost, deliverability, SIM-swap risk), Atlas's email one-time codes, magic links and passkeys give you passwordless options with no external provider to configure.

See the messaging and SMS guide.

Yes — Atlas supports one-time codes over SMS as a sign-in and verification method when you configure an SMS provider. Email one-time codes and magic links work out of the box with no provider setup.

- Do you support magic links?
- Does Atlas support MFA?
- Which sign-in methods does Atlas support?

Do you support SSO and SAML?

Atlas provides enterprise single sign-on:

- SAML — connect a customer's identity provider (Okta, Entra ID / Azure AD, Google Workspace, OneLogin, etc.) so their employees sign in through it. Atlas can act as a SAML service provider, and includes SAML IdP capabilities for the reverse direction.
- OIDC enterprise connections — for IdPs that speak OpenID Connect.
- Enforced SSO — require members of an organization to authenticate via their configured SSO connection (Enterprise plan).

SSO connections are configured per organization, so each B2B customer can bring their own IdP. See the SSO/SAML guide.

Yes. Atlas supports enterprise single sign-on including SAML, so your B2B customers can log in through their own identity provider. Enforced SSO (requiring org members to use SSO) is available on Enterprise.

- Do you support SCIM directory sync?
- How do organizations (multi-tenancy) work?
- Which social login providers are supported?

Do you support SCIM directory sync?

Atlas implements SCIM 2.0 for automated directory synchronization. An enterprise customer's IdP (Okta, Entra ID, etc.) can create, update, deactivate and delete users — and sync groups — directly into their Atlas organization, with no manual user management.

- Per-organization SCIM tokens control access.
- Provisioning events are forwarded by the background worker and appear in the audit log.
- Pairs naturally with SAML enforced SSO for a complete enterprise onboarding/offboarding lifecycle.

Configure SCIM in the organization's settings. See the SCIM guide.

Yes. Atlas supports SCIM 2.0 so enterprise customers can automatically provision and de-provision users and groups from their identity provider into your app.

- Do you support SSO and SAML?
- How do organizations (multi-tenancy) work?
- Is Atlas SOC 2 and GDPR compliant?

How do organizations (multi-tenancy) work?

Organizations are how Atlas models B2B multi-tenancy:

- Membership — users join one or more organizations. The active organization is part of the session, and appears in the JWT so your backend knows the tenant context.
- Isolation — org-scoped data is scoped at the repository layer, so no request can read across organizations.
- Per-org configuration — each organization can have its own roles, invitations, SSO connection (SAML/OIDC), SCIM provisioning, domains and policies.
- Org switching — the SDKs expose the user's organizations and let them switch the active one.

This makes Atlas suitable for B2B SaaS from day one, not as a bolt-on. See the organizations guide.

Organizations are Atlas's multi-tenant primitive. A user can belong to one or more organizations; all org-scoped data is isolated at the data layer, and each org can have its own roles, members, SSO and settings.

- How do roles and permissions (RBAC) work?
- How do I invite users to an organization?
- Can a user belong to multiple organizations?

How do roles and permissions (RBAC) work?

Atlas's RBAC lets you control who can do what:

- Permissions — granular capabilities you define (for example billing:read, members:invite).
- Roles — named bundles of permissions (for example admin, member, viewer).
- Assignment — members are given roles within an organization, so the same user can be an admin in one org and a viewer in another.
- Enforcement — the user's roles and permissions are included in the session and JWT. The SDKs expose helpers (like has({ permission })) for the frontend, and your backend can check the same claims.

For relationship-based, fine-grained authorization beyond roles (for example per-document access), Atlas also offers fine-grained authorization (FGA). See the RBAC guide.

Atlas has role-based access control: you define roles, each role grants a set of granular permissions, and members are assigned roles per organization. Permissions surface in the session so your app and backend can enforce them.

- How do organizations (multi-tenancy) work?
- How do I invite users to an organization?
- How do I verify a session on my backend?

How do I invite users to an organization?

Inviting a member is straightforward:

- From the organization's Members view (or via the API/SDK), create an invitation with the invitee's email and the role to grant.
- Atlas emails them an invitation link.
- When they accept, they're added to the organization with the assigned role — creating an account first if they don't have one.

Invitations are auditable, can be revoked before acceptance, and respect the organization's SSO settings. See the invitations guide.

Send an invitation from the dashboard or via the API/SDK with the invitee's email and the role they should get. They receive an email, accept, and land in the organization with that role already assigned.

- How do roles and permissions (RBAC) work?
- How do organizations (multi-tenancy) work?
- Can a user belong to multiple organizations?

Can a user belong to multiple organizations?

A user has one identity but can belong to any number of organizations. This is important for real B2B products where the same person might be an admin of their own company and a guest in a partner's org.

- Each membership carries its own roles and permissions.
- The session and JWT include the active organization, which your backend uses as tenant context.
- The SDKs list a user's organizations and expose an org switcher so they can change the active one.

See the organizations guide.

Yes. A single Atlas identity can be a member of many organizations, each with its own roles. The session carries the active organization, and users can switch between the orgs they belong to.

- How do organizations (multi-tenancy) work?
- How do roles and permissions (RBAC) work?
- How do I verify a session on my backend?

Which languages and frameworks have SDKs?

One identity model, a first-class SDK for wherever you meet it:

- Frontend (8): vanilla JavaScript, React, Next.js, Vue, Nuxt, Angular, Svelte, React Native. Each is idiomatic — hooks and context in React, composables in Vue/Nuxt, stores in Svelte.
- Backend (7): Node/TypeScript, Python, Go, Ruby, PHP, Java, .NET. Each verifies the session token your frontend sends and hands you the authenticated user, organization and permissions.
- Mobile (3): Swift (iOS, Keychain-backed), Kotlin (Android), Flutter (cross-platform) — same sign-in, sessions and passkeys, with secure token storage.
- Infrastructure: a Terraform provider for managing Atlas as code, plus an MCP server.

A user, organization, role and permission mean the same thing in every SDK. See the framework guides at https://developers.atlasauth.net.

Atlas ships typed SDKs across frontend (JS, React, Next.js, Vue, Nuxt, Angular, Svelte, React Native), backend (Node, Python, Go, Ruby, PHP, Java, .NET), and mobile (Swift, Kotlin, Flutter), plus a Terraform provider.

- What's the difference between publishable and secret keys?
- How do I verify a session on my backend?
- Do you have mobile SDKs?

What's the difference between publishable and secret keys?

Atlas uses two kinds of keys, and the split is enforced by construction, not by convention:

- Publishable key (pk_live_…) — used by frontend and mobile SDKs. Safe to embed in a browser bundle because it is scoped to exactly the public operations a logged-out visitor could already perform. If it leaks, nothing privileged leaks with it.
- Secret key (sk_live_…) — used by backend SDKs only. Stays on your server. Grants privileged, server-side operations (managing users, verifying sessions, administering organizations).

Both keys are scoped to a single instance, so a key can never touch another instance's data. Client SDKs are simply incapable of privileged operations. Manage and rotate keys in the dashboard's API Keys view. See the keys guide.

Publishable keys (pk_live_…) are safe to ship in browser/mobile code and can only do what a logged-out visitor could. Secret keys (sk_live_…) stay on your server and grant privileged operations. Both are scoped to one instance.

- Which languages and frameworks have SDKs?
- How do I verify a session on my backend?
- How do I add Atlas to my app?

How do I verify a session on my backend?

Your frontend sends the session token with each request; your backend verifies it. With a backend SDK it's one call. In Go:

```
session, err := atlas.Authenticate(r)
if err != nil {
    http.Error(w, "unauthorized", http.StatusUnauthorized)
    return
}
userID := session.User.ID
orgID := session.OrgID
```

The SDK verifies the token cryptographically against Atlas's signed JWKS (cached, so it's fast and works offline of the API), checks expiry and rotation, and gives you the user, active organization and permissions. The same helper exists in Node, Python, Ruby, PHP, Java and .NET.

If you're verifying JWTs manually (no SDK), fetch the JWKS from your instance's well-known endpoint and validate the signature, iss, aud and exp. See the backend guide.

Use a backend SDK with your secret key: call its authenticate helper on the incoming request. It verifies the session token (against the signed JWKS) and returns the authenticated user, organization and permissions.

- My JWT/session token fails to verify — what's wrong?
- What's the difference between publishable and secret keys?
- How do roles and permissions (RBAC) work?

How do I migrate from Clerk or Auth0?

Migrating from Clerk or Auth0 has three parts:

- Import users. Use the Import/Export tools to bring in your existing users. Password hashes can be imported when the hashing algorithm is compatible, so users don't have to reset passwords; otherwise you can trigger a set-password or magic-link flow on first sign-in.
- Map your model. Recreate roles/permissions and organizations, or import them. Atlas's organization + RBAC model maps cleanly onto Clerk organizations and Auth0 tenants/roles.
- Swap the SDK. Replace the Clerk/Auth0 provider with <AtlasProvider> and update your keys. The component and hook surface is intentionally familiar.

Plan the cutover so social/SSO connections are configured before you flip. See the migration guide.

Atlas provides an import path: bring your users (including password hashes where the algorithm is compatible) and map your existing roles and organizations. You swap the SDK provider and keys, and cut over sign-in.

- How does Atlas compare to Clerk and Auth0?
- Which languages and frameworks have SDKs?
- How do I add Atlas to my app?

Do you have mobile SDKs?

Atlas ships three native mobile SDKs:

- Swift (iOS) — session tokens stored in the iOS Keychain.
- Kotlin (Android) — tokens stored in EncryptedSharedPreferences.
- Flutter (cross-platform) — tokens stored via flutter_secure_storage.

All three expose the same AtlasClient surface — sign-in, sign-up, session management and passkey support — so behavior is consistent across platforms and matches the web SDKs' identity model. Mobile clients use a publishable key, just like the web. See the mobile guides at https://developers.atlasauth.net.

Yes — native SDKs for Swift (iOS), Kotlin (Android) and Flutter (cross-platform). They bring the same sign-in, sessions and passkey support to mobile, with tokens kept in the platform's secure storage.

- Which languages and frameworks have SDKs?
- Do you support passkeys / WebAuthn?
- What's the difference between publishable and secret keys?

What plans are available?

Three plans:

Plan	Price	Highlights
Free	$0	10,000 MAU, all 56 SSO providers, passkeys, MFA, magic links, hosted pages, all SDKs
Pro	$25/mo + usage	Everything in Free, plus organizations & RBAC, custom JWT claims & templates, webhooks, audit log, custom domains (14-day trial)
Enterprise	Custom	SAML & enforced SSO, 99.99% uptime SLA, data residency & DPA, dedicated support

The core security features — encryption, isolation, rotating tokens, audit — are on every plan. See the pricing page.

Free ($0), Pro ($25/mo + usage) and Enterprise (custom). Free includes 10,000 MAU, all 56 SSO providers, passkeys, MFA and all SDKs. Pro adds organizations, RBAC, custom JWT and webhooks. Enterprise adds SAML, SLA, residency and a DPA.

- How do I upgrade or downgrade my plan?
- What happens if I exceed my free MAU limit?
- How does per-app billing work?

How do I upgrade or downgrade my plan?

Manage your plan in the dashboard's Billing section:

- Upgrade — switch to Pro (starts with a 14-day trial) or start an Enterprise conversation. Upgrades apply immediately so you get the new features right away.
- Downgrade — takes effect at the end of your current billing period, so you keep access to what you already paid for until it ends.
- Invoices and payment method are managed in the same section, and you'll get renewal reminders and payment-failure notices by email.

For Enterprise terms, contact sales.

Change plans from the Billing section of the dashboard. Upgrades take effect immediately (Pro starts with a 14-day trial); downgrades take effect at the end of the current billing period so you keep what you paid for.

- What plans are available?
- How does per-app billing work?
- What happens if I exceed my free MAU limit?

My JWT or session token fails to verify — what's wrong?

Session/JWT verification failures almost always come down to one of these:

- Wrong instance or key. The token is signed by one instance but you're verifying against another's JWKS. Make sure your backend uses the same instance's keys as the frontend that issued the token.
- Stale JWKS cache. If you rotated signing keys, a cached JWKS can miss the new key. Refresh the JWKS (SDKs do this automatically on an unknown kid).
- Clock skew. If your server's clock is off, a valid token can look exp-expired or nbf-not-yet-valid. Sync via NTP and allow a small leeway.
- Wrong aud / iss. Confirm you're validating the audience and issuer that your instance actually sets.
- Using the wrong token. Verify the session/access token, not a refresh token.

The backend SDKs handle JWKS fetching, caching and rotation for you — prefer them over hand-rolled verification. See session verification.

The usual causes are a JWKS/signing-key mismatch (wrong instance or stale cache), a clock skew making the token look expired, or checking the wrong aud/iss. Verify against your instance's JWKS and confirm the claims.

- How do I verify a session on my backend?
- How secure are sessions and tokens?
- What's the difference between publishable and secret keys?

How does per-app (per-instance) billing work?

Atlas bills per instance, and each app you build gets its own instance. That means:

- One app can be on Free while another is on Pro or Enterprise.
- MAU limits and usage are counted independently per instance.
- Keys, users, settings and billing are all scoped to the instance, so there's no bleed between apps.

This lets you keep a side project on Free indefinitely while your production app runs on a paid plan. Manage each instance's plan from its own Billing section. See the billing docs.

Each application has its own Atlas instance, and plans and usage are tracked per instance. You can run one app on Free and another on Pro; billing and MAU limits apply independently to each.

- What plans are available?
- How do I upgrade or downgrade my plan?
- What happens if I exceed my free MAU limit?

What happens if I exceed my free MAU limit?

The Free plan covers 10,000 monthly active users (MAU) — a user counts as active in a month if they authenticate at least once. As you approach the limit, Atlas notifies you so there are no surprises.

To grow beyond 10,000 MAU you upgrade to Pro ($25/mo plus usage), which scales with your active users. Enterprise offers custom volume terms.

Atlas is designed so you only pay when your user base grows — the security features stay the same across plans. See the pricing page and billing docs.

The Free plan includes 10,000 monthly active users. As you approach the limit you'll be notified; to grow beyond it you upgrade to Pro, which continues on a usage basis. Your users are never locked out mid-month by surprise.

- What plans are available?
- How do I upgrade or downgrade my plan?
- How does per-app billing work?

My webhooks aren't arriving — how do I debug?

Work through these in order:

- Endpoint reachability. Your URL must be publicly reachable over HTTPS and return a 2xx promptly. Slow responses can time out; do heavy work asynchronously after acknowledging.
- Signature verification. Atlas signs every webhook. If you compute the signature over the wrong payload (e.g. a re-serialized body instead of the raw bytes) it won't match, and you may be rejecting valid events. Verify against the raw request body.
- Delivery logs. The dashboard shows each delivery attempt, the response code, and retries. Failed deliveries are retried with backoff — check there before assuming an event never fired.
- Event subscription. Confirm your endpoint is subscribed to the event types you expect.
- Log streams. For high-volume debugging, forward events to your own sink via log streams.

See the webhooks guide.

Check that your endpoint returns a 2xx quickly, verify the webhook signature correctly (a signature mismatch looks like a silent failure), and inspect delivery attempts and retries in the dashboard's webhook logs.

- Why am I getting rate-limited (429)?
- What is your security disclosure and breach policy?
- How do I verify a session on my backend?

Why am I getting rate-limited (429 responses)?

A 429 Too Many Requests means you've hit a rate limit. Atlas rate-limits every endpoint per instance to protect against abuse, and returns standard headers so you can adapt:

- Retry-After — how long to wait before retrying.
- RateLimit-* headers — your remaining budget and reset window.

To fix it:

- Honor Retry-After and use exponential backoff on retries.
- Cache things like the JWKS instead of fetching per request (the SDKs do this).
- Batch management operations rather than looping one call per item.
- Use the right key — heavy server-side work should go through the backend SDK/secret key, not the frontend.

Sensitive auth endpoints also apply account lockout after repeated failures, which can look like rate limiting. See rate limits.

Atlas applies per-instance rate limits with standard rate-limit headers. A 429 means you've exceeded the limit for that endpoint — back off using the Retry-After header, and batch or cache where you can.

- My webhooks aren't arriving — how do I debug?
- How secure are sessions and tokens?
- How do I verify a session on my backend?

A social login redirect fails with a redirect_uri error — how do I fix it?

redirect_uri_mismatch (and similar) errors come from the OAuth provider rejecting the callback URL because it doesn't exactly match what you registered in that provider's console.

Checklist:

- Exact match. Scheme (https), host, port and path must match character-for-character — including or excluding a trailing slash consistently.
- Use Atlas's callback URL. Copy the callback URL shown in the connection's settings in the Atlas dashboard and paste it into the provider's allowed redirect URIs. Don't hand-type it.
- Custom domains. If you use a custom domain, register the callback for that domain, not the default one.
- Credentials. Confirm the client ID/secret in Atlas match the provider app you registered the redirect on.

Atlas enforces exact-origin redirect checks and CSRF-bound state on its side, so also make sure you aren't tampering with the state parameter. See the connections/OAuth guide.

The redirect/callback URL registered with the social provider must exactly match the one Atlas uses, including scheme, host and path. A trailing-slash or http-vs-https mismatch is the most common cause.

- Which social login providers are supported?
- Why am I getting rate-limited (429)?
- Which sign-in methods does Atlas support?