API Reference

SDKs

SDKs

Atlas ships official SDKs across three tiers. Pick the frontend SDK for your framework, the backend SDK for your server language, and (for native apps) a mobile SDK.

Tier	What it does	Credential
Frontend	Mount the provider, render sign-in/account UI, read the user, call getToken()	Publishable key pk_…
Backend	Manage the instance over the BAPI; verify session JWTs locally	Secret key sk_… + JWKS
Mobile	Native sign-in and session storage over FAPI	Publishable key pk_…

Frontend (8)

@atlas/js (framework-agnostic core) · @atlas/react · @atlas/nextjs · @atlas/vue · @atlas/nuxt · @atlas/angular · @atlas/svelte · @atlas/react-native

Backend (7)

Node (@atlas/backend) · Python (atlas-backend) · Go (atlas-go) · Ruby (atlas-auth) · PHP (atlas-auth/atlas-php) · Java (net.atlasauth:atlas-java) · .NET (Atlas.Sdk)

Mobile (3)

Swift (iOS/macOS) · Kotlin/Android (net.atlasauth:atlas-android) · Flutter/Dart (atlas_auth)

The shared contract

Every SDK targets the same API and the same authorization primitive: a condition like { permission: 'org:billing:manage' } evaluates identically on the frontend (has()), on the backend (authenticateRequest().has()), and on mobile. A check written for one surface behaves the same everywhere. The wire is snake_case JSON and the SDKs type it directly rather than inventing a second dialect.

> Generate a client for any other language from the API surface documented in the Backend endpoint index.

Users

Users

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

12 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/users		Search users by email, name or id, filtered by banned, MFA or provider.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users		Create a user with a verified email and an optional password. Owner or admin.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId		Full user profile for an administrator, including private metadata.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/ban		Ban a user and revoke every session they hold.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/unban		Lift a ban. Sessions are not restored.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/unlock		Clear an automated lockout for a user locked out by guessing.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/reset_mfa		Remove every second factor. Requires owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/revoke_sessions		Sign a user out everywhere and bump sessions_version.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/sessions/:sessionId/revoke		Revoke one of a user’s sessions (per-device sign-out).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/impersonate		Mint a single-use impersonation token. Audited by person.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/delete		

Soft-delete a user. PII is scrubbed after 30 days.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/users/:userId/metadata

Edit the public, private and unsafe metadata bags. Refuses non-objects, prototype-polluting keys, oversized and over-nested documents.

Appearance

Appearance

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/appearance		The embeddable widget's public theme + enabled providers/strategies, resolved by publishable key or host. Public only — no secrets, no customCss/customHtml.

Oauth2

Oauth2

OIDC / OAuth2 — client authentication or a bearer access token, per the flow.

18 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /oauth2/authorize		Authorization endpoint. Exact redirect_uri match, PKCE for public clients; authenticates via the hosted session and returns a single-use code.
POST /oauth2/authorize		Authorization endpoint (form post). Exact redirect_uri match, PKCE for public clients; returns a single-use code.
POST /oauth2/authorize/consent		Consent screen submit. Re-derives client, redirect_uri and scope from the signed token and the stored client (never the POST); approve records consent and mints a code, deny redirects access_denied.
POST /oauth2/token		Token endpoint. authorization_code (single-use, PKCE-verified), refresh_token (rotated, theft-detected), client_credentials and device_code grants; mints access, id and refresh tokens.
POST /oauth2/par		Pushed Authorization Request (RFC 9126). Client-authenticated; lodges a validated authorization request and returns a single-use, client-bound request_uri for /authorize.
POST /oauth2/backchannel_authentication		CIBA backchannel authentication (OpenID Client-Initiated Backchannel Auth). Confidential client requests out-of-band user approval; returns an auth_req_id to poll / token with.
POST /oauth2/device_authorization		Device Authorization endpoint (RFC 8628). Client-authenticated; returns a device_code, a user_code and the verification URIs for the "log in on a TV" flow.
GET /oauth2/device		Device verification page (RFC 8628). Requires a hosted session; with a user_code it renders an approve/deny screen carrying a signed, session-bound token, else the code-entry form.

POST /oauth2/device/verify

Device verification submit. Re-derives the device code from the signed session-bound token; approve binds the signed-in user and records consent, deny marks the code denied.

POST /oauth2/register

Dynamic Client Registration (RFC 7591). Gated by the per-instance oauthProvider.dynamicRegistration flag (403 when off); validates https/localhost redirect_uris and returns the new client_id and, for confidential clients, a secret.

GET /oauth2/register/:clientId

Read a dynamically-registered client's configuration (RFC 7592). Authenticated by the registration_access_token issued at registration.

PUT /oauth2/register/:clientId

Update a dynamically-registered client's configuration (RFC 7592), authenticated by its registration_access_token.

DELETE /oauth2/register/:clientId

Delete a dynamically-registered client (RFC 7592), authenticated by its registration_access_token.

GET /oauth2/logout

RP-Initiated Logout end_session_endpoint (OIDC). Verifies the id_token_hint signature, revokes the end-user hosted session, fires Back-Channel and Front-Channel Logout to other RPs, and redirects only to an exact-match registered post_logout_redirect_uri.

POST /oauth2/logout

RP-Initiated Logout end_session_endpoint (form post). Verifies the id_token_hint, revokes the hosted session, notifies other RPs (Back-/Front-Channel Logout), and redirects only to an exact-match registered post_logout_redirect_uri.

GET /oauth2/userinfo

UserInfo endpoint. Returns sub plus profile and email claims filtered by the granted scope, for a Bearer access token; refuses a token whose grant has been revoked.

POST /oauth2/revoke

Token revocation (RFC 7009). Client-authenticated; revokes the grant behind a refresh or access token. Always 200 for a well-formed request; only the owning client revokes anything.

POST /oauth2/introspect

Token introspection (RFC 7662). Client-authenticated; returns active:true with the token metadata for a live, non-revoked token owned by the caller, else {active:false}.

Enterprise SSO connections

Enterprise SSO connections

An SSO connection wires an enterprise customer's IdP (OIDC, SAML, or DiscourseConnect) to an organization in your instance. Users from that IdP are JIT-provisioned into the org on first sign-in, with roles optionally driven by IdP claims.


```
{
  "object": "sso_connection",
  "id": "ssoc_…",
  "organization_id": "org_9f…",
  "type": "saml",
  "status": "active",
  "oidc_issuer": null,
  "oidc_client_id": null,
  "has_secret": false,
  "saml_idp_entity_id": "https://idp.acme.com/…",
  "saml_idp_sso_url": "https://idp.acme.com/sso",
  "has_saml_certificate": true,
  "saml_allow_idp_initiated": true,
  "allowed_domains": ["acme.com"],
  "claim_role_mappings": [{ "claim": "groups", "value": "admins", "roleKey":
"org:admin" }],
  "default_role_id": null,
  "created_at": 1757000000000,
  "updated_at": 1757600000000
}
```


Secrets and certificates are write-only: the IdP oidc_client_secret, saml_idp_certificate, and discourse_secret are stored encrypted and never returned — reads report has_secret / has_saml_certificate markers.

Endpoints

Method & path	Scope	Notes
GET /v1/sso_connections	sso_connections:read	List (never the secret)
POST /v1/sso_connections	sso_connections:write	Create (secret write-only). idempotent
GET /v1/sso_connections/:id	sso_connections:read	Reports has_secret, never the secret
PATCH /v1/sso_connections/:id	sso_connections:write	Update (type immutable; secret re-encrypted only if supplied). idempotent
DELETE /v1/sso_connections/:id	sso_connections:write	Delete + its domain routing. idempotent
GET /v1/sso_connections/:id/saml_metadata	sso_connections:read	

SP EntityDescriptor XML to hand the IdP

Claim !’ role mapping

claim_role_mappings maps an IdP claim value to an org role key, so the IdP drives role assignment. allowed_domains routes users by email domain to this connection.

Example

```
const conn = await atlas.ssoConnections.create({
  organization_id: 'org_9f',
  type: 'saml',
  status: 'active',
  saml_idp_entity_id: 'https://idp.acme.com/…',
  saml_idp_sso_url: 'https://idp.acme.com/sso',
  saml_idp_certificate: '-----BEGIN CERTIFICATE-----…', // write-only
  allowed_domains: ['acme.com'],
  claim_role_mappings: [{ claim: 'groups', value: 'admins', roleKey:
'org:admin' }],
});

// Hand this SP metadata to the customer's IdP admin
const { metadata_xml, acs_url, sp_entity_id } = await
atlas.ssoConnections.samlMetadata(conn.id);
```

Users sign in

End users start enterprise SSO from the frontend with POST /v1/client/sign_ins/sso (OIDC) or POST /v1/client/sign_ins/saml (SAML), resolved by connection id or email domain. The IdP redirects back through the callbacks documented in SAML. To let customers configure their own connection without a dashboard account, see Self-service SSO.

OpenID Connect & OAuth2

OpenID Connect & OAuth2

Atlas is a standards-compliant OpenID Provider — "Sign in with Atlas." Third-party (and first-party) apps you register as OAuth clients run the standard flows against these endpoints. Issuer is your instance host; signing is RS256; PKCE is S256.

Core endpoints

```
GET|POST /oauth2/authorize          !’ authorization endpoint (exact redirect_uri
match, PKCE for public clients); returns a single-use code
POST     /oauth2/authorize/consent  !’ consent screen submit (re-derives client/
scope from the signed token)
POST     /oauth2/token              !’ authorization_code, refresh_token
(rotated, theft-detected), client_credentials, device_code
GET      /oauth2/userinfo           !’ sub + profile/email claims filtered by
granted scope (Bearer access token)
POST     /oauth2/introspect         !’ RFC 7662 token introspection (client-
authenticated)
POST     /oauth2/revoke             !’ RFC 7009 token revocation (client-
authenticated)
```

Grants supported at /oauth2/token: authorization_code (single-use, PKCE-verified), refresh_token (rotated with theft detection), client_credentials, and device_code.

Advanced flows

```
POST /oauth2/par                        !’ Pushed Authorization Request (RFC
9126): lodge a request, get a request_uri
POST /oauth2/backchannel_authentication!’ CIBA (out-of-band user approval);
returns an auth_req_id to poll
POST /oauth2/device_authorization       !’ Device Authorization (RFC 8628):
device_code + user_code + verification URIs
GET  /oauth2/device                     !’ device verification page (approve/deny)
POST /oauth2/device/verify              !’ device verification submit
```

RP-initiated logout

```
GET|POST /oauth2/logout   !’ end_session_endpoint: verifies id_token_hint,
revokes the hosted session,
                           fires Back-/Front-Channel Logout to other RPs,
redirects only to an exact-match
                           registered post_logout_redirect_uri
```

Dynamic Client Registration

See Dynamic Client Registration for POST /oauth2/register (RFC 7591) and the RFC 7592 management endpoints.

Example: authorization code + PKCE

```
1. Redirect the browser to:
   /oauth2/authorize?response_type=code&client_id=…&redirect_uri=…
&scope=openid%20profile%20email
     &code_challenge=…&code_challenge_method=S256&state=…

2. Exchange the returned code:
   POST /oauth2/token
     grant_type=authorization_code&code=…&redirect_uri=…&client_id=…
&code_verifier=…

3. Call your resource server with the access_token, or fetch claims:
   GET /oauth2/userinfo   Authorization: Bearer <access_token>
```

Verify access tokens either by introspection (/oauth2/introspect) or locally against the JWKS.

Apps

Apps

Platform API — a platform secret key (Authorization: Bearer).

13 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/platform/apps	platform:apps:write	'	Provision a new app (instance) owned by the platform: mints its publishable key, signing key and first secret key. The secret is revealed exactly once.
GET /v1/platform/apps	platform:apps:read		List the apps (instances) this platform owns. Never returns any secret.
GET /v1/platform/apps/:id	platform:apps:read		Fetch one owned app. Another platform's app id is 404, never 403. Never a secret.
PATCH /v1/platform/apps/:id	platform:apps:write		Update an owned app's safe config subset (name, allowed_origins). Not owned is 404.
DELETE /v1/platform/apps/:id	platform:apps:write		Disable an owned app: refuse sign-ins and sign-ups via its kill switches. Soft and reversible — never a hard drop of user data. Not owned is 404.
POST /v1/platform/apps/:id/rotate_secret	platform:secrets:write	'	Rotate an owned app's secret key: revoke its prior keys and mint a new one, revealed exactly once. Not owned is 404.
POST /v1/platform/apps/:id/instances	platform:apps:write	'	Provision an additional instance (e.g. development beside production) under an owned app. Returns the new instance id and its first secret key, revealed exactly once. Not owned is 404.
GET /v1/platform/apps/:id/instances			

platform:apps:read

List every instance under an owned app — its development/production siblings. Never returns any secret. Not owned is 404.

POST /v1/platform/apps/:id/instances/:instanceId/rotate_secret

platform:secrets:write
'

Rotate one specific instance's secret under an owned app: revoke its prior keys and mint a new one, revealed once. Not owned is 404.

POST /v1/platform/accountless_apps

platform:apps:write
'

Mint an app owned by no platform and return a one-time claim_token (hashed at rest, single-use, expiring). The provision-then-hand-over agency flow — no secret is minted until the app is claimed.

POST /v1/platform/apps/claim

platform:apps:write
'

Claim an accountless app into the calling platform by its claim_token. Single-use and expiring; sets ownership and mints the first secret key, revealed once. A consumed, expired or unknown token is 4xx.

POST /v1/platform/apps/:id/transfer

platform:apps:write
'

Initiate a transfer of an owned app: returns a one-time transfer_token bound to it (hashed, single-use, expiring). Only the current owner can initiate; a non-owned app is 404.

POST /v1/platform/apps/accept_transfer

platform:apps:write
'

Accept a transfer by transfer_token: move the app's ownership to the calling platform and re-parent it to its account. The previous owner loses access immediately. Single-use and expiring; a consumed/expired/unknown token is 4xx.

Metrics

Metrics

Public — no authentication required.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /metrics		Prometheus exposition. Guarded by METRICS_TOKEN; 404s when unset, since it reports per-instance sign-in volume.

Frontend API overview

Frontend API overview

The Frontend API (FAPI) is what browsers and native apps talk to. It powers sign-in and sign-up, the signed-in "me" surface (profile, emails, MFA, sessions, organizations), and session-token minting. It is authenticated with a publishable key plus the Atlas session — never a secret key.

- Base: your instance's Frontend API host (*.fapi.atlasauth.net or your custom accounts domain).
- Auth: x-publishable-key: pk_… (or ?publishable_key=) plus the session — an HttpOnly cookie on web, a bearer token on native.
- Paths: /v1/client/…, /v1/appearance, /v1/client/environment.

You will rarely call FAPI by hand. Use an SDK — @atlas/react, @atlas/vue, @atlas/js, the mobile SDKs — which drive these multi-step flows and manage the session for you. The endpoints are documented so you understand what the SDK does and can build a custom UI when you need one.

Booting the client

```
GET /v1/client        !’ the current client; session: null for a signed-out
visitor (never 401)
GET /v1/client/me     !’ the signed-in user with emails, linked accounts, passkeys
GET /v1/client/environment  !’ enabled social providers (and native client ids)
GET /v1/appearance    !’ the embeddable widget's public theme + enabled strategies
```

The flow model

Sign-in and sign-up are multi-step attempts. You create an attempt, then advance it — each response carries a status that tells you the next step (verify email, submit a first factor, provide a second factor, complete). This models real-world flows: MFA, magic links, passwordless, OAuth, enterprise SSO, and passkeys all fit the same shape.

- Sign-in & sign-up — the attempt state machine and every first/second-factor method.
- Account management — the me surface: profile, emails, identities, MFA, passkeys, organizations, data export/deletion.
- Sessions & tokens — session listing, multi-session, getToken, and template tokens.

For the full list, see the Frontend endpoint index.

Personal access tokens

A signed-in user can mint a scoped personal access token (uat_) to drive their own account over the me-surface programmatically:

```
POST /v1/client/me/api_tokens   !’ mint (secret shown once)
GET  /v1/client/me/api_tokens   !’ list
DELETE /v1/client/me/api_tokens/:id !’ revoke
```

These are session-only and scoped; a revoked token fails closed immediately.

Backend API overview

Backend API overview

The Backend API (BAPI) is the secret-key surface your servers call to manage everything in an instance: users, sessions, organizations, roles, OAuth clients, enterprise SSO, SCIM, webhooks, JWT templates, and instance configuration. It is the peer of the dashboard — everything a human can do in the console is also a REST call, so provisioning can be automated.

- Base URL: https://api.atlas.dev
- Auth: Authorization: Bearer sk_…
- Every route is under /v1/… and declares a scope (e.g. users:read). A key missing a route's scope gets 403 SCOPE_MISSING.

```
import { createAtlasClient } from '@atlas/backend';
const atlas = createAtlasClient({ secretKey: process.env.ATLAS_SECRET_KEY! });
```

Resource groups

Group

Reference

Users, emails, identities, grants, imports/exports

Users

Sessions & session minting

Sessions

Organizations, memberships, invitations, domains, hierarchy

Organizations

Roles, permissions, group!’role grants

Roles & permissions

"Sign in with Atlas" OAuth clients & resource servers

OAuth clients

End-user API keys (ak_)

End-user API keys

Instance-level invitations

Invitations

Custom JWT claims

JWT templates

Authoritative token verification

Token verification

Fine-grained authorization (ReBAC / Zanzibar)

Fine-grained authorization

Audit log

Audit logs

Enterprise SSO connections, SAML, SCIM, self-service

SSO connections

Webhooks, events, signatures

Webhooks

For the complete machine-readable list of every BAPI route with its scope, see the Backend endpoint index.

Conventions recap

- Cursor pagination on list routes — limit + starting_after; response { data, has_more, next_cursor }.
- Idempotency on creates/mutations via the Idempotency-Key header (marked idempotent).
- Secrets shown once — client secrets, webhook signing secrets, API-key and SCIM-token secrets are revealed only at creation.
- Write-only secrets — provider client secrets, SAML IdP certs, captcha/kerberos secrets are stored encrypted and never read back; reads report a has_* marker instead.
- Cross-tenant is 404. An id from another instance is never 403.

Beyond the core

The BAPI also administers the whole instance: branding, email/SMS templates, localizations, OAuth providers (social sign-in config), messaging (BYOK email/SMS), Actions (auth-pipeline hooks), log streams, network ACLs, managed WAF, RADIUS clients, LTI platforms, attack protection, risk-based MFA, bot-signal export, data-subject (GDPR) requests, allow/block lists, waitlist, custom domains, billing plans, and instance security. Each of these appears with its endpoints and scope in the Backend endpoint index.

Webhooks overview

Webhooks overview

Webhooks push events to your server as they happen — a new user, a revoked session, an org change — so you never have to poll. Register an endpoint, subscribe it to events, and verify each delivery's signature.

Register an endpoint

The signing secret (whsec_…) is revealed exactly once at creation. Store it — you need it to verify deliveries.

Method & path	Scope	Notes
POST /v1/webhook_endpoints	webhooks:write	Register. Secret revealed once.
GET /v1/webhook_endpoints	webhooks:read	List. Never re-reveals secrets.
GET /v1/webhook_endpoints/:id/deliveries	webhooks:read	Delivery log for an endpoint
POST /v1/webhook_endpoints/:id/rotate_secret	webhooks:write	Rotate; new secret once; old-signed deliveries rejected immediately
POST /v1/webhook_endpoints/:id/deliveries/:deliveryId/redeliver	webhooks:write	Redeliver a past event (409 after 90-day retention)
POST /v1/webhook_endpoints/:id/test	webhooks:write	Send a synthetic webhook.test to this endpoint only
DELETE /v1/webhook_endpoints/:id	webhooks:write	Disable an endpoint

```
const ep = await atlas.webhooks.endpoints.create({
  url: 'https://acme.com/atlas/webhooks',
  enabled_events: ['user.created', 'user.deleted', 'organization.updated'],
});
console.log(ep.secret); // whsec_… — store it now, shown once
```

Subscribe to "*" for all events, or list specific event names. See the full event catalog.

The delivery

Atlas POSTs a JSON payload of full resource objects (not diffs), so a consumer processing events out of order can still reconstruct state:

```
{
  "id": "evt_…",
  "type": "user.created",
```

```
  "timestamp": 1757600000000,
  "instance_id": "ins_…",
  "data": { "object": "user", "id": "user_2a…", "…": "…" }
}
```

Signature and replay-protection headers accompany every delivery — see Verifying webhooks. Respond 2xx quickly; Atlas retries failed deliveries with backoff and records each attempt in the delivery log.

Retries & retention

- Failed deliveries are retried automatically; inspect attempts via the deliveries log.
- Events are retained for 90 days; a manual redeliver after that returns 409.
- Rotating the secret takes effect immediately — deliveries signed with the old secret are rejected.

Oauth Providers

Oauth Providers

Backend API — secret key (Authorization: Bearer sk_…).

7 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/oauth_providers	oauth_providers:read		List the social sign-in provider catalog with which are configured/enabled and each derived redirect URI. Never returns a client secret.
GET /v1/oauth_providers/:provider	oauth_providers:read		Fetch one social sign-in provider: its catalog entry and configured state, no secret.
PUT /v1/oauth_providers/:provider	oauth_providers:write	'	Set a social sign-in provider's client id/secret (and scopes/settings). The secret is write-only and stored encrypted; pasting your own clears the shared-dev-keys flag.
POST /v1/oauth_providers/:provider/enabled	oauth_providers:write		Enable or disable a social sign-in provider. Enabling one with no credentials is refused.
POST /v1/oauth_providers/:provider/scope	oauth_providers:write		Scope a provider to sign-in, sign-up, or both (allow_sign_in / allow_sign_up). Enforced at the resolver, so a sign-in-only provider never JIT-creates an account. Refuses "neither" — disable the provider instead.
POST /v1/oauth_providers/:provider/test	oauth_providers:read		Test a social sign-in provider's stored credentials against the provider's token endpoint. Proves the client id/secret only, not redirect-URI registration or scope approval.
DELETE /v1/oauth_providers/:provider	oauth_providers:write		Remove a social sign-in provider’s credentials. Existing linked accounts are preserved.

Users

Users

SCIM — a per-organization SCIM bearer token.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /scim/v2/Users		List provisioned users, filterable by userName eq and index-paginated.
POST /scim/v2/Users		Provision a user and seat them in the token organization directory.
GET /scim/v2/Users/:id		Fetch one provisioned user by id. Non-members are 404.
PUT /scim/v2/Users/:id		Replace a provisioned user, including its active (deprovision) state.
PATCH /scim/v2/Users/:id		PatchOp a user; active:false deprovisions and revokes every session.
DELETE /scim/v2/Users/:id		Deprovision a user: revoke sessions and remove the directory membership.

Requests & responses

Requests & responses

Content type

Request and response bodies are JSON. Send Content-Type: application/json on any request with a body. The wire is snake_case and the SDKs mirror it exactly — they type the API rather than inventing a second dialect.

```
curl -X PATCH https://api.atlas.dev/v1/users/user_2a \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"first_name":"Ada","public_metadata":{"plan":"pro"}}'
```

Object shapes

Most resources carry an object discriminator and a string id:

```
{
  "object": "user",
  "id": "user_2a…",
  "first_name": "Ada",
  "last_name": null,
  "public_metadata": { "plan": "pro" },
  "mfa_enabled": false,
  "banned": false,
  "created_at": 1757600000000,
  "updated_at": 1757600000000
}
```

- Ids are opaque. Prefixes (user_, org_, sess_, role_, perm_) are a readability aid, not a contract to parse.
- Timestamps are epoch milliseconds (integers), or null where a value has never been set (e.g. last_sign_in_at).
- Nullable fields are explicit — a field is present as null rather than omitted.

Metadata bags

Users and organizations carry free-form JSON metadata in up to three bags:

Bag	Visible to the frontend?	Writable from the frontend?
public_metadata	Yes (in the session/user object)	No — backend only
private_metadata	No — never returned by read routes	Backend only
unsafe_metadata	Yes	Yes (the user can set it)

PATCH merges the bag you send; PUT …/metadata replaces the named bags wholesale.

Mutations & deletions

- POST creates, PATCH partially updates, PUT replaces, DELETE removes.

- Deletes return a minimal acknowledgement: { "object": "user", "id": "user_2a…", "deleted": true }.
- Some revocations return a small custom shape (e.g. { "object": "session", "id": "…", "status": "revoked" }).

Reading raw formats

A few routes return non-JSON when asked (e.g. SAML SP metadata XML is carried inside a JSON envelope with a metadata_xml string). The SDKs expose these as typed helpers.

Introduction

Atlas API Reference

Atlas is a complete authentication and identity platform. This reference documents every public HTTP surface, the conventions that hold across all of them, and the SDKs that wrap them.

Atlas exposes three developer surfaces, each with its own audience and credential:

Surface	Audience	Base	Credential
Backend API (BAPI)	Your servers, agents, Terraform	https://api.atlas.dev	Secret key sk_… as Authorization: Bearer
Frontend API (FAPI)	Browsers and native apps	https://<instance>.fapi.atlasauth.net	Publishable key pk_… + session cookie/bearer
OIDC / OAuth2 & SCIM	Third party apps, IdPs, directories	Your instance host	Standards based (client creds, bearer, SCIM token)

The one rule that governs everything: authentication and authorization are enforced on the server. Client SDKs decide what a user sees; only a check on the request decides what they can do.

Where to start

- Authentication — publishable vs secret keys, how each surface authenticates.
- Base URLs & surfaces — which host to call for what.
- Errors, Pagination, Idempotency, Rate limits, Versioning — the cross cutting conventions.
- Backend API overview — the sk_ management surface, resource by resource.
- SDKs — official libraries for 8 frontend frameworks, 7 backend languages, and 3 mobile platforms.

Conventions at a glance

- JSON everywhere. Request and response bodies are JSON; the wire is snake_case.
- Stable, string ids. user_…, org_…, sess_…. Ids are opaque — do not parse them.
- Epoch millisecond timestamps. Fields like created_at are integers, not ISO strings.
- Cross tenant reads return 404, never 403. A 403 would confirm a resource exists in another instance — itself a disclosure.
- Secrets are revealed exactly once. Client secrets, webhook signing secrets, API key and SCIM token secrets are shown at creation and never read back.

This reference is grounded in the live Atlas API. Endpoint signatures, scopes, error codes, and the webhook signature scheme are taken directly from the running service.

Discovery & JWKS

Discovery & JWKS

Atlas publishes standard discovery documents so clients configure themselves rather than hard-coding hosts and keys.

OpenID Provider discovery

```
GET /.well-known/openid-configuration
```

Returns the OpenID Provider metadata for your instance: the host-based issuer, the authorization/token/userinfo/jwks endpoints, S256 PKCE support, and RS256 signing. Point any standards-compliant OIDC client at this URL.

JWKS (public signing keys)

```
GET /.well-known/jwks.json
GET /instances/:instanceId/.well-known/jwks.json
```

The public keys for verifying session JWTs and OIDC tokens. Cacheable with stale-if-error. Always fetch and cache rather than pinning a key — keys rotate.

```
import { AtlasBackend } from '@atlas/backend';

const atlas = new AtlasBackend({
  jwksUrl: 'https://accounts.yourapp.com/.well-known/jwks.json',
  issuer: 'https://accounts.yourapp.com',
});
```

SCIM discovery

```
GET /scim/v2/ServiceProviderConfig   !’ features + authentication scheme
GET /scim/v2/ResourceTypes           !’ the User and Group resource types
GET /scim/v2/Schemas                 !’ core User and Group schema definitions
```

SAML metadata

```
GET /v1/saml/idp/metadata                    !’ Atlas-as-IdP metadata (entityId,
signing cert, SSO bindings)
GET /v1/sso/saml/:connectionId/metadata      !’ SP metadata for a specific
connection (entityID + ACS URL)
```

Operational endpoints

```
GET /v1/health           !’ liveness (does not touch the database)
GET /v1/health/ready      !’ readiness (includes a database check)
GET /metrics             !’ Prometheus exposition (guarded by METRICS_TOKEN; 404
when unset)
```

Usage

Usage

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/usage	usage:read		Read usage analytics: daily sign-in/active-user series with peaks and trends over a window, plus monthly active users. active_users has no additive total by design.

Client Signin

Client Signin

Frontend API — publishable key (x-publishable-key) plus the user session.

29 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/client/sign_ups		Start an email + password sign-up. Returns an attempt whose status drives the next step.
POST /v1/client/sign_ups/:id/prepare_verification		Reissue a verification code. Rate limited to 1/30s and 5/hour per address.
POST /v1/client/sign_ups/:id/attempt_verification		Submit an email verification code. Three attempts, then the code is invalidated.
POST /v1/client/password_resets		Request a reset. Identical response for an address that does not exist.
POST /v1/client/password_resets/:id/attempt_verification		Submit the emailed code. Advances to MFA when the account has it.
POST /v1/client/password_resets/:id/attempt_second_factor		Second factor during a reset. Reset never bypasses MFA.
POST /v1/client/password_resets/:id/set_new_password		Set the new password, revoke every session, and bump sessions_version.
POST /v1/client/sign_ins		Start a sign-in. Returns the instance factor list, identical for unknown identifiers.
POST /v1/client/qr_sign_ins		Start a QR cross-device sign-in: returns the QR token/verification URL, a number-match code, and this tab’s poll secret. Opt-in; unauthenticated (publishable key).
GET /v1/client/qr_sign_ins/:qr_token		Phone lookup for a pending QR sign-in: the requesting browser’s device + coarse location and the match code to confirm. Requires the phone’s session.
POST /v1/client/qr_sign_ins/:qr_token/approve		

Phone approves a pending QR sign-in after confirming the number-match code; completes the attempt so the browser's poll mints a session. Requires the phone's session.

POST /v1/client/qr_sign_ins/:qr_token/deny

Phone denies a pending QR sign-in, abandoning the attempt so the browser's poll never completes. Requires the phone's session.

POST /v1/client/sign_ins/:id/prepare_first_factor

Send an email code or magic link. Returns the polling secret to this tab.

GET /v1/client/sign_ins/verify

Magic-link target. Completes the attempt; signs in no one here.

GET /v1/client/sign_ins/:id

Poll an attempt with its poll_secret; returns a ticket once complete.

POST /v1/client/sign_ins/:id/attempt_first_factor

Submit a first factor. Advances to MFA or completes, as the server decides.

POST /v1/client/sign_ins/:id/prepare_second_factor

Offer a passkey as the second factor. Credentials are named safely here.

POST /v1/client/sign_ins/:id/prepare_mfa_enrollment

Start TOTP enrollment for an attempt parked by a required MFA policy.

POST /v1/client/sign_ins/:id/attempt_mfa_enrollment

Confirm enrollment with two consecutive codes and complete the sign-in.

POST /v1/client/sign_ins/:id/attempt_second_factor

Submit a TOTP or recovery code. The only route out of needs_second_factor.

POST /v1/client/sign_ins/passkey/begin

Begin passwordless sign-in. Names no user and lists no credentials.

POST /v1/client/sign_ins/passkey/finish

Verify an assertion and issue a session. The credential names the user.

POST /v1/client/sign_ins/oauth

Begin an OAuth sign-in and receive the provider authorization URL.

POST /v1/client/sign_ins/id_token

Native / One-Tap sign-in: verify a provider id_token (Google GSI, Apple, Facebook Limited Login) and complete the sign-in, honouring the MFA gate.

POST /v1/client/sign_ins/ldap

LDAP/AD inbound sign-in: authenticate against a configured directory (connection_id + username + password) and JIT create/link the Atlas account, honouring the MFA gate.

POST /v1/client/sign_ins/kerberos

Kerberos / Integrated Windows Auth (IWA / SPNEGO) sign-in: a reverse proxy authenticates the principal and forwards it in a header; Atlas trusts it only behind a matching x-atlas-kerberos-secret, then resolves/JIT-links the account honouring the MFA gate. Opt-in, fail-closed.

POST /v1/client/sign_ins/external_jwt

Migration interop: verify a foreign provider JWT (a configured externalJwt issuer) against its JWKS and complete the sign-in (JIT + MFA gate), to run Atlas alongside an old provider.

POST /v1/client/sign_ins/sso

Begin an enterprise SSO sign-in. Resolves an OIDC connection by id or email domain and returns the IdP authorization URL.

POST /v1/client/sign_ins/saml

Begin an enterprise SAML sign-in. Resolves a SAML connection by id or email domain and returns the IdP AuthnRequest URL.

Users

Users

Create, read, update, and lifecycle-manage the people in your instance, plus their email addresses, linked identities, and OAuth consent grants.

The User object never includes private_metadata on a read. Timestamps are epoch ms.

```
{
  "object": "user",
  "id": "user_2a…",
  "username": null,
  "first_name": "Ada",
  "last_name": "Lovelace",
  "image_url": null,
  "public_metadata": { "plan": "pro" },
  "mfa_enabled": true,
  "banned": false,
  "locked": false,
  "last_sign_in_at": 1757600000000,
  "created_at": 1757000000000,
  "updated_at": 1757600000000
}
```

Endpoints

Method & path	Scope	Notes
GET /v1/users	users:read	List, cursor-paginated
GET /v1/users/:id	users:read	Never includes private_metadata
POST /v1/users	users:write	Create from email + optional password/name/metadata. Emits user.created. idempotent
PATCH /v1/users/:id	users:write	Update profile or metadata. idempotent
PUT /v1/users/:id/metadata	users:write	Replace the metadata bags wholesale. idempotent
DELETE /v1/users/:id	users:delete	Soft-delete; PII scrubbed after 30 days. idempotent
POST /v1/users/:id/ban	users:write	Ban and revoke every session. idempotent
POST /v1/users/:id/unban		

users:write
Lift a ban (sessions are not restored). idempotent
POST /v1/users/:id/lock
users:write
Lock out sign-in for an optional duration (default 1 year). idempotent
POST /v1/users/:id/unlock
users:write
Clear a lockout. idempotent
POST /v1/users/:id/reset_mfa
users:write
Remove every second factor + recovery codes. idempotent
DELETE /v1/users/:id/mfa/:factorId
users:write
Remove one factor (last removal turns MFA off). idempotent
GET /v1/users/:id/sessions
users:read
The user's active sessions
POST /v1/users/:id/sessions/revoke
users:write
Revoke all sessions, bump sessions_version. idempotent
POST /v1/users/:id/email_addresses
users:write
Add an address (starts unverified). idempotent
POST /v1/users/:id/email_addresses/:eid/verify
users:write
Mark verified on the backend authority. idempotent
POST /v1/users/:id/email_addresses/:eid/primary
users:write
Make a verified address primary. idempotent
GET /v1/users/:id/identities
users:read
Base Atlas identity + linked providers. No tokens.
POST /v1/users/:id/identities
users:write
Merge a secondary user into this one. idempotent
DELETE /v1/users/:id/identities/:identityId
users:write
Unlink a provider into a new standalone user. idempotent
POST /v1/users/:id/external_accounts/connect
users:write
Backend-initiated OAuth link; returns an authorization_url
GET /v1/users/:id/oauth_access_tokens/:provider
oauth_tokens:read
Stored provider token, refreshed if stale

GET /v1/users/:id/grants	
grants:read	
OAuth clients the user authorized	
DELETE /v1/users/:id/grants	
grants:write	
Revoke all consent grants + tokens. idempotent	
DELETE /v1/grants/:id	
grants:write	
Revoke one consent grant. idempotent	

Bulk import / export

Method & path	Scope	Notes
POST /v1/user_imports	jobs:write	Bulk-create users (dedupe by verified email). Argon2id hashes imported verbatim; plaintext is hashed; other formats rejected per-row. idempotent
POST /v1/user_exports	jobs:write	Produce a serialised export job. Never contains password hashes / tokens. idempotent
GET /v1/jobs · GET /v1/jobs/:id · GET /v1/jobs/:id/errors	jobs:read	Track jobs, counters, and per-row errors

Examples

```
# Create a user, idempotently
curl https://api.atlas.dev/v1/users \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: signup-ada-001" \
  -d
'{"email_address":"ada@example.com","first_name":"Ada","email_verified":true}'

# Ban a user (revokes every session)
curl -X POST https://api.atlas.dev/v1/users/user_2a/ban \
  -H "Authorization: Bearer sk_live_xxx"
```

```
// @atlas/backend
const user = await atlas.users.create({ email_address: 'ada@example.com',
first_name: 'Ada' });
await atlas.users.update(user.id, { public_metadata: { plan: 'pro' } });
await atlas.users.lock(user.id, { duration_in_seconds: 3600 });

// Merge a duplicate account into this one
const { collisions } = await atlas.users.linkIdentity(user.id,
{ secondary_user_id: 'user_dupe' });

// Read a stored Google token (auto-refreshed)
const { token } = await atlas.users.getOAuthAccessToken(user.id, 'google');
```

```
# atlas-backend (Python)
user = atlas.users.create({"email_address": "ada@example.com", "first_name":
"Ada"})
for u in paginate(atlas.users.list):
    print(u["id"])
```

SAML

SAML

Atlas speaks SAML in both directions.

Atlas as Service Provider (inbound SSO)

Your customers' IdPs sign users into your app. Configure the connection via SSO connections, hand the IdP the SP metadata, and users flow through:

```
POST /v1/client/sign_ins/saml            !’ begin: resolve a SAML connection by id or email domain, return the IdP AuthnRequest URL
POST /v1/sso/saml/acs                     !’ Assertion Consumer Service: verifies the signed assertion (audience, timestamps,
                                            XSW-hardened), JIT-provisions org membership, redirects back with a one-time ticket
GET  /v1/sso/saml/:connectionId/metadata !’ SP metadata XML (entityID + ACS URL) the customer registers with their IdP
```

Assertions are verified for a signed response, audience restriction, timestamp validity, and are hardened against XML Signature Wrapping (XSW). IdP-initiated flows are allowed only when the connection opts in (saml_allow_idp_initiated).

Atlas as Identity Provider (outbound)

Atlas can itself be the IdP for downstream SPs — your users sign into other SaaS with their Atlas identity.

```
GET  /v1/saml/idp/metadata   !’ Atlas-as-IdP metadata (entityId, signing certificate, SSO bindings) for a downstream SP to import
GET|POST /v1/saml/idp/sso    !’ the IdP SSO endpoint (session-gated): IdP-initiated (?sp=) or SP-initiated (SAMLRequest);
                               returns a signed SAML Response auto-POST form
```

OIDC enterprise SSO

For IdPs that speak OIDC instead of SAML, the equivalent inbound flow is:

```
POST /v1/client/sign_ins/sso   !’ begin (resolve OIDC connection by id or email domain), return the IdP authorization URL
GET  /v1/sso/callback          !’ IdP redirect target: verify the id_token, JIT-provision membership, redirect back with a ticket
```

DiscourseConnect

```
GET /discourse/sso   !’ DiscourseConnect provider endpoint: verifies the forum HMAC (constant-time),
                       authenticates via the hosted session, signs the identity payload back to the pinned return_sso_url
```

Organizations

Organizations

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

21 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations		List or search organizations, with member counts and the O4 orphaned flag.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations		Create an organization in this instance. Starts adminless until its first invite.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId		One organization with its members and its full invitation history.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId		Delete an organization. The O4 remedy; requires owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/clone		Clone an org's config (name, slug, policy, email domains, groups + role grants — not members) into the application's other environment (prod!”test).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/members		Add an existing user of this instance to an organization (O5 enforced).
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/members/:userId		Change a member's role. Demoting the last admin is 409 LAST_ADMIN (O1).
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/members/:userId		Remove a member. Removing the last admin is 409 LAST_ADMIN (O1).
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/groups		List the organization directory groups, so a role grant can pick one.

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/policy

Read the org security policy (requireMfa, ssoRequired, sessionIdleOverrideMs).

PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/policy

Update the org security policy (requireMfa, ssoRequired, sessionIdleOverrideMs).

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/groups/:groupId/roles

List the org roles granted to a directory group.

PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/groups/:groupId/roles/:roleId

Grant an org role to a directory group; members inherit its permissions.

DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/groups/:groupId/roles/:roleId

Revoke a role from a directory group.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/invitations

Issue an invitation with no member behind it; the audit row names the operator.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/invitations/:invitationId/revoke

Revoke a pending invitation. Already-accepted ones are 409.

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/domains

List an organization email domains and their verification state.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/domains

Claim an email domain for auto-join. Returns the DNS TXT record to publish.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/domains/:domainId/verify

Check the DNS TXT record now and mark the domain verified.

PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/domains/:domainId

Update a claimed domain auto-join and default role. The domain and its verification state are immutable.

DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/organizations/:organizationId/domains/:domainId

Remove a claimed email domain.

Pagination

Pagination

List endpoints are cursor-paginated. There is no page number and no total count — cursors are stable under insertion and deletion, which page numbers are not.

Parameters

Param	Meaning
limit	Page size. The server clamps to its own maximum.
starting_after	Opaque cursor: the id after which to continue. Pass back the previous page's next_cursor.

Response envelope

```
{
  "data": [ { "object": "user", "id": "user_1", "…": "…" } ],
  "has_more": true,
  "next_cursor": "user_1"
}
```

When has_more is false, next_cursor is null and you have reached the end.

```
# First page
curl "https://api.atlas.dev/v1/users?limit=100" \
  -H "Authorization: Bearer sk_live_xxx"

# Next page
curl "https://api.atlas.dev/v1/users?limit=100&starting_after=user_1" \
  -H "Authorization: Bearer sk_live_xxx"
```

> Some list routes (memberships, roles, permissions, invitations by org) use a lighter { "object": "list", "data": [...], "has_more": … } envelope without a cursor, because the sets are small and bounded.

Walking every page with an SDK

The Node SDK ships paginate (an async generator) and collect (the whole set), which work with any cursor-paginated list:

```
import { paginate, collect } from '@atlas/backend';

// Stream users one at a time
for await (const user of paginate(atlas.users.list)) {
  console.log(user.id);
}

// Or gather them all
const orgs = await collect(atlas.organizations.list);
```

The Ruby, Python, Go, and other SDKs expose the same walk (Atlas.paginate / paginate(...)). Prefer streaming for large sets so you make extra round trips only when you actually need the next page.

Frontend SDKs

Frontend SDKs

All frontend SDKs authenticate with a publishable key and manage the session for you. Mount the provider once, then use the framework-idiomatic hooks/components.

@atlas/js (framework-agnostic core)

The core the other web SDKs build on — use it directly for vanilla JS or an unsupported framework. Ships a framework-agnostic has(claims, condition).

```
npm install @atlas/js
```

```
import { Atlas } from '@atlas/js';

const atlas = new Atlas({ publishableKey: 'pk_live_…' });
await atlas.load();

if (atlas.session) {
  const token = await atlas.session.getToken();
}
```

@atlas/react

```
npm install @atlas/react
```

```
import { AtlasProvider, SignedIn, SignedOut, SignInButton, UserButton,
useAuth } from '@atlas/react';

function Root() {
  return (
    <AtlasProvider publishableKey="pk_live_…">
      <SignedOut><SignInButton /></SignedOut>
      <SignedIn><UserButton /></SignedIn>
    </AtlasProvider>
  );
}

// Call your API with a session token
function useApi() {
  const { getToken } = useAuth();
  return async () => fetch('/api/x', { headers: { Authorization: `Bearer
${await getToken()}` } });
}
```

Gate UI with <Protect> / useAuth().has() — a rendering aid, never a security boundary (keep the server check).

@atlas/nextjs

App Router provider + middleware + a server-side auth() helper exposing has() / protect():

```
npm install @atlas/nextjs
```

```
// app/layout.tsx
import { AtlasProvider } from '@atlas/nextjs';
export default function Layout({ children }) {
  return <AtlasProvider>{children}</AtlasProvider>;
}
```

```
// app/api/things/route.ts
import { auth } from '@/atlas'; // your createAuthHelper() instance
export async function POST(request: Request) {
  const a = await auth(request);
  a.protect({ permission: 'org:billing:manage' }); // 403 if not held
  // …privileged work
}
```

@atlas/vue

Official Vue 3 SDK — plugin + Composition API composables + host-DOM components.

```
npm install @atlas/vue
```

```
import { createApp } from 'vue';
import { createAtlas } from '@atlas/vue';
import App from './App.vue';

createApp(App)
  .use(createAtlas({ publishableKey: 'pk_live_…' }))
  .mount('#app');
```

@atlas/nuxt

Nuxt 3 module — the client half is @atlas/vue; the Nitro server half verifies the __session JWT locally via @atlas/backend.

```
pnpm add @atlas/nuxt
```

```
// nuxt.config.ts
export default defineNuxtConfig({
  modules: ['@atlas/nuxt'],
  atlas: {
    publishableKey: 'pk_test_...',
    frontendApi: 'https://your-instance.fapi.atlasauth.net',
  },
  runtimeConfig: {
    atlas: { jwksUrl: '', issuer: '' },        // server-only
    public: { atlas: { publishableKey: '', frontendApi: '' } },
  },
});
```

@atlas/angular

Standalone APIs, RxJS + Signals, directives, route guards. Angular 17+.

```
pnpm add @atlas/angular
```

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideAtlas } from '@atlas/angular';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, {
  providers: [provideAtlas({ publishableKey: 'pk_test_…' })],
});
```

@atlas/svelte

Svelte / SvelteKit — client via context, stores, .svelte components; optional SvelteKit server helper.

```
pnpm add @atlas/svelte
```

```
<!-- src/routes/+layout.svelte -->
<script lang="ts">
  import { AtlasProvider } from '@atlas/svelte';
</script>

<AtlasProvider publishableKey="pk_live_…" frontendApi="https://
<instance>.atlas.dev">
  <slot />
</AtlasProvider>
```

@atlas/react-native

Native peer of @atlas/react; the session token lives in a TokenStorage adapter and is sent as a bearer header (no cookie jar). See Mobile SDKs for the fully native options.

```
npm install @atlas/react-native
```

```
import { AtlasProvider } from '@atlas/react-native';
import { secureStorage } from './storage';

export default function App() {
```

```
  return (
    <AtlasProvider
      publishableKey="pk_live_…"
      frontendApi="https://fapi.your-instance.com"
      storage={secureStorage}
    >
      <RootNavigator />
    </AtlasProvider>
  );
}
```

Groups

Groups

SCIM — a per-organization SCIM bearer token.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /scim/v2/Groups		List directory groups, filterable by displayName eq and index-paginated.
POST /scim/v2/Groups		Create a directory group with an optional initial member set.
GET /scim/v2/Groups/:id		Fetch one directory group with its member references.
PUT /scim/v2/Groups/:id		Replace a group displayName and its full membership set.
PATCH /scim/v2/Groups/:id		PatchOp a group: add or remove members, or rename it.
DELETE /scim/v2/Groups/:id		Delete a directory group. Membership rows cascade with it.

Sign-in & sign-up

Sign-in & sign-up

Sign-in and sign-up are modelled as attempts: you create one, then advance it step by step. Each response carries a status describing what is needed next. The SDKs drive this for you; this page documents the endpoints for custom UIs.

Sign-up

```
POST /v1/client/sign_ups                                !’ start an email+password
sign-up
POST /v1/client/sign_ups/:id/prepare_verification     !’ (re)send a verification
code (1/30s, 5/hr per address)
POST /v1/client/sign_ups/:id/attempt_verification     !’ submit the code (3
attempts, then invalidated)
```

Sign-in — first factor

```
POST /v1/client/sign_ins                                !’ start; returns the
factor list (identical for unknown identifiers)
POST /v1/client/sign_ins/:id/prepare_first_factor     !’ send an email code or
magic link; returns this tab's poll_secret
POST /v1/client/sign_ins/:id/attempt_first_factor     !’ submit a first factor;
advances to MFA or completes
GET  /v1/client/sign_ins/:id                            !’ poll an attempt (with
poll_secret); returns a ticket once complete
GET  /v1/client/sign_ins/verify                         !’ magic-link target;
completes the attempt
```

The response for an unknown identifier is deliberately identical to a known one — the API never reveals which accounts exist.

Second factor (MFA)

```
POST /v1/client/sign_ins/:id/prepare_second_factor    !’ offer a passkey as the
second factor
POST /v1/client/sign_ins/:id/attempt_second_factor    !’ submit a TOTP or
recovery code
POST /v1/client/sign_ins/:id/prepare_mfa_enrollment   !’ start TOTP enrollment
for an attempt parked by a required-MFA policy
POST /v1/client/sign_ins/:id/attempt_mfa_enrollment   !’ confirm with two
consecutive codes and complete
```

Passwordless & passkeys

```
POST /v1/client/sign_ins/passkey/begin    !’ begin passwordless sign-in (names no
user)
POST /v1/client/sign_ins/passkey/finish   !’ verify an assertion and issue a
session
```

QR cross-device sign-in ("scan to sign in on this TV/desktop"):

```
POST /v1/client/qr_sign_ins                 !’ returns QR token, number-match
code, this tab's poll secret
GET  /v1/client/qr_sign_ins/:qr_token       !’ phone lookup (device, coarse
location, match code)
POST /v1/client/qr_sign_ins/:qr_token/approve | /deny   !’ phone confirms the
number match
```

Password reset

```
POST /v1/client/password_resets                           !’ request a reset
(identical response whether or not the address exists)
POST /v1/client/password_resets/:id/attempt_verification !’ submit the emailed
code
POST /v1/client/password_resets/:id/attempt_second_factor!’ second factor (reset
```

```
never bypasses MFA)
POST /v1/client/password_resets/:id/set_new_password     !’ set new password,
revoke every session
```

Social, enterprise & federated

```
POST /v1/client/sign_ins/oauth        !’ begin an OAuth sign-in; returns the
provider authorize URL
POST /v1/client/sign_ins/id_token     !’ native / One-Tap (Google GSI, Apple,
Facebook Limited Login)
POST /v1/client/sign_ins/sso          !’ begin enterprise OIDC SSO (resolve by
connection id or email domain)
POST /v1/client/sign_ins/saml         !’ begin enterprise SAML SSO
POST /v1/client/sign_ins/ldap         !’ LDAP/AD inbound sign-in (JIT create/link)
POST /v1/client/sign_ins/kerberos     !’ Kerberos / IWA (SPNEGO) behind a trusted
proxy
POST /v1/client/sign_ins/external_jwt !’ verify a foreign provider JWT (migration
interop)
```

Web3 wallet sign-in (SIWE / SIWS / SIWF) and Telegram Login are also supported via /v1/oauth/… nonce+verify pairs — see the Frontend endpoint index. Every path honours the instance MFA gate.

Completing an attempt

A completed attempt yields a one-time ticket. Exchange it for session cookies:

```
POST /v1/client/tickets/exchange   !’ exchange a one-time ticket for a session
```

Example (SDK)

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

const { signIn } = useSignIn();
// The SDK walks create !’ attempt !’ ticket exchange and manages the session:
await signIn.create({ identifier: 'ada@example.com', password: '…' });
```

Webhook events

Webhook events

Subscribe an endpoint to any of these event types (or "*" for all). Each delivery carries the full resource object in data. The catalog is versioned — a new event is an additive change.

Users

- user.created
- user.updated
- user.deleted
- user.banned
- user.unbanned

Sessions

- session.created
- session.revoked
- session.ended
- session.pending — entered the pending state (awaiting a second factor / MFA-enrollment task)
- session.removed

Messaging

- email.created
- sms.created

Organizations

- organization.created
- organization.updated
- organization.deleted
- organizationMembership.created
- organizationMembership.updated
- organizationMembership.deleted
- organizationInvitation.created
- organizationInvitation.accepted
- organizationInvitation.revoked
- organizationDomain.created
- organizationDomain.updated
- organizationDomain.deleted

Roles & permissions

- role.created
- role.updated
- role.deleted
- permission.created
- permission.updated
- permission.deleted

Waitlist

- waitlistEntry.created
- waitlistEntry.updated

Billing (Stripe-driven)

- subscription.created
- subscription.updated
- subscription.deleted

SCIM & SSO

- scimToken.created
- scimToken.revoked
- ssoConnection.created
- ssoConnection.updated
- ssoConnection.deleted

Test

- webhook.test — emitted only by POST /v1/webhook_endpoints/:id/test, delivered to that endpoint only.

GET /v1/events (scope events:read) reports which event types this instance has actually emitted, so you can discover what is live before subscribing.

Health

Health

Public — no authentication required.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/health		'Liveness probe. Does not touch the database.', surface: 'public' }
GET /v1/health/ready		Readiness probe including a database check.

Authentication

Authentication

Every Atlas surface authenticates differently, because each has a different trust boundary. Pick the credential that matches the surface you are calling.

API keys

Your instance issues two kinds of key. They are not interchangeable.

	Key	Prefix	Where it lives	Sent as
Publishable key	pk_live_… / pk_test_…	The browser / mobile app — safe to ship in client code	x-publishable-key header (or publishable_key query param) to FAPI
Secret key	sk_live_… / sk_test_…	Your server only — never expose it	Authorization: Bearer sk_… to BAPI

The _live_ / _test_ segment selects the environment. A publishable key identifies your instance to the Frontend API; the SDKs derive the FAPI host from it (or you pass frontendApi explicitly). A secret key carries full administrative power over the instance — treat it like a password, keep it out of source control, and rotate it if it leaks.

Backend API (BAPI) — secret key

Server-to-server calls use the secret key as a bearer token:

```
curl https://api.atlas.dev/v1/users \
  -H "Authorization: Bearer sk_live_xxx"
```

The key is never placed in a URL and never logged. A missing or malformed key is 401 UNAUTHENTICATED; a revoked or wrong key is 401 INVALID_KEY. A valid key that lacks the scope a route requires is 403 SCOPE_MISSING with meta.required_scope.

Scopes

Secret keys are scoped. Each BAPI route declares the scope it needs — for example users:read, users:write, organizations:write, webhooks:write, sessions:read. Mint a key with only the scopes an integration needs. Every endpoint in this reference lists its required scope.

Frontend API (FAPI) — publishable key + session

Browser and native clients call FAPI with the publishable key and (once signed in) the Atlas session:

```
curl "https://<instance>.fapi.atlasauth.net/v1/client" \
  -H "x-publishable-key: pk_live_xxx"
```

- On the web, the session travels as an HttpOnly cookie set by Atlas; the SDK also mints short-lived session JWTs for your own API.
- On native, there is no cookie jar — the SDK stores the session token and sends it as a bearer header.
- GET /v1/client boots the client and returns session: null for a signed-out visitor (never 401).

Personal access tokens (uat_)

A signed-in user can mint a scoped personal access token (POST /v1/client/me/api_tokens) so an agent or script can drive their own account over the me-surface with a bearer header. These are session-only, scoped, and the secret is shown exactly once.

Standards-based surfaces

- OIDC / OAuth2 clients authenticate at /oauth2/token with client_secret_basic, client_secret_post, or PKCE (public clients). Access tokens are bearer tokens accepted at /oauth2/userinfo and your own resource servers.
- SCIM directory sync authenticates with a per-organization SCIM bearer token (minted via POST /v1/scim_tokens, shown once).
- Webhooks are the reverse direction: Atlas signs each delivery so you can authenticate it — see Verifying webhooks.

End-user API keys (ak_)

Distinct from your instance keys: a tenant can mint API keys for its own users/organizations (POST /v1/api_keys, returns an ak_ secret once) and later verify a presented one with POST /v1/api_keys/verify. See End-user API keys.

Discovery

Discovery

SCIM — a per-organization SCIM bearer token.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /scim/v2/ServiceProviderConfig		SCIM discovery: the features and authentication scheme this server supports.
GET /scim/v2/ResourceTypes		SCIM discovery: the User and Group resource types this server exposes.
GET /scim/v2/Schemas		SCIM discovery: the core User and Group schema definitions.

Base URLs & surfaces

Base URLs & surfaces

Atlas is a single deployable that hosts several surfaces on separate route trees with separate authentication, so the trust boundary stays legible.

Backend API (BAPI)

- Base: https://api.atlas.dev (the SDK default; self-hosted and per-instance deployments override it).
- Auth: secret key, Authorization: Bearer sk_….
- Paths: everything under /v1/….
- Audience: your servers, background jobs, agents, Terraform. Full administrative surface.

```
curl https://api.atlas.dev/v1/organizations \
  -H "Authorization: Bearer sk_live_xxx"
```

Frontend API (FAPI)

- Base: your instance's Frontend API host — a *.fapi.atlasauth.net subdomain, or your own custom domain (e.g. accounts.yourapp.com). The publishable key encodes which instance you are, and the SDKs resolve the host from it.
- Auth: publishable key (x-publishable-key) plus the Atlas session (cookie on web, bearer on native).
- Paths: /v1/client/… (the signed-in me surface and sign-in/sign-up flows), /v1/appearance, /v1/client/environment.
- Audience: browsers and native apps. Never sees a secret.

OIDC / OAuth2, SAML & SCIM

Served on your instance host (the same host as your hosted pages / accounts domain):

- OIDC / OAuth2: /oauth2/authorize, /oauth2/token, /oauth2/userinfo, /oauth2/revoke, /oauth2/introspect, plus discovery at /.well-known/openid-configuration and keys at /.well-known/jwks.json.
- SAML: Atlas as SP (/v1/sso/saml/acs, /v1/sso/saml/:connectionId/metadata) and Atlas as IdP (/v1/saml/idp/sso, /v1/saml/idp/metadata).
- SCIM 2.0: /scim/v2/Users, /scim/v2/Groups, and discovery under /scim/v2/….

Hosted pages

Zero-integration UI, rendered with your instance branding:

- /hosted/sign-in, /hosted/sign-up, /hosted/user and the account-less /hosted/sso-setup self-service flow.

Environments

live and _test_ keys select production and test data on the same host. Keep test keys in development and CI; never ship an sk_ key of any environment to a browser.

Dynamic Client Registration

Dynamic Client Registration

For ecosystems where clients register themselves (RFC 7591) rather than being provisioned by an admin. Gated by a per-instance flag — when off, registration is 403.

```
POST   /oauth2/register            !’ register a client (RFC 7591). Validates
https/localhost redirect_uris.
                                      Returns the new client_id and, for
confidential clients, a secret.
GET    /oauth2/register/:clientId  !’ read config (RFC 7592)
PUT    /oauth2/register/:clientId  !’ update config (RFC 7592)
DELETE /oauth2/register/:clientId  !’ delete (RFC 7592)
```

The management endpoints (RFC 7592) authenticate with the registration_access_token issued at registration — not your instance keys.

```
curl https://accounts.yourapp.com/oauth2/register \
  -H "Content-Type: application/json" \
  -d '{
    "client_name": "Partner App",
    "redirect_uris": ["https://partner.example.com/callback"],
    "grant_types": ["authorization_code", "refresh_token"],
    "token_endpoint_auth_method": "client_secret_basic"
  }'
```

The response includes client_id, (for confidential clients) client_secret, and a registration_access_token + registration_client_uri for later management.

Admin-provisioned clients

If you provision clients yourself instead, use the BAPI OAuth clients endpoints (POST /v1/oauth_clients), which give you scoping, secret rotation, and resource-server grants.

Errors

Errors

The envelope

Every failed request answers with the same shape — an ordered array, because one request can produce more than one problem:

```
{
  "errors": [
    {
      "code": "SCOPE_MISSING",
      "message": "This key is missing the required scope: users:write.",
      "param": null,
      "meta": { "required_scope": "users:write" }
    }
  ]
}
```

- code is the stable, machine-readable contract. Branch on it. Codes are never renamed silently.
- message is human-readable and may change.
- param names the offending field on a validation error.
- meta carries structured extras (e.g. retry_after on a rate limit, required_scope on a scope error, an existing account's providers on ACCOUNT_EXISTS).

Internal (5xx) faults return an opaque INTERNAL message on purpose — a detailed server error is a classic accidental disclosure channel.

HTTP status codes

Status	Meaning
200 / 201	Success
204	Success, no body
400	VALIDATION_FAILED — malformed request
401	UNAUTHENTICATED / INVALID_KEY — missing or bad credential
403	FORBIDDEN / SCOPE_MISSING — authenticated but not allowed
404	NOT_FOUND — including any cross-tenant access (never 403, which would confirm the resource exists elsewhere)
409	Conflict — LAST_ADMIN, ROLE_IN_USE, IDEMPOTENCY_CONFLICT, CONFLICT, …
422	Unprocessable — e.g. an action refused by policy
429	RATE_LIMITED — see Rate limits
503	

NOT_CONFIGURED — an optional capability has no credentials on this deployment

Handling errors in an SDK

The Node SDK throws AtlasApiError on any non-2xx, exposing status, the full errors array, and helpers:

```
import { AtlasApiError } from '@atlas/backend';

try {
  await atlas.organizations.memberships.update(orgId, userId, { role:
'org:member' });
} catch (e) {
  if (e instanceof AtlasApiError && e.hasCode('LAST_ADMIN')) {
    // Can't demote the last admin — pick another admin first.
  } else {
    throw e;
  }
}
```

Error code catalog

Request

VALIDATION_FAILED · NOT_FOUND · RATE_LIMITED · IDEMPOTENCY_CONFLICT · NOT_CONFIGURED · CONFLICT · INTERNAL

Authentication & authorization

UNAUTHENTICATED · INVALID_KEY · FORBIDDEN · SCOPE_MISSING

Identity & credentials

IDENTIFIER_EXISTS · IDENTIFIER_NOT_FOUND · VERIFICATION_EXPIRED · VERIFICATION_FAILED · TOO_MANY_ATTEMPTS · PASSWORD_BREACHED · PASSWORD_POLICY · PASSWORD_EXPIRED · LOCKED · BANNED · MFA_REQUIRED · MFA_ALREADY_ENROLLED · MFA_NOT_ENROLLED · STEP_UP_REQUIRED · LAST_EMAIL · SESSION_EXPIRED · TOKEN_REUSE_DETECTED

OAuth & linking

OAUTH_STATE_INVALID · PROVIDER_TOKEN_UNAVAILABLE · OAUTH_EXCHANGE_FAILED · LAST_AUTH_METHOD · IDENTITY_ALREADY_LINKED · ACCOUNT_EXISTS · WALLET_GATE_UNMET

Organizations

LAST_ADMIN · ALREADY_MEMBER · ROLE_IN_USE · MEMBERSHIP_LIMIT_REACHED · SSO_REQUIRED

Sign-up gating & bot defence

STRATEGY_DISABLED · SIGN_UP_RESTRICTED · WAITLISTED · IDENTIFIER_NOT_ALLOWED · DISPOSABLE_EMAIL_BLOCKED · SMS_REGION_BLOCKED · AUTHENTICATOR_NOT_ALLOWED · CAPTCHA_REQUIRED · CAPTCHA_FAILED · CHALLENGE_UNAVAILABLE

Kill switches, network & policy

READ_ONLY_MODE · SIGN_UPS_DISABLED · SIGN_INS_DISABLED · PROVIDER_DISABLED · PROVIDER_SIGN_UP_DISABLED · PROVIDER_SIGN_IN_DISABLED · IP_NOT_ALLOWED

Extensibility & risk

ACTION_BLOCKED (a fail-closed Action denied the flow) · HIGH_RISK_BLOCKED (adaptive MFA blocked a high-risk sign-in with no enrolled factor)

Verifying webhooks

Verifying webhooks

Every delivery is signed so you can prove it came from Atlas and has not been tampered with or replayed. Verify before you trust the body.

Headers

Header	Meaning
atlas-id	The event id (evt_…) — also your dedupe key
atlas-timestamp	Delivery time, epoch milliseconds
atlas-signature	v1,<base64> — the HMAC over the signed payload

The signed payload

The signature is HMAC-SHA256 over the string:

`<atlas-id>.<atlas-timestamp>.<raw request body>`

keyed with your endpoint's signing secret (whsec_…), base64-encoded, and prefixed v1,.

The timestamp is inside the signed material, not merely a header — otherwise an attacker could replay a captured body with a fresh timestamp and the signature would still verify. Reject any delivery whose timestamp is more than 5 minutes from now (in either direction).

Verify it yourself

Use the raw request body — parse JSON only after the signature checks out. Compare with a constant-time equality.

```
import { createHmac, timingSafeEqual } from 'node:crypto';

const TOLERANCE_MS = 5 * 60_000;

function verifyAtlasWebhook(req, secret: string): boolean {
  const id = req.headers['atlas-id'];
  const ts = Number(req.headers['atlas-timestamp']);
  const sig = String(req.headers['atlas-signature']); // "v1,<base64>"
  const body = req.rawBody; // the exact bytes Atlas sent

  const [version, provided] = sig.split(',');
  if (version !== 'v1' || !provided) return false;
  if (Math.abs(Date.now() - ts) > TOLERANCE_MS) return false; // replay window

  const expected = createHmac('sha256', secret)
    .update(`${id}.${ts}.${body}`)
    .digest('base64');

  const a = Buffer.from(provided);
  const b = Buffer.from(expected);
  return a.length === b.length && timingSafeEqual(a, b);
}
```

```
import hmac, hashlib, base64, time

def verify_atlas_webhook(headers, raw_body: bytes, secret: str) -> bool:
    version, _, provided = headers["atlas-signature"].partition(",")
    if version != "v1" or not provided:
```

```
        return False
    ts = int(headers["atlas-timestamp"])
    if abs(time.time() * 1000 - ts) > 5 * 60_000:
        return False
    signed = f"{headers['atlas-id']}.{ts}.".encode() + raw_body
    expected = base64.b64encode(hmac.new(secret.encode(), signed,
hashlib.sha256).digest()).decode()
    return hmac.compare_digest(provided, expected)
```

Or let the SDK do it

The backend SDKs ship a verifier so the contract has exactly one definition:

```
import { verifyWebhook } from '@atlas/backend';

const result = verifyWebhook({
  eventId: req.headers['atlas-id'],
  timestampMs: Number(req.headers['atlas-timestamp']),
  body: req.rawBody,
  signature: req.headers['atlas-signature'],
  secret: process.env.ATLAS_WEBHOOK_SECRET!,
});
if (!result.valid) return res.status(400).end(); // result.reason: 'malformed'
| 'stale' | 'mismatch'
```

Idempotency on your side

Deliveries can repeat (retries, manual redelivery). Dedupe on atlas-id so processing the same event twice is a no-op.

Hosted

Hosted

Public — no authentication required.

11 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /hosted/sign-in		Hosted sign-in page with instance branding. Zero-integration path.
GET /hosted/sign-up		Hosted sign-up page with instance branding.
GET /hosted/user		Hosted account management page with instance branding.
GET /hosted/app.js		Client script for the hosted pages; drives the sign-in flow.
GET /hosted/sso-setup		Account-less hosted SSO self-service setup page, authenticated only by a scoped one-time ticket token.
GET /hosted/sso-setup.js		Client script for the hosted SSO self-service setup page.
POST /hosted/sso-setup/save		Create or edit ONLY the ticket-bound connection for ONLY the ticket-bound organization. Client-supplied org/connection ids are ignored; secrets are write-only.
POST /hosted/sso-setup/complete		Mark a self-service onboarding ticket completed so it can no longer be used.
POST /hosted/sso-setup/scim-token		Mint a SCIM directory-sync bearer token scoped to the ticket-bound organization (only when the profile allows SCIM). Shown once; never read back.
POST /hosted/sso-setup/domains		Claim an email domain for the ticket-bound organization; returns the DNS TXT record to publish.

POST /hosted/sso-setup/domains/verify

Run the DNS-TXT verification for one of the ticket-bound organization’s claimed domains.

Client Telemetry

Client Telemetry

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/client/telemetry		Anti-bot signal beacon: the SDK posts anonymised device + behavioural signals for the sign-in/sign-up form; the server enriches (salt-hashed IP, coarse geo, heuristic score) and appends to the bot_signals lake. Fire-and-forget (202), never blocks a sign-in.

Sessions

Sessions

A session represents a signed-in device. From the backend you can mint one headlessly (for agents, testing, or impersonation), list a user's sessions, and revoke them.

```
{
  "object": "session",
  "id": "sess_…",
  "user_id": "user_2a…",
  "status": "active",
  "last_active_organization_id": "org_9f…",
  "impersonated_by": null,
  "last_active_at": 1757600000000,
  "expire_at": 1757603600000,
  "abandon_at": 1757800000000,
  "created_at": 1757600000000
}
```

Endpoints

Method & path	Scope	Notes
POST /v1/sessions	sessions:write	Mint a session for a user without the sign-in flow. Returns the bearer jwt + refresh_token in-body. Refused for a banned user.
GET /v1/sessions	sessions:read	List a user's sessions. Requires ?user_id=.
GET /v1/sessions/:id	sessions:read	Fetch one session (another instance's id is 404).
POST /v1/sessions/:id/revoke	sessions:write	Revoke one session, signing that device out.
POST /v1/tokens/verify	sessions:read	Authoritatively verify a session token, including revocation — see Token verification.

Related, on the Users API: GET /v1/users/:id/sessions and POST /v1/users/:id/sessions/revoke (revoke all).

Minting a session

POST /v1/sessions returns the credentials in-body — the sk_ caller already holds full power, so this saves the browser redeem/exchange hops. Store the refresh_token to keep the session alive.

```
curl https://api.atlas.dev/v1/sessions \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"user_id":"user_2a"}'
```

```
{
  "object": "session",
```

```
  "id": "sess_…",
  "user_id": "user_2a…",
  "jwt": "eyJhbGciOiJSUzI1Ni$#…",
  "refresh_token": "…",
  "expires_in": 60
}
```

Impersonation

Pass an actor.sub to mint an impersonation session; the minted session carries the RFC 8693 act claim naming the impersonator, and the action is always audited.

```
const s = await atlas.sessions.create({ user_id: 'user_2a', actor: { sub:
'user_admin' } });
```

For a redeemable impersonation credential instead of a live session, use POST /v1/actor_tokens (scope impersonation:write) or the one-time POST /v1/sign_in_tokens (scope sign_in_tokens:write).

Listing & revoking

```
const page = await atlas.sessions.list({ user_id: 'user_2a' });
await atlas.sessions.revoke('sess_123');

// Revoke everything a user holds (also bumps sessions_version)
await atlas.users.revokeSessions('user_2a');
```

SCIM directory (inbound)

SCIM directory (inbound)

Atlas is a SCIM 2.0 Service Provider: an enterprise customer's IdP (Okta, Entra ID, etc.) provisions and deprovisions users and groups into one organization's directory. Authentication is a per-organization SCIM bearer token.

Mint a SCIM token (BAPI)

Method & path	Scope	Notes
GET /v1/scim_tokens	scim:read	Lists the recognisable prefix only, never the token
POST /v1/scim_tokens	scim:write	Mint for an organization. Token revealed once. idempotent
POST /v1/scim_tokens/:id/revoke	scim:write	Revoke; provisioning fails immediately (record kept). idempotent

```
const t = await atlas.scimTokens.create({ organization_id: 'org_9f', name:
'Okta' });
console.log(t.secret); // shown once — paste into the IdP's SCIM config
```

SCIM 2.0 endpoints

Called by the IdP with Authorization: Bearer <scim token>:

```
GET    /scim/v2/Users            !’ list (filter by userName eq; index-paginated)
POST   /scim/v2/Users            !’ provision a user + seat them in the token's
org directory
GET    /scim/v2/Users/:id        !’ fetch (non-members are 404)
PUT    /scim/v2/Users/:id        !’ replace (including the active/deprovision
state)
PATCH  /scim/v2/Users/:id        !’ PatchOp; active:false deprovisions and
revokes every session
DELETE /scim/v2/Users/:id        !’ deprovision: revoke sessions + remove the
directory membership

GET    /scim/v2/Groups           !’ list (filter by displayName eq; index-
paginated)
POST   /scim/v2/Groups           !’ create a group with an optional initial
member set
GET    /scim/v2/Groups/:id       !’ fetch with member references
PUT    /scim/v2/Groups/:id       !’ replace displayName + full membership
PATCH  /scim/v2/Groups/:id       !’ add/remove members or rename
DELETE /scim/v2/Groups/:id       !’ delete (membership rows cascade)
```

Discovery

```
GET /scim/v2/ServiceProviderConfig   !’ supported features + auth scheme
GET /scim/v2/ResourceTypes           !’ User and Group resource types
GET /scim/v2/Schemas                 !’ core User and Group schemas
```

Groups drive roles

A SCIM group can be granted an org role (see Organizations !’ group role grants), so your

customer's directory groups map onto Atlas roles automatically.

Outbound provisioning

To push users from Atlas to a downstream SCIM endpoint (the reverse direction), see Outbound SCIM provisioning.

Account management (me)

Account management (me)

The me surface lets a signed-in user manage their own account. It requires the user's session (or a personal access token) — never a secret key. Build a custom account UI on these, or use @atlas/react's <UserProfile /> and hooks.

Profile

```
GET   /v1/client/me                !’ the user with emails, linked accounts, passkeys
PATCH /v1/client/me                !’ update profile + unsafe_metadata (public_metadata refused)
POST  /v1/client/me/avatar         !’ upload avatar (raw image bytes as the body)
POST  /v1/client/me/change_password!’ verify current, apply policy, revoke other sessions
POST  /v1/client/me/set_password   !’ set a first password (OAuth-only / guest account)
POST  /v1/client/me/reauthenticate !’ step-up re-auth to refresh the step-up window in place
```

Email addresses & identities

```
POST   /v1/client/me/email_addresses                          !’ add an address (sends a code)
POST   /v1/client/me/email_addresses/:id/attempt_verification!’ verify with the code
POST   /v1/client/me/email_addresses/:id/primary              !’ make a verified address primary
DELETE /v1/client/me/email_addresses/:id                      !’ remove (last verified one refused)

POST   /v1/client/me/external_accounts/connect                !’ link a new provider ("Connect GitHub")
POST   /v1/client/me/external_accounts/:id/reauthorize        !’ request additional scopes
POST   /v1/client/me/external_accounts/:id/revoke             !’ revoke the stored provider token (keep the link)
DELETE /v1/client/me/external_accounts/:id                    !’ unlink (removing the only sign-in method refused)
GET    /v1/client/me/external_accounts/:provider/token        !’ the user's own provider token (refreshed if stale)
```

Multi-factor

```
GET    /v1/client/me/mfa                    !’ list factors (never secrets)
POST   /v1/client/me/mfa/totp               !’ begin authenticator enrollment (secret shown once)
POST   /v1/client/me/mfa/totp/:id/verify    !’ confirm; receive recovery codes once
POST   /v1/client/me/mfa/sms                !’ enroll a phone as SMS second factor
POST   /v1/client/me/mfa/sms/:id/verify     !’ confirm SMS enrollment
POST   /v1/client/me/mfa/push               !’ register a device for push MFA (number-matching)
POST   /v1/client/me/mfa/backup_codes       !’ issue a fresh set of recovery codes
DELETE /v1/client/me/mfa/:id                !’ remove a factor (last one also drops recovery codes)

GET    /v1/client/me/passkeys               !’ list passkeys
POST   /v1/client/me/passkeys/begin | /finish!’ register a passkey (WebAuthn)
GET    /v1/client/me/trusted_devices        !’ browsers trusted to skip 2FA
```

Organizations (from the user's side)

```
GET  /v1/client/me/organizations               !’ the user's memberships
GET  /v1/client/me/organization_suggestions     !’ orgs discoverable via verified
email domain
POST /v1/client/me/organization_membership_requests !’ request to join a matching
org
POST /v1/client/organizations                    !’ create an org (if the
instance allows it)
POST /v1/client/organization_invitations/:invitationId/accept !’ accept an
invitation
```

Managing an active org (invite, update, list members) requires the relevant org:sys_* permission — see Roles & permissions.

Privacy (GDPR self-service)

```
POST /v1/client/me/data_export        !’ request an export of your own data
(async job)
GET  /v1/client/me/data_export/:id    !’ the produced package once ready (no
secrets/hashes)
POST /v1/client/me/deletion_request   !’ request account erasure (grace/
cancellation window)
POST /v1/client/me/deletion_request/:id/cancel !’ cancel within the window
```

Example

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

// Prebuilt UI
<UserProfile />;

// Or read the user directly
const { user } = useUser();
console.log(user?.primaryEmailAddress, user?.publicMetadata);
```

Backend SDKs

Backend SDKs

Backend SDKs do two jobs: manage your instance over the secret-key BAPI, and verify session JWTs locally against JWKS. The base URL defaults to https://api.atlas.dev and is overridable.

Node — @atlas/backend

```
npm install @atlas/backend
```

```
import { createAtlasClient, AtlasBackend, paginate } from '@atlas/backend';

// Management
const atlas = createAtlasClient({ secretKey: process.env.ATLAS_SECRET_KEY! });
const user = await atlas.users.create({ email_address: 'ada@example.com' },
'signup-ada-001');
for await (const org of paginate(atlas.organizations.list))
console.log(org.name);

// Verification
const backend = new AtlasBackend({
  jwksUrl: 'https://accounts.yourapp.com/.well-known/jwks.json',
  issuer: 'https://accounts.yourapp.com',
});
const auth = await backend.authenticateRequest(req);
if (auth.ok) auth.protect({ permission: 'org:billing:manage' });
```

Python — atlas-backend

Sync AtlasClient and async AsyncAtlasClient.

```
pip install atlas-backend
```

```
from atlas_backend import AtlasClient, AtlasError, paginate

atlas = AtlasClient("sk_live_...", base_url="https://api.atlas.dev")

user = atlas.users.create(
    {"email_address": "ada@example.com", "first_name": "Ada"},
    idempotency_key="signup-ada-001",
)

page = atlas.organizations.list(limit=20)
print(page["data"], page["has_more"], page["next_cursor"])
```

Go — atlas-go

Idiomatic, stdlib-only client plus local JWT verification via AtlasBackend.

```
go get github.com/atlas-auth/atlas-go
```

```
package main

import (
’&6öçFW‡B
’&f×B
’&Æör

– FÆ 2 &v—F‡V"æ6öÒö FÆ 2Ö WF,ö FÆ 2Övò
)

func main() {
–2 £Ò  FÆ 2äæWr,'6µöÆ—fUòâââ"Â  FÆ 2åv—F„& 6UU$Â,&‡GG 3¢òö  ’æ FÆ 2æFWb"’•
—W6W"Â W'" £Ò 2åW6W'2ä7&V FR†6öçFW‡Bä& 6¶w&÷VæB,’Â  FÆ 2ä7&V FUW6W%  & ×7°
™EmailAddress: "ada@example.com",
™FirstName:    "Ada",
—Ò•
––b W'"  Ò æ–Â °
```

```
™log.Fatal(err)
—Ð
–f×Bå &–çFÆâ,&7&V FVB"Â W6W"ä”B•
}
```

Token verification: atlas.NewAtlasBackend(...) then backend.Verify(ctx, token).

Ruby — atlas-auth

Ruby 3.0+.

```
# Gemfile
gem "atlas-auth"
```

```
require "atlas"

atlas = Atlas::Client.new(ENV.fetch("ATLAS_SECRET_KEY")) # "sk_live_..."

user = atlas.users.create(email_address: "ada@example.com", password: "••••••••")
page = atlas.users.list(limit: 20)
Atlas.paginate(atlas.organizations).each { |org| puts org["name"] }
```

PHP — atlas-auth/atlas-php

PSR-18/PSR-17 (Guzzle default) plus a local SessionVerifier. PHP 8.1+.

```
composer require atlas-auth/atlas-php
```

```
use Atlas\Client;

$atlas = new Client('sk_live_…');
$user  = $atlas->users->get('user_123');

$page = $atlas->users->list(['limit' => 50]);
foreach ($page['data'] as $u) { /* … */ }
```

Java — net.atlasauth:atlas-java

Typed AtlasClient plus a local SessionVerifier. Java 17+.

```
// Gradle
implementation("net.atlasauth:atlas-java:1.0.0")
```

```
<!-- Maven -->
<dependency>
  <groupId>net.atlasauth</groupId>
  <artifactId>atlas-java</artifactId>
  <version>1.0.0</version>
</dependency>
```

```
import net.atlasauth.atlas.AtlasClient;
import net.atlasauth.atlas.models.User;

AtlasClient atlas = AtlasClient.builder()
        .secretKey(System.getenv("ATLAS_SECRET_KEY"))
        .build();

User user = atlas.users().get("usr_123");
```

.NET — Atlas.Sdk

AtlasClient (management) and AtlasBackend (local JWT verification). Targets netstandard2.1 and net8.0.

```
dotnet add package Atlas.Sdk
```

```
using Atlas;
using Atlas.Resources;

using var atlas = new AtlasClient("sk_live_...");
User user = await atlas.Users.GetAsync("user_123");

User created = await atlas.Users.CreateAsync(
    new CreateUserBody { EmailAddress = "ada@example.com" },
    idempotencyKey: "signup-8f3c");
```

Auth Config

Auth Config

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/auth_config		The instance auth config, with the bounds the UI must respect.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/auth_config		Patch one or more config sections. Owner or admin; every change is audited.

Events

Events

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/events	events:read		Read the product activity feed (events), filterable by type and time and cursor-paginated, with the list of event types this instance has actually emitted.

Quickstart

Quickstart

This walks the two most common first steps: managing users from your backend, and gating a request by a verified session.

1. Call the Backend API

Grab your secret key from the dashboard (API keys), then create and list users:

```
# Create a user
curl https://api.atlas.dev/v1/users \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: signup-ada-001" \
  -d '{"email_address":"ada@example.com","first_name":"Ada"}'

# List users (one cursor page)
curl "https://api.atlas.dev/v1/users?limit=20" \
  -H "Authorization: Bearer sk_live_xxx"
```

With an SDK:

```
import { createAtlasClient } from '@atlas/backend';

const atlas = createAtlasClient({ secretKey: process.env.ATLAS_SECRET_KEY! });

const user = await atlas.users.create(
  { email_address: 'ada@example.com', first_name: 'Ada' },
  'signup-ada-001', // Idempotency-Key
);

const page = await atlas.users.list({ limit: 20 });
console.log(page.data, page.has_more, page.next_cursor);
```

2. Verify a session on your API

Your frontend sends the Atlas session JWT to your API. Verify it locally against your instance's JWKS — fast, offline, correct within the token lifetime:

```
import { AtlasBackend } from '@atlas/backend';

const atlas = new AtlasBackend({
  jwksUrl: 'https://accounts.yourapp.com/.well-known/jwks.json',
  issuer: 'https://accounts.yourapp.com',
});

const auth = await atlas.authenticateRequest(req);
if (!auth.ok) return res.status(401).end();

// Gate the action by a permission (throws ForbiddenError if unmet)
auth.protect({ permission: 'org:billing:manage' });
```

For the rare request where "signed out a moment ago" must mean now, call the authoritative check POST /v1/tokens/verify, which consults the revocation set. See Token verification.

3. Mount a frontend

```
// React
import { AtlasProvider } from '@atlas/react';

<AtlasProvider publishableKey="pk_live_xxx">
  <App />
</AtlasProvider>
```

Next: read Authentication for the full key model, then jump to the Backend API overview or the SDKs.

Outbound SCIM provisioning

Outbound SCIM provisioning

The reverse of inbound SCIM: Atlas is the SCIM client, pushing users and organizations to downstream SCIM 2.0 endpoints (a customer's app, a directory). The downstream bearer is write-only; the cursor starts at the current event so only new users forward — use sync_user to backfill.

Endpoints

Method & path	Scope	Notes
GET /v1/scim_provisioning_targets	scim_provisioning:read	List targets (reports has_bearer_token, never the value)
POST /v1/scim_provisioning_targets	scim_provisioning:write	Create (base_url must be https; bearer write-only). idempotent
GET /v1/scim_provisioning_targets/:id	scim_provisioning:read	Sync status, cursor, last error
PATCH /v1/scim_provisioning_targets/:id	scim_provisioning:write	Update mapping, deprovision action, status
DELETE /v1/scim_provisioning_targets/:id	scim_provisioning:write	Delete; pushing stops immediately
POST /v1/scim_provisioning_targets/:id/test	scim_provisioning:write	Verify connectivity/auth (GET ServiceProviderConfig) with the real bearer
POST /v1/scim_provisioning_targets/:id/sync_user	scim_provisioning:write	Force one user's sync (first = POST, re-sync = PUT)
POST /v1/scim_provisioning_targets/:id/sync_group	scim_provisioning:write	Force one org's sync to /Groups (only already-provisioned members)

Downstream failures are reported, never thrown — a flaky target never breaks an Atlas operation.

Example

```
const target = await atlas.scimProvisioning.create({
  name: 'Customer App',
  base_url: 'https://app.customer.com/scim/v2',
  bearer_token: '…', // write-only
} as any);

await atlas.scimProvisioning.test(target.id);      // check connectivity/auth
await atlas.scimProvisioning.syncUser(target.id, 'user_2a'); // backfill one
user
```

Assets

Assets

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/assets/*		Public read for an uploaded image (avatar/logo) — streamed from object storage with an immutable long cache-control. Unauthenticated by design; only validated images are ever stored here.

Organizations

Organizations

Organizations model B2B tenancy: a set of members with roles, optional email-domain auto-join, a security policy, and (for B2B2B) a parent/child hierarchy.

```
{
  "object": "organization",
  "id": "org_9f…",
  "name": "Acme",
  "slug": "acme",
  "image_url": null,
  "public_metadata": {},
  "max_allowed_memberships": 100,
  "created_by": "user_2a…",
  "created_at": 1757000000000,
  "updated_at": 1757600000000
}
```

Core endpoints

Method & path	Scope	Notes
GET /v1/organizations	organizations:read	List, cursor-paginated
POST /v1/organizations	organizations:write	Create + seat the creator as admin
GET /v1/organizations/:id	organizations:read	Never includes private_metadata
PATCH /v1/organizations/:id	organizations:write	Update name/slug/image/seat cap/metadata. Emits organization.updated. idempotent
DELETE /v1/organizations/:id	organizations:write	Delete; memberships + invitations cascade. idempotent
PUT /v1/organizations/:id/metadata	organizations:write	Replace metadata bags wholesale. idempotent
PATCH /v1/organizations/:id/policy	organizations:write	Set requireMfa, ssoRequired, sessionIdleOverrideMs
PUT /v1/organizations/:id/parent	organizations:write	Set/clear parent (B2B2B); refuses a cycle
GET /v1/organizations/:id/hierarchy		

organizations:read
Ancestor chain + direct children
GET /v1/organizations/:id/entitlements
organizations:read
Resolved plan + feature keys

Memberships

Method & path
Scope
Notes
GET /v1/organizations/:id/memberships
organizations:read
Members with roles
POST /v1/organizations/:id/memberships
organizations:write
Add an existing user with a role (seat cap enforced). idempotent
PATCH /v1/organizations/:id/memberships/:userId
organizations:write
Change a role. Demoting the last admin !’ LAST_ADMIN.
DELETE /v1/organizations/:id/memberships/:userId
organizations:write
Remove a member. Removing the last admin !’ LAST_ADMIN.

Invitations

Method & path
Scope
Notes
GET /v1/organizations/:id/invitations
organizations:read
Invitations with roles + status
POST /v1/organizations/:id/invitations
organizations:write
Invite an email to an org (expires in 7 days)
POST /v1/organizations/:id/invitations/:invitationId/revoke
organizations:write
Revoke a pending invitation

Email domains (auto-join)

Method & path
Scope
Notes
GET /v1/organizations/:id/domains
organizations:read
Claimed domains + verification state
POST /v1/organizations/:id/domains

organizations:write

Claim a domain; returns the DNS TXT record to publish. idempotent

POST /v1/organizations/:id/domains/:domainId/verify

organizations:write

Check the TXT record now, mark verified

DELETE /v1/organizations/:id/domains/:domainId

organizations:write

Remove a claimed domain

Group !’ role grants

Directory groups (from SCIM) can be granted an org role; members inherit its permissions (augment-only).

Method & path

Scope

PUT /v1/organizations/:orgId/groups/:groupId/roles/:roleId

organizations:write

DELETE /v1/organizations/:orgId/groups/:groupId/roles/:roleId

organizations:write

Examples

```
curl https://api.atlas.dev/v1/organizations \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"name":"Acme","slug":"acme","created_by":"user_2a"}'
```

```
const org = await atlas.organizations.create({ name: 'Acme', slug: 'acme',
created_by: 'user_2a' });
await atlas.organizations.memberships.add(org.id, { user_id: 'user_zoe', role:
'org:member' });
await atlas.organizations.memberships.update(org.id, 'user_zoe', { role:
'org:admin' });

// Claim a domain for auto-join, then verify after publishing the TXT record
const domain = await atlas.organizations.domains.create(org.id, { domain:
'acme.com', auto_join: true });
console.log(domain.verification); // { record_name, record_type: 'TXT',
record_value }
await atlas.organizations.domains.verify(org.id, domain.id);

// Enforce MFA + SSO for the org
await atlas.organizations.updatePolicy(org.id, { requireMfa: true, ssoRequired:
true });
```

Embed Js

Embed Js

Public — no authentication required.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /embed.js		Serves the built @atlas/embed IIFE bundle so a tenant can drop in <script src=".../embed.js"> with no build step.

Messaging Providers

Messaging Providers

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/messaging_providers	messaging:read		Read the instance's BYOK email and SMS provider config: transport, from-address and which secret keys are set. Never returns a secret value.
PUT /v1/messaging_providers/email	messaging:write		Configure the email delivery provider (Resend/SES/SMTP/Brevo/Mailgun). The provider secret is write-only; an omitted secret keeps the stored one.
PUT /v1/messaging_providers/sms	messaging:write		Configure the SMS delivery provider (Twilio). The auth token is write-only.
DELETE /v1/messaging_providers/:channel	messaging:write		Remove the email or sms provider configuration for the instance.
POST /v1/messaging_providers/:channel/test	messaging:write		Send a clearly-marked test message to a destination to verify the configured email or SMS provider delivers, without triggering a real auth flow. Rate-limited; 422 when nothing is configured.

Rate limits

Rate limits

Atlas rate-limits to protect instances from abuse and credential-stuffing. Limits apply per secret key on the BAPI, and per IP on the sensitive FAPI flows (sign-in, sign-up create, verification-code sends).

The 429 response

When you exceed a limit you get 429 with the RATE_LIMITED code and a Retry-After header (seconds). The wait is also in meta.retry_after:

```
{
  "errors": [
    {
      "code": "RATE_LIMITED",
      "message": "Too many requests. Please retry later.",
      "meta": { "retry_after": 30 }
    }
  ]
}
```

Back off and retry after the indicated delay; jitter your retries so a fleet does not synchronize.

Sensitive-flow limits

Some flows are limited more tightly by design and independently of the global budget — for example verification-code reissue is capped at 1 per 30 seconds and 5 per hour per address. Exceeding these returns TOO_MANY_ATTEMPTS rather than a plain RATE_LIMITED.

Tuning the policy

The per-instance budgets are configurable (sign-in / sign-up create per IP, and BAPI per key):

```
# Read the policy and its bounds
curl https://api.atlas.dev/v1/rate_limit_policy \
  -H "Authorization: Bearer sk_live_xxx"     # scope: rate_limit:read

# Tighten it (values are bounded — protection cannot be disabled or set to self-
DoS)
curl -X PATCH https://api.atlas.dev/v1/rate_limit_policy \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{ "…": "…" }'                              # scope: rate_limit:write
```

Defaults equal the built-in fixed limits; a lower value tightens the limiter. You cannot raise a budget high enough to disable protection.

Related controls

- Network ACLs (/v1/network_acls) — per-instance IP allow/deny rules that gate access before rate limiting even applies.
- Attack protection (/v1/attack_protection) — brute-force lockout and breached-password checks.
- Adaptive (risk-based) MFA (/v1/risk_based_mfa) — step up when a sign-in looks risky.

Sessions & tokens (frontend)

Sessions & tokens (frontend)

On the client, the SDK manages the session for you — but the underlying FAPI endpoints are here for custom UIs and to understand what getToken() does.

The session lifecycle

```
GET  /v1/client                        !’ boot; session: null when signed out
(never 401)
GET  /v1/client/sessions               !’ the user's own active sessions/devices
(flags the current)
POST /v1/client/sessions/:id/touch     !’ switch the active organization
(membership resolved server-side)
POST /v1/client/sessions/:id/tokens    !’ rotate the refresh token and mint a
session JWT
POST /v1/client/sessions/:id/revoke    !’ revoke one session and clear its cookies
POST /v1/client/sessions/revoke_all    !’ sign out every OTHER device; bumps
sessions_version
```

Getting a token for your own API

Your frontend calls your backend with a short-lived Atlas session JWT. The SDK's getToken() mints/refreshes it:

```
import { useAuth } from '@atlas/react';

const { getToken } = useAuth();

// Standard session JWT
const token = await getToken();
await fetch('/api/things', { headers: { Authorization: `Bearer ${token}` } });
```

Your backend then verifies it — see Session & JWT verification.

Template tokens

For a downstream that needs bespoke claims (Hasura, RLS, a partner API), mint a token from a named JWT template:

```
POST /v1/client/sessions/:id/tokens/:template   !’ mint a JWT for the active
session using a named template
```

```
const hasuraToken = await getToken({ template: 'hasura' });
```

Token lifetimes

Session JWTs are short-lived (about 60 seconds) and rotated transparently by the SDK from the refresh token. This is what makes local JWKS verification safe: a revoked session stops working within one lifetime. For operations that cannot tolerate that window, verify authoritatively with POST /v1/tokens/verify.

CIBA (backchannel) approvals

For decoupled flows where approval happens on another device:

```
GET  /v1/client/me/backchannel_requests       !’ pending CIBA approval requests
POST /v1/client/me/backchannel_requests/:id    !’ approve or deny (session-only;
a PAT cannot approve)
```

Captcha Secret

Captcha Secret

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/captcha_secret		Store the captcha provider secret. Write-only; never returned. Owner or admin.

Mobile SDKs

Mobile SDKs

Native SDKs authenticate with a publishable key over FAPI and persist the session in platform-secure storage (Keychain / EncryptedSharedPreferences). They mirror @atlas/js: a bare frontendApi host is upgraded to https://. For React Native, see @atlas/react-native.

Swift (iOS / macOS)

Dependency-light Swift Package over FAPI (URLSession, async/await, Codable). Persists the session JWT + refresh to the Keychain.

```
// Package.swift
.package(url: "https://github.com/atlas-auth/atlas-swift", from: "0.1.0")
// or Xcode !’ File !’ Add Package Dependencies
```

```
import Atlas

let atlas = AtlasClient(
    publishableKey: "pk_live_…",
    frontendApi: "accounts.your-domain.com"   // bare host upgraded to https://
)

// Password sign-in: create attempt !’ attempt first factor !’ exchange ticket
let user = try await atlas.signIn(email: "ada@example.com", password: "…")

let me = try await atlas.currentUser()
try await atlas.refresh()   // rotate the token before expiry / on a 401 retry
try await atlas.signOut()   // revokes server-side and clears the Keychain
```

Kotlin / Android

OkHttp + coroutines + kotlinx.serialization. Publishes as net.atlasauth:atlas-android. create wires an encrypted token store namespaced by the publishable key.

```
// build.gradle.kts (app module)
dependencies {
    implementation("net.atlasauth:atlas-android:0.1.0")
}
```

```
import net.atlasauth.atlas.AtlasClient

val atlas = AtlasClient.create(
    context = applicationContext,
    publishableKey = "pk_live_…",
    frontendApi = "accounts.your-domain.com",
)

lifecycleScope.launch {
    val user = atlas.signIn(email = "ada@example.com", password = "…")
    val me = atlas.currentUser()
    atlas.refresh()
    atlas.signOut()
}
```

Flutter / Dart

Cross-platform FAPI client with the publishable key.

```
flutter pub add atlas_auth
```

```
import 'package:atlas_auth/atlas_auth.dart';

final client = AtlasClient(
  publishableKey: 'pk_live_…',
  frontendApi: 'accounts.example.com', // bare host upgraded to https://
);
```

```
try {
  final user = await client.signIn(email: 'ada@example.com', password:
'hunter2');
  print('Signed in as ${user.firstName} (${user.id})');
} on AtlasException catch (e) {
  if (e.code == 'form_password_incorrect') {
    print('Wrong password: ${e.message}');
  } else {
    rethrow;
  }
}
```

Native sign-in helpers

The mobile SDKs support native identity flows over FAPI: Google One-Tap / Apple / Facebook Limited Login via POST /v1/client/sign_ins/id_token, passkeys (WebAuthn), and the standard email/password + MFA attempt flow. See Sign-in & sign-up.

Kerberos Secret

Kerberos Secret

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/kerberos_secret		Store the per-instance Kerberos/IWA trusted-proxy secret. Write-only, encrypted; never returned. Refused while the kerberos strategy is off. Owner or admin.

Frontend endpoint index

Frontend endpoint index

The complete Frontend API surface — 122 endpoints. FAPI is browser- and native-facing: authenticated with a publishable key (x-publishable-key) plus the Atlas session (cookie on web, bearer on native). Most routes live under /v1/client/…. These are normally driven by an SDK, not called by hand.

Other client routes

Method & path	Description
GET /v1/appearance	The embeddable widget's public themeenabled providers/strategies, resolved by publishable key or host. Public only — no secrets, no customCss/customHtml.
POST /v1/client/telemetry	Anti-bot signal beacon: the SDK posts anonymised devicebehavioural signals for the sign-in/sign-up form; the server enriches (salt-hashed IP, coarse geo, heuristic score) and appends to the bot_signals lake. Fire-and-forget (202), never blocks a sign-in.
GET /v1/assets/*	Public read for an uploaded image (avatar/logo) — streamed from object storage with an immutable long cache-control. Unauthenticated by design; only validated images are ever stored here.
GET /v1/client/environment	The instance's enabled social providers (and native client ids) for a publishable-key client. Never exposes secrets.

Sign-in & sign-up

Method & path	Description
POST /v1/client/sign_ups	Start an emailpassword sign-up. Returns an attempt whose status drives the next step.
POST /v1/client/sign_ups/:id/prepare_verification	Reissue a verification code. Rate limited to 1/30s and 5/hour per address.
POST /v1/client/sign_ups/:id/attempt_verification	Submit an email verification code. Three attempts, then the code is invalidated.
POST /v1/client/password_resets	Request a reset. Identical response for an address that does not exist.
POST /v1/client/password_resets/:id/attempt_verification	Submit the emailed code. Advances to MFA when the account has it.
POST /v1/client/password_resets/:id/attempt_second_factor	Second factor during a reset. Reset never bypasses MFA.
POST /v1/client/password_resets/:id/set_new_password	Set the new password, revoke every session, and bump sessions_version.
POST /v1/client/sign_ins	Start a sign-in. Returns the instance factor list, identical for unknown identifiers.
POST /v1/client/qr_sign_ins	

Start a QR cross-device sign-in: returns the QR token/verification URL, a number-match code, and this tab's poll secret. Opt-in; unauthenticated (publishable key).

GET /v1/client/qr_sign_ins/:qr_token

Phone lookup for a pending QR sign-in: the requesting browser's devicecoarse location and the match code to confirm. Requires the phone's session.

POST /v1/client/qr_sign_ins/:qr_token/approve

Phone approves a pending QR sign-in after confirming the number-match code; completes the attempt so the browser's poll mints a session. Requires the phone's session.

POST /v1/client/qr_sign_ins/:qr_token/deny

Phone denies a pending QR sign-in, abandoning the attempt so the browser's poll never completes. Requires the phone's session.

POST /v1/client/sign_ins/:id/prepare_first_factor

Send an email code or magic link. Returns the polling secret to this tab.

GET /v1/client/sign_ins/verify

Magic-link target. Completes the attempt; signs in no one here.

GET /v1/client/sign_ins/:id

Poll an attempt with its poll_secret; returns a ticket once complete.

POST /v1/client/sign_ins/:id/attempt_first_factor

Submit a first factor. Advances to MFA or completes, as the server decides.

POST /v1/client/sign_ins/:id/prepare_second_factor

Offer a passkey as the second factor. Credentials are named safely here.

POST /v1/client/sign_ins/:id/prepare_mfa_enrollment

Start TOTP enrollment for an attempt parked by a required MFA policy.

POST /v1/client/sign_ins/:id/attempt_mfa_enrollment

Confirm enrollment with two consecutive codes and complete the sign-in.

POST /v1/client/sign_ins/:id/attempt_second_factor

Submit a TOTP or recovery code. The only route out of needs_second_factor.

POST /v1/client/sign_ins/passkey/begin

Begin passwordless sign-in. Names no user and lists no credentials.

POST /v1/client/sign_ins/passkey/finish

Verify an assertion and issue a session. The credential names the user.

POST /v1/client/sign_ins/oauth

Begin an OAuth sign-in and receive the provider authorization URL.

POST /v1/client/sign_ins/id_token

Native / One-Tap sign-in: verify a provider id_token (Google GSI, Apple, Facebook Limited Login) and complete the sign-in, honouring the MFA gate.

POST /v1/client/sign_ins/ldap

LDAP/AD inbound sign-in: authenticate against a configured directory (connection_idusernamepassword) and JIT create/link the Atlas account, honouring the MFA gate.

POST /v1/client/sign_ins/kerberos

Kerberos / Integrated Windows Auth (IWA / SPNEGO) sign-in: a reverse proxy authenticates the principal and forwards it in a header; Atlas trusts it only behind a matching x-atlas-kerberos-secret, then resolves/JIT-links the account honouring the MFA gate. Opt-in, fail-closed.

POST /v1/client/sign_ins/external_jwt

Migration interop: verify a foreign provider JWT (a configured externalJwt issuer) against its JWKS and complete the sign-in (JITMFA gate), to run Atlas alongside an old provider.

POST /v1/client/sign_ins/sso

Begin an enterprise SSO sign-in. Resolves an OIDC connection by id or email domain and returns the IdP authorization URL.

POST /v1/client/sign_ins/saml

Begin an enterprise SAML sign-in. Resolves a SAML connection by id or email domain and returns the IdP AuthnRequest URL.

Account management (me)

Method & path

Description

POST /v1/client/me/avatar

Upload the signed-in user's avatar. The RAW image bytes are the body (Content-Type image/png|jpeg|gif|webp), validated by magic number; sets the user's image_url. Returns 503 when image storage is not configured.

GET /v1/client/me/passkeys

List the signed-in user passkeys. Never returns credential material.

POST /v1/client/me/passkeys/begin

Options for navigator.credentials.create, with a single-use challenge.

POST /v1/client/me/passkeys/finish

Verify and store a new passkey. Requires a session.

PATCH /v1/client/me/passkeys/:id

Rename a passkey so the user can tell their devices apart.

DELETE /v1/client/me/passkeys/:id

Remove a passkey. Another user passkey id is 404, never 403.

GET /v1/client/me/trusted_devices

List browsers trusted to skip 2FA (). Never returns the token hash.

DELETE /v1/client/me/trusted_devices/:id

Revoke one trusted device; its next sign-in must pass 2FA again.

DELETE /v1/client/me/trusted_devices

Revoke every trusted device for the signed-in user.

GET /v1/client/me/oauth_grants

List the third-party apps this user authorized (Sign in with <Tenant>).

DELETE /v1/client/me/oauth_grants/:id

Disconnect an authorized app: forget consent and revoke its live tokens.

GET /v1/client/me

The signed-in user with emails, linked accounts and passkeys.

POST /v1/client/me/api_tokens

Mint a scoped personal access token (uat_) so an agent or script can drive your own account over the me-surface with a bearer header. Session-only; the one-time secret is shown exactly once.

GET /v1/client/me/api_tokens

List your own personal access tokens with their scopes, expiry and last-used time. Never returns a secret or its hash.

DELETE /v1/client/me/api_tokens/:id

Revoke one of your own personal access tokens. Revocation is immediate and a revoked token fails closed on every later use.

GET /v1/client/me/mcp

Discover the user-scoped MCP tools your personal access token can use, plus the recommended scope set for an account-management agent.

POST /v1/client/me/mcp

JSON-RPC MCP endpoint: manage your own account as MCP tools (read/update profile, list sessions, revoke-all, list factors and more), authenticated by a personal access token.

POST /v1/client/me/data_export

Request an export of your own data (Article 15/20). Creates an async job; poll for the produced package. Rate-limited: one in flight, and a cooldown between requests.

GET /v1/client/me/data_export

List your own data-export requests and their status.

GET /v1/client/me/data_export/:id

A data-export request with, once ready, the produced package — your profile, emails, linked accounts, memberships and more, with no password hash, token or secret.

POST /v1/client/me/deletion_request

Request erasure of your own account (Article 17). Scheduled after a grace/cancellation window, then fulfilled through the same soft-deletePII-purge path a backend user-delete uses.

GET /v1/client/me/deletion_request

List your own erasure requests and their status.

POST /v1/client/me/deletion_request/:id/cancel

Cancel a pending erasure within the grace window. A fulfilled one is a 409.

DELETE /v1/client/me/deletion_request/:id

Cancel a pending erasure within the grace window (alias of /cancel).

PATCH /v1/client/me

Update profile and unsafe_metadata. public_metadata is refused.

POST /v1/client/me/change_password

Change your password. Verifies the current one, applies the sign-up policy, and revokes other sessions.

POST /v1/client/me/set_password

Set a first password on an account that has none (anonymous guest or OAuth-only). Applies the sign-up policy; may graduate a guest to a permanent account.

POST /v1/client/me/email_addresses

Add an address. Starts unverified and sends a code.

POST /v1/client/me/email_addresses/:id/attempt_verification

Verify an added address with its emailed code.

POST /v1/client/me/email_addresses/:id/primary

Make a verified address primary. Unverified addresses are refused.

DELETE /v1/client/me/email_addresses/:id

Remove an address. Removing the last verified one is refused.

DELETE /v1/client/me/external_accounts/:id

Unlink a provider. Removing the only sign-in method is refused.

GET /v1/client/me/external_accounts/:provider/token

The signed-in user's own provider access token, refreshed single-flight if stale. Never returns the refresh token.

POST /v1/client/me/external_accounts/connect

Start an OAuth flow to link a NEW provider to the signed-in user (for a custom "Connect GitHub" button); returns the authorize URL. Binds to the current user, so no new session and no email resolution.

POST /v1/client/me/external_accounts/:id/reauthorize

Re-initiate the provider OAuth flow to request additional scopes for a linked account; returns the authorize URL. On callback the account scopes/token update.

POST /v1/client/me/external_accounts/:id/revoke

Revoke the stored provider token (best-effort at the provider) and clear it locally WITHOUT unlinking the account.

GET /v1/client/me/organizations

Organization memberships for the signed-in user.

POST /v1/client/me/organization_membership_requests

Request to join an organization whose verified domain matches your verified email.

GET /v1/client/me/organization_suggestions

Organizations you could join, discovered via your verified email domain.

POST /v1/client/me/organization_suggestions/:organizationId/accept

Accept a suggestion, creating a pending join request an admin then decides.

POST /v1/client/me/reauthenticate

Step-up re-authentication: re-prove a credential (strategy "password" or "id_token") to refresh the session step-up window in place — no sign-out — so the next 2FA change is permitted. An id_token must already be linked to this user; it never attaches a new identity.

GET /v1/client/me/backchannel_requests

List the current user’s pending CIBA backchannel approval requests.

POST /v1/client/me/backchannel_requests/:id

Approve or deny a pending CIBA backchannel request (session-only; a PAT cannot approve).

Organizations (client)

Method & path

Description

POST /v1/client/organizations

Create an organization, if the instance allows user-created ones.

GET /v1/client/organizations/:id

Read the active organization. Gated by the ACTIVE org role.

PATCH /v1/client/organizations/:id

Update the active organization. Requires org:sys_profile:manage.

GET /v1/client/organizations/:id/memberships

List members of the active organization as a display directory.

POST /v1/client/organizations/:id/invitations

Invite someone. Requires org:sys_memberships:manage.

DELETE /v1/client/organizations/:id/memberships/:userId

Remove a member. Removing the last admin returns LAST_ADMIN.

POST /v1/client/organization_invitations/:invitationId/accept

Accept an invitation addressed to one of your verified emails. Seats you as a member.

GET /v1/client/organizations/:id/membership_requests

List pending requests to join the active org. Requires org:sys_memberships:manage.

POST /v1/client/organizations/:id/membership_requests/:reqId/accept

Accept a join request, seating the user at the default member role (never admin).

POST /v1/client/organizations/:id/membership_requests/:reqId/reject

Reject a pending join request. No membership is created.

Multi-factor (me)

Method & path

Description

GET /v1/client/me/mfa

List the signed-in user second factors. Never returns secrets or codes.

POST /v1/client/me/mfa/totp

Begin authenticator enrollment. Returns the secret exactly once.

POST /v1/client/me/mfa/totp/:id/verify

Confirm enrollment with a code, and receive recovery codes once.

POST /v1/client/me/mfa/sms

Enroll a phone as an SMS second factor. Texts a code; returns the number masked.

POST /v1/client/me/mfa/sms/:id/verify

Confirm SMS enrollment with the texted code, and receive recovery codes once.

POST /v1/client/me/mfa/push

Register a device for in-house push MFA. Pushes a number-matching challenge; never returns the device token.

GET /v1/client/me/mfa/push

List registered push devices. Never returns the raw device token.

POST /v1/client/me/mfa/push/challenges/:id/respond

Device approve/denyselected number for a push challenge (enrollment or sign-in).

DELETE /v1/client/me/mfa/push/:deviceId

Remove a push device. Removing the last verified one removes the push factor.

POST /v1/client/me/mfa/backup_codes

Issue a fresh set of recovery codes, invalidating the previous set.

DELETE /v1/client/me/mfa/:id

Remove a factor. Removing the last one also drops recovery codes.

Web3, SAML-IdP & other sign-in

Method & path

Description

GET /v1/saml/idp/sso

Atlas-as-SAML-IdP SSO endpoint (session-gated). IdP-initiated (?sp=) or SP-initiated (SAMLRequest); returns a signed SAML Response auto-POST form.

POST /v1/saml/idp/sso

Atlas-as-SAML-IdP SSO endpoint (session-gated), HTTP-POST binding — see GET /v1/saml/idp/sso.

PUT /v1/saml/idp/sso

Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.

PATCH /v1/saml/idp/sso

Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.

DELETE /v1/saml/idp/sso

Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.

POST /v1/oauth/telegram

Telegram Login Widget sign-in: verify the widget payload HMAC against the bot token and complete the sign-in, honouring the MFA gate.

POST /v1/oauth/siwe/nonce

Sign-in with Ethereum (EIP-4361): mint a single-use nonce bound to a new attempt.

POST /v1/oauth/siwe/verify

Sign-in with Ethereum (EIP-4361): verify a signed message (domain, chain, nonce, validity, EIP-191/EIP-1271) and complete the sign-in, honouring the MFA gate.

POST /v1/oauth/siws/nonce

Sign-in with Solana (SIWS): mint a single-use nonce bound to a new attempt.

POST /v1/oauth/siws/verify

Sign-in with Solana (SIWS): verify a signed message (domain, network, nonce, validity, ed25519 signature) and complete the sign-in, honouring the MFA gate.

POST /v1/oauth/siwf/nonce

Sign-in with Farcaster (SIWF): mint a single-use nonce bound to a new attempt.

POST /v1/oauth/siwf/verify

Sign-in with Farcaster (SIWF): verify a signed SIWE message (domain, chain, nonce, validity, signature), prove on-chain that the custody address owns the claimed FID (Optimism ID Registry), and complete the sign-in, honouring the MFA gate.

GET /discourse/sso

DiscourseConnect provider endpoint. Verifies the forum HMAC (constant-time), authenticates via the hosted session, and signs the identity payload back to the return_sso_url whose origin is pinned to the connection.

Tickets

Method & path

Description

POST /v1/client/tickets/exchange

Exchange a one-time ticket for session cookies.

Client & sessions

Method & path

Description

GET /v1/client

Boot the client. Returns session: null for a signed-out visitor, never 401.

GET /v1/client/sessions

List the signed-in user's own active sessions/devices, flagging the current one.

POST /v1/client/sessions/revoke_all

Sign out of every other device. Spares the current session; bumps sessions_version.

POST /v1/client/sessions/:id/touch

Switch the active organization. Membership is resolved server-side.

POST /v1/client/sessions/:id/tokens

Rotate the refresh token and mint a session JWT. Reads the refresh cookie only.

POST /v1/client/sessions/:id/revoke

Revoke one session and clear its cookies.

POST /v1/client/sessions/:id/tokens/:template

Mint a short-lived JWT for the active session using a named JWT template (getToken).

Billing (client)

Method & path

Description

GET /v1/client/billing/plans

Public list of the instance's active end-user billing plans for display (no Stripe price ids or secrets).

POST /v1/client/billing/checkout

Create a Stripe Checkout Session to subscribe the signed-in user (or an org they administer) to a plan. Returns {url}; never changes subscription state.

GET /v1/client/billing/subscription

Read the signed-in user's subscription, or an org's (org_id) when the caller is a member — status plus the active plan and its features.

POST /v1/client/billing/portal

Create a Stripe Billing Portal session for the subject's customer (self, or an org the caller administers) to manage or cancel. Returns {url}.

Client Me

Client Me

Frontend API — publishable key (x-publishable-key) plus the user session.

54 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/client/me/avatar		Upload the signed-in user's avatar. The RAW image bytes are the body (Content-Type image/png|jpeg|gif|webp), validated by magic number; sets the user's image_url. Returns 503 when image storage is not configured.
GET /v1/client/me/passkeys		List the signed-in user passkeys. Never returns credential material.
POST /v1/client/me/passkeys/begin		Options for navigator.credentials.create, with a single-use challenge.
POST /v1/client/me/passkeys/finish		Verify and store a new passkey. Requires a session.
PATCH /v1/client/me/passkeys/:id		Rename a passkey so the user can tell their devices apart.
DELETE /v1/client/me/passkeys/:id		Remove a passkey. Another user passkey id is 404, never 403.
GET /v1/client/me/trusted_devices		List browsers trusted to skip 2FA . Never returns the token hash.
DELETE /v1/client/me/trusted_devices/:id		Revoke one trusted device; its next sign-in must pass 2FA again.
DELETE /v1/client/me/trusted_devices		Revoke every trusted device for the signed-in user.
GET /v1/client/me/oauth_grants		List the third-party apps this user authorized (Sign in with <Tenant>).
DELETE /v1/client/me/oauth_grants/:id		

Disconnect an authorized app: forget consent and revoke its live tokens.

GET /v1/client/me

The signed-in user with emails, linked accounts and passkeys.

POST /v1/client/me/api_tokens

Mint a scoped personal access token (uat_) so an agent or script can drive your own account over the me-surface with a bearer header. Session-only; the one-time secret is shown exactly once.

GET /v1/client/me/api_tokens

List your own personal access tokens with their scopes, expiry and last-used time. Never returns a secret or its hash.

DELETE /v1/client/me/api_tokens/:id

Revoke one of your own personal access tokens. Revocation is immediate and a revoked token fails closed on every later use.

GET /v1/client/me/mcp

Discover the user-scoped MCP tools your personal access token can use, plus the recommended scope set for an account-management agent.

POST /v1/client/me/mcp

JSON-RPC MCP endpoint: manage your own account as MCP tools (read/update profile, list sessions, revoke-all, list factors and more), authenticated by a personal access token.

POST /v1/client/me/data_export

Request an export of your own data (Article 15/20). Creates an async job; poll for the produced package. Rate-limited: one in flight, and a cooldown between requests.

GET /v1/client/me/data_export

List your own data-export requests and their status.

GET /v1/client/me/data_export/:id

A data-export request with, once ready, the produced package — your profile, emails, linked accounts, memberships and more, with no password hash, token or secret.

POST /v1/client/me/deletion_request

Request erasure of your own account (Article 17). Scheduled after a grace/cancellation window, then fulfilled through the same soft-delete + PII-purge path a backend user-delete uses.

GET /v1/client/me/deletion_request

List your own erasure requests and their status.

POST /v1/client/me/deletion_request/:id/cancel

Cancel a pending erasure within the grace window. A fulfilled one is a 409.

DELETE /v1/client/me/deletion_request/:id

Cancel a pending erasure within the grace window (alias of /cancel).

PATCH /v1/client/me

Update profile and unsafe_metadata. public_metadata is refused.

POST /v1/client/me/change_password

Change your password. Verifies the current one, applies the sign-up policy, and revokes other sessions.

POST /v1/client/me/set_password

Set a first password on an account that has none (anonymous guest or OAuth-only). Applies the sign-up policy; may graduate a guest to a permanent account.

POST /v1/client/me/email_addresses

Add an address. Starts unverified and sends a code.

POST /v1/client/me/email_addresses/:id/attempt_verification

Verify an added address with its emailed code.

POST /v1/client/me/email_addresses/:id/primary

Make a verified address primary. Unverified addresses are refused.

DELETE /v1/client/me/email_addresses/:id

Remove an address. Removing the last verified one is refused.

DELETE /v1/client/me/external_accounts/:id

Unlink a provider. Removing the only sign-in method is refused.

GET /v1/client/me/external_accounts/:provider/token

The signed-in user's own provider access token, refreshed single-flight if stale. Never returns the refresh token.

POST /v1/client/me/external_accounts/connect

Start an OAuth flow to link a NEW provider to the signed-in user (for a custom "Connect GitHub" button); returns the authorize URL. Binds to the current user, so no new session and no email resolution.

POST /v1/client/me/external_accounts/:id/reauthorize

Re-initiate the provider OAuth flow to request additional scopes for a linked account; returns the authorize URL. On callback the account scopes/token update.

POST /v1/client/me/external_accounts/:id/revoke

Revoke the stored provider token (best-effort at the provider) and clear it locally WITHOUT unlinking the account.

GET /v1/client/me/organizations

Organization memberships for the signed-in user.

POST /v1/client/me/organization_membership_requests

Request to join an organization whose verified domain matches your verified email.

GET /v1/client/me/organization_suggestions

Organizations you could join, discovered via your verified email domain.

POST /v1/client/me/organization_suggestions/:organizationId/accept

Accept a suggestion, creating a pending join request an admin then decides.

GET /v1/client/me/mfa

List the signed-in user second factors. Never returns secrets or codes.

POST /v1/client/me/reauthenticate

Step-up re-authentication: re-prove a credential (strategy "password" or "id_token") to refresh the session step-up window in place — no sign-out — so the next 2FA change is permitted. An id_token must already be linked to this user; it never attaches a new identity.

POST /v1/client/me/mfa/totp

Begin authenticator enrollment. Returns the secret exactly once.

POST /v1/client/me/mfa/totp/:id/verify

Confirm enrollment with a code, and receive recovery codes once.

POST /v1/client/me/mfa/sms

Enroll a phone as an SMS second factor. Texts a code; returns the number masked.

POST /v1/client/me/mfa/sms/:id/verify

Confirm SMS enrollment with the texted code, and receive recovery codes once.

POST /v1/client/me/mfa/push

Register a device for in-house push MFA. Pushes a number-matching challenge; never returns the device token.

GET /v1/client/me/mfa/push

List registered push devices. Never returns the raw device token.

POST /v1/client/me/mfa/push/challenges/:id/respond

Device approve/deny + selected number for a push challenge (enrollment or sign-in).
DELETE /v1/client/me/mfa/push/:deviceId

Remove a push device. Removing the last verified one removes the push factor.
POST /v1/client/me/mfa/backup_codes

Issue a fresh set of recovery codes, invalidating the previous set.
DELETE /v1/client/me/mfa/:id

Remove a factor. Removing the last one also drops recovery codes.
GET /v1/client/me/backchannel_requests

List the current user’s pending CIBA backchannel approval requests.
POST /v1/client/me/backchannel_requests/:id

Approve or deny a pending CIBA backchannel request (session-only; a PAT cannot approve).

Instance

Instance

Backend API — secret key (Authorization: Bearer sk_…).

10 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/instance/security	instance:read		Read the instance kill switches (per-flow on/off) and the customer IP allowlist.
PATCH /v1/instance/security	instance:write		Set the instance kill switches and/or IP allowlist. Every CIDR is validated; a malformed one is refused rather than stored as a rule that never matches.
PUT /v1/instance/captcha_secret	instance:write		Store the captcha provider secret (write-only, encrypted; never read back). Refused while the captcha provider is none.
DELETE /v1/instance/captcha_secret	instance:write		Clear the stored captcha provider secret for the current provider.
PUT /v1/instance/kerberos_secret	instance:write		Store the per-instance Kerberos/IWA trusted-proxy secret (write-only, encrypted; never read back). Refused while the kerberos strategy is off.
DELETE /v1/instance/kerberos_secret	instance:write		Clear the stored Kerberos/IWA trusted-proxy secret for this instance.
PUT /v1/instance/ldap_connections/:connectionId/bind_password	instance:write		Store an LDAP connection service-account bind password (write-only, encrypted; never read back). The connection must exist in auth_config.ldap.connections.
DELETE /v1/instance/ldap_connections/:connectionId/bind_password			

instance:write

Clear the stored bind password for an LDAP connection.
GET /v1/instance
instance:read

Read instance configuration. Never returns the private signing key.
PATCH /v1/instance
instance:write

Update allowed origins and auth config. Wildcard origins are refused.

Well Known

Well Known

Public — no authentication required.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /.well-known/jwks.json		Public JWKS for the instance on this host. The documented path.
GET /.well-known/openid-configuration		OpenID Provider discovery document for this instance. Host-based issuer, S256 PKCE, RS256.

Self-service SSO

Self-service SSO

Rather than configuring every customer's IdP yourself, issue a scoped, one-time ticket that opens a hosted setup page — the customer's IT admin configures their own connection without a dashboard account. This mirrors the WorkOS Admin Portal / Auth0 self-service model.

1. Create a profile

A profile fixes which protocols a customer may configure and, optionally, binds an organization.

Method & path	Scope	Notes
GET /v1/sso_onboarding_profiles	sso_connections:read	List profiles
POST /v1/sso_onboarding_profiles	sso_connections:write	Create (allowed connection types + optional bound org). idempotent
GET /v1/sso_onboarding_profiles/:id	sso_connections:read	
DELETE /v1/sso_onboarding_profiles/:id	sso_connections:write	Deletes; issued tickets cascade

2. Issue a ticket

Method & path	Scope	Notes
POST /v1/sso_onboarding_profiles/:id/tickets	sso_connections:write	One-time, expiring ticket for one org. Token + hosted URL revealed once. idempotent
POST /v1/sso_onboarding_tickets/:id/revoke	sso_connections:write	Revoke a pending ticket

```
const profile = await atlas.ssoOnboarding.create({
  name: 'Enterprise SSO',
  allowed_connection_types: ['oidc', 'saml'],
  allow_scim: true,
});

const ticket = await atlas.ssoOnboarding.createTicket(profile.id, {
  organization_id: 'org_9f',
  expires_in_seconds: 3 * 24 * 3600, // clamped to 5 min … 30 days; default 3
days
});

// Send this URL to the customer's IT admin — no account needed:
```

```
console.log(ticket.url, ticket.token); // shown once
```

3. The customer configures it

The hosted flow (authenticated only by the ticket token) lets the admin create exactly the ticket-bound connection for the ticket-bound org — client-supplied ids are ignored, secrets are write-only:

```
GET  /hosted/sso-setup                    !’ the hosted setup page (ticket-
authenticated)
POST /hosted/sso-setup/save               !’ create/edit only the ticket-bound
connection
POST /hosted/sso-setup/scim-token         !’ mint a SCIM token scoped to the org
(if the profile allows SCIM)
POST /hosted/sso-setup/domains            !’ claim an email domain (returns the
DNS TXT record)
POST /hosted/sso-setup/domains/verify     !’ run DNS verification
POST /hosted/sso-setup/complete           !’ mark the ticket completed (no longer
usable)
```

Idempotency

Idempotency

Network calls fail halfway. Idempotency lets you retry a write without risking a duplicate — the same key replays the first result instead of performing the action twice.

How it works

Send an Idempotency-Key header (any unique string you choose — a UUID, or a natural key like signup-ada-001) on a supported POST:

```
curl https://api.atlas.dev/v1/users \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: signup-ada-001" \
  -d '{"email_address":"ada@example.com"}'
```

- The first request with a given key performs the action and records the response.
- A retry with the same key returns the recorded response — the user is created once.
- Reusing a key with a different body is a conflict: 409 IDEMPOTENCY_CONFLICT.

Which endpoints honour it

Idempotency applies to creates and state-changing mutations that would be unsafe to repeat. Every such endpoint is marked idempotent in this reference — for example:

- POST /v1/users, PATCH /v1/users/:id, POST /v1/users/:id/ban, DELETE /v1/users/:id
- POST /v1/organizations, PATCH /v1/organizations/:id
- POST /v1/oauth_clients, POST /v1/sso_connections, POST /v1/scim_tokens
- POST /v1/user_imports, POST /v1/user_exports (bulk jobs)

GET and naturally-idempotent PUT/DELETE do not need a key.

With an SDK

The Node SDK takes the key as a trailing argument on the methods that support it:

```
await atlas.users.create({ email_address: 'ada@example.com' }, 'signup-
ada-001');
await atlas.organizations.create({ name: 'Acme', slug: 'acme', created_by:
'user_2a' }, 'org-acme-001');
```

Other SDKs accept it as idempotency_key / idempotencyKey.

Choosing keys

Derive the key from the intent of the operation (the natural business key), not from a random per-attempt value — that way an automatic retry reuses the same key and collapses to one action.

Roles & permissions

Roles & permissions

Atlas gives you two authorization tools. Roles & permissions (RBAC) is the coarse, per-organization one — "admins can manage billing; members can't." For per-object relationship checks, see Fine-grained authorization.

> The server is the boundary. Client helpers decide what a user sees; only a check on the request decides what they can do.

Keys are the contract

Keys travel in the session token and get hardcoded into your checks, so their shape is fixed and they are immutable once created:

- A role key is org:<name> — e.g. org:admin, org:member. It names who someone is.
- A permission key is org:<resource>:<action> — e.g. org:billing:manage. It names what they may do.

The sys_ prefix is reserved for Atlas's built-ins (e.g. org:sys_memberships:manage). Every instance is seeded with two system roles: org:admin (holds every permission) and org:member (read-only). You can relabel but not delete them.

What reaches the session

When a user has an active organization, their role and the union of its permissions (direct and group-granted) are minted into the session JWT:

```
{
  "sub": "user_2a…",
  "org_id": "org_9f…",
  "org_role": "org:admin",
  "org_permissions": ["org:sys_memberships:manage", "org:billing:manage"]
}
```

Your code reads authorization straight off the token — no call back to Atlas on the hot path. A change takes effect within one token lifetime.

Endpoints

Method & path	Scope	Notes
GET /v1/roles	organizations:read	Roles + permissions + member counts
POST /v1/roles	organizations:write	Create a custom role (keys namespaced; sys_ reserved)
PATCH /v1/roles/:id	organizations:write	Relabel (the key is immutable)
PUT /v1/roles/:id/permissions	organizations:write	Replace a role's permission set (system roles are fixed)
DELETE /v1/roles/:id		

organizations:write

Delete an unused role; one members hold !’ ROLE_IN_USE (pass reassign_to to move members first)

GET /v1/permissions

organizations:read

Permissions available on this instance

POST /v1/permissions

organizations:write

Define a custom permission (org:invoices:manage)

PATCH /v1/permissions/:id

organizations:write

Update description/metadata

DELETE /v1/permissions/:id

organizations:write

Delete; removed from every role that held it

Manage from your backend

```
await atlas.roles.create({
  key: 'org:auditor',
  name: 'Auditor',
  permissions: ['org:logs:read', 'org:billing:read'],
});

// Retire a role people still hold — move its members first, atomically
await atlas.roles.delete('role_123', { reassignTo: 'role_member' });

await atlas.permissions.create({ key: 'org:reports:export', name: 'Export
reports' });
await atlas.permissions.delete('perm_456'); // 409 ROLE_IN_USE if a role still
grants it
```

Check on the server

The verified request carries has() (boolean) and protect() (throws ForbiddenError):

```
const auth = await atlas.authenticateRequest(req);
if (!auth.ok) return res.status(401).end();

if (auth.has({ permission: 'org:billing:manage' })) { /* … */ }
if (auth.has({ anyPermission: ['org:billing:manage', 'org:billing:read'] })) { /
* … */ }
auth.protect({ role: 'org:admin' }); // 403 if unmet
```

In Next.js, auth() exposes the same has() / protect() in server components and route handlers.

Gate UI on the client

A rendering aid, never a security boundary — always keep the matching server check.

```
import { Protect, useAuth } from '@atlas/react';

<Protect permission="org:billing:manage" fallback={<Locked />}>
  <BillingSettings />
</Protect>;

const { has } = useAuth();
if (has({ anyPermission: ['org:billing:manage', 'org:billing:read'] })) { /* …
*/ }
```

@atlas/react-native exposes the same useAuth().has(condition), and @atlas/js ships a framework-agnostic has(claims, condition). All of them evaluate the identical condition.

IdP-driven roles

Grant a directory group an org role and its members inherit the permissions — so your IdP drives role assignment through SCIM. See Organizations !’ group role grants.

OAuth clients & resource servers

OAuth clients & resource servers

Atlas can be an OAuth/OpenID provider — "Sign in with Atlas". You register the third-party (or first-party) apps that authenticate against your instance as OAuth clients, and the machine-to-machine APIs they call as resource servers. The protocol endpoints those clients hit are documented under OpenID Connect & OAuth2.

OAuth clients

```
{
  "object": "oauth_client",
  "id": "oac_…",
  "client_id": "atlas_client_…",
  "name": "Acme Web",
  "redirect_uris": ["https://acme.com/callback"],
  "allowed_scopes": ["openid", "profile", "email"],
  "grant_types": ["authorization_code", "refresh_token"],
  "token_endpoint_auth_method": "client_secret_basic",
  "is_public": false,
  "first_party": true,
  "created_at": 1757000000000,
  "updated_at": 1757600000000
}
```

Method & path	Scope	Notes
GET /v1/oauth_clients	oauth_clients:read	Secret never returned
POST /v1/oauth_clients	oauth_clients:write	Register a confidential or public (PKCE) client. A confidential client's secret is revealed once. idempotent
GET /v1/oauth_clients/:id	oauth_clients:read	Never the secret
PATCH /v1/oauth_clients/:id	oauth_clients:write	Update name/redirect_uris/scopes (secret immutable here). idempotent
POST /v1/oauth_clients/:id/rotate_secret	oauth_clients:write	Rotate; new secret revealed once. idempotent
DELETE /v1/oauth_clients/:id	oauth_clients:write	Delete; codes + grants cascade. idempotent

Client !’ resource-server grants (client credentials)

Method & path	Scope

GET /v1/oauth_clients/:clientId/grants

oauth_clients:read

POST /v1/oauth_clients/:clientId/grants

oauth_clients:write

DELETE /v1/oauth_clients/:clientId/grants/:grantId

oauth_clients:write

Resource servers (M2M APIs)

An API you protect with Atlas-issued access tokens: an audience identifier, its scopes, and a token TTL.

Method & path

Scope

Notes

GET /v1/resource_servers

resource_servers:read

List APIs + scopes

POST /v1/resource_servers

resource_servers:write

Register (audience, scopes, TTL)

GET /v1/resource_servers/:id

resource_servers:read

PATCH /v1/resource_servers/:id

resource_servers:write

Identifier (audience) is immutable

DELETE /v1/resource_servers/:id

resource_servers:write

Client grants cascade

Example

```
// Register a confidential client — capture the one-time secret
const client = await atlas.oauthClients.create({
  name: 'Acme Web',
  redirect_uris: ['https://acme.com/callback'],
  allowed_scopes: ['openid', 'profile', 'email'],
});
console.log(client.client_id, client.client_secret); // secret shown once

// Register an API and authorize the client for it (client_credentials)
const api = await atlas.resourceServers.create({ /* identifier, scopes, ttl
*/ } as any);
await atlas.oauthClients.grants.create(client.id, { resource_server_id: api.id,
scopes: ['reports:read'] });
```

Clients then run the standard flows at /oauth2/authorize and /oauth2/token; verify access tokens at /oauth2/introspect or against the JWKS. See OpenID Connect & OAuth2.

Client Organizations

Client Organizations

Frontend API — publishable key (x-publishable-key) plus the user session.

10 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/client/organizations		Create an organization, if the instance allows user-created ones.
GET /v1/client/organizations/:id		Read the active organization. Gated by the ACTIVE org role.
PATCH /v1/client/organizations/:id		Update the active organization. Requires org:sys_profile:manage.
GET /v1/client/organizations/:id/memberships		List members of the active organization as a display directory.
POST /v1/client/organizations/:id/invitations		Invite someone. Requires org:sys_memberships:manage.
DELETE /v1/client/organizations/:id/memberships/:userId		Remove a member. Removing the last admin returns LAST_ADMIN.
POST /v1/client/organization_invitations/:invitationId/accept		Accept an invitation addressed to one of your verified emails. Seats you as a member.
GET /v1/client/organizations/:id/membership_requests		List pending requests to join the active org. Requires org:sys_memberships:manage.
POST /v1/client/organizations/:id/membership_requests/:reqId/accept		Accept a join request, seating the user at the default member role (never admin).
POST /v1/client/organizations/:id/membership_requests/:reqId/reject		Reject a pending join request. No membership is created.

Instances

Instances

Public — no authentication required.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /instances/:instanceId/.well-known/jwks.json		Public JWKS for local token verification. Cacheable, stale-if-error.

Versioning

Versioning

Path versioning

The Atlas API is versioned in the path. Every endpoint lives under /v1/…:

```
https://api.atlas.dev/v1/users
https://<instance>.fapi.atlasauth.net/v1/client
```

Within a major version the contract only grows in backward-compatible ways:

- New endpoints, new optional request fields, and new response fields may be added at any time.
- Existing fields are not removed or repurposed, and error codes are never renamed silently.

Write your clients to ignore unknown fields so an additive change never breaks you.

Standards discovery

The OIDC/OAuth surface publishes its own capabilities so clients configure themselves rather than hard-coding:

- GET /.well-known/openid-configuration — the OpenID Provider metadata (host-based issuer, S256 PKCE, RS256 signing).
- GET /.well-known/jwks.json — the current public signing keys.

Signing keys rotate; always fetch JWKS (and cache with stale-if-error) rather than pinning a key.

SCIM & webhook contracts

- SCIM advertises its supported features and schemas at /scim/v2/ServiceProviderConfig, /scim/v2/ResourceTypes, and /scim/v2/Schemas.
- Webhooks carry a type per event and full resource objects (not diffs), so a consumer processing events out of order can still reconstruct state. The event catalog is published — see Webhook events.

Deprecations

When a field or behaviour is scheduled for removal in a future major version, it is documented here well ahead of time. Nothing in /v1 is removed without a new major path.

Lti Platforms

Lti Platforms

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/lti_platforms	lti:read		List the LTI 1.3 platform registrations allowed to launch this instance.
GET /v1/lti_platforms/:id	lti:read		Fetch one LTI 1.3 platform registration.
POST /v1/lti_platforms	lti:write		Register an LTI 1.3 platform (issuer, client_id, auth_login_url, jwks_uri, deployment_ids). https URLs required; a duplicate (issuer, client_id) is a 409.
PATCH /v1/lti_platforms/:id	lti:write		Update an LTI 1.3 platform registration (URLs, deployment ids, org link).
DELETE /v1/lti_platforms/:id	lti:write		Remove an LTI 1.3 platform registration.

Wallet Rpc Secret

Wallet Rpc Secret

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/wallet_rpc_secret		Store (or clear, with an empty url) a BYOK per-chain wallet RPC endpoint for token-gating. Keyed by EIP-155 chainId; write-only, encrypted; never returned. Owner or admin.

Saml

Saml

Frontend API — publishable key (x-publishable-key) plus the user session.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/saml/idp/sso		Atlas-as-SAML-IdP SSO endpoint (session-gated). IdP-initiated (?sp=) or SP-initiated (SAMLRequest); returns a signed SAML Response auto-POST form.
POST /v1/saml/idp/sso		Atlas-as-SAML-IdP SSO endpoint (session-gated), HTTP-POST binding — see GET /v1/saml/idp/sso.
PUT /v1/saml/idp/sso		Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.
PATCH /v1/saml/idp/sso		Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.
DELETE /v1/saml/idp/sso		Atlas-as-SAML-IdP SSO endpoint (app.all method fan-out; behaves as GET/POST) — see GET /v1/saml/idp/sso.

End-user API keys

End-user API keys

Distinct from your instance's sk_/pk_ keys: your tenant can mint API keys for its own users and organizations (so your customers can script your product), then ask Atlas to verify a presented key. Only a hash is stored — the ak_ secret is shown exactly once at mint.

```
{
  "object": "api_key",
  "id": "apikey_…",
  "subject_type": "user",
  "subject_id": "user_2a…",
  "name": "CI token",
  "prefix": "ak_live_ab12",
  "claims": { "scope": "read" },
  "last_used_at": null,
  "expires_at": null,
  "revoked_at": null,
  "created_at": 1757600000000
}
```

Endpoints

Method & path	Scope	Notes
GET /v1/api_keys	api_keys:read	List (optionally narrowed by subject_type/subject_id). Never the secret.
POST /v1/api_keys	api_keys:write	Mint for a user or organization. Returns the ak_ secret once.
POST /v1/api_keys/verify	api_keys:read	Verify a presented secret. Rate-limited.
GET /v1/api_keys/:id	api_keys:read	Prefix + metadata, never the secret
PATCH /v1/api_keys/:id	api_keys:write	Update name, claims, or expiry
DELETE /v1/api_keys/:id	api_keys:write	Revoke; stops verifying immediately (record kept)

Mint & verify

```
// Mint — capture the secret now; it is never shown again
const key = await atlas.apiKeys.create({
  subject_type: 'user',
  subject_id: 'user_2a',
  name: 'CI token',
  claims: { scope: 'read' },
```

```
});
console.log(key.secret); // ak_live_… (once)

// Later, when a caller presents a key:
const v = await atlas.apiKeys.verify(presentedSecret);
if (v.valid) {
  console.log(v.subject_type, v.subject_id, v.claims);
} else {
  // Every negative — unknown, malformed, revoked, expired — is the SAME
  // { valid: false }, so a caller learns nothing about which keys exist.
}
```

```
curl https://api.atlas.dev/v1/api_keys/verify \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"secret":"ak_live_xxx"}'
```

The verify verdict returns { valid: false } uniformly for every failure mode (unknown, malformed, revoked, expired, or subject-deleted) — the check is safe to expose behind your own API without leaking key existence.

Branding

Branding

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/branding	branding:read		Read the instance branding / hosted-page appearance and its resolved (renderer) form.
PATCH /v1/branding	branding:write		Change branding with a PARTIAL merge (unlike auth_config full-replace), validated the way the renderer resolves it — a value it would drop is refused, not silently ignored.
POST /v1/branding/preview	branding:read		Render a draft branding through the real sign-in hosted-page renderer (same sanitizers + CSP) so an agent sees what would ship before saving.

Security

Security

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/security		Read the instance kill switches and customer IP allowlist.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/security		Edit the instance kill switches and customer IP allowlist. Empty allowlist = unrestricted. Owner or admin.

Saml

Saml

Public — no authentication required.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/saml/idp/metadata		Atlas-as-SAML-IdP metadata (pk-resolved): entityId, signing certificate and SSO bindings for a downstream SP to import.

Invitations

Invitations

Instance-level invitations let you invite an email to the whole instance (as opposed to an organization invitation, which seats someone in one org). On matching-email sign-up, the invitation's public_metadata is copied onto the new user.

```
{
  "object": "invitation",
  "id": "inv_…",
  "email_address": "ada@example.com",
  "status": "pending",
  "public_metadata": { "referred_by": "campaign_x" },
  "expires_at": 1758204800000,
  "accepted_at": null,
  "created_at": 1757600000000
}
```

Endpoints

Method & path	Scope	Notes
GET /v1/invitations	invitations:read	List, cursor-paginated
POST /v1/invitations	invitations:write	Create; expires in 7 days. Metadata copied to the new user on sign-up.
POST /v1/invitations/:id/revoke	invitations:write	Revoke a pending invitation

Example

```
const invite = await atlas.invitations.create({
  email_address: 'ada@example.com',
  public_metadata: { referred_by: 'campaign_x' },
});

// Later
await atlas.invitations.revoke(invite.id);
```

```
curl https://api.atlas.dev/v1/invitations \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"email_address":"ada@example.com","public_metadata":
{"referred_by":"campaign_x"}}'
```

Waitlist

For sign-up gating rather than proactive invites, an instance can run a waitlist. List entries and decide them from the backend:

- GET /v1/waitlist_entries (waitlist:read) — entries + per-status counts, cursor-paginated.
- POST /v1/waitlist_entries/:id/decide (waitlist:write) — approve or deny an entry.

Oauth

Oauth

Public — no authentication required.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/oauth/callback/itchio		itch.io implicit-flow landing page. Serves a page that reads the fragment token in the browser and posts it to /v1/oauth/itchio.
POST /v1/oauth/itchio		itch.io token post-back: resolve the instance from state, exchange the implicit token for the profile, and complete the sign-in (MFA gate). Returns the continuation URL.
GET /v1/oauth/callback/lastfm/:state		Last.fm callback (state in the path). Exchanges the approval token for a session via an MD5-signed auth.getSession and redirects back with a one-time ticket.
GET /v1/oauth/bluesky/client-metadata.json		AT Protocol OAuth client metadata document. Its URL is the Bluesky client_id; the authorization server fetches it during PAR to learn redirect URIs and scopes.
GET /v1/oauth/callback/:provider		Provider redirect target. Redirects back with a one-time ticket, never a token.
POST /v1/oauth/callback/:provider		Provider form_post callback (Apple).

Waitlist

Waitlist

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/waitlist		The sign-up waitlist queue, with counts by status.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/waitlist/:entryId/decide		Approve or deny a waitlist entry. Approval is what lets that address sign up.

Client Environment

Client Environment

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/client/environment		The instance's enabled social providers (and native client ids) for a publishable-key client. Never exposes secrets.

Users

Users

Backend API — secret key (Authorization: Bearer sk_…).

24 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path

Scope

Idem

Explanation

GET /v1/users

users:read

List users, cursor-paginated.

GET /v1/users/:id

users:read

Fetch a single user. Never includes private_metadata.

PATCH /v1/users/:id

users:write

'

Update a user profile or metadata.

POST /v1/users/:id/ban

users:write

'

Ban a user and revoke every session.

DELETE /v1/users/:id

users:delete

'

Soft-delete a user; PII is scrubbed after 30 days.

POST /v1/users

users:write

'

Create a user from an email and optional password, name and metadata. Emits user.created.

POST /v1/users/:id/unban

users:write

'

Lift a ban. Sessions are not restored.

POST /v1/users/:id/lock

users:write

'

Lock an account so it cannot sign in, for an optional duration (default one year).

POST /v1/users/:id/unlock
users:write
'
Clear a lockout so the account can sign in again.
POST /v1/users/:id/reset_mfa
users:write
'
Remove every second factor and its recovery codes.
DELETE /v1/users/:id/mfa/:factorId
users:write
'
Remove one second factor. The last factor removed also turns MFA off.
PUT /v1/users/:id/metadata
users:write
'
Replace the metadata bags wholesale. Each bag present overwrites the stored one.
GET /v1/users/:id/sessions
users:read

List a user's active sessions.
POST /v1/users/:id/sessions/revoke
users:write
'
Revoke every session a user holds and bump sessions_version.
POST /v1/users/:id/email_addresses
users:write
'
Add an email address to a user. Starts unverified.
POST /v1/users/:id/email_addresses/:eid/verify
users:write
'
Mark an address verified on the backend authority, no code round-trip.
POST /v1/users/:id/email_addresses/:eid/primary
users:write
'
Make a verified address primary. Unverified addresses are refused.
GET /v1/users/:id/identities
users:read

List a user's linked identities: their Atlas email/password identity plus each linked external/OAuth provider account. No tokens.
POST /v1/users/:id/external_accounts/connect
users:write

Backend-initiated connect: start an OAuth flow that links a NEW provider identity to this user (the caller supplies the user id). Returns an authorization_url to redirect the browser to; the callback links the provider without minting a session. Refuses an identity already owned by another user (IDENTITY_ALREADY_LINKED).

POST /v1/users/:id/identities

users:write
'

Merge a secondary user into this primary: moves the secondary's provider accounts and verified identifiers over (skipping collisions), re-points org memberships without escalating role, revokes the secondary's sessions and retires it. Never clears the primary's MFA or ban.

DELETE /v1/users/:id/identities/:identityId

users:write
'

Unlink a provider identity, extracting it into a new standalone user (Auth0 semantics). The sole/primary identity cannot be unlinked.

GET /v1/users/:id/grants

grants:read

List the OAuth clients a user has authorized, with the granted scopes and grant time. Never returns tokens.

DELETE /v1/users/:id/grants

grants:write
'

Revoke every consent grant a user holds and cascade-revoke all associated tokens through the standard revocation path.

GET /v1/users/:id/oauth_access_tokens/:provider

oauth_tokens:read

Fetch a stored provider token, refreshing it single-flight if stale.

JWT templates

JWT templates

A JWT template defines a named set of custom claims to mint into a token — so a downstream service (Hasura, a database RLS policy, your own API) gets exactly the claims it expects. The frontend requests a token for a named template via getToken(template).

```
{
  "object": "jwt_template",
  "name": "hasura",
  "claims": {
    "https://hasura.io/jwt/claims": "{{org.id}}"
  }
}
```

Claim values are template strings resolved against the user/session/org at mint time. Reserved claims (sub, iss, aud, exp, iat, …) and private fields are refused — a template can add claims, never forge the ones Atlas controls.

Endpoints

Method & path	Scope	Notes
GET /v1/jwt_templates	instance:read	List templates + claim maps
GET /v1/jwt_templates/:id	instance:read	Fetch by name
POST /v1/jwt_templates	instance:write	Define claims (reserved/private refused)
PATCH /v1/jwt_templates/:id	instance:write	Replace the claim map
DELETE /v1/jwt_templates/:id	instance:write	Delete so it can no longer be minted

Example

```
await atlas.jwtTemplates.create({
  name: 'hasura',
  claims: {
    'https://hasura.io/jwt/claims': JSON.stringify({
      'x-hasura-default-role': '{{org.role}}',
      'x-hasura-user-id': '{{user.id}}',
    }),
  },
});
```

```
curl https://api.atlas.dev/v1/jwt_templates \
  -H "Authorization: Bearer sk_live_xxx" \
  -H "Content-Type: application/json" \
  -d '{"name":"hasura","claims":{"x-hasura-user-id":"{{user.id}}"}}'
```

Minting a template token from the frontend

The client mints a short-lived JWT for the active session using a named template:

`POST /v1/client/sessions/:id/tokens/:template  (FAPI)`

In the SDKs this is getToken({ template: 'hasura' }). The default getToken() (no template) returns the standard session JWT. See Sessions & tokens.

Restrictions

Restrictions

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/restrictions		The allow and block lists plus whether allow-list-only sign-up is on.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/restrictions		Add an email, @domain or E.164 phone to the allow or block list. Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/restrictions/:id		Remove an allow/block list entry. Owner or admin.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/restrictions		Toggle allow-list-only sign-up, the switch that gives the allow list effect.

Sso

Sso

Public — no authentication required.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/sso/callback		Enterprise SSO IdP redirect target. Verifies the OIDC id_token, JIT-provisions org membership, and redirects back with a one-time ticket.
POST /v1/sso/saml/acs		SAML Assertion Consumer Service. Verifies the signed assertion (audience, timestamps, XSW-hardened), JIT-provisions org membership, and redirects back with a one-time ticket.
GET /v1/sso/saml/:connectionId/metadata		SAML SP metadata XML for a connection: the entityID and ACS URL the operator registers with their IdP.

Grants

Grants

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
DELETE /v1/grants/:id	grants:write	'	Revoke one consent grant and cascade-revoke its access/refresh grants through the standard revocation path, so the app stops working on its next request.

Oauth

Oauth

Frontend API — publishable key (x-publishable-key) plus the user session.

7 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/oauth/telegram		Telegram Login Widget sign-in: verify the widget payload HMAC against the bot token and complete the sign-in, honouring the MFA gate.
POST /v1/oauth/siwe/nonce		Sign-in with Ethereum (EIP-4361): mint a single-use nonce bound to a new attempt.
POST /v1/oauth/siwe/verify		Sign-in with Ethereum (EIP-4361): verify a signed message (domain, chain, nonce, validity, EIP-191/EIP-1271) and complete the sign-in, honouring the MFA gate.
POST /v1/oauth/siws/nonce		Sign-in with Solana (SIWS): mint a single-use nonce bound to a new attempt.
POST /v1/oauth/siws/verify		Sign-in with Solana (SIWS): verify a signed message (domain, network, nonce, validity, ed25519 signature) and complete the sign-in, honouring the MFA gate.
POST /v1/oauth/siwf/nonce		Sign-in with Farcaster (SIWF): mint a single-use nonce bound to a new attempt.
POST /v1/oauth/siwf/verify		Sign-in with Farcaster (SIWF): verify a signed SIWE message (domain, chain, nonce, validity, signature), prove on-chain that the custody address owns the claimed FID (Optimism ID Registry), and complete the sign-in, honouring the MFA gate.

Lti

Lti

Public — no authentication required.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /lti/login		LTI 1.3 OIDC login initiation. Looks up the platform by (iss, client_id) and 302s to its authorization endpoint with a state+nonce (response_type=id_token, response_mode=form_post, prompt=none).
POST /lti/login		LTI 1.3 OIDC login initiation (form-post variant of GET /lti/login).
POST /lti/launch		LTI 1.3 launch callback. Verifies the platform id_token (JWKS signature, iss, aud=client_id, single-use nonce), enforces the deployment_id allowlist and LtiResourceLinkRequest/1.3.0, JITs the user, and lands them on target_link_uri with SameSite=None session cookies for the LMS iframe.

User Imports

User Imports

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/user_imports	jobs:write	'	Bulk-create users from an array, deduping by verified email, through the same path sign-up uses. Plaintext passwords are Argon2id-hashed; an Argon2id hash is imported verbatim; other hash formats are rejected per row. Per-row failures are recorded, never fatal.

Network Acls

Network Acls

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/network_acls		The per-instance IP allow/deny rules, priority-ordered.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/network_acls		Add an IP/CIDR allow or deny rule. Owner or admin.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/network_acls/:id		Edit a network ACL rule (action, CIDR, priority, enabled). Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/network_acls/:id		Remove a network ACL rule. Owner or admin.

Client Tickets

Client Tickets

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/client/tickets/exchange		Exchange a one-time ticket for session cookies.

Stripe

Stripe

Public — no authentication required.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /stripe/webhook		Stripe webhook. Verifies the signature over the raw body, then moves the account plan free!”pro on checkout.session.completed and customer.subscription.updated/deleted. The only writer of plan; a bad signature is a 400 that changes nothing.

Discourse

Discourse

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /discourse/sso		DiscourseConnect provider endpoint. Verifies the forum HMAC (constant-time), authenticates via the hosted session, and signs the identity payload back to the return_sso_url whose origin is pinned to the connection.

Sms Templates

Sms Templates

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path

Idem

Explanation

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates

The per-instance SMS templates — built-in default alongside any override.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates

Customise an SMS template body (upsert). Must keep {{code}}. Owner or admin.

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates/:name

One SMS template — its built-in default and stored override, if any.

PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates/:name

Edit an SMS template body/enabled. Owner or admin.

DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates/:name

Reset an SMS template to the built-in default (drops the override). Owner or admin.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sms_templates/:name/preview

Render an SMS body with a fixed sample code — never a live OTP.

User Exports

User Exports

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/user_exports	jobs:write	'	Produce a job whose result is a present()-serialised array of the instance's users. Never contains a password hash, recovery code or provider token.

Managed Waf

Managed Waf

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/managed_waf		Managed AWS WAF config + provisioning state. Secrets are never returned, only has_* markers.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/managed_waf/status		The lean managed WAF provisioning status (enabled, status, WebACL, last error).
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/managed_waf		Set managed WAF config + write-only AWS credentials. Enabling points captcha at awswaf. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/managed_waf/provision		Provision the WebACL (idempotent create/update + associate). Fail-safe. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/managed_waf/deprovision		Disassociate and delete the WebACL. Fail-safe. Owner or admin.

Client

Client

Frontend API — publishable key (x-publishable-key) plus the user session.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/client		Boot the client. Returns session: null for a signed-out visitor, never 401.

Jobs

Jobs

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/jobs	jobs:read		List bulk import/export jobs, newest first (cursor pagination).
GET /v1/jobs/:id	jobs:read		Fetch a job: its status, counters and (for an export) result.
GET /v1/jobs/:id/errors	jobs:read		List a job's per-row errors ({ index, email, code, message }).

Client Sessions

Client Sessions

Frontend API — publishable key (x-publishable-key) plus the user session.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/client/sessions		List the signed-in user's own active sessions/devices, flagging the current one.
POST /v1/client/sessions/revoke_all		Sign out of every other device. Spares the current session; bumps sessions_version.
POST /v1/client/sessions/:id/touch		Switch the active organization. Membership is resolved server-side.
POST /v1/client/sessions/:id/tokens		Rotate the refresh token and mint a session JWT. Reads the refresh cookie only.
POST /v1/client/sessions/:id/revoke		Revoke one session and clear its cookies.
POST /v1/client/sessions/:id/tokens/:template		Mint a short-lived JWT for the active session using a named JWT template (getToken).

Scim Provisioning Targets

Scim Provisioning Targets

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

8 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets		Outbound SCIM provisioning targets Atlas pushes users to; bearer shown as has_* only.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets		Create an outbound SCIM target. The downstream bearer is write-only. Owner or admin.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id		One outbound SCIM target with its sync health, bearer redacted.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id		Edit an outbound SCIM target (name, base URL, bearer, mapping, pause/resume). Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id		Remove an outbound SCIM target. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id/test		Verify connectivity/auth to the downstream (ServiceProviderConfig). Writes no data. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id/sync_user		Force one user to sync to the target now (backfill/re-push). Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_provisioning_targets/:id/sync_group		Force one organization (group) to sync now; skips unprovisioned members. Owner or admin.

Data Subject Requests

Data Subject Requests

Backend API — secret key (Authorization: Bearer sk_…).

4 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/data_subject_requests	data_subject_requests:read		List GDPR/DSAR export and erasure requests, filterable by type/status/user (cursor pagination).
GET /v1/data_subject_requests/:id	data_subject_requests:read		Fetch a data-subject request, including an export's produced package.
POST /v1/data_subject_requests/:id/fulfill	data_subject_requests:write '		Fulfil a request now: build the export package, or run the erasure through the same soft-delete + PII-purge path a user-delete uses, ahead of the grace window. Terminal — a fulfilled erasure is irreversible.
POST /v1/data_subject_requests/:id/reject	data_subject_requests:write '		Reject a request with a recorded reason. Terminal — cannot be re-actioned.

Jwt Templates

Jwt Templates

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/jwt_templates		The named JWT templates (custom-claim maps) stored in the instance auth config.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/jwt_templates		Create a JWT template (name + claims map). Owner or admin.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/jwt_templates/:id		One JWT template with its claims map.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/jwt_templates/:id		Edit a JWT template (claims, lifetime). Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/jwt_templates/:id		Remove a JWT template. Owner or admin.

Client Billing

Client Billing

Frontend API — publishable key (x-publishable-key) plus the user session.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/client/billing/plans		Public list of the instance's active end-user billing plans for display (no Stripe price ids or secrets).
POST /v1/client/billing/checkout		Create a Stripe Checkout Session to subscribe the signed-in user (or an org they administer) to a plan. Returns {url}; never changes subscription state.
GET /v1/client/billing/subscription		Read the signed-in user's subscription, or an org's (org_id) when the caller is a member — status plus the active plan and its features.
POST /v1/client/billing/portal		Create a Stripe Billing Portal session for the subject's customer (self, or an org the caller administers) to manage or cancel. Returns {url}.

Webhook Endpoints

Webhook Endpoints

Backend API — secret key (Authorization: Bearer sk_…).

7 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/webhook_endpoints	webhooks:write		Register a webhook endpoint. The secret is revealed once.
GET /v1/webhook_endpoints	webhooks:read		List webhook endpoints. Never re-reveals secrets.
GET /v1/webhook_endpoints/:id/deliveries	webhooks:read		Delivery log for an endpoint.
POST /v1/webhook_endpoints/:id/rotate_secret	webhooks:write		Rotate the endpoint signing secret. The new secret is revealed exactly once; deliveries signed with the old one are rejected immediately.
POST /v1/webhook_endpoints/:id/deliveries/:deliveryId/redeliver	webhooks:write		Manually redeliver a past event as a fresh attempt. 409 if the event has passed its 90-day retention.
POST /v1/webhook_endpoints/:id/test	webhooks:write		Send a synthetic webhook.test event to this endpoint only. Records a real delivery; never fanned out to other endpoints.
DELETE /v1/webhook_endpoints/:id	webhooks:write		Disable a webhook endpoint.

Localizations

Localizations

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/localizations		Per-language overrides for emails, SMS and hosted-page copy, plus the resource catalog.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/localizations		Create or replace a localization override (validated like the base template).
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/localizations/:id		Edit a localization override’s content or enabled flag.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/localizations/:id		Remove a localization override; the base template renders again.

Sign In Tokens

Sign In Tokens

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/sign_in_tokens	sign_in_tokens:write		Mint a one-time sign-in token for impersonation. Always audited.

User Imports

User Imports

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/user_imports		Bulk-import users from an array; small batches run inline, large ones queue. Owner or admin.

Actor Tokens

Actor Tokens

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/actor_tokens	impersonation:write		Mint a one-time actor token; the redeemed session carries the RFC 8693 act claim naming the impersonator. Always audited.
POST /v1/actor_tokens/:id/revoke	impersonation:write		Revoke an unredeemed actor token so it can no longer be exchanged for a session.

User Exports

User Exports

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/user_exports		Export every user as a JSON array (no secrets, by construction). Owner or admin.

Invitations

Invitations

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/invitations	invitations:read		List instance-level invitations, cursor-paginated.
POST /v1/invitations	invitations:write		Create an instance-level invitation. On matching-email sign-up its public_metadata is copied onto the new user. Expires in 7 days.
POST /v1/invitations/:id/revoke	invitations:write		Revoke a pending instance invitation so it can no longer be accepted.

Organizations

Organizations

Backend API — secret key (Authorization: Bearer sk_…).

23 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/organizations	organizations:read		List organizations, cursor-paginated.
POST /v1/organizations	organizations:write		Create an organization and seat its creator as admin.
GET /v1/organizations/:id	organizations:read		Fetch an organization. Never includes private_metadata.
GET /v1/organizations/:id/memberships	organizations:read		List members of an organization with their roles.
PATCH /v1/organizations/:id/memberships/:userId	organizations:write		Change a member's role. Demoting the last admin returns LAST_ADMIN.
DELETE /v1/organizations/:id/memberships/:userId	organizations:write		Remove a member. Removing the last admin returns LAST_ADMIN.
POST /v1/organizations/:id/invitations	organizations:write		Invite an email address to an organization. Expires in 7 days.
POST /v1/organizations/:id/invitations/:invitationId/revoke	organizations:write		Revoke a pending invitation so the address can be re-invited.
PATCH /v1/organizations/:id/policy			

organizations:write

Update the org security policy (requireMfa, ssoRequired, sessionIdleOverrideMs).
PUT /v1/organizations/:orgId/groups/:groupId/roles/:roleId
organizations:write

Grant an org role to a directory group; members inherit its permissions (augments only).
DELETE /v1/organizations/:orgId/groups/:groupId/roles/:roleId
organizations:write

Revoke a role from a directory group. Members lose the inherited permissions.
PATCH /v1/organizations/:id
organizations:write
'
Update an organization name, slug, image, seat cap or metadata. Emits organization.updated.
DELETE /v1/organizations/:id
organizations:write
'
Delete an organization; its memberships and invitations cascade with it.
PUT /v1/organizations/:id/parent
organizations:write

Set or clear (null) an organization’s parent in the B2B2B hierarchy; refuses a cycle.
GET /v1/organizations/:id/hierarchy
organizations:read

The organization’s ancestor chain and direct children in the B2B2B hierarchy.
GET /v1/organizations/:id/entitlements
organizations:read

The organization’s resolved entitlement plan and feature keys.
PUT /v1/organizations/:id/metadata
organizations:write
'
Replace the organization public and private metadata bags wholesale.
POST /v1/organizations/:id/memberships
organizations:write
'
Add an existing instance user to an organization with a role. The seat cap is enforced.
GET /v1/organizations/:id/invitations
organizations:read

List an organization's invitations with their roles and status.

GET /v1/organizations/:id/domains
organizations:read

List an organization's email domains and their verification state.
POST /v1/organizations/:id/domains
organizations:write
'
Claim an email domain for auto-join. Returns the DNS TXT record to publish.
POST /v1/organizations/:id/domains/:domainId/verify
organizations:write

Check the DNS TXT record now and mark the domain verified.
DELETE /v1/organizations/:id/domains/:domainId
organizations:write

Remove a claimed email domain.

Jobs

Jobs

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/jobs		List import/export jobs with live progress counters.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/jobs/:id		Poll one import/export job (an export carries its result payload).
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/jobs/:id/errors		The per-row failures recorded for an import job.

Sessions

Sessions

Backend API — secret key (Authorization: Bearer sk_…).

4 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/sessions	sessions:write		Mint a session for a user in one call — returns jwt + refresh_token. For agents/testing; bypasses the sign-in gate. Optional actor.sub for an impersonation session.
GET /v1/sessions	sessions:read		List a user's active sessions. Requires a user_id query parameter.
GET /v1/sessions/:id	sessions:read		Fetch a single session. Another instance session id is 404, never 403.
POST /v1/sessions/:id/revoke	sessions:write		Revoke one session, signing that device out.

Data Subject Requests

Data Subject Requests

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/data_subject_requests		The GDPR/DSAR queue — export and erasure requests, filterable by type/status/user.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/data_subject_requests/:id		One data-subject request, including an export package if it was fulfilled.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/data_subject_requests/:id/fulfill		Fulfil now — build the export or run the erasure ahead of schedule. Terminal. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/data_subject_requests/:id/reject		Reject a request with a recorded reason. Terminal. Owner or admin.

Resource Servers

Resource Servers

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/resource_servers	resource_servers:read		List resource servers (machine-to-machine APIs) and their scopes.
POST /v1/resource_servers	resource_servers:write		Register a resource server (API): its audience identifier, scopes and token TTL.
GET /v1/resource_servers/:id	resource_servers:read		Fetch one resource server. Another instance id is 404, never 403.
PATCH /v1/resource_servers/:id	resource_servers:write		Update a resource server. The identifier (audience) is immutable.
DELETE /v1/resource_servers/:id	resource_servers:write		Delete a resource server; its client grants cascade with it.

Log Streams

Log Streams

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

6 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams		Log-stream sinks (HTTP/Datadog/Splunk); secrets shown as has_* markers only.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams		Create a log-stream sink. Destination secrets are write-only. Owner or admin.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams/:id		One log-stream sink with its delivery health, secrets redacted.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams/:id		Edit a log-stream sink (name, destination, filter, pause/resume). Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams/:id		Remove a log-stream sink. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/log_streams/:id/test		Send one synthetic event to the sink and report its status. Owner or admin.

Oauth Providers

Oauth Providers

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

7 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers		The whole provider catalog with per-instance state and the redirect URI to register.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider		Store a client id and secret. Write-only: no route ever returns the secret.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider		Forget the credentials. Linked accounts are preserved (E17).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider/enabled		Enable or disable a configured provider without deleting anyone identity.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider/scope		Scope a provider to sign-in, sign-up, or both. Enforced at the resolver.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider/shared_keys		Opt in or out of Atlas shared development credentials. Refused for production.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_providers/:provider/test		Probe the provider token endpoint to check the client id and secret only.

Oauth Clients

Oauth Clients

Backend API — secret key (Authorization: Bearer sk_…).

9 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path

Scope

Idem

Explanation

GET /v1/oauth_clients/:clientId/grants

oauth_clients:read

List the resource-server grants (client_credentials authorizations) of a client.

POST /v1/oauth_clients/:clientId/grants

oauth_clients:write

Authorize a confidential client for a resource server and its scopes.

DELETE /v1/oauth_clients/:clientId/grants/:grantId

oauth_clients:write

Revoke a client's grant for a resource server.

GET /v1/oauth_clients

oauth_clients:read

List "Sign in with Atlas" OAuth clients. The client secret is never returned.

POST /v1/oauth_clients

oauth_clients:write

'

Register an OAuth client (confidential or public PKCE). A confidential client's secret is revealed exactly once.

GET /v1/oauth_clients/:id

oauth_clients:read

Fetch one OAuth client. Another instance id is 404, never 403. Never the secret.

PATCH /v1/oauth_clients/:id

oauth_clients:write

'

Update an OAuth client's registration (name, redirect_uris, scopes). The secret is immutable here.

POST /v1/oauth_clients/:id/rotate_secret

oauth_clients:write

'

Rotate a confidential client's secret; the new one is revealed exactly once.
DELETE /v1/oauth_clients/:id
oauth_clients:write
'
Delete an OAuth client; its codes and grants cascade with it.

Sso Connections

Sso Connections

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

12 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections		List enterprise SSO connections. The OIDC client secret is never returned.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections		Create an enterprise SSO connection. The client secret is write-only.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId		Read one SSO connection. Reports has_secret only, never the secret.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId		Update an SSO connection. Omitting the secret leaves the stored one untouched.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId		Delete an SSO connection and its domain routing.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/atlas		Whether "Sign in with Atlas" is enabled on this instance, and whether the deployment offers it.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/atlas		Enable "Sign in with Atlas" — provisions an Atlas-ID OP client + preset OIDC connection. Owner/admin. Idempotent.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/atlas		Disable "Sign in with Atlas" — removes the preset connection. Owner/admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId/test		Validate discovery: issuer reachable and issuer-match (RFC 8414). Not the client.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId/test_login		

Begin an interactive OIDC test login (opened in a popup); the callback shows the returned claims and provisions nothing.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId/import_saml_metadata

Parse an IdP SAML EntityDescriptor and populate the entity id, SSO URL and signing certificate.

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_connections/:connectionId/saml_metadata

SP metadata XML for a SAML connection, plus the ACS URL and SP entity id the operator registers with their IdP.

Sso Connections

Sso Connections

Backend API — secret key (Authorization: Bearer sk_…).

6 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/sso_connections	sso_connections:read		List enterprise SSO connections (OIDC / SAML / DiscourseConnect). Never the client secret.
POST /v1/sso_connections	sso_connections:write	'	Create an enterprise SSO connection. The IdP client secret is write-only.
GET /v1/sso_connections/:id	sso_connections:read		Fetch one SSO connection. Another instance id is 404, never 403. Reports has_secret, never the secret.
PATCH /v1/sso_connections/:id	sso_connections:write	'	Update an SSO connection. A secret is only re-encrypted when a new one is supplied.
DELETE /v1/sso_connections/:id	sso_connections:write	'	Delete an SSO connection and its domain routing.
GET /v1/sso_connections/:id/saml_metadata	sso_connections:read		The SP EntityDescriptor XML to hand a SAML IdP (entityID + ACS URL). Public; carries no secret.

Sso Onboarding Profiles

Sso Onboarding Profiles

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path

Scope

Idem

Explanation

GET /v1/sso_onboarding_profiles

sso_connections:read

List self-service SSO onboarding profiles (the templates tickets are issued from).

POST /v1/sso_onboarding_profiles

sso_connections:write

'

Create a self-service SSO onboarding profile: allowed connection types and an optional bound organization.

GET /v1/sso_onboarding_profiles/:id

sso_connections:read

Fetch one onboarding profile. Another instance id is 404, never 403.

DELETE /v1/sso_onboarding_profiles/:id

sso_connections:write

Delete an onboarding profile; its issued tickets cascade away with it.

POST /v1/sso_onboarding_profiles/:id/tickets

sso_connections:write

'

Issue a scoped, one-time, expiring onboarding ticket for one organization. Returns the token and hosted URL exactly once.

Sso Onboarding Profiles

Sso Onboarding Profiles

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_profiles		List self-service SSO onboarding profiles.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_profiles		Create a self-service SSO onboarding profile (owner/admin).
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_profiles/:profileId		Delete an SSO onboarding profile.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_profiles/:profileId/tickets		List the onboarding tickets issued for a profile.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_profiles/:profileId/tickets		Issue a one-time SSO onboarding ticket + hosted setup URL (revealed once).

Sso Onboarding Tickets

Sso Onboarding Tickets

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/sso_onboarding_tickets/:id/revoke	sso_connections:write		Revoke a pending onboarding ticket so it can no longer configure a connection.

Sso Onboarding Tickets

Sso Onboarding Tickets

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/sso_onboarding_tickets/:ticketId/revoke		Revoke an outstanding SSO onboarding ticket.

Scim Tokens

Scim Tokens

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/scim_tokens	scim:read		List SCIM provisioning tokens. Reports the recognisable prefix only, never the token.
POST /v1/scim_tokens	scim:write	'	Mint a SCIM provisioning token for an organization. The token is revealed exactly once.
POST /v1/scim_tokens/:id/revoke	scim:write	'	Revoke a SCIM token; provisioning with it fails immediately. The record is kept for the audit log.

Resource Servers

Resource Servers

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/resource_servers		List resource servers (APIs that accept machine tokens).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/resource_servers		Create a resource server (audience + scopes) for client-credentials tokens.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/resource_servers/:id		Read one resource server.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/resource_servers/:id		Update a resource server (identifier is immutable).
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/resource_servers/:id		Delete a resource server; its client grants cascade.

Actions

Actions

Backend API — secret key (Authorization: Bearer sk_…).

8 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/actions	actions:read		List Actions (tenant code that runs at auth-pipeline hooks). Secret values are never returned, only their names.
POST /v1/actions	actions:write	'	Create an Action for a trigger (post_login, pre_user_registration, post_user_registration). Secrets are write-only.
GET /v1/actions/:id	actions:read		Read one Action. Reports secret names only, never their values.
PATCH /v1/actions/:id	actions:write		Update an Action's name, code, enabled flag, or secrets. A secret set to "" is deleted; one left out is preserved.
DELETE /v1/actions/:id	actions:write		Delete an Action. Its trigger bindings cascade away, so it stops running everywhere.
POST /v1/actions/:id/test	actions:write		Dry-run an Action against a sample event in the sandbox and return its claims/deny/logs. Touches no real data.
GET /v1/actions/bindings/:trigger	actions:read		Read the ordered list of Action ids bound to a trigger.
PUT /v1/actions/bindings/:trigger			

actions:write

Replace the ordered list of Actions bound to a trigger — the single source of truth for what runs, and in what order.

Oauth Clients

Oauth Clients

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

9 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId/grants		List a client’s machine-to-machine grants.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId/grants		Grant a confidential client client-credentials access to a resource server.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId/grants/:grantId		Revoke a client’s machine-to-machine grant.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients		List registered "Sign in with Atlas" clients. The client secret is never re-revealed.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients		Register a "Sign in with Atlas" client. redirect_uris are validated as absolute https; the secret is revealed once (none for a public PKCE client).
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId		Edit an OAuth client's name, redirect_uris, scopes, logo or auth method. Never changes the client_id or the secret.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId/rotate_secret		Rotate a confidential client secret. The old secret stops working immediately.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId		Delete an OAuth client. Its authorization codes and grants cascade away with it.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/oauth_clients/:clientId/consents		The end-user consents (authorized users + scopes) granted to this OAuth client.

Branding

Branding

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/branding		Branding as stored, plus what the renderers will actually use.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/branding		Update logo and colours for the hosted pages and emails.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/branding/preview		Render an unsaved branding draft through the real hosted-page renderer; persists nothing.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/branding/logo		Upload the instance logo (raw image bytes, validated by magic number) to object storage and point branding.logoUrl at it. 503 when image storage is not configured.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/branding/image		Upload a branding image (logo) as raw bytes; returns its hosted asset URL. Owner or admin.

Log Streams

Log Streams

Backend API — secret key (Authorization: Bearer sk_…).

6 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/log_streams	log_streams:read		List log streams (event forwarding to Datadog, Splunk or a generic HTTP sink). Secret values are never returned, only has_* markers.
POST /v1/log_streams	log_streams:write	'	Create a log stream. destination secrets (api_key / token / auth headers) are write-only; the cursor starts at the current event so only new events forward.
GET /v1/log_streams/:id	log_streams:read		Read one log stream, including its forwarding status, cursor and last error. Reports secret markers only, never their values.
PATCH /v1/log_streams/:id	log_streams:write		Update a log stream's name, destination, event_filter, enabled flag, or status (active/paused). A secret left out of destination is preserved.
DELETE /v1/log_streams/:id	log_streams:write		Delete a log stream. Forwarding stops immediately; no further events are sent to the sink.
POST /v1/log_streams/:id/test	log_streams:write		Send one synthetic event to the sink with the stream's real credentials and return the sink's status. Touches no real event and never moves the cursor.

Email Templates

Email Templates

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/email_templates		Every template with its override and the variable schema it is validated against.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/email_templates/:template		Store a template override. Validated by the same code the mailer applies.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/email_templates/:template		Drop the override and go back to the built-in template.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/email_templates/:template/preview		Compile an unsaved draft to HTML with sample values. Sends nothing.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/email_templates/:template/test_send		Send a test to the operator own verified address. The address cannot be chosen.

Scim Provisioning Targets

Scim Provisioning Targets

Backend API — secret key (Authorization: Bearer sk_…).

8 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/scim_provisioning_targets	scim_provisioning:read		List outbound SCIM provisioning targets (downstream SCIM 2.0 endpoints Atlas pushes users to). The downstream bearer is never returned, only has_bearer_token.
POST /v1/scim_provisioning_targets	scim_provisioning:write	'	Create an outbound SCIM provisioning target. base_url must be https; the downstream bearer_token is write-only; the cursor starts at the current event so only new users forward (use sync_user to backfill).
GET /v1/scim_provisioning_targets/:id	scim_provisioning:read		Read one outbound SCIM provisioning target, including its sync status, cursor and last error. Reports has_bearer_token only, never the bearer value.
PATCH /v1/scim_provisioning_targets/:id	scim_provisioning:write		Update a target's name, base_url (https), attribute_mapping, deprovision_action, enabled flag, or status (active/paused). A bearer_token left out is preserved.
DELETE /v1/scim_provisioning_targets/:id	scim_provisioning:write		Delete an outbound SCIM provisioning target. Pushing stops immediately; no further users are sent downstream.
POST /v1/scim_provisioning_targets/:id/test	scim_provisioning:write		Verify connectivity and auth to the downstream (GET ServiceProviderConfig) with the target's real bearer, and return the result. Writes no data.
POST /v1/scim_provisioning_targets/:id/sync_user	scim_provisioning:write		

Force a single user's outbound sync now (backfill / re-push). A first sync POSTs and stores the remote id; a re-sync PUTs the same id. A downstream failure is reported, never thrown.

POST /v1/scim_provisioning_targets/:id/sync_group

scim_provisioning:write

Force a single group's (organization's) outbound sync now to the downstream /Groups. Members are the org's memberships, but only users already provisioned to this target (unprovisioned members are skipped). A first sync POSTs and stores the remote id; a re-sync PUTs the same id. A downstream failure is reported, never thrown.

Roles

Roles

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/roles		Roles with membership counts and permissions, plus which roles grant each permission.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/roles		Create a custom role. Keys are namespaced org: and may not use sys_.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/roles/:roleId		Relabel a role. The key is immutable because it appears in issued tokens.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/roles/:roleId/permissions		Replace a custom role permission set. System role sets are fixed.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/roles/:roleId		Delete a role, optionally reassigning its members first with ?reassign_to.

Billing

Billing

Backend API — secret key (Authorization: Bearer sk_…).

6 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/billing/plans	billing:read		List the plans the tenant defines for their app's users (end-user billing). Admin view — includes the Stripe price id.
POST /v1/billing/plans	billing:write '		Create an end-user billing plan (name, slug, Stripe price, features, user/org audience). The slug becomes the pla session claim and must be unique per instance.
GET /v1/billing/plans/:id	billing:read		Read one end-user billing plan, including its Stripe price id and granted features.
PATCH /v1/billing/plans/:id	billing:write		Update an end-user billing plan's name, slug, price, interval, features, audience or active flag.
DELETE /v1/billing/plans/:id	billing:write		Delete an end-user billing plan. Existing subscriptions retain their plan_id (the FK restricts deletion of a plan still referenced).
GET /v1/billing/subscriptions	billing:read		List end-user subscriptions, optionally filtered by subject_type (user/org) and subject_id. Read-only — status is moved solely by the Stripe webhook.

Permissions

Permissions

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/permissions		Define a custom permission key.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/permissions/:permissionId		Update a custom permission's name/description (the key is immutable). System permissions are refused.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/permissions/:permissionId		Delete a permission no role grants. Refused with the granting roles named.

Domains

Domains

Backend API — secret key (Authorization: Bearer sk_…).

4 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/domains	domains:read		List custom domains with DNS/ACME/cookie-domain status and setup instructions.
POST /v1/domains	domains:write	'	Claim a custom domain (fapi or accounts). Returns the CNAME instructions to follow.
POST /v1/domains/:id/verify	domains:write	'	Check the domain's CNAME now; a verified fapi host is written onto the instance.
DELETE /v1/domains/:id	domains:write	'	Remove a custom domain. Refused for a live production fapi host, which would sign everyone out.

Waitlist Entries

Waitlist Entries

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/waitlist_entries	waitlist:read		List sign-up waitlist entries with per-status counts. Cursor-paginated.
POST /v1/waitlist_entries/:id/decide	waitlist:write	'	Approve or deny a waitlist entry, letting someone in or keeping them out.

Webhooks

Webhooks

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

9 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks		Endpoints with their failure streak, plus the event catalog and retry schedule.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks		Create an endpoint. The signing secret is returned once here.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId		Change the URL, the subscribed events, or re-enable a disabled endpoint.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId		Delete an endpoint and its delivery history.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId/reveal		Reveal the signing secret. Owner or admin, and audited before it is returned.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId/rotate		Mint a new signing secret. Old-secret verifiers start failing immediately.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId/deliveries		The delivery log, with the payload sent and the response received.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId/deliveries/:deliveryId/redeliver		Queue a fresh attempt for an event. A new row; the failed one is kept.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/webhooks/:endpointId/test		Send a synthetic webhook.test event to one endpoint. Records a real delivery; never fanned out.

Api Keys

Api Keys

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/api_keys		Keys with last-used and a stale flag, plus the scope catalog.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/api_keys/:keyId/rotate		Mint a replacement with the same scopes. The old key keeps working until revoked.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/api_keys/:keyId/revoke		Revoke a key. The row is kept so the audit log can still name it.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/api_keys		Mint a secret key. Returned exactly once.

Network Acls

Network Acls

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/network_acls	network_acls:read		List network ACLs — per-instance IP allow/deny rules that gate access to the instance, in priority order.
POST /v1/network_acls	network_acls:write		Create a network ACL rule (allow|deny, a CIDR or single address, a priority). A malformed CIDR is refused. First match in priority order decides; an allow-list denies every unnamed IP.
GET /v1/network_acls/:id	network_acls:read		Read one network ACL rule.
PATCH /v1/network_acls/:id	network_acls:write		Update a network ACL rule's action, cidr, description, priority or enabled flag.
DELETE /v1/network_acls/:id	network_acls:write		Delete a network ACL rule. With no rules left, enforcement falls back to the ip_allowlist.

Radius Clients

Radius Clients

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/radius_clients	radius_clients:read		List the RADIUS NAS clients — the VPN/WiFi/firewall devices allowed to authenticate this instance's users over RADIUS. The per-NAS shared secret is never returned, only whether one is set.
POST /v1/radius_clients	radius_clients:write		Register a RADIUS NAS client (name, NAS-Identifier, optional source-IP bind, write-only shared secret). The NAS-Identifier is unique across the deployment; a duplicate is a 409.
GET /v1/radius_clients/:id	radius_clients:read		Read one RADIUS NAS client. The shared secret is never included.
PATCH /v1/radius_clients/:id	radius_clients:write		Update a RADIUS NAS client's name, NAS-Identifier, source-IP bind, enabled flag or shared secret. An omitted or empty shared_secret keeps the stored one.
DELETE /v1/radius_clients/:id	radius_clients:write		Delete a RADIUS NAS client. That NAS can no longer authenticate against the listener.

Events

Events

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/events		The activity feed, filterable by type and date. ?format=csv exports it.

Fga

Fga

Backend API — secret key (Authorization: Bearer sk_…).

13 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/fga/stores	fga:read		List fine-grained authorization (ReBAC / Zanzibar) stores in the instance. Each store namespaces its own authorization models and relationship tuples.
POST /v1/fga/stores	fga:write	'	Create a fine-grained authorization store — an isolated namespace for authorization models and relationship tuples.
GET /v1/fga/stores/:id	fga:read		Read one FGA store. A store from another instance is a 404.
DELETE /v1/fga/stores/:id	fga:write		Delete an FGA store, cascading its authorization models and relationship tuples.
GET /v1/fga/stores/:id/authorization-models	fga:read		List the immutable authorization-model versions of a store, newest first.
POST /v1/fga/stores/:id/authorization-models	fga:write	'	Write a new immutable authorization-model version (OpenFGA-shaped type_definitions with this/computedUserset/tupleToUserset and union/intersection/difference). Unknown relation or type references are rejected.
GET /v1/fga/stores/:id/authorization-models/:mid	fga:read		Read one authorization-model version, including its full model definition.
POST /v1/fga/stores/:id/write			

fga:write
'

Transactionally write and/or delete relationship tuples in a store. Writes are idempotent (a re-written tuple is a no-op); the whole batch is all-or-nothing.

POST /v1/fga/stores/:id/read

fga:read

Query stored relationship tuples in a store, optionally filtered by user, relation and/or object.

POST /v1/fga/stores/:id/check

fga:read

Check whether a user has a relation on an object, resolving the authorization model (direct tuples, computed usersets, tuple-to-userset hops, set operators) with cycle + depth safety. Supports contextual tuples and a pinned authorization_model_id.

POST /v1/fga/stores/:id/list-objects

fga:read

List the object ids of a type that a user has a relation on. Exact (check-filtered over the complete candidate set); cost is linear in the objects of that type.

POST /v1/fga/stores/:id/expand

fga:read

Expand the userset tree for an (object, relation) — who would hold the relation, mirroring the model rewrite. Not user-specific.

POST /v1/fga/stores/:id/batch-check

fga:read

Check up to 100 (user, relation, object) tuples in one request.

Audit Logs

Audit Logs

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/audit_logs		The audit log with actor, action and target filters. ?format=csv exports it.

Domains

Domains

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/domains		Custom domains with CNAME instructions, TLS status and the cookie-domain check.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/domains		Claim a hostname for the FAPI or hosted-pages role. Subdomains only.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/domains/:domainId/verify		Check the CNAME now and record what DNS actually returned.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/domains/:domainId		Remove a domain. Refused for a live production FAPI host.

Rate Limit Policy

Rate Limit Policy

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/rate_limit_policy	rate_limit:read		Read the per-instance rate-limit policy (sign-in/up create per IP, and BAPI per key) and its bounds. Defaults equal the built-in fixed limits.
PATCH /v1/rate_limit_policy	rate_limit:write		Tune the per-instance rate-limit budgets. Values are bounded so protection cannot be disabled or set to self-DoS; a lower value tightens the limiter.

Managed Waf

Managed Waf

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/managed_waf	managed_waf:read		Read the managed AWS WAF configuration and provisioning state. AWS credential secrets are never returned, only has_* markers.
GET /v1/managed_waf/status	managed_waf:read		Read just the managed AWS WAF provisioning state: status, WebACL id/arn, last error and last-provisioned time.
PUT /v1/managed_waf	managed_waf:write		Configure the managed AWS WAF: scope/region, edge target, token domains, gated paths, and the write-only BYO AWS credentials. Enabling points auth_config.captcha at awswaf.
POST /v1/managed_waf/provision	managed_waf:write		Provision now: idempotently create-or-update the AWS WAF WebACL with the CAPTCHA rule and associate it with the edge (ALB or CloudFront). Fail-safe: AWS errors are recorded, not thrown into auth.
POST /v1/managed_waf/deprovision	managed_waf:write		Deprovision: disassociate the WebACL from the edge and delete it, then clear the stored state. Fail-safe like provision.

Staff

Staff

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/staff/accounts		staff: search customer accounts. Requires the staff instance, an MFA-verified session and an allowlisted IP.
GET /v1/staff/accounts/:accountId		staff: one account with its applications and instances. Read-only; user counts, never user lists.
GET /v1/staff/platform_switches		staff: read the global kill switches. Staff instance, MFA and IP allowlist.
PATCH /v1/staff/platform_switches		staff: flip the global kill switches (read-only mode, disable all sign-ups). Audited.

Attack Protection

Attack Protection

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/attack_protection	attack_protection:read		Read the instance's attack-protection settings: brute-force lockout, breached-password checking, IP allowlist and captcha status.
PATCH /v1/attack_protection	attack_protection:write		Toggle attack protections — the brute-force lockout and the breached-password (HIBP) check.

Usage

Usage

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/usage		Daily sign-in, sign-up, email and active-user series plus trailing 30-day MAU. MAU is a distinct count, never a sum of the daily active-user rows.

Risk Based Mfa

Risk Based Mfa

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/risk_based_mfa	risk_based_mfa:read		Read the instance's adaptive (risk-based) MFA policy: enabled, step-up threshold, no-factor handling, and the signal weights/toggles.
PATCH /v1/risk_based_mfa	risk_based_mfa:write		Tune adaptive MFA: enable it, set the step-up threshold and the high-risk-no-factor policy, and adjust the per-signal weights/toggles. Off by default; it can only add friction, never weaken the gate.

Root

Root

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts		Accounts the signed-in dashboard user belongs to.
POST /v1/dashboard/accounts		Create an account with the caller as its first owner, atomically.

Bot Signals

Bot Signals

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/bot_signals	bot_signals:read		Export the anonymised anti-bot signal lake (device + behavioural + network features, risk verdict, heuristic score), newest first with a before epoch-ms cursor. Training data for an in-house bot-vs-human model. No raw IPs/PII.

Onboarding

Onboarding

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/onboarding/bootstrap		First-run: provision a whole workspace in one call — account + owner + an application with development and production instances. Names optional (Skip sends none, gets defaults).

Bot Labels

Bot Labels

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/bot_labels	bot_signals:read		Export the supervised training labels (bot / human / suspect) with their subject and source.
POST /v1/bot_labels	bot_signals:write		Add a supervised training label to a subject (user / device / ip_hash / attempt) — an analyst confirming a bot or a human. Same label set that banning-as-bot writes.

Applications

Applications

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

9 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/applications		Applications and their instances. Secret keys are never listed.
POST /v1/dashboard/accounts/:accountId/applications		Create an application with both a development and production instance.
PATCH /v1/dashboard/accounts/:accountId/applications/:appId		Rename an application the account owns. Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/applications/:appId		Delete an application. Refused for the last remaining application in the account.
GET /v1/dashboard/accounts/:accountId/applications/:appId/team		The application team: members and pending invitations for this application.
POST /v1/dashboard/accounts/:accountId/applications/:appId/team/invitations		Invite a collaborator to help manage this application; returns a one-time invite link. Owner or admin.
POST /v1/dashboard/accounts/:accountId/applications/:appId/team/invitations/:id/revoke		Revoke a pending application-team invitation. Owner or admin.
PATCH /v1/dashboard/accounts/:accountId/applications/:appId/team/members/:userId		Change an application-team member’s role. Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/applications/:appId/team/members/:userId		Remove a collaborator from the application team. Owner or admin.

Members

Members

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/members		List account members and their roles.
POST /v1/dashboard/accounts/:accountId/members		Add a member. Requires owner or admin.
PATCH /v1/dashboard/accounts/:accountId/members/:userId		Change a member's role. Demoting the last owner returns LAST_ADMIN.
DELETE /v1/dashboard/accounts/:accountId/members/:userId		Remove a member. Removing the last owner returns LAST_ADMIN.

Allowlist Identifiers

Allowlist Identifiers

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/allowlist_identifiers	restrictions:read		List allowlist identifiers (emails, @domains, phones) permitted in allow-list-only mode.
POST /v1/allowlist_identifiers	restrictions:write		Add an allowlist identifier: an exact email, an @domain, or an E.164 phone.
DELETE /v1/allowlist_identifiers/:id	restrictions:write		Remove an allowlist identifier. Another instance id is 404, never 403.

Blocklist Identifiers

Blocklist Identifiers

Backend API — secret key (Authorization: Bearer sk_…).

3 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/blocklist_identifiers	restrictions:read		List blocklist identifiers that cannot sign up or sign in.
POST /v1/blocklist_identifiers	restrictions:write		Block an identifier (email, @domain, or E.164 phone) from signing up and signing in.
DELETE /v1/blocklist_identifiers/:id	restrictions:write		Remove a blocklist identifier. Another instance id is 404, never 403.

Actions

Actions

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

7 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/actions		List the instance Actions (tenant code that runs at auth-pipeline triggers). Secret values never returned.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/actions		Create an Action (name, trigger, code, secrets). Owner/admin only; secrets are write-only.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/actions/:id		Update an Action’s name, code, enabled state or secrets. Owner/admin only.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/actions/:id		Delete an Action; its trigger bindings cascade so it stops running everywhere. Owner/admin only.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/actions/:id/test		Dry-run an Action against a sample event in the sandbox; touches no real data. Owner/admin only.
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/actions/bindings/:trigger		The ordered pipeline of Actions bound to a trigger (the flow).
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/actions/bindings/:trigger		Set the ordered pipeline of Actions for a trigger (reorder/add/remove). Owner/admin only.

Fga

Fga

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

8 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path

Idem

Explanation

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores

List fine-grained (ReBAC) authorization stores.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores

Create an FGA authorization store. Owner/admin only.

DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id

Delete an FGA store and its models + tuples. Owner/admin only.

GET /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id/models

List a store’s authorization models; the latest includes its full definition.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id/models

Write a new (immutable, versioned) authorization model; validated before persist. Owner/admin only.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id/read

Read relationship tuples in a store, optionally filtered by user/relation/object.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id/write

Write and/or delete relationship tuples in a store. Owner/admin only.

POST /v1/dashboard/accounts/:accountId/instances/:instanceId/fga/stores/:id/check

Run a fine-grained access check (user, relation, object) against the store’s latest model.

Roles

Roles

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/roles	organizations:read		List roles with their permissions and how many members hold each.
POST /v1/roles	organizations:write		Create a custom role. Keys are namespaced and sys_ is reserved.
PATCH /v1/roles/:id	organizations:write		Relabel a role. The key is immutable once published.
PUT /v1/roles/:id/permissions	organizations:write		Replace a role permission set. System roles are fixed.
DELETE /v1/roles/:id	organizations:write		Delete an unused custom role. A role members hold returns ROLE_IN_USE.

Permissions

Permissions

Backend API — secret key (Authorization: Bearer sk_…).

4 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/permissions	organizations:read		List permissions available to roles on this instance.
POST /v1/permissions	organizations:write		Define a custom permission. Keys look like org:invoices:manage.
PATCH /v1/permissions/:id	organizations:write		Update a permission’s description or metadata.
DELETE /v1/permissions/:id	organizations:write		Delete a permission; it is removed from every role that held it.

Scim Tokens

Scim Tokens

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_tokens		List SCIM provisioning tokens with last-used. Never re-reveals a token.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_tokens		Mint a SCIM token for an organization directory. Returned exactly once.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/scim_tokens/:tokenId/revoke		Revoke a SCIM token. The row is kept so the audit log can still name it.

Billing

Billing

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/billing		Read the account plan, whether a subscription is live, and whether Stripe billing is configured on this deployment.
POST /v1/dashboard/accounts/:accountId/billing/checkout		Start a Stripe Checkout Session to upgrade to pro and return its URL. Owner or admin; 503 when Stripe is not configured. Never changes the plan itself.
POST /v1/dashboard/accounts/:accountId/billing/portal		Open a Stripe Billing Portal session to manage or cancel the subscription and return its URL. Owner or admin; 503 when Stripe or the customer is absent.
POST /v1/dashboard/accounts/:accountId/billing/subscriptions/:subscriptionId/allocate		Allocate (move) a platform subscription to one application. Each subscription powers a single app.

Mcp

Mcp

Backend API — secret key (Authorization: Bearer sk_…).

2 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/mcp	instance:read		MCP server descriptor and the tools this key may call.
POST /v1/mcp	(per-operation)		MCP JSON-RPC endpoint: initialize, tools/list, tools/call, ping.

Messaging Providers

Messaging Providers

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

5 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/messaging_providers		Read the BYOK email and SMS providers with the plan and whether managed sending is available. Secrets are never returned, only which are set.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/messaging_providers/email		Store the email provider (resend, ses or smtp). Secrets are write-only; a blank or omitted secret keeps the stored one.
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/messaging_providers/sms		Store the SMS provider (twilio). The auth_token is write-only; a blank or omitted value keeps the stored one.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/messaging_providers/:channel		Forget a channel's BYOK provider. The instance falls back to managed-if-paid.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/messaging_providers/:channel/test		Send a clearly-marked test message to a destination to verify the configured email or SMS provider delivers, without triggering a real auth flow. Owner or admin; rate-limited; 422 when nothing is configured.

Email Templates

Email Templates

Backend API — secret key (Authorization: Bearer sk_…).

4 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/email_templates	instance:read		List email templates with their defaults and any instance override.
POST /v1/email_templates/:name/preview	instance:read		Render an email template (an unsaved draft if supplied, else the stored one) with SAMPLE variables — never a real code — so an agent can see the result before saving.
PUT /v1/email_templates/:name	instance:write		Override a template. Dropping a required placeholder is refused.
DELETE /v1/email_templates/:name	instance:write		Revert a template to the Atlas built-in.

Lti Platforms

Lti Platforms

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/lti_platforms		List the registered LTI 1.3 platforms (issuer, client_id, jwks_uri, deployment_ids).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/lti_platforms		Register an LTI 1.3 platform. (issuer, client_id) is unique per instance.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/lti_platforms/:platformId		Update an LTI platform’s endpoints, deployment_ids, or organization binding.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/lti_platforms/:platformId		Delete an LTI platform registration. Future launches from it are refused.

Api Keys

Api Keys

Backend API — secret key (Authorization: Bearer sk_…).

6 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/api_keys	api_keys:read		List end-user API keys the tenant minted for its users/organizations. Never the secret.
POST /v1/api_keys	api_keys:write		Mint an end-user API key for a user or organization. Returns the ak_ secret once.
POST /v1/api_keys/verify	api_keys:read		Verify a presented ak_ secret. Returns the subject + claims, or valid:false. Rate-limited.
GET /v1/api_keys/:id	api_keys:read		Read one end-user API key by id (prefix + metadata, never the secret).
PATCH /v1/api_keys/:id	api_keys:write		Update an end-user API key: name, claims, or expiry.
DELETE /v1/api_keys/:id	api_keys:write		Revoke an end-user API key. It stops verifying immediately; the record is kept.

Localizations

Localizations

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/localizations	localizations:read		List per-language localization overrides (email/SMS templates, hosted-page copy).
POST /v1/localizations	localizations:write		Create or replace a localization override for one resource + locale. Validated on save.
GET /v1/localizations/:id	localizations:read		Read one localization override by id.
PATCH /v1/localizations/:id	localizations:write		Update a localization override content or enabled flag. Re-validated on save.
DELETE /v1/localizations/:id	localizations:write		Delete a localization override, reverting that locale to the base template/copy.

Radius Clients

Radius Clients

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/radius_clients		List the RADIUS NAS clients (name, NAS-Identifier, optional source-IP bind, enabled). The shared secret is never returned.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/radius_clients		Register a RADIUS NAS client with a write-only shared secret. The NAS-Identifier is unique across the deployment.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/radius_clients/:clientId		Update a NAS client. An omitted or empty shared_secret keeps the stored one.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/radius_clients/:clientId		Delete a RADIUS NAS client. That NAS can no longer authenticate against the listener.

Sms Templates

Sms Templates

Backend API — secret key (Authorization: Bearer sk_…).

6 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/sms_templates	sms_templates:read		List SMS templates with their built-in defaults and any instance override.
POST /v1/sms_templates	sms_templates:write		Set an SMS body template. A body dropping the code placeholder is refused.
GET /v1/sms_templates/:name	sms_templates:read		Read one SMS template: its built-in default and the instance override.
PATCH /v1/sms_templates/:name	sms_templates:write		Update an SMS template body or its enabled flag. Re-validated on save.
DELETE /v1/sms_templates/:name	sms_templates:write		Revert an SMS template to the Atlas built-in default body.
POST /v1/sms_templates/:name/preview	sms_templates:read		Render an SMS body with a sample code — never a live OTP — for review.

App Invitations

App Invitations

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
POST /v1/dashboard/app_invitations/accept		Accept an application-team invitation with its token (the invitee’s own session).

Ldap Connections

Ldap Connections

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/ldap_connections		Which configured LDAP connections have a stored bind password (state only, never the secret).
PUT /v1/dashboard/accounts/:accountId/instances/:instanceId/ldap_connections/:connectionId/bind_password		Set a connection’s LDAP bind password (write-only, encrypted). Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/ldap_connections/:connectionId/bind_password		Clear a connection’s stored LDAP bind password. Owner or admin.

Jwt Templates

Jwt Templates

Backend API — secret key (Authorization: Bearer sk_…).

5 routes. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/jwt_templates	instance:write		Define custom JWT claims. Reserved claims and private fields are refused.
GET /v1/jwt_templates	instance:read		List the defined JWT templates and their claim maps.
GET /v1/jwt_templates/:id	instance:read		Fetch one JWT template by name.
PATCH /v1/jwt_templates/:id	instance:write		Replace a JWT template claim map. Reserved claims and private fields are refused.
DELETE /v1/jwt_templates/:id	instance:write		Delete a JWT template so it can no longer be minted.

Billing Plans

Billing Plans

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

4 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/billing_plans		List the end-user subscription plans defined for this instance.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/billing_plans		Create an end-user billing plan (slug + Stripe price + features). Owner or admin.
PATCH /v1/dashboard/accounts/:accountId/instances/:instanceId/billing_plans/:id		Update an end-user billing plan. Owner or admin.
DELETE /v1/dashboard/accounts/:accountId/instances/:instanceId/billing_plans/:id		Delete an end-user billing plan. Owner or admin.

Tokens

Tokens

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
POST /v1/tokens/verify	sessions:read		Authoritatively verify a session token, including revocation.

Audit Logs

Audit Logs

Backend API — secret key (Authorization: Bearer sk_…).

1 route. Each row is one endpoint with its explanation, required scope, and whether it honours the Idempotency-Key header.

Method & path	Scope	Idem	Explanation
GET /v1/audit_logs	audit:read		Query the audit log, filterable and cursor-paginated.

Bot Signals

Bot Signals

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

1 route. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/bot_signals		Recent anti-bot signal rows (risk verdict, network origin, geo, captcha), newest first.

Invitations

Invitations

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

3 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/accounts/:accountId/instances/:instanceId/invitations		List instance invitations (pending, accepted, revoked, expired).
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/invitations		Create an instance invitation for an email address. Owner or admin.
POST /v1/dashboard/accounts/:accountId/instances/:instanceId/invitations/:id/revoke		Revoke a pending instance invitation. Owner or admin.

Me

Me

Dashboard API — authenticated by an Atlas console (platform-instance) session, not a key.

2 routes. Each row is one endpoint with its explanation and whether it honours the Idempotency-Key header.

Method & path	Idem	Explanation
GET /v1/dashboard/me/notification_prefs		The signed-in admin’s email-notification preferences (security, product, billing, team).
PUT /v1/dashboard/me/notification_prefs		Update the signed-in admin’s email-notification preferences. Partial patch per category.