Corporate TV Agora

Corporate TV Agora

Flussonic

Table of contents

1. Products 3
2. Agora 4
2.1 Architecture 6
2.2 Admin 17
2.3 Manage 35
2.4 Security 47

1. Products

2. Agora

2.0.1 Agora

Agora is a corporate TV system that lets you:

- run your own TV channels;
- prepare studio-camera feeds for broadcast;
- distribute TV signals over the corporate network and, when needed, the internet;
- deliver video to employees in a protected way.

Key capabilities

- Ingest live feeds and file-based content
- Process and prepare video streams for distribution
- Organize internal broadcast channels
- Record broadcasts and build a VOD catalog
- Manage users and audiences
- Integrate with corporate authentication (OIDC / OAuth2 / LDAP / AD / SSO)
- Deliver content to Smart TV, STBs, web clients, and mobile devices
- Administer via a web UI
- Use APIs to integrate with internal IT systems

Documentation map

This documentation is organized as follows:

- Architecture overview
- Architecture patterns
  - Cluster Ingest
  - Twincast
  - Double Publish
  - Standby Push
- Content management
  - Content management overview
  - Stream management
  - Source capture
  - Stream processing
  - Outgoing publications
- Administrator guide
  - I/O devices
  - Streamers
  - CDN
  - Monitoring
- Security guide
  - Users and roles
  - OIDC/OAuth2
  - Security log
  - User sessions
  - Network interactions

2.1 Architecture

2.1.1 Agora architecture

Agora is a modular on-prem corporate TV platform. Architecture splits the control plane, video processing plane, and content delivery plane so administration, ingest, transcoding, archive, and playback can scale independently.

Architectural principles

Design principles:

- all major components can stay inside the customer network;
- control services are separated from video traffic servers;
- roles such as `Ingress server`, `origin`, `vod transcoder`, `edge` servers, and `storage` may be collocated or split by deployment scale;
- critical streams support redundancy for sources, servers, and delivery paths;
- the system can start small and grow distributed without changing the management model.

Logical layout

A typical deployment includes:

- `Controller` for centralized management and monitoring;
- `Ingress server` in the DMZ for external video sources;
- `origin` receiving from `Ingress server`, studio feeds, HDMI/SDI, and other internal sources, with transcoding when needed;
- `vod transcoder` to prepare files and archive clips for publishing;
- `edge` servers forming CDN over the corporate network;
- `storage` for long-term recordings and material from archive workflows;
- `player` for end-user playback.

Simplified interaction:

```
flowchart LR
    central["controller"] --> config["Config / Audit / Monitoring"]
    central --> ingress["Ingress server"]
    central --> origin["origin"]
    central --> vod["vod transcoder"]
    central --> edge["edge servers"]

    external["External sources"] --> ingress
    ingress --> origin
    studio["Studio / HDMI / SDI / internal IP sources"] --> origin

    origin --> edge
    origin --> storage["storage system"]

    vod --> storage
    vod --> edge

    edge --> player["player"]

    player -.- telemetry -.-> central
```

Delivery model

Agora delivery is hybrid: some components are software-only, others ship as appliances (hardware-integrated bundles).

Software-only components usually deploy on the customer's standard server fleet:

- `controller`;
- `origin`;
- `vod transcoder`;
- `edge` servers;
- `storage`.

Appliances target components that need pre-validated hardware, compatibility, and guaranteed I/O behavior.

Specialized capture cards such as `HDMI` or `SDI` typically ship as appliances.

Control plane

The control plane owns configuration, security, and observability.

CONTROLLER

The `Controller` is the central management component. Administrators and operators use it to:

- configure system nodes;
- register and monitor server roles;
- create and change streams;
- monitor status, metrics, and events;
- manage recording, delivery, and protection policies;
- review audit and health-check results.

Architecturally the `controller`:

- stores the logical model of streams and servers;
- exposes a unified management interface;
- manages accounts, roles, and sessions;
- aggregates server and stream status;
- acts as the hub for monitoring and configuration control.

DATABASE

A separate database holds:

- stream configuration;
- server and input device descriptions;
- accounts and roles;
- sessions;
- administrative and security event logs.

It is not for raw video. Video archives, DVR, and VOD files live in dedicated storage tiers.

Media processing plane

INGRESS SERVER

The `Ingress server` sits in the DMZ and ingests video from external sources as the controlled entry point for inbound video.

It can accept:

- IP streams over `SRT`, `RTMP`, and other supported protocols;
- feeds from external sites and contractors;
- backup external sources.

Main jobs:

- receive external streams in the DMZ;
- separate the external perimeter from internal media servers;
- forward video to `origin`;
- provide a monitored, controlled entry point.

Critical channels may use redundant ingress with several independent paths.

ORIGIN

`origin` prepares the stream for delivery to staff and TVs.

It:

- receives from the `Ingress server`;
- takes studio feeds;
- takes HDMI/SDI and other local sources;
- transcodes when required;
- builds the master stream for downstream delivery;
- sends content to `edge` servers;
- can serve clients directly at small scale;
- sends archive and derived assets to `storage`.

In Agora, `origin` combines master aggregation of internal and external sources with transcoding duties.

ARCHIVE AND ASSET PREPARATION

Recording provides:

- live DVR with a limited retention window;
- saving broadcasts after they end;
- moving assets into `storage` and publishing recordings as VOD.

Temporary DVR and short-term archive usually sit near `origin`; long-term storage uses the `storage` tier.

VOD TRANSCODER

`vod transcoder` prepares files, broadcast recordings, and archive clips for publishing. It:

- receives sources from `origin` or `storage`;
- transcodes to the required output profiles;
- normalizes containers, codecs, and bitrates;
- writes prepared assets to `storage` for `edge` servers to consume.

Delivery plane

EDGE SERVERS

`edge` servers deliver video to viewers on the corporate network. They may sit:

- in the user segment of a site;
- in a branch;
- close to large viewer groups to offload traffic.

Goals:

- offload `origin`;
- reduce cross-segment traffic;
- localize client connections;
- improve resilience when some nodes fail.

Depending on requirements, `edge` servers may act as:

- relay servers into branches;
- multicast or `UDP MPEG-TS` injection points on the LAN.

PLAYER

`player` is end-user playback. We provide:

- a web player suitable for embedding in a corporate portal;
- an Android app for STBs, including appliance STBs we ship;
- a web app usable on TVs.

Content may be delivered:

- from `origin`;
- via `edge` servers.

Content storage

Storage splits into:

`storage` is for long-term video — individual files and clips cut from saved archives.

Operational storage covers:

- temporary live DVR;
- short-lived archives on `origin`;
- intermediate transcoding files.

Long-term storage covers:

- broadcast recordings;
- uploaded videos;
- prepared VOD assets.

That split lets you choose:

- fast disks for live workloads;
- capacity-optimized tiers for archives;
- different backup and retention policies.

Typical data flows

LIVE BROADCAST

Typical live path:

1. External source feeds the `Ingress server`, or an internal source goes straight to `origin`.
2. `origin` ingests, transcodes if needed, and builds the master representation.
3. `origin` sends content to `edge` servers.
4. `edge` servers serve viewers.
5. In parallel the feed may be written to DVR/archive.

PUBLISHING A RECORDING

Typical VOD path:

1. Save the broadcast to a temporary archive.
2. Move the recording into `storage`.
3. Build output profiles and metadata in `vod transcoder`.
4. Publish VOD through `origin` and `edge` servers.

Deployment patterns

COMPACT DEPLOYMENT

Small sites may combine roles on one or two servers:

- `Controller`;
- `origin`;
- `vod transcoder`;
- `storage`.

Suited to pilots and limited channels/viewers.

DISTRIBUTED DEPLOYMENT

Corporate production usually separates roles:

- dedicated `Controller`;
- `Ingress server` in the DMZ;
- one or more `origin` nodes;
- dedicated `vod transcoder` for file prep;
- `edge` servers in user or branch segments;
- dedicated `storage`.

With network isolation, external sources only reach the `Ingress server`, while viewers use `edge` servers.

High availability

Availability needs differ by role:

- the management server is critical for administration but not always for ongoing playback;
- `Ingress server` and `origin` matter for external-source live;
- `edge` servers matter for mass delivery;
- `vod transcoder` matters for VOD prep and archive workflows;
- `storage` matters for VOD and archives, not always for every live path.

Main video resilience mechanisms:

- Cluster Ingest
- Twincast
- Double Publish
- Standby Push
- redundant ingest via primary/backup sources;
- paired `Ingress server` and `origin` for critical paths;
- multiple `edge` servers with balancing and fallback;
- split between operational and archive storage;
- backup of configuration, accounts, and audit data;
- monitoring of server availability, time sync, configuration, and stream quality.

Security and network boundaries

Corporate deployments usually segment:

- administrative;
- internal media;
- storage;
- user or DMZ delivery segments.

That enables:

- blocking direct user access to internal master nodes;
- separating web control from video traffic;
- placing external ingest on a dedicated `Ingress server` in the DMZ;
- centralizing audit and change control;
- integrating with corporate security tooling.

External system boundaries

Architecturally Agora interacts with:

- external video sources;
- corporate directories and access roles;
- monitoring and SIEM systems;
- external media players and IPTV devices;
- corporate portals and `player`;
- external publishing and production tools.

Summary

Agora is a layered platform where `controller`, `Ingress server`, `origin`, `vod transcoder`, `edge` servers, `storage`, and `player` play distinct roles. You can deploy a compact single-site system or a resilient distributed estate with centralized management, protected external ingest, internal master processing, separate VOD preparation, and scalable delivery on the corporate network.

2.1.2 Twincast

`Twincast` is a resilience pattern where the same live feed is ingested simultaneously over two independent paths: primary and backup. The platform monitors both delivery paths and can use the backup when the primary degrades.

How Twincast works

In a typical setup one source feeds two independent ingest paths; the system compares their state and uses the working stream.

```
flowchart LR
    source["Source"] --> primary["Primary ingest"]
    source --> backup["Backup ingest"]
    primary --> origin["origin"]
    backup --> origin
```

Under normal conditions the `primary` path is preferred. If it fails, the system can switch to `backup` while keeping the broadcast going.

An important limitation: `primary` and `backup` ingest independently, so frame timestamps and video `GOP` structure are not synchronized between them. Failover between `primary` and `backup` is therefore less seamless than in modes where both paths share aligned stream structure.

When to use it

`Twincast` is especially useful when:

- you run studio or business-critical live channels;
- you ingest an important external source;
- even a few seconds of outage is undesirable;
- you need continuous visibility into primary and backup inputs.

What this scenario provides

Benefits of `Twincast`:

- redundancy for the whole ingest path, not only the server;
- ongoing monitoring of `primary` and `backup`;
- fast switch to the working path;
- lower risk of interruption from partial hardware or signal failure.

Limitations and requirements

For a correct `Twincast` deployment, consider:

- two independent ingest paths;
- clear identification of primary vs. backup;
- monitoring of quality and status on both paths;
- no timestamp/`GOP` alignment between `primary` and `backup`;
- rules for automatic or manual switching between them.

2.1.3 Double Publish

`Double Publish` means the original signal source sends the stream to two streamers at once. The streamers are configured so downstream relays can obtain the same video from either node.

How Double Publish works

The source publishes to two streamers. The second streamer at the same time:

- receives the stream directly from the source;
- pulls the stream from the first streamer;
- treats the first streamer's feed as higher priority.

That keeps identical timestamps, video structure, and stream representation on both streamers for downstream publication.

```
flowchart LR
    source["Original source"] -- publish --> streamerA["Streamer A"]
    source -- publish --> streamerB["Streamer B"]
    streamerA --> streamerB
    streamerA --> relayA["relays / downstream"]
    streamerB --> relayB["relays / downstream"]
```

While the first streamer's feed is available, it is the preferred source for the second. If that feed drops, the second streamer immediately uses its direct ingest from the original source and continues publishing without waiting for manual action.

When to use it

Use `Double Publish` when:

- the source can send to two streamers at once;
- you need the same stream on two publishing streamers;
- downstream relays must be able to take video from either streamer;
- fast failover without timestamp/structure drift matters.

This mode fits software encoders well. For `OBS`, `vMix`, and similar tools, `Double Publish` is natural because the source can publish to multiple targets out of the box.

What this scenario provides

Benefits of `Double Publish`:

- resilient publishing from two streamers;
- aligned video structure and timestamps across streamers;
- immediate cutover to the second streamer's direct path if the first fails;
- downstream relays can attach to either streamer with the same content.

Limitations and requirements

When designing `Double Publish`, ensure:

- the source supports simultaneous publish to two streamers;
- the second streamer can hold direct ingest and pull from the first at the same time;
- priority from the first streamer is configured correctly;
- switching from pulled to direct ingest is monitored;
- downstream relays can connect to either streamer.

2.1.4 Standby Push

`Standby Push` is a resilience pattern where a standby push path is prepared next to the main publish direction. While the primary path is healthy, the backup stays idle and only activates on primary failure or severe degradation.

How Standby Push works

In this scenario an `origin` or other publishing node has two egress directions:

- primary push;
- standby push.

The common combination is `Standby Push` with `Twincast`: primary and backup ingest already exist, and `Standby Push` lets you move onward publication between them quickly.

Typical diagram:

```
flowchart LR
    primaryStreamer["Primary streamer"] --> multicast["multicast"]
    multicast --> backupStreamer["Backup streamer"]
    primaryStreamer --> primary["Primary push"]
    primary --> edge["edge servers"]
    backupStreamer --> standby["Standby push"]
    standby --> reserve["backup edge path"]
```

While the primary push is healthy, the backup path is not actively delivering (or stays passive). On primary failure the standby path activates.

Handover time is often under a second. `Twincast` + `Standby Push` is the fastest switching option but needs a LAN with reliably low latency.

For `Standby Push`, the backup streamer must receive multicast from the primary streamer.

When to use it

`Standby Push` fits when:

- you need a backup publication path without constant double load;
- resilience is required on outgoing push;
- you want less traffic than `Double Publish`;
- the backup must be pre-provisioned but used only on failure;
- you pair with `Twincast` for fastest switchover;
- legacy gear should remain backup without reconfiguration.

What this scenario provides

Benefits of `Standby Push`:

- backup publish path without always sending on both;
- more bandwidth-efficient than `Double Publish`;
- fast activation of the backup path;
- stronger delivery resilience if the primary egress fails;
- legacy equipment can serve as backup without changing its role.

Limitations and requirements

For a correct `Standby Push` design, define:

- criteria to move from primary to backup push;
- standby activation time;
- rules to return to primary after recovery;
- monitoring of primary and backup publish paths;
- a low-latency LAN between primary and backup;
- multicast delivery from the primary streamer to the backup.

2.1.5 Cluster Ingest

`Cluster Ingest` is a resilient capture mechanism that preserves a single-active-ingest guarantee from the source. Use it when you need reliable video reception but each extra connection to the source is costly, undesirable, or impossible.

How Cluster Ingest works

In this scenario only one streamer captures the external feed at any time. Others stay in reserve and do not open a second connection to the source.

A typical diagram:

```
flowchart LR
    source["Source"] --> primary["active origin"]
    primary --> edge["edge server"]
    edge --> player["player"]
    backup -.-> edge

    central["controller"] -- controls --> primary["primary streamer"]
    central -- controls --> backup["backup streamer"]
```

In `Agora` the controller implements this pattern. It tracks streamer health and switches capture. If the primary streamer is down, the controller tells the backup to start capture.

Resilience comes from controlled handover of active capture between streamers, not from parallel duplicate ingest.

When to use it

`Cluster Ingest` fits when:

- each subscription to the source is expensive;
- duplicate connections are undesirable;
- bandwidth cannot carry two copies of the same stream;
- you need resilience at capture time without duplicating the input.

A typical case is moving a video feed from headquarters to a region where a second full copy would be too costly or not allowed.

What this scenario provides

Main benefits of `Cluster Ingest`:

- resilience at the capture layer;
- preserves single-active ingest from the source;
- avoids constant duplicate traffic from the source;
- can move capture to a backup streamer when the primary fails.

Limitations and requirements

When designing `Cluster Ingest`, account for:

- correct streamer health checks from the `controller`;
- time to detect primary failure;
- time to switch capture to the backup;
- source behavior on reconnect after failure;
- configuration consistency between primary and backup streamers.

2.2 Admin

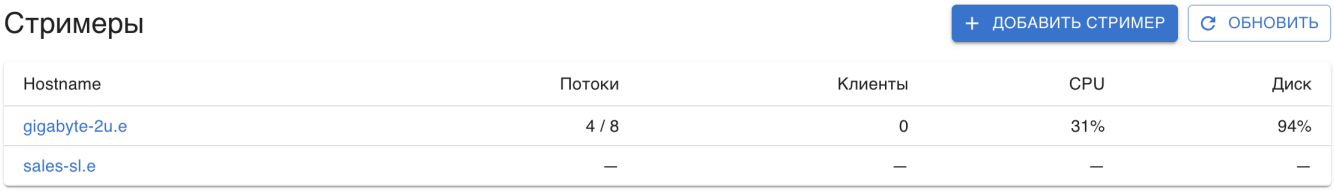
2.2.1 Streamer management

Streamers in Agora are execution nodes that handle ingest, publication, hardware devices, and operational telemetry. Managing them connects real servers to the platform, tracks their health, and uses them in streams and `IO` devices.

From the streamers section an administrator can:

- register `origin` and `edge` servers;
- set how the controller connects to them;
- monitor load and health;
- prepare the environment for streams and `IO` devices.

Streamer list



The list shows every registered node. Per streamer you see:

- `hostname`;
- running vs. configured stream counts;
- connected client count;
- CPU load;
- disk usage.

From the list you can:

- open a streamer card;
- add a streamer;
- refresh manually.

This screen is a quick health view of Agora infrastructure nodes.

Creating a streamer

To create a streamer in the current UI you only need:

- `hostname`.

After creation the streamer gets an internal ID and can be configured further.

Streamer card

In the streamer card an administrator can:

- change `hostname`;
- choose connection scheme: `HTTP` or `HTTPS`;
- set the API port;
- configure `config_api_key`;
- configure `edit_auth`;
- save changes;
- reload streamer data;
- delete the streamer.

If there are unsaved edits, the UI warns before navigating away.

Connection parameters

Remote access is defined by:

- `HTTP` / `HTTPS` scheme;
- API port;
- `config_api_key` for external configuration access;
- `edit_auth` for edit operations.

These control how the `controller` talks to the streamer.

`config_api_key` lets the streamer use `config_external` and receive full server configuration from the controller — it acts as the password for centralized configuration.

Working with secrets

Secret fields support:

- show/hide value;
- copy `config_api_key`;
- generate a new `config_api_key`.

That simplifies bootstrap and key rotation.

curl command for external configuration

For an existing streamer the UI can show a ready-made `curl` command for the external configuration endpoint.

It helps:

- see which config the streamer will receive;
- verify the configuration URL;
- confirm `config_api_key` is correct.

Relations to other areas

Configured streamers are used when:

- binding `IO` devices;
- configuring `HDMI` / `SDI` hardware capture;
- designing clustered ingest and publication;
- monitoring infrastructure health.

2.2.2 Intranet CDN

Intranet `CDN` in Agora delivers streams to remote offices and corporate network segments where broadcasting straight from the central origin would overload slow or expensive links.

In this mode Agora does not send the viewer straight to the central origin but to the appropriate restreamer inside their network zone. The same stream is delivered to the remote site once, then served locally via the restreamer.

How CDN works in Agora

Corporate `CDN` uses these entities:

- `origin` streamers — nodes where the stream is ingested or produced in Agora;
- `restreamer` — distribution node that pulls from origin and serves viewers in its zone;
- `zone` — logical corporate network area, often a remote office;
- `CIDR` routes — rules mapping the viewer's IP to a zone;
- `fallback zone` — backup zone used when the primary zone has no available restreamers.

The viewer opens the stream in the browser. Agora determines the client IP, finds the matching zone, picks an available restreamer with the lowest current load in that zone, and returns a playback address.

If the zone has no available restreamers, Agora uses the fallback zone. If the client IP matches no route, CDN routing does not apply for that client.

When administrators configure CDN

Configure `CDN` when:

- you have many viewers and a single `origin` cannot serve them all;
- staff watch the same live feeds from remote branches;
- the link between HQ and a branch is slow or costly;
- you need fewer simultaneous connections to the central origin;
- IP ranges must map to local distribution nodes.

Prerequisites

Before CDN setup, prepare:

- at least one working `origin` streamer with the stream already available;
- at least one server to register as `restreamer`;
- branch or office network ranges in `CIDR` form, e.g. `10.20.30.0/24`;
- a public-facing playback base URL per restreamer for viewers, e.g. `https://edge-1.office.example`;
- optionally a fallback zone for failover.

Before CDN, confirm origin streamers appear under Streamers and are healthy.

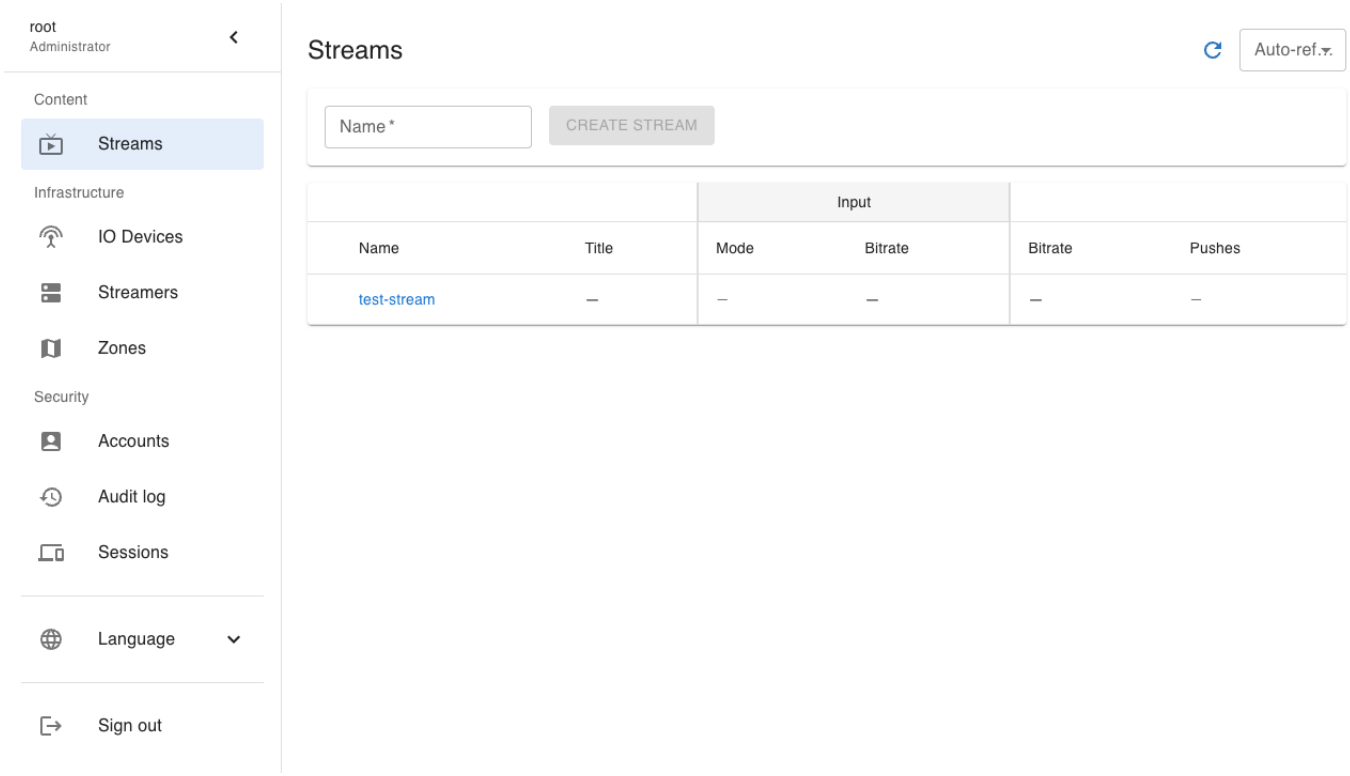
Recommended setup order

Suggested order:

1. Configure origin streamers.
2. Create zones.
3. Configure zone `CIDR` routes.
4. Set fallback zones where needed.
5. Configure restreamers.
6. Assign restreamers to zones.
7. Verify restreamer external configuration delivery.
8. Test the viewer portal from the right network.

This order reduces mistakes from empty zones, missing playback URLs, and wrong client routing.

Streams section: stream list



Streams are created and edited under **Streams** in the left menu. For CDN you need at least one stream on origin that viewers open through the viewer portal and balancer. See Streams for stream configuration.

Creating a zone

A zone represents an office, branch, or network segment where viewers should be served by a local restreamer.

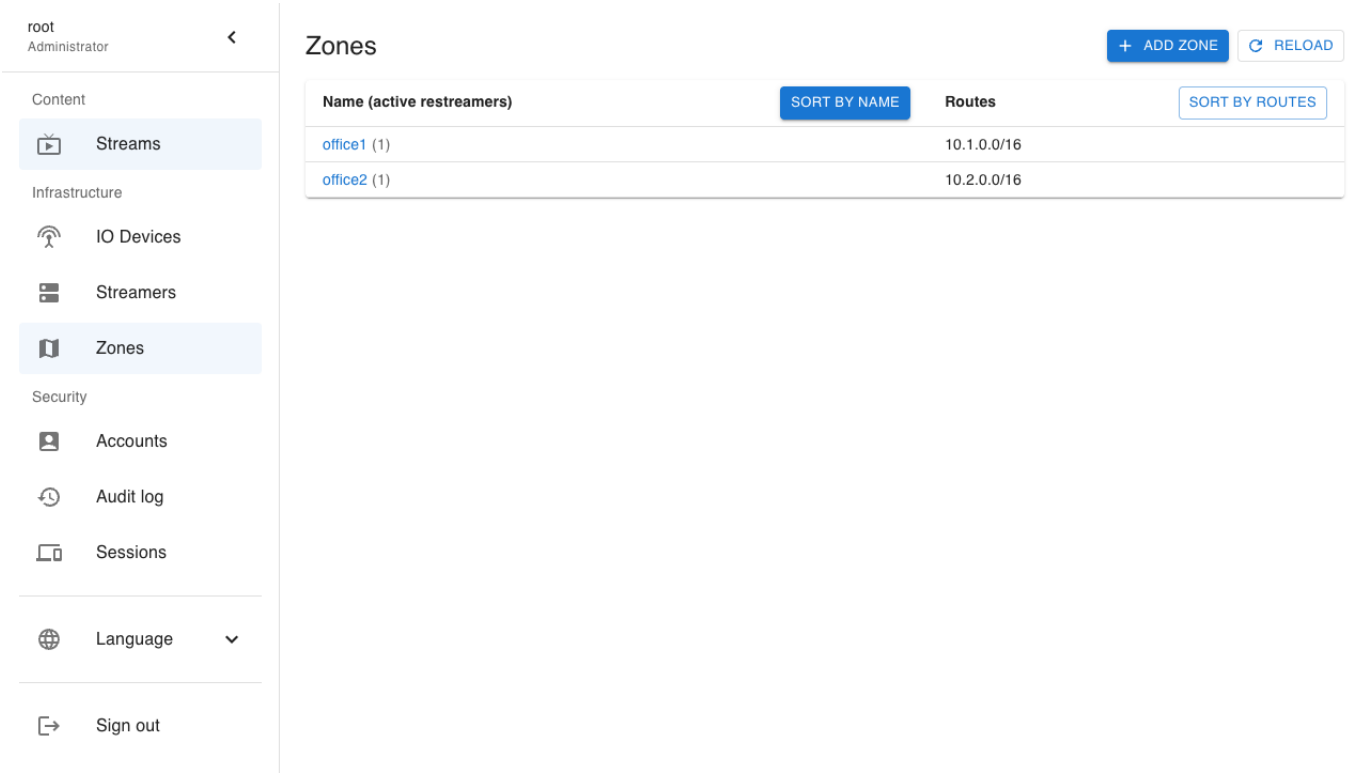
In the zone card an administrator sets:

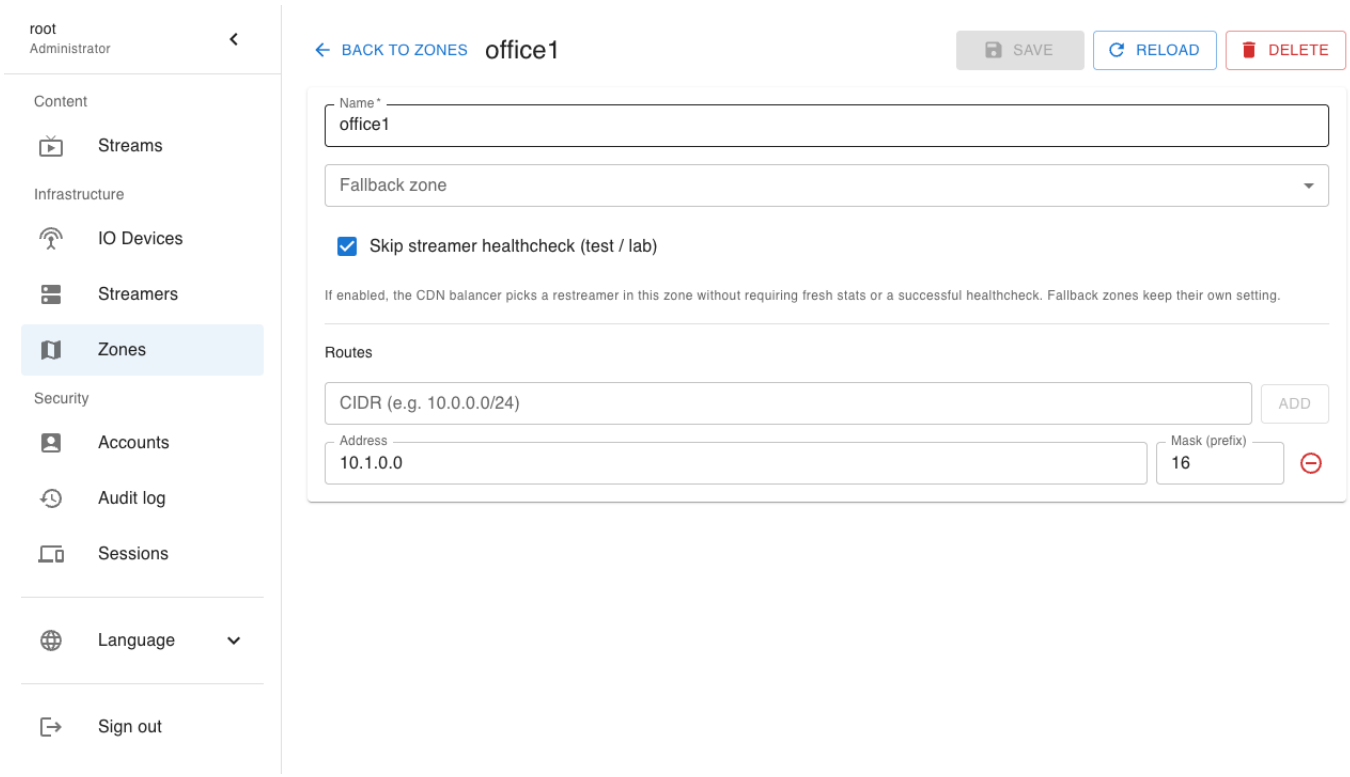
- `name`;
- `fallback zone` if a backup is required;
- the list of `CIDR` routes;
- **"Skip streamer liveness check"** (`skip_streamer_healthcheck` in the API): when enabled, the built-in balancer, when routing **into this zone**, picks a restreamer without requiring fresh stats or a successful edge healthcheck. Disabled (`disabled`) restreamers still never participate. The fallback zone has its own setting. Intended for lab CDN tests only, not production with real viewers.

The zone name should be unique and clear, for example:

- `msk-office-1`;
- `spb-office`;
- `plant-a`;
- `vpn-users`.

The fallback zone is used when the primary zone has no available restreamer.





The card screenshot shows a zone with a saved `CIDR`, no fallback zone, and **"Skip streamer liveness check"** enabled for lab testing.

Configuring CIDR routes

Zone routes use `CIDR` notation.

In the admin API each route is sent as `address` (string — IPv4 network address in **dot notation**, e.g. `10.0.0.0`) and `mask` (integer — prefix length as in CIDR: `24` for `/24`). Internal storage may differ; at the API boundary the address is always a dotted string. A value like `10.0.0.0/24` in the admin UI maps to that field pair.

Valid route examples:

- `10.12.13.0/24`;
- `10.12.0.0/16`;
- `192.168.100.0/24`;
- `0.0.0.0/0`.

Agora matches the viewer IP against routes in all zones and picks the longest prefix match, so more specific rules win over broader ones.

For example:

- zone `Office-A` has `10.12.0.0/16`;
- zone `Office-A-floor-3` has `10.12.13.0/24`.

A viewer at `10.12.13.42` maps to `Office-A-floor-3` because `/24` is more specific than `/16`.

If the same `CIDR` is already used in another zone, move or remove it from the old zone. A route must not belong to multiple zones at once.

`0.0.0.0/0` is the default route. Without it, a client that matches no route gets no restreamer and no playback.

Configuring a fallback zone

Use a fallback zone when all restreamers in the primary zone are unavailable or disabled.

When configuring fallback:

- pick a fallback zone the site can actually reach;
- avoid circular fallback chains;
- keep the design simple for operations.

Typical example:

- zone `office-ekb` has a primary local restreamer;
- zone `office-msk` is the `fallback zone` for `office-ekb`.

If the local restreamer in `office-ekb` is down, viewers in that zone receive the stream via `office-msk`.

Configuring a restreamer

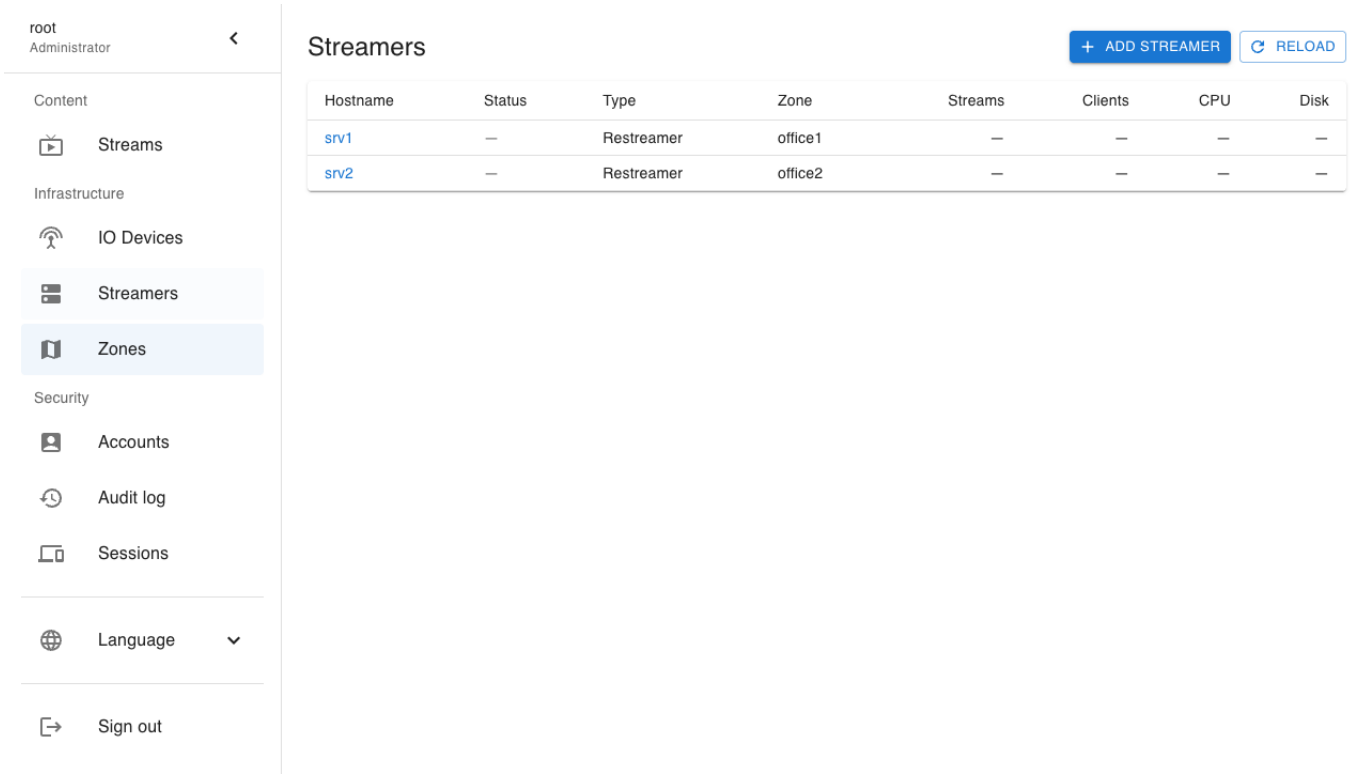
A restreamer is created in the same streamers section as a normal streamer, with node type `restreamer`.

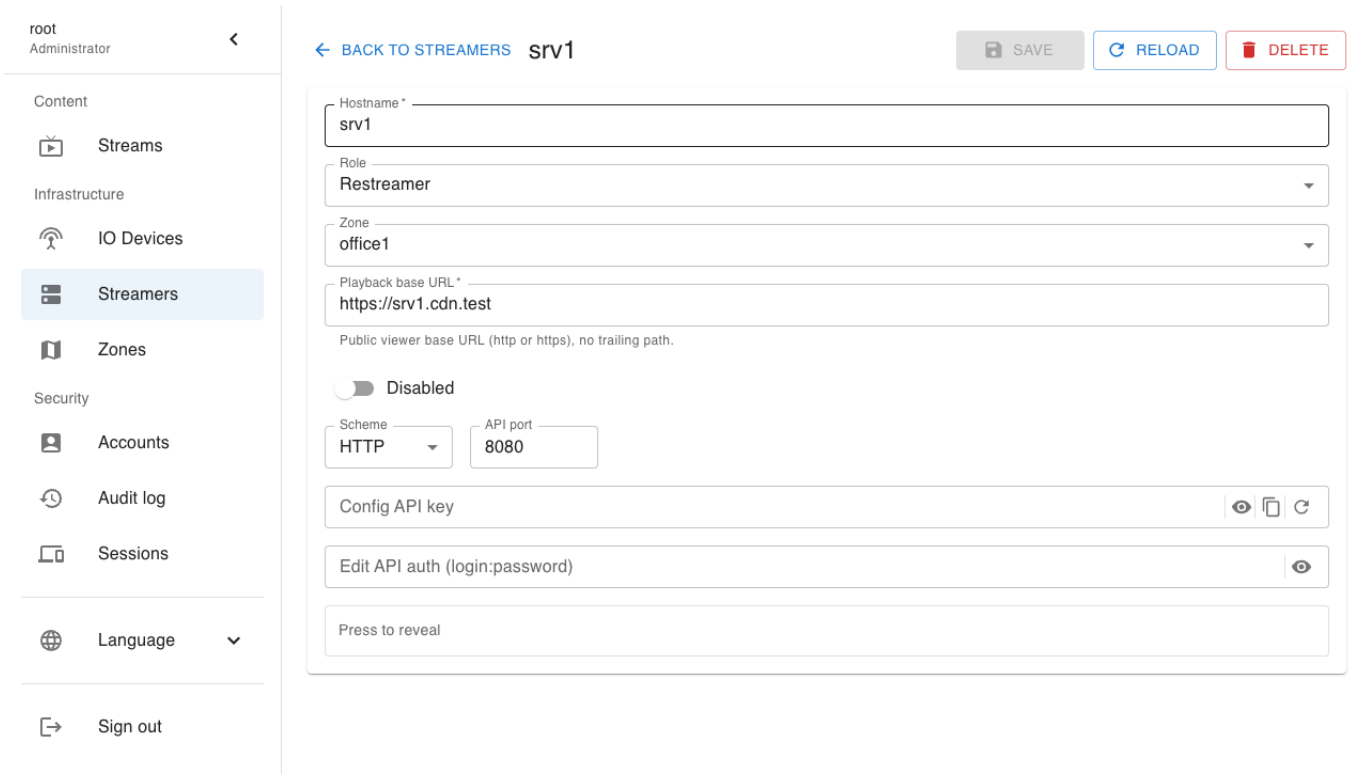
The restreamer card includes:

- `hostname`;
- node type `restreamer`;
- connection scheme `HTTP` or `HTTPS`;
- API port;
- `zone`;
- `playback_base_url`.

`playback_base_url` is the URL Agora returns to viewers for playback — usually the restreamer's external URL on the corporate or user network, e.g. `https://edge-1.office.example`.

A restreamer must be bound to a zone. Each restreamer belongs to exactly one zone.





What the restreamer receives from Agora

Like other streamers, a restreamer pulls configuration via `config_external`. In the `CDN` scenario Agora serves on-demand configuration for stream distribution.

That means:

- the stream on the restreamer is not started until a viewer requests it;
- the restreamer ingests from origin as `m4s`;
- publication on the restreamer starts on the first client request.

This lowers steady-state load and avoids keeping every possible stream active at once.

How Agora picks a restreamer for a viewer

When a viewer requests a stream, Agora:

1. Determines the client IP.
2. Finds a zone by `CIDR`.
3. Lists available restreamers in the zone.
4. Drops disabled and unavailable nodes.
5. Chooses the restreamer with the fewest current clients.
6. Returns a playback URL on that restreamer to the viewer portal.

“Available” restreamers are those that:

- are not disabled in the admin UI;
- pass healthcheck as healthy;
- have up-to-date operational statistics.

If **“Skip streamer liveness check”** is enabled for the zone, that selection step **for that zone** ignores healthcheck and freshness requirements: every zone-bound restreamer except admin-disabled ones is a candidate. When following a `fallback` chain, the target zone’s settings apply again.

Debugging CDN: source_ip query parameter

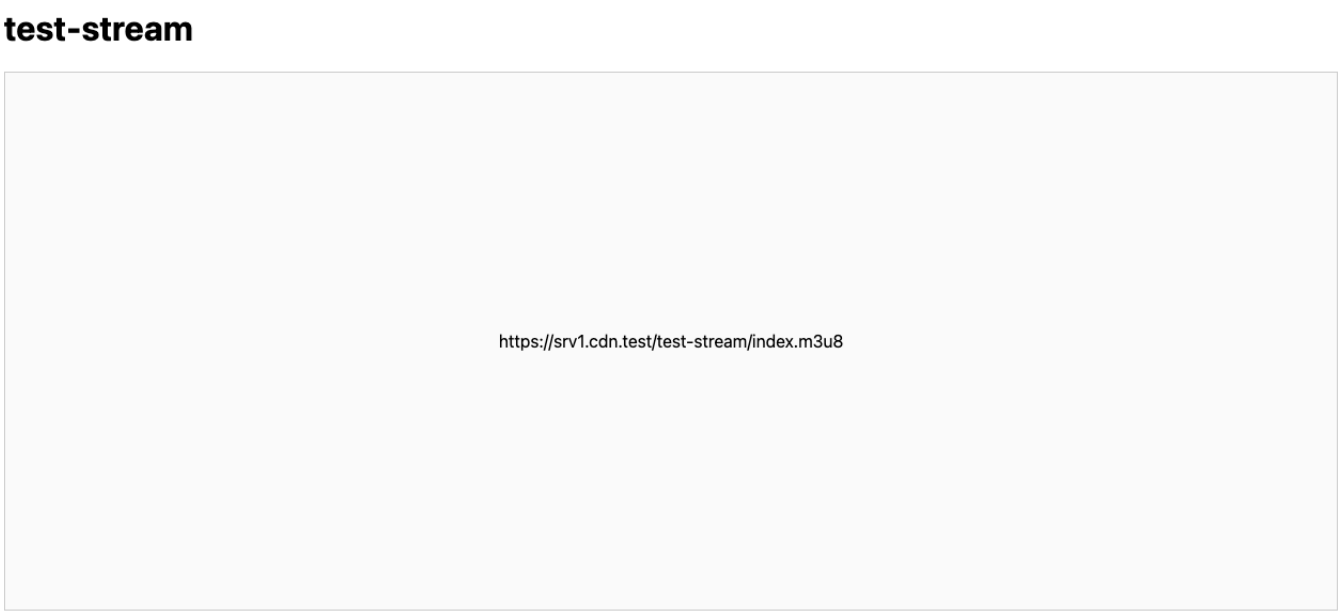
To test balancer and viewer portal routing from a workstation **outside** the target corporate subnet, add **`source_ip`** to the portal page URL: Agora treats the given IPv4 or IPv6 as the “viewer address” and applies `CIDR` routing as if the request came from that IP. The portal forwards the same parameter to `balancer/streams/{stream}` ; the viewer API `token` is forwarded when required.

Typical lab workflow with **`skip_streamer_healthcheck`** :

- enable **“Skip streamer liveness check”** on zones under test so you are not blocked on live stats or healthcheck;
- enter the correct `CIDR` routes for those zones;
- open the viewer portal with `?source_ip=…` and confirm the balancer response maps to the expected zone and `playback_url` .

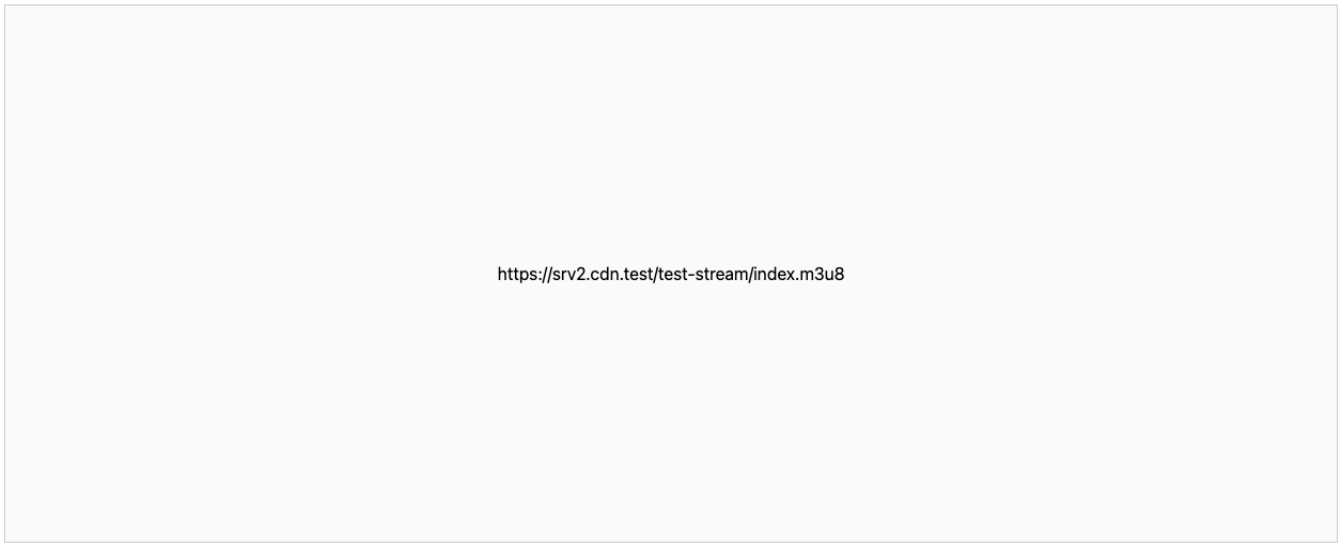
You can validate zones, fallback, and restreamer selection without being on the real subnet or having a “green” edge by metrics. Do **not** use **`source_ip`** in production: it overrides real client IP mapping and is for diagnostics and lab only.

Below are viewer portal examples: the panel shows the final `playback_url` from the balancer. With `source_ip=10.1.0.1` (zone with route `10.1.0.0/16`) the URL shows the first office edge (`srv1` in the sample lab):



With `source_ip=10.2.0.2` the zone `10.2.0.0/16` and matching restreamer (`srv2`) are selected:

test-stream



Verifying CDN configuration

After setup, walk through:

1. Open streamers and confirm the restreamer exists and is not disabled.
2. Confirm `zone` and `playback_base_url` are set.
3. Open the zone card and verify the `CIDR` is saved correctly.
4. If using fallback, confirm it is correct and skip-liveness is enabled only where you intend for tests.
5. Confirm the stream exists and works on origin.
6. Confirm the restreamer receives external configuration via `config_external`.
7. Open the viewer portal from a network whose IP maps to the intended zone.
8. Confirm playback goes through the local restreamer, not directly from origin.

Practically, test from a workstation in that branch or a test host in the right segment.

What the user sees when CDN is correct

For end users `CDN` is invisible. They open a normal stream link while Agora:

- picks the appropriate zone;
- chooses the best restreamer;
- directs playback to the local distribution URL.

Users do not manually pick office, node, or broadcast point.

Common misconfiguration issues

If `CDN` misbehaves, check:

- restreamer is `restreamer`, not a plain origin;
- restreamer is not disabled;
- `playback_base_url` is set;
- restreamer is bound to the correct zone;
- the zone has the right `CIDR`;
- the client IP actually matches the intended route;
- fallback zone is not wrong;
- origin streamer is up and serving the stream;
- restreamer receives `config_external`;
- the zone has at least one available restreamer.

If the client IP matches no route, CDN routing does not apply. Then verify masks, routing, and how traffic actually reaches Agora in your network.

Operational recommendations

For stable intranet `CDN`:

- name zones by physical or org topology;
- do not mix unrelated offices in one zone;
- keep at least one backup service path for critical sites;
- monitor restreamer health regularly;
- track origin and edge availability separately;
- document which IP ranges belong to which office.

That simplifies operations, scaling, and failover when the network or individual nodes fail.

2.2.3 Monitoring Agora

Agora monitoring has two parts:

- a system that collects raw metrics;
- a system that aggregates and can raise alerts.

Raw data is exposed as metrics over HTTP(S) as JSON or OpenMetrics (Prometheus). Most values are cumulative counters (the usual Prometheus style): bytes, error counts, and similar.

The aggregating system (Agora's own or the customer's) is expected to differentiate those counters over time.

We provide alert templates to bootstrap monitoring and extend them for your environment.

Built-in admin monitoring

Besides external metric collection, Agora includes monitoring screens in the admin UI so operators and admins can assess platform state without a separate observability stack.

Streamer monitoring

The streamers page shows aggregated metrics per registered streamer.

The list includes:

- streamer hostname;
- online stream count vs. total configured streams;
- connected clients;
- CPU load;
- disk usage.

You can:

- open a streamer card;
- refresh data manually;
- use the page as a quick infrastructure overview.

It helps spot node degradation, overload, or general platform load.

Stream monitoring

Stream monitoring is available on the stream list and inside each stream card.

On the overview list you see:

- overall stream status;
- input bitrate;
- current output bitrate;
- input operating mode;
- outgoing publication state.

The list supports manual refresh and auto-refresh on an interval.

Detailed stream statistics

The stream card shows detailed cluster statistics.

Depending on configuration, the UI may show:

- overall cluster status;
- `primary` and `backup` state in `Twincast` mode;
- effective input source;
- input bitrate;
- input statistics:
- bytes;
- frames;
- retries;
- input switches;
- media info changes;
- errors;
- last `DTS` time;
- outgoing publication state and stats.

In `Twincast`, stats are split for `primary` and `backup` to simplify troubleshooting per ingest path.

Outgoing publication monitoring

Agora also shows egress / outgoing publication statistics.

Per push you may see:

- publication URL;
- push status;
- bytes sent;
- frame count;
- error count.

In `Twincast`, egress stats may be split across servers so operators can compare `primary` and `backup`.

External monitoring

Built-in views are convenient for day-to-day work; for production, connect an external monitoring system.

That enables:

- long metric history;
- charts and dashboards;
- alerting;
- integration with corporate observability;
- feeding `Prometheus`, `Zabbix`, or other stacks.

What to watch first

For initial production monitoring, commonly track:

- streamer and relay availability;
- CPU, disk, and bandwidth load;
- active streams and clients;
- input bitrate and input errors;
- `Twincast` primary / backup health;
- outgoing publication errors;
- recording, archive, and content delivery health.

2.2.4 I/O devices

`IO` devices in Agora describe hardware inputs and outputs bound to specific streamers. They are configured separately from streams and can then be selected in a stream as signal sources, e.g. for `HDMI` or `SDI`.

Why IO devices matter

`IO` devices link Agora's logical configuration to real hardware on each streamer. Without that link you cannot reliably bind a stream to a physical `HDMI` or `SDI` source.

You define them in Agora so streams are assigned to the correct physical `origin` servers.

Why IO devices are needed

`IO` devices tell Agora:

- which streamer hosts a given card or port;
- what type of hardware source it is;
- how that source should be used.

Without prior `IO` setup the operator cannot pick the right hardware input in stream settings.

Binding to a streamer

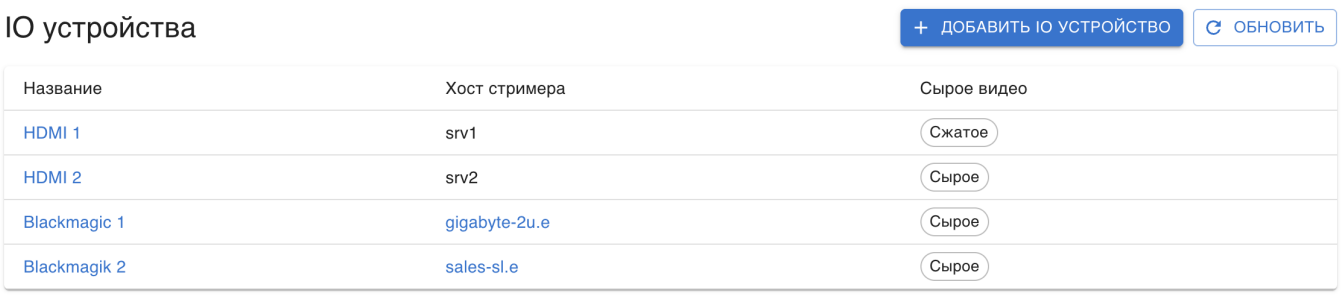
Each `IO` device is bound to a streamer. Hardware inputs exist on a specific capture server.

Therefore:

- configure `IO` devices in their own section;
- when creating a device, specify its streamer;
- only then can the stream use it as an `HDMI` or `SDI` source.

Hardware-capture streams are always tied to both input type and the streamer where the port lives.

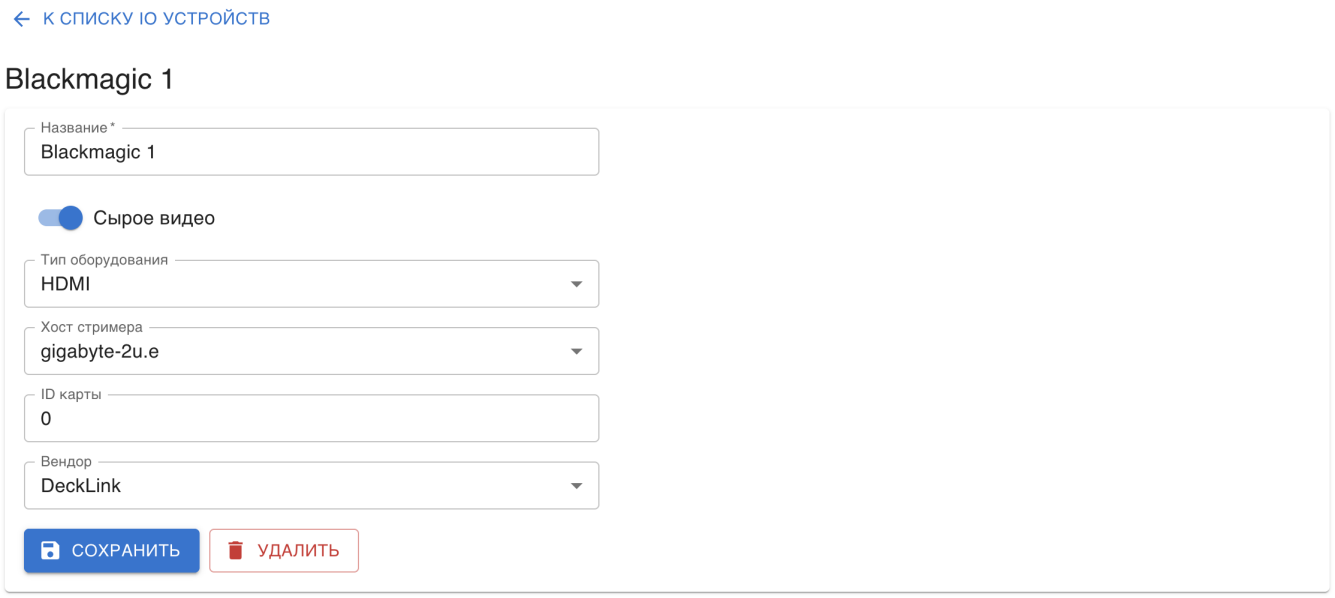
IO device list



On the list page you can:

- view all registered devices;
- see device name;
- see the bound streamer;
- see `raw` vs. normal mode;
- open a device card;
- create a device;
- refresh the list.

IO device card



The card exposes:

- device name;
- `raw` flag;
- hardware type `hw_type`;
- bound streamer;
- `card_id`;
- device vendor.

HARDWARE DEVICE TYPE

The UI supports:

- `HDMI`;
- `SDI`.

for the corresponding capture scenarios.

STREAMER BINDING

The form selects a streamer from known Agora nodes, binding the device to that capture node.

Without a streamer binding the device is not usable for production hardware ingest.

CARD ID AND VENDOR

For capture cards you can set:

- `card_id` — board or port identifier on the streamer;
- `vendor` — device manufacturer.

These tie Agora's logical config to the actual hardware resource.

Use in streams

Once an `IO` device exists and is bound to a streamer, it can be chosen as a stream source.

Especially for:

- `HDMI` capture;
- `SDI` capture;
- cases where the source is a physical port, not a network address.

The stream UI lists available devices; you pick by:

- streamer hostname;
- device name.

Device actions

Standard actions apply:

- create;
- edit;
- save;
- delete.

If the form changed, the UI warns about unsaved changes before leaving the page.

2.3 Manage

2.3.1 Content management

Agora manages several types of media content that differ in lifecycle, publishing model, and usage scenarios.

The main content types are:

- live broadcasts;
- recordings of past broadcasts;
- `VOD` files.

All of these are supported within a single management platform but follow different rules for preparation, publishing, storage, and delivery.

Each scenario is described in more detail on dedicated documentation pages.

2.3.2 Streams

Stream management in Agora

Separate independent video streams in Agora include:

- TV channels broadcast continuously (typically in-house production);
- one-off broadcasts of major events;
- feeds from fixed cameras that contribute to the content mix.

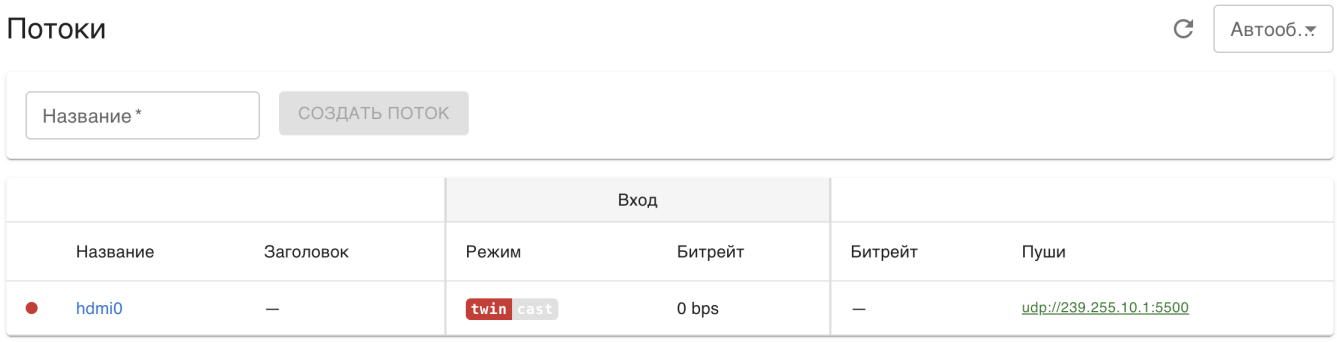
A stream in Agora is a cluster-wide view of all streamer and relay paths involved in delivery.

Source capture settings and clustering strategy are described on a dedicated page: Source capture.

WHY STREAM MANAGEMENT MATTERS

Stream management brings ingest, processing, outgoing publications, and health into one entity. The stream is the operator's main unit: through it you define broadcast logic, redundancy, transcoding, and content delivery.

STREAM LIST



On the streams page the operator can:

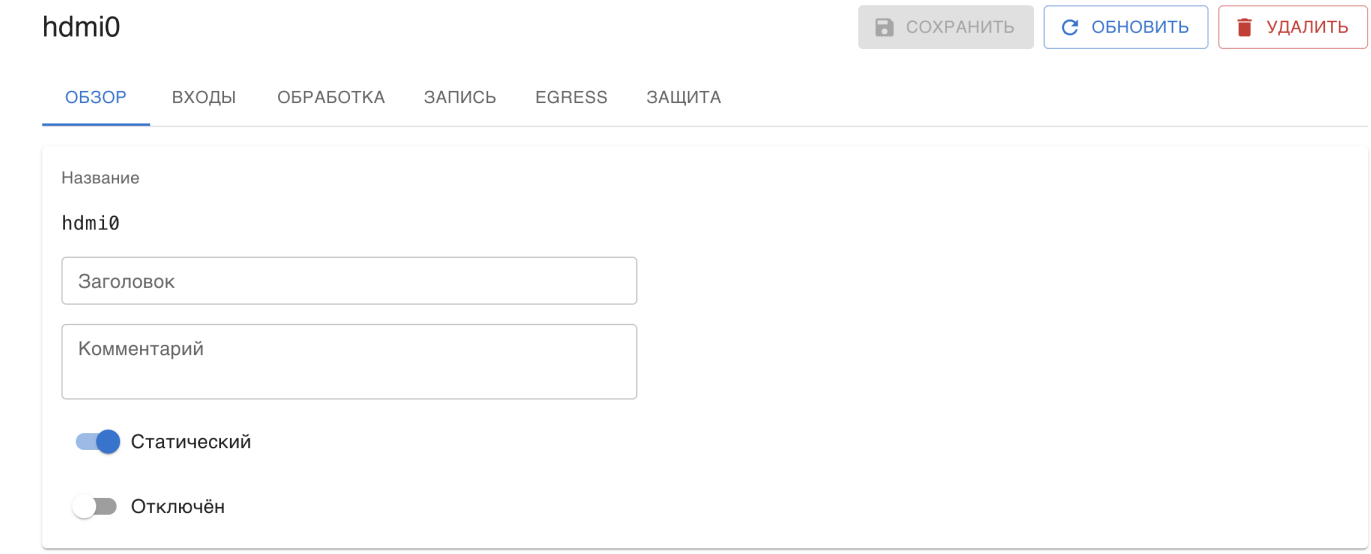
- view the list of all streams;
- create a new stream by name;
- open a specific stream card;
- refresh the list manually;
- enable automatic refresh every `5`, `15`, or `60` seconds.

The stream table shows key operational fields:

- overall stream status;
- system stream name;
- display title;
- input mode;
- input bitrate;
- current output bitrate;
- outgoing publications and their state.

For streams in `twincast` mode, the UI shows `primary` and `backup` path state separately. For outgoing publications, URLs and status are listed so you can see which pushes are active and which are waiting.

STREAM CARD



From the list you can open a stream card. There you can:

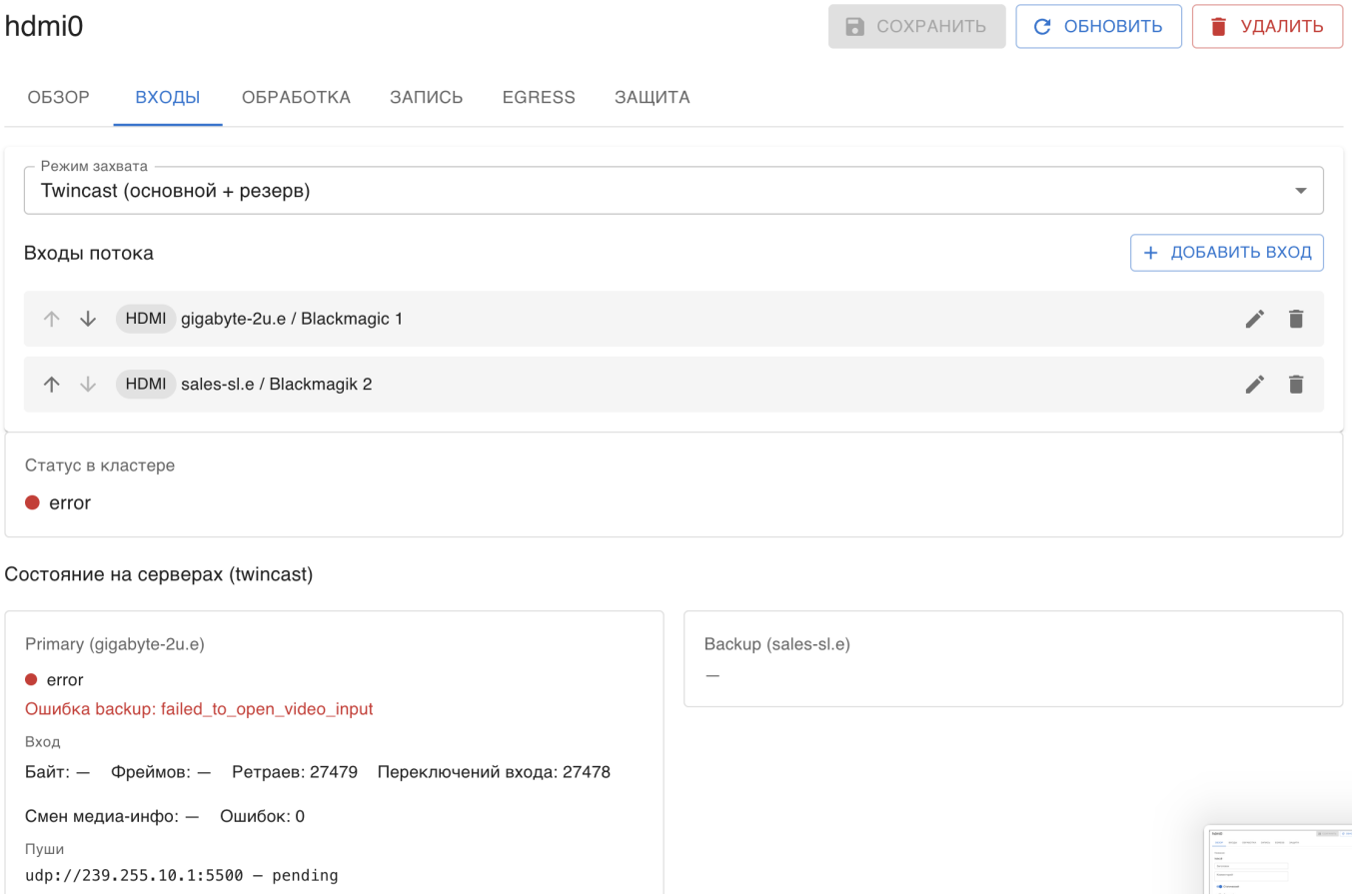
- view and edit basic stream properties;
- edit inputs;
- configure stream processing;
- manage publication and outgoing publications;
- view stream and push statistics;
- save changes;
- reload configuration;
- delete the stream.

Detailed editing for individual tabs is covered on dedicated documentation pages.

Source capture

Source capture settings in Agora are configured in the stream card on the Inputs tab. There the operator defines where the stream is ingested from, how inputs are ordered, and which clustering strategy applies to ingest.

WHAT YOU CONFIGURE ON THE CAPTURE PAGE



On the capture page the operator can:

- choose the input clustering strategy;
- add a new source;
- edit an existing source;
- remove a source;
- reorder inputs;
- view a short summary for each input.

When multiple inputs are configured, order matters, so the UI lets you move inputs up and down.

CLUSTERING STRATEGY

When configuring capture, the operator chooses a clustering strategy:

- `cluster ingest`;
- `double publish`;
- `twincast`.

That choice defines how the platform ingests the signal and behaves when one ingest path fails.

The default is `cluster ingest`. This is the standard mechanism that ensures an external stream is taken on one of the `origin` nodes or related capture paths.

In corporate TV, however, `cluster ingest` is less common than in classic broadcasting or video surveillance, because it assumes a source the system can pull from. Corporate TV more often relies on publish into Agora rather than autonomous pull.

These modes are described in detail on dedicated architecture pages:

- `Cluster Ingest`;
- `Double Publish`;
- `Twincast`.

SUPPORTED INPUT TYPES

The current Agora UI supports the following input types for a stream:

- `publish`;
- `multicast`;
- `SRT`;
- `RTSP`;
- `HDMI`.

Each input type has its own parameter set.

Multicast

For `multicast` you specify:

- IP address;
- port;
- program number, if used.

This input suits receiving a stream over the enterprise IP network.

SRT

For `SRT` in `caller` mode (Agora initiates the connection) you configure:

- host;
- port;
- passphrase, if the link is protected.

This mode is used for reliable ingest over IP.

`SRT` in Agora is commonly used in two scenarios:

- pulling a stream from the internet or a remote device when Agora connects to the source;
- receiving a publication when an external encoder publishes into Agora.

If the source publishes over SRT, use `publish` mode and configure the listening port for SRT.

RTSP

For `RTSP` you set the source URL.

This input is used for cameras, audio encoders, and other IP sources that speak `RTSP`.

HDMI

For `HDMI` the operator picks a device from the list of available `IO` devices.

The list shows:

- streamer hostname;
- device name.

If no free `HDMI` devices are available, the UI reports that separately.

Choosing such a device binds the stream to a specific streamer. For reliability, configure more than one device.

WORKING WITH THE INPUT LIST

For each input the list shows:

- input type;
- short parameter summary;
- edit actions;
- delete actions;
- reorder controls.

This lets the operator quickly verify ingest configuration and change input priority without editing internal data by hand.

AUTOMATIC FAILOVER BETWEEN SOURCES

Automatic failover between sources is one of the most important input mechanisms in Agora.

A stream may have several sources sorted by priority. Higher entries are preferred; lower ones act as backup or alternates.

If no frames arrive on the active source within `source_timeout`, the system switches to the next available source by priority.

That lets you prepare backup playout scenarios without manual operator action.

Example source priorities

A typical corporate TV setup:

- first priority: stream publication;
- second priority: external source capture.

Then:

- while publication is present, viewers see the studio feed;
- if studio publication drops and no frames arrive within `source_timeout`, the system switches to the second source;
- the second source may be a slate, a backup channel, or another prepared feed.

You can run studio live in normal conditions and fall back automatically without operator intervention.

PUBLISH-WAIT MODE

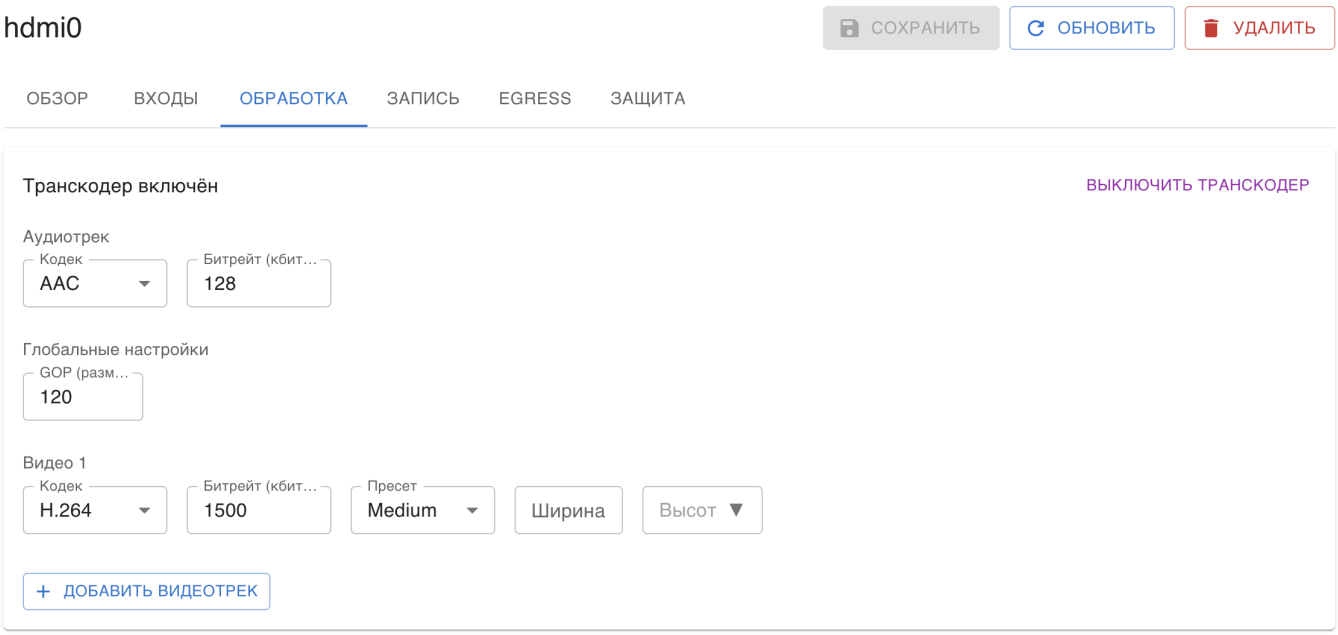
In some corporate TV scenarios the source is not fixed up front. The stream is created in publish-wait mode.

In that mode:

- the stream may have no predefined source;
- Agora does not start pull ingest on its own;
- the system waits for an external encoder or app to publish into Agora.

This is typical for studio workflows, software encoders, and one-off events: the operator prepares the stream in the system while actual signal delivery starts only at publish time.

Stream processing



Incoming streams in Agora usually need transcoding for correct playback on all target devices. Corporate TV sources vary widely in codec, bitrate, resolution, and structure; Agora prepares them for stable delivery over the enterprise network to typical clients.

This section configures the stream transcoder.

WHY PROCESSING MATTERS

With processing configured, Agora can:

- accept heterogeneous sources;
- normalize the input to a predictable format;
- ensure compatibility with client devices;
- cap and stabilize bitrate for corporate network delivery;
- prepare the stream for single-bitrate or multi-bitrate publication.

This is especially important in corporate TV, where the same service may combine:

- studio sources;
- `HDMI` / `SDI` hardware inputs;
- network publications;
- remote IP sources.

The transcoder makes these streams suitable for reliable delivery and playback on devices used in the organization.

WHY STREAM PROCESSING IS NEEDED

Transcoding is used to:

ENABLING AND DISABLING THE TRANSCODER

On the processing tab the operator can:

- enable the transcoder;
- disable the transcoder;
- change parameters for an enabled transcoder.

When enabled, Agora creates a baseline configuration by default:

- one audio track;
- one video track.

This matches typical single-bitrate delivery.

AUDIO TRACK

For the audio track the current UI lets you set:

- codec;
- bitrate.

Audio codec options:

- `AAC`;
- `Opus`.

VIDEO TRACKS

For each video track you can configure:

- codec;
- bitrate;
- encoding preset;
- width;
- height.

Supported video codecs:

- `H.264`;
- `H.265`;
- `AV1`.

SINGLE-BITRATE AND MULTI-BITRATE DELIVERY

By default Agora uses single-bitrate delivery: one output profile.

For multi-bitrate delivery, add several video tracks with different parameters:

- different bitrate;
- different resolution;
- different encoding presets if needed.

One input can thus become several outputs for different network conditions and devices.

BITRATE BEHAVIOR

By default Agora targets near-`CBR` over a `1` second window. That helps corporate networks because it makes bandwidth planning more predictable.

It helps you:

- estimate required capacity;
- reduce short overload spikes;
- simplify delivery planning inside the enterprise.

GOP

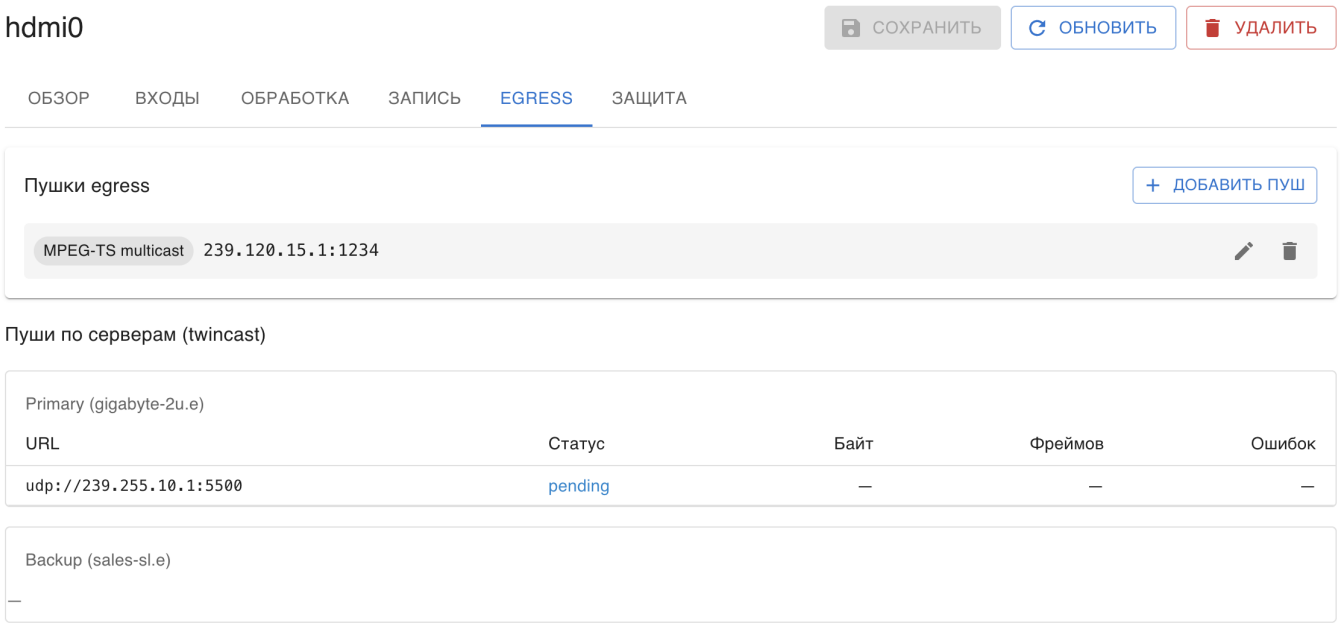
Global transcoder settings include `GOP`.

`GOP` affects the stream as follows:

- larger `gop size` tends to reduce bitrate at similar visual quality;
- larger `GOP` increases playback latency vs. real time;
- startup delay also increases.

For LAN corporate use, a `GOP` around `2` seconds is a practical default.

Outgoing publications with Agora



Outgoing publications define where Agora sends the prepared stream after ingest and processing. Here the operator configures egress targets for delivery inside the corporate network, to external systems, or to adjacent platform nodes.

WHY PUSHES MATTER

Outgoing publications (pushes) target systems that cannot pull video on their own, for example:

- social networks;
- external streaming hosts;
- multicast distribution.

WHAT YOU CAN CONFIGURE

On the outgoing publications page the operator can:

- add a new outgoing publication;
- edit an existing publication;
- remove a publication;
- view a short summary per push;
- see basic statistics for active publications.

SUPPORTED PUBLICATION TYPES

The current Agora UI supports:

- `MPEG-TS multicast`;
- `SRT`;
- `RTMP`.

Each type has its own parameters.

MPEG-TS multicast

For `MPEG-TS multicast` you specify:

- multicast IP;
- port;
- program number, if used.

This suits distributing a stream over the enterprise LAN in multicast scenarios.

SRT

For `SRT` publication you configure:

- host;
- port;
- passphrase, if the connection is protected.

This mode suits reliable delivery to a remote network segment, backup server, or external receiver.

RTMP

For `RTMP` you set the publish URL.

Use this when the downstream system accepts `RTMP`.

WORKING WITH THE PUBLICATION LIST

For each outgoing publication the UI shows:

- publication type;
- short parameter summary;
- edit actions;
- delete action.

When statistics exist, you also see:

- bytes transferred;
- frame count;
- error count.

That helps spot healthy vs. problematic publications quickly.

OUTGOING PUBLICATION STATISTICS

Agora exposes separate statistics for stream egress.

Per push you may see:

- publication URL or address;
- current status;
- bytes sent;
- frame count;
- error count.

Statuses indicate whether the publication is active, waiting, retrying, or in error.

STATISTICS IN TWINCAST MODE

If the stream runs in `Twincast` mode, outgoing publication statistics are split between:

- `primary`;
- `backup`.

That helps diagnose which path is publishing and whether the two sides differ.

2.4 Security

2.4.1 User management in Agora

Agora provides user management and access control.

Users exist only for authorization; Agora does not store or process personal data about individuals.

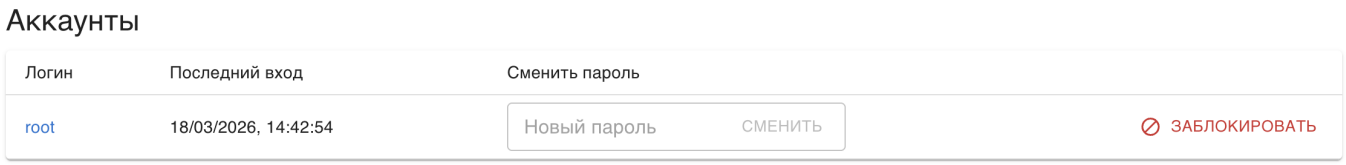
User management lives in the security area of the admin UI. The current UI offers an account list, per-user card, password change, lockout status, and links to related security data.

Bootstrap administrator

On the server run:

```
./create-account -u USERNAME -p PASSWORD
```

User list



On the user list an administrator can:

- view all accounts;
- create accounts;
- see login and role;
- see last login time;
- change password inline;
- open a user card;
- lock a user.

Locked accounts are indicated in the row.

CHANGE PASSWORD FROM THE LIST

To change a password quickly:

1. Find the user in the list.
2. Enter the new password in `New password`.
3. Confirm with the button at the field or press `Enter`.

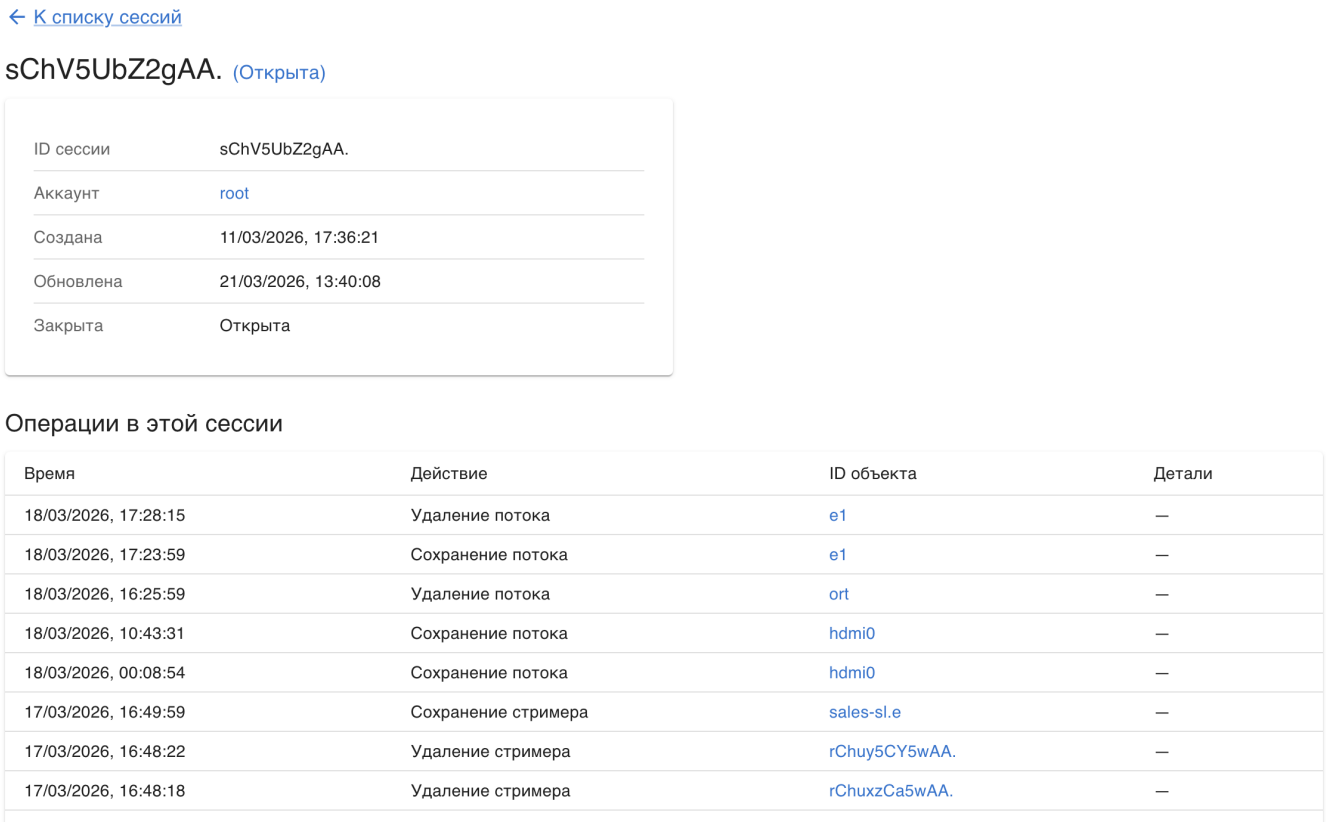
On success the UI shows a confirmation.

LOCK FROM THE LIST

Locking a user immediately terminates all active sessions.

Unlock on the account page.

User card



From the list, open a user card. There you see:

- `account_id`;
- login;
- `external_account_id` to link the account to an external system;
- account creation time;
- last modification time;
- last login time;
- lock time if locked;
- current open sessions and past closed ones.

The card also lets you:

- change the password;
- lock the user;
- unlock the user;
- return to the account list.

If password fields changed, the UI warns on navigate-away.

CHANGE PASSWORD IN THE USER CARD

To change password in the card:

1. Open the account page.
2. Enter the new password.
3. Confirm the change.

Errors appear under the password field.

LOCKING A USER

As in the list, the card provides lock with a confirmation dialog.

Locking immediately ends all open sessions for that user.

Related security features

User management ties to other security areas:

- the audit log shows user actions;
- sessions lists open and closed sessions;
- open sessions can be force-ended via logout.

User-related actions should appear in the audit log for later review and incident response.

Even if user lock APIs are incomplete, security staff can still monitor accounts and end sessions when needed.

Role model

Agora defines several roles:

- administrator — manages streamers, devices, and external settings;
- content manager — manages streams and `VOD` assets;
- observer — can view monitoring but cannot edit streams or settings;
- security — can read the audit log and lock users or sessions when needed.

This model separates admin, content, and security duties. Detailed permissions per role will be documented on dedicated security pages.

2.4.2 Using OIDC for user provisioning and authorization

Agora can integrate its user store with a corporate directory over OIDC.

The local user database can be disabled (optional — you can keep it) and becomes fully driven by the external system (usually the central corporate directory, below **IdP** — identity provider). User rights are configured externally; user create, lock, and delete happen externally.

Important: Agora does not call the IdP on every API request — it **validates the JWT** (signature via **JWKS** or a local key) and reads access rights for that request from the token.

JIT provisioning into the database happens on the first successful request with a valid IdP token **only if** the token contains a **mappable role** (see `OIDC_ROLE_CLAIM` / `OIDC_ROLE_MAP`). There is **no** default fallback role; without a mappable role the account is **not** created and the request is denied (**403**). If local login stays enabled, built-in accounts and JWT issuance via `POST /login` continue to work alongside the IdP (hybrid mode).

All communication with the IdP uses HTTP(S). LDAP and similar are handled by the external system (the IdP itself).

OIDC is not the only mode: by default Agora uses a built-in user database and local user management, issuing its own JWTs.

OIDC setup can be complex, with many knobs and variation across corporate deployments.

The rest of this document walks through:

1. Running OIDC with Authentik locally
2. Which JWT fields we use
3. Agora OIDC configuration options
4. A glossary of terms used here

1. Running OIDC with Authentik

1.1. DEPLOYING AUTHENTIK

Full Authentik installation (Docker Compose, Kubernetes, LDAP integration, etc.) is covered in the official Authentik documentation.

Here we run Agora and Authentik on a server in the LAN with Docker Compose to illustrate OIDC configuration. Moving to production does not change the approach.

Start Authentik locally:

```
echo "PG_PASS=$(openssl rand -base64 36 | tr -d '\n')" >> .authentik-env
echo "AUTHENTIK_SECRET_KEY=$(openssl rand -base64 60 | tr -d '\n')" >> .authentik-env
curl -o authentik-compose.yml https://docs.goauthentik.io/compose.yml
docker compose -f authentik-compose.yml --env-file .authentik-env -p authentik up -d
```

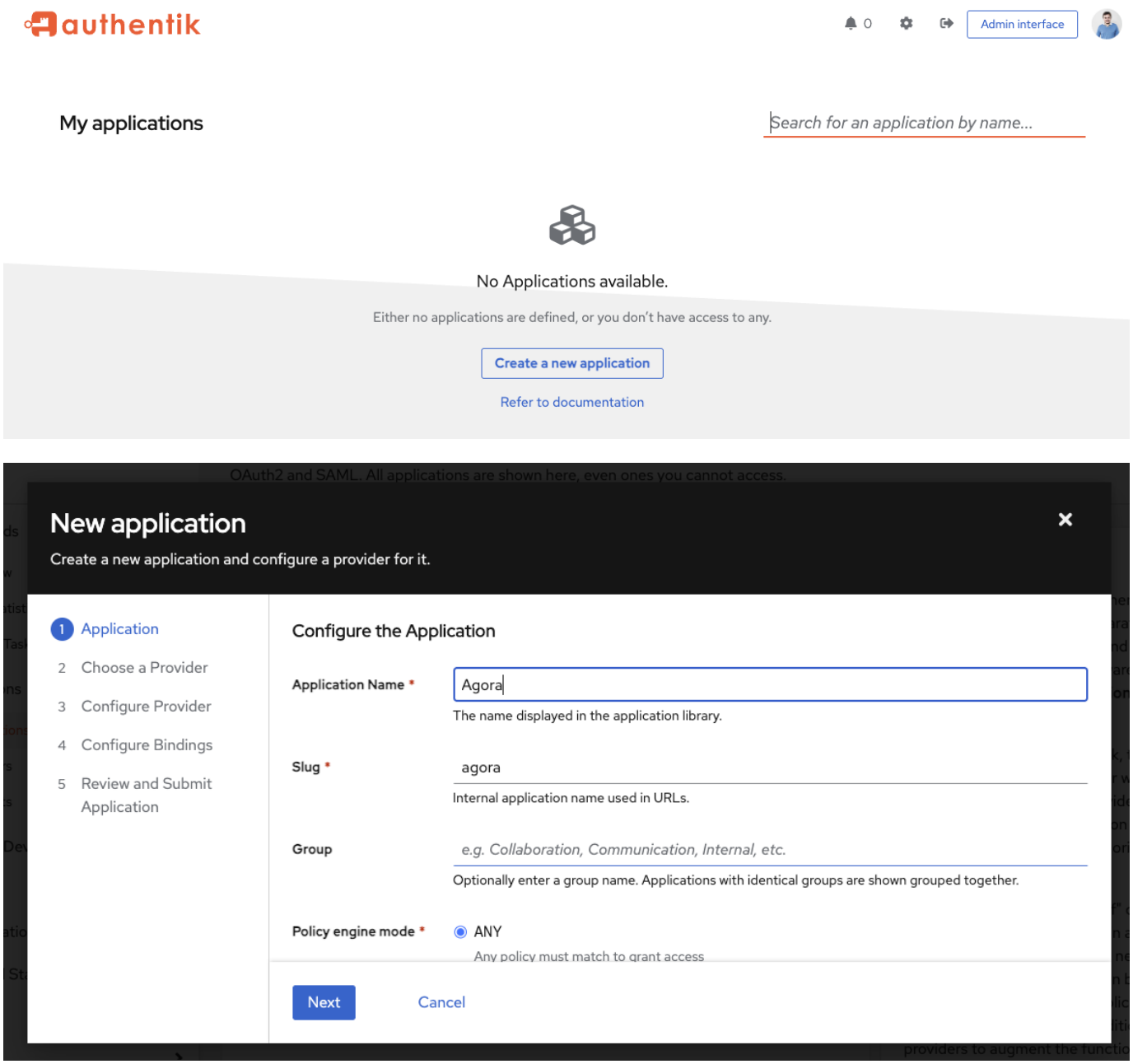
If something goes wrong, you can stop and fully remove Authentik:

```
docker compose -f authentik-compose.yml --env-file .authentik-env -p authentik down -v
```

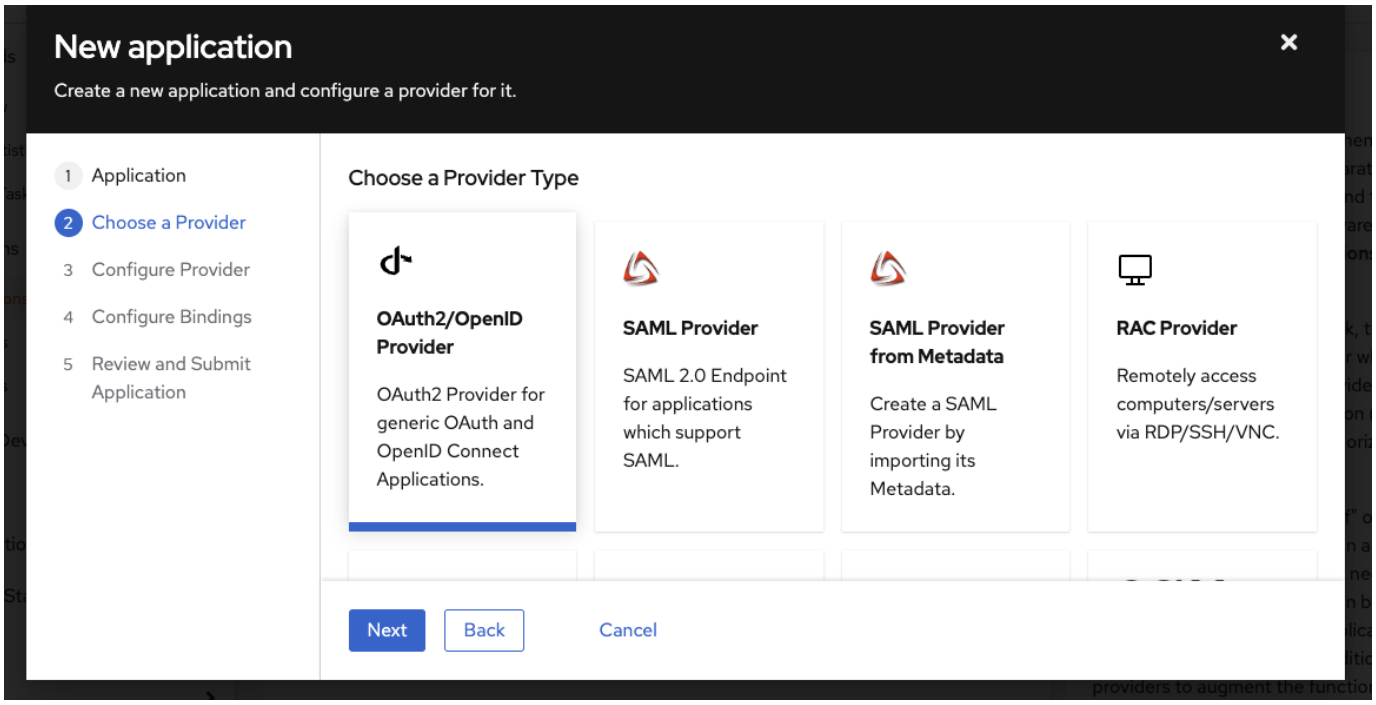
Open `http://authentik.local:9000/if/flow/initial-setup/` — here `authentik.local` is the host where you run the IdP (often `localhost`, but the browser must reach it). Create an Authentik administrator in that UI. Values here do not matter much; this is a temporary install.

1.2. CONFIGURE AUTHENTIK

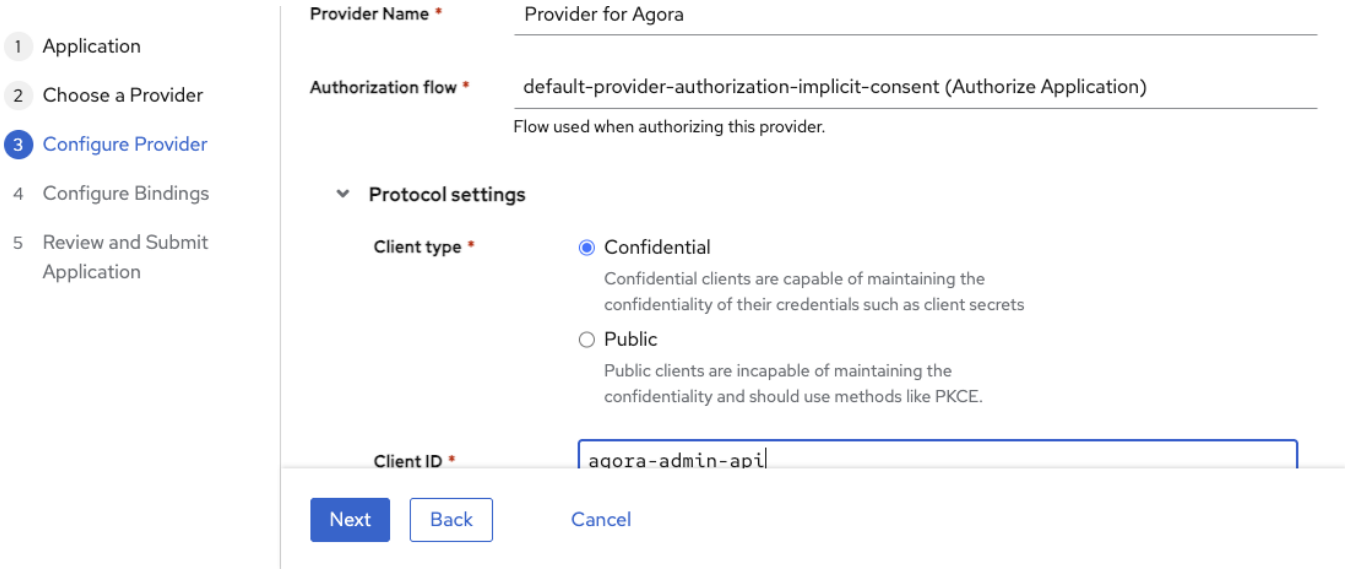
1. In Authentik create an **Application** named Agora (the slug is filled automatically).



1. Create an OAuth2/OIDC Provider (pick it from the list).

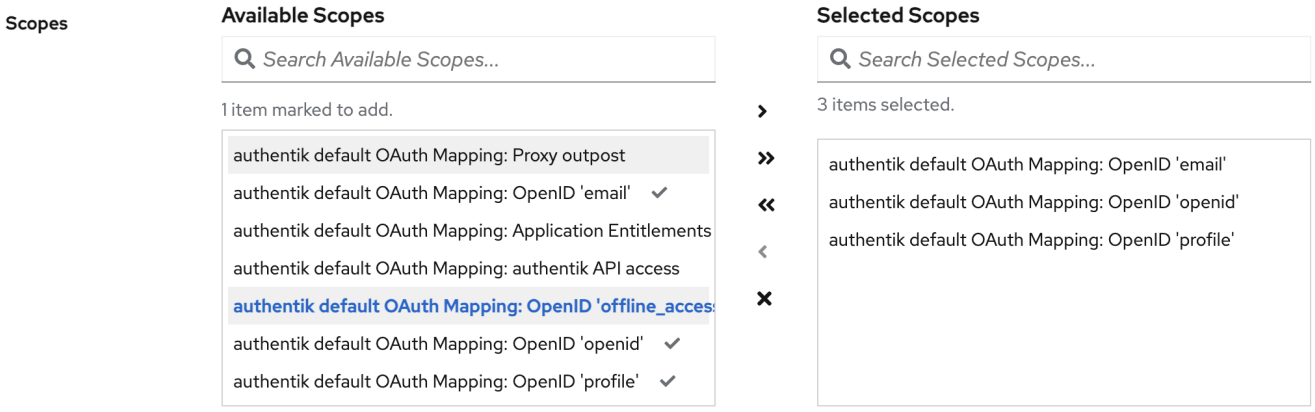


1. Choose **Authorization flow** implicit (default-provider-authorization-implicit-consent).

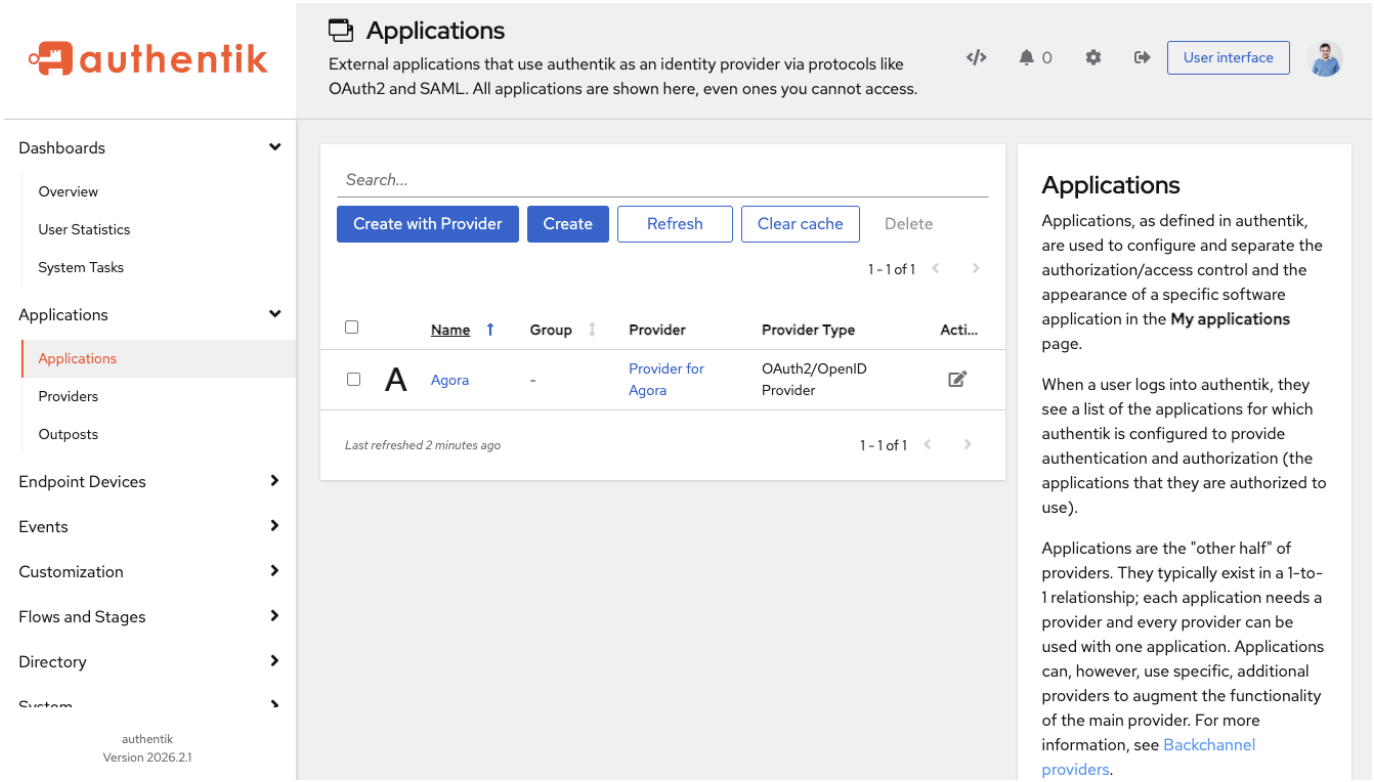


1. On that page set Client ID to `agora-admin-api` for convenience. Copy Client Secret from the page; you will put them in `OIDC_CLIENT_ID` and `OIDC_CLIENT_SECRET` (see Agora settings below).

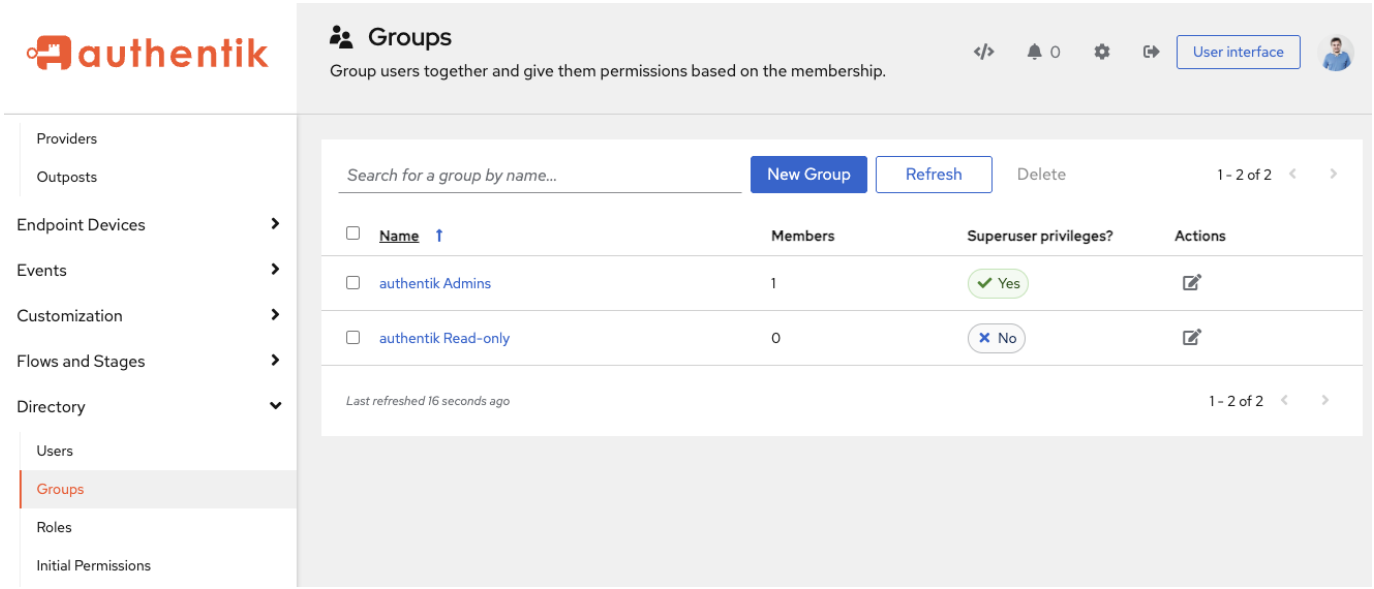
In Advanced protocol settings add `offline_access` to allowed Scopes.



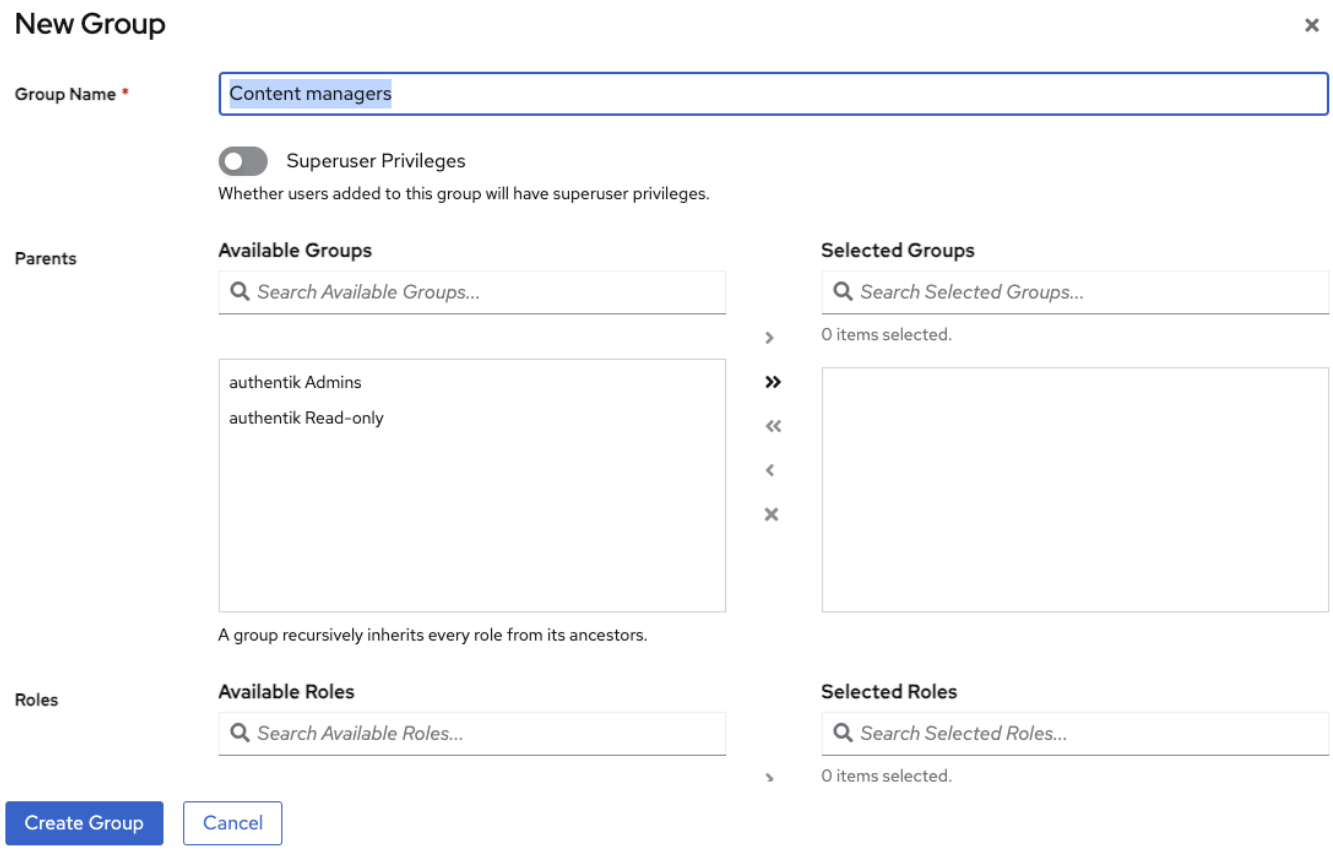
1. On the next page skip adding groups because none exist yet.
2. On the last page choose Submit — the application is created.



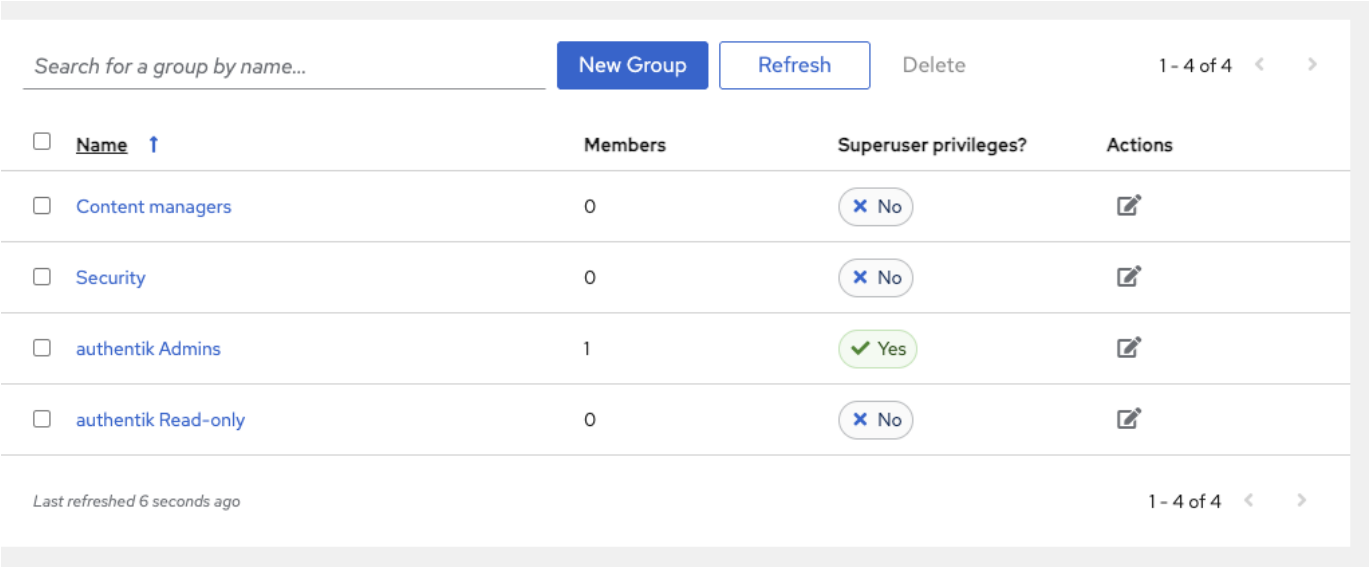
1. Open the Groups section.



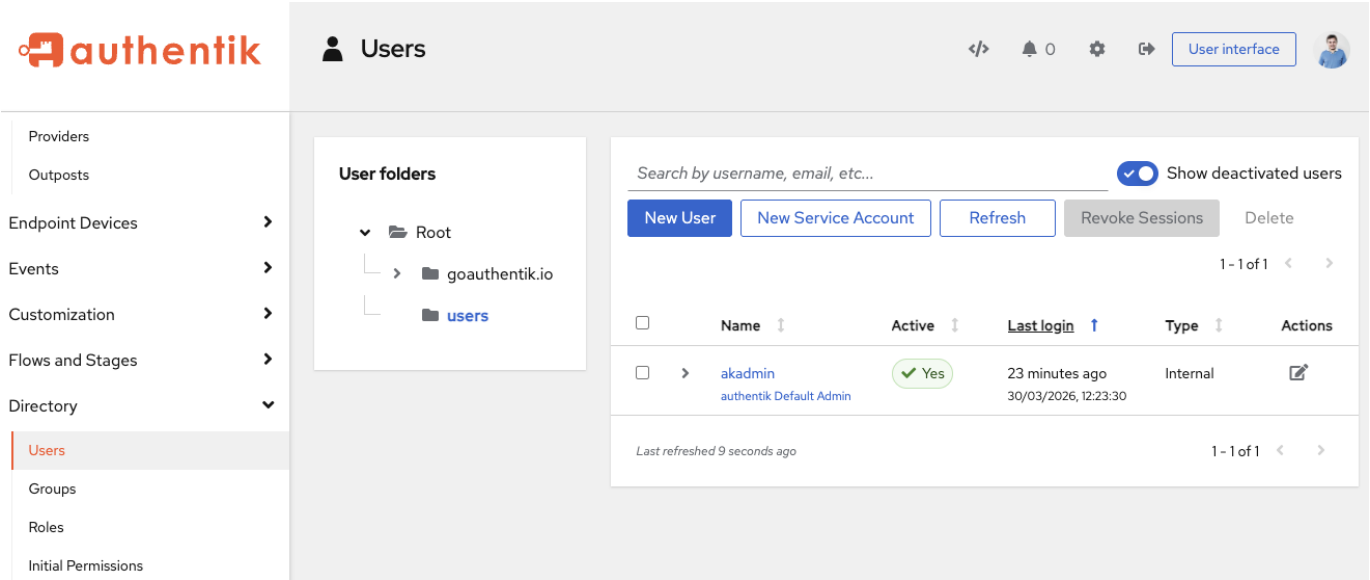
1. Create two groups, e.g. "Content managers" and "Security".



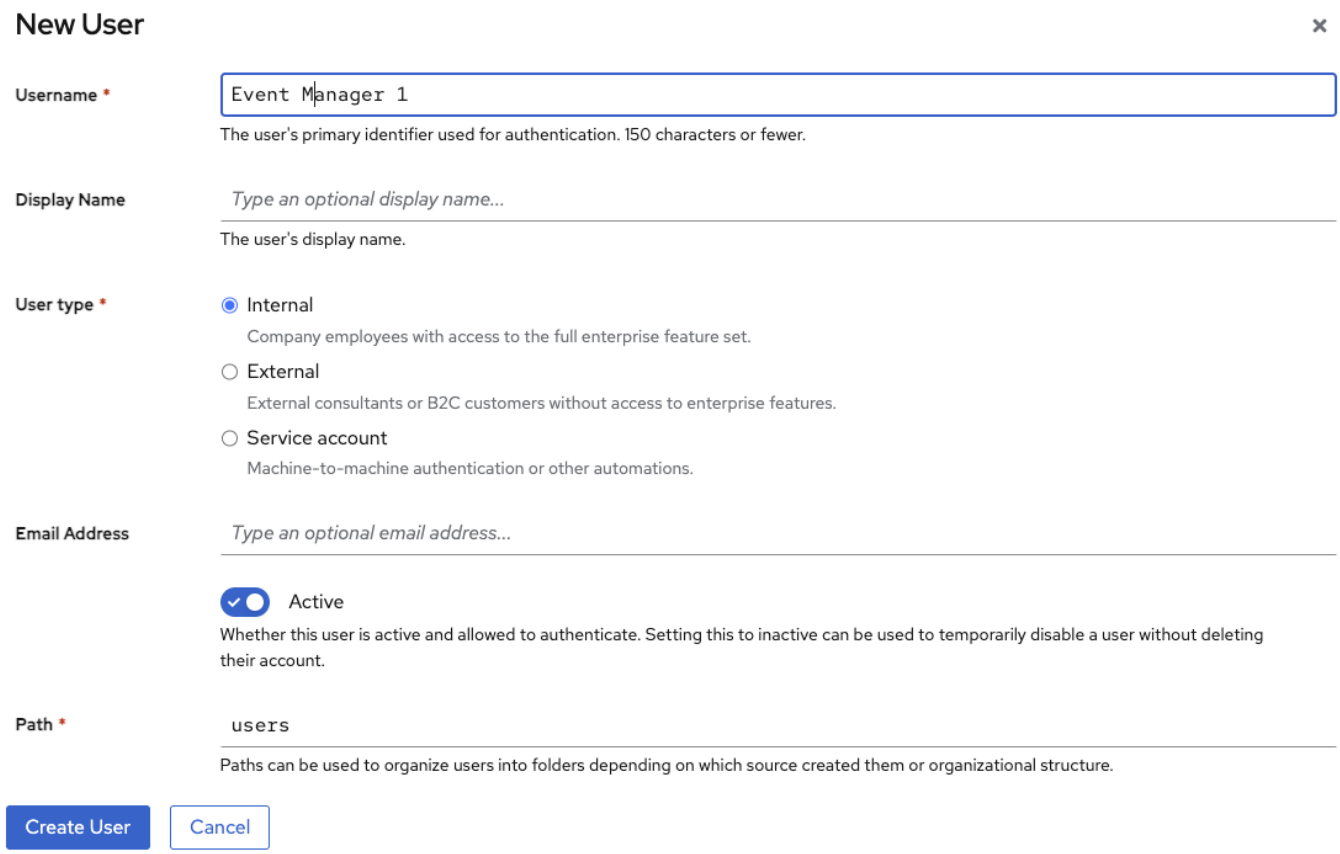
1. You will see a short list of groups.



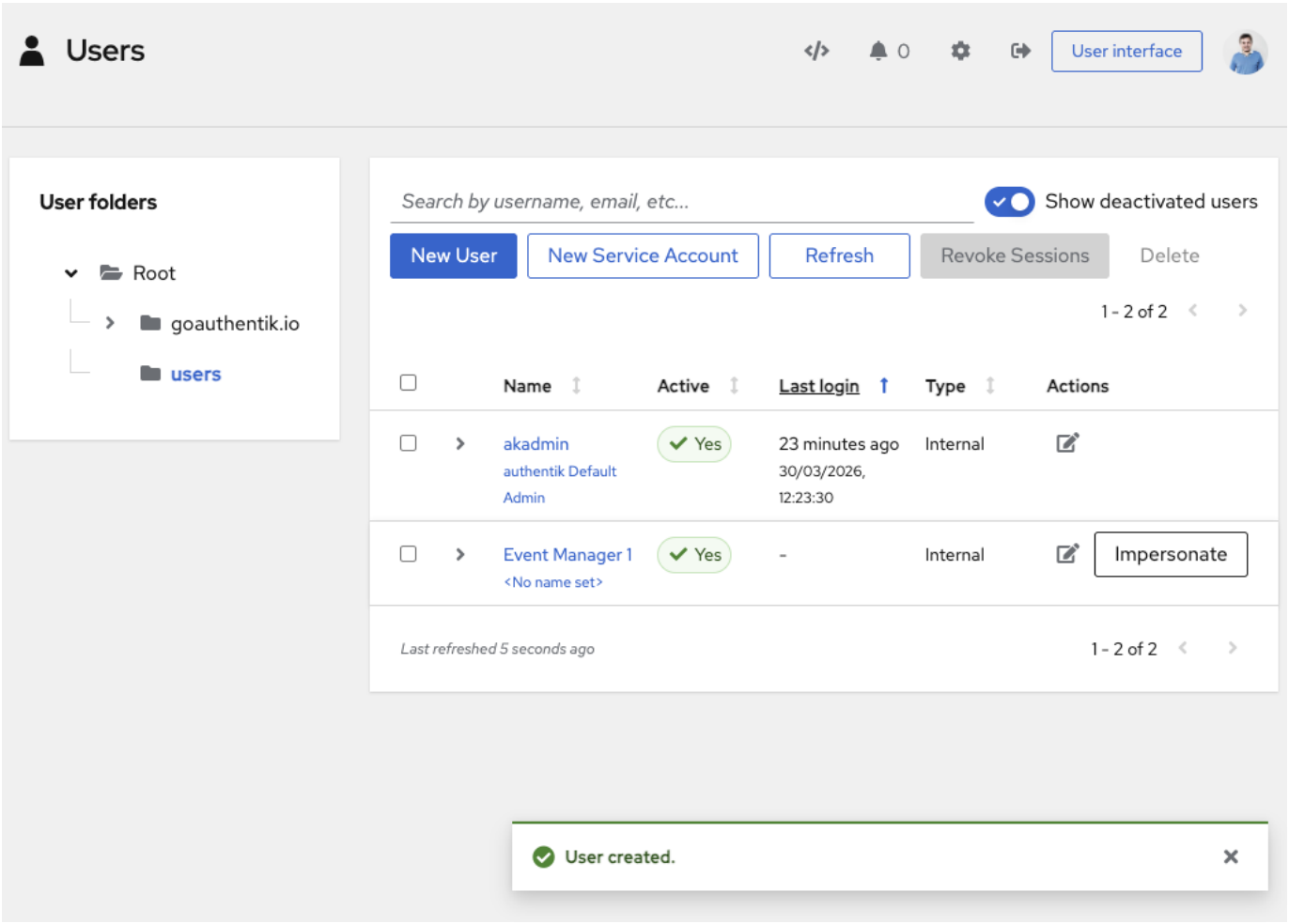
1. Open Users to create a test user.



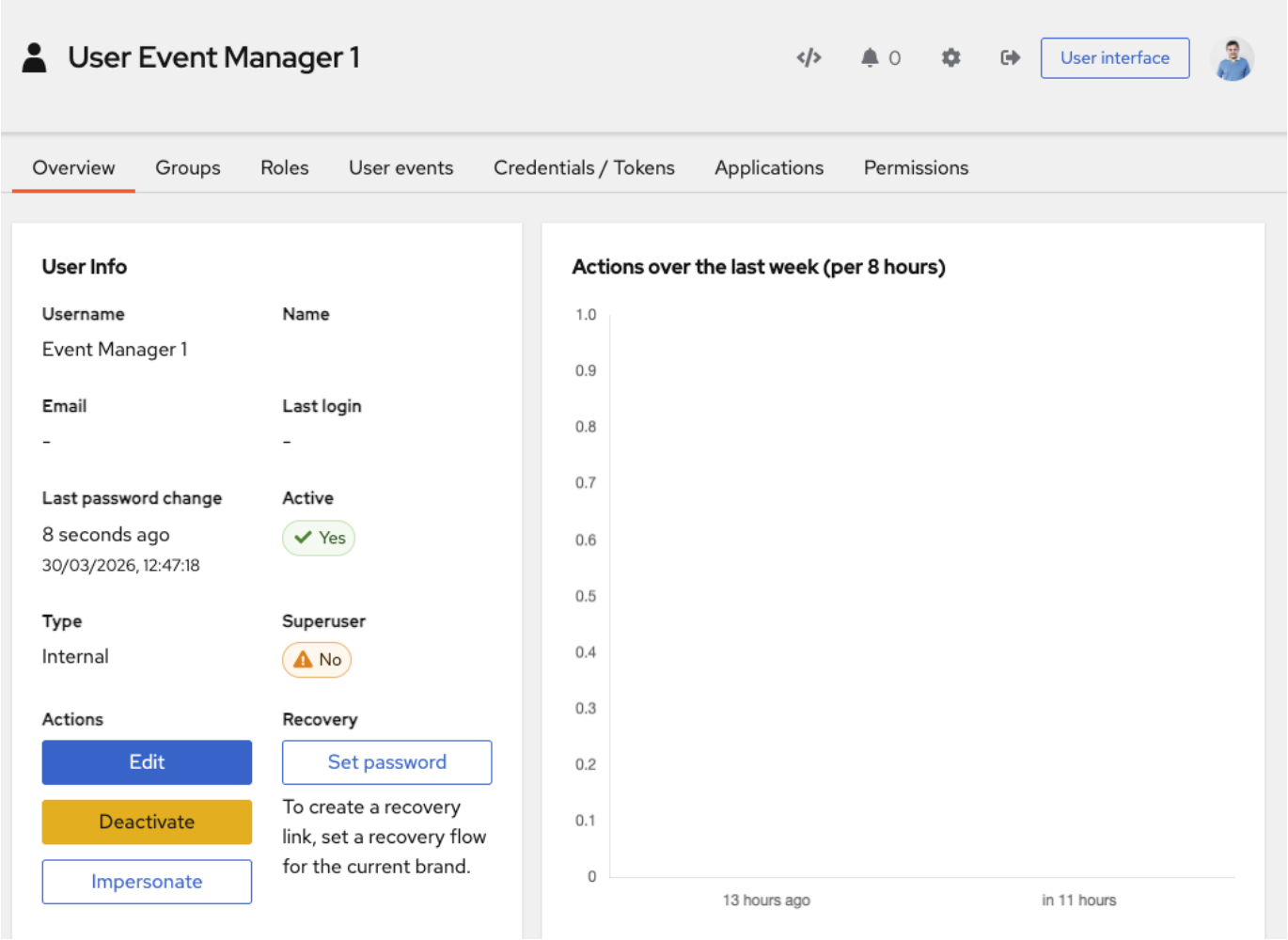
1. Create a user, e.g. "Event Manager 1".



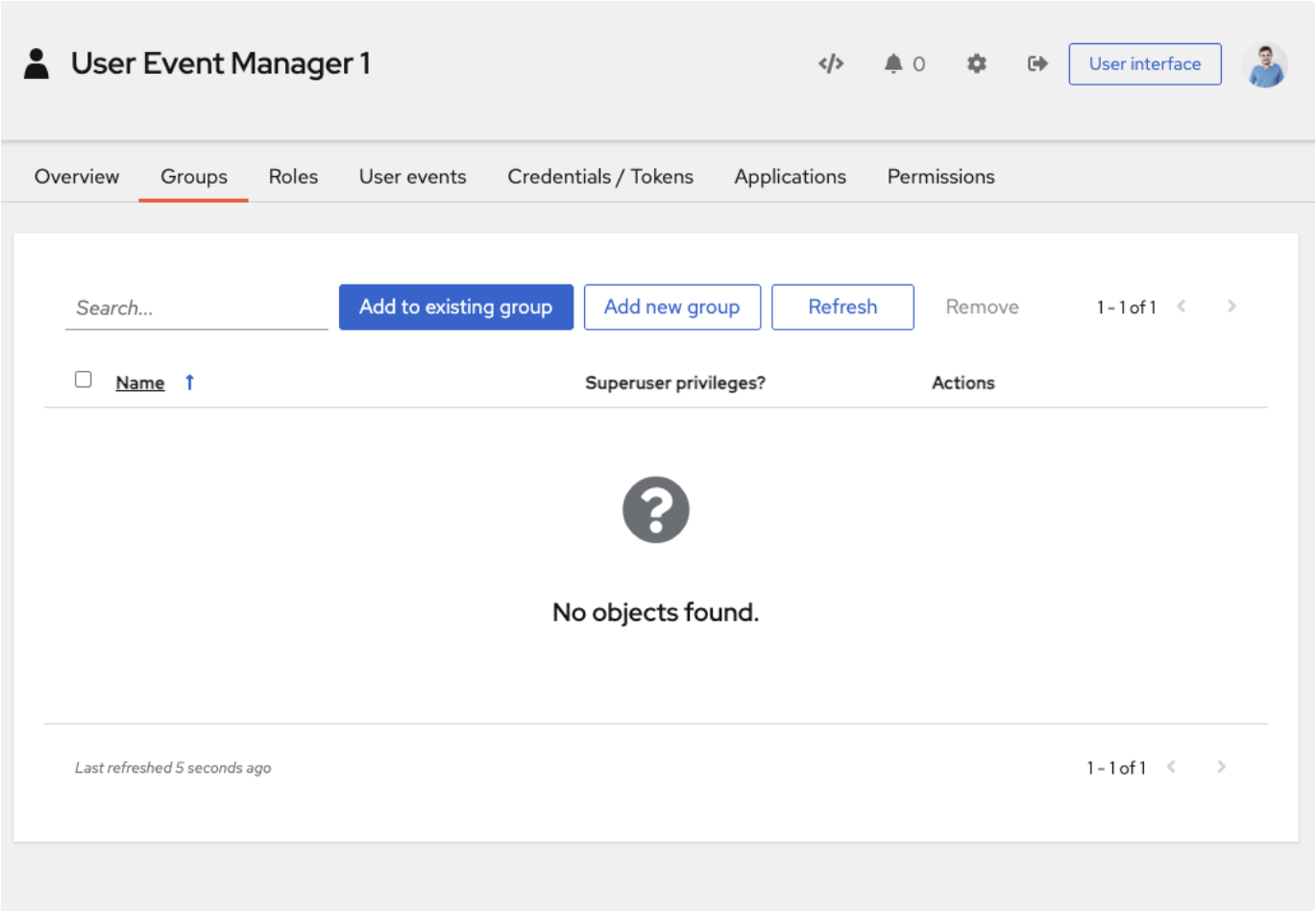
1. User created.



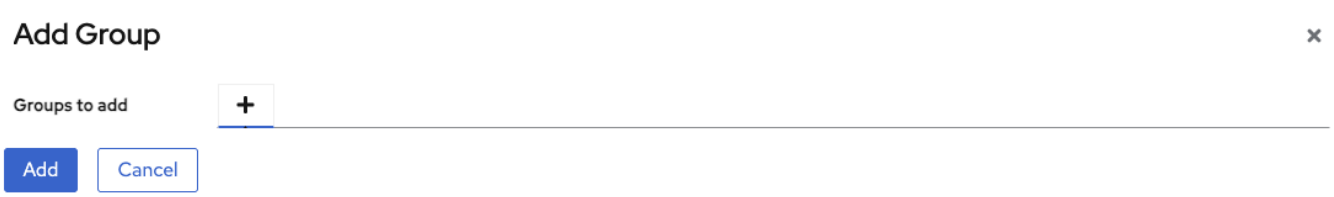
1. Open the user card.



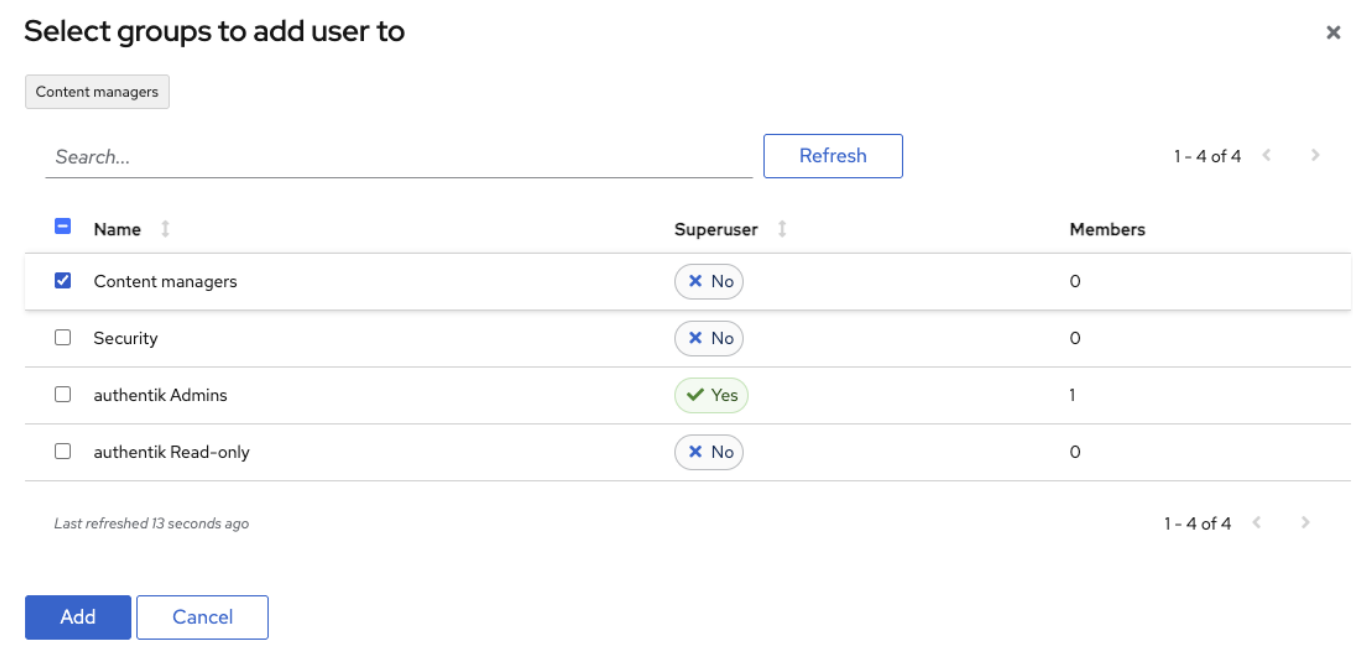
1. Open the "Groups" tab.



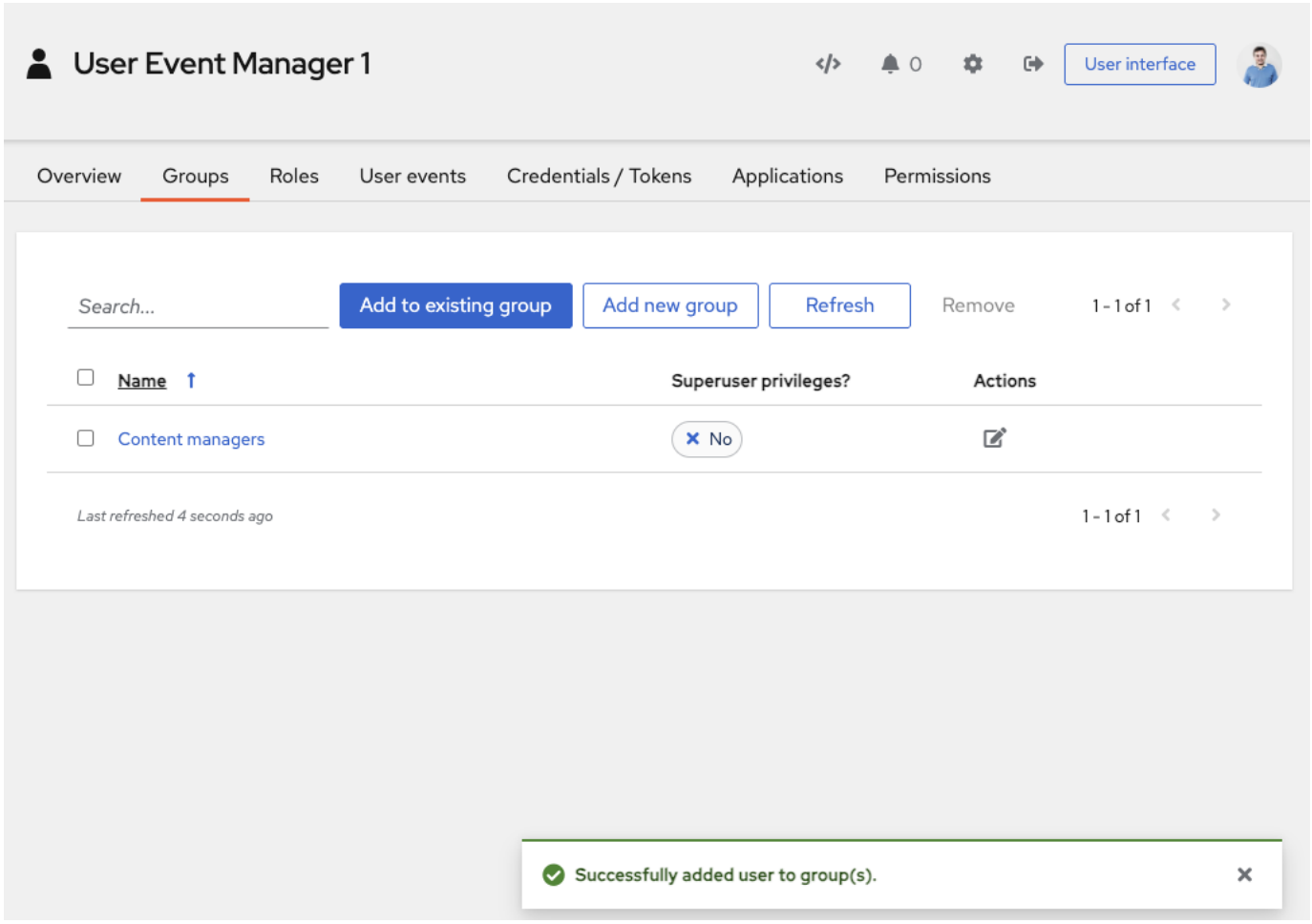
1. Add a group.



1. Pick the existing "Content managers" group.



1. Save and return to the user's Groups tab.



1. Finally set the user's password.

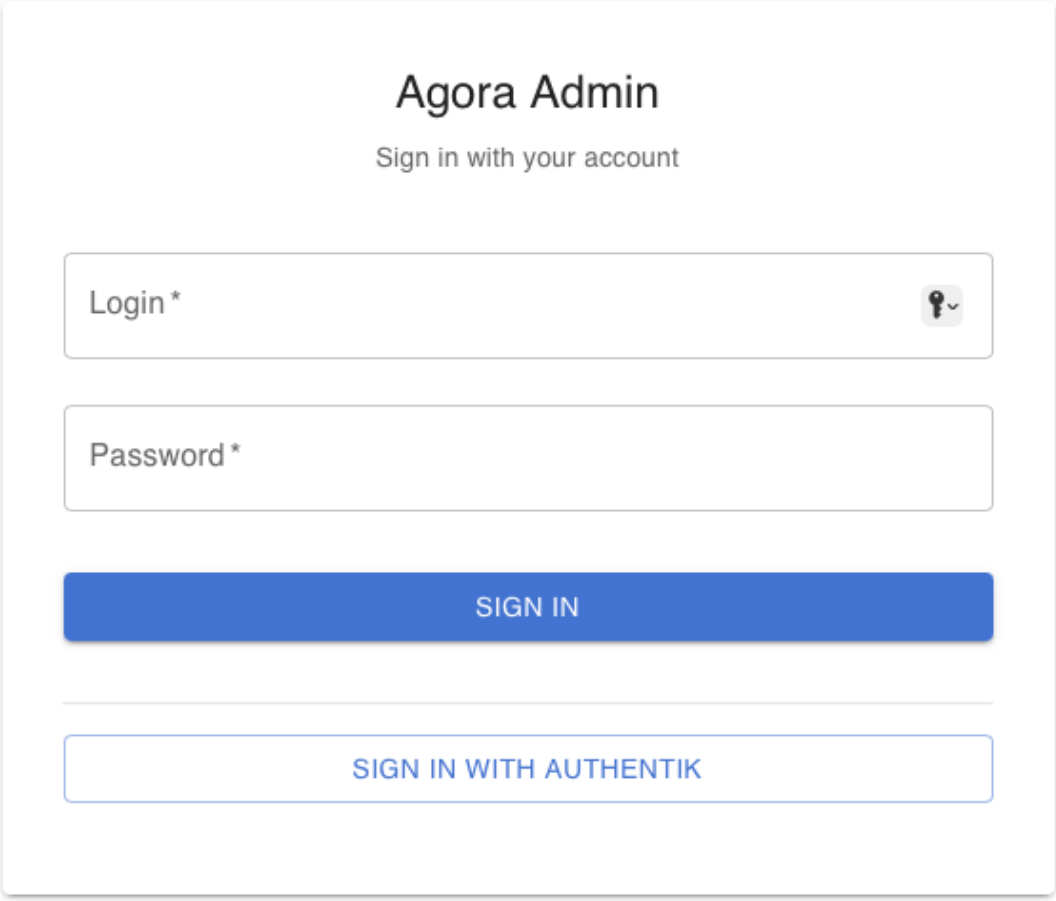
1.2. CONFIGURE AGORA

Set these environment variables so Agora can use OIDC with the IdP:

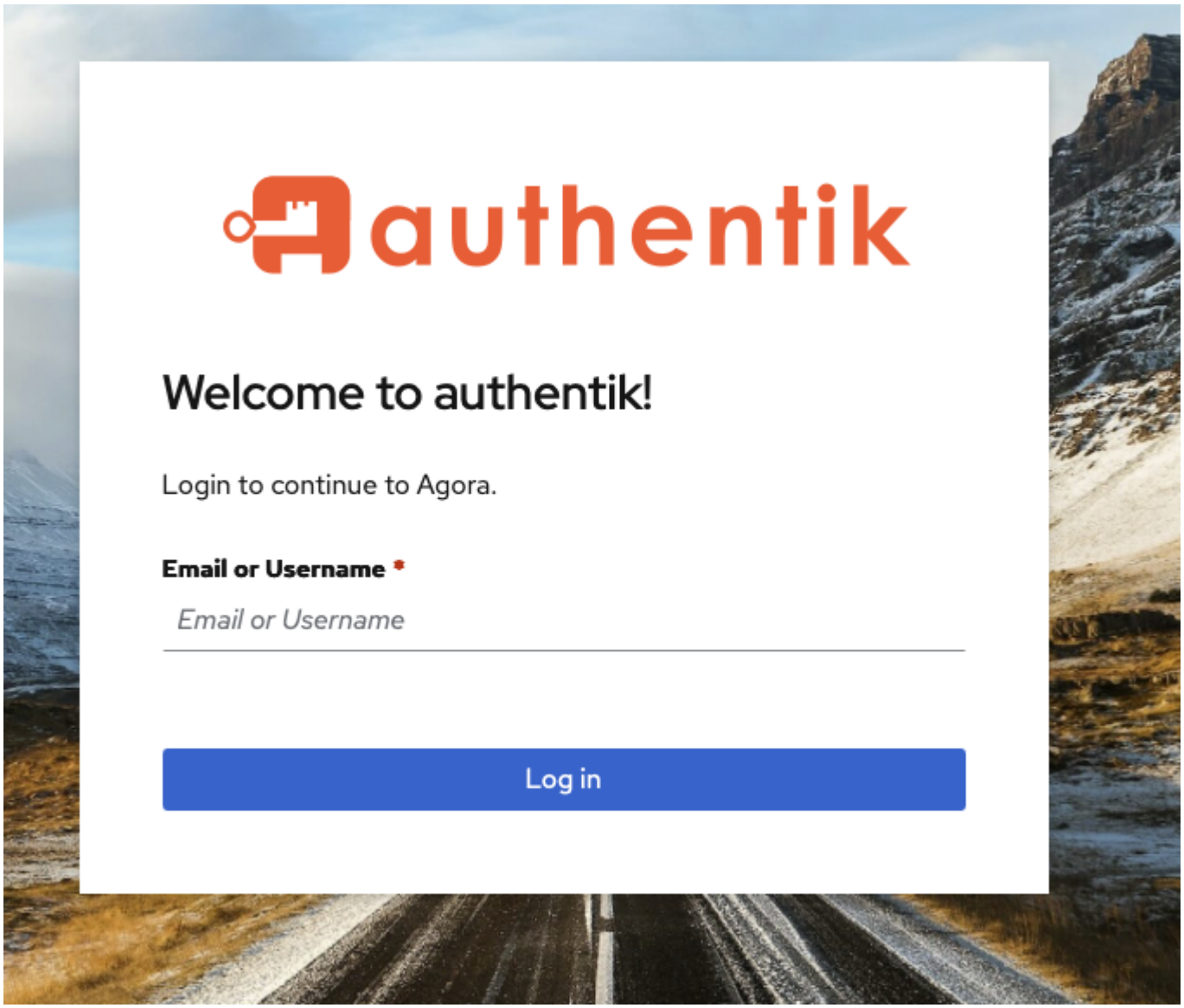
- `OIDC_AUDIENCE=agora-admin-api` — expected **`aud`** claim in the IdP JWT
- `OIDC_CLIENT_ID=agora-admin-api` — effectively the application login registered with the IdP
- `OIDC_CLIENT_SECRET=oCSarud3XmFrYyjVl....ko1VW8n2O3SJLHfwCoTiy` — the application secret from the IdP
- `OIDC_SCOPES="openid profile offline_access"` — optional; these are Agora defaults. `offline_access` allows refresh tokens
- `OIDC_ISSUER=http://authentik.local:9000/application/o/agora/` — your server URL, e.g. the Authentik from above. The `agora` segment is Authentik's convention: the application slug is part of the issuer path
- `OIDC_TITLE=Authentik` — label shown to users; OIDC stays off without this option
- `OIDC_ROLE_CLAIM=groups` — optional; which JWT field carries group membership
- `OIDC_ROLE_MAP='{"authentik Admins":"administrator","Content managers":"content_manager","Sysops":"monitoring"}'` — required mapping from IdP groups to Agora roles (see role list below)
- `LOCAL_LOGIN_ENABLED=false` — optional; disable username/password and use only the IdP

1.3. END-USER STEPS

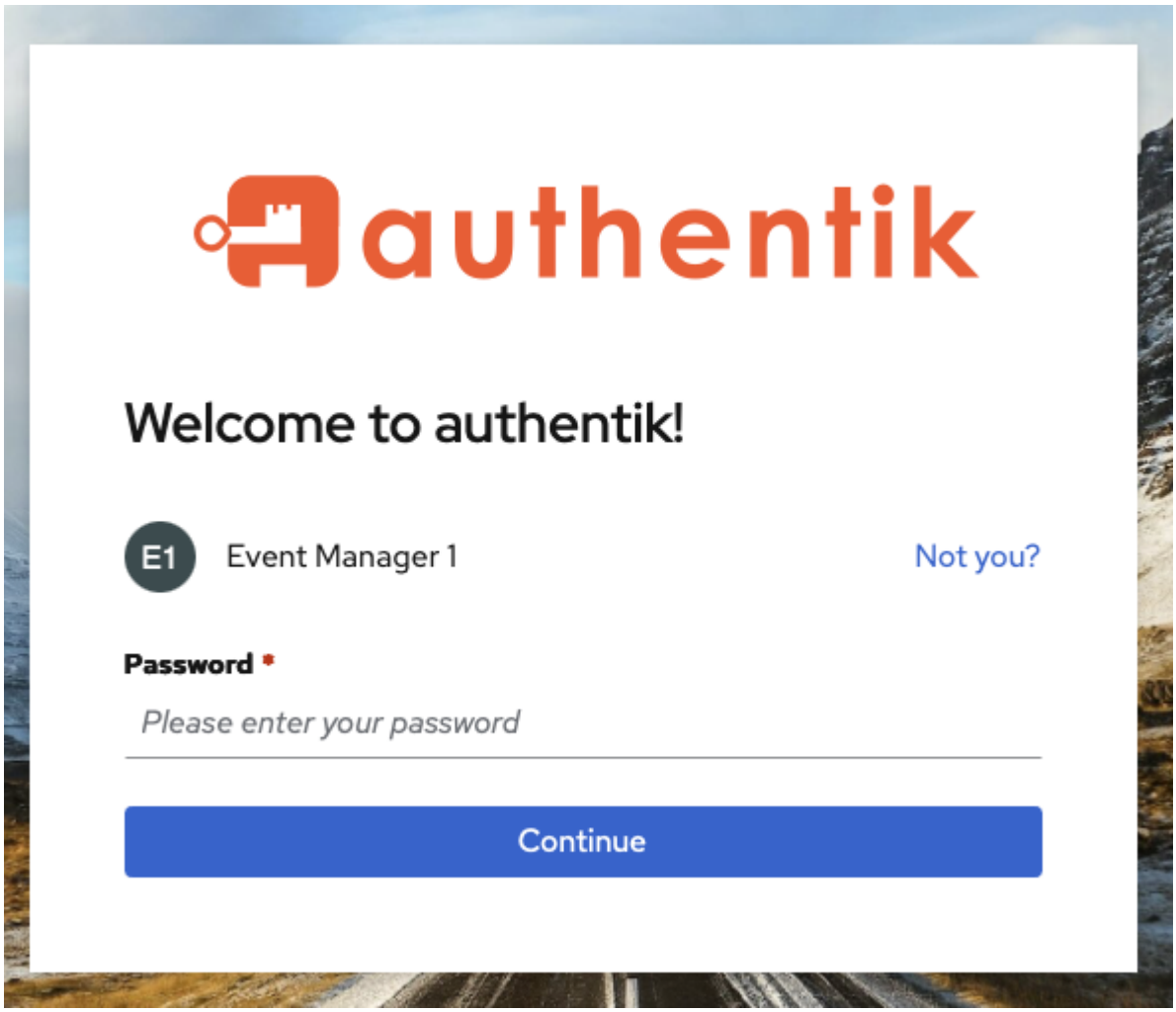
1. On the login screen the user sees local login and OIDC.



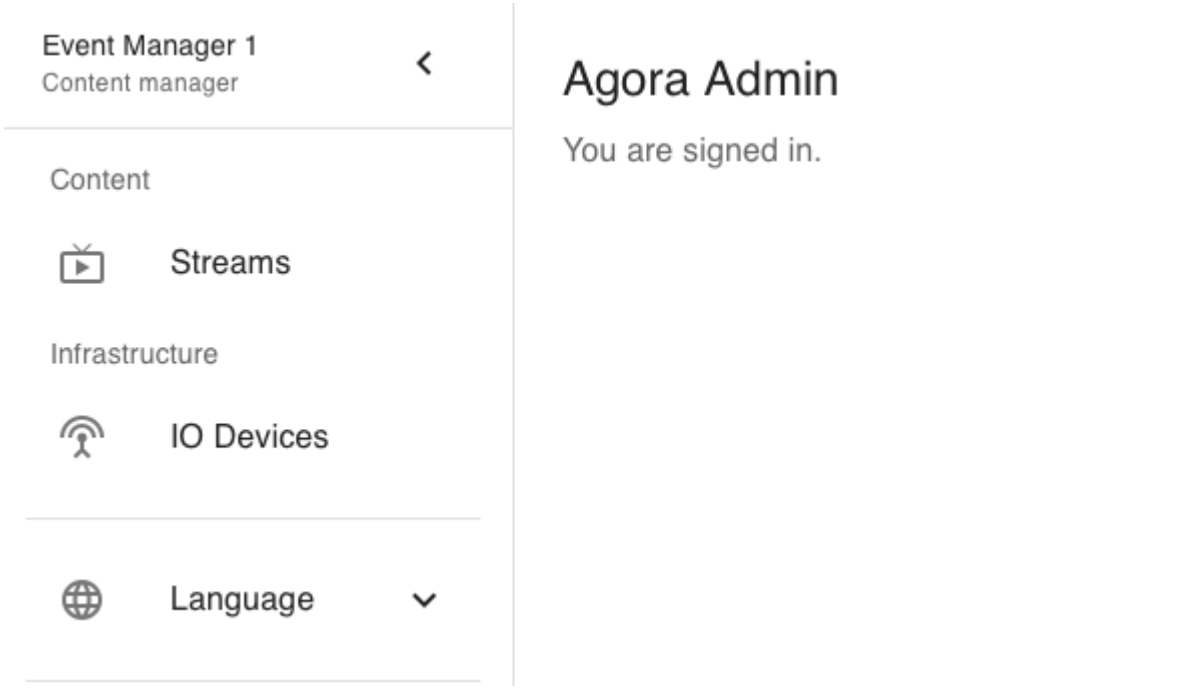
1. They choose IdP login and land in Authentik.



1. They enter login (here "Event Manager 1") and the password screen.



1. After success they return to Agora and are logged into the app.



The user is created in Agora.

2. Configuration details

2.1. IDP ROLES AND MAPPING TO AGORA

Practical steps for admins mapping IdP → Agora roles.

Agora reads roles from the **access token**:

- claim name is set by **`OIDC_ROLE_CLAIM`**;
- string-to-role mapping is set by **`OIDC_ROLE_MAP`**.

Allowed Agora roles:

- **`administrator`**;
- **`content_manager`**;
- **`monitoring`**;
- **`security`**.

If a new user matches nothing in `OIDC_ROLE_MAP`, login ends with `403` and no JIT account is created.

If the account already exists and the new token has no mappable role, existing `Account.role` is unchanged.

Step-by-step

1. Decide which claim carries roles.

Often `groups` (string array), sometimes a dedicated string claim.

1. Configure the IdP so that claim is present in the **access token**.

In Authentik you typically use Scope/Property Mapping on the OAuth2/OpenID Provider; defaults often already expose `groups`.

1. Keep claim values stable and easy to map.

Examples: `agora-administrator`, `agora-content-manager`, or human-readable names like `Content managers`.

1. Set Agora variables.

```
OIDC_ROLE_CLAIM=groups
OIDC_ROLE_MAP='{"agora-administrator":"administrator","agora-content-manager":"content_manager","agora-monitoring":"monitoring","agora-security":"security"}'
```

1. Restart Agora, log in via IdP, verify.

Verification and debugging

Agora logs include the decoded token:


```
{"level":"info","token_endpoint":"http://localhost:9000/application/o/token/","has_refresh_token":true,"expires_in":300,"access_token_payload":"{\"iss\":
\"http://localhost:9000/application/o/agora/\",\"sub\":\"4b34f838bad570b486a2168b5353666fc19a77a230f9df03aff8c1bf7c1baf77\",\"aud\":\"agora-admin-api\",
\"exp\":1775037446,\"iat\":1775037146,\"auth_time\":1775037103,\"acr\":\"goauthentik.io/providers/oauth2/default\",\"amr\":[\"pwd\"],\"sid\":
\"1799899e3648369f24a47c987310fd96a5c5678dce14bbdcf61ce67c4e61b430\",\"jti\":\"oUgCDmUr9jtxiBn2zN4N0ExsZV4rptZ8eUuajmOy\",\"name\":\"\",\"given_name\":\"\",
\"preferred_username\":\"Event Manager 1\",\"nickname\":\"Event Manager 1\",\"groups\":[\"Content managers\"],\"azp\":\"agora-admin-api\",\"uid\":
\"yfgs9bVO0Lg2Okc3c0HdDjN5OxxGG1SlHen4b1l4\",\"scope\":\"openid profile offline_access\"}","time":1775037146,"message":"oidc: token response received"}
{"level":"info","jwks_url":"http://localhost:9000/application/o/agora/jwks/","keys":1,"ttl":3600000,"time":1775037146,"message":"oidc: jwks cache refreshed"}
{"level":"info","issuer":"http://localhost:9000/application/o/
agora/","subject":"4b34f838bad570b486a2168b5353666fc19a77a230f9df03aff8c1bf7c1baf77","jti":"oUgCDmUr9jtxiBn2zN4N0ExsZV4rptZ8eUuajmOy","exp":1775037446,"time":
1775037146,"message":"oidc: access token verified"}
```


Check what arrives in **`groups`**.

Backend logs:

- mapping issues: `oidc login: no mappable role for JIT user`;
- success: `POST /login/oidc` and account create/login.

For shell configs keep `OIDC_ROLE_MAP` as one quoted JSON string (as above) so `source .env` does not break it.

1.6. SESSIONS IN AGORA AND REVOCATION

Agora follows a **BFF (Backend for Frontend)** pattern: the browser does not store or send the IdP JWT to the Agora API. The browser uses Agora's opaque session token; the backend holds IdP access/refresh tokens.

This matches IETF guidance for browser apps: OAuth 2.0 for Browser-Based Applications (BFF).

Agora `session_id` for OIDC login is initialized from the first access token's `jti` and stays stable for that local session.

- `GET /sessions`, `GET /sessions/{id}` — `{id}` is `jti`.
- `session_logout` — ends session by `jti`.

OIDC logout closes the local Agora session. If the IdP exposes `revocation_endpoint`, the backend also revokes the refresh token (best effort).

After session close, Agora returns **401** for that session token.

1.7. VERIFICATION AFTER IDP LINKING

1. `OIDC_ISSUER`, `OIDC_AUDIENCE`, and JWKS match the real access token.
2. OIDC login in the UI succeeds.
3. JIT: new `sub` plus mappable role → account in MongoDB with correct role; without mappable role → **403**, no account.
4. Session revocation by `jti`.

2. Access token fields used by the Agora backend

Below are **IdP access token** fields the Agora backend validates and uses in OIDC flows. The browser does **not** send the IdP JWT to Agora APIs.

Claim / aspect	Purpose for Agora
`iss`	Must match `OIDC_ISSUER`.
`aud`	Must include `agora-admin-api` (string or array element).
`sub`	Stable user id at the IdP; stored as `external_account_id` in MongoDB.
`jti`	**Required**; session id in the API and audit (`session_id`). Missing `jti` → **401**.
`exp`	Expiry; validated with **120 s leeway** (clock skew).
`nbf`	Optional "not before"; same **120 s leeway** when present.
`iat`	Issued-at; auxiliary use.
Claim from `OIDC_ROLE_CLAIM`	String or string array (e.g. groups); mapped via `OIDC_ROLE_MAP`.
`preferred_username`, `email`	Help choose **login** on JIT (with `sub`).

Agora also uses the JWT header `kid` to pick JWKS keys and cache them (`OIDC_JWKS_CACHE_TTL`; JWKS reload on signature failure depends on your build).

4. Glossary

- **OIDC** — OpenID Connect. Login and **JWT** issuance without the app querying the directory on every user action.
- **JWT** — JSON Web Token. Signed payload (**claims**): who the user is, expiry, audience. Readable content; cannot be forged without the key.
- **IdP** — Identity Provider. Central directory holding users and access policies; issues tokens after login.
- **Claim** — one named field inside a JWT (`sub`, `aud`, `groups`, etc.).
- **Access token** — IdP JWT the Agora backend uses to validate identity and roles. The browser does not send it to Agora APIs.
- **ID token** — JWT about the browser login event. **Not** used for Agora admin APIs (different `aud` and purpose).
- **JWKS** — JSON Web Key Set. IdP public keys at a URL; Agora verifies access token signatures with them.

- **RS256** — RSA-SHA256 JWT signing. Typical for IdP access tokens.
- **OAuth2 / OpenID Provider** — IdP component that issues **access** (and optionally **ID**) tokens over OAuth2/OIDC.
- **Discovery** — JSON at `{issuer}/.well-known/openid-configuration`: `issuer`, `jwks_uri`, and other endpoints.
- **Scope** — rights/data the **client requests** at authorization; controls which **claims** appear in the token.
- **Audience (`aud`)** — which application the JWT is for. Agora expects `agora-admin-api` in the access token (or in the `aud` list).
- **Bearer** — HTTP scheme `Authorization: Bearer <token>`. Here the browser sends Agora's session Bearer token, not the IdP JWT.
- **JIT** (Just-In-Time) — first successful Agora request with a valid IdP token **creates** the DB account if missing **and** a **mappable role** exists; otherwise **403** and no account.
- **Property Mapping** (Authentik) — rule for fields added to the **access token** (e.g. groups); without it groups may be missing from the JWT.
- `iss` **(issuer)** — who issued the JWT (IdP URL). Agora picks **JWKS** from `iss` for signature verification.
- `sub` **(subject)** — stable user id at the **IdP**; stored as `external_account_id` in Agora.
- `jti` **(JWT ID)** — unique id for an IdP access token issuance. Agora uses the first successful `jti` as stable `session_id` for OIDC sessions.
- `exp` **/** `nbf` — token validity window (and optional not-before).
- **SSO** — single sign-on: user authenticates once to the **IdP**; apps consume issued tokens.
- **End session** — IdP URL to end the browser session ("sign out everywhere"), unlike clearing only client-side tokens.

2.4.3 Security log

The audit log in Agora tracks administrative actions, supports incident response, and enables later security analysis. This section shows the history of operations users performed in the system.

Why the audit log matters

The audit log reconstructs who did what and when, supporting investigations, internal controls, and security reviews.

What the audit log shows

Аудит лог

Время	Сессия	Аккаунт	Действие	ID объекта	Детали
18/03/2026, 17:28:15	sChV5UbZ2gAA.	root	Удаление потока	e1	—
18/03/2026, 17:23:59	sChV5UbZ2gAA.	root	Сохранение потока	e1	—
18/03/2026, 16:25:59	sChV5UbZ2gAA.	root	Удаление потока	ort	—
18/03/2026, 14:42:54	sCh5Uv_Pk0AA.	root	Вход	sCh5Uv_Pk0AA.	—
18/03/2026, 10:43:31	sChV5UbZ2gAA.	root	Сохранение потока	hdmi0	—
18/03/2026, 00:08:54	sChV5UbZ2gAA.	root	Сохранение потока	hdmi0	—
17/03/2026, 16:49:59	sChV5UbZ2gAA.	root	Сохранение стримера	sales-sl.e	—
17/03/2026, 16:48:22	sChV5UbZ2gAA.	root	Удаление стримера	rChuy5CY5wAA.	—
17/03/2026, 16:48:18	sChV5UbZ2gAA.	root	Удаление стримера	rChuxzCa5wAA.	—
17/03/2026, 16:48:14	sChV5UbZ2gAA.	root	Удаление стримера	rChbgt-srkAA.	—
17/03/2026, 15:44:08	sChV5UbZ2gAA.	root	Сохранение стримера	sales-sl.e	—
17/03/2026, 15:35:40	sChV5UbZ2gAA.	root	Сохранение стримера	sales-sl.e	—
17/03/2026, 15:35:32	sChV5UbZ2gAA.	root	Сохранение стримера	gigabyte-2u.e	—
17/03/2026, 15:35:17	sChV5UbZ2gAA.	root	Сохранение стримера	gigabyte-2u.e	—
17/03/2026, 15:10:32	sChV5UbZ2gAA.	root	Сохранение потока	hdmi0	—
17/03/2026, 13:51:24	sChV5UbZ2gAA.	root	Сохранение потока	ort	—

Each audit event shows:

- operation time;
- session ID;
- account that performed the action;
- event type;
- target object;
- extra data in `payload`.

When linked objects exist, the UI can jump to:

- the session card;
- the user card;
- the stream;
- the streamer.

That speeds up moving from a log line to full context.

Search and filter controls

Controls include:

- date or time range;
- filter by event types.

Use them to:

- narrow to a period;
- focus on relevant operation classes;
- separate user actions, configuration changes, and security events.

Event types

Logged classes include:

- user login and logout;
- stream create, update, delete;
- streamer create, update, delete;
- account operations including lock.

The set may grow as the system and admin API evolve.

Paginated log loading

The log loads in chunks. When there are many rows, the UI loads additional pages sequentially.

That supports:

- large event volumes;
- lower UI load;
- comfortable sequential review.

Relation to sessions

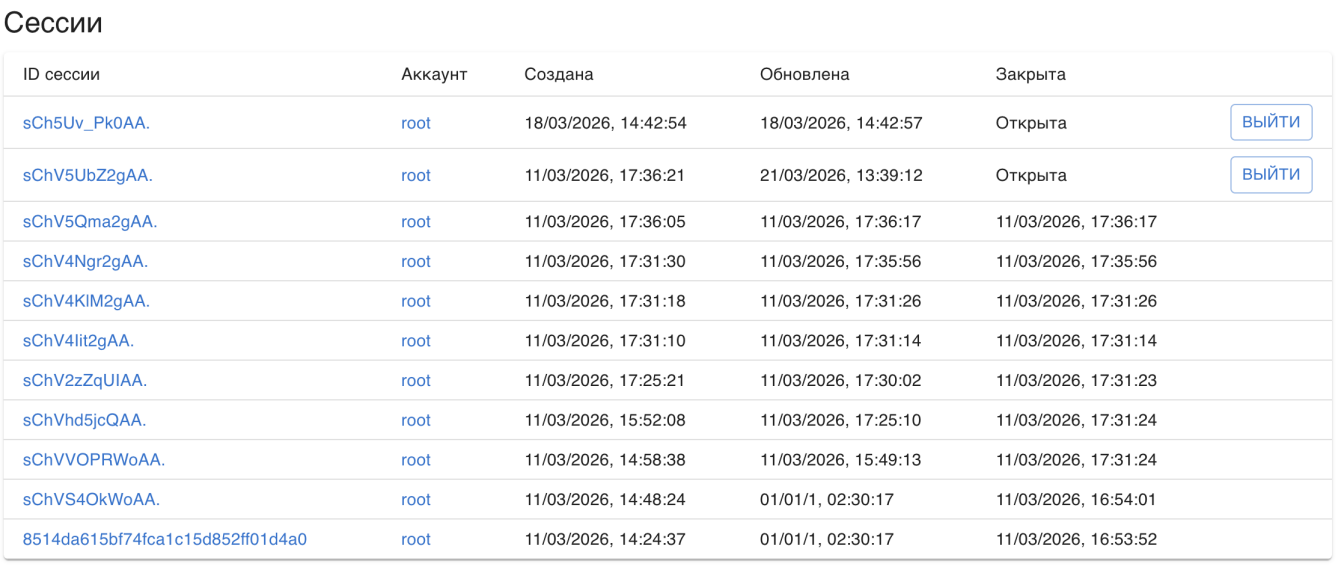
The audit log links closely to user sessions. Each session can show related operations, and from the log you can open a session to continue analysis.

That helps with suspicious activity, admin change reviews, and per-login user behavior.

2.4.4 User sessions in Agora

Sessions in Agora support access control, activity monitoring, and security investigations. In the sessions section an administrator or security user can view active and completed sessions and force-logout open sessions when needed.

Session list



The list shows:

- session ID;
- owning user;
- session creation time;
- last update time;
- close time or indication that the session is still open.

From the list you can:

- open a session card;
- open the owning user's card;
- force-logout an open session.

If a session is already closed, logout is not available.

Terminating an open session

To force-terminate an active session:

1. Find the session in the list.
2. Click `logout`.
3. Wait for success confirmation.

After termination the session is no longer open and the list updates.

Further API calls with that session token are rejected.

Session card

The session card shows:

- `session_id`;
- associated user;
- creation time;
- last update time;
- close time or open status.

From the card you can return to the list or open the related user.

Audit events for a session

Each session lists related audit events so you can:

- see actions performed in that session;
- tie operations to a specific user;
- review the sequence of actions within one login.

The session page supports both live session oversight and incident investigation.

IP address collection

For security control Agora records IP addresses for user connections and sessions.

Uses include:

- analyzing login origin;
- investigating suspicious activity;
- correlating user actions with a network address;
- later audit and incident response.

2.4.5 Network interactions

Agora is designed for corporate networks with separated component roles and segments. A typical deployment separates flows so that:

- administrative traffic is isolated from media traffic;
- external sources only talk to the `Ingress server`;
- internal master nodes are not directly reachable from user networks;
- content delivery to viewers uses `edge` servers.

Main network segments

Typical segments include:

- administrative segment for `controller` access;
- internal media segment for `origin`, `vod transcoder`, streamers, and relays;
- DMZ for the `Ingress server` when external live sources are accepted;
- user or branch segment where `edge` servers and clients sit;
- storage segment for the `storage` system.

This layout reduces direct exposure of internal media servers and aligns with corporate security policy.

Primary traffic flows

CONTROLLER AND STREAMERS / RELAYS

The `controller` talks to streamers and relays over `HTTP` or `HTTPS`.

Common ports:

- `80` for plain `HTTP`;
- `443` for `HTTPS`.

This traffic carries management, status, configuration, and monitoring.

BETWEEN STREAMERS

Streamer-to-streamer traffic may include:

- `HTTP` / `HTTPS` for control and publish interactions;
- `multicast` for `Standby Push`.

For `Standby Push` the backup streamer receives `multicast` from the primary. That expects a LAN with reliably low latency.

BETWEEN RELAYS AND STREAMERS

Relays pull video from streamers over:

- `HTTP`;
- `HTTPS`.

Depending on resilience design, a relay may connect to one or several streamers.

INGRESS SERVER AND EXTERNAL SOURCES

The `Ingress server` accepts external sources over the protocols used in your ingest design, commonly:

- `HTTP`;
- `HTTPS`;
- `RTMP` on port `1935`;
- `SRT` on agreed ports;
- `WebRTC` with a fixed port set when policy requires it.

For `SRT` and `WebRTC`, port ranges are defined during rollout and must be agreed with the customer network team.

INGRESS SERVER AND INTERNAL STREAMERS

When `Ingress server` and internal streamers are separate, they use:

- `HTTP`;
- `HTTPS`.

This path should be allowed only between trusted internal nodes because it moves the stream from DMZ into the internal media segment.

Typical firewall requirements

Network policies usually need to:

- allow admin access to the `controller` only from the admin segment;
- allow external video ingest only to the `Ingress server`;
- block direct external access to `origin` and other internal media servers;
- allow internal connectivity between streamers, relays, and `edge` servers only on agreed protocols;
- restrict `storage` access to trusted platform components.

Practical notes

When designing connectivity, remember:

- `Twincast` needs two independent ingest paths;
- `Standby Push` needs low-latency LAN and `multicast` between streamers;
- for isolated ingest of external sources, place the `Ingress server` in DMZ;
- allowed ports and protocols should be recorded in the deployment design doc.