Identity Provider: Centralized Authentication Foundations and SSO Federation

Identity Management and Single Sign-On (SSO)

Managing scattered credentials across disparate applications is a primary driver of operational overhead and credential-stuffing vulnerability. Fragmented password policies often lead to insecure workarounds, including password reuse across heterogeneous environments.

Deploying a **Single Sign-On (SSO)** architecture addresses this problem at its core: by delegating authentication to an authoritative Identity Provider (IdP), peripheral services never handle user passwords directly, but instead rely on cryptographically signed tokens issued by the IdP.

Authentik

Authentik is an open-source, enterprise-grade Identity & Access Management (IAM) platform. It provides authentication brokering via OpenID Connect (OIDC), SAML, and LDAP, alongside forward-auth integration for edge reverse proxies.

The following guide details the deployment of Authentik via Docker, configured behind an Nginx reverse proxy.

Step 1: Preparing the Environment

Once logged into *antares* via your preferred method:

```
# mkdir /opt/authentik
# cd /opt/authentik
```

Let's start with a nice `docker-compose.yml` whose content will be:

```
services:
  postgresql:
    env_file:
    - .env
    environment:
      POSTGRES_DB: ${PG_DB:-authentik}
      POSTGRES_PASSWORD: ${PG_PASS:?database password required}
      POSTGRES_USER: ${PG_USER:-authentik}
    healthcheck:
      interval: 30s
      retries: 5
      start_period: 20s
      test:
      - CMD-SHELL
      - pg_isready -d $${POSTGRES_DB} -U $${POSTGRES_USER}
      timeout: 5s
    image: docker.io/library/postgres:16-alpine
    restart: unless-stopped
    volumes:
      #- database:/var/lib/postgresql/data
    - ./database:/var/lib/postgresql/data
  server:
    command: server
    depends_on:
      postgresql:
        condition: service_healthy
    env_file:
    - .env
    environment:
      AUTHENTIK_POSTGRESQL__HOST: postgresql
      AUTHENTIK_POSTGRESQL__NAME: ${PG_DB:-authentik}
      AUTHENTIK_POSTGRESQL__PASSWORD: ${PG_PASS}
```

```
      AUTHENTIK_POSTGRESQL__USER: ${PG_USER:-authentik}
      AUTHENTIK_SECRET_KEY: ${AUTHENTIK_SECRET_KEY:?secret key required}
    image: ${AUTHENTIK_IMAGE:-ghcr.io/goauthentik/server}:${AUTHENTIK_TAG:-2025.10.2}
    ports:
    - ${COMPOSE_PORT_HTTP:-9000}:9000
    - ${COMPOSE_PORT_HTTPS:-9443}:9443
    restart: unless-stopped
    volumes:
    - ./media:/media
    - ./custom-templates:/templates
  worker:
    command: worker
    depends_on:
      postgresql:
        condition: service_healthy
    env_file:
    - .env
    environment:
      AUTHENTIK_POSTGRESQL__HOST: postgresql
      AUTHENTIK_POSTGRESQL__NAME: ${PG_DB:-authentik}
      AUTHENTIK_POSTGRESQL__PASSWORD: ${PG_PASS}
      AUTHENTIK_POSTGRESQL__USER: ${PG_USER:-authentik}
      AUTHENTIK_SECRET_KEY: ${AUTHENTIK_SECRET_KEY:?secret key required}
    image: ${AUTHENTIK_IMAGE:-ghcr.io/goauthentik/server}:${AUTHENTIK_TAG:-2025.10.2}
    restart: unless-stopped
    user: root
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ./media:/media
    - ./certs:/certs
    - ./custom-templates:/templates
```

Now, one thing needs to be said. Named volumes have always disgusted me, so even though the official documentation uses them, I converted them into relative volumes. Also because this way, when things are quiet, you can just make a nice tarball and the backup is done. `${AUTHENTIK_TAG:-2025.10.2}` simply means that if the variable is not defined, it defaults to 2025.10.2. The keen eye will notice that we need a `.env` file with environment variables.

```
# echo "PG_PASS=$(openssl rand -base64 36 | tr -d '\n')" >> .env
# echo "AUTHENTIK_SECRET_KEY=$(openssl rand -base64 60 | tr -d '\n')" >> .env
# echo "AUTHENTIK_TAG=2026.2" >> .env
# echo "AUTHENTIK_ERROR_REPORTING__ENABLED=true" >> .env
```

Okay, now let's stop for a moment. Authentik listens internally on ports 9000/tcp and 9443/tcp, and if we look at the tags we inserted in the docker-compose.yaml, we see they are also the default mappings. If we don't like them and want to expose them on other ports, it's sufficient to:

```
# echo "COMPOSE_PORT_HTTPS=443" >> .env
# echo "COMPOSE_PORT_HTTP=80" >> .env
```

This is fine if we want to expose Authentik directly to the Internet, but it is my personal opinion that this solution is quite obscene. That's why I leave 9000 and 9443, and our fantastic reverse proxy made with `nginx` will connect to these.

```
# echo
"AUTHENTIK_LISTEN__TRUSTED_PROXY_CIDRS=100.64.0.0/10,127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16,fe80::/10,::1/128,1
>> .env
```

That series of variables tells us which reverse proxies are accepted (or rather, what their IP addresses are). I have no idea where you want to put yours. Mine is on another machine to which I'm connected via `Netbird`, which is why the first one in the list is `100.64.0.0/10`. You just need to adapt the list to your needs.

That said, you can proceed with `docker compose pull ; docker compose up -d ; docker compose logs -f`.

Step 2: akadmin and errno0x0d

We have reached the moment to configure the administrator user: **akadmin**. To do this, we need to go to `http://[ip-del-server]/if/flow/initial-setup/`.

Initially, we will face a nice blank screen. Let's click on `Admin interface` and realize a couple of things:

- There are no other users: maybe it's time to create a normal one
- Users cannot self-register. Are we okay with that? We don't like it? It depends on the use case. If we want to populate the users ourselves, we are fine. If we want users to register autonomously (maybe with administrator approval), then things need to change a bit.

Anyway, let's keep one thing in mind:

- users can belong to groups
- users / user groups can authenticate on applications
- applications must have a provider.

Suppose we have a service like `Discourse`. We want whoever authenticates on Authentik to have direct access to the forum. The steps are:

- Create users who have access to the forum
- optional: ensure that these users are part of the group of those authorized to access the forum
- create a provider that identifies the forum (here *client id* and *secret key* will be provided to configure inside `Discourse`)
- create an application associated with the above provider
- associate the application with the group of those who, if authenticated, can use the forum.

First of all, it makes me sick to think of having to use `akadmin`, so I will create another user to whom I will eventually delegate admin status.

Admin Interface --> Directory --> Users --> New User (yes, the translation is quite faulty). For a basic setup, it's enough to fill in the fields:

- Username: errno0x0d
- Display Name: Permission Denied
- User Type: Internal
- Email address: admin@example.org <-- Email is not mandatory, but half the services want it, so better to put it
- Path: users

And of course *Active* must be checked. Let's create the user.

Clicking on the newly created user takes you to the overview. We already have a feeling of what "Set password" is for, so let's set it. Once that's done, at the top, next to *Overview* is *Groups*. Click on it and add the user to the existing group *Authentik Admins*. Going back to the overview, if we still see "Superuser" set to *No*, we should perform a very delicate operation: refresh the browser page.

Step 3: group creation

Since we don't want to miss anything, let's see how to create groups, so we can differentiate who can access which application. From the left sidebar menu of Authentik, *Directory* --> *Groups* --> *New Group*, because in this case too, the Italian language does not have concepts like "New Group", which is why they left it in English.

- Group name: Allow-Discourse <-- Put whatever you want, as long as you remember why you created it

That's it. Go ahead and click *Create Group*.

Clicking on the name of the newly created group, at the top you have the *Overview* just like with users. To its immediate right are the *Users* you can add. In this case too, click *Add existing user* and put good old *errno0x0d* there.

Step 4: provider creation

Okay, we have the user, we have the group. What do we do? Install *Discourse* somewhere (which obviously I have no intention of doing now, since it's just an example) and configure it to use Authentik. To do this, we need to create a provider: from the left menu *Applications* --> *Providers* --> *Create*.

For our application *Discourse*, we will use the *OAuth2/OpenID Provider*:

- Provider Name: Discourse-Provider <-- Or whatever you like
- Authorization flow: default-provider-authorization-explicit-consent (Authorize Application)
- Client type: Confidential (not everyone wants a secret, but generally yes: it depends on the application)
- Client ID and Client Secret: write these down somewhere because `Discourse` needs these keys
- Redirect URIs/Origins (Regex): this also depends on the application, we should check the manual for the one we are configuring OIDC for. But one thing must be said: if you don't set it, Authentik will accept the first URL redirected by the calling application, so feel free to leave it blank
- Signing Key: authentik Self-signed Certificate <-- Leave this for now, it's how tokens are signed
- Advanced protocol settings: here you can set how often the client must re-authenticate, which is subjective. Regarding scopes, unless there are special cases, the defaults are fine. Now, an important thing: **Subject mode**: I highly recommend using **Based on the user's UUID** because if you leave it on hashed ID and a user does DIRTY things (infamy, slander, etc.), you won't quickly figure out who it was to ban them.

The rest is fine as it is.

Step 5: application creation

From the left menu: *Applications* --> *Applications* --> Create (not with provider, we already created the provider):

- Application name: Discourse
- Slug: discourse <-- how it's identified internally
- Group: can be empty
- Provider: Discourse-provider
- Policy engine mode: ANY

Go ahead and create.

If you want to associate it with a group, click on it, the usual *Overview* appears, and to its right is *Policy/Group/User Binding*. Clicking on it, we can select "Bind existing Policy/Group/User", then we can select whether to bind to a Policy, a group, or a user: choosing the middle one, click on the *Group* field and select *Allow-Discourse*.

From this moment on, when a user wants to authenticate on the site hosting `Discourse`, they are redirected to Authentik: this will check their credentials; if they are valid, it will check the group, and if that also allows access to `Discourse`, they will gain entry.

> ***Warning**: often registrations on applications and authentications fail because the user hasn't configured an email. Always populate that field.*

Step 6: enabling self-registration

If we want patrons to be able to self-register in our tavern, albeit with activation conditioned on our blessing (which happens by activating the user under *Directory -> Users*), we need to create the registration flow. But even before that, we need to modify the login flow.

From the left menu *Flows and Stages --> Stages* and let's create a new one:

- Type: Prompt Stage
- Name: Stage - Data Registration
- Stage-specific settings: select only the following fields:
  - default-user-settings-field-email
  - default-user-settings-field-name
  - default-user-settings-field-username
  - initial-setup-field-password

    - initial-setup-field-password-repeat
- Validation policies: for now delete everything (these are for validation policies like complex passwords, etc.)

Now we need to create another stage, namely the one to request TOTP. So, as just done, let's create a new one:

- Type: TOTP Authenticator Setup Stage
- Name: Mandatory TOTP Setup
- Authenticator type name: Dreadful TOTP Auth
- Digits: 6 digits
- Setup flow: default-authenticator-totp-setup (default-authenticator-totp-setup)

And finally, let's create the stage for user registration with approval (i.e. disabled by default):

- Type: User Write Stage
- Name: Write Stage - Inactive
- Create users when required *enabled*
- Create users as inactive *enabled*

The rest can remain as is.

Let's move to flows.

From the left menu *Flows and Stages --> Flows* and let's create a new flow:

- Name: Registration Flow
- Title: Registration
- Slug: registration-flow
- Designation: Enrollment
- Authentication: No authentication required

The rest can be left as default. Click on the newly created flow to get to the overview. At the top choose *Stage Binding* and proceed with *Bind Existing Stage*, because the translation here is also stellar:

- Stage: Stage - Data Registration (the one we created earlier)
- Order: 10

Finish and now bind the next stage: again "Bind existing stage":

- Stage: Write Stage - Inactive
- Order: 20

And lastly the TOTP stage, same procedure:

- Stage: Mandatory TOTP Setup
- Order: 30

Finishing blow: let's go back to the stages section from the left menu *Flows and Stages --> Stages* and modify *default-authentication-identification*. Scroll down to the end of *Flow settings --> Enrollment flow* and set it to *Registration-flow (Registration Flow)*.

This way, whoever accesses the main Authentik page can register (mandatory with TOTP) and will need our approval to proceed.

Step 7: do we want to continue with port 9000?

The question is rhetorical and the answer is: **NO**.

We have several options, but the main two are: either we serve everything behind a *Cloudflare Tunnel*, which would be good to avoid exposing our IP address, or we use a reverse proxy on one of our machines. Yes, exactly, we can also use the same one where we installed Authentik, but you can understand on your own that it's better to have application frontends on different hosts than the core. Anyway, assuming we have the domain *auth.example.org* dedicated to publishing Authentik, the reverse proxy configuration with `nginx` will be:

```
server {
    listen 80;
    server_name auth.example.org;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name auth.example.org;

    ssl_certificate /etc/letsencrypt/live/auth.example.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/auth.example.org/privkey.pem;

    # Additional security for the IdP
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;
    # Authentik handles uploads of icons/logos and heavy flows
    client_max_body_size 50M;

    location / {
        # Forward to Antares on port 9443 via NetBird DNS
        proxy_pass https://node01.netbird.example.org:9443;

        # Bypass SSL verification (like on Cloudflare)
        proxy_ssl_verify off;
        proxy_ssl_server_name on;

        # Fundamental headers for authentication
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # WebSocket for the Authentik admin interface
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Generous timeouts to prevent errors during complex login flows
        proxy_read_timeout 900;
        proxy_connect_timeout 900;
        proxy_send_timeout 900;
    }
}
```

In this specific case you will notice that host `antares` is identified as *node01.netbird.example.org*: as you might have understood, I use Netbird for connectivity; you put the address with which the reverse proxy identifies the host running Authentik (if they are on the same machine, 127.0.0.1 fits nicely). Remember, of course, to put the IP address with which the reverse proxy presents itself to antares in your `.env` as seen in the first steps:

```
AUTHENTIK_LISTEN__TRUSTED_PROXY_CIDRS=100.64.0.0/10,127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16,fe80::/10,::1/128,19
```

[May the big mole be with you]

[EOF]