Identity Provider: Fondamenti di autenticazione centralizzata e federazione SSO

Gestione delle identità e Single Sign-On (SSO)

La gestione distribuita delle credenziali tra molteplici applicazioni eterogenee è storicamente una delle maggiori cause di vulnerabilità informatica e inefficienza operativa. L'adozione di policy di complessità isolate spesso incentiva pratiche insicure, come il riutilizzo delle password tra ambienti diversi o la persistenza di credenziali amministrative deboli.

L'adozione di un'architettura **Single Sign-On (SSO)** risolve tale complessità alla radice: delegando l'autenticazione a un Identity Provider (IdP) centralizzato e autoritativo, i servizi applicativi periferici non trattano direttamente le credenziali degli utenti, ma si fidano dei token di sessione firmati crittograficamente emessi dall'IdP.

Authentik

Authentik è una piattaforma aperta e completa di Identity & Access Management (IAM). Funziona come broker di autenticazione per applicazioni moderne (via OpenID Connect, SAML e LDAP) e può proteggere reverse proxy di frontiera tramite Forward Auth.

Di seguito viene descritta la procedura di implementazione pratica di Authentik su base Docker, configurata per operare dietro un reverse proxy Nginx.

Passo 1: la preparazione dell'ambiente

Una volta entrati su *antares* via il vostro metodo preferito

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

Iniziamo con un bel `docker-compose.yml` il cui contenuto sarà:

```
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
```

Ora c'è da dire una cosa. A me i *named volumes* hanno sempre suscitato un grande ribrezzo, per cui sebbene la documentazione ufficiale li usi, io li ho trasformati in volumi relativi. Anche perché così, a palle ferme, si fa un bel tarball e il backup è fatto. `${AUTHENTIK_TAG:-2025.10.2}` significa semplicemente che se non viene definita la variabile, di default vale 2025.10.2. L'occhio attento noterà che necessitiamo di un file `.env` con le variabili di ambiente.

```
# 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
```

Ok ora fermiamoci un attimo. Authentik ascolta internamente sulle porte 9000/tcp e 9443/tcp e se vediamo i tag che abbiamo inserito nel docker-compose.yaml vediamo che sono anche le mappature di default. Se non ci piacciono e vogliamo esporle su altre porte è sufficiente:

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

Questo è buono se vogliamo esporre direttamente Authentik su Internet, ma è mia personale opinione che come soluzione sia abbastanza oscena. Ecco perché io lascio la 9000 e la 9443 e a queste si connetterà il nostro fantastico reverse proxy fatto con `nginx`.

```
# echo
"AUTHENTIK_LISTEN__TRUSTED_PROXY_CIDRS=100.64.0.0/10,127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,19
>> .env
```

Quella serie di variabili ci dice quali sono i reverse proxy accetati (o meglio quali sono i loro indirizzi IP). Io non ho idea di dove lo vogliate mettere il vostro. Il mio sta su un'altra macchina con cui sono connesso via `Netbird`, ecco perché il primo della lista è `100.64.0.0/10` Dovete semplicemente adattare la lista alle vostre esigenze.

Detto questo si può procedere con un `docker compose pull ; docker compose up -d ; docker compose logs -f`.

Passo 2: akadmin e errno0x0d

Siamo giunti al momento di configurare l'utente amministratore: **akadmin**. Per farlo dobbiamo andare su `http://[ip-del-server]/if/flow/initial-setup/`.

Inizialmente ci troveremo davanti a una bella schermata vuota. Clicchiamo su `Interfaccia amministrativa` e rendiamoci conto di un paio di cose

- Non ci sono altri utenti: forse è il caso di crearne uno normale
- gli utenti non possono autoregistrarsi. Ci va bene? Non ci piace? Dipende dal tipo di utilizzo. Se vogliamo essere noi a popolare gli utenti, allora siamo a posto così. Se vogliamo che gli utenti si registrino in autonomia (magari con approvazione da parte di un amministratore) allora le cose vanno cambiate un poco.

Comunque sia teniamo conto di una cosa:

- gli utenti possono appartenere a gruppi
- gli utenti / i gruppi di utenti possono autenticarsi sulle applicazioni
- le applicazioni devono avere un provider.

Supponiamo di avere un servizio come `Discourse`. Vogliamo che chi si autentica su Authentik abbia accesso diretto al forum. Gli step sono:

- Creare gli utenti che hanno accesso al forum
- opzionale: far sì che tali utenti facciano parte del gruppo di coloro che sono abilitati ad accedere al forum
- creare un provider che identifichi il forum (qui verranno forniti *cliend id* e *secret key* da configurare dentro `Discourse`)
- creare un'applicazione associata al provider di cui sopra
- associare l'applicazione al gruppo di coloro che, se autenticati, possono usare il forum.

Anzitutto mi fa onco pensare di dover usare `akadmin`, per cui creo un altro utente a cui eventualmente delegherò lo status di admin.

Interfaccia amministrativa --> Cartella --> Utenti --> New User (sì la traduzione è abbastanza fallace). Per un funzionamento di base è sufficiente compilare i campi

- Username: errno0x0d
- Display Name: Permission Denied
- Tipo di utente: Interno
- Email address: admin@example.org <-- La mail non è obbligatoria, ma mezzi servizi la vogliono quindi conviene metterla
- Percorso: users

E ovviamente *Attivo* deve essere flaggato. Creiamo l'utente.

Cliccando sul nuovo utente appena creato si è sulla panoramica. *Imposta password* abbiamo già il sentore di a che cosa serva, per cui impostiamola. Fatto questo, in alto, accanto a *Panoramica* c'è *Gruppi*. Clicchiamoci sopra e aggiungiamo l'utente al gruppo esistente *Authentik Admins*. Tornando sulla panoramica, qualora vedessimo ancora "Superutente" valorizzato con *No*, dovremmo effettuare un'operazione molto delicata: aggiornare la pagina del browser.

Passo 3: la creazione del gruppo

Siccome non vogliamo farci mancare niente, vediamo come creare i gruppi, in modo poi da differenziare chi può accedere a quali applicativi. Dal menù laterale a sinistra di Authentik, *Cartella --> Gruppi --> New Group*, perché anche in questo caso la lingua italiana non prevede concetti come "Nuovo Gruppo", motivo per cui l'hanno lasciato in inglese.

- Group name: Allow-Discourse <-- Metteteci quello che vi pare, basta che vi ricordiate per che cosa lo avete creato

Fine. Cliccate pure su *Crea Gruppo*.

Cliccando sul nome del gruppo appena creato, in alto avete la *Panoramica* proprio come nel caso degli utenti. Alla sua immediata destra ci sono gli *Utenti* che potete aggiungere. Anche in questo caso *Aggiungi utente esistente* e mettoamoci il buon vecchio caro *errno0x0d*.

Passo 4: la creazione del provider

Ok abbiamo l'utente, abbiamo il gruppo. Che cosa facciamo? Installiamo da qualche parte *Discourse* (cosa che ovviamente non ho la minima intenzione di fare adesso, visto che è solo un esempio) e configuriamolo per fargli usare Authentik. Per farlo è necessario creare un provider: sempre dal menù a sinistra *Appicazioni --> Providers --> Crea*.

Per il nostro applicativo *Discourse* utilizzeremo il *Fornitore Oauth2/ OpenID*

- Provider Name: Discourse-Provider <-- O come vi pare
- Flusso di autorizzazione: default-provider-authorization-explicit-consent (Authorize Application)
- Tipologia client: Confidenziale (non tutti vogliono anche il secret, ma generalmente sì: dipende dall'applicativo)
- Client ID e Client Secret: appuntiamoli da qualche parte perché `Discourse` ha bisogno di queste chiavi
- Reindirizzamento URI/Orifini (Regex): anche questo dipende da applicazione ad applicazione, dobbiamo consultare il manuale di quella per cui stiamo configurando OIDC. Ma c'è da dire una cosa: se non la mettete, sarà Authentik a prendere per buona la prima

URL che gli viene reindirizzata dall'applicazione chiamante, quindi lasciatela pure vuota
- Chiave di firma: authentik Self-signed Certificate <-- Per il momento lasciamo questo, è come vengono firmati i token
- Impostazioni avanzate del protocollo: qui potete impostare ogni quanto il client si deve riautenticare, ed è soggettivo. Per quanto riguarda gli ambiti, salvo casi particolari, quelli di default vanno bene. Ora una cosa importante: **Modalità soggetto** consiglio vivamente di utilizzare **Basato sull'UUID dell'utente** perché se lasciate quello sull'hashed ID, in caso un utente faccia cose BRUTTE (infamie, calunnie, etc), non capirete velocemente chi è stato per inibirlo.

Il resto va bene così.

Passo 5: la creazione dell'applicazione

Sempre dal menù a sinistra: *Applicazioni* --> *Applicazioni* --> Crea (non con fornitore, lo abbiamo già creato il provider):

- Application name: Discourse
- Slug: discourse <-- è come viene identificato internamente
- Gruppo: anche vuoto
- Provider: Discourse-provider
- Modalità motore criteri: ANY

Create pure

Se volete associarlo ad un gruppo, ci cliccate sopra, compare la solita *Panoramica*, e alla sua destra c'è *Associazione Criterio/Gruppo/Utente*. Cliccandoci sopra possiamo selezionare "Associa esistente Criterio/Gruppo/Utente", possiamo selezionare quindi se associare ad un Criterio, ad un gruppo o ad un utente: scegliendo quello di mezzo, clichiamo sul campo *Gruppo* e selezioniamo *Allow-Discourse*.

Da questo momento, quando un utente si vuole autenticare sul sito su cui è ospitato `Discourse`, viene indirizzato ad Authentik: questo controllerà le credenziali, se sono valide controllerà il gruppo e, se anche questo permette l'accesso a `Discourse`, allora avrà accesso.

> ***Attenzione**: spesso le registrazioni sugli applicativi e le autenticazioni vanno buca perché l'utente non ha configurato una mail. Valorizzate sempre quel campo.*

Passo 6: permettiamo il *self register*

Se vogliamo che gli avventori possano auto registrarsi nella nostra locanda, seppur con attivazione condizionata alla nostra benedizione (che avviene attivando l'utente da *Cartella -> Utenti*), dobbiamo creare il flusso di registrazione. Ma prima ancora bisogna modificare il flusso di login.

Dal menù a sinistra *Flussi e fasi --> Fasi* e creiamone una nuova:

- Tipo: Fase Richiesta
- Nome: Fase - Registrazione dati
- Impostazione specifiche per la fase: selezionare solo i seguenti campi:
  - default-user-settings-field-email
  - default-user-settings-field-name
  - default-user-settings-field-username
  - initial-setup-field-password
  - initial-setup-field-password-repeat
- Criteri di validazione: per il momento cancelliamo tutto (sono per i criteri di validazione come password complesse etc)

Adesso dobbiamo creare un'altra fase, ovvero quella per la richiesta del TOTP. Quindi, come appena fatto, procediamo con il crearne una nuova:

- Tipo: Fase di configurazione dell'autenticatore TOTP
- Nome: Setup TOTP Obbligatorio
- Nome del tipo di autenticatore: Dreadful TOTP Auth
- Cifre: 6 cifre
- Flusso di configurazione: default-authenticator-totp-setup (default-authenticator-totp-setup)

E per finire creiamo la fase per la registrazione utente con approvazione (ovvero di default inattiva):

- Tipo: Fase di scrittura dell'utente
- Nome: Stage Scrittura - Inattivo

- Creare utenti quando richiesto
- Creare utenti come inattivi *abilitato*

Il resto può restare così com'è.

Spostiamoci sui flussi.

Dal menù a sinistra *Flussi e fasi --> Flussi* e creiamo un nuovo flusso:

- Nome: Registration Flow
- Titolo: Registrazione
- Slug: registration-flow
- Designazione: Iscrizione
- Autenticazione: Non richiedere autenticazione

Il resto può essere lasciato come di default. Clicchiamo quindi sul flusso appena creato, arrivando alla panoramica. In alto scegliamo *Associazione Fase* e proseguiamo con *Bind Existing Stage*, perché anche in questo caso la traduzione fa faville:

- Fase: Fase - Registrazione dati (quella che abbiamo creato prima)
- Ordine: 10

Cocludiamo e associamo adesso la fase successiva: di nuovo "Bind existing stage*

- Fase: Stage Scrittura - Inattivo
- Ordine: 20

E per ultima la fase di TOTP, stesso procedimento:

- Fase: Setup TOTP Obbligatorio
- Ordine: 30

Colpo di grazia: torniamo nella sezione delle fasi dal menù a sinistra *Fussi e fasi --> Fasi* e modifichiamo il *default-authentication-identification*. Scorriamo fino al termine di *Impostazioni flusso --> Flusso di iscrizione* valorizzandolo con *Registration-flow (Registration Flow).

In questo modo chi accede alla pagina principale di Authentik, può iscriversi (obbligatoriamente con TOTP) e necessiterà della nostra approvazione per proseguire.

Passo 7: vogliamo continuare con la porta 9000?

La domanda è retorica e la risposta è: **NO**.

Abbiamo più scelte, ma le principali sono due: o serviamo tutto dietro *Cloudflare Tunnel*, che sarebbe anche buona come cosa per evitare di esporre il nostro indirizzo, oppure usiamo un reverse proxy su una nostra macchina. Sì esatto possiamo usare anche la stessa su cui abbiamo installato Authentik, ma capite da soli che è meglio avere i front end delle applicazioni su host diversi rispetto al core. Ad ogni modo, supponendo di avere il dominio *auth.example.org* adibito alla pubblicazione di Authentik, il reverse proxy fatto con `nginx` sarà:

```
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;

    # Sicurezza aggiuntiva per l'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 gestisce upload di icone/loghi e flussi pesanti
    client_max_body_size 50M;

    location / {
        # Inoltro verso Antares sulla porta 9443 tramite il DNS di NetBird
        proxy_pass https://node01.netbird.example.org:9443;

        # Bypass verifica SSL (come su Cloudflare)
        proxy_ssl_verify off;
        proxy_ssl_server_name on;

        # Header fondamentali per l'autenticazione
```

```
        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 per l'interfaccia amministrativa di Authentik
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Timeout generosi per evitare errori durante i flussi di login
complessi
        proxy_read_timeout 900;
        proxy_connect_timeout 900;
        proxy_send_timeout 900;
    }
}
```

In questo caso specifico noterete che l'host `antares` è identificato con *node01.netbird.example.org*: come avrete capito io utilizzo Netbird per la connettività; voi mettete l'indirizzo con cui il reverse proxy identifica l'host che ha Authentik (se sono sulla stessa macchina il 127.0.0.1 ci sta bene). Ricordate, ovviamente, di mettere l'indirizzo IP con cui il reverse proxy si presenta ad antares nel vostro `.env` come abbiamo visto nei primi passi:

```
AUTHENTIK_LISTEN__TRUSTED_PROXY_CIDRS=100.64.0.0/10,127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,19
```

[Che il talpone sia con voi]

[EOF]

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).