# Matrix: Server Synapse

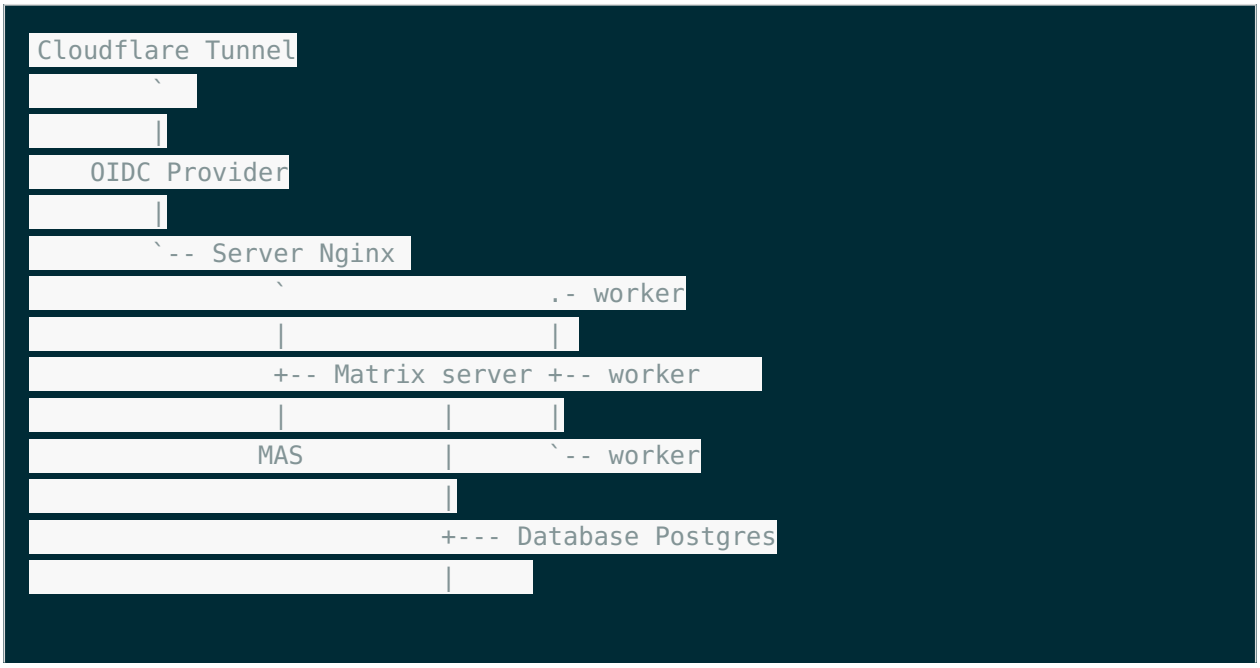
# Introduzione e Principi Architetturali

Il server **Synapse** rappresenta l'implementazione di riferimento per il protocollo aperto **Matrix**, standard de facto per la messaggistica istantanea federata con crittografia end-to-end (E2EE). In un contesto federato, la riservatezza delle comunicazioni è garantita dalla crittografia del payload, mentre il self-hosting dell'istanza garantisce la piena sovranità sui metadati di transito, le politiche di retention dei messaggi e i log applicativi.

> ***Nota bene**: **Ambito operativo e alta disponibilità**: La presente guida illustra la configurazione standard di un'istanza Synapse multi-worker su singolo host/nodo, evidenziando il disaccoppiamento dei carichi di sync, federation e autenticazione. Per scenari di livello enterprise ad alta disponibilità (HA), l'architettura può essere scalata duplicando i nodi applicativi, adottando un cluster PostgreSQL in replica/pooling (es. PgBouncer/Patroni) e segregando i worker su macchine distinte.*

### Tassonomia

Il tutto sarà così fatto



```
                                |
                                `--- Bridges
```

- Provider OIDC: gestisce la registrazione degli utenti
- MAS, aka Matrix Authentication Service: è il connettore tra OIDC e Matrix
- Worker: servono per bilanciare i carichi per scalabilità
- Server Nginx: è il bilanciatore ed è importantissimo configurarlo per smistare il carico tra i vari worker
- Bridges: sono per l'interoperabilità con altri protocolli (come Whatsapp, Signal, Telegram, ...)

**NOTA**: utilizzeremo lo SPLIT domain. Significa che

- il server Synapse sarà in ascolto su : *matrix.example.org*
- l'handle sarà: *user:example.org* e non *user:matrix.example.org*
- Quando si trova qualcosa di **[HIDDEN]**, significa che o è stato configurato dall'applicazione o è stato generato con `openssl rand -hex 32`.

**ATTENZIONE**: il provider OIDC dovete averlo già installato! Come fare è già stato affrontato, quindi non coprirò l'argomento in questa pagina. In alternativa potete utilizzare il MAS come OIDC provider a tutti gli effetti per il vostro server Synapse abilitando le registrazioni.

# Installazione di Synapse: la base da cui tutto parte

Iniziamo con il creare la directory di lavoro

```
sudo mkdir -p /opt/matrix && cd opt/matrix
```

### Generazione del file di configurazione

La prima configurazione richiede il comando *generate*:

```
docker run -it --rm \
-v $(pwd)/data:/data \
-e SYNAPSE_SERVER_NAME=example.org \
```

```
-e SYNAPSE_REPORT_STATS=no \
ghcr.io/element-hq/synapse:develop generate
```

Questo comando genererà il file di configurazione sotto `data/homeserver.yaml`, e sarà il file che dovrà essere modificato per aggiungere la `public_baseurl`.

```
server_name: "example.org"
public_baseurl: "matrix.example.org" # <-- Da aggiungere
pid_file: /data/homeserver.pid
listeners:
  - port: 8008
    resources:
    - compress: false
      names:
      - client
      - federation
    tls: false
    type: http
    x_forwarded: true
database:
  name: sqlite3
  args:
    database: /data/homeserver.db
log_config: "/data/example.org.log.config"
media_store_path: /data/media_store
registration_shared_secret: "[HIDDEN]"
report_stats: false
macaroon_secret_key: "[HIDDEN]"
form_secret: "[HIDDEN]"
signing_key_path: "/data/example.org.signing.key"
trusted_key_servers:
  - server_name: "matrix.org"
```

**Creazione docker network**

Utilizzeremo una docker network appositamente per Synapse:

```
docker network create synapse
be636df73d51c5275918e9f8f07fbc748d0da6b4ddd250a2cba8c408a1f609ab
```

**Il Docker Compose File**

Vista la complessità del dispiegamento che stiamo intentando, è il caso di creare un docker compose file. Sarà poi modificato nel tempo per costruire l'implementazione finale.

Partiamo con l'environment: `.env` (password del DB generata con openssl);

```
# Configurazione Synapse
SYNAPSE_CONFIG_PATH=/data/homeserver.yaml
SYNAPSE_SERVER_NAME=example.org
SYNAPSE_REPORT_STATS=yes

VIRTUAL_HOST=example.org
VIRTUAL_PORT=8008

# Configurazione Database
POSTGRES_USER=synapse
POSTGRES_DB=synapse
POSTGRES_PASSWORD=[HIDDEN]
```

Il `docker-compose.yml` invece sarà

```
services:
  synapse:
    image: ghcr.io/element-hq/synapse:develop
    restart: unless-stopped
    networks:
      - synapse
    environment:
      - SYNAPSE_CONFIG_PATH=${SYNAPSE_CONFIG_PATH}
      - VIRTUAL_HOST=${VIRTUAL_HOST}
      - VIRTUAL_PORT=${VIRTUAL_PORT}
      - SYNAPSE_SERVER_NAME=${SYNAPSE_SERVER_NAME}
      - SYNAPSE_REPORT_STATS=${SYNAPSE_REPORT_STATS}
    volumes:
      - ./data:/data
    ports:
      - 8008:8008/tcp
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
```

```
  redis:
    image: redis:7-alpine
    restart: unless-stopped
    networks:
      - synapse
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
      timeout: 3s
      retries: 5
    volumes:
      - ./redis:/data

  db:
    image: docker.io/postgres:16-alpine
    restart: unless-stopped
    networks:
      - synapse
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_INITDB_ARGS=--encoding=UTF-8 --lc-collate=C --lc-ctype=C
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 5s
      timeout: 5s
      retries: 5
    volumes:
      - ./postgres:/var/lib/postgresql/data

networks:
  synapse:
    external: true
```

Niente docker compose up ancora: dobbiamo dire a synapse che deve utilizzare *postgres* e non *sqlite*. Dobbiamo editare `data/homeserver.yml`, sostituendo

```
database:
  name: sqlite3
  args:
    database: /data/homeserver.db
```

con

```
database:
    name: psycopg2
    args:
        user: synapse
        password: [HIDDEN]
        host: db
        database: synapse
        cp_min: 5
        cp_max: 10
```

Ovviamente la password del DB è quella generata con openssl, e che è stata scritta nel file *.env*. Solo a questo punto si può procedere con il prossimo:

```
docker compose up -d; docker compose logs -f
```

È quindi utilizzabile? No per niente, ma almeno vediamo che non ci siano errori.

**Reverse proxy con nginx (e i balletti con cloudflare)**

Ora c'è un distinguo:

- matrix.example.org
- example.org Mentre il primo si occupa solo di Synapse, il secondo deve contenere i well-known per permettere la gestione degli handle. Quindi mentre il primo può essere containerizzato qui, il secondo dovrà essere modificato a mano perché si presuppone che sia altrove, a servire direttamente il dominio, per esempio. Di conseguenza dentro questo container andremo a gestire unicamente `matrix.example.org`.

Essendo dietro Cloudflare, è necessario andare a mettere le mani direttamente sul pannell di amministrazione del dominio.

Nella gestione del dominio, a destra: `SSL/TLS -> Server di Origine`. Io utilizzo la chiave `ECC` e lascio `*.example.org` e `example.org`.

Torniamo sulla nostra macchina lasciando la pagina coi certificati visibile:

```
mkdir -p nginx/conf.d
mkdir nginx/certs
```

Salviamo quindi la chiave privata e quella pubblica in:

- nginx/certs/origin.key
- nginx/certs/origin.crt

Nuovamente sulla pagina di Cloudflare, andiamo nella sezione Zero Trust e creiamo un connettore docker: segnamoci il `TUNNEL_TOKEN`

Popoliamo quindi `nginx/conf.d/matrix.example.org.conf`

```
server {
    listen 443 ssl http2;
    server_name matrix.example.org;

    # Usiamo i certificati Origin CA di Cloudflare
    ssl_certificate /etc/nginx/certs/origin.crt;
    ssl_certificate_key /etc/nginx/certs/origin.key;

    # Ottimizzazioni per la sicurezza
    ssl_protocols TLSv1.2 TLSv1.3;

    client_max_body_size 100M;

    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Forwarded-Proto https;

    location / {
        proxy_pass http://synapse:8008;
        proxy_read_timeout 600s;
    }

    # Federazione Matrix
    location /.well-known/matrix/server {
        return 200 '{"m.server": "matrix.example.org:443"}';
        add_header Content-Type application/json;
        add_header Access-Control-Allow-Origin *;
    }

    location /.well-known/matrix/client {
        return 200 '{"m.homeserver": {"base_url": "https://
matrix.example.org"}}';
```

```
        add_header Content-Type application/json;
        add_header Access-Control-Allow-Origin *;
    }
}
```

Modifichiamo il docker compose file per contenere anche nginx aggiungendo:

```
nginx:
    image: nginx:alpine
    restart: unless-stopped
    networks:
      - synapse
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d:ro
      - ./nginx/certs:/etc/nginx/certs:ro # Montiamo la cartella con
origin.crt e origin.key

  tunnel:
    image: cloudflare/cloudflared:latest
    restart: unless-stopped
    networks:
      - synapse
    command: tunnel run
    environment:
      - TUNNEL_TOKEN=${CLOUDFLARE_TUNNEL_TOKEN}
      - TUNNEL_ORIGIN_CERT_CHECK=false
```

Aggiungiamo anche al file `.env` la parte necessaria per nginx:

```
CLOUDFLARE_TUNNEL_TOKEN=TUNNEL_TOKEN_SEGNATO_PRIMA
```

Facciamo lo startup sia di nginx che del tunnel

```
docker compose up -d nginx tunnel
```

E torniamo nuovamente su cloudflare per configurare il connettore appena creato aggiungendo l'applicazione pubblicata:

- Subdomain: matrix
- Dominio: example.org
- Servizio: https
- Url: nginx:443

- Altre impostazioni di applicazione:
    - TLS
        - Nessuna verifica TLS: on
        - Connessione HTTP2: on

A questo punto visitando `https://matrix.example.org` comparirà la pagina base di Synapse su protocollo Matrix.

Siccome vogliamo gli handle diversi dall'effettivo nome di dominio, è **necessario modificare anche nginx che serve il dominio principale, ovvero example.org**. Nello specifico è necessario aggiungere:

```
location = /.well-known/matrix/server {
    default_type application/json;
    add_header Access-Control-Allow-Origin *;
    return 200 '{"m.server": "matrix.example.org:443"}';
}

location = /.well-known/matrix/client {
    default_type application/json;
    add_header Access-Control-Allow-Origin *;
    return 200 '{"m.homeserver": {"base_url": "https://matrix.example.org"},
"m.identity_server": {"base_url": "https://vector.im"}}';
}
```

Ora possiamo vedere se è tutto correttamente configurato per lo split domain andando su `https://federationtester.matrix.org/` e inserendo `example.org` nel test.

# I worker

Un problema abbastanza devastante è il fatto che se il processo principale fa tutto (sincronizzazione, federazione, scambi, etc) il tutto va in overload ed è lento. È necessario scalare. Ecco perché creiamo più *workers* ognuno dei quali è specializzato in qualcosa e fa solo quello.

Modifichiamo `docker-compose.yml` in modo da contenerli, aggiungendo:

```
  generic-federation-presence:
    image: "ghcr.io/element-hq/synapse:develop"
    container_name: generic-federation-presence
    restart: unless-stopped
```

```
    entrypoint: ["/start.py", "run", "--config-path=/data/homeserver.yaml",
"--config-path=/data/workers/generic-federation-presence.yaml"]
    volumes:
      - ./data:/data
    environment:
        SYNAPSE_WORKER: synapse.app.generic_worker
    depends_on:
      - db
      - redis
      - synapse
    networks:
      - synapse
  generic-sync-1:
    image: "ghcr.io/element-hq/synapse:develop"
    container_name: generic-sync-1
    restart: unless-stopped
    entrypoint: ["/start.py", "run", "--config-path=/data/homeserver.yaml",
"--config-path=/data/workers/generic-sync-1.yaml"]
    volumes:
      - ./data:/data
    environment:
        SYNAPSE_WORKER: synapse.app.generic_worker
    depends_on:
      - db
      - redis
      - synapse
    networks:
      - synapse
  generic-sync-2:
    image: "ghcr.io/element-hq/synapse:develop"
    container_name: generic-sync-2
    restart: unless-stopped
    entrypoint: ["/start.py", "run", "--config-path=/data/homeserver.yaml",
"--config-path=/data/workers/generic-sync-2.yaml"]
    volumes:
      - ./data:/data
    environment:
        SYNAPSE_WORKER: synapse.app.generic_worker
    depends_on:
      - db
      - redis
      - synapse
    networks:
      - synapse
  generic-sync-initial:
    image: "ghcr.io/element-hq/synapse:develop"
    container_name: generic-sync-initial
```

```
    restart: unless-stopped
    entrypoint: ["/start.py", "run", "--config-path=/data/homeserver.yaml",
"--config-path=/data/workers/generic-sync-initial.yaml"]
    volumes:
      - ./data:/data
    environment:
        SYNAPSE_WORKER: synapse.app.generic_worker
    depends_on:
      - db
      - redis
      - synapse
    networks:
      - synapse
  event-persister:
    image: "ghcr.io/element-hq/synapse:develop"
    container_name: event-persister
    restart: unless-stopped
    entrypoint: ["/start.py", "run", "--config-path=/data/homeserver.yaml",
"--config-path=/data/workers/event-persister.yaml"]
    volumes:
      - ./data:/data
    environment:
      SYNAPSE_WORKER: synapse.app.generic_worker
    depends_on:
      - synapse
      - db
      - redis
    networks:
      - synapse
```

Per ognuno di questi deve essere creato un worker file all'interno di `data/workers/` quindi

```
mkdir data/workers
```

Al suo interno avremo: `event-persister.yaml`

```
worker_app: synapse.app.generic_worker
worker_name: event-persister

# Listener per la comunicazione interna (Replication)
worker_listeners:
  - port: 8086
    type: http
```

```
    resources:
      - names: [replication]

# Configurazione Redis (deve puntare al nome del servizio nel compose)
redis:
  enabled: true
  host: redis
  port: 6379
```

poi `generic-federation-presence.yaml`

```
worker_app: synapse.app.generic_worker
worker_name: generic-federation-presence
worker_types:
  - federation_sender
  - federation_reader
  - federation_event_auth
  - presence
  - typing
  - to_device

worker_listeners:
  - port: 8083
    type: http
    tls: false
    resources:
      - names: [federation]
        compress: true
read_only: true


redis:
  enabled: true
  host: redis
```

poi `generic-sync-1.yaml`

```
worker_app: synapse.app.generic_worker
worker_name: generic-sync-1

worker_types:
  - sync

worker_listeners:
  - port: 8082
```

```
    type: http
    tls: false
    resources:
      - names: [client]
        compress: true
read_only: true

redis:
  enabled: true
  host: redis
```

e ancora `generic-sync-2.yaml`

```
worker_app: synapse.app.generic_worker
worker_name: generic-sync-2

worker_types:
  - sync

worker_listeners:
  - port: 8084
    type: http
    tls: false
    resources:
      - names: [client]
        compress: true
read_only: true

redis:
  enabled: true
  host: redis
```

e per finire (per ora) `generic-sync-initial.yaml`

```
worker_app: synapse.app.generic_worker
worker_name: generic-sync-initial

worker_types:
  - sync

worker_listeners:
  - port: 8085
    type: http
```

```
    tls: false
    resources:
      - names: [client]
        compress: true
read_only: true

redis:
  enabled: true
  host: redis
```

mettiamo a posto i permessi

```
chown 991:991 data/workers -R
```

e modifichiamo il mapping di `data/homeserver.yml`

```
server_name: "example.org"
public_baseurl: "https://matrix.example.org"
pid_file: /data/homeserver.pid

listeners:
  - port: 8008
    tls: false
    type: http
    x_forwarded: true
    resources:
      - names: [client, federation, replication]
        compress: true

worker_types:
  - frontend_proxy
  - federation_sender

instance_map:
  main:
    host: synapse
    port: 8008
  generic-sync-1:
    host: generic-sync-1
    port: 8082
  generic-federation-presence:
    host: generic-federation-presence
    port: 8083
  generic-sync-2:
```

```
    host: generic-sync-2
    port: 8084
  generic-sync-initial:
    host: generic-sync-initial
    port: 8085
  event-persister:
    host: event-persister
    port: 8086
stream_writers:
  events:
    - event-persister
federation_rr_transactions_per_second: 100
federation_rr_concurrent_transmissions: 50


redis:
  enabled: true
  host: redis
  port: 6379



database:
    name: psycopg2
[...resto della configurazione ...]
```

Tecnicamente adesso possiamo anche proseguire con la configurazione di nginx per bilanciare sui worker, ma non lo faremo ancora. Dobbiamo prima installare il *Matrix Authentication Service* o *MAS*.

## Matrix Authentication Service

Il Matrix Authentication Service (MAS) è un'implementazione di riferimento di un Authorization Server conforme agli standard OAuth 2.0 e OpenID Connect (OIDC), progettato specificamente per fungere da identity provider (IdP) nell'ecosistema del protocollo Matrix.

**Caratteristiche Formali dell'Architettura**

- Disaccoppiamento dell'Autenticazione (Separation of Concerns)

Tradizionalmente, l'autenticazione in Matrix era integrata direttamente nel Homeserver (ad esempio in Synapse). Il MAS sposta questa responsabilità all'esterno. Formalmente, questo trasforma il

Homeserver in un Resource Server (RS) secondo il framework OAuth 2.0, delegando la validazione delle identità e l'emissione dei token al MAS (Authorization Server).

- Standardizzazione dei Protocolli

Il MAS implementa le specifiche standard del settore per eliminare le API di autenticazione custom di Matrix (definite "Legacy"):

- RFC 6749 (OAuth 2.0): Per l'autorizzazione dell'accesso alle risorse.
- OpenID Connect Core 1.0: Per lo strato di identità sopra OAuth 2.0.
- Protocolli di backend: Supporta l'interfacciamento con database relazionali (PostgreSQL) per la persistenza dei dati e protocolli come LDAP o altri IdP upstream per l'autenticazione federata.
- Gestione del Ciclo di Vita delle Sessioni Il MAS non si limita al login, ma gestisce formalmente:
    - Emissione e Revoca dei Token: Gestione di Access Tokens a breve scadenza e Refresh Tokens.
    - Device Management: Associazione univoca tra sessioni OIDC e device_id del protocollo Matrix.
    - Compatibilità (Compatibility Layer): Fornisce un'interfaccia di traduzione per i client che non supportano ancora nativamente OIDC, simulando le vecchie API di login di Matrix.

**Installazione del MAS**

Siamo ancora sotto `/opt/matrix`. Creiamo il file di configurazione "vergine"

```
mkdir mas-config
docker run ghcr.io/element-hq/matrix-authentication-service config generate > mas-config/config.yaml
```

Vediamo un po' come è fatto questo file incredibile: `cat mas-config/config.yaml`

```yaml
http:
  listeners:
  - name: web
    resources:
    - name: discovery
    - name: human
    - name: oauth
    - name: compat
    - name: graphql
    - name: assets
    binds:
    - address: '[::]:8080'
    proxy_protocol: false
  - name: internal
    resources:
    - name: health
    binds:
    - host: localhost
      port: 8081
    proxy_protocol: false
  trusted_proxies:
  - 192.168.0.0/16
  - 172.16.0.0/12
  - 10.0.0.0/10
  - 127.0.0.1/8
  - fd00::/8
  - ::1/128
  public_base: http://[::]:8080/
  issuer: http://[::]:8080/
database:
  uri: postgresql://
  max_connections: 10
  min_connections: 0
  connect_timeout: 30
  idle_timeout: 600
  max_lifetime: 1800
email:
  from: '"Authentication Service" <root@localhost>'
  reply_to: '"Authentication Service" <root@localhost>'
  transport: blackhole
secrets:
  encryption:       [HIDDEN]
  keys:
  - key: |
      -----BEGIN RSA PRIVATE KEY-----
      [HIDDEN]
      -----END RSA PRIVATE KEY-----
```

```
  - key: |
      -----BEGIN EC PRIVATE KEY-----
      [HIDDEN]
      -----END EC PRIVATE KEY-----
  - key: |
      -----BEGIN EC PRIVATE KEY-----
      [HIDDEN]
      -----END EC PRIVATE KEY-----
  - key: |
      -----BEGIN EC PRIVATE KEY-----
      [HIDDEN]
      -----END EC PRIVATE KEY-----
passwords:
  enabled: true
  schemes:
  - version: 1
    algorithm: argon2id
  minimum_complexity: 3
matrix:
  kind: synapse
  homeserver: localhost:8008
  secret:       [HIDDEN]
  endpoint: http://localhost:8008/
```

Qui ci sono un po' di modifiche da fare, come è facile immaginare.

**ATTENZIONE**: secondo le specifiche il MAS deve stare su un dominio a sé stante rispetto a Synapse. Di conseguenza utilizzeremo `auth.example.org`.

Il file che vediamo è da personalizzare pesantemente. Ecco come deve risultare:

```
http:
  public_base: https://auth.example.org
  issuer: https://auth.example.org
  listeners:
  - name: web
    resources:
    - name: discovery
    - name: human
    - name: oauth
    - name: compat
    - name: graphql
    - name: assets
    binds:
    - host: 0.0.0.0
```

```
      port: 8080
    proxy_protocol: false
database:
  uri:  postgres://synapse:[HIDDEN]@mas-db/synapse
  max_connections: 10
  min_connections: 0
  connect_timeout: 30
  idle_timeout: 600
  max_lifetime: 1800

email:
  from: '"Authentication Service" <auth@example.org>'
  reply_to: '"Authentication Service" <auth@example.org>'
  transport: blackhole
secrets:
  encryption: [HIDDEN]
  keys:
  - key: |
      -----BEGIN RSA PRIVATE KEY-----
      [HIDDEN]
      -----END RSA PRIVATE KEY-----
  - key: |
      [HIDDEN]
  - key: |
      [HIDDEN]
      -----END EC PRIVATE KEY-----
  - key: |
      [HIDDEN]
passwords:
  enabled: true
  schemes:
  - version: 1
    algorithm: argon2id
  minimum_complexity: 3
matrix:
  kind: synapse
  homeserver: example.org
  secret: [HIDDEN]
  endpoint: http://synapse:8008
upstream_oauth2:
  providers:
    - id: 01HFRQFT5QFMJFGF01P7JAV2ME
      human_name: Authentik
      issuer: "https://auth.example.org/application/o/matrix-synapse/" #
      client_id: "[HIDDEN]"
      client_secret: "[HIDDEN]"
      token_endpoint_auth_method: client_secret_basic
```

```
62      scope: "openid profile email"
63      claims_imports:
64        localpart:
65          action: require
66          template: "{{ "{{" }} user.preferred_username }}"
67        displayname:
68          action: suggest
69          template: "{{ "{{" }} user.name }}"
70        email:
71          action: suggest
72          template: "{{ "{{" }} user.email }}"
73        avatar:
74          action: suggest
75          template: "{{ "{{" }} user.picture }}"
```

Ok questa sarà lunga da digerire, analizziamo le righe salienti

- 2: `public_base: https://auth.example.org`
- 3: `issuer: https://auth.example.org`: queste due devono puntare al MAS perché indicano chi farà da oidc provider per Synapse
- 18: la password deve essere generata con `openssl rand -hex 32` e deve essere corrispondente a quella che metteremo per il DB di MAS nel docker compose (ora lo abbiamo girato con docker run)
- [54-75]: questa è la configurazione dell'OIDC provider. Esatto. Dovete aver configurato un Provider + App in modalità *confidential* su Authentik: issuer, client_id, client_secret sono forniti da Authentik. I campi hanno *suggest* invece di *require* per permettere all'utente di valorizzare o meno i campi. Se ci sono, li suggerisce, ma sono modificabili.
- 56: Questo ID deve tassativamente essere un ULID valido generabile anche online https://ulidgenerator.com/ (https://ulidgenerator.com/)

**Se volete abilitare le registrazioni** e/o non utilizzare l'OIDC provider esterno dovete aggiungere/sostituire il blocco `upstream_oauth2` con:

```
account:
  password_registration_enabled: true
  email_change_allowed: true
  displayname_change_allowed: true
  password_change_allowed: true
  password_recovery_enabled: true
```

In questo caso vi conviene abilitare anche le mail in uscita (per esempio con Brevo), cancellando completamente il blocco:

```
email:
  from: '"Authentication Service" <auth@example.org>'
  reply_to: '"Authentication Service" <auth@example.org>'
  transport: blackhole
```

e posizionando:

```
email:
  smtp_host: "smtp-relay.brevo.com"
  smtp_port: 587
  smtp_user: "[HIDDEN]"
  smtp_pass: "[HIDDEN]"
  require_transport_security: true
  notif_from: "matrix@example.org"
  app_name: multihopnetworkmatrix
  client_base_url: https://matrix.example.org
  invite_client_location: https://example.org
```

**Modifica del docker compose file**

Aggiungiamo quindi il servizio MAS e relativo DB al `docker-compose.yml`:

```
mas-db:
    image: docker.io/postgres:16-alpine
    restart: unless-stopped
    networks:
      - synapse
    environment:
      - POSTGRES_USER=${MAS_POSTGRES_USER}
      - POSTGRES_DB=${MAS_POSTGRES_DB}
      - POSTGRES_PASSWORD=${MAS_POSTGRES_PASSWORD}
      - POSTGRES_INITDB_ARGS=--encoding=UTF-8 --lc-collate=C --lc-ctype=C
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${MAS_POSTGRES_USER} -d $
{MAS_POSTGRES_DB}"]
      interval: 5s
      timeout: 5s
      retries: 5
    volumes:
      - ./postgres-mas:/var/lib/postgresql/data
```

```
  mas:
    image: ghcr.io/element-hq/matrix-authentication-service:latest
    restart: unless-stopped
    networks:
      - synapse
    command: ["server"]
    volumes:
      - "./mas-config/config.yaml:/config.yaml:ro"
    depends_on:
      mas-db:
        condition: service_healthy
      synapse:
        condition: service_started
```

Va da sé che dentro `.env` devono essere aggiunte:

```
MAS_POSTGRES_USER=synapse
MAS_POSTGRES_DB=synapse
MAS_POSTGRES_PASSWORD=[HIDDEN]
```

**Modifica dell'homeserver**

Anche `data/homeserver.yml` deve essere aggiustato. Dobbiamo aggiungere:

```
matrix_authentication_service:
  enabled: true
  endpoint: http://mas:8080
  secret: "[HIDDEN]"
```

Al posto del `secret` ci dobbiamo scrivere il `secret` trovato nel file `mas-config/config.yml` alla riga 52.

**Nginx per auth**

Avendo adesso anche questo dominio in più da gestire, ovvero `auth.example.org`, è necessario servirlo proprio come abbiamo fatto con `matrix.example.org`. Creiamo quindi il file `nginx/conf.d/auth.example.org.conf`:

```
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
```

```
    server_name auth.example.org;

    ssl_certificate /etc/nginx/certs/origin.crt;
    ssl_certificate_key /etc/nginx/certs/origin.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        proxy_http_version 1.1;

        # Nome del servizio Docker per il MAS
        proxy_pass http://mas:8080;

        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Real-IP $remote_addr;

        proxy_redirect off;
    }
}

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

Anche in questo caso **dobbiamo andare a pubblicare il connettore, sempre quello di prima, su Cloudflare in modo che punti a https://nginx:443 per il dominio auth.example.org**

## Configurazione Ottimizzata di Nginx per lo Smistamento dei Worker

L'efficienza di un'infrastruttura Synapse multi-worker dipende interamente dalla precisione del layer di reverse proxying. Nginx agisce come bilanciatore di carico applicativo (Layer 7), instradando ogni richiesta HTTP verso il worker specializzato di competenza.

La configurazione ottimizzata adottata in produzione implementa tre principi ingegneristici chiave:

1. **Connessioni persistenti (`keepalive`)**: mantiene aperti i socket TCP verso i container di backend, abbattendo la latenza di handshake e prevenendo l'esaurimento dei descrittori di file del kernel.
2. **Consistent Hashing su base Utente (`$mxid_localpart`)**: estrae l'identificativo utente dall'access token (sia dall'header HTTP `Authorization` che dal parametro URL `access_token`), indirizzando in modo deterministico tutte le sincronizzazioni dello stesso account verso il medesimo worker, massimizzando l'efficacia della cache in-memory.
3. **Instradamento deterministico su `$uri` e Long-Polling senza buffering**: l'analisi dei path viene eseguita su `$uri` (privo della query string), mentre la disattivazione del buffering (`proxy_buffering off;`) e timeout estesi (`proxy_read_timeout 3600s;`) garantiscono la stabilità delle notifiche push e del sync sui dispositivi mobili.

Aggiorniamo il file `nginx/conf.d/matrix.example.org.conf`:

```
#
================================================================================
# UPSTREAMS - MATRICE DI SMISTAMENTO SYNAPSE & WORKER
#
================================================================================

# 1. Master Synapse (Scritture critiche, Eventi, Device e Amministrazione)
upstream synapse_master {
    server synapse:8008;
    keepalive 32;
}

# 2. MAS (Matrix Authentication Service - OIDC / SSO)
upstream synapse_mas {
    server mas:8080;
    keepalive 32;
}

# 3. Federation Worker (Federazione in entrata e scambio chiavi)
upstream synapse_federation {
    server generic-federation-presence:8083;
    keepalive 32;
}
```

```
# 4. Presence Worker
upstream synapse_presence_worker {
    server generic-presence:8087;
    keepalive 16;
}

# 5. Typing Worker
upstream synapse_typing {
    server generic-typing:8088;
    keepalive 16;
}

# 6. Client Sync & Read Pool (Load balancing coerente per utente)
upstream synapse_client_pool {
    hash $mxid_localpart consistent;
    server generic-sync-1:8082;
    server generic-sync-2:8084;
    server generic-sync-initial:8085;
    keepalive 64;
}

#
================================================================================
# ESTRAZIONE UTENTE PER CONSISTENT HASHING
#
================================================================================
map $arg_access_token $accesstoken_from_urlparam {
    default $arg_access_token;
    "~syt_(?<username>.*?)_.*" $username;
}

map $http_authorization $mxid_localpart {
    default $http_authorization;
    "~Bearer syt_(?<username>.*?)_.*" $username;
    "" $accesstoken_from_urlparam;
}

#
================================================================================
# MAPPA DI ROUTING LOGICO DEI PATH SYNAPSE (Basata su $uri senza query string)
#
================================================================================
map $uri $synapse_backend {
    # Default su master: qualsiasi endpoint non classificato o scrittura nuova
va al master
    default synapse_master;
```

```
    # --- AUTENTICAZIONE E OIDC (MAS) ---
    ~*^/_matrix/client/(r0|v3|unstable)/(login|logout|refresh)$ synapse_mas;
    ~*^/_matrix/client/v3/auth_issuer$                          synapse_mas;
    ~*^/_matrix/client/unstable/org.matrix.msc2965/auth_issuer$ synapse_mas;
    ~*^/_matrix/client/unstable/org.matrix.msc3861/             synapse_mas;

    # --- FEDERATION IN ENTRATA ---
    ~*^/_matrix/federation/ synapse_federation;
    ~*^/_matrix/key/        synapse_federation;

    # --- PRESENCE E TYPING ---
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/presence/
synapse_presence_worker;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/rooms/.*/typing/
synapse_typing;

    # --- SCRITTURE CRITICHE (Forzate esplicitamente su Master) ---
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/rooms/.*/(send|state|
redact)       synapse_master;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/rooms/[^/]+/(join|invite|leave|
ban|unban|kick)$ synapse_master;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/(createRoom|
join/)                synapse_master;
    ~*^/_matrix/client/(r0|v3|unstable)/user/[^/]+/
account_data/                   synapse_master;
    ~*^/_matrix/client/(r0|v3|unstable)/
sendToDevice/                               synapse_master;
    ~*^/_matrix/client/(r0|v3|unstable)/keys/
upload                                  synapse_master;
    ~*^/_matrix/client/(r0|v3|unstable)/
room_keys/                                  synapse_master;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/
profile/                              synapse_master;
    ~*^/_matrix/client/(r0|v3|unstable)/
devices                                     synapse_master;

    # --- SYNC E LETTURE FREQUENTI (Distribuite sul client pool bilanciato)
---
    ~*^/_matrix/client/(r0|v3)/sync$
synapse_client_pool;
    ~*^/_matrix/client/(api/v1|r0|v3)/events$
synapse_client_pool;
    ~*^/_matrix/client/(api/v1|r0|v3)/initialSync
synapse_client_pool;
    ~*^/_matrix/client/(api/v1|r0|v3)/rooms/[^/]+/initialSync$
synapse_client_pool;
```

```
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/rooms/.*/(context|members|
joined_members|state|messages) synapse_client_pool;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/rooms/.*/event/
synapse_client_pool;
    ~*^/_matrix/client/v1/rooms/.*/hierarchy$
synapse_client_pool;
    ~*^/_matrix/client/(r0|v3|unstable)/joined_rooms$
synapse_client_pool;
    ~*^/_matrix/client/(api/v1|r0|v3|unstable)/publicRooms$
synapse_client_pool;
    ~*^/_matrix/client/(r0|v3|unstable)/keys/(claim|query|changes)$
synapse_client_pool;
    ~*^/_matrix/client/(r0|v3|unstable)/user_directory/search$
synapse_client_pool;
    ~*^/_matrix/client/(r0|v3|unstable)/account/(whoami|3pid)$
synapse_client_pool;
    ~*^/_matrix/client/(r0|v3|unstable)/.*/tags$
synapse_client_pool;
    ~*^/_matrix/client/versions$
synapse_client_pool;
}

#
# SERVER HTTPS
#
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    http2 on;
    server_name matrix.example.org;

    # Ripristino indirizzo IP reale dal proxy perimetrale / Cloudflare
    set_real_ip_from  172.16.0.0/12;
    real_ip_header    CF-Connecting-IP;
    real_ip_recursive on;

    # Parametri crittografici TLS
    ssl_certificate /etc/nginx/certs/origin.crt;
    ssl_certificate_key /etc/nginx/certs/origin.key;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_session_cache shared:SSL:50m;
    ssl_session_timeout 1d;
    add_header Strict-Transport-Security "max-age=15768000" always;

    access_log /var/log/nginx/matrix.access.log;
```

```
    error_log /var/log/nginx/matrix.error.log;

    # Header di inoltro conformi RFC
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_http_version 1.1;
    client_max_body_size 50M;

    # 1. Well-known statico
    location ^~ /.well-known {
        root /var/www/html/matrix;
    }

    # 2. Endpoint Matrix Core (Sync, API, Admin)
    location ~ ^(/_matrix|/_synapse) {
        proxy_pass http://$synapse_backend;

        # Disattivazione del buffering e timeout lunghi per il long-polling
mobile
        proxy_buffering off;
        proxy_read_timeout 3600s;
        proxy_send_timeout 3600s;
    }

    # 3. Fallback per la radice
    location / {
        add_header X-Frame-Options SAMEORIGIN;
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header Content-Security-Policy "frame-ancestors 'none'";
        proxy_pass http://$synapse_backend;
        proxy_read_timeout 300s;
    }
}

# Redirect permanente HTTP -> HTTPS
server {
    listen 80;
    server_name matrix.example.org;
    return 301 https://$server_name$request_uri;
}
```

## BACKUP iniziale

Adesso: un bel backup di tutto. E teniamolo.

```
cd /opt/matrix
sudo tar cvf vector.iniziale.tar vector/
cd vector
```

## Creazione dell'admin

Per manipolare il MAS la questione non è proprio user-friendly. Tanto per cominciare tutta l'autenticazione è delegata a lui e quindi non è possibile creare utenti direttamente su Synapse. Ho creato questo script per evitare di dover scrivere sempre tutto il comando docker a mano. È da mettere dentro la directory che ha il docker compose e l'ho chiamato `mas-cli.sh`:

```
#!/usr/bin/env bash

docker_command="docker run --network synapse -p 59062:8080 -v \"./mas-config/
config.yaml:/config.yaml\" ghcr.io/element-hq/matrix-authentication-service"

for param in "$@"; do
  docker_command="$docker_command $param"
done

# Esegui il comando docker completo
echo "Esecuzione del comando: $docker_command" # Utile per il debug
eval "$docker_command"
```

Creiamo quindi l'admin:

```
./mas-cli.sh manage register-user admin \
  --password 'SuperPass1234!' \
  --admin \
  --email admin@example.org \
  --ignore-password-complexity \
  --yes
Esecuzione del comando: docker run --network synapse -p 59062:8080 -v "./mas-
config/config.yaml:/config.yaml" ghcr.io/element-hq/matrix-authentication-
service manage register-user admin --password SuperPass1234! --admin --email
admin@example.org --ignore-password-complexity --yes
User attributes
         Username: admin
        Matrix ID: @admin:localhost:8008
         Password: ********
            Email: admin@example.org
```

```
Can request admin: true
2026-04-13T20:50:42.782269Z  INFO mas_cli::commands::manage:936 User
registered user.id=01KP49RQYDZJHQC31KF8RKGSN6
```

## Element

Perfetto. A questo punto possiamo tranquillamente avviare Element e gustarci il nuovo homeserver puntandolo a `matrix.example.org`

## Ketesa ((Synapse Admin))

Ketesa è il tool de facto per l'amministrazione via web di Synapse. Aggiungiamolo al `docker-compose-yml` (pubblicatelo pure con un altro dominio Cloudflare):

```
  Ketesa:
    hostname: ketesa
    build:
      context: https://github.com/etkecc/ketesa.git
      dockerfile: docker/Dockerfile.build

      args:
        - BUILDKIT_CONTEXT_KEEP_GIT_DIR=1
        - REACT_APP_SERVER=https://matrix.example.org
    ports:
      - "9000:8080"
    restart: unless-stopped
    networks:
      - synapse
```

# Tuning?

## Redis

Iniziamo con il migliorare un po' la questione di redis, sostituendo il suo servizio in nel `docker-compose.yml` con

```
  redis:
    image: redis:7-alpine
    restart: unless-stopped
```

```
    networks:
      - synapse
    mem_limit: 1024
    mem_reservation: 256m
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
      timeout: 3s
      retries: 5
    volumes:
      - ./redis:/data
    command: >
      redis-server
      --appendonly yes
      --appendfsync everysec
      --maxmemory 768mb
      --maxmemory-policy noeviction
      --auto-aof-rewrite-percentage 100
      --auto-aof-rewrite-min-size 64mb
```

## Pgbouncer

Utilizziamolo per limitare le connessioni verso Postrgresql. Aggiungiamo il servizio a `docker-compose.yml`:

```
  pgbouncer:
    image: edoburu/pgbouncer:latest
    restart: unless-stopped
    networks:
      - synapse
    environment:
      - DB_HOST=db
      - DB_USER=${POSTGRES_USER}
      - DB_PASSWORD=${POSTGRES_PASSWORD}
      - POOL_MODE=transaction
      - DEFAULT_POOL_SIZE=10
      - MAX_CLIENT_CONN=50
      - AUTH_TYPE=scram-sha-256
      - RESERVE_POOL_SIZE=5
      - SERVER_RESET_QUERY=DISCARD ALL
      - IGNORE_STARTUP_PARAMETERS=extra_float_digits
```

e modifichiamo il file `data/homeserver.yaml` in modo che usi la nuova configurazione.

```
database:
    name: psycopg2
    args:
        user: synapse
        password: [HIDDEN]
        #host: db
        host: pgbouncer
        port: 5432
        database: synapse
        cp_min: 1
        cp_max: 5
```

La password rimane invariata, cambiano host, *cp_min* e *cp_max*

## Postgresql

Siamo su un N150, quindi ottimizziamo anche Postgres:

```
mkdir postgres-config
```

e poi

```
cat > /opt/matrix/postgres-config/postgresql.conf << 'EOF'
listen_addresses = '*'
shared_buffers = '1.5GB'
effective_cache_size = '3GB'
work_mem = '4MB'
maintenance_work_mem = '128MB'

max_connections = 50

wal_buffers = '4MB'
checkpoint_completion_target = 0.9
checkpoint_timeout = '15min'
max_wal_size = '2GB'
min_wal_size = '512MB'

autovacuum = on
autovacuum_max_workers = 2
autovacuum_naptime = '30s'
```

```
autovacuum_vacuum_scale_factor = 0.02
autovacuum_analyze_scale_factor = 0.01
autovacuum_vacuum_cost_limit = 1500
autovacuum_vacuum_cost_delay = '5ms'

max_worker_processes = 2
max_parallel_workers = 1
max_parallel_workers_per_gather = 1

log_min_duration_statement = '1000ms'
log_checkpoints = off
log_autovacuum_min_duration = '1000ms'
```

Questo file deve essere montato nel container di Postgres in `docker-compose.yml` quindi modifichiamo la sezione:

```
db:
  image: docker.io/postgres:16-alpine
  restart: unless-stopped
  networks:
    - synapse
  environment:
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_DB=${POSTGRES_DB}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_INITDB_ARGS=--encoding=UTF-8 --lc-collate=C --lc-ctype=C
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
    interval: 5s
    timeout: 5s
    retries: 5
  volumes:
    - ./postgres:/var/lib/postgresql/data
    - ./postgres-config/postgresql.conf:/etc/postgresql/postgresql.conf:ro
  command: postgres -c config_file=/etc/postgresql/postgresql.conf
```

# LiveKit aka Videoconferenze native

[ To Be Continued]

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).