Matrix: Synapse Server

Introduction and Architectural Overview

The **Synapse** server represents the reference implementation for the open **Matrix** protocol, the standard for federated instant messaging with end-to-end encryption (E2EE). In a federated ecosystem, communication secrecy is guaranteed through cryptographic payloads, while self-hosting the homeserver ensures complete sovereignty over transit metadata, message retention policies, and administrative access logs.

> *[!NOTE]* ***Operational Scope & High Availability****: This guide details the standard deployment of a multi-worker Synapse instance on a single host node, focusing on decoupling sync, federation, and authentication workloads. For enterprise high-availability (HA) requirements, this architecture can be scaled horizontally by deploying a redundant PostgreSQL cluster (e.g. PgBouncer/Patroni) and distributing workers across dedicated nodes.*

Taxonomy

Everything will be set up like this:



- OIDC Provider: handles user registration
- MAS, aka Matrix Authentication Service: the connector between OIDC and Matrix
- Workers: used to balance loads for scalability
- Nginx Server: the balancer, and it is crucial to configure it to distribute the load among the various workers
- Bridges: for interoperability with other protocols (such as WhatsApp, Signal, Telegram, ...)

NOTE: we will use a SPLIT domain. This means that:

- the Synapse server will listen on: *matrix.example.org*
- the handle will be: *user:example.org* and not *user:matrix.example.org*

- When you find something marked as **[HIDDEN]**, it means it was either configured by the application or generated with `openssl rand -hex 32`.

WARNING: you must already have the OIDC provider installed! How to do this has already been covered, so I will not cover it on this page. Alternatively, you can use MAS as a full OIDC provider for your Synapse server by enabling registration.

Installing Synapse: the foundation where everything starts

Let's start by creating the working directory:

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

Generating the configuration file

The initial configuration requires the *generate* command:

```
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
```

This command will generate the configuration file under `data/homeserver.yaml`, and this is the file that must be modified to add the `public_baseurl`.

```
server_name: "example.org"
public_baseurl: "matrix.example.org" # <-- To add
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"
```

Creating the docker network

We will create a docker network specifically for Synapse:

```
docker network create synapse
be636df73d51c5275918e9f8f07fbc748d0da6b4ddd250a2cba8c408a1f609ab
```

The Docker Compose File

Given the complexity of the deployment we are attempting, it's appropriate to create a docker-compose file. It will be modified over time to build the final implementation.

Let's start with the environment: `.env` (DB password generated with openssl);

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

VIRTUAL_HOST=example.org
VIRTUAL_PORT=8008

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

The `docker-compose.yml` instead will be:

```
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
```

No docker compose up yet: we need to tell Synapse to use *Postgres* instead of *SQLite*. We must edit `data/homeserver.yaml`, replacing:

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

with:

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

Obviously the DB password is the one generated with openssl, and which was written in the `.env` file. Only at this point can we proceed with the next step:

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

Is it usable then? Not at all, but at least we can see there are no errors.

Reverse proxy with Nginx (and Cloudflare dances)

Now there's a distinction:

- matrix.example.org
- example.org While the first one only handles Synapse, the second one must contain the well-known files to allow handle resolution. So, while the first one can be containerized here, the second one will have to be modified manually as it is assumed to be elsewhere, serving the domain directly, for example. Consequently, in this container we will only manage `matrix.example.org`.

Being behind Cloudflare, it is necessary to go and get hands dirty directly on the domain's admin panel.

In the domain management, on the right: `SSL/TLS -> Origin Server`. I use the `ECC` key and set it for `*.example.org` and `example.org`.

Let's go back to our machine leaving the certificate page visible:

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

Let's save the private and public keys in:

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

Again on the Cloudflare page, let's go to the Zero Trust section and create a docker connector: let's note down the `TUNNEL_TOKEN`.

Then let's populate `nginx/conf.d/matrix.example.org.conf`:

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

    # We use Cloudflare Origin CA certificates
    ssl_certificate /etc/nginx/certs/origin.crt;
    ssl_certificate_key /etc/nginx/certs/origin.key;

    # Security optimizations
    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;
    }

    # Matrix Federation
    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 *;
    }
}
```

Let's modify the docker-compose file to also include Nginx by adding:

```
nginx:
    image: nginx:alpine
    restart: unless-stopped
    networks:
      - synapse
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d:ro
      - ./nginx/certs:/etc/nginx/certs:ro # We mount the directory with origin.crt and
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
```

Let's also add the necessary part for Nginx to the `.env` file:

```
CLOUDFLARE_TUNNEL_TOKEN=TUNNEL_TOKEN_SEGNATO_PRIMA
```

Let's start up both Nginx and the tunnel:

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

And let's go back to Cloudflare to configure the newly created connector by adding the published application:

- Subdomain: matrix
- Domain: example.org
- Service: https
- URL: nginx:443
- Other application settings:
    - TLS
        - No TLS verification: on
        - HTTP2 connection: on

At this point, visiting `https://matrix.example.org` will show the basic Synapse page on the Matrix protocol.

Since we want the handles to be different from the actual domain name, it is **necessary to also modify the Nginx configuration serving the main domain, namely example.org**. Specifically, we need to add:

```
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"}}';
}
```

Now we can see if everything is correctly configured for the split domain by going to `https://federationtester.matrix.org/` and entering `example.org` in the test.

The workers

A rather devastating problem is that if the main process does everything (synchronization, federation, exchanges, etc.), everything gets overloaded and runs slow. We need to scale. That's why we create multiple *workers*, each specialized in something and doing only that.

Let's modify `docker-compose.yml` to include them, adding:

```
  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
```

For each of these, a worker file must be created inside `data/workers/`, so:

```
mkdir data/workers
```

Inside we will have: `event-persister.yaml`

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

# Listener for internal communication (Replication)
worker_listeners:
  - port: 8086
    type: http
    resources:
      - names: [replication]

# Redis config (must point to the service name in compose)
redis:
  enabled: true
  host: redis
  port: 6379
```

then `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
```

then `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
```

and also `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
```

and finally (for now) `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
```

let's fix the permissions:

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

and modify the mapping in `data/homeserver.yaml`:

```
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
[...rest of configuration ...]
```

Technically, we could now proceed with the Nginx configuration to balance across the workers, but we won't do it yet. We first need to install the *Matrix Authentication Service* or *MAS*.

Matrix Authentication Service

The Matrix Authentication Service (MAS) is a reference implementation of an Authorization Server compliant with OAuth 2.0 and OpenID Connect (OIDC) standards, specifically designed to act as an identity provider (IdP) in the Matrix protocol ecosystem.

Formal Architecture Characteristics

- Authentication Decoupling (Separation of Concerns)

Traditionally, authentication in Matrix was integrated directly into the Homeserver (for example, in Synapse). MAS moves this responsibility externally. Formally, this transforms the Homeserver into a Resource Server (RS) according to the OAuth 2.0 framework, delegating identity validation and token issuance to the MAS (Authorization Server).

- Standardization of Protocols

MAS implements industry-standard specifications to eliminate custom Matrix authentication APIs (defined as "Legacy"):

- RFC 6749 (OAuth 2.0): For resource access authorization.
- OpenID Connect Core 1.0: For the identity layer on top of OAuth 2.0.
- Backend protocols: Supports interfacing with relational databases (PostgreSQL) for data persistence and protocols such as LDAP or other upstream IdPs for federated authentication.
- Session Lifecycle Management MAS does not limit itself to login, but formally manages:
  - Token Issuance and Revocation: Management of short-lived Access Tokens and Refresh Tokens.
  - Device Management: Unique association between OIDC sessions and Matrix protocol device_ids.
  - Compatibility (Compatibility Layer): Provides a translation interface for clients that do not yet natively support OIDC, simulating legacy Matrix login APIs.

Installing MAS

We are still under `/opt/matrix`. Let's create the "virgin" configuration file:

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

Let's see what this incredible file looks like: `cat mas-config/config.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/
```

Here there are some modifications to make, as is easy to imagine.

WARNING: according to the specifications, MAS must reside on a separate domain from Synapse. Consequently, we will use `auth.example.org`.

The file we see needs to be heavily customized. Here is how it should look:

```
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]"
```

```
61      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 }}"
```

Okay, this will be a lot to digest, let's analyze the key lines:

- 2: `public_base: https://auth.example.org`
- 3: `issuer: https://auth.example.org`: these two must point to MAS as they indicate who will act as the OIDC provider for Synapse
- 18: the password must be generated with `openssl rand -hex 32` and must correspond to what we will set for the MAS DB in docker-compose (now we ran it with docker run)
- [54-75] (or appropriate range): this is the OIDC provider configuration. Correct. You must have configured a Provider + App in *confidential* mode on Authentik: issuer, client_id, client_secret are provided by Authentik. The fields have *suggest* instead of *require* to allow the user to populate the fields or not. If they exist, it suggests them, but they are editable.
- 56: This ID must strictly be a valid ULID, which can also be generated online at https://ulidgenerator.com/ (https://ulidgenerator.com/)

If you want to enable registrations and/or not use the external OIDC provider, you must add/replace the `upstream_oauth2` block with:

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

In this case, you should also enable outgoing emails (for example with Brevo), completely deleting the block:

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

and placing:

```
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
```

Modifying the docker-compose file

Let's add the MAS service and its DB to `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
```

It goes without saying that in `.env` you must add:

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

Modifying the homeserver

Also `data/homeserver.yaml` must be adjusted. We need to add:

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

Instead of the `secret`, we must write the `secret` found in the file `mas-config/config.yaml` at line 52.

Nginx for auth

Having now this additional domain to manage, namely `auth.example.org`, it is necessary to serve it just like we did with `matrix.example.org`. Let's create the 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;

        # Docker service name for 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;
}
```

Also in this case, **we must publish the connector, the same one as before, on Cloudflare so that it points to https://nginx:443 for the domain auth.example.org**

Optimized Nginx Configuration for Worker Dispatching

The efficiency of a multi-worker Synapse infrastructure depends entirely on the precision of the reverse proxy layer. Nginx acts as an application load balancer (Layer 7), routing every HTTP request to its dedicated specialized worker.

The optimized configuration adopted in production implements three key engineering principles:

1. **Persistent connections (`keepalive`)**: keeps TCP sockets open to backend containers, eliminating handshake latency and preventing exhaustion of kernel file descriptors.
2. **User-based Consistent Hashing (`$mxid_localpart`)**: extracts the user identifier from the access token (both from the `Authorization` HTTP header and the `access_token` URL query parameter), deterministically directing all sync requests for the same account to the same worker, thereby maximizing in-memory cache hit rates.
3. **Deterministic routing on `$uri` and Buffering-free Long-Polling**: path pattern matching is performed against `$uri` (without query parameters), while disabling buffering (`proxy_buffering off;`) and extending timeouts (`proxy_read_timeout 3600s;`) guarantees stability for mobile push notifications and device sync.

Let us update `nginx/conf.d/matrix.example.org.conf`:

```
# ============================================================================
# UPSTREAMS - SYNAPSE & WORKER DISPATCH MATRIX
# ============================================================================

# 1. Master Synapse (Critical writes, Events, Device management, and Administration)
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 (Inbound federation and key exchange)
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 (Consistent hashing per user)
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;
}

# ==============================================================================
# USER EXTRACTION FOR 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;
}

# ==============================================================================
# SYNAPSE PATH ROUTING MAP (Evaluated on $uri without query string)
# ==============================================================================
map $uri $synapse_backend {
    # Default to master: unclassified endpoints or state-changing writes go to master
    default synapse_master;

    # --- AUTHENTICATION & 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;

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

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

    # --- CRITICAL WRITES (Explicitly pinned to 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 & HIGH-FREQUENCY READS (Distributed across the balanced client pool) ---
    ~*^/_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;
}

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

    # Restore real client IP behind perimeter proxy / Cloudflare
    set_real_ip_from  172.16.0.0/12;
    real_ip_header    CF-Connecting-IP;
    real_ip_recursive on;

    # TLS Cryptographic Parameters
    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;

    # RFC-compliant proxy forwarding headers
    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. Static well-known
    location ^~ /.well-known {
```

```
        root /var/www/html/matrix;
    }

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

        # Disable buffering and enable long timeouts for mobile long-polling
        proxy_buffering off;
        proxy_read_timeout 3600s;
        proxy_send_timeout 3600s;
    }

    # 3. Fallback root location
    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;
    }
}

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

Initial BACKUP

Now: a nice backup of everything. And let's keep it.

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

Creating the admin

Manipulating MAS is not exactly user-friendly. To begin with, all authentication is delegated to it, so it's not possible to create users directly on Synapse. I created this script to avoid having to write the entire docker command by hand every time. It should be placed inside the directory containing the docker-compose file, and I named it `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

# Execute full docker command
echo "Executing command: $docker_command" # Useful for debugging
eval "$docker_command"
```

So, let's create the admin:

```
./mas-cli.sh manage register-user admin \
  --password 'SuperPass1234!' \
  --admin \
  --email admin@example.org \
  --ignore-password-complexity \
  --yes
Executing command: 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

Perfect. At this point, we can easily start Element and enjoy the new homeserver by pointing it to `matrix.example.org`

Ketesa ((Synapse Admin))

Ketesa is the de facto tool for web-based Synapse administration. Let's add it to `docker-compose.yml` (feel free to publish it under another Cloudflare domain):

```
  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

Let's start by improving the Redis situation, replacing its service in `docker-compose.yml` with:

```
  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

Let's use it to limit connections to PostgreSQL. Add the service to `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
```

and modify the `data/homeserver.yaml` file to use the new configuration:

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

The password remains unchanged; host, *cp_min*, and *cp_max* change.

Postgresql

We are on an N150, so let's optimize Postgres as well:

```
mkdir postgres-config
```

and then:

```
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'

EOF
```

This file must be mounted in the Postgres container in `docker-compose.yml`, so let's modify the section:

```
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 Native Videoconferencing

[To Be Continued]

[May the big mole be with you]

[EOF]