Mastodon Glitch Soc

MASTODON GLITCH SOC

You all know **X** aka ex-**TWITTER**. You've probably also heard of the fediverse. But tell me, kids, do you know what the fediverse is? Well, I'm sure you can easily search for it elsewhere, like on Wikipedia. What I care to make clear is that to enter the fediverse you have two main ways:

- register on an instance
- create your own instance and participate. It goes without saying that what interests me, and consequently you since you are here, is creating your own.

To enter the fediverse, we start with Mastodon Glitch Soc edition. The reason we start with this and not Vanilla is because it is objectively more versatile: it allows writing in Markdown, allows writing posts up to 5k characters by default instead of 500, and various other things.

Ingredients

To create the instance you will need Docker, a domain, an outgoing mail server, and Cloudflare in order not to expose your IP address. And obviously a machine running 24/7.

If you are looking for one-click solutions like YunoHost, this is not the right place. Here we work understanding every component of the stack, because when something breaks in production, the magic button is not enough.

Cloudflare Tunnel: Preparing the ground

Even before writing a single line of docker-compose, we need to prepare the Cloudflare tunnel. Why? Because we don't want to expose any ports on our machine, not even 80 or 443. The tunnel will handle everything.

1. Go to **Cloudflare Dashboard** → **Zero Trust** → **Networks** → **Tunnels**.
2. Create a new tunnel (let's call it `mastodon-tunnel`).
3. In the "Public Hostname" section, add:
   - **Subdomain**: `nebula` (or whatever you want)
   - **Domain**: your domain (e.g., `example.org`)
   - **Service**: `http://localhost:80`
4. Cloudflare will give you a long, ugly **token**. Save it somewhere, you'll need it.
5. Now take that token and create a `.env` file in the directory we will work in (it doesn't exist yet, but we'll create it shortly):

```
echo "TUNNEL_TOKEN=your_token" > .env
```

This file is needed because the docker-compose contains `${TUNNEL_TOKEN}`. If you don't create it, the cloudflared container won't start.

Note on DNS: Cloudflare Tunnel doesn't require special A or CNAME records if you use the tunnel in cloudflared mode. The daemon will handle connecting automatically. It's enough that the domain is managed by Cloudflare.

Docker Compose

Let's create our nice working directory:

```
sudo mkdir /opt/mastodon && sudo chown 1000:1000 /opt/mastodon && cd /opt/mastodon
```

let's create the `.env.production` file: LET'S CREATE IT because otherwise, if it doesn't exist, half the scripts might not work.

```
touch .env.production
```

Let's also create the `.env` file for the Cloudflare token (if you haven't already done so):

```
echo "TUNNEL_TOKEN=your_token_brutto" > .env
```

let's throw down the `docker-compose.yml`:

```
services:
  db:
    restart: always
    image: postgres:14-alpine
    shm_size: 256mb
    networks:
      - internal_network
    healthcheck:
      test: ['CMD', 'pg_isready', '-U', 'postgres']
    volumes:
      - ./postgres14:/var/lib/postgresql/data
    environment:
      - 'POSTGRES_HOST_AUTH_METHOD=trust'

  redis:
    restart: always
    image: redis:7-alpine
    networks:
      - internal_network
    healthcheck:
      test: ['CMD', 'redis-cli', 'ping']
    volumes:
      - ./redis:/data

  es:
    restart: always
    image: docker.elastic.co/elasticsearch/elasticsearch:9.3.3
    environment:
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m -Des.enforce.bootstrap.checks=true"
      - "xpack.license.self_generated.type=basic"
      - "xpack.security.enabled=false"
      - "xpack.watcher.enabled=false"
      - "xpack.graph.enabled=false"
      - "xpack.ml.enabled=false"
      - "bootstrap.memory_lock=true"
      - "cluster.name=es-mastodon"
      - "discovery.type=single-node"
      - "thread_pool.write.queue_size=1000"
    networks:
       - external_network
       - internal_network
    healthcheck:
       test: ["CMD-SHELL", "curl --silent --fail localhost:9200/_cluster/health || exit 1"]
    volumes:
       - ./elasticsearch:/usr/share/elasticsearch/data
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
  web:
    image: ghcr.io/glitch-soc/mastodon:latest
    restart: always
    env_file: .env.production
    command: bundle exec puma -C config/puma.rb
    networks:
      - external_network
      - internal_network
    healthcheck:
      # prettier-ignore
      test: ['CMD-SHELL',"curl -s --noproxy localhost localhost:3000/health | grep -q 'OK' || exit 1"]
    depends_on:
```

```
      - db
      - redis
      - es
    volumes:
      - ./public/system:/mastodon/public/system
  streaming:
    image: ghcr.io/glitch-soc/mastodon-streaming:latest
    restart: always
    env_file: .env.production
    command: node ./streaming/index.js
    networks:
      - external_network
      - internal_network
    healthcheck:
      # prettier-ignore
      test: ['CMD-SHELL', "curl -s --noproxy localhost localhost:4000/api/v1/streaming/health | grep -q 'OK' || exit 1"]
    depends_on:
      - db
      - redis
  sidekiq:
    image: ghcr.io/glitch-soc/mastodon:latest
    restart: always
    env_file: .env.production
    command: bundle exec sidekiq
    depends_on:
      - db
      - redis
    networks:
      - external_network
      - internal_network
    volumes:
      - ./public/system:/mastodon/public/system
    healthcheck:
      test: ['CMD-SHELL', "ps aux | grep '[s]idekiq\ 8' || false"]
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf:ro
    networks:
      - external_network
      - internal_network
    depends_on:
      - web
      - streaming
    restart: always
  tunnel:
    image: cloudflare/cloudflared:latest
    networks:
      - external_network
      - internal_network
    restart: "always"
    command: tunnel --no-autoupdate run --token ${TUNNEL_TOKEN}

networks:
  external_network:
  internal_network:
    internal: true
```

As you will notice, there are various services here, including `elasticsearch` for full-text search, and ... no exposed ports. Yes, because we use `cloudflared`, so no exposed ports.

Key creation

In the `.env.production` file, we must write some keys that we will obtain with `docker compose run --rm web bundle exec rails secret`. **This command must be run SIX times** because we need **SIX** keys. After that, we can run `docker compose run --rm web bundle exec rake mastodon:webpush:generate_vapid_key` to create the VAPID keys. So the correct procedure is:

```
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rails secret
docker compose run --rm web bundle exec rake mastodon:webpush:generate_vapid_key
```

The first 6 commands will return 6 random keys: each must be placed in `.env.production` next to the `=` symbol:

```
SECRET_KEY_BASE=
OTP_SECRET=
PAPERCLIP_SECRET=
ACTIVE_RECORD_ENCRYPTION_PRIMARY_KEY=
ACTIVE_RECORD_ENCRYPTION_KEY_DERIVATION_SALT=
ACTIVE_RECORD_ENCRYPTION_DETERMINISTIC_KEY=
```

The last command will return the VAPID keys, and they also want to be placed in `.env.production`:

```
VAPID_PRIVATE_KEY=
VAPID_PUBLIC_KEY=
```

Reverse proxy creation

We will use `nginx` as a reverse proxy (in my case, the server will be named `social.example.org`):

```
mkdir nginx/
vim nginx/nginx.conf
```

Its content will be populated with:

```
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}
upstream mbackend {
    server web:3000 fail_timeout=0;
}

upstream mstreaming {
    server streaming:4000 fail_timeout=0;
}

# Cache for media and assets
proxy_cache_path /var/cache/mnginx levels=1:2 keys_zone=MCACHE:10m inactive=7d max_size=1g;



# Redirect HTTP -> HTTPS
server {
    listen 80;
    server_name social.example.org;
    keepalive_timeout    70;
    sendfile             on;
    client_max_body_size 80m;
    root /var/www/mastodon/public;

    # Gzip Compression
    gzip on;
    gzip_disable "msie6";
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_buffers 16 8k;
    gzip_http_version 1.1;
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss
```

```
text/javascript image/svg+xml image/x-icon;

    # Static file management and Rails routing
    location / {
        try_files $uri @proxy;
    }

    # Cache Control for static files (Avatar, Emoji, Packs)
    location ~ ^/(assets|avatars|emoji|headers|packs|shortcuts|sounds|system)/ {
        add_header Cache-Control "public, max-age=2419200, must-revalidate";
        add_header Strict-Transport-Security "max-age=63072000; includeSubDomains";
        try_files $uri @proxy;
    }

    location = /sw.js {
        add_header Cache-Control "public, max-age=604800, must-revalidate";
        try_files $uri @proxy;
    }

    # --- STREAMING PROXY (Realtime API) ---
    location ^~ /api/v1/streaming {

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr; # Passes the real IP now
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header Proxy "";

        proxy_pass http://mstreaming;
        proxy_buffering off;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;

        tcp_nodelay on;
    }

    # --- MAIN PROXY (Web & API) ---
    location @proxy {
        # Use debug log to monitor real IP passing

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr; # Crucial to let Mastodon see the true IP
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header Proxy "";
        proxy_pass_header Server;

        proxy_pass http://mbackend;
        proxy_buffering on;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;

        # Cache configured for 200 responses
        proxy_cache MCACHE;
        proxy_cache_valid 200 7d;
        proxy_cache_valid 410 24h;
        proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;
        add_header X-Cached $upstream_cache_status;

        tcp_nodelay on;
    }

    error_page 404 500 501 502 503 504 /500.html;
}
```

.env.production

Let's reopen the `.env.production` file and fill in what is missing.

Warning: there are two cases here.

Case 1: Split domain (recommended) If you want your username to be @user@example.org but the instance to be on social.example.org, use this configuration:

Split domain: identity is on example.org, server on social.example.org

```
WEB_DOMAIN=social.example.org
LOCAL_DOMAIN=example.org
```

Case 2: Single domain If you prefer to have everything on social.example.org (username @user@social.example.org):

```
LOCAL_DOMAIN=social.example.org
```

I use split domain, so my configuration continues like this:

```
# I use split domain so my LOCAL_DOMAIN will be example.org and not social.example.org
WEB_DOMAIN=social.example.org
LOCAL_DOMAIN=example.org
SINGLE_USER_MODE=false
SECRET_KEY_BASE= XXXXXXXXXX
OTP_SECRET= XXXXXXXXXX
PAPERCLIP_SECRET= XXXXXXXXXX
VAPID_PRIVATE_KEY= XXXXXXXXXX
VAPID_PUBLIC_KEY= XXXXXXXXXX
ACTIVE_RECORD_ENCRYPTION_PRIMARY_KEY= XXXXXXXXXX
ACTIVE_RECORD_ENCRYPTION_KEY_DERIVATION_SALT= XXXXXXXXXX
ACTIVE_RECORD_ENCRYPTION_DETERMINISTIC_KEY= XXXXXXXXXX
DB_HOST=db
DB_PORT=5432
DB_NAME=postgres
DB_USER=postgres
DB_PASS=
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=
# I use Brevo, you will use yours
SMTP_SERVER=smtp-relay.brevo.com
SMTP_PORT=587
SMTP_LOGIN= XXXXXXXXXX
SMTP_PASSWORD= XXXXXXXXXX
SMTP_AUTH_METHOD=plain
SMTP_OPENSSL_VERIFY_MODE=peer
SMTP_ENABLE_STARTTLS=auto
SMTP_FROM_ADDRESS=mastodon@example.org

RAILS_SERVE_STATIC_FILES=true
ES_ENABLED=true
ES_HOST=es
ES_PORT=9200

MAX_TOOT_CHARS=5000

# If you have your OIDC provider you can put these directives here. I use the one created back then

OIDC_ENABLED=true
OIDC_DISPLAY_NAME=Authentik  # Or whatever you want to appear on the button
OIDC_ISSUER=https://auth.example.org/application/o/dreadful-mastodon/ # Authentik Issuer URL
OIDC_DISCOVERY=true
OIDC_CLIENT_ID= XXXXXXXXXX
OIDC_CLIENT_SECRET= XXXXXXXXXX
OIDC_REDIRECT_URI=https://social.example.org/auth/auth/openid_connect/callback
OIDC_SCOPE=openid,email,profile
OIDC_INFO_FIELDS=email
OIDC_UID_FIELD=preferred_username
```

```
OIDC_AUTOLINK=true
OIDC_SECURITY_ASSUME_EMAIL_IS_VERIFIED=true
# This must be set to true if you want to enable registrations. I don't do it because I want it bound to my IDP.
LOCAL_SIGNUP_ENABLED=false
TRUSTED_PROXY_IP="192.168.0.0/16,172.16.0.0/12,10.0.0.0/8,173.245.48.0/20,103.21.244.0/22,103.22.200.0/22,103.31.4.0/22,141.
```

Startup

Once this is done we can proceed:

```
docker compose run --rm -e DISABLE_DATABASE_ENVIRONMENT_CHECK=1 web bundle exec rake db:setup
```

This will build the database; after that:

```
mkdir -p ./public/system
sudo chown 991:991 ./public/system
```

And to finish:

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

Should there be permission issues on ElasticSearch:

```
mkdir elasticsearch; sudo chown 1000:1000 elasticsearch
touch .env.es
```

And finally:

```
docker compose down; docker compose up -d
```

Creating the admin

```
docker compose run --rm web bin/tootctl accounts create errno0x0d --email admin@example.org --confirmed --role Owner
```

Split Domain

If you chose split domain (like me), you must add a piece of configuration on the server managing the main domain example.org (the one without nebula.). This is not the Mastodon server, but your main web server (e.g. the one serving your personal website).

In the nginx (or Apache, or Caddy) configuration file of your main domain, add:

```
location ~ ^/.well-known/(webfinger|nodeinfo|host-meta) {
    # No need for proxy_pass or CORS headers here anymore,
    # because the client will be sent directly to nebula.

    return 301 https://social.example.org$request_uri;
}
```

[May the big mole be with you]

[EOF]