Mastodon Glitch Soc

MASTODON GLITCH SOC

Conoscete tutti **X** aka ex **TWITTER**. Avrete anche sentito parlare del fediverso. Ma ditemi, bambini, sapete che cosa è il fediverso? Ebbene, sono sicuro che siete tranquillamente in grado di cercarvelo altrove, come per esempio su Wikipedia. Quello che mi preme far capire è che per entrare nel fediverso avete due modi principali:

- registrarvi su un'istanza
- crearvi la vostra istanza e parteciparvi. Va da sé che ciò che interessa me, e di conseguenza voi visto che siete qui, è crearvi la vostra.

Per entrare nel fediverso partiamo da Mastodon versione Glitch Soc. Il motivo per cui partiamo da questa e non dal Vanilla è perché oggettivamente è più versatile: permette di scrivere in linguaggio mark down, permette di scrivere di default post fino a 5k caratteri e non 500, e varie altre ed eventuali.

Ingredienti

Per creare l'istanza avrete bisogno di Docker, un dominio, un server di posta in uscita e Cloudflare in modo da non esporre il vostro indirizzo IP. E ovviamente una macchina accesa h24.

Se cercate soluzioni one-click come YunoHost, questo non è il posto giusto. Qui si lavora comprendendo ogni componente dello stack, perché quando qualcosa si rompe in produzione il pulsante magico non basta.

Cloudflare Tunnel: Prepariamo il terreno

Prima ancora di scrivere una riga di docker-compose, dobbiamo preparare il tunnel Cloudflare. Perché? Perché non vogliamo esporre nessuna porta della nostra macchina, nemmeno la 80 o la 443. Il tunnel si occuperà di tutto.

1. Andate su **Cloudflare Dashboard → Zero Trust → Networks → Tunnels**.
2. Create un nuovo tunnel (chiamiamolo `mastodon-tunnel`).

3. Nella sezione "Public Hostname", aggiungete:
    - **Subdomain**: `nebula` (o quello che volete)
    - **Domain**: il vostro dominio (es. `example.org`)
    - **Service**: `http://localhost:80`
4. Cloudflare vi darà un **token** lungo e brutto. Salvatelo da qualche parte, servirà.
5. Ora prendete quel token e create un file `.env` nella directory dove lavoreremo (ancora non esiste, ma lo creiamo tra poco):

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

Questo file serve perché nel docker-compose c'è scritto ${TUNNEL_TOKEN}. Se non lo create, il container cloudflared non parte.

Nota sui DNS: Cloudflare Tunnel non richiede record A o CNAME particolari se usate il tunnel in modalità cloudflared. Il demone si occuperà di collegarsi automaticamente. Basta che il dominio sia gestito da Cloudflare.

Docker Compose

Creiamo la nostra bella working directory:

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

creiamo il file `.env.production`: CREIAMOLO perché altrimenti se non esiste mezzi script potrebbero non andare.

```
touch .env.production
```

Creiamo anche il file `.env` per il token di Cloudflare (se non l'avete già fatto):

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

buttiamo giù il `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
```

Come noterete qui ci sono vari servizi tra cui `elasticsearch` per fare la ricerca full text e ... nessuna porta esposta. Già, perché usiamo `cloudflared` quindi nessuna porta esposta.

Creazione delle chiavi

Nel file `.env.production` dobbiamo scrivere alcune chiavi che otterremo con `docker compose run --rm web bundle exec rails secret`. **Questo comando va eseguito SEI volte*** perché abbiamo necessità di **SEI** chiavi. Dopodiché possiamo dare un `docker compose run --rm web bundle exec rake mastodon:webpush:generate_vapid_key` per creare le vapid keys. Quindi il procedimento corretto è

```
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
```

I primi 6 comandi daranno 6 chiavi random: ciascuna dovrà essere messa in `docker-compose.yml` accanto al simbolo di `=`:

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

L'ultimo comando invece vi restituirà le Vapid keys, e anche loro vogliono essere messe in `.env.production`:

```
VAPID_PRIVATE_KEY=
VAPID_PUBLIC_KEY=
```

Creazione del reverse proxy

Utilizzeremo `nginx` come reverse proxy (nel mio caso il server si chiamera `social.example.org`):

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

Il suo interno sarà popolato con:

```
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 per i media e gli asset
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;

    # Compressione Gzip
    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;

    # Gestione file statici e routing Rails
    location / {
        try_files $uri @proxy;
    }

    # Cache Control per file statici (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;
    }

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

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr; # Ora passa l'IP reale
        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;
    }

    # --- PROXY PRINCIPALE (Web & API) ---
    location @proxy {
        # Usa il log di debug per monitorare il passaggio dell'IP reale

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr; # Fondamentale per far vedere
l'IP vero a Mastodon
        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 configurata per le risposte 200
        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

Riapriamo il file `.env.production` e compiliamolo con quanto manca.

Attenzione: qui ci sono due casi.

Caso 1: Split domain (consigliato) Se volete che il vostro username sia @utente@example.org ma l'istanza sia su social.example.org, usate questa configurazione:

Split domain: l'identità è su example.org, il server su social.example.org

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

Caso 2: Domain unico Se preferite avere tutto su social.example.org (username @utente@social.example.org):

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

Io uso split domain, quindi la mia configurazione prosegue così:

```
# Io uso lo split domain quindi il mio LOCAL_DOMAIN sarà example.org e non
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=
# Io uso Brevo, voi utilizzerete il vostro
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

# Se avete il vostro provider OIDC potete metterci queste direttive. Io
utilizzo quello creato a suo tempo

OIDC_ENABLED=true
OIDC_DISPLAY_NAME=Authentik  # O quello che vuoi che appaia sul bottone
OIDC_ISSUER=https://auth.example.org/application/o/dreadful-mastodon/ # URL
dell'Issuer di Authentik
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
# Qui va messo a true se volete abilitare le iscrizioni. Io non lo faccio
perché lo voglio vincolato al mio IDP.
```

```
LOCAL_SIGNUP_ENABLED=false
TRUSTED_PROXY_IP="192.168.0.0/16,172.16.0.0/12,10.0.0.0/8,173.245.48.0/20,103.21.244.0/22,10
```

Startup

Fatto questo possiamo proseguire:

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

Questo costruirà il database; dopodiché:

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

E per terminare

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

Qualora ci fossero problemi di permessi su elastic search:

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

E per finire

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

Creazione dell'admin

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

Split Domain

Se avete scelto lo split domain (come me), dovete aggiungere un pezzo di configurazione sul server che gestisce il dominio principale

example.org (quello senza il nebula.). Questo non è il server di Mastodon, ma il vostro web server principale (es. quello che serve il vostro sito personale).

Nel file di configurazione di nginx (o Apache, o Caddy) del vostro dominio principale, aggiungete:

```
location ~ ^/.well-known/(webfinger|nodeinfo|host-meta) {
    # Non serve più il proxy_pass né gli header CORS qui,
    # perché il client verrà mandato direttamente su nebula.

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

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).