Ubuntu 22.04 LTS に Docker を使用して Mastodon Social Network をインストールする方法

このチュートリアルは、これらの OS バージョン用に存在します

- Ubuntu 22.04 (Jammy Jellyfish)
- Ubuntu 18.04 (Bionic Beaver)

このページでは

Mastodon は、無料の分散型オープンソース ソーシャル ネットワークです。 Twitterの代替として作成されました。 Twitter と同じように、ユーザーは相互にフォローして、メッセージ、画像、動画を投稿できます。ただし、Twitter とは異なり、コンテンツの中央ストアや権限はありません。

代わりに、Mastodon は、コミュニティのさまざまなメンバーによってそれぞれが実行される何千もの異なるサーバーで動作します。 1 つのサーバーにサインアップしたユーザーは、他のネットワーク上のユーザーに簡単に接続し、インスタンス間で相互にフォローできます。

誰でも自分の Mastodon サーバーのインスタンスをインストールできます。このチュートリアルでは、Docker を使用して Ubuntu 22.04 を搭載したサーバーに Mastodon のインスタンスをセットアップする方法を説明します。 Docker は、必要なすべてのパッケージとサービスをコンテナーに含めることで、Mastodon を簡単にインストールできるようにします。

前提条件

- A server running Ubuntu 22.04 with a minimum of 2 CPU cores and 2GB of memory. You will need to upgrade the server as per requirements.
- A non-root user with sudo privileges.
- A fully qualified domain name (FQDN) pointing to your server. For our purposes, we will use `mastodon.example.com` as the domain name.
- Mastodon sends email notifications to users. We recommend you use a 3rd party Transactional mail service like Mailgun, Sendgrid, Amazon SES, or Sparkpost. The instructions in the guide will be using Amazon SES.
- Make sure everything is updated.

  ```
  $ sudo apt update
  ```

- Install basic utility packages. Some of them may already be installed.

  ```
  $ sudo apt install wget curl nano software-properties-common dirmngr apt-transport-https gnupg gnupg2 ca-certificates lsb-release ubuntu-keyring unzip -y
  ```

ステップ 1 - ファイアウォールの構成

最初のステップは、ファイアウォールを構成することです。 Ubuntu には、デフォルトで ufw (複雑でないファイアウォール) が付属しています。

ファイアウォールが実行されているかどうかを確認します。

```
$ sudo ufw status
```

次の出力が得られるはずです。

```
Status: inactive
```

SSH ポートを許可して、ファイアウォールを有効にしても現在の接続が切断されないようにします。

```
$ sudo ufw allow OpenSSH
```

HTTP および HTTPS ポートも許可します。

```
$ sudo ufw allow http
$ sudo ufw allow https
```

ファイアウォールを有効にする

```
$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

ファイアウォールの状態を再度確認してください。

```
$ sudo ufw status
```

同様の出力が表示されるはずです。

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443                        ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
```

ステップ 2 - Docker と Docker Compose をインストールする

Ubuntu 22.04 には、古いバージョンの Docker が付属しています。最新バージョンをインストールするには、まず Docker GPG キーをインポートします。

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

Docker リポジトリ ファイルを作成します。

```
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

システム リポジトリ リストを更新します。

```
$ sudo apt update
```

Docker の最新バージョンをインストールします。

```
$ sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

実行されていることを確認します。

```
$ sudo systemctl status docker
? docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2022-12-30 15:55:17 UTC; 29min ago
TriggeredBy: ? docker.socket
       Docs: https://docs.docker.com
   Main PID: 1966 (dockerd)
      Tasks: 8
     Memory: 20.8M
        CPU: 740ms
     CGroup: /system.slice/docker.service
             ??1966 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

デフォルトでは、Docker には root 権限が必要です。 `docker` コマンドを実行するたびに `sudo` を使用したくない場合は、ユーザー名を `docker` に追加してください。 > グループ。

```
$ sudo usermod -aG docker $(whoami)
```

この変更を有効にするか、次のコマンドを使用するには、サーバーからログアウトして同じユーザーとして再度ログインする必要があります。

```
$ su - ${USER}
```

ユーザーが Docker グループに追加されていることを確認します。

```
$ groups
navjot wheel docker
```

ステップ 3 - インストールの準備

Elasticsearch の mmap カウントのデフォルトの制限は非常に低いです。次のコマンドを実行して、デフォルト値を確認します。

```
$ sysctl vm.max_map_count
```

次の出力が得られます。

```
vm.max_map_count = 65530
```

次のコマンドを使用して値を増やします。

```
$ echo "vm.max_map_count=262144" | sudo tee /etc/sysctl.d/90-max_map_count.conf
$ sudo sysctl --load /etc/sysctl.d/90-max_map_count.conf
```

ステップ 4 - マストドンをインストールする

ディレクトリを作成して所有権を設定する

Mastodon および関連サービス用のディレクトリを作成します。

```
$ sudo mkdir -p /opt/mastodon/database/{postgresql,pgbackups,redis,elasticsearch}
$ sudo mkdir -p /opt/mastodon/web/{public,system}
$ sudo mkdir -p /opt/mastodon/branding
```

Elasticsearch、Web、およびバックアップ ディレクトリに適切な所有権を設定します。

```
$ sudo chown 991:991 /opt/mastodon/web/{public,system}
$ sudo chown 1000 /opt/mastodon/database/elasticsearch
$ sudo chown 70:70 /opt/mastodon/database/pgbackups
```

Mastodon ディレクトリに切り替えます。

```
$ cd /opt/mastodon
```

環境と docker 構成ファイルを作成する

アプリケーションとデータベースの環境ファイルを作成します。

```
$ sudo touch application.env database.env
```

編集用に Docker 構成ファイルを作成して開きます。

```
$ sudo nano docker-compose.yml
```

その中に次のコードを貼り付けます。

```yaml
version: '3'

services:
  postgresql:
    image: postgres:15-alpine
    env_file: database.env
    restart: always
    shm_size: 512mb
    healthcheck:
      test: ['CMD', 'pg_isready', '-U', 'postgres']
    volumes:
      - postgresql:/var/lib/postgresql/data
      - pgbackups:/backups
    networks:
      - internal_network

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

  redis-volatile:
    image: redis:7-alpine
    restart: always
    healthcheck:
      test: ['CMD', 'redis-cli', 'ping']
    networks:
      - internal_network

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.7
    restart: always
    env_file: database.env
    environment:
      - cluster.name=elasticsearch-mastodon
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - xpack.security.enabled=true
      - ingest.geoip.downloader.enabled=false
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m -Des.enforce.bootstrap.checks=true"
      - xpack.license.self_generated.type=basic
      - xpack.watcher.enabled=false
      - xpack.graph.enabled=false
      - xpack.ml.enabled=false
      - thread_pool.write.queue_size=1000
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    healthcheck:
      test: ["CMD-SHELL", "nc -z elasticsearch 9200"]
    volumes:
      - elasticsearch:/usr/share/elasticsearch/data
    networks:
      - internal_network
    ports:
      - '127.0.0.1:9200:9200'

  website:
    image: tootsuite/mastodon:v4.0.2
    env_file:
      - application.env
      - database.env
    command: bash -c "bundle exec rails s -p 3000"
```

```
    restart: always
    depends_on:
      - postgresql
      - redis
      - redis-volatile
      - elasticsearch
    ports:
      - '127.0.0.1:3000:3000'
    networks:
      - internal_network
      - external_network
    healthcheck:
      test: ['CMD-SHELL', 'wget -q --spider --proxy=off localhost:3000/health || exit 1']
    volumes:
      - uploads:/mastodon/public/system

  shell:
    image: tootsuite/mastodon:v4.0.2
    env_file:
      - application.env
      - database.env
    command: /bin/bash
    restart: "no"
    networks:
      - internal_network
      - external_network
    volumes:
      - uploads:/mastodon/public/system
      - static:/static

  streaming:
    image: tootsuite/mastodon:v4.0.2
    env_file:
      - application.env
      - database.env
    command: node ./streaming
    restart: always
    depends_on:
      - postgresql
      - redis
      - redis-volatile
      - elasticsearch
    ports:
      - '127.0.0.1:4000:4000'
    networks:
      - internal_network
      - external_network
    healthcheck:
      test: ['CMD-SHELL', 'wget -q --spider --proxy=off localhost:4000/api/v1/streaming/health || exit 1']

  sidekiq:
    image: tootsuite/mastodon:v4.0.2
    env_file:
      - application.env
      - database.env
    command: bundle exec sidekiq
    restart: always
    depends_on:
      - postgresql
      - redis
      - redis-volatile
      - website
    networks:
      - internal_network
      - external_network
    healthcheck:
      test: ['CMD-SHELL', "ps aux | grep '[s]idekiq\ 6' || false"]
    volumes:
      - uploads:/mastodon/public/system

networks:
  external_network:
```

```
  internal_network:
         internal:true

volumes:
  postgresql:
    driver_opts:
      type: none
      device: /opt/mastodon/database/postgresql
      o: bind
  pgbackups:
    driver_opts:
      type: none
      device: /opt/mastodon/database/pgbackups
      o: bind
  redis:
    driver_opts:
      type: none
      device: /opt/mastodon/database/redis
      o: bind
  elasticsearch:
    driver_opts:
      type: none
      device: /opt/mastodon/database/elasticsearch
      o: bind
  uploads:
    driver_opts:
      type: none
      device: /opt/mastodon/web/system
      o: bind
  static:
    driver_opts:
      type: none
      device: /opt/mastodon/web/public
      o: bind
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

チュートリアルを書いている時点で利用可能な最新バージョンの Mastodon は v4.0.2 です。 Mastodon GitHub リリース ページを確認し、Docker 構成ファイルのバージョンを適切に調整します。また、PostgreSQL と Redis の最新バージョンも使用しています。要件に応じて調整できます。現在、Elasticsearch 7.x を使用しています。 Docker Hub ページで追跡できる Elasticsearch のメジャー バージョンはないため、Java に関連するセキュリティ更新のために手動で更新し続ける必要があります。

アプリケーション シークレットの作成

次のステップは、アプリケーション シークレット値を作成することです。

次のコマンドを 2 回実行して、`SECRET_KEY_BASE` と `OTP_SECRET` の値を生成します。初回は画像の取り込みに時間がかかります。

```
$ docker compose run --rm shell bundle exec rake secret
```

同じために `openssl` ユーティリティを使用することもできます。

```
$ openssl rand -hex 64
```

次のコマンドを使用して、`VAPID_PRIVATE_KEY` および `VAPID_PUBLIC_KEY` の値を生成します。

```
$ docker compose run --rm shell bundle exec rake mastodon:webpush:generate_vapid_key
```

同様の出力が得られます。

```
VAPID_PRIVATE_KEY=u2qsCs5JdmdmMLnUuU0sgmFGvZedteJz-lFB_xF4_ac=
VAPID_PUBLIC_KEY=BJXjE2hIXvFpo6dnHqyf1i-2PcP-cBoL95UCmhhxwlAgtFw_vnrYp4GBneR7_cmI9LZUYjHFh-TBAPSb9WTqH9A=
```

`openssl` ユーティリティを使用して、PostgreSQL および Elasticsearch のパスワードを生成します。

```
$ openssl rand -hex 15
```

Mastodon 環境ファイル

`application.env` ファイルを開いて編集します。

```
$ sudo nano application.env
```

次の行を貼り付けます。

```
# environment
RAILS_ENV=production
NODE_ENV=production

# domain
LOCAL_DOMAIN=mastodon.example.com

# redirect to the first profile
SINGLE_USER_MODE=false

# do not serve static files
RAILS_SERVE_STATIC_FILES=false

# concurrency
WEB_CONCURRENCY=2
MAX_THREADS=5

# pgbouncer
#PREPARED_STATEMENTS=false

# locale
DEFAULT_LOCALE=en

# email, not used
SMTP_SERVER=email-smtp.us-west-2.amazonaws.com
SMTP_PORT=587
SMTP_LOGIN=AES_USER
SMTP_PASSWORD=AES_PWD


# secrets
SECRET_KEY_BASE=c09fa403575e0b431e54a2e228f20cd5a5fdfdbba0da80598959753b829a4e3c0266eedbac7e3cdf9f3345db36c56
OTP_SECRET=febb7dbb0d3308094083733fc923a430e52ccec767d48d7d2e0c577bfcb6863dbdfc920b1004b1f8c2967b9866bd7a0b4a

# Changing VAPID keys will break push notifications
VAPID_PRIVATE_KEY=13RgrfOY2tkwuUycylDPOkoHennkJ0ZAPV_fUwDy7-g=
VAPID_PUBLIC_KEY=BDAQuGwPbh1kbCV904adYXHvz9lLRaJHkiQkihRDPyBn3QmkAYbR21WHYoP8TkyG6dylG6IXpEVfLwdoW7fJVns=

# IP and session retention
# -----------------------
# Make sure to modify the scheduling of ip_cleanup_scheduler in config/sidekiq.yml
# to be less than daily if you lower IP_RETENTION_PERIOD below two days (172800).
# -----------------------
IP_RETENTION_PERIOD=2592000
SESSION_RETENTION_PERIOD=2592000
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

Amazon SES メール サービスを有効にしました。不要な場合は、セクションを削除できます。 Mastodon はデフォルトで IP アドレスを 1 年間保持しますが、30 日 (2592000 秒) に変更しました。必要に応じて変更できます。必ず 2 日以上保持してください。そうしないと、チュートリアルの範囲外である、もう少しいじくり回す必要があります。

`database.env` ファイルを編集用に開きます。

```
$ sudo nano database.env
```

次の行を貼り付けます。

```
# postgresql configuration
POSTGRES_USER=mastodon
POSTGRES_DB=mastodon
POSTGRES_PASSWORD=15ff12dcb93aa60680d2aadb4032ee
PGPASSWORD=15ff12dcb93aa60680d2aadb4032ee
PGPORT=5432
PGHOST=postgresql
PGUSER=mastodon

# pgbouncer configuration
#POOL_MODE=transaction
#ADMIN_USERS=postgres,mastodon
#DATABASE_URL="postgres://mastodon::5432/mastodon"

# elasticsearch
ES_JAVA_OPTS=-Xms512m -Xmx512m
ELASTIC_PASSWORD=13382e99f6b2d4dc7f3d66e4b9872d

# mastodon database configuration
#DB_HOST=pgbouncer
DB_HOST=postgresql
DB_USER=mastodon
DB_NAME=mastodon
DB_PASS=15ff12dcb93aa60680d2aadb4032ee
DB_PORT=5432

REDIS_HOST=redis
REDIS_PORT=6379

CACHE_REDIS_HOST=redis-volatile
CACHE_REDIS_PORT=6379

ES_ENABLED=true
ES_HOST=elasticsearch
ES_PORT=9200
ES_USER=elastic
ES_PASS=13382e99f6b2d4dc7f3d66e4b9872d
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

マストドンの準備

Nginx によって提供される準備ができている静的ファイルを取得します。 Docker が初めてすべてのイメージをプルするため、このステップには時間がかかります。

```
$ docker compose run --rm shell bash -c "cp -r /opt/mastodon/public/* /static/"
```

データ層を表示します。

```
$ docker compose up -d postgresql redis redis-volatile
```

コンテナの状態を確認します。

```
$ watch docker compose ps
```

`running (healthy)` を待ってから `Ctrl + C` を押し、次のコマンドを使用してデータベースを初期化します。

```
$ docker compose run --rm shell bundle exec rake db:setup
```

ステップ 5 - Nginx をインストールする

Ubuntu 22.04 には、古いバージョンの Nginx が付属しています。最新バージョンをインストールするには、公式の Nginx リポジトリをダウンロードする必要があります。

Nginxs 署名キーをインポートします。

```
$ curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
        | sudo tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null
```

Nginxs 安定版のリポジトリを追加します。

```
$ echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg arch=amd64] \
http://nginx.org/packages/ubuntu `lsb_release -cs` nginx" \
    | sudo tee /etc/apt/sources.list.d/nginx.list
```

システム リポジトリを更新します。

```
$ sudo apt update
```

Nginxをインストールします。

```
$ sudo apt install nginx
```

インストールを確認します。

```
$ nginx -v
nginx version: nginx/1.22.1
```

Nginx サーバーを起動します。

```
$ sudo systemctl start nginx
```

サーバーの状態を確認してください。

```
$ sudo systemctl status nginx
? nginx.service - nginx - high performance web server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2022-12-31 02:23:12 UTC; 6s ago
       Docs: https://nginx.org/en/docs/
    Process: 22129 ExecStart=/usr/sbin/nginx -c /etc/nginx/nginx.conf (code=exited, status=0/SUCCESS)
   Main PID: 22130 (nginx)
      Tasks: 3 (limit: 4575)
     Memory: 2.5M
        CPU: 17ms
     CGroup: /system.slice/nginx.service
             ??22130 "nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf"
             ??22131 "nginx: worker process" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
             ??22132 "nginx: worker process" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

ステップ 6 - SSL をインストールする

SSL 証明書を生成するには、Certbot をインストールする必要があります。 Ubuntu リポジトリを使用して Certbot をインストールするか、Snapd ツールを使用して最新バージョンを取得できます。 Snapdバージョンを使用します。

Ubuntu 22.04 には、デフォルトで Snapd がインストールされています。次のコマンドを実行して、Snapd のバージョンが最新であることを確認します。 Snapd のバージョンが最新であることを確認します。

```
$ sudo snap install core
$ sudo snap refresh core
```

証明書ボットをインストールします。

```
$ sudo snap install --classic certbot
```

次のコマンドを使用して、`/usr/bin` ディレクトリへのシンボリック リンクを作成することにより、Certbot コマンドが確実に実行されるようにします。

```
$ sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

次のコマンドを実行して、SSL 証明書を生成します。

```
$ sudo certbot certonly --nginx --agree-tos --no-eff-email --staple-ocsp --preferred-challenges http -m  -d
mastodon.example.com
```

上記のコマンドは、証明書をサーバーの `/etc/letsencrypt/live/mastodon.example.com` ディレクトリにダウンロードします。

Diffie-Hellman グループ証明書を生成します。

```
$ sudo openssl dhparam -dsaparam -out /etc/ssl/certs/dhparam.pem 4096
```

SSL の更新が正常に機能しているかどうかを確認するには、プロセスの予行演習を行います。

```
$ sudo certbot renew --dry-run
```

エラーが表示されない場合は、すべて設定されています。証明書は自動的に更新されます。

ステップ 7 - Nginx を構成する

ファイル `/etc/nginx/nginx.conf` を開いて編集します。

```
$ sudo nano /etc/nginx/nginx.conf
```

`include /etc/nginx/conf.d/*.conf;` 行の前に次の行を追加します。

```
server_names_hash_bucket_size  64;
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

ファイル `/etc/nginx/conf.d/mastodon.conf` を作成して開き、編集します。

```
$ sudo nano /etc/nginx/conf.d/mastodon.conf
```

その中に次のコードを貼り付けます。

```
map $http_upgrade $connection_upgrade {
  default upgrade;
  ''      close;
}

upstream backend {
    server 127.0.0.1:3000 fail_timeout=0;
}

upstream streaming {
    server 127.0.0.1:4000 fail_timeout=0;
}

proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=CACHE:10m inactive=7d max_size=1g;

server {
  listen 80 default_server;
  server_name mastodon.example.com;
  location / { return 301 https://$host$request_uri; }
}

server {
   listen 443 ssl http2;
   server_name mastodon.example.com;

   access_log  /var/log/nginx/mastodon.access.log;
   error_log   /var/log/nginx/mastodon.error.log;

   http2_push_preload on; # Enable HTTP/2 Server Push

   ssl_certificate /etc/letsencrypt/live/mastodon.example.com/fullchain.pem;
   ssl_certificate_key /etc/letsencrypt/live/mastodon.example.com/privkey.pem;
   ssl_trusted_certificate /etc/letsencrypt/live/mastodon.example.com/chain.pem;
   ssl_session_timeout 1d;

   # Enable TLS versions (TLSv1.3 is required upcoming HTTP/3 QUIC).
   ssl_protocols TLSv1.2 TLSv1.3;

   # Enable TLSv1.3's 0-RTT. Use $ssl_early_data when reverse proxying to
   # prevent replay attacks.
   #
   # @see: https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_early_data
   ssl_early_data on;

   ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384';
   ssl_prefer_server_ciphers on;
   ssl_session_cache shared:SSL:10m;
   ssl_session_tickets off;

   keepalive_timeout    70;
   sendfile             on;
   client_max_body_size 80m;

   # OCSP Stapling ---
   # fetch OCSP records from URL in ssl_certificate and cache them
   ssl_stapling on;
   ssl_stapling_verify on;
   ssl_dhparam /etc/ssl/certs/dhparam.pem;

   add_header X-Early-Data $tls1_3_early_data;

   root /opt/mastodon/web/public;

   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;

   add_header Strict-Transport-Security "max-age=31536000" always;

  location / {
    try_files $uri @proxy;
  }

  location ~ ^/(system/accounts/avatars|system/media_attachments/files) {
    add_header Cache-Control "public, max-age=31536000, immutable";
    add_header Strict-Transport-Security "max-age=31536000" always;
    root /opt/mastodon/;
    try_files $uri @proxy;
  }

  location ~ ^/(emoji|packs) {
    add_header Cache-Control "public, max-age=31536000, immutable";
    add_header Strict-Transport-Security "max-age=31536000" always;
    try_files $uri @proxy;
  }

  location /sw.js {
    add_header Cache-Control "public, max-age=0";
    add_header Strict-Transport-Security "max-age=31536000" always;
    try_files $uri @proxy;
  }

  location @proxy {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Proxy "";
    proxy_pass_header Server;

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

    proxy_cache CACHE;
    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;
    add_header Strict-Transport-Security "max-age=31536000" always;

    tcp_nodelay on;
  }

  location /api/v1/streaming {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Proxy "";

    proxy_pass http://streaming;
    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;
  }

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

```
# This block is useful for debugging TLS v1.3. Please feel free to remove this
# and use the `$ssl_early_data` variable exposed by NGINX directly should you
# wish to do so.
map $ssl_early_data $tls1_3_early_data {
  "~." $ssl_early_data;
  default "";
}
```

完了したら、Ctrl + X を押してファイルを保存し、プロンプトが表示されたら Y を入力します。

Nginx 構成ファイルの構文を確認します。

```
$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Nginx サーバーを再起動します。

```
$ sudo systemctl restart nginx
```

ステップ 8 - マストドンを起動する

Tootctl CLI ツール

Tootctl CLI ツールは、Mastodon で管理タスクを実行するために使用されます。ホストシェルでアクセスできるようにする必要があります。

ファイル `/usr/local/bin/tootctl` を作成し、編集用に開きます。

```
$ sudo nano /usr/local/bin/tootctl
```

その中に次のコードを貼り付けます。

```
#!/bin/bash
docker compose -f /opt/mastodon/docker-compose.yml run --rm shell tootctl ""
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

ファイルに実行権限を付与します。

```
$ sudo chmod +x /usr/local/bin/tootctl
```

マストドンサービスファイル

Docker compose コマンドを使用して Mastodon コンテナーを開始できますが、systemd ユニット ファイルを使用する方が簡単です。

Mastodon サービス ファイルを作成して開き、編集します。

```
$ sudo nano /etc/systemd/system/mastodon.service
```

その中に次のコードを貼り付けます。

```
[Unit]
Description=Mastodon service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes

WorkingDirectory=/opt/mastodon
ExecStart=/usr/bin/docker compose -f /opt/mastodon/docker-compose.yml up -d
ExecStop=/usr/bin/docker compose -f /opt/mastodon/docker-compose.yml down

[Install]
WantedBy=multi-user.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

システム デーモンをリロードして、サービス ファイルを開始します。

```
$ sudo systemctl daemon-reload
```

Mastodon サービスを有効にして開始します。

```
$ sudo systemctl enable --now mastodon.service
```

Docker コンテナのステータスを確認します。

```
$ watch docker compose -f /opt/mastodon/docker-compose.yml ps
```

コンテナーのステータスが `running (healthy)` に変わったら、Ctrl + C を押して画面を終了します。

Mastodon の管理者ユーザーを作成し、提供されたパスワードをメモします。

```
$ tootctl accounts create navjot --email  --confirmed --role Owner
OK
New password: 1338afbe1b4e06e823b6625da80cb537
```

ユーザー登録をクローズする場合は、次のコマンドを使用します。

```
$ tootctl settings registrations close
```

登録を再度開くには、次のコマンドを発行します。

```
$ tootctl settings registrations open
```

検索の初期化

Elasticsearch インデックスを作成して入力する前に、ツールを作成する必要があります。トゥートを作成したら、次のコマンドを発行します。

```
$ tootctl search deploy
```

次のエラーが表示される場合があります。

```
/opt/mastodon/vendor/bundle/ruby/3.0.0/gems/ruby-progressbar-1.11.0/lib/ruby-progressbar/progress.rb:76:in
`total=': You can't set the item's total value to less than the current progress.
(ProgressBar::InvalidProgressError)
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/ruby-progressbar-1.11.0/lib/ruby-
progressbar/base.rb:178:in `block in update_progress'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/ruby-progressbar-1.11.0/lib/ruby-
progressbar/output.rb:43:in `with_refresh'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/ruby-progressbar-1.11.0/lib/ruby-
progressbar/base.rb:177:in `update_progress'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/ruby-progressbar-1.11.0/lib/ruby-
progressbar/base.rb:101:in `total='
        from /opt/mastodon/lib/mastodon/search_cli.rb:67:in `deploy'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/command.rb:27:in `run'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/invocation.rb:127:in
`invoke_command'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor.rb:392:in `dispatch'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/invocation.rb:116:in `invoke'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor.rb:243:in `block in subcommand'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/command.rb:27:in `run'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/invocation.rb:127:in
`invoke_command'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor.rb:392:in `dispatch'
        from /opt/mastodon/vendor/bundle/ruby/3.0.0/gems/thor-1.2.1/lib/thor/base.rb:485:in `start'
        from /opt/mastodon/bin/tootctl:8:in `<main>'
```

この場合、Web サイトのコンテナー シェルに入ります。

```
$ docker exec -it mastodon-website-1 /bin/bash
```

次のコマンドを実行します。

```
$ sed -E '/progress.total = /d' -i lib/mastodon/search_cli.rb
```

コンテナ シェルを終了します。

```
$ exit
```

Elasticsearch deploy コマンドを再度実行します。コマンドが後で機能する場合があります。これはマストドンで進行中の問題であるため、現時点では明確な修正はありません。

```
$ tootctl search deploy
```

追加のヘルパー サービス

ダウンロードしたメディア ファイルを削除するための別のサービスを作成してみましょう。

Mastodon メディア削除サービスを作成して開き、編集します。

```
$ sudo nano /etc/systemd/system/mastodon-media-remove.service
```

その中に次のコードを貼り付けます。

```
[Unit]
Description=Mastodon - media remove service
Wants=mastodon-media-remove.timer

[Service]
Type=oneshot
StandardError=null
StandardOutput=null

WorkingDirectory=/opt/mastodon
ExecStart=/usr/bin/docker compose -f /opt/mastodon/docker-compose.yml run --rm shell tootctl media remove

[Install]
WantedBy=multi-user.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

メディアの削除をスケジュールする場合は、タイマー サービスを設定できます。

```
$ sudo nano /etc/systemd/system/mastodon-media-remove.timer
```

次のコードを貼り付けます。

```
[Unit]
Description=Schedule a media remove every week

[Timer]
Persistent=true
OnCalendar=Sat *-*-* 00:00:00
Unit=mastodon-media-remove.service

[Install]
WantedBy=timers.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

別のサービスをセットアップして、OpenGraph タグを使用して生成されたリッチ プレビュー カードを削除できます。

```
$ sudo nano /etc/systemd/system/mastodon-preview_cards-remove.service
```

次のコードを貼り付けます。

```
[Unit]
Description=Mastodon - preview cards remove service
Wants=mastodon-preview_cards-remove.timer

[Service]
Type=oneshot
StandardError=null
StandardOutput=null

WorkingDirectory=/opt/mastodon
ExecStart=/usr/bin/docker compose -f /opt/mastodon/docker-compose.yml run --rm shell tootctl preview_cards
remove

[Install]
WantedBy=multi-user.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

対応するタイマー サービスを設定します。

```
$ sudo nano /etc/systemd/system/mastodon-preview_cards-remove.timer
```

次のコードを貼り付けます。

```
[Unit]
Description=Schedule a preview cards remove every week

[Timer]
Persistent=true
OnCalendar=Sat *-*-* 00:00:00
Unit=mastodon-preview_cards-remove.service

[Install]
WantedBy=timers.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

システム デーモンをリロードします。

```
$ sudo systemctl daemon-reload
```

タイマーを有効にして開始します。

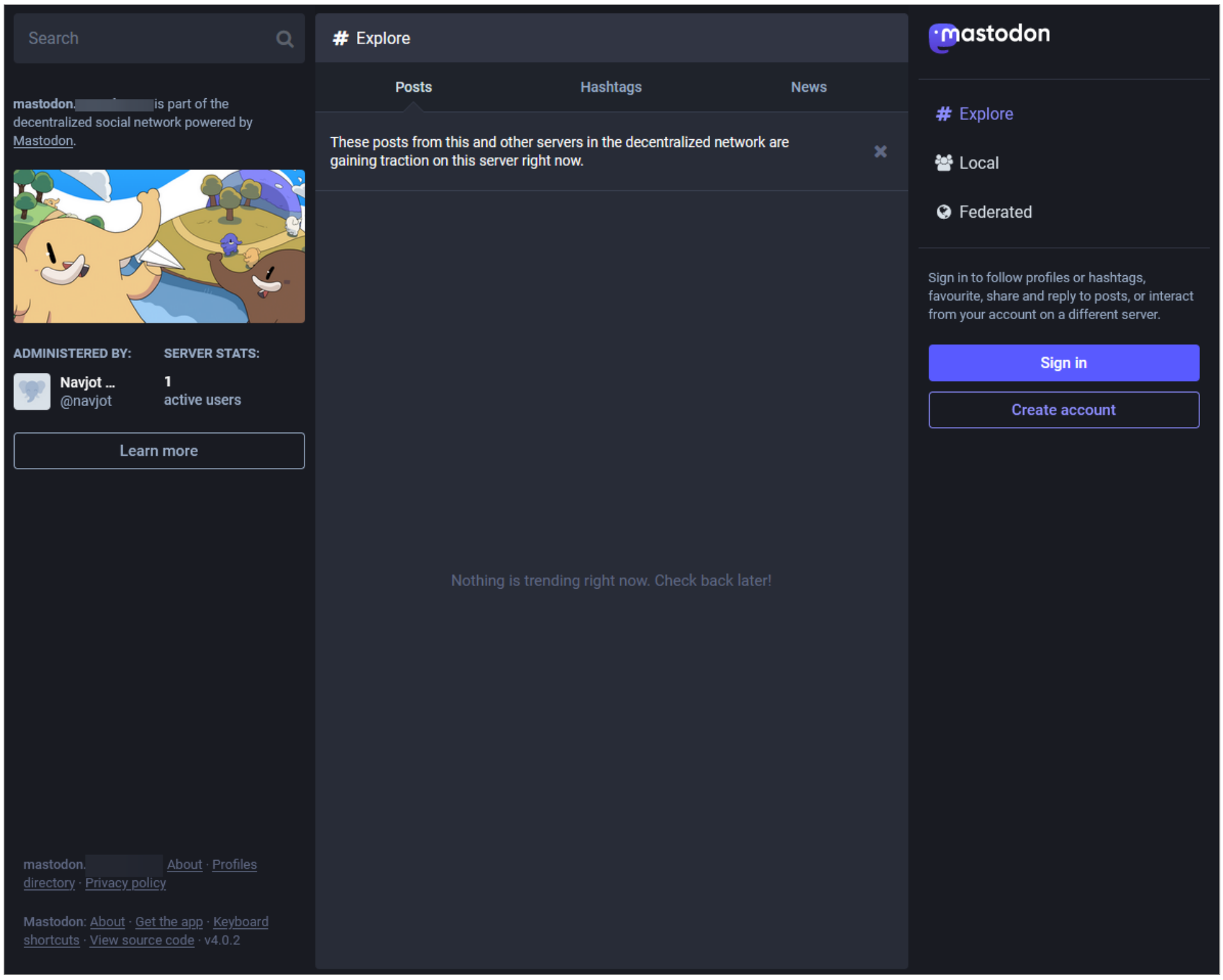
```
$ sudo systemctl enable --now mastodon-preview_cards-remove.timer
$ sudo systemctl enable --now mastodon-media-remove.timer
```

すべてのタイマーをリストして、Mastodon サービスのスケジュールを確認します。

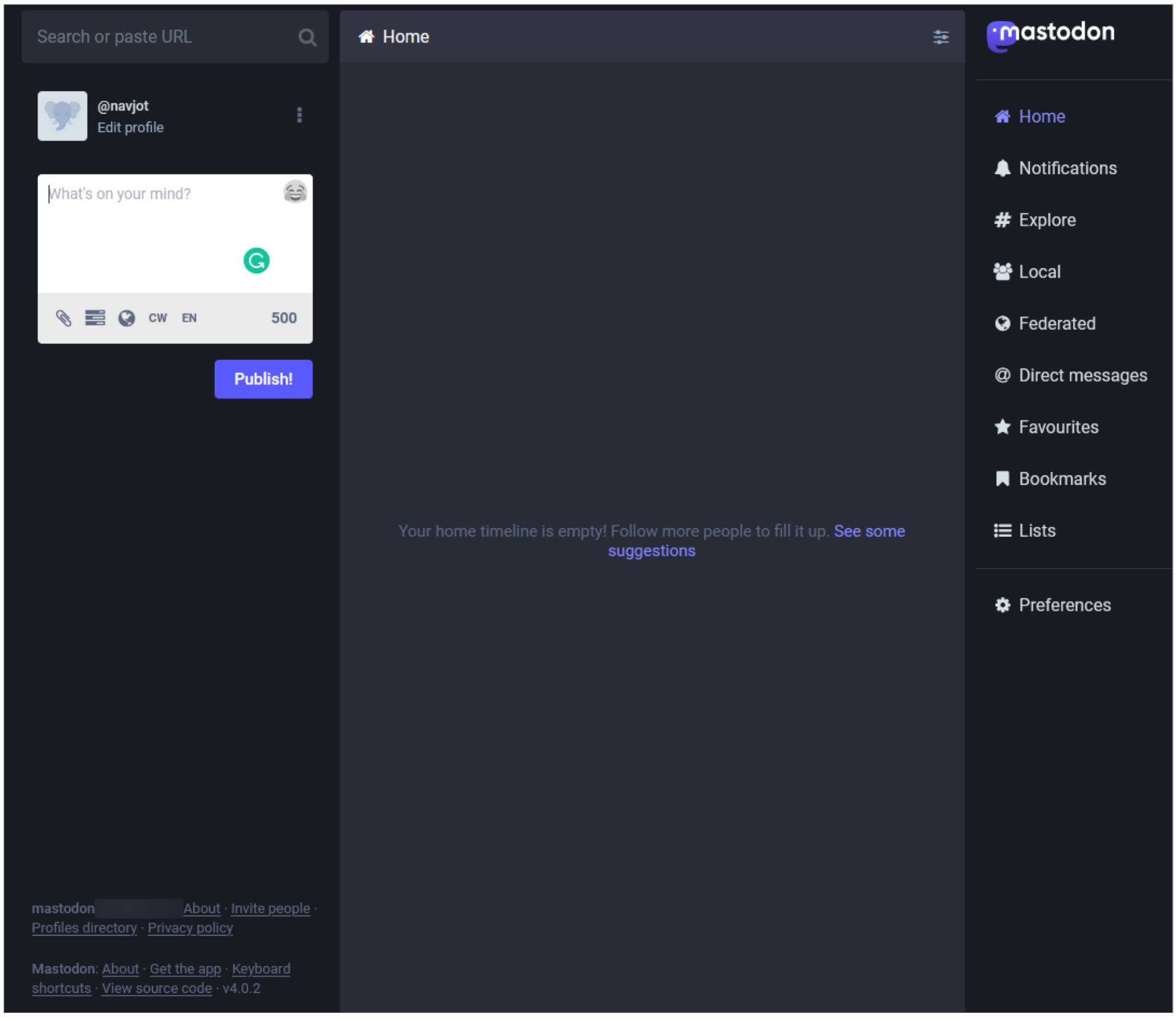
```
$ systemctl list-timers
.....
Sat 2023-01-07 00:00:00 UTC 6 days left    n/a                          n/a                mastodon-media-
remove.timer         mastodon-media-remove.service
Sat 2023-01-07 00:00:00 UTC 6 days left    n/a                          n/a                mastodon-
preview_cards-remove.timer mastodon-preview_cards-remove.service
```

アクセス マストドン

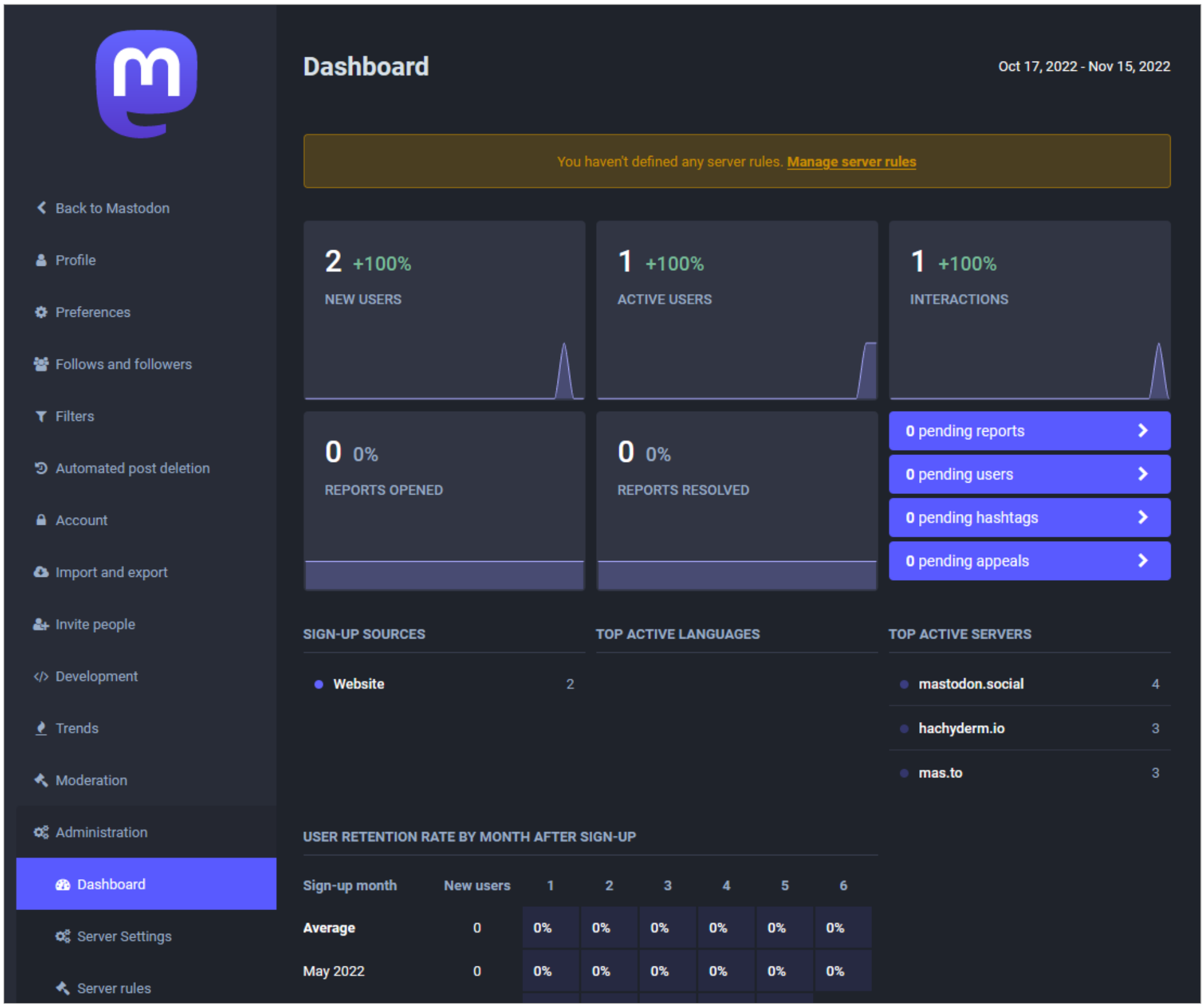
URL `https://mastodon.example.com` にアクセスしてインスタンスにアクセスすると、同様のページが表示されます。



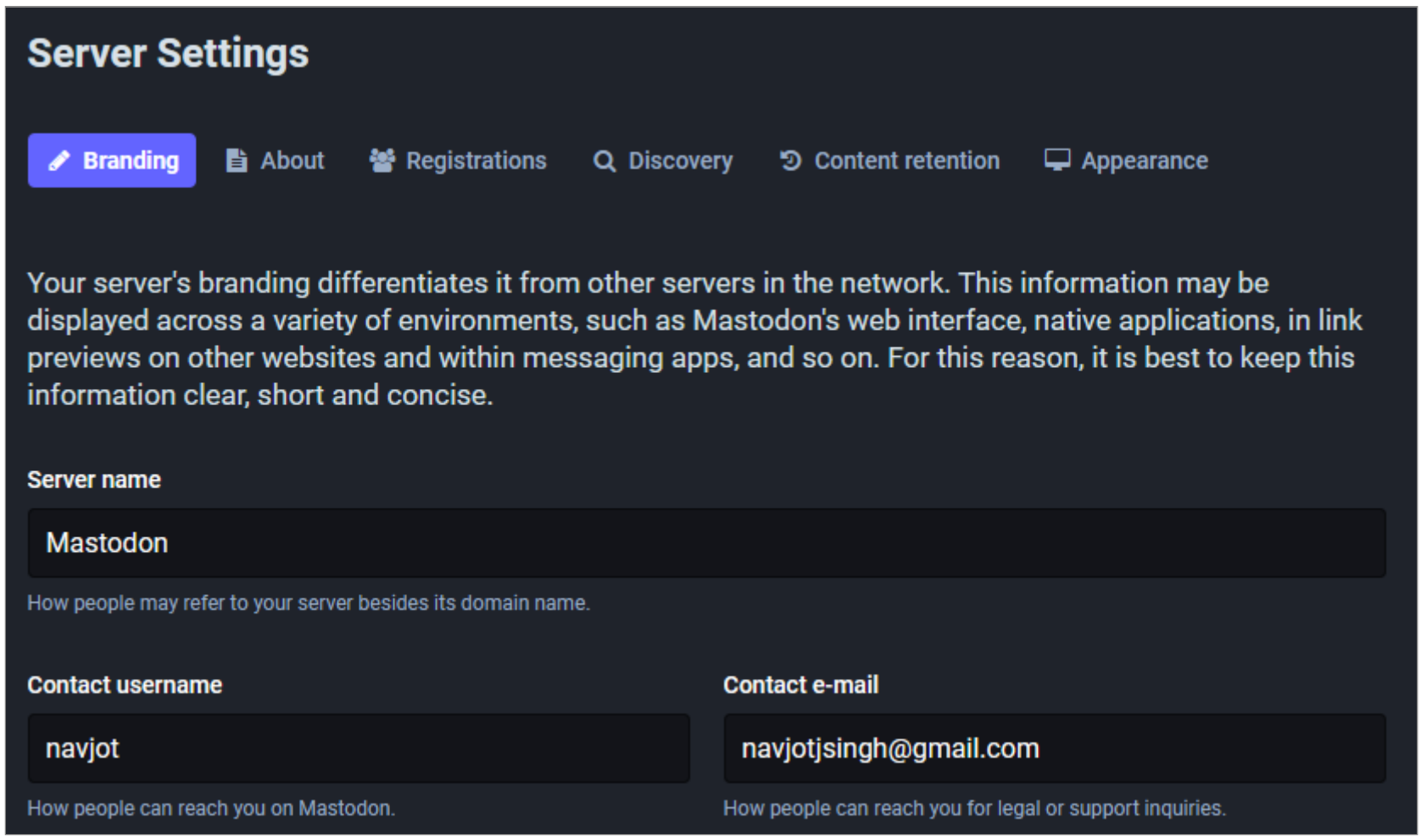
上のスクリーンショットでは、2 人のユーザーがいて、そのうちの 1 人 (私) が管理者として設定されていることがわかります。これは通常は当てはまりません。管理者アカウントを作成しても、初回実行時にはメイン ページに表示されません。これを行うには、インスタンスにログインすると、次のページに移動します。



右側のサイドバーから [設定] オプションをクリックして、設定にアクセスします。そこから、左側のメニューから [管理] オプションをクリックして、Mastodons 管理パネルにアクセスします。



左側のサイドバーから [サイトの設定] オプションをクリックします。

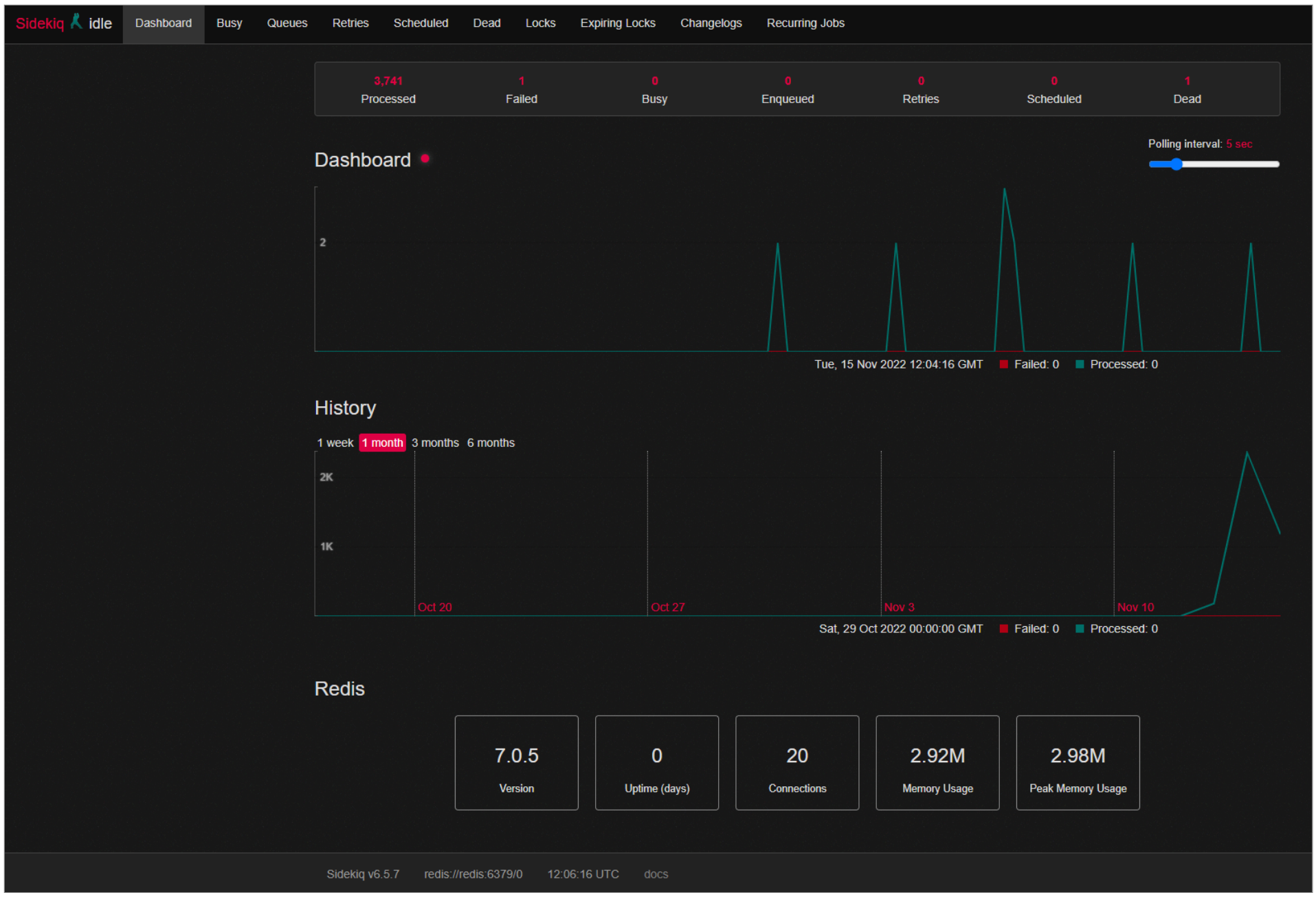


ここで、サーバーのホームページに反映される連絡先のユーザー名とビジネス用の電子メールを入力します。また、サーバーの説明、ロゴ、サーバー ルールなど、その他のさまざまな情報を入力して、Mastodon インスタンスをカスタマイズします。

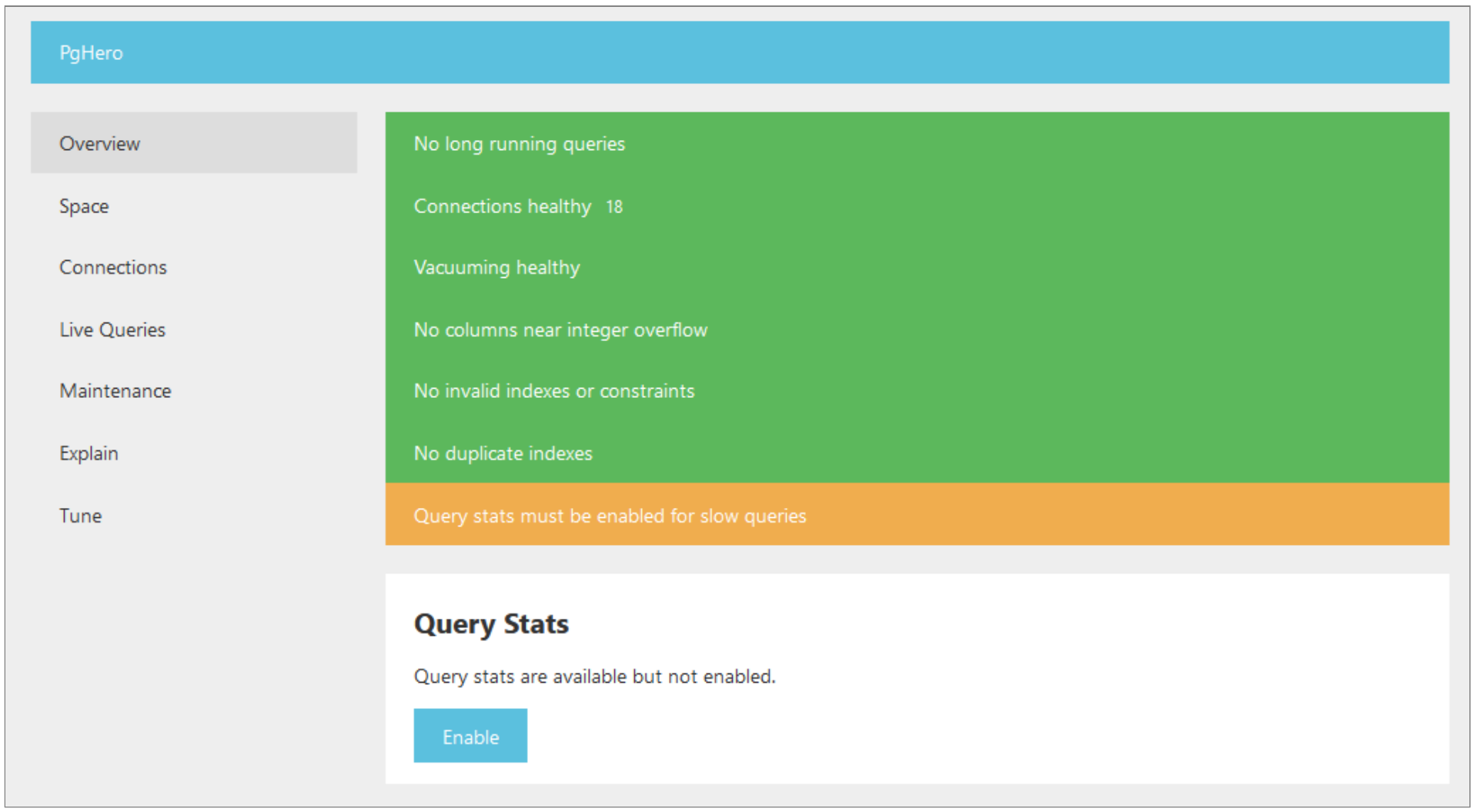
ステップ 9 - マストドンのメンテナンス

Mastodon インスタンスのパフォーマンスとログを表示するには、`https://mastodon.example.com/sidekiq/` にアクセスしてください。

ここでは、Mastodon インスタンスに関連するさまざまなプロセスとスケジュールされたタスクのリストを表示できます。 Dead または Retries セクションで、失敗したタスクを確認することもできます。インスタンスのメモリ使用量も表示されます。



`https://mastodon.example.com/pghero/` からインスタンス データベースの状態を確認できます。



データベースのメンテナンスを実行し、SQL クエリを実行し、未使用のインデックスを削除できます。クエリ統計を有効にするには、上記のページで [有効にする] ボタンをクリックすると、次の情報が表示されます。

Query Stats

Make them available by adding the following lines to postgresql.conf:

```
shared_preload_libraries = 'pg_stat_statements'
pg_stat_statements.track = all
```

Restart the server for the changes to take effect.

root ユーザーに切り替えます。

```
$ sudo -i su
```

`/opt/mastodon/database/postgresql` ディレクトリに切り替えます。

```
$ cd /opt/mastodon/database/postgresql
```

`postgresql.conf` ファイルを開きます。

```
$ nano postgresql.conf
```

行 `#shared_preload_libraries=# (change requires restart)` を見つけて、次のように置き換えます。

```
shared_preload_libraries = 'pg_stat_statements'
```

ファイルの最後に次の行を追加します。

```
pg_stat_statements.track = all
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

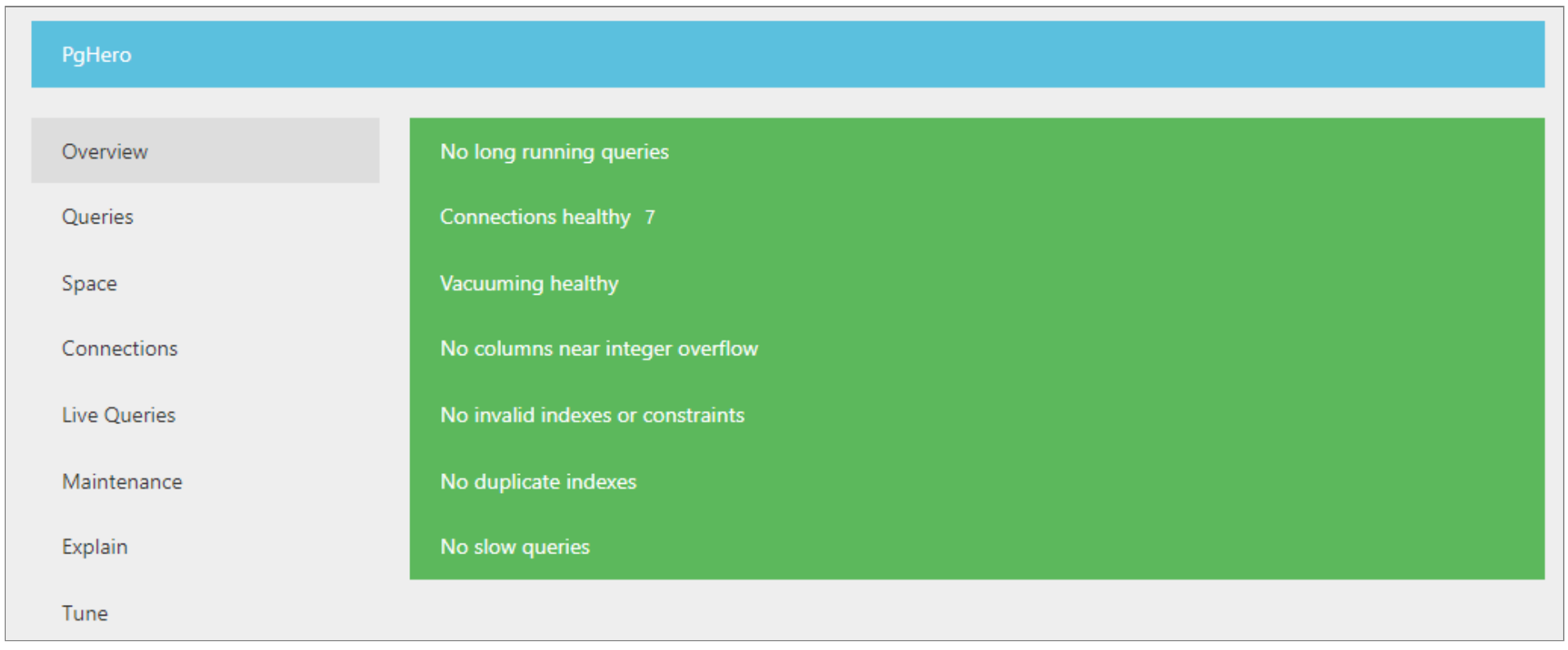
Mastodon コンテナを再起動します。

```
$ systemctl restart mastodon.service
```

ルート シェルを終了します。

```
$ exit
```

データベースの正常性ページを確認すると、現在低速のクエリがあるかどうかを確認できます。



注: [設定] メニューから PgHero および Sidekiq の URL を起動することもできます。

何らかの理由でサイトが読み込まれない場合は、Docker によって生成されたログを確認できます。

```
$ docker logs <container-name>
```

ステップ 10 - マストドンのバックアップ

Mastodon のバックアップには Restic というサードパーティ ツールを使用します。 Restic を使用してバックアップする最初のステップは、すべてのファイルとディレクトリをリポジトリ リストに追加することです。

リポジトリ リスト ファイルを作成し、編集用に開きます。

```
$ sudo nano /opt/mastodon/backup-files
```

次の行を貼り付けます。

```
/etc/nginx
/etc/letsencrypt
/etc/systemd/system
/root
/opt/mastodon/database/pgbackups
/opt/mastodon/*.env
/opt/mastodon/docker-compose.yml
/opt/mastodon/branding
/opt/mastodon/database/redis
/opt/mastodon/web/system
/opt/mastodon/backup-files
/opt/mastodon/mastodon-backup
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

レスティックをインストールします。

```
$ sudo apt install restic
```

バックアップ リポジトリを作成し、初期バックアップを作成します。データを S3 サービスにバックアップしています。

```
$ restic -r s3:https://$SERVER:$PORT/mybucket init
$ restic -r s3:https://$SERVER:$PORT/mybucket backup $(cat /opt/mastodon/backup-files) --exclude
/opt/mastodon/database/postgresql
```

Mastodon バックアップ サービス タイマーを作成し、編集用に開きます。

```
$ sudo nano /etc/systemd/system/mastodon-backup.timer
```

その中に次のコードを貼り付けます。

```
[Unit]
Description=Schedule a mastodon backup every hour

[Timer]
Persistent=true
OnCalendar=*:00:00
Unit=mastodon-backup.service

[Install]
WantedBy=timers.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

Mastodon バックアップ サービス ファイルを作成し、編集用に開きます。

```
$ sudo nano /etc/systemd/system/mastodon-backup.service
```

その中に次のコードを貼り付けます。

```
[Unit]
Description=Mastodon - backup service
# Without this, they can run at the same time and race to docker compose,
# double-creating networks and failing due to ambiguous network definition
# requiring `docker network prune` and restarting
After=mastodon.service

[Service]
Type=oneshot
StandardError=file:/var/log/mastodon-backup.err
StandardOutput=file:/var/log/mastodon-backup.log

WorkingDirectory=/opt/mastodon
ExecStart=/bin/bash /opt/mastodon/mastodon-backup

[Install]
WantedBy=multi-user.target
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

次に、編集用に `/opt/mastodon/mastodon-backup` ファイルを作成して開きます。これには、実際のバックアップ コマンドが含まれます。

```
$ sudo nano /opt/mastodon/mastodon-backup
```

その中に次のコードを貼り付けます。

```
#!/bin/bash

set -e

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
SERVER=
PORT=
RESTIC_PASSWORD_FILE=/root/restic-pasword

docker compose -f /opt/mastodon/docker-compose.yml run --rm postgresql sh -c "pg_dump -Fp  mastodon | gzip >
/backups/dump.sql.gz"
restic -r s3:https://$SERVER:$PORT/mybucket --cache-dir=/root backup $(cat /opt/mastodon/backup-files) --
exclude  /opt/mastodon/database/postgresql
restic -r s3:https://$SERVER:$PORT/mybucket --cache-dir=/root forget --prune --keep-hourly 24 --keep-daily 7
--keep-monthly 3
```

Ctrl + X を押し、プロンプトが表示されたら Y を入力して、ファイルを保存します。

バックアップ スクリプトに実行権限を付与します。

```
$ sudo chmod +x /opt/mastodon/mastodon-backup
```

サービス デーモンをリロードし、バックアップ サービスとタイマーを開始します。

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now mastodon-backup.service
$ sudo systemctl enable --now mastodon-backup.timer
```

次のコマンドを使用して、1 時間ごとのバックアップが実行され、アクセスできることを確認します。

```
$ restic -r s3:https://$SERVER:$PORT/mybucket snapshots
$ restic -r s3:https://$SERVER:$PORT/mybucket mount /mnt
```

ステップ 11 - マストドンのアップグレード

Mastodon のアップグレードにはいくつかの手順が必要です。まず、ディレクトリに切り替えます。

```
$ cd /opt/mastodon
```

Mastodon の最新のコンテナー イメージをプルします。

```
$ docker compose pull
```

必要に応じて、`docker-compose.yml` に変更を加えます。

すべてのデータベース移行を実行します。

```
$ docker compose run --rm shell bundle exec rake db:migrate
```

静的ファイルのコピーを更新します。

```
$ docker compose run --rm shell bash -c "cp -r /opt/mastodon/public/* /static/"
```

Mastodon コンテナを再起動します。

```
$ sudo systemctl restart mastodon.service
```

上記の手順は、一般的な更新手順です。 Mastodon の GitHub リリース ページを常にチェックして、バージョン間の特定の更新タスクとコマンドを探し、すべてがスムーズに進むようにしてください。

結論

これで、Docker を使用して Ubuntu 22.04 サーバーに Mastodon Social Network をインストールするチュートリアルは終了です。質問がある場合は、以下のコメントに投稿してください。