How to run a #Mastodon server using #Docker (and docker-compose)

This is based on this guide that I used to set my Mastodon server, but without the parts about building from source, and with some parts I was able to smooth out in my experience.

What you need to start

Ensure you have a machine that is running a recent, stable version of docker and docker-compose. If you're not there yet, there are a bunch of ways this can be accomplished, so I'll leave you to google that.

Make a folder that will contain all the volumes that we're going to create. I called this `mastodon`, and put it in the same folder as all the other folders for services that I run via docker. Change into this directory so that all the subsequent commands will be in that scope. Get the default `docker-compose.yml` file to start with and put it in your `mastodon` folder. I used this one from the guide linked at the top.

Choosing a Mastodon image

In general, i think it's good to take the latest and greatest version of whatever software you're going to use. Version 4.0.1 of Mastodon has just been released, so I changed the lines that read:

```
    build: .
    image: tootsuite/mastodon
```

to read the following instead

```
    image: tootsuite/mastodon:v4.0
```

Having a tag at the end of the image name will specifically reference a particular tagged instance of that image. without a tag, you'll get `tootsuite/mastodon:latest` by default. I chose my particular tag because I don't want the underlying image to change without me specifically changing it. the `latest` tag will change whenever a new stable version is released, but since there are upgrade procedures associated with each new version, I want upgrading to be a manual process that do purposefully. However, not including the patch version means that we'll get whatever the latest patch of v4.0 is, and those minor updates are typically safe and usually only contain bug fixes.

Docker Configuration

Get the sample environment variable file from the repository. Save it in your `mastodon` folder as `.env.production` (i.e., remove `.sample` from the end of the filename). Don't worry about the particulars yet...we'll get to that.

Make the following folders under your `mastodon` folder:

- `postgres14`
- `redis`
- `elasticsearch` *(only if you plan to use elastic search)*
- `system`
- `assets` Each of these folders will be used to hold some persistent data that the containers will produce and use. If you don't persist data in containers, then it disappears with the container.

I modified the `docker-compose.yml` file here to reference my folders so instead of

```
    volumes:
      - ./public/system:/mastodon/public/system
```

on the containers that use the `tootsute\mastodon` image, we'll have

```
    volumes:
      - ./system:/mastodon/public/system
      - ./assets:/mastodon/public/assets
```

Lastly, we need to set these folders to be owned by the user that is going to run the mastodon services. The following command will do that:

```
sudo chown -R 991:991 ./system ./assets
```

This may ask you for your root password.

Mastodon configuration

Finally we're ready to run something. The following command will start up the `web` container and its dependencies in order to run mastodon's setup script. You should be able to answer with the default for most things. I had to change the default database name from `postgres` to `mastodon_production` to match what is in the `docker-compose.yml` file.

```
docker-compose run --rm web rake: mastodon:setup
```

This command will output a file of environment variables that you'll want to merge with the ones in the copied `env.production` sample file that should already be in your `mastodon` folder.

Run your mastodon server!

Run the entire stack of containers with the command

```
docker-compose up -d
```

The `-d` puts the containers in the background to run as a daemon, but if you wish to run the stack synchronously to see any errors that might occur, you can omit the `-d`.

With the `docker-compose.yml` file as it currently is, The website is exposed only on localhost, so you won't be able to even access it over the network. This is intended for a reverse proxy to serve it up to the public, I would highly recommend this because Mastodon doesn't do SSL by default. I'm not going to explain how to do this here, because that deserves its own tutorial, but there are many of them online. You can choose between one of several different bits of software that will all do the trick. In my case, my reverse proxy is on a separate machine, so I needed to expose the mastodon port to the local network. In order to do this, I changed the port configuration of `web` container from

```
    ports:
      - '127.0.0.1:3000:3000'
```

to

```
    ports:
      - 8030:3000
```

because I wanted to expose mastodon to all network interfaces on the machine using the public port 8030, while keeping the docker container's port as 3000.

Running `docker-compose up -d` will refresh all containers with changed configuration.

Upgrading mastodon

There will come a time when you want to upgrade your mastodon server to a newer version. However, don't get hasty. The first thing you'll always want to do is back up your `mastodon` folder. You can do this easily by tarring and gzipping everything into an archive file. First, bring your mastodon server down so we're not trying to zip up a database that is currently changing.

```
docker-compose down
```

Then zip up the current `mastodon` folder and save it as a tar.gz file in the parent folder.

```
tar -zcf ../mastodon.tar.gz .
```

Next, edit the `docker-compose.yml` file to reflect the new tag for the version you're upgrading to. Finally we'll need to run some commands to download the new image(s), upgrade the database, ~~and pre-compile the assets~~. *I've learned that the assets are pre-compiled for you in the docker container version and this step is only necessary when you compile from source.*

```
docker-compose pull
docker-compose run --rm web rake db:migrate
```

Then a final `docker-compose up -d` and we should be running the upgraded version.

If I've missed anything, or made an grevious errors, please let me know. You can find me on mastodon at ***@steve@social.dinn.ca***

Appendix 1: My docker-compose.yml

```
version: '3'

networks:
  external_network:
  internal_network:
```

```
    internal: true

services:
  db:
    restart: always
    hostname: db
    image: postgres:14-alpine
    shm_size: 256mb
    networks:
      - internal_network
    volumes:
      - ./postgres14:/var/lib/postgresql/data
    environment:
      POSTGRES_HOST_AUTH_METHOD: "trust"
      POSTGRES_DB: "mastodon_production"
      POSTGRES_USER: "mastodon"
      POSTGRES_PASSWORD: "[Initial Postgres password]"

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

  es:
    restart: always
    hostname: es
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.4
    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
    ports:
      # Don't think this port needs to be exposed
      - '9200:9200'

  web:
    #build: .
    hostname: mastodon-web
    image: tootsuite/mastodon:v4.0
    restart: always
    env_file: .env.production
    command: bash -c "rm -f /mastodon/tmp/pids/server.pid; bundle exec rails s -p 3000"
    networks:
      - external_network
      - internal_network
    ports:
      - '8030:3000'
```

```
    depends_on:
      - db
      - redis
      - es
    volumes:
      - ./system:/mastodon/public/system
      - ./assets:/mastodon/public/assets
    environment:
      RAILS_ENV: "production"
      NODE_ENV: "production"

  streaming:
    #build: .
    hostname: mastodon-streaming
    image: tootsuite/mastodon:v4.0
    restart: "no"
    env_file: .env.production
    command: node ./streaming
    networks:
      - external_network
      - internal_network
    volumes:
      - ./system:/mastodon/public/system
      - ./assets:/mastodon/public/assets
    environment:
      RAILS_ENV: "production"
      NODE_ENV: "production"
    ports:
      # Don't think this port needs to be exposed.
      - '8031:4000'
    depends_on:
      - db
      - redis

  sidekiq:
    #build: .
    hostname: mastodon-sidekiq
```

```
    image: tootsuite/mastodon:v4.0
    restart: always
    env_file: .env.production
    command: bundle exec sidekiq
    depends_on:
      - db
      - redis
    networks:
      - external_network
      - internal_network
    volumes:
      - ./system:/mastodon/public/system
      - ./assets:/mastodon/public/assets
    environment:
      RAILS_ENV: "production"
      NODE_ENV: "production"
```

Appendix 2: Scaling Mastodon

Link: Scaling Mastodon in the face of an exodus

Revision #17
Created 2022-11-13 13:28:04 AST by Steve Dinn
Updated 2023-05-04 12:05:08 ADT by Steve Dinn