Steve's Blog

- How to mirror the list of server blocks from another Mastodon instance
- How many Mastodon instances are out there?
- How to run a #Mastodon server using #Docker (and docker-compose)
- Kitchen Chairs

How to mirror the list of server blocks from another Mastodon instance

If you're bringing up a new Mastodon server, there will probably come a time when you want, or have to block another server. I've heard there are some unsavoury ones out there. AD-blocking software had had the concept of a blocklist of many years, but I haven't really found a good central repository of blocked servers, and most importantly, the *reasons* they are blocked.

This set of PowerShell functions will let you pull the server block list from another Mastodon instance. If we ever decide to put together a master block list, these can easily be adapted to use that list as a source.

Get-MastodonServerBlock.ps1

```
# You need to run the .ps file in PowerShell so that the functions get
# loaded but after that you can just use them like regular commands.
#
# e.g.
# Get-MastodonServerBlock `
#     -domain source.social `
#     -accessToken "source_domain_token" |
# Foreach-Object {
#     Add-MastodonServerBlock `
#         -domain target.social `
#         -accessToken "target_domain_token" `
#         -block $_
# }
#
# If you don't have admin access to another server, and that server is
# publicly allowing their blocks to bee seen, you can leave off the
```

```
# -accessToken parameter and get that version of the list. Some of the
# domains may be obfuscated in this case, and the obfuscated domains
# will be ignored by Add-MastodonServerBlock.

#############################################################################

# Creates an object used by Add-MastodonServerBlock from its constituent
# parameters. Really just a helper method.
function New-MastodonServerBlock {
    param (
        [Parameter(Mandatory = $true)][string] $domain,
        [Parameter(Mandatory = $true)][string] $severity,
        [Parameter(Mandatory = $false)][string] $justification = "",
        [Parameter(Mandatory = $false)][string] $comment = "",
        [Parameter(Mandatory = $false)][switch] $rejectMedia = $false,
        [Parameter(Mandatory = $false)][switch] $rejectReports = $false,
        [Parameter(Mandatory = $false)][switch] $obfuscate = $false
    )

    $block = @{
        domain          = $domain
        severity        = $severity
        public_comment  = $justification
        private_comment = $comment
        reject_media    = $rejectMedia
        reject_reports  = $rejectReports
        obfuscate       = $obfuscate
    }

    New-Object -TypeName PSObject -Property $block
}

# Retrieves the list of server blocks from a server.
# If the accessToken is specified, it will attempt to use the admin
# API otherwise if will use the instance API where domains names may
# be obfuscated.
function Get-MastodonServerBlock {
```

```
    param (
        [Parameter(Mandatory = $true)][string] $domain,
        [Parameter(Mandatory = $false)][string] $accessToken = $null
    )
    if ($accesstoken -ne $null) {
        (Invoke-WebRequest `
            -Uri "https://$domain/api/v1/admin/domain_blocks" `
            -Method GET `
            -Headers @{ Authorization = "Bearer $accesstoken" }).Content | ConvertFrom-Json
    }
    else {
        (Invoke-WebRequest `
            -Uri "https://$domain/api/v1/instance/domain_blocks" `
            -Method GET).Content | ConvertFrom-Json
    }
}

# Adds a server block to a server.
# Requires an access token with appropriate access.
function Add-MastodonServerBlock {
    param (
        [Parameter(Mandatory = $true)][string] $domain,
        [Parameter(Mandatory = $true)][string] $accesstoken,
        [Parameter(Mandatory = $true, ValueFromPipeline = $true)] $block
    )

    if ($block.domain.IndexOf("*") -ge 0) {
        Write-Warning "Ignoring obfuscated domain $($_.domain)"
        return
    }

    Invoke-WebRequest `
        -Uri "https://$domain/api/v1/admin/domain_blocks" `
        -Method POST `
        -Headers @{ Authorization = "Bearer $accesstoken" } `
        -Body $($_ | ConvertTo-Json) `
        -ContentType "application/json"
```

```
}
```

How many Mastodon instances are out there?

Ever since I spun up my own Mastodon server and I've been watching my web logs just to see all the other unique instances that connect to mine. My personal server with only one user currently knows about ~1800 other servers. But there's got to be more than that out there, right? I thought so too. So I wrote a fairly simple bit of PowerShell that lets me crawl the reference tree from one server to the next and try and count all the instances on the internet.

Get-MastodonInstances.ps1

```
$instances = New-Object System.Collections.Generic.HashSet[string] -ArgumentList
([StringComparer]::OrdinalIgnoreCase)
$erroredInstances = New-Object System.Collections.Generic.HashSet[string] -ArgumentList
([StringComparer]::OrdinalIgnoreCase)
$newInstances = New-Object System.Collections.Generic.List[string]

$instances.Add("mastodon.social")
$newInstances.Add("mastodon.social")

for ($i = 0; $i -lt $newInstances.Count; $i++) {
    $instance = $newInstances[$i]
    Write-Host "$($i + 1) / $($newInstances.Count) ($instance)"

    try {
        $response = Invoke-WebRequest `
            -Uri "https://$instance/api/v1/instance/peers" `
            -Method GET `
            -ErrorAction SilentlyContinue
    }
    catch {
        $erroredInstances.add($instance) | Out-Null
        continue
```

```
    }

    if ([string]::IsNullOrWhiteSpace($response.Content)) {
        continue
    }

    $response.Content |
    ConvertFrom-Json |
    Where-Object { ($_.IndexOf("*") -lt 0) -and $instances.Add($_) } |
    ForEach-Object { $newInstances.Add($_.ToString()) }
}

$instances | Out-File -FilePath "MastodonInstances.txt"
$erroredInstances | Out-File -FilePath "MastodonErroredInstances.txt"

$instances.Count
```

There are some caveats to this approach, of course. Some of the instances the script encountered were offline, some were not actually Mastodon, but other implementors of ActivityPub (Friendica, Pleroma, etc.), and some just didn't like me querying them anonymously. There is also the case where certain "social circles" have completely cut themselves off from the rest of the Fediverse. I'll never know about those servers.

My script has been running for about an hour now, and it's only now querying the ~2700th host, and has added **~44,000** to the backlog. It'll be a while before it finishes. Offline domains slow it down a lot while waiting for the timeout. I suppose I could have worked in multi-threading, but this was just a quick-and-dirty experiment. I'll update this post if and when it ever comes up with a final number.

Update

I had to restart the script because when I came back in the morning, it had maxed out my laptop's memory. I've slightly optimized the script by adding to a .NET `System.Collections.Generic.List<string>` rather than re-creating a PowerShell array each iteration through the loop. Also added a try/catch block to filter out the errors I was seeing in the console and just to keep track of them.

Luckily, it's not very taxing to run this script because most of the run time is waiting for the other servers to respond.

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

Kitchen Chairs

Part 1. Kitchen renovation and some new chairs

In the early months of 2022, we started a kitchen renovation. Right down to the studs. Long story short, We ended up putting in an island at which we can sit to eat. In order to do so, we needed to have some counter-height chairs or stools. Looked some up online, found some I liked, and pulled the trigger on 4 chairs from Home Depot.

As we waited for the Home Depot chairs to be delivered, I anxiously tracked the UPS delivery. The delivery date came and went and there were no chairs. I called UPS to see what was up, and despite having tracked the delivery with a UPS tracking number, and that tracking information having details about the package, UPS was now telling me that the chairs were never picked up from the shipper (Home Depot) and that I should take it up with them. Ok, your tracking system says differently, but whatever, I'll call Home Depot.

Home depot is adamant that the chairs were indeed picked up by UPS and that UPS had them. **What. The. Fuck.** Home Depot is telling me that UPS has the chairs and UPS is telling me that Home Depot never gave them the chairs. I'm starting to question whether or not the chairs even exist. I called both companies back several times over the next two weeks, and eventually had to settle for a refund because they were the last of their kind in stock and nobody had no fucking clue what had happened to them.

Part 2. The second (and third) order of chairs

After giving up on the first set of chairs, the search begain anew for some other nice, affordable chairs. I fould a similar set at Wayfair and ordered them. They showed up a few days late, near the beginning of July, but no big deal; at least they're here. As a result of placing the order, I started getting Wayfair emails with sales, and such. After two weeks of sitting on these chairs, I saw that the chairs were reduced in price by nearly %50. That's pretty significant, especially when you buy 4 of them, so I called Wayfair's support and asked if there was any kind of price guarantee because I was still well within the return window of 30 days.

Wayfair told me that they did not do that sort of thing. I talked them through the fact that I was still in the return window and that I could just re-order the chairs, and return the first set and end up with the chairs for the lower price, and that we could both save ourselvers some trouble by avoiding a round-trip shipment of 4 chairs, but they would not go for it. I realized that arguing was pointless, so I did exactly what I said I could do: I ordered a new set of 4 chairs, got a return label for my existing chairs, and waited for the second set to arrive.

When they finally did arrive, I didn't even take them out of the boxes. I just slapped the return labels on the boxes on the second set of chairs and brought them right back to FedEx. I got my refund and continued using the original set of Wayfair chairs the entire time.

Part 3. The unexpected chairs

Fast forward to October 2022. I get an unprompted email from UPS saying that my Home Depot shipment will be delivered in about a week. **What?** I haven't ordered anything from Home Depot since the original chair order. I thought to myself, "Maybe they actually found those chairs and are finally shipping them to me." Turns out that is exactly what happened. Mid-October, I took delivery of the original set of 4 chairs, ordered back in June 2022 that were lost somewhere between Home Depot and UPS. The same chairs for which I've already had my purchase price refunded, and for which I received a $50 Home Depot gift certificate for my troubles.

Anyway, they're much nicer than the Wayfair chairs -- they were my first choice, after all. And it upon these chairs that we now sit at our renovated island counter. I guess I'm looking to sell my Wayfair chairs. So...win?