VaultWarden

- Backing up and Restoring the Vaultwarden SQL DB
- Backing up your vault
- Getting back to the vaultwarden organizaition admin page
- Self host password manager with Vaultwarden and Traefik
- Vaultwarden
- VaultWarden BW CLI

Backing up and Restoring the Vaultwarden SQL DB

Postgresql databases

Configure database connections in pgAdmin

run the tcdbinfo.sh script to see the connection details for both the old and the new database, and set them up in pgAdmin.

tcdbinfo PG Admin Connect
Image not found or type unknown Image not found or type unknown

Create database Backup

In `pgAdmin`, right click `vaultwarden->Databases->vaultwarden` and click `Backup...`. Give the file a name (e.g. `vaultwarden.sql`) and click `Backup`.

PG Admin Select Backup PG Admin Backup
Image not found or type unknown Image not found or type unknown

Restore database backup

In `pgAdmin`, right click `testwarden->Databases->vaultwarden` and click `Restore...`. Select the sql file (`vaultwarden.sql`).

PG Admin Restore PG Admin Restore
Image not found or type unknown Image not found or type unknown

On the 2nd tab page, select the first 3 options (`Pre-data`, `Data` and `Post-data`). On the last tab, select `Clean before restore`. Now click `Restore`.

PG Admin Restore PG Admin Restore
Image not found or type unknown Image not found or type unknown

Backing up your vault

https://github.com/dani-garcia/vaultwarden/wiki/Backing-up-your-vault

Overview

vaultwarden data should be backed up regularly, preferably via an automated process (e.g., cron job). Ideally, at least one copy should be stored remotely (e.g., cloud storage or a different computer). Avoid relying on filesystem or VM snapshots as a backup method, as these are more complex operations where more things can go wrong, and recovery in such cases can be difficult or impossible for the typical user. Adding an extra layer of encryption on your backups would generally be a good idea (especially if your backup also includes config data like your admin token), but you might choose to skip this step if you're confident that your master password (and those of your other users, if any) is strong.

Backing up data

By default, vaultwarden stores all of its data under a directory called `data` (in the same directory as the `vaultwarden` executable). This location can be changed by setting the DATA_FOLDER environment variable. If you run vaultwarden with SQLite (this is the most common setup), then the SQL database is just a file in the data folder. If you run with MySQL or PostgreSQL, you will have to dump that data separately -- this is beyond the scope of this article, but a web search will turn up many other tutorials that cover this topic.

When running with the default SQLite backend, the vaultwarden `data` directory has this structure:

```
data
├── attachments          # Each attachment is stored as a separate file under this dir.
│   └── <uuid>           # (The attachments dir won't be present if no attachments have been created.)
│       └── <random_id>
├── config.json          # Stores admin page config; only exists if the admin page has been enabled before.
├── db.sqlite3           # Main SQLite database file.
├── db.sqlite3-shm       # SQLite shared memory file (not always present).
```

```
├── db.sqlite3-wal       # SQLite write-ahead log file (not always present).
├── icon_cache           # Site icons (favicons) are cached under this dir.
│   ├── <domain>.png
│   ├── example.com.png
│   ├── example.net.png
│   └── example.org.png
├── rsa_key.der          # `rsa_key.*` files are used to sign authentication tokens.
├── rsa_key.pem
├── rsa_key.pub.der
└── sends                # Each Send attachment is stored as a separate file under this dir.
    └── <uuid>           # (The sends dir won't be present if no Send attachments have been created.)
        └── <random_id>
```

When running with MySQL or PostgreSQL backends, the directory structure is the same, except there are no SQLite files. You'll still want to back up files in the `data` directory, as well as a dump of your MySQL or PostgreSQL tables.

Each set of files is discussed in more detail next.

SQLite database files

Backup required.

The SQLite database file (`db.sqlite3`) stores almost all important vaultwarden data/state (database entries, user/org/device metadata, etc.), with the main exception being attachments, which are stored as separate files on the filesystem.

You should generally use the `.backup` command in the SQLite CLI (`sqlite3`) to back up the database file. This command uses the Online Backup API, which SQLite documents as the best way to back up a database file that may be in active use. If you can ensure the database will not be in use when a backup runs, you can also use other methods such as the `.dump` command, or simply copying all the SQLite database files (including the `-wal` file, if present).

A basic backup command looks like this, assuming your data folder is `data` (the default):

```
sqlite3 data/db.sqlite3 ".backup '/path/to/backups/db-$(date '+%Y%m%d-%H%M').sqlite3'"
```

You can also use `VACUUM INTO`, which will compact empty space, but takes somewhat more processing time:

```
sqlite3 data/db.sqlite3 "VACUUM INTO '/path/to/backups/db-$(date '+%Y%m%d-%H%M').sqlite3'"
```

Assuming this command is run on January 1, 2021 at 12:34pm (local time), this backs up your SQLite database file to `/path/to/backups/db-20210101-1234.sqlite3`.

You can run this command via a cron job periodically (preferably at least once a day). If you are running via Docker, note that the Docker images do not include an `sqlite3` binary or `cron` daemon, so you would generally install these on the Docker host itself and run the cron job outside of the container. If you really want to run backups from within the container for some reason, you can install any necessary packages during container startup, or create your own custom Docker image with your preferred `vaultwarden/server:<tag>` image as the parent.

If you want to copy your backup data to cloud storage, rclone is a useful tool for interfacing with various cloud storage systems. restic is another good option, especially if you have larger attachments and want to avoid recopying them as part of each backup.

The `attachments` dir

Backup required.

File attachments are the only important class of data not stored in database tables, mainly because they can be arbitrarily large, and SQL databases generally aren't designed to handle large blobs efficiently. This directory won't be present if no file attachments have ever been created.

The `sends` dir

Backup optional.

Like regular file attachments, Send file attachments are not stored in database tables. (Send text notes are stored in the database, however.)

Unlike regular attachments, Send attachments are intended to be ephemeral. Therefore, you might choose not to back up this directory if you want to minimize the size of your backups. On the other hand, if it's more important to maintain proper functionality of existing Sends across a restore, then you should back up this directory.

This directory won't be present if no Send attachments have ever been created.

The `config.json` file

Backup recommended.

If you use the admin page to configure your vaultwarden instance and don't have your configuration backed up some other way, then you probably want to back up this file so you don't have to figure out your preferred configuration all over again.

Keep in mind that this file does contain some data in plaintext that could be considered sensitive (admin token, SMTP credentials, etc.), so make sure to encrypt this data if you're concerned that someone else might be able to access it (e.g., when uploaded to cloud storage).

The `rsa_key*` files

Backup recommended.

These files are used to sign the JWTs (authentication tokens) of users currently logged in. Deleting them would simply log out each user, forcing them to log in again and it would also invalidate any open invitation tokens that have been sent via mail.

The `rsa_key.pem` (private key) file could be considered somewhat sensitive. In principle, it could be used to forge vault login sessions to your server, though in practice, doing so would require additional knowledge of various UUIDs (e.g., taken from a copy of your database). Also, any data obtained with a forged session would still be encrypted with personal and/or organization keys, so brute-forcing the relevant master password in order to obtain those keys would still be required. Admin panel login sessions, however, could be forged easily (this would only work if the admin panel is enabled). This wouldn't provide access to vault data, but it would allow some administrative actions like deleting users or removing 2FA.

Overall, encrypting the private key is recommended if you're concerned that someone else might be able to access it (e.g., when uploaded to cloud storage).

The `icon_cache` dir

Backup optional.

The icon cache stores website icons so that they don't need to be fetched from the login site repeatedly. It's probably not worth backing up unless you really want to avoid refetching a large cache of icons.

Restoring backup data

Make sure vaultwarden is stopped, and then simply replace each file or directory in the `data` dir with its backed up version.

When restoring a backup created using `.backup` or `VACUUM INTO`, make sure to first delete any existing `db.sqlite3-wal` file, as this could potentially result in database corruption when SQLite tries to recover `db.sqlite3` using a stale/mismatched WAL file. However, if you backed up the database using a straight copy of `db.sqlite3` and its matching `db.sqlite3-wal` file, then you must restore both files as a pair. You don't need to back up or restore the `db.sqlite3-shm` file.

It's a good idea to run through the process of restoring from backup periodically, just to verify that your backups are working properly. When doing this, make sure to move or keep a copy of your original data in case your backups do not in fact work properly.

Examples

This section contains an index of third-party backup examples. You should review an example thoroughly and understand what it's doing before using it.

- https://github.com/ttionya/vaultwarden-backup
- https://github.com/shivpatel/bitwarden_rs-local-backup
- https://github.com/shivpatel/bitwarden_rs_dropbox_backup
- https://gitlab.com/1O/vaultwarden-backup
- https://github.com/jjlin/vaultwarden-backup
- https://github.com/jmqm/vaultwarden_backup

Getting back to the vaultwarden organizaition admin page

https://vaultvixen.coltscomputer.services/#/organizations/0cef578d-41a6-49fd-9270-778f66bfc028/vault

Self host password manager with Vaultwarden and Traefik

https://blog.puvvadi.me/posts/selfhost-paswword-manager-vaultwarden-traefik/

https://tech.aufomm.com/deploy-bitwarden-with-docker-and-traefik/

Vaultwarden is light weight feature rich drop in replacement for Bitwarden server. It’s essentially debloated version of the Bitwarden.

To use Vaultwarden, SSL is required. Otherwise, singing in to the server is impossible. That’s where traefik comes in. Here I’m assuming you have a domain, cloudflare account & domain added to your account, and docker is already set up and ready to go.

Cloudflare

Please keep cloudflare’s `Email`, `API KEY` or `API TOKEN` ready.

Traefik setup

Create required files

directory structure

```
touch docker-compose.yml
touch config.yml
mkdir data && cd data
touch acme.json
touch traefik.yml
```

Directory structure should be like this

```
```

```
├── docker-compose.yml
├── config.yml
└── data
    ├── acme.json
    └── traefik.yml
```

Docker network

Create a network with following

```
```

```
docker network create -d bridge proxy
```

Docker compose

First open `docker-compose.yml` and add following

```
```

```
version: '3'

services:
  traefik:
    image: traefik:latest
    container_name: traefik
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    networks:
      - proxy
    ports:
      - 80:80
      - 443:443
    dns:
      - 1.1.1.1
      - 8.8.8.8
    environment:
      - CF_API_EMAIL=email@example.com
    # - CF_API_KEY= # use either api key or api token based on you usecase
    #   - CF_API_TOKEN=
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /home/user/traefik/data/traefik.yml:/traefik.yml:ro
      - /home/user/traefik/data/acme.json:/acme.json
      - /home/user/traefik/config.yml:/config.yml:ro
    labels:
      - "traefik.enable=true"
      # http entrypoint
      - "traefik.http.routers.traefik.entrypoints=http"
      # Dashboard
      - "traefik.http.routers.traefik.rule=Host(`traefik.internal.example.net`)"
      # To create a user:password pair, the following command can be used:
      # echo $(htpasswd -nb user password) | sed -e s/\\$/\\$\\$/g
      - "traefik.http.middlewares.traefik-auth.basicauth.users=<user & password>"
      # redirect middleware
      - "traefik.http.middlewares.traefik-https-redirect.redirectscheme.scheme=https"
      - "traefik.http.middlewares.sslheader.headers.customrequestheaders.X-Forwarded-Proto=https"
      - "traefik.http.routers.traefik.middlewares=traefik-https-redirect"
      # https entrypoint
      - "traefik.http.routers.traefik-secure.entrypoints=https"
      - "traefik.http.routers.traefik-secure.rule=Host(`traefik.internal.example.net`)"
      - "traefik.http.routers.traefik-secure.middlewares=traefik-auth"
      - "traefik.http.routers.traefik-secure.tls=true"
      - "traefik.http.routers.traefik-secure.tls.certresolver=cloudflare"
      - "traefik.http.routers.traefik-secure.tls.domains[0].main=internal.example.net"
      - "traefik.http.routers.traefik-secure.tls.domains[0].sans=*.internal.example.net"
      - "traefik.http.routers.traefik-secure.service=api@internal"

networks:
  proxy:
    external: true
```

> dns records should already pointed to you docker host. e.g. if docker host ip is `10.20.20.5` A record for `traeif.internal` should point to `10.20.20.5`.

traefik config

```
api:
  dashboard: true
  debug: true
entryPoints:
  http:
    address: ":80"
    http:
      redirections:
        entryPoint:
          to: https
          scheme: https
  https:
    address: ":443"
serversTransport:
  insecureSkipVerify: true
providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false
  file:
    filename: /config.yml
certificatesResolvers:
  cloudflare:
    acme:
      email: email@example.net
      storage: acme.json
      dnsChallenge:
        provider: cloudflare
        disablePropagationCheck: true
        resolvers:
          - "1.1.1.1:53"
          - "1.0.0.1:53"
```

To spin up the traefik docker container, run

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

Once docker container created, traefik will generate ssl certs for `internel.example.net` & wildcard cert for `*.internel.example.net`. traefik dashboard will be available at `traefik.internal.example.net`.

Vaultwarden

Volume

I’m using named volumes here for the sake. You can use any directory on the host and bind that.

```
```

```
docker volume create vaultwarden
```

Reason for creating volume outside the compose file, in case, container destroyed with `rm`, data would be still available from the volume.

Docker-compose

Create a new directory in your home `vaultwarden` and add new file `docker-compose.yml`.

```
```

```
version: '3'

services:
  vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    networks:
      - proxy
    ports:
      - 8100:80
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - vaultwarden:/data
    environment:
      - DOMAIN=https://vaultwarden.internal.example.net
      - SMTP_HOST=smtp.example.com
      - SMTP_FROM=email@example.com
      - SMTP_FROM_NAME=Vaultwarden
      - SMTP_SECURITY=SECURITYMETHOD
      - SMTP_PORT=XXXX
      - SMTP_USERNAME=email@example.com
      - SMTP_PASSWORD=YourReallyStrongPasswordHere
      - SMTP_AUTH_MECHANISM="Mechanism"
    labels:
      - traefik.enable=true
      - traefik.docker.network=proxy
      - traefik.http.middlewares.redirect-https.redirectScheme.scheme=https
      - traefik.http.middlewares.redirect-https.redirectScheme.permanent=true
      - traefik.http.routers.vaultwarden-https.rule=Host(`vaultwarden.internal.example.net`)
      - traefik.http.routers.vaultwarden-https.entrypoints=https
      - traefik.http.routers.vaultwarden-https.tls=true
      - traefik.http.routers.vaultwarden-https.service=vaultwarden
      - traefik.http.routers.vaultwarden-http.rule=Host(`vaultwarden.internal.example.net`)
      - traefik.http.routers.vaultwarden-http.entrypoints=http
      - traefik.http.routers.vaultwarden-http.middlewares=redirect-https
      - traefik.http.routers.vaultwarden-http.service=vaultwarden
      - traefik.http.services.vaultwarden.loadbalancer.server.port=80
      - traefik.http.routers.vaultwarden-websocket-https.rule=Host(`vaultwarden.internal.example.net`) && Path(`
      - traefik.http.routers.vaultwarden-websocket-https.entrypoints=https
      - traefik.http.routers.vaultwarden-websocket-https.tls=true
      - traefik.http.routers.vaultwarden-websocket-https.service=vaultwarden-websocket
      - traefik.http.routers.vaultwarden-websocket-http.rule=Host(`vaultwarden.internal.example.net`) && Path(`/
      - traefik.http.routers.vaultwarden-websocket-http.entrypoints=http
      - traefik.http.routers.vaultwarden-websocket-http.middlewares=redirect-https
      - traefik.http.routers.vaultwarden-websocket-http.service=vaultwarden-websocket
      - traefik.http.services.vaultwarden-websocket.loadbalancer.server.port=3012

networks:
  proxy:
    external: true

volumes:
  vaultwarden:
    external: true
```

> dns records should already pointed to you docker host. e.g. if docker host ip is `10.20.20.5` A record for `vaultwarden.internal` should point to `10.20.20.5`.

To spin up the vaultwarden docker container, run

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

Conclusion

For more details and documentation, visit Official github repo. Any queries, feel free to drop a comment. `Au Revoir`.

Vaultwarden

Vaultwarden is hosted at https://vaultvixen.coltscomputer.services

VaultWarden BW CLI

Export using bw cli

bw export --organizationid c5eab4a2-c584-4e89-9e7d-c73b9d9f9c7c --format json --output "Z:\Bitlocker, Keys, Oh My!\Configuration files\VaultWarden\"

current 2025 `bw export --output "Z:\Bitlocker, Keys, Oh My!\Configuration files\VaultWarden\$date" --format json`