Deploy Vaultwarden in multi-site environment in HA (Docker, OPNSense, Galera cluster, Nginx)

Written By	Jan Bachelor
Published URL:	https://bachelor-tech.com/detailed-guides/deploy-vaultwarden-in-multi-site-environment-in-ha-docker-opnsense-galera-cluster-nginx/
Last Updated	@January 16, 2026 9:27 AM

✨ **Table of Content**

Introduction

Running a mission-critical website on a single server creates a significant risk, as routine maintenance or unexpected hardware failures can take your service offline instantly. This tutorial guides you through transforming a standalone web server into a resilient High Availability (HA) cluster using accessible open-source tools.

We will utilize **Proxmox** to clone your existing environment and deploy **OPNsense** with **HAProxy** to intelligently distribute traffic between multiple nodes. To handle dynamic content, we solve the challenge of file synchronization by implementing **Syncthing**, a decentralized peer-to-peer tool that keeps folders like WordPress uploads identical across servers in real-time.

We will configure **Syncthing** on headless Linux instances, tunnel securely to its GUI for management, and set up automated health monitoring using **UptimeKuma**. By the end, you will have a robust architecture that can survive individual server reboots without downtime, offering a simpler alternative to complex enterprise-grade distributed file systems.

1. Vaultwarden or Bitwarden?

- While **Bitwarden** offers a robust, enterprise-ready self-hosted option, it is resource-intensive. The official Bitwarden server relies on a complex stack of MSSQL (Microsoft SQL Server) and multiple .NET containers, often requiring 2GB+ of RAM just to idle.
- **Vaultwarden** is a lightweight rewrite of the Bitwarden API in Rust. It is fully compatible with official Bitwarden apps (browser extensions, mobile, desktop) but runs on a fraction of the resources (often <100MB RAM).
- Bitwarden offers a free option for individuals. If you have a family or an organization and want to share a collection of passwords, you would need to go on the paid tier. The advantages include convenience (no maintenance on your part) and no missing features.
- The obvious pitfall with Vaultwarden is that if you deploy it yourself, you will also need to handle the maintenance. We counter that by using docker containers that can be updated very easily. What takes more effort is to ensure your web and database clusters are up to date and secure as well, which is beyond the scope of this tutorial. In addition, if Bitwarden introduces some flashy new features, you may need to wait till the Rust devs update them for Vaultwarden.

Pre-requisites

- A web server (or two) on site 1 + 2 running nginx on Debian 12 or newer. Their set up is beyond the scope of this article - you can refer to my previous article on **How to configure High Availability for a Web Server using Syncthing and HAProxy (on OPNSense)**.
- A Galera cluster (MySQL) deployed in HA on your two sites, ideally with a Galera witness on site 3 (all connected via WireGuard). Again, the set up is beyond the scope of this article - you can complete them using previous tutorials titled **Deploy MariaDB Galera Cluster on Proxmox** and **Set up a Galera Witness on Hetzner VPS using Terraform + Ansible (AWX)**
- OPNSense (or pfsense or similar) with an HAProxy module to act as a reverse proxy + virtual IP interface for our DB cluster on site 1 and 2. Previously, this was completed in a tutorial called **OPNSense in HA with CARP with dual WANs**.

Topology

- **Site 1 (Main)** - all connected to a UPS managed from an RPI
  - OPNSense in HA (192.168.8.254) - CARP 0 - see **this guide to set it up**.
    - `OPNSense1` (192.168.8.1) - Proxmox host 1
    - `OPNSense2` (192.168.8.2) - Proxmox host 2
    - Site-to-site VPN WireGuard) with the interface of 10.10.10.1/24
  - Web servers in HA - reachable via a shared back-end pool in HAProxy within OPNSense
    - `web1` (192.168.8.9) - Proxmox host 1
      - Syncthing from web1 to web2 & web1 to web3 - to sync user data in near real-time
      - Connected to a local Gitea server - to update app data on demand
    - `web2` (192.168.8.10) - Proxmox host 2
      - Syncthing from web2 to web1 & web2 to web3 - to sync user data in near real-time

        - Connected to a local Gitea server - to update app data on demand
    - Database Cluster - MariaDB Galera cluster - reachable on virtual IP 192.168.8.70 configured in OPNSense
        - `galera1-2` (192.168.8.71 + 72) - Proxmox host 1
        - `galera3-4` (192.168.8.73 + 74) - Proxmox host 2
        - `galera-template` - a pre-prepared LXC template ready for easy deployment in case a replacement is needed (can be automated with AWX for deployment)
    - `Gitea LXC` (192.168.8.21) - to sync app data and deployment scripts + config.xml for each OPNSense host.
    - `Uptimekuma1` (192.168.8.60) - to monitor all Site 1 hosts' uptime inc. services like Syncthing
    - `RPI` (192.168.8.16)
        - Corosync for HA for the Proxmox cluster) - ensure a host is reachable if one is down
        - Proxmox Backup Server with an added drive - for Site 1 + 2.
        - `APCupsd` with scripts for smooth graceful shutdown of Proxmox hosts - see **this tutorial** for more information.
        - Can act as a local Galera witness before Site 3 is set up.

- **Site 2 (Online Backup)**
    - `OPNSense3` (192.168.6.1) - Proxmox host 3
        - Site-to-site VPN (WireGuard) with the interface of 10.10.10.2/24
    - `Web3` (192.168.6.10) - Proxmox host 3
        - Syncthing from web3 to web1 & web3 to web2 - to sync user data in near real-time
        - Connected to Site 1's Gitea server - to update app data on demand
    - `Galera4+5` (192.168.6.75 + 76) - Proxmox host 3

    - `Uptimekuma2` (192.168.6.60) - monitors Site 2 services inc. web2's Syncthing jobs.

- **Site 3 (Witness)**
    - Galera witness VPS deployed automatically via AWX in Hetzner - see **this tutorial** on how to achieve that.
        - Site-to-site VPN (WireGuard) with the interface of 10.10.10.3/24
        - `Garb` - daemon that does not hold data and provides HA if one site is down (quorum voting as +1)
        - `Uptimekuma3` - monitors uptime for websites that are provided by either site (such as bachelor-tech.com)

Software Versions at the time of write-up

For the purpose of reproducibility, as a side note, you can find the versions that I worked with:

- OPNSense:
    - Core Version: 25.7.10 (commit: c2f076f30)
    - os-haproxy plugin: 4.6_1
    - os-acme plugin: 4.11
    - os-ddclient plugin: 1.28
    - os-git-backup: 1.1_1
- Vaultwarden Docker Image: 1.35.2
- Web servers
    - OS: Debian 12 (Bookworm, kernel 6.1.0-26-amd64)
    - nginx: 1.28.0
    - Docker Engine (client + server): 29.1.3
    - Syncthing: 2.0.12 (Hafnium Hornet)
    - Fail2ban (server + client): 1.0.2
- Galera cluster:

    - OS: Debian 13 (Trixie, kernel 6.17.2-2-pve)
    - Maria DB Server: 11.8.3
    - Maria DB Client: 15.2
- UptimeKuma: 2.0.0-beta.4

Gotchas with Vaultwarden & Syncthing

There are two specific limitations to be aware of during an Active-Active setup for Vaultwarden:

- **Token Signing Keys:** Users will get logged out if they hit `web1` but their token was signed by `web2` or `web3` using a different key. You **must** ensure the RSA key files in the data directory are identical across all nodes.
    - We will handle that in this guide by using Syncthing to distribute the RSA key between the web nodes.
- **WebSockets:** Vaultwarden does not currently have a "message bus" (like Redis) to broadcast WebSocket events between nodes. If a user updates a password on `web1`, a device connected to `web2` or `web3` won't get the live update immediately. It will sync the next time the user manually refreshes or performs an action. This is a minor inconvenience but acceptable for most.
- Vaultwarden tables utilize Primary Keys, which is good (Galera requires them). However, ensure your Galera nodes are not running in `ENFORCING` mode for `wsrep_drupal_282555_workaround` or strict checking that might block `INSERT` statements if they momentarily lack a PK. Most likely, this will not be an issue.

---

2. Create a Vaultwarden DB + Install Dependencies

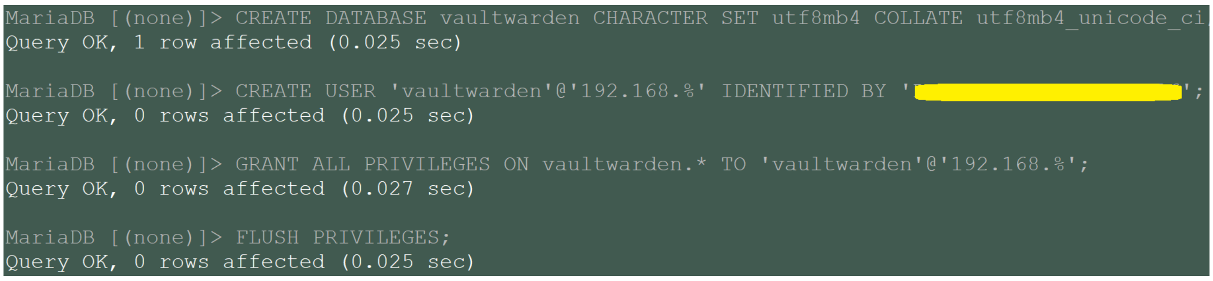
Now since the topology and the specifics of Vaultwarden have been covered, let us move forward with the installation and set up. We will be setting it up on Debian 13 (Trixie) - other versions will likely work in a very similar fashion.

Create a DB on your cluster

- SSH into any of the Galera cluster node (as they will sync) and run the following:

- Replace the ‘`your_secure_password`’ string with your own! It is recommended to avoid using special characters for compatibility - the length is what matters the most here! If you do use special chars, you will need to escape them properly later on in the `docker-compose.yml` file for Vaultwarden.
- If you are not sure about the subnet required for the user’s privileges, you can just use ‘`%`’ instead of specifying the IP range.

```
mysql -u root -p

CREATE DATABASE vaultwarden CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'vaultwarden'@'192.168.%' IDENTIFIED BY 'your_secure_password';
GRANT ALL PRIVILEGES ON vaultwarden.* TO 'vaultwarden'@'192.168.%';
FLUSH PRIVILEGES;
```



```
MariaDB [(none)]> CREATE DATABASE vaultwarden CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
Query OK, 1 row affected (0.025 sec)

MariaDB [(none)]> CREATE USER 'vaultwarden'@'192.168.%' IDENTIFIED BY '[redacted]';
Query OK, 0 rows affected (0.025 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON vaultwarden.* TO 'vaultwarden'@'192.168.%';
Query OK, 0 rows affected (0.027 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.025 sec)
```

Install dependencies + Vaultwarden itself

- Take a snapshot of each web server before you start the process!
- Install the following on EACH web node you have:

```
# Update and install basic tools
sudo apt update && sudo apt install -y ca-certificates curl

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
# Add the repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/doc
ker.asc] https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Refresh apt & install Docker
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plu
gin docker-compose-plugin

# Enable Docker Systemd service (so containers start on boot)
sudo systemctl enable --now docker
```

- Create a user, folder and a Docker Compose file:

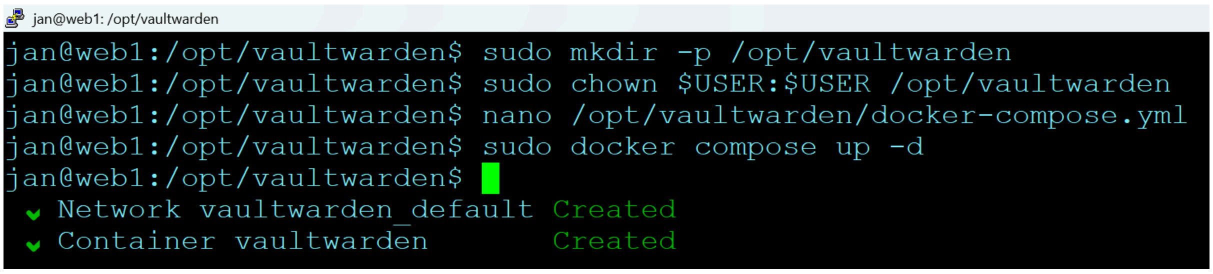
```
# Create the directory
sudo mkdir -p /opt/vaultwarden

# Set permissions so your current user can edit files there
sudo chown $USER:$USER /opt/vaultwarden
cd /opt/vaultwarden

# Create the Compose file
nano /opt/vaultwarden/docker-compose.yml
```

- The `compose.yml` file:
    - Note that in my set up, I have a virtual IP set up using OPNSense that I point the web application onto. If you do not have it set up yet, you can check out this part of the tutorial titled **Virtual IP Set Up on OPNSense** (you will then need a service such as HAProxy listening on that IP to receive and forward requests to your Galera cluster nodes).

```
services:
  vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: always
    ports:
      - "127.0.0.1:8000:80"
    environment:
      - DATABASE_URL=mysql://vaultwarden:yourpwd@1.2.3.4/vaultwarden
      - DOMAIN=https://vault.yourdomain.com
      - SIGNUPS_ALLOWED=false
      - INVITATIONS_ALLOWED=true
      - IP_HEADER=X-Real-IP
      - LOG_LEVEL=info
      - EXTENDED_LOGGING=true
      # --- SMTP Email Settings ---
      - SMTP_HOST=smtp.gmail.com  # Replace with your provider's host
      - SMTP_FROM=vaultwarden@your-domain.tld
      - SMTP_FROM_NAME=Vaultwarden
      - SMTP_SECURITY=starttls    # Options: starttls, force_tls, off
      - SMTP_PORT=587             # Usually 587 for starttls, 465 for force_tls
      - SMTP_USERNAME=your_email@gmail.com
      - SMTP_PASSWORD=your_app_password
      - SMTP_AUTH_MECHANISM="Plain" # This is safe if used with TLS
    logging:
      driver: "journald"
      options:
        tag: "{{.Name}}"
    volumes:
      # This creates a 'vw-data' sub-folder as a form of persistent storage
      - ./vw-data:/data
```

- Run it:

```
cd /opt/vaultwarden

# Sudo is needed to pull the image
sudo docker compose up -d
```



```
jan@web1:/opt/vaultwarden$ sudo mkdir -p /opt/vaultwarden
jan@web1:/opt/vaultwarden$ sudo chown $USER:$USER /opt/vaultwarden
jan@web1:/opt/vaultwarden$ nano /opt/vaultwarden/docker-compose.yml
jan@web1:/opt/vaultwarden$ sudo docker compose up -d
jan@web1:/opt/vaultwarden$
 ✔ Network vaultwarden_default Created
 ✔ Container vaultwarden       Created
```

- You can check its status after creation by running `sudo docker container ps -a` .

💡 Once your `docker-compose.yml` file is launched and you make changes to it, it is best to then re-create the container by running `sudo docker compose up -d --force-recreate`

Configure Nginx with Vaultwarden

- Depending on how your nginx instance is already configured, your set up may look like something like this:
  - Ensure the path + port number match!
  - SSL offloading is handled by HAProxy/OPNSense, so Nginx here listens on 8081. Ensure HAProxy passes the `X-Forwarded-Proto` header so Vaultwarden knows it's secure.

```
sudo nano /etc/nginx/conf.d/vaultwarden.conf

server {
    # We listen on 80 because HAProxy handles the SSL/HTTPS before it get
s here.
    listen 8081;

    # Replace with your actual FQDN
    server_name vault.yourdomain.com;

        # Replace with OPNSense's actual IP to ensure fail2ban blocks the cor
rect users:
    set_real_ip_from 192.168.0.0/16;
    real_ip_header X-Forwarded-For;
    real_ip_recursive on;
    # Allow large attachments (optional, but good for saving PDFs/Images in
vault)
    client_max_body_size 128M;

    location / {
        proxy_pass http://127.0.0.1:8000;

        # WebSocket support (used by /notifications/hub)
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Pass headers so Vaultwarden knows the real IP of the user, not just
"127.0.0.1"
        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;
    }
```

- To clarify the ports used (these remain internal and do not need firewall rules):
    - **Port 8081**: What nginx listens on to receive traffic from HAProxy
    - **Port 8000**: The Vaultwarden application (web interface, API, and WebSockets)

> 💡 Note: In Vaultwarden versions prior to 1.29.0, WebSockets required a separate
> port (3012). Modern versions serve WebSockets from the main application port,
> simplifying the configuration.

- Check nginx config and reload it:

```
# Confirm there is no typo in the syntax before reloading it
sudo nginx -t
```

```
# Reload nginx for the changes to take an effect
sudo systemctl reload nginx
```

- Remember to complete the above on all web server nodes, although you may wish to do the first one first and have HAProxy set up just with that one node until you got it working fully.

---

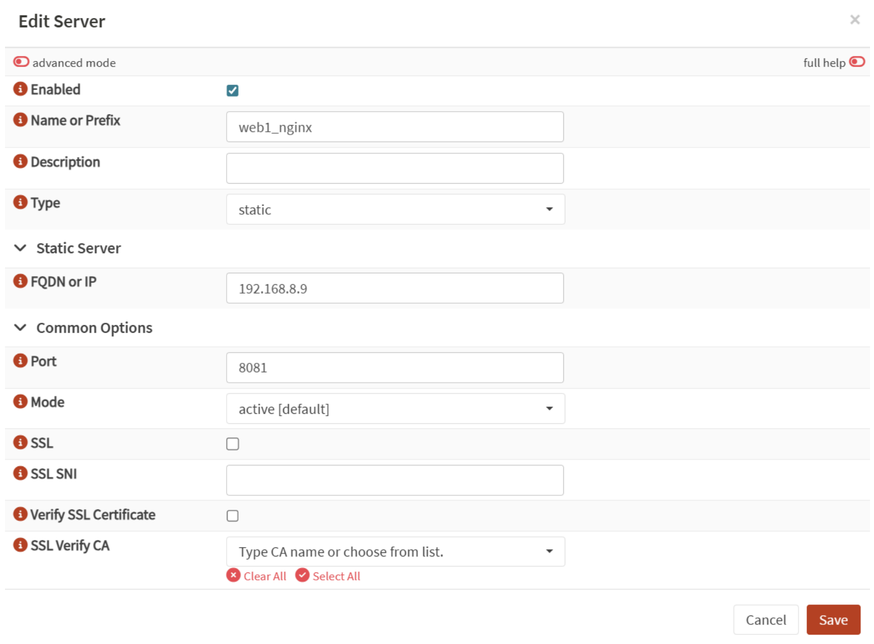
3. Configure OPNSense + HAProxy for Vaultwarden

- For true High Availability (ideally on each site or at least on the main site), you should **run your OPNSense in a master-backup configuration** running on separate hardware on the same site.
- You would need to open a port for HTTPS communication and since your OPNSense HA relies on having the web interface running on port 443, you will need to open another port on HAProxy to receive web traffic on (I use port 4443 that the ISP-provided router changes from 443 to 4443). See here for **more details on which ports to open for web traffic to work** on OPNSense, followed up by **NAT rules**.
- Create a DNS record for your `vault.yourdomain.tld` record pointing to the public IP of your OPNSense - you can **follow these steps to set up dynamic DNS with Cloudflare**.
    - If you prefer a more secure set up, you do not actually need to expose a public DNS record, you can simply set one up on your network using tools such as Unbound DNS.
- Before setting up HAProxy, you should also **set up an SSL certificate using the ACME plugin** in OPNSense.

Add your ‘real’ hosts

In case you do not have it set up yet, apart from defining your web hosts (under the ‘Real Servers’ tab) with the local port (I’m using 8081), we should check the health of each web host.

- In OPNSense, go to Services → HAPRoxy → Settings. Then move to the other menu and click on Real Servers → Real Servers. **Add a new one for each of your web servers**:
  - Name: `web1_nginx`
  - Type: `static`
  - IP: your actual IP
  - Port: your chosen port configured on nginx, e.g. `8081`
  - No SSL config here!



Edit Server

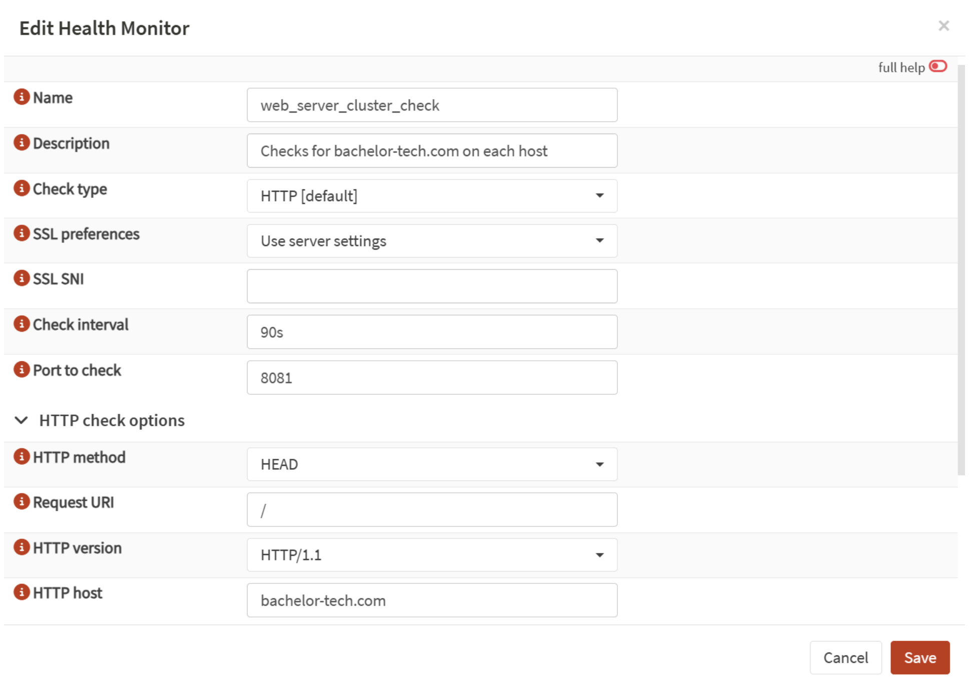
advanced mode	full help
Enabled	✓
Name or Prefix	web1_nginx
Description	
Type	static
Static Server	
FQDN or IP	192.168.8.9
Common Options	
Port	8081
Mode	active [default]
SSL	☐
SSL SNI	
Verify SSL Certificate	☐
SSL Verify CA	Type CA name or choose from list.

Set up a health monitor

We should also ensure that there is a health check in place for the web servers.

- Go to Rules & Checks → Health Monitors. Add a new monitor:
  - Name: `web_server_cluster_check` (or whatever you prefer)
  - Check type: `HTTP (default)`
  - SSL preferences: `use server settings` (default)
  - Check interval: `15-90s` (you may get some false positives if too aggressive)
  - Port to check: `8081`

- HTTP method: `HEAD`
- Request URI: `/`
- HTTP version: `HTTP/1.1`
- HTTP host: `yourdomain.tld`



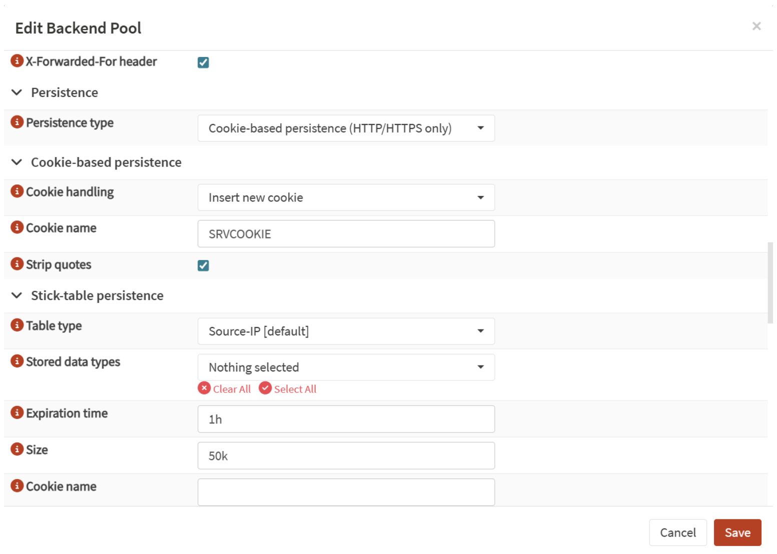
Name	web_server_cluster_check
Description	Checks for bachelor-tech.com on each host
Check type	HTTP [default]
SSL preferences	Use server settings
SSL SNI	
Check interval	90s
Port to check	8081
HTTP check options	
HTTP method	HEAD
Request URI	/
HTTP version	HTTP/1.1
HTTP host	bachelor-tech.com

Configure your back-end pool

You may already have a back-end pool configured for your existing web servers, yet we will need to create another pool with the same real hosts in it, since different parameters will need to be set up for Vaultwarden.

💡 The challenge with the default session rate period is that **it** is typically about 10 seconds long as well as no pass-through option is defined, which would result in the client having to re-establish connection with the server (users would see having a ‘Connecting…’ spinning wheel appear regularly). While for example PHP-based applications receive a payload and close the connection after responding, for web sockets, we want the session to remain active.

- Create a new back-end pool as follows (enable the advanced options):
    - Name: `backend_vaultwarden_nginx`
    - Mode: `HTTP (Layer 7)`
    - Balancing Algorithm: `Round Robin`
    - Random Draws: `2` (entirely up to you)
    - Proxy Protocol: none
    - Servers: add your 'real' servers here
    - ...
    - Enable health checking: `Ticked`
    - Health monitor: `web_server_cluster_check` (previously defined in the health check section)
    - Proxy Protocol: `none`
    - Servers: add your web servers
    - ...
    - X-Forwarded-For header: `Ticked`
    - Persistence Type: `Cookie-based persistence`
    - Cookie Handling: `Insert new cookie`
    - Cookie name: `SRVCOOKIE` (or another name as you desire)
    - Strip Quotes: `Ticked` (Default)
    - Expiration Time: `1h` (match your tunnel timeout)
    - ...
    - Connection Timeout: `5s` (fail-over fast if the server is not responding)
    - Check Timeout: `5s` (fail fast if health check fails)
    - Server Timeout: `3600s` (allows the server to keep the pipe open)
    - Retries: `3` (default)
    - Option pass-through: `timeout tunnel 3600s` (tells HAProxy that it is a tunnel)
    - You can leave the rest as default unless you have other specifics in your set up.

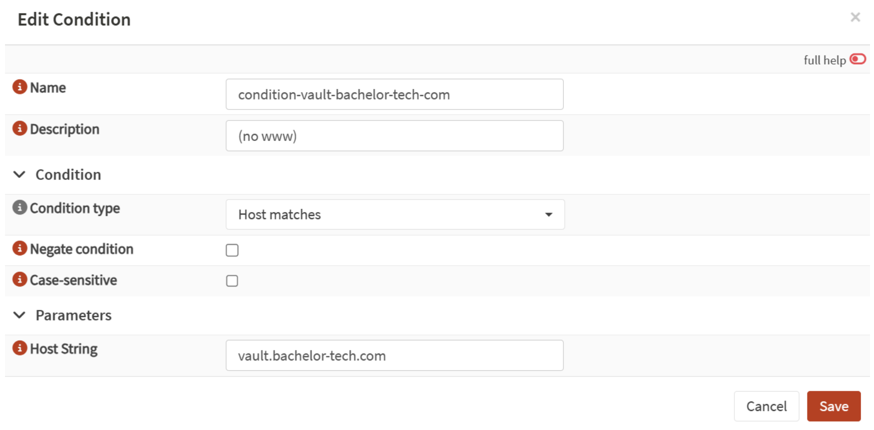


Edit Backend Pool	
X-Forwarded-For header	✓
Persistence	
Persistence type	Cookie-based persistence (HTTP/HTTPS only)
Cookie-based persistence	
Cookie handling	Insert new cookie
Cookie name	SRVCOOKIE
Strip quotes	✓
Stick-table persistence	
Table type	Source-IP [default]
Stored data types	Nothing selected
Expiration time	1h
Size	50k
Cookie name	

The most crucial part of the back-end pool set up - use a new pool with the existing web servers to configure cookie persistence + x-forwarded-for header + 1h expiration time (then scroll below for the other options mentioned above).

Configure a condition and a rule on HAProxy to listen to

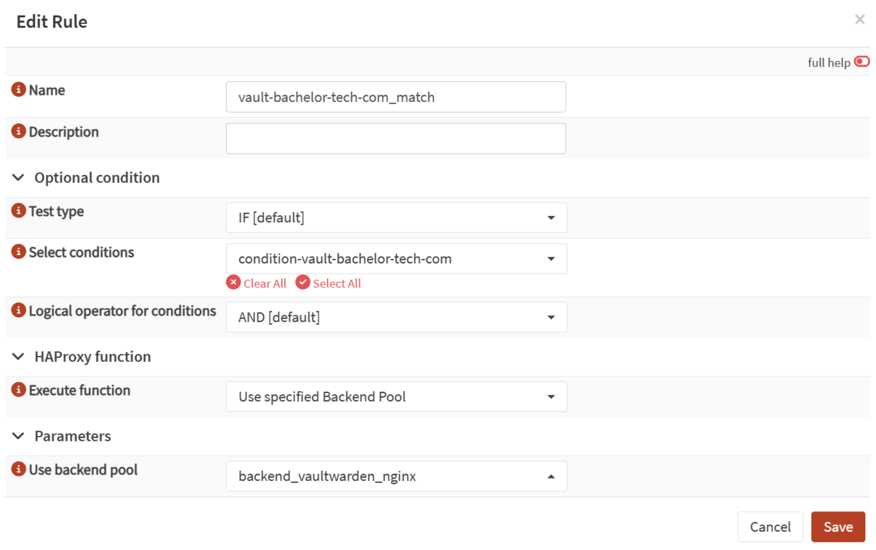
- While still under HAProxy's settings, go to Rules & Checks → Conditions and add a new condition:
    - Name: `condition-vault-your-domain-tld`
    - Condition type: `Host matches`
    - Host string: `vault.yourdomain.tld`



Edit Condition	
Name	condition-vault-bachelor-tech-com
Description	(no www)
Condition	
Condition type	Host matches
Negate condition	☐
Case-sensitive	☐
Parameters	
Host String	vault.bachelor-tech.com

- And then still under Rules & Checks, go to Rules and add a new rule:
    - Name: vault-your-domain-tld_match (do not use spaces)

- Test type: `IF`
- Select conditions: `condition-your-domain-tld` (find your own)
- Execute function: Use specified Backend Pool
- Use backend pool: `backend_vaultwarden_nginx` (as defined previously)



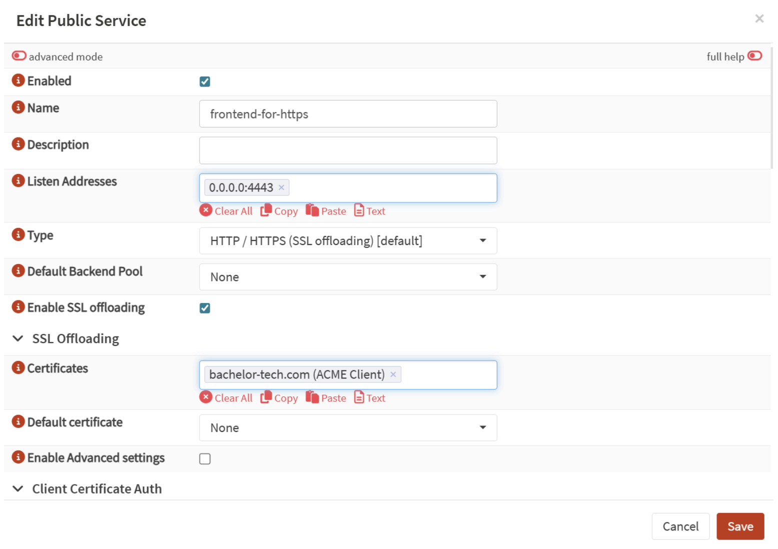
Edit Rule

Name	vault-bachelor-tech-com_match
Description	
Optional condition	
Test type	IF [default]
Select conditions	condition-vault-bachelor-tech-com
Logical operator for conditions	AND [default]
HAProxy function	
Execute function	Use specified Backend Pool
Parameters	
Use backend pool	backend_vaultwarden_nginx

Set up your public (front-end) service

Go to Virtual Services → Public Services. If you already have one defined for HTTPS, then simply add the rule at the end, test & apply the syntax and you are done.

- If you have not set up HTTPS, then create a new one. The key here is to firstly have firewall rules set up to forward traffic to 'This host' on a port that HAProxy is listening to (must be different from the web interface that you use to access OPNSense!) and then in the 'Public Service' window, tick the '`SSL offloading`' box to ensure that traffic is decrypted on your LAN. You would need to use the ACME certificate created earlier.


Edit Public Service

advanced mode | full help

Enabled	✓
Name	frontend-for-https
Description	
Listen Addresses	0.0.0.0:4443
Type	HTTP / HTTPS (SSL offloading) [default]
Default Backend Pool	None
Enable SSL offloading	✓
SSL Offloading	
Certificates	bachelor-tech.com (ACME Client)
Default certificate	None
Enable Advanced settings	☐
Client Certificate Auth	

💡 Ensure that you follow these steps on all of your sites to have the same config and can thus expect consistent behavior.

---

4. Troubleshoot Vaultwarden Docker/Web UI service

Let's try accessing the web UI. Try accessing the Vaultwarden interface on `https://vault.yourdomain.com` .

Troubleshooting web UI reachability

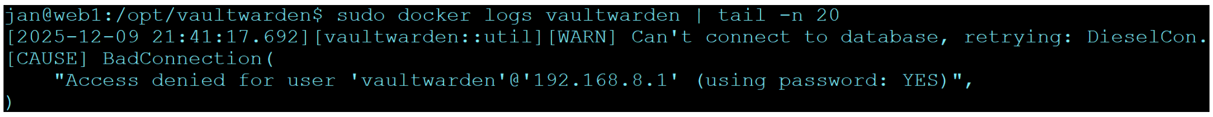
In case the web UI is not coming up, we need to determine where the traffic got stuck. Here are the layers:

- `DNS` - e.g. CloudFlare - is your DNS record set up? Does it point to the correct IP address?
- `OPNSense` - do you have a firewall + NAT rule set up for forwarding traffic to a port that HAPRoxy is listening on?
- `HAProxy` - is your public (front-end) service on and listening on the desired port? Have you attached your SSL certificate for decrypting traffic? Is your back-end pool forwarding traffic to the right host? Is your 'real' host configure on the right port?

- `Nginx` on the VM - is nginx on and listening on the right port? Is the syntax of your virtual host for forwarding ok?
- `Docker` - is the docker instance up? What do the logs say?
- `Galera DB Cluster` - is it reachable for the web servers? Is the virtual IP operational? Is the DB up, esp. if you are reaching it on a virtual IP via OPNSense / pfSense?

Example issue no.1 - Web server not accepted by the DB server

The example below shows that the Docker instance is running but spent the last two hours in a crash loop due to incorrect DB details:



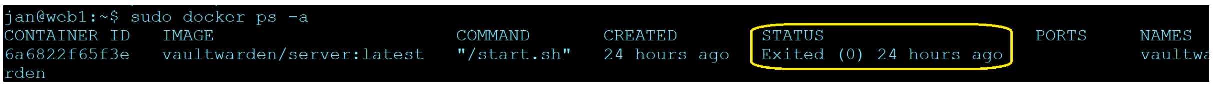
```
jan@web1:/opt/vaultwarden$ sudo docker logs vaultwarden | tail -n 20
[2025-12-09 21:41:17.692][vaultwarden::util][WARN] Can't connect to database, retrying: DieselCon.
[CAUSE] BadConnection(
    "Access denied for user 'vaultwarden'@'192.168.8.1' (using password: YES)",
)
```

- You may notice the IP 192.168.8.1 that points to the OPNSense host from which it is forwarded. This is because the Galera cluster is reachable on a virtual IP (192.168.8.70) provided by OPNSense. Such behavior is normal and expected.
- The main issue that I simulated here was that the user was defined for the wrong subnet - after dropping the user and re-creating it with the correct GRANT privileges fixed the issue.

Example issue no.2 - No Docker container running

Let's say you set up just one web server VM and were able to reach the web UI and then later, it stopped loading. You may wish you double check that the docker container is still up.

```
sudo docker container ps -a
```



```
jan@web1:~$ sudo docker ps -a
CONTAINER ID   IMAGE                          COMMAND       CREATED         STATUS                       PORTS     NAMES
6a6822f65f3e   vaultwarden/server:latest      "/start.sh"   24 hours ago    Exited (0) 24 hours ago                vaultwa
rden
```

- To fix it, go to the folder where your container is located and start it. Ensure that you set up Docker to start on boot.

```
cd /opt/vaultwarden
sudo docker compose up -d
sudo systemctl enable docker
```

Example issue no.3 - 401 Unauthorized" loops

If you log in successfully but are immediately logged out when refreshing or performing an action, your nodes likely have different signing keys.

- **The Cause:** Vaultwarden generates RSA key pair files in `/data` on first startup. If `web1` generated its own keys and `web2` generated different ones, a login token signed by `web1` will be rejected by `web2`.
- **The Fix:** Ensure Syncthing is successfully syncing the `rsa_key.pem`, `rsa_key.pub.pem`, and `rsa_key.der` files. You may need to manually copy the keys from the "primary" node to the others once to establish a baseline, then let Syncthing handle future updates.

First login

- Your first step in the Web UI would be to create your account.
- Since we set `SIGNUPS_ALLOWED=false` in the config, you will need to temporarily change that to `true` in `docker-compose.yml`, run `docker compose up -d` to apply, create your accounts, and then flip it back to `false` and restart again. This is intentional as a matter of exercising how you can easily tweak settings with Docker Compose, esp. when applying updates in the future.

---

5. Set up Syncthing for Vaultwarden data sync

To ensure that the uploaded data stay consistent across multiple web servers, we will employ Syncthing, which I already use on my web servers. If you would like some help with deploying it, check out my **previous tutorial on How to Configure HA for Web Servers**.

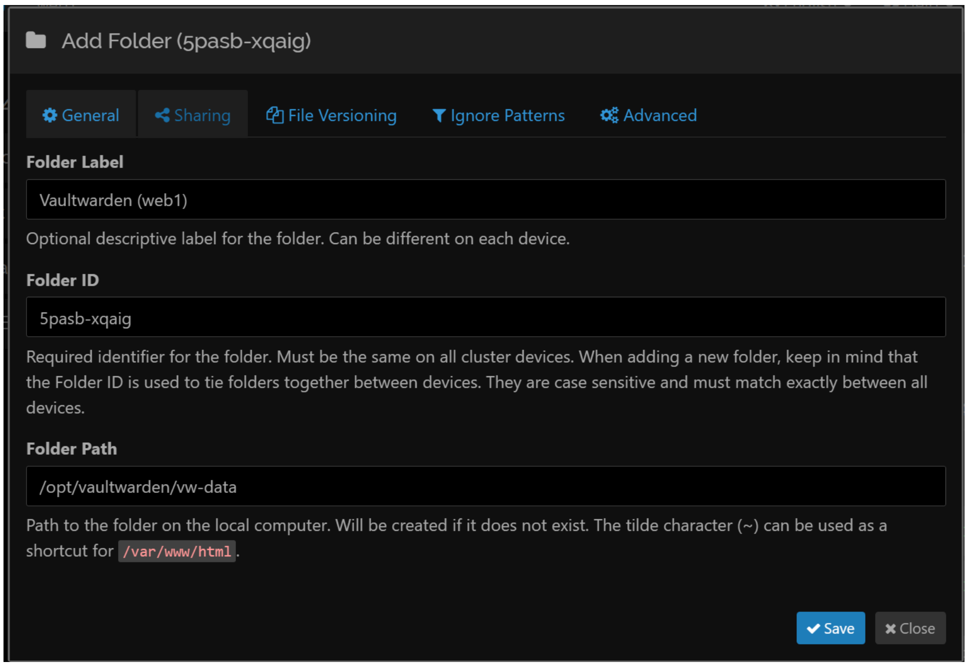
- The permissions depend on which user runs the Syncthing service. In my case, this is `www-data`. We will need to ensure that this user has access to Docker for monitoring purposes:

```
# Stop the container briefly to ensure the folder is free:
cd /opt/vaultwarden
sudo docker compose down

# Grant the 'www-data' user full Read/Write access to the docker volume
# using ACLs (Access Control Lists). This persists even if Docker recreates files.
sudo setfacl -R -m u:www-data:rwx /opt/vaultwarden/vw-data
sudo setfacl -d -m u:www-data:rwx /opt/vaultwarden/vw-data

# Start the container again (in a detached mode to free up the shell):
sudo docker compose up -d
```

- Assuming you have Syncthing already deployed, add a new folder on web1:
  - **General** tab:
    - Label: Vaultwarden (web1)
    - Path: `/opt/vaultwarden/vw-data`
  - **Sharing** tab → if you already have your other Syncthing web servers defined, then tick the boxes.
  - **File versioning** tab→ It is recommended to set up 'Trash can versioning' for 30 days+.
  - Skip to the **Advanced** tab → Check **[x] Ignore Permissions** (Docker manages the ownership, Syncthing just moves the data bits).
  - Go back to the **Ignore Patterns** tab: (tick the box and once you complete the other tabs and go forward, copy paste this list):

```
icon_cache/
tmp/
```

```
temp/
*.sqlite3
*.sqlite3-wal
*.sqlite3-shm
*.log
```



Folder Label

Vaultwarden (web1)

Optional descriptive label for the folder. Can be different on each device.

Folder ID

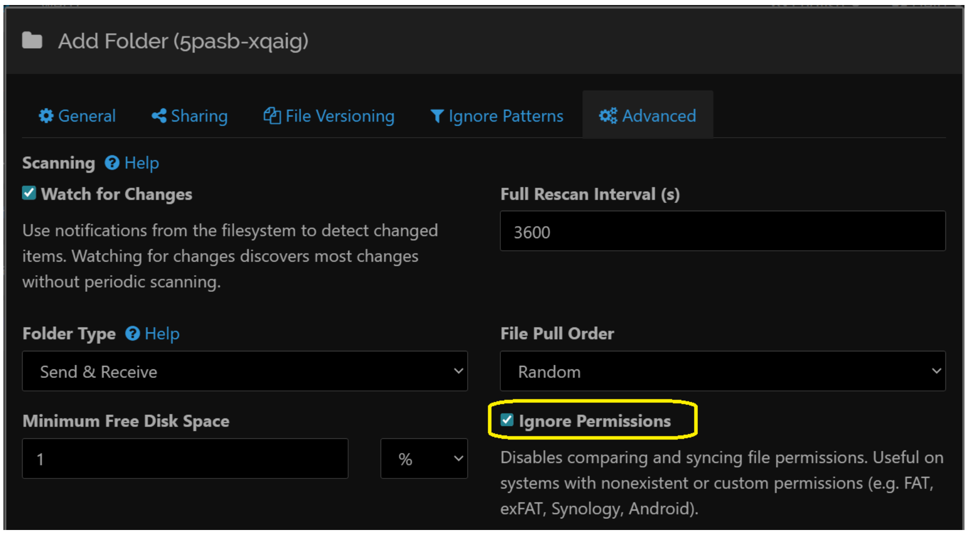
5pasb-xqaig

Required identifier for the folder. Must be the same on all cluster devices. When adding a new folder, keep in mind that the Folder ID is used to tie folders together between devices. They are case sensitive and must match exactly between all devices.

Folder Path

/opt/vaultwarden/vw-data

Path to the folder on the local computer. Will be created if it does not exist. The tilde character (~) can be used as a shortcut for `/var/www/html`.



Scanning

☑ **Watch for Changes**

Use notifications from the filesystem to detect changed items. Watching for changes discovers most changes without periodic scanning.

Full Rescan Interval (s)

3600

Folder Type

Send & Receive

File Pull Order

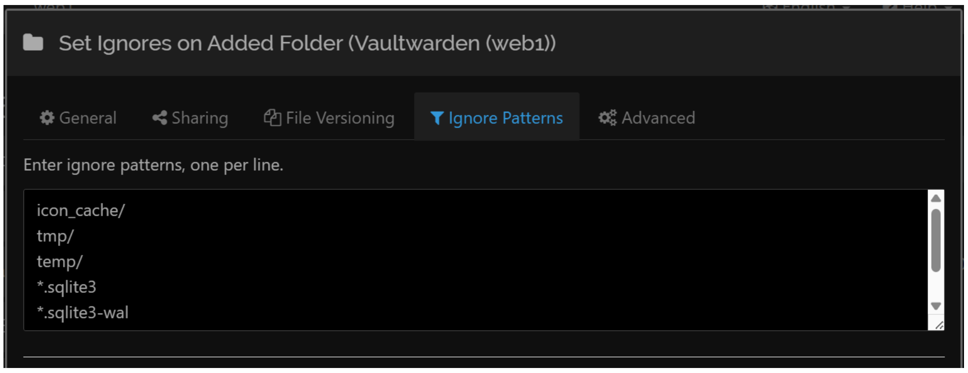
Random

Minimum Free Disk Space

1 %

☑ **Ignore Permissions**

Disables comparing and syncing file permissions. Useful on systems with nonexistent or custom permissions (e.g. FAT, exFAT, Synology, Android).

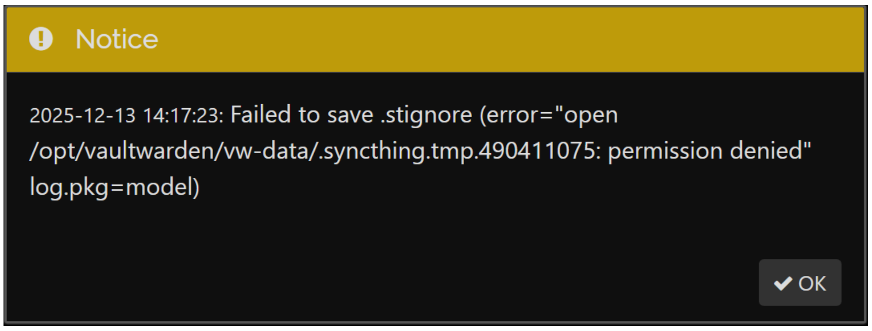


Enter ignore patterns, one per line.

```
icon_cache/
tmp/
temp/
*.sqlite3
*.sqlite3-wal
```

Troubleshooting Syncthing folder addition

- In case you add the sync job and immediately get a permission error like the one below, it means your permissions on the `/opt/vaultwarden` folder are not correct:



```
2025-12-13 14:17:23: Failed to save .stignore (error="open
/opt/vaultwarden/vw-data/.syncthing.tmp.490411075: permission denied"
log.pkg=model)
```

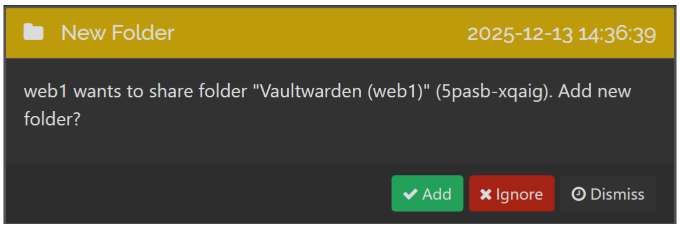
- While in my case, the user is www-data, in your case, it is likely different. How can you find out? Let's take a look:

```
 # List all running processes, filter 'syncthing' and filter out the command it self:
 sudo ps aux | grep syncthing | grep -v grep
```

- Then adjust the '`sudo setfacl -R`' and `'sudo setfacl -d`' commands above accordingly.
- For additional troubleshooting scenarios, you can check my **previous guide for Syncthing** that includes data loss situations and recovery options.

Add Syncthing on other web nodes

- SSH into your other web server nodes and set up the file permissions accordingly as we have done at the beginning of this Step.
- Open Syncthing on each of your other web nodes and accept the invitation:



web1 wants to share folder "Vaultwarden (web1)" (5pasb-xqaig). Add new folder?

- Set up the name, path, trash can, ignore permissions in the Advanced tab and then set up the file/folder patterns to be ignored, as we have done previously.

---

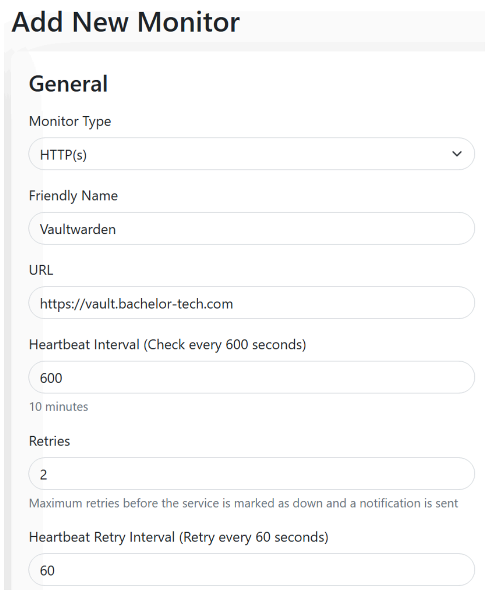
6. Set up Monitoring for Vaultwarden's Docker Container + Website using UptimeKuma

There are three types of monitors that would be good to set up. If you only have one site or do not have your 'witness' site set up yet, then you can have all these checks running of just one UptimeKuma instance.

Monitor A. Site 3 - Website Uptime

On Site 3 (VPS - witness site), you can monitor the service as a whole (e.g. does the website load regardless of which site it is being loaded from?).

- **Monitor Type**: `HTTP(s)`
- **Friendly name**: `Vaultwarden or similar`
- **URL**: Your actual URL
- **Heartbeat**: 30-600 seconds (1 to 10 minutes)
- **Retries**: 1-3
- **Heartbeat Retry:** a small amount, such as 30-60 seconds


Add New Monitor

General

Monitor Type: HTTP(s)

Friendly Name: Vaultwarden

URL: https://vault.bachelor-tech.com

Heartbeat Interval (Check every 600 seconds): 600

10 minutes

Retries: 2

Maximum retries before the service is marked as down and a notification is sent

Heartbeat Retry Interval (Retry every 60 seconds): 60

Monitor B. Site 1+2 Docker Container Monitor

For UptimeKuma to see the container, we will need to deploy a **Socket Proxy container.** The idea is to use the proxy as a 'gatekeeper' that provides read-only access to UptimeKuma on a specific port. We will need to apply this on **each web server node**.

- Create a new folder and a compose file:

```
# Create a directory for the proxy
sudo mkdir -p /opt/socket-proxy-container

# Go into that directory
cd /opt/socket-proxy-container

# Create the compose file
sudo nano docker-compose.yaml
```

- Enter the following:

```
services:
  docker-socket-proxy:
    image: tecnativa/docker-socket-proxy
    container_name: docker-socket-proxy
    restart: unless-stopped
    # High privilege is required to access the raw docker socket
    privileged: true
    ports:
      - "2375:2375"
    volumes:
      # We give it access to the host's docker socket (read-only)
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      # ENABLE specific read-only permissions
      - CONTAINERS=1 # Allows listing and checking container status
      - INFO=1       # Allows checking general docker info (optional)
      # BLOCK write permissions (security)
      - POST=0       # Blocks any commands that change state (restart/kill/cre
ate)
```

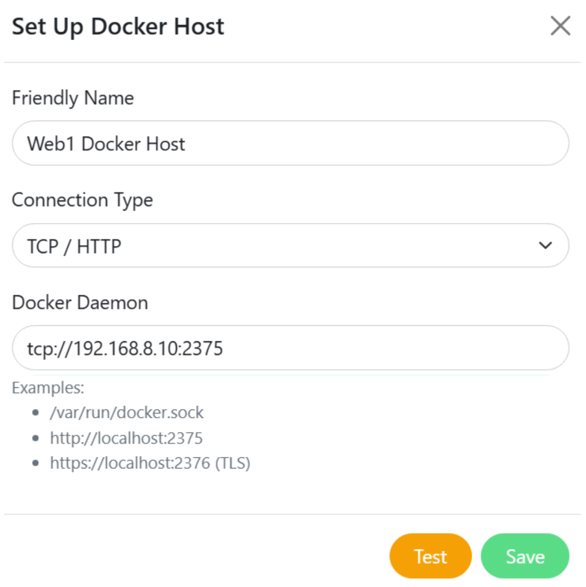
> 💡 Security Note: The socket proxy exposes read-only Docker API access on port 2375.
> In a homelab environment on a trusted LAN, this is generally acceptable. For stricter
> security, bind to a specific IP (`192.168.8.9:2375:2375`) or implement firewall rules to allow only your UptimeKuma host on port `2375`. The most secure method would be to bind it to `localhost` only and use SSH tunneling, instead.

- Then start the container:

```
cd /opt/socket-proxy-container
sudo docker compose up -d

# Check that it is running together with the Vaultwarden container:
sudo docker ps -a
```

- On Site1/Site2 UptimeKuma, add a new monitor:
    - **Monitor Type**: `Docker Container`
    - **Friendly name**: `Web1 - Docker Monitor - Vaultwarden`

- **Container name**: `vaultwarden` (as per how it's shown when you run `docker container ps -a` on your web server.
- **Docker host** - add a new one:
    - Friendly name: `Web1 Docker Host`
    - Connection type: `TCP / HTTP`
    - Docker Daemon: `tcp://192.168.8.9:2375` (use your local IP with port 2375)



Set Up Docker Host

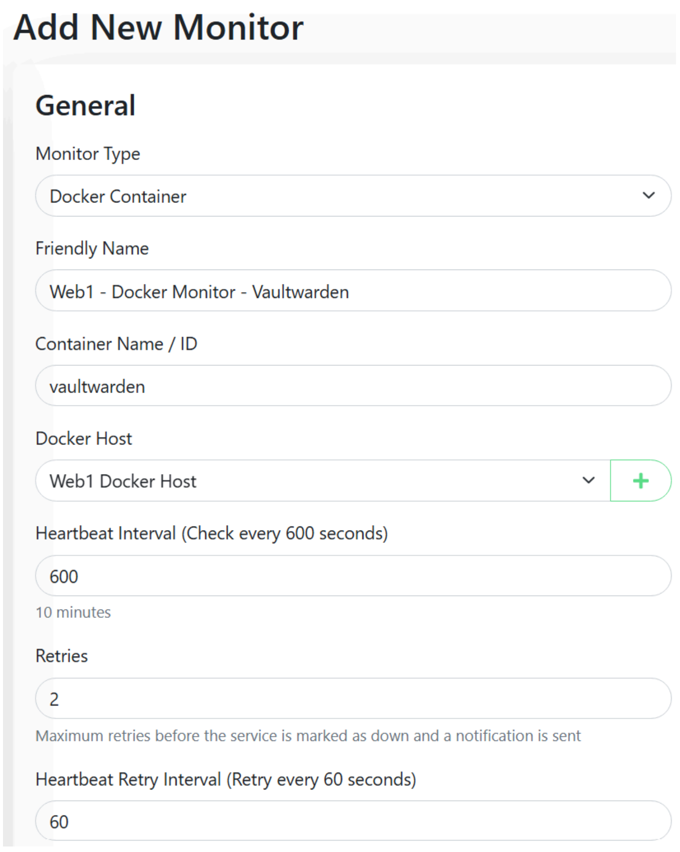
Friendly Name: Web1 Docker Host

Connection Type: TCP / HTTP

Docker Daemon: tcp://192.168.8.10:2375

Examples:
- /var/run/docker.sock
- http://localhost:2375
- https://localhost:2376 (TLS)

- The overall monitor for `web1` may look like this - remember to repeat this for each web server:


Add New Monitor

General

Monitor Type: Docker Container

Friendly Name: Web1 - Docker Monitor - Vaultwarden

Container Name / ID: vaultwarden

Docker Host: Web1 Docker Host

Heartbeat Interval (Check every 600 seconds): 600

10 minutes

Retries: 2

Maximum retries before the service is marked as down and a notification is sent

Heartbeat Retry Interval (Retry every 60 seconds): 60

Monitor C. Site 1+2 Syncthing Monitor

The other thing that could stop working over time is Syncthing itself. We can leverage Syncthing's API for that and push information regularly from each web server into UptimeKuma as a passive 'push' monitor.

- Since I have described this method already in my recent guide, please follow the steps from the the **Syncthing Web HA tutorial**.
- You will get notified whenever ANY of the sync jobs are down or even paused, as per your config settings.

---

7. Harden Vaultwarden with Fail2ban

On the VM that runs docker with our Vaultwarden container, we should install fail2ban and secure the docker container. In case you do not have it set up already, you can also secure SSH and your other sites.

Install fail2ban

```
# 1. Install Fail2ban
sudo apt update
sudo apt install fail2ban -y

# 2. Create a local configuration file (never edit jail.conf directly)
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Protect SSH on a custom port

- In this case, SSH port is set to a custom port 2222 under `/etc/ssh/sshd_config` .

```
sudo nano /etc/fail2ban/jail.local

# Un-commment these:
ignoreself = true

# Add your own IP range (oneself + LAN + docker subnet)
ignoreip = 127.0.0.1/8 ::1 192.168.0.0/16 172.20.0.0/16

# Comment out
# ignorecommand =

# Find an existing section called [sshd] and modify it as follows:

[sshd]
enabled = true
port    = 2222
mode    = aggressive
# The more modern method instead of reading text files
backend = systemd
maxretry = 3
bantime = 1h
```

- If you have not started the fail2ban client yet, run `sudo fail2ban-client start` . Otherwise run `sudo fail2ban-client reload` for the changes above to kick in.

Protect Vaultwarden’s docker container

- In your docker-compose.yml file, ensure that you have the following parameters set:

```
# Ensure these three are somewhere in your 'environment':
environment:
    - LOG_FILE=/data/vaultwarden.log
    - LOG_LEVEL=info
    - EXTENDED_LOGGING=true
```

- After changing the values below, run `docker compose up -d` to apply changes.
- Let’s create a config that instructs `fail2ban` on what a failed login to Vaultwarden looks like:

```
sudo nano /etc/fail2ban/filter.d/vaultwarden.conf

[Definition]
# Matches "Admin login attempt failed" and "Invalid password for user"
failregex = .*Username or password is incorrect.*IP: <HOST>
```

- We will need to define a deny action:

```
sudo nano /etc/fail2ban/action.d/nginx-deny.conf

[Definition]
actionstart =
actionstop =
actioncheck =
actionban = printf "deny <ip>;\n" >> /etc/nginx/blocklist.conf; systemctl rel
oad nginx
actionunban = sed -i "/deny <ip>;/d" /etc/nginx/blocklist.conf; systemctl re
load nginx
```

- This path does not yet exist, let’s create it and set correct permissions for it:

```
sudo touch /etc/nginx/blocklist.conf
sudo chmod 664 /etc/nginx/blocklist.conf
```

- Then we should configure nginx’s config to read from it:

```
sudo nano /etc/nginx/conf.d/vaultwarden.conf

# Block Banned IPs from fail2ban
include /etc/nginx/blocklist.conf;
```

- Now we can finally add a new jail for Vaultwarden.

```
sudo nano /etc/fail2ban/jail.local

[vaultwarden]
enabled = true
# Read logs directly from Docker
backend = systemd
# Which container to watch
journalmatch = CONTAINER_NAME=vaultwarden
# The attacker's port, not the internal port
port    = 80,443,8081
filter  = vaultwarden
# The 'chain=DOCKER-USER' blocks the IP before Docker forwards it
action  = nginx-deny
maxretry = 3
bantime  = 300m # 5min
findtime = 300m # 5 min
```

- Reload the fail2ban service - `sudo fail2ban-client reload` .
- Then let’s run a real test: Attempt some failed login attempts from your phone while on mobile network. Attempt some bad logins and refresh the page. You should see your custom 403 page. Then check: `sudo fail2ban-client status vaultwarden` . Then you can remove your banned IP by running `sudo fail2ban-client set vaultwarden unbanip 1.2.3.4` on the affected web server.

Troubleshooting fail2ban for Vaultwarden on Docker

- Ensure that the log is seeing repeated failed login attempts: `sudo docker logs --tail 20 vaultwarden`
- Compare the errors with the filter we created earlier - check `sudo nano /etc/fail2ban/filter.d/vaultwarden.conf` .
  - In case you make modifications, restart the fail2ban service with `sudo fail2ban-client restart` .
- Run a dry-run using `sudo fail2ban-regex systemd-journal /etc/fail2ban/filter.d/vaultwarden.conf`
- Is the count increasing but the connection with the device is not being blocked?
  - Ensure that in your nginx config, you have this line included: `include /etc/nginx/blocklist.conf;` , as otherwise, whatever is there to be blocked will be ignored.
  - Confirm that your nginx-deny config exists, such as by running `sudo nano /etc/fail2ban/action.d/nginx-deny.conf` .

Ensure local + Cloudflare traffic does not get blocked

If you are using Cloudflare or another proxy-like traffic management that provides you with CDN and traffic filtering and an attacker tries to log into your instance of Vaultwarden, nginx will end up blocking the IP address of the proxy server from Cloudflare rather than their actual one. Such behavior is undesirable, as it would then block legitimate traffic coming from the proxy to your nginx server.

The solution is to whitelist the local and Cloudflare servers as shown on their website. Yet the list changes (not often but from time to time) and it would be tedious to keep it up to date manually. So let's automate it!

- Create a folder and script:

```
sudo mkdir /opt/cloudflare-proxies
sudo nano /opt/cloudflare-proxies/update-cloudflare-ips.sh
```

```
#!/bin/bash

# Download Cloudflare IPs
echo "# Cloudflare IPs - Auto Updated" > /etc/nginx/snippets/trusted-proxies.conf

# Internal IPs - modify this to reflect your local subnet
echo "set_real_ip_from 192.168.0.0/16;" >> /etc/nginx/snippets/trusted-proxies.conf

# IPv4
for ip in $(curl -s https://www.cloudflare.com/ips-v4); do
        echo "set_real_ip_from $ip;" >> /etc/nginx/snippets/trusted-proxies.conf
done

# IPv6
for ip in $(curl -s https://www.cloudflare.com/ips-v6); do
        echo "set_real_ip_from $ip;" >> /etc/nginx/snippets/trusted-proxies.conf
done

# Headers
echo "real_ip_header X-Forwarded-For;" >> /etc/nginx/snippets/trusted-proxies.conf
echo "real_ip_recursive on;" >> /etc/nginx/snippets/trusted-proxies.conf

# Reload Nginx as part of the script
sudo systemctl reload nginx
```

- Run this script to verify it works as expected, then make it executable:

```
# Run it
sudo bash /opt/cloudflare-proxies/update-cloudflare-ips.sh

# You should see the IPs from the script in there
cat /etc/nginx/snippets/trusted-proxies.conf
```

```
# Make the script executable
sudo chmod +x /opt/cloudflare-proxies/update-cloudflare-ips.sh

# Add it to run weekly via a cron job
sudo crontab -e

# Add the following entry in there
0 4 * * 1 /bin/bash /opt/cloudflare-proxies/update-cloudflare-ips.sh
```

- Add it into your virtual host on nginx:

```
sudo nano /etc/nginx/conf.d/vaultwarden.conf
 # Add the text below under the server { directive:

# Whitelist Cloudflare proxies & internal subnet:
include /etc/nginx/snippets/trusted-proxies.conf;
```

- Double check the syntax and reload nginx:

```
sudo nginx -t
sudo systemctl reload nginx
```

---

8. Bonus: Customize the 403 Forbidden page on Nginx

While you might have a custom page on your reverse proxy side for when your nodes are down, you may not have one for nginx. Let’s create it.

- Create a custom snippet configuration file that we will then include with our nginx config:

```
sudo nano /etc/nginx/snippets/custom-error-403.conf
```

```
        error_page 403 /custom-403.html;
        location = /custom-403.html {
                allow all;
                root /var/www/html/;
                internal;
                sub_filter 'SERVER_ID' $hostname;  # Replace placeholder with h
ostname
                sub_filter_once on;
        }
```

- In the custom 403 error file, we point to the `/var/www/html` folder, which is typical for Ubuntu and Debian installations. Some other flavors may use `/usr/share/nginx/html` . The choice is yours :)
- Then let's create the actual file:

```
sudo nano /var/www/html/custom-403.html
```

- The template I downloaded from **this source** - kudos to dr5hn.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>403 Forbidden</title>
    <style>
        @import url("https://fonts.googleapis.com/css?family=Press+Start+2
P");

        html,
        body {
            width: 100%;
            height: 100%;
            margin: 0;
        }

        * {
            font-family: 'Press Start 2P', cursive;
            box-sizing: border-box;
        }

        #app {
            padding: 1rem;
            background: black;
            display: flex;
            height: 100%;
            justify-content: center;
            align-items: center;
            color: #54FE55;
            text-shadow: 0px 0px 10px;
            font-size: 6rem;
            flex-direction: column;
        }
        #app .txt {
            font-size: 1.8rem;
            text-align: center; /* This centers the text content */
            /* Removed justify-content and align-items as they don't apply here
*/
        }
        @keyframes blink {
            0% {
                opacity: 0;
            }

            49% {
                opacity: 0;
            }

            50% {
                opacity: 1;
            }
```

```
          100% {
              opacity: 1;
          }
      }

      .blink {
          animation-name: blink;
          animation-duration: 1s;
          animation-iteration-count: infinite;
      }
   </style>
</head>

<body>
   <div id="app">
      <div>403</div>
      <div class="txt"> Blocked by fail2ban (SERVER_ID)<span class="blin
k">_</span> </div>
   </div>
</body>

</html>
```

- Ensure that the html file is accessible to the user that executes nginx - in my case, this was `www-data`:

```
sudo chown www-data:www-data /var/www/html/custom-403.html
```

- Now you can edit any of the sites where you would like to apply this config (such as in `/etc/nginx/conf.d/vaultwarden.conf`) and add a snippet in there:

```
sudo nano /etc/nginx/conf.d/vaultwarden.conf

# Find the server directive and add the following:
include /etc/nginx/snippets/custom-error-403.conf;
```

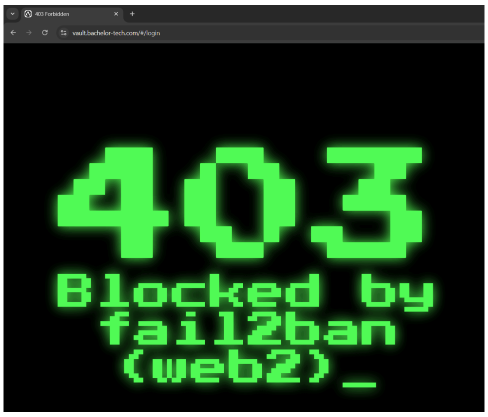
- For completeness, here is the full Nginx Vaultwarden config file:

```
server {
    # We listen on 80 because HAProxy handles the SSL/HTTPS before it gets here.
    listen 8081;

    # Replace with your actual FQDN
    server_name vault.yourdomain.tld;

    # Handled below by the whitelisting script already - do not duplicate!
    # Replace OPNSense's IP with the actual one to block the right users with fail2ban:
    # set_real_ip_from 192.168.0.0/16; real_ip_header X-Forwarded-For; real_ip_recursive on;

    # Whitelist Cloudflare proxies & internal subnet:
    include /etc/nginx/snippets/trusted-proxies.conf;

    # Find the server directive and add the following:
    include /etc/nginx/snippets/custom-error-403.conf;

    # Block Banned IPs from fail2ban
    include /etc/nginx/blocklist.conf;

    # Allow large attachments (optional, but good for saving PDFs/Images in vault)
    client_max_body_size 128M;

    location / {
        proxy_pass http://127.0.0.1:8000;

        # WebSocket support (used by /notifications/hub)
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Pass headers so Vaultwarden knows the real IP of the user, not just "127.0.0.1"
        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;
    }
}
```

- Then remember to reload the nginx service

```
sudo nginx -t
sudo systemctl reload nginx
```

- Run a test. In my case (zoomed in a bit), it looks like this - while indicating which web server blocked it. For production use, you may wish to modify it.



403
Blocked by fail2ban (web2)_

As mentioned before, if you have more nginx servers that you spread the load amongst, you would need to apply these steps on each of them.

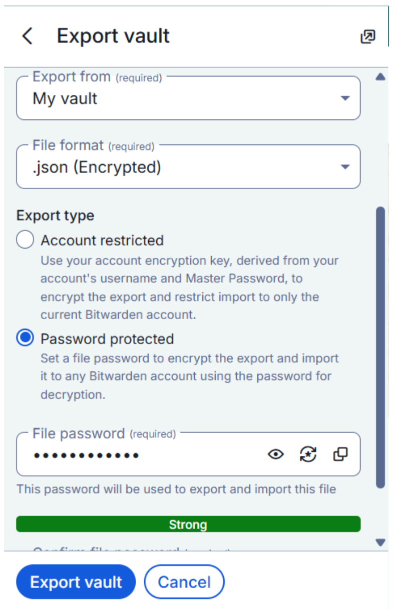
9. Migrate your data from Bitwarden to Vaultwarden

Congratulations, the hardest part is done! Your infrastructure is in place including detailed monitoring on a granular level. Let's migrate your existing data from Bitwarden. How can you move them over?

Migrating your Bitwarden data

Each user (you, your spouse, kids, etc.) imports their own private items.

- **Export:** Log in to your existing Bitwarden Web Vault.
    - Go to **Tools** > **Export Vault**.
    - Select format **.json (Encrypted)** if you know how to decrypt it, or **.json** (Plaintext). *Warning: Plaintext is readable, handle with care. You can import and encrypted file just fine.*



Export vault

Export from (required): My vault

File format (required): .json (Encrypted)

Export type

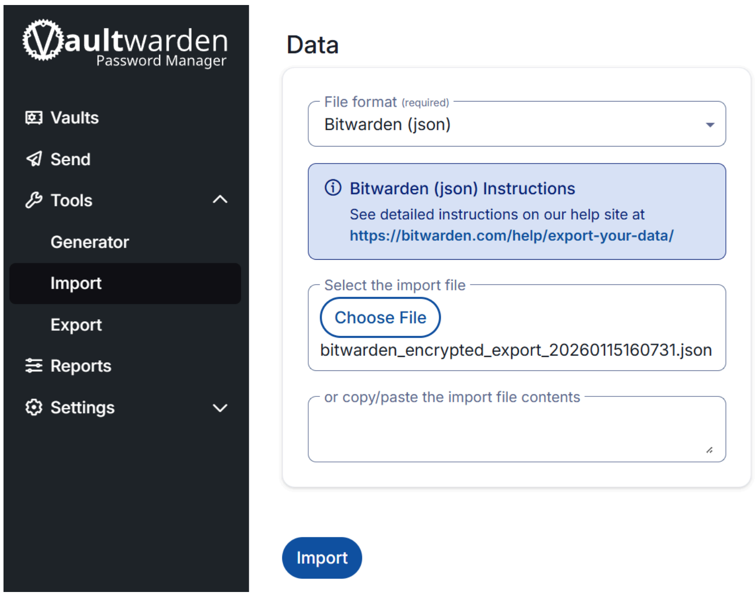
- ○ Account restricted
  Use your account encryption key, derived from your account's username and Master Password, to encrypt the export and restrict import to only the current Bitwarden account.
- ● Password protected
  Set a file password to encrypt the export and import it to any Bitwarden account using the password for decryption.

File password (required): ••••••••••••

This password will be used to export and import this file

Strong

- **Import:** Log in to your new Vaultwarden instance.
    - Create your user account (ensure `SIGNUPS_ALLOWED=true` in the `docker-compose.yml` file at least at the beginning for your own account, the others you can invite).
    - Go to **Tools** > **Import Data**.
    - Select **Bitwarden (json)**.
    - Paste the content or upload the file. If you are using the recommended encrypted .json file, you will be prompted for a password.



Data

File format (required)
Bitwarden (json)

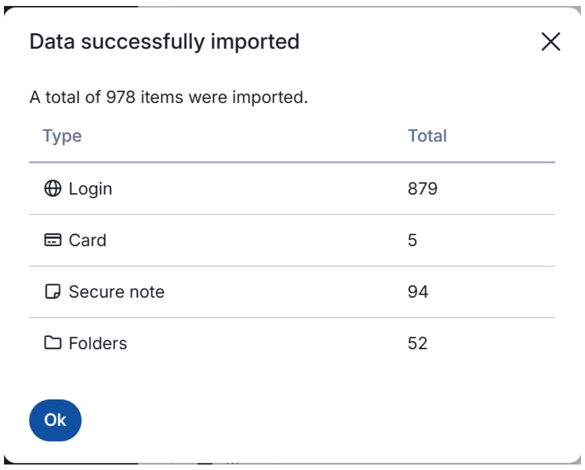
Bitwarden (json) Instructions
See detailed instructions on our help site at https://bitwarden.com/help/export-your-data/

Select the import file
Choose File
bitwarden_encrypted_export_20260115160731.json

or copy/paste the import file contents

Import

- Give it some time without refreshing the page for the credentials to import.
- You will then be greeted with a confirmation that the import has finished.



Data successfully imported

A total of 978 items were imported.

Type	Total
Login	879
Card	5
Secure note	94
Folders	52

Ok

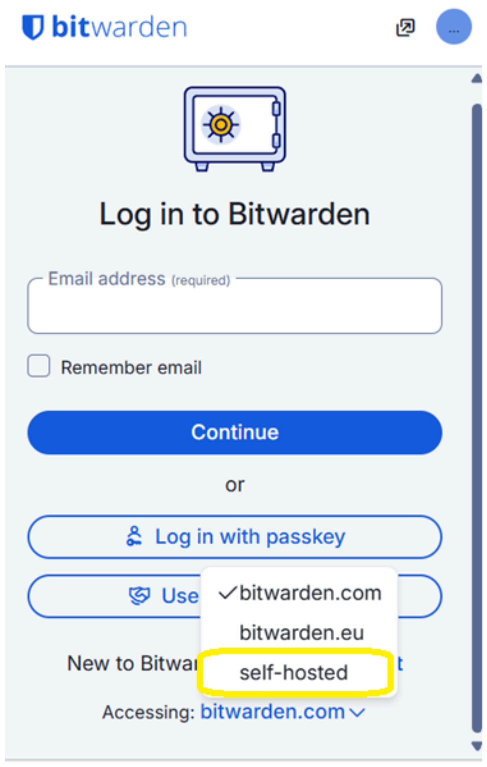
💡 Moving an Organization (or a Family org) is done in the same way as the personal vault, you just need to export them and import them separately. Collections are re-created automatically during the import, you do not need to create them before the import.

Handling special items in Vaultwarden

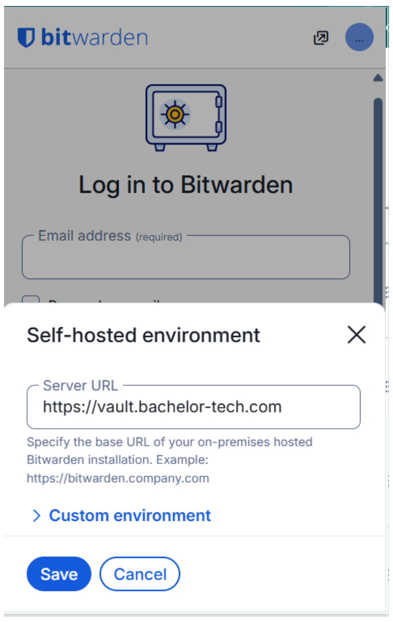
- **SSH Keys** are stored as standard "Items" in the database (similar to Logins or Cards). They **will** be included in the JSON export and should appear in Vaultwarden automatically.
- **Attachments** - standard JSON exports **do not** contain file attachments (images, PDFs, keys attached as files). You must move these manually. While a `.zip` export that includes attachments is available in `Bitwarden`, importing this into a self-hosted Organization often fails or isn't fully supported by the importer yet. Manual is the safest bet for integrity.
- **Sends** (temporary links) - these **cannot** be exported. If you have active Sends, you will need to manually recreate them in the new instance.

Log into your Vaultwarden using Bitwarden browser plugin

- If, like many other users, you use the browser plugin for Bitwarden, you can also use it for Vaultwarden.
- Simply log out your Bitwarden account and, as shown below, click on the 'Accessing' option and choose 'self-hosted'.



- Enter the URL of your instance.



Server URL
https://vault.bachelor-tech.com

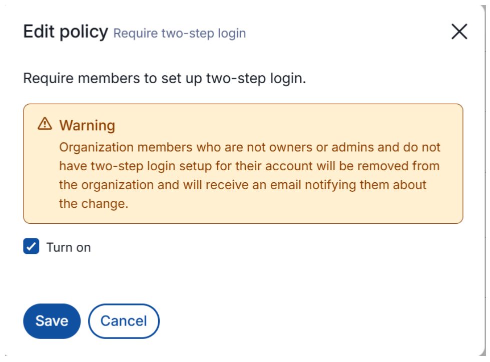
Specify the base URL of your on-premises hosted Bitwarden installation. Example: https://bitwarden.company.com

💡 Once your initial accounts are set up and no more sign ups are required, remember to open each `docker-compose.yml` file on each web server and ensure that `SIGNUPS_ALLOWED=true` are set to true. This is especially sensitive if your Vaultwarden instance is public-facing, as otherwise, anyone could create accounts and use your service, including bots.

Set up user/org policies

Before inviting users, it is vital to set some 'ground' rules for authentication and other security policies.

- Go to Settings → Policies → Require two-step login. Turn it on.

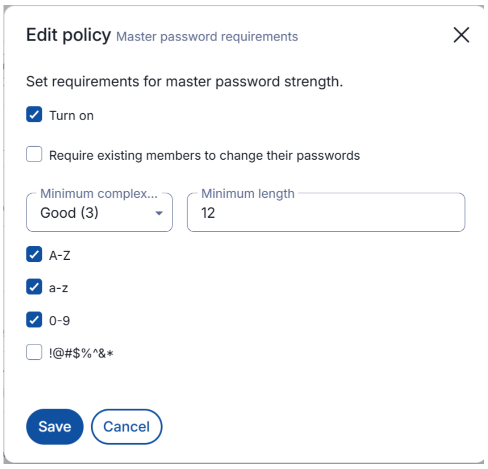

Edit policy Require two-step login

Require members to set up two-step login.

Warning
Organization members who are not owners or admins and do not have two-step login setup for their account will be removed from the organization and will receive an email notifying them about the change.

☑ Turn on

- Similarly, set up Master password requirements. Their length matters way more than complexity. Minimum recommended is 12 characters (marked as 'Good (3)' in Vaultwarden).
- Based on your preferences, you can enforce additional policies.


Edit policy Master password requirements

Set requirements for master password strength.

- [x] Turn on
- [ ] Require existing members to change their passwords

Minimum complex...: Good (3)
Minimum length: 12

- [x] A-Z
- [x] a-z
- [x] 0-9
- [ ] !@#$%^&*

Invite other Vaultwarden members

Since this is a new server, your family members technically need "new" accounts.

- Have family members sign up on your new Vaultwarden URL.
- Go to your **Organization** > **Members** > **Invite Member**.
- Type their email (must match what they signed up with). Verify that they received an email based on the SMTP settings you placed in your `docker-compose.yml` file previously.
- Once they accept the invite, go back to **Members** and 'Confirm' them in the Members tab.
- Assign them to the correct Collections (e.g. children manage only their own collection).

Feel free to play with additional settings in Vaultwarden web UI. Are there are additional considerations to take into account? How about backup and restoration? Or how can you go about updating your Vaultwarden instance? We will review these in our last Step.

---

10. Backups, Restoration & Additional Security Considerations

In this article, I did not cover how to best harden your web and DB servers. It is assumed that it is already done.

Backup & Recovery

While this set up is fully HA in terms of the web and DB servers being available on two sites (with a Galera witness on a VPS outside), HA does not equal backups. In the event of a breach (like ransomware) or a cascading service failure, you will need a form of archived backups, ideally going back a few months, so that you can determine the nearest and still safest version of your data.

- The recommended approach is to follow the 3-2-1 principle, i.e. by having 3 total copies of your data (the original plus two backups), storing them on 2 different media types (e.g. SSDs and cloud-based), and keeping at least 1 copy in an **off-site location** to protect against data loss from hardware failure, cyberattacks, or natural disasters.
- In addition, it is highly advisable to have a copy stored not just off-site but also **off-line** from secure threats. Because consider what happens if you lose all your passwords and stored SSH keys! Proper care needs to be given, even if it is not done as often. In a homelab environment, an example could be copying an encrypted version of the exported Vaultwarden data onto a memory stick that is stored in your parents' house twice a year.
- Components to back up:

Component	Location	Priority	Notes
Vaultwarden data directory	`/opt/vaultwarden/vw-data/`	Critical	Contains RSA keys, attachments, icons, config
MariaDB/Galera database	Vaultwarden database	Critical	Contains all vault entries, users, organizations
Docker Compose file	`/opt/vaultwarden/docker-compose.yml`	High	Contains your environment configuration
Nginx virtual host	`/etc/nginx/conf.d/vaultwarden.conf`	Medium	Can be recreated but saves time

Component	Location	Priority	Notes
SSL certificates	Managed by ACME/OPNsense	Low	Can be regenerated

Check out my **previous article on how to back up VMs and LXCs from Proxmox onto a GCP's archive storage**!

Recovery Situations

Below are three common scenarios that may occur:

- **Scenario A**: Single Web Node Failure If one web server fails but others are operational:
    - Restore the VM from Proxmox backup (or rebuild from template)
    - Syncthing will automatically sync `/opt/vaultwarden/vw-data/` from healthy nodes
    - Verify RSA keys match: `md5sum /opt/vaultwarden/vw-data/rsa_key.*` (compare across nodes)
    - Start the container: `cd /opt/vaultwarden && sudo docker compose up -d`
    - Verify in HAProxy that the node rejoins the backend pool

- **Scenario B**: Database Corruption or Loss
    - Stop Vaultwarden containers on ALL web nodes to prevent writes: `cd /opt/vaultwarden && sudo docker compose down`
    - On your preferred (primary) Galera node, restore the database: `gunzip < /var/backups/vaultwarden/vaultwarden_YYYY-MM-DD_HHMMSS.sql.gz | mysql -u root -p vaultwarden`
    - Verify Galera sync status: `mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"`
    - Restart Vaultwarden containers on all nodes

- **Scenario C**: Complete Site Loss
    - If both sites are compromised (e.g., ransomware), do NOT connect backup media to infected systems.
    - Rebuild infrastructure from clean Proxmox templates

    - Restore database from off-site/offline backup
    - Restore `/opt/vaultwarden/vw-data/` from backup (this includes the RSA keys).
    - Update DNS if IP addresses changed - do not open a public DNS record but rather a local record (such as on OPNSense by utilizing the Unbound DNS service).
    - Have all users verify their vaults and re-authenticate devices

Testing Your Backups

Backups are worthless if you cannot restore from them. Schedule quarterly restore tests:

```
# On a test VM (not production!), verify database backup integrity:*

gunzip -t /var/backups/vaultwarden/vaultwarden_latest.sql.gz && echo "Ar
chive OK"

# Test actual restoration to a temporary database:*
mysql -u root -p -e "CREATE DATABASE vaultwarden_test;"
gunzip < /var/backups/vaultwarden/vaultwarden_latest.sql.gz | mysql -u ro
ot -p vaultwarden_test
mysql -u root -p -e "SELECT COUNT(*) FROM vaultwarden_test.users;"
mysql -u root -p -e "DROP DATABASE vaultwarden_test;"
```

Updating Vaultwarden

This part is simple, yet for completion, let’s cover it.

- It is always good to **look into the release notes** before updating in case some larger changes have been made that could break a dependency.
- Take a snapshot of your web node before you carry out any changes.
- Then proceed on each web node:

```
# Pull the latest image:
sudo docker pull vaultwarden/server
```

```
# Restart the docker container
cd /opt/vaultwarden
sudo docker compose down && sudo docker compose up --force-recreate
# Once it loads and you can see the new version has loaded, press 'd' to d
etach it
```

Securing other components of your stack

Yet if you are looking for some tips, look at my **previous article on how to deploy a DB Galera cluster** that touch on how you can use tools like `ufw` firewall, `fail2ban` for securing your `SSH` access.

Putting Vaultwarden behind a VPN

So what else could we consider? You could consider **NOT having a resolvable public DNS name for your Vaultwarden instance** and instead, have it available only internally on your LAN. Then, if you have a VPN client deployed (such as a WireGuard plugin in your OPNSense instance), you could make it available only via that VPN connection. Security by obscurity is still a thing!

On your phone and other mobile devices, you can then install the VPN client to reach your Vaultwarden server. Alternatively, if you do not often update your passwords, simply let it sync whenever you are on your LAN - when outside, your mobile device will work off its local cache, preserving access to all the passwords up to the time of your last sync.

What else could go wrong?

While this guide is already rather comprehensive, there are quite a few other things that could go wrong (mostly related to the infrastructure) that could affect your Vaultwarden's uptime. Examples include:

- **Galera cluster split-brain scenarios** - What happens if sites lose connectivity while both are accepting writes?
- **Certificate renewal failures** - ACME/Let's Encrypt certificates expire; what if renewal fails?

- **Syncthing out-of-sync states** - How to identify and resolve when folders get stuck in 'Syncing' state
- **HAProxy backend health check failures** - How to diagnose when health checks fail but the service appears to be running
- **WebSocket connection issues** - Users report 'offline' status in apps despite the site being reachable
- **Database migration version conflicts** - What if nodes are running different Vaultwarden versions with different schema expectations?

Let me know in the comments below in case you have come across any or if you would like me to expand on how to recover from these before you encounter them yourself 😇

This concludes our guide. I hope you enjoyed it.