poste.io-addons



Antonio Sanna
poste.io-addons
Copyright © 2026 Antonio Sanna

poste.io-addons

Home

- poste.io-addons 6
  - Features 7
    - External Mailboxes 7
    - Dark theme 9
    - Calendars and contacts over ActiveSync 9
    - ActiveSync logs 9
    - Backup and restore 10
    - Upstream fixes 10
  - Quick start 11
  - License 11

Installation

- Installation 12
  - Requirements 12
  - 1. Build the image 12
  - 2. Run it 12
  - 3. Existing installations: the /data volume 13
  - 4. Check that it works 13
  - Upgrading Poste.io 13
  - Uninstall 14

Choosing the add-ons

- Choosing the add-ons 15
  - Option 1: select-addons.sh 15
  - Option 2: edit the Dockerfile 15

- Before you leave an add-on out of an existing installation 15

External Mailboxes

- External Mailboxes 17
  - What it does 17
  - How it works 19
    - Log rotation 20
  - Security 20
  - Notes and limitations 21

Dark theme

- Dark theme for the admin 22
  - Behaviour 22
  - How it works 22
  - Found a page that stays light? 23

Calendars and contacts over ActiveSync

- Calendars and contacts over ActiveSync 24
  - What users get 24
  - Enabling calendars and contacts for a mailbox 24
  - Seeing whether a colleague is busy: read-only sharing 24
  - The default calendar 24
  - Setting up the phone 25
  - When the DAV server is off or broken 25
  - Installing on an existing installation: devices must resync 25
  - Removing the add-on 25
  - Upgrading Poste.io 25
  - Limitations 25

- How it works 26
- Z-Push issues worked around 26
- Troubleshooting 26
- Tested 26

ActiveSync logs

- ActiveSync logs in the admin 27
  - The page 27
  - Log rotation 28
  - Existing installations 28
  - Files 28
  - Removing it 28

Backup and restore

- Backup and restore 29
  - What is backed up 30
  - How it works 30
  - The encryption key 31
  - Destinations 31
    - SMB / Samba 31
    - SFTP 31
    - S3 31
    - A volume mounted by the host (NFS, a second disk, ...) 32
    - REST server, append-only (rest-server) 32
    - Verification 33
  - Backup jobs 33
  - Watching the backups 33

- Restoring 34
  - The list of the messages 34
  - Removing the backups of a job 35
  - From the command line 35
- Reading the backups without the server 35
- Rebuilding the whole server 37
- Good practice 38
- A backup of a server that is running 38
- Security 38
- If the mail server is compromised 39
- Files and folders 39
- Troubleshooting 39
- Limits 39
- Removing it 40

Upstream fixes

- Upstream fixes 41
  - 1. The learning scripts are world-writable 41
  - 2. Junk learning goes into a bucket nobody reads 41
  - 3. Every learned message stays in /tmp 41
  - Are they still needed? 41
  - Checking that learning works 42

Tests

- Tests 43
  - What runs 43
  - How a run works 43
  - What is checked 43

- Running the tests yourself 45

poste.io-addons





(https://github.com/atsanna/poste.io-addons/blob/main/LICENSE)



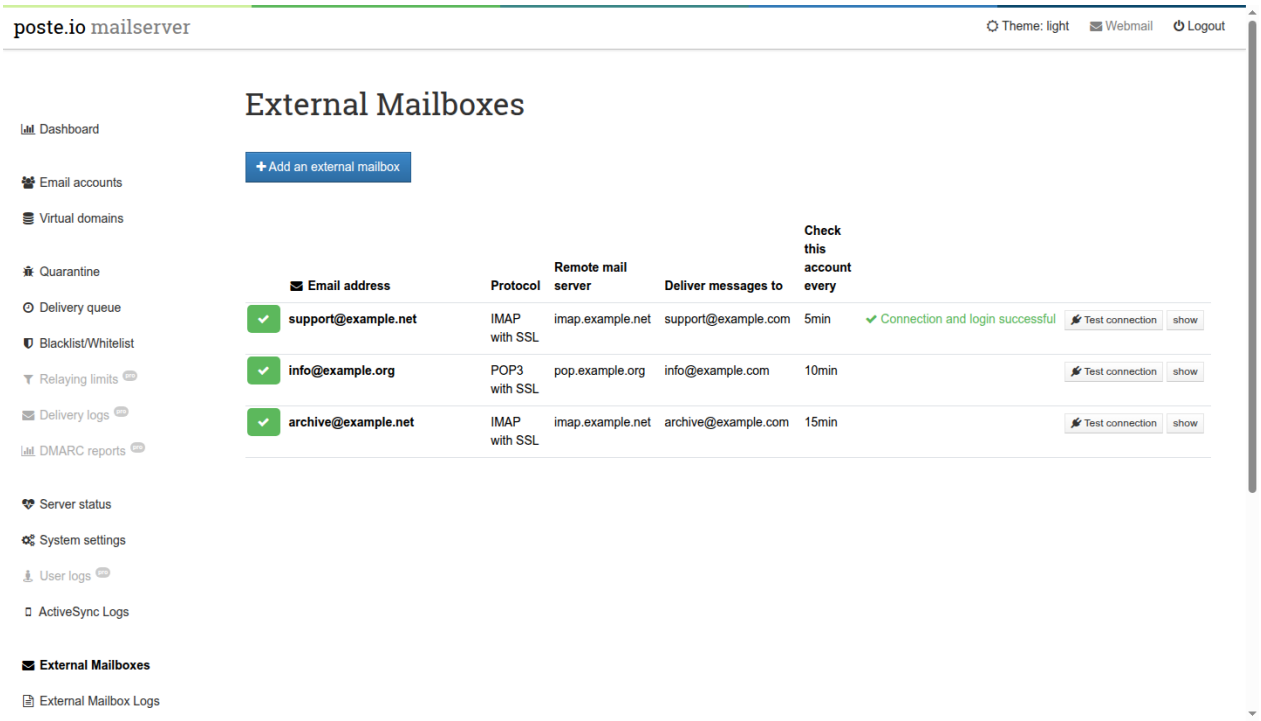
Add-ons for the Poste.io *(https://poste.io)* mail server, built as a Docker image on top of the official one:

- **External Mailboxes** — fetch mail from POP3/IMAP accounts hosted elsewhere and deliver it into Poste.io on a schedule, managed from the admin.
- **Dark theme for the admin** — automatic (follows the system) with a switch in the top bar.
- **Calendars and contacts over ActiveSync** — the phone's Exchange account gets the calendars and contacts of Poste.io's DAV server, not only the mail.
- **ActiveSync logs in the admin** — read the Z-Push logs with filters, and rotate them (Poste.io never does).
- **Backup and restore** — encrypted, scheduled backups of the configuration and the mail to SMB, SFTP, S3 or a mounted volume, set up and watched from the admin.
- Up to 2.5.19, **three upstream fixes** that make "move to Junk" actually teach rspamd, without leaving a copy of each message in `/tmp` — adopted by Poste.io in 2.5.20, so the add-on retired with release 1.2.0.

Take all of them or only the ones you want: each add-on is one section of the Dockerfile (choosing the add-ons).

English and Czech, like the Poste.io admin. Dockerfiles for the Pro and the open source edition: in production on **Poste.io 2.5.24 PRO**; the open source image is built and tested end to end on every push, with all the add-ons and with each one alone (tests).

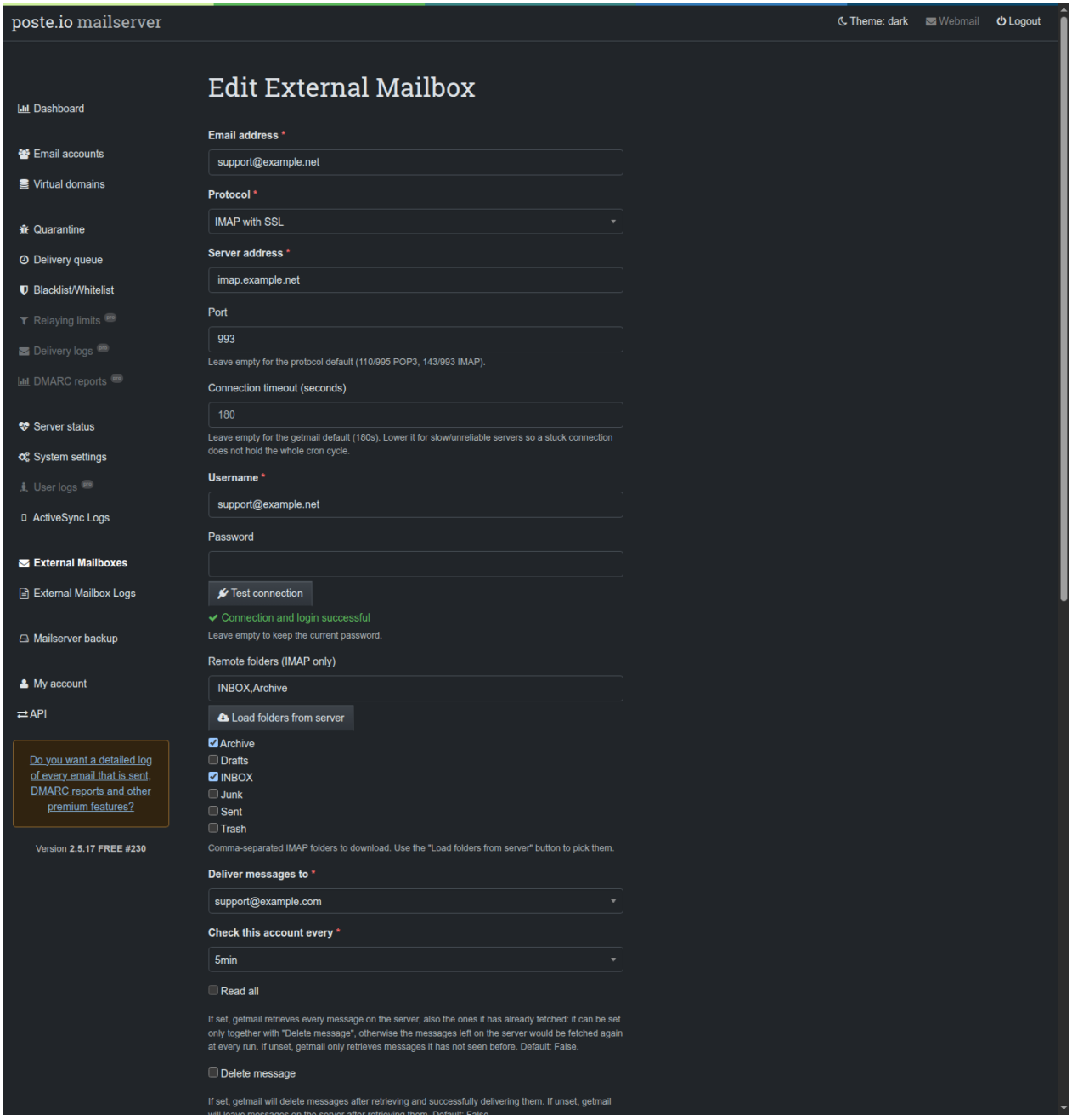
> *Not affiliated with or endorsed by Poste.io / Analogic. "Poste.io" is a trademark of its owners; this project only contains add-ons and no Poste.io code.*



Copyright © 2026 Antonio Sanna

Features

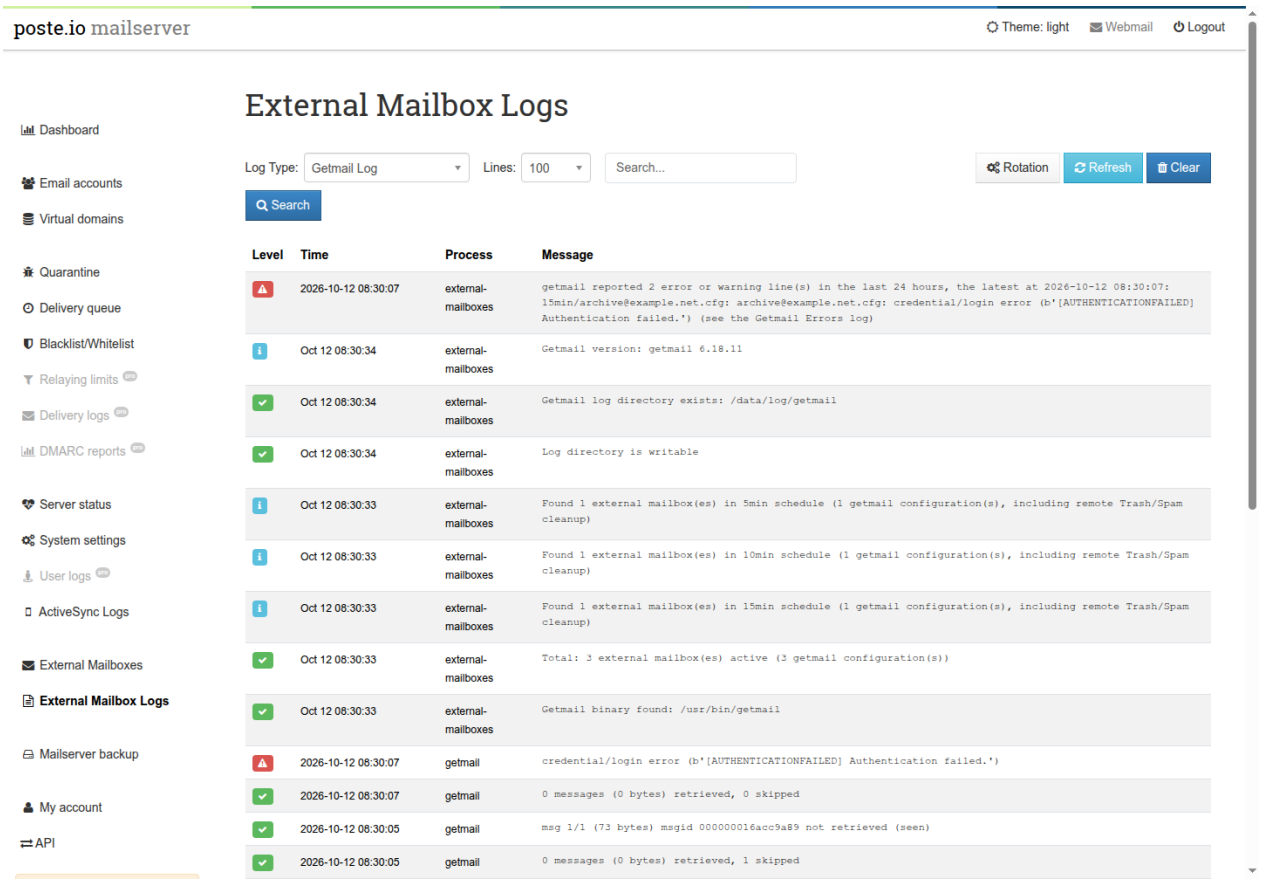
External Mailboxes



- POP3, POP3 over SSL, IMAP and IMAP over SSL, via getmail6 *(https://github.com/getmail6/getmail6)*, every 5–60 minutes.
- Delivery to any local mailbox **or redirect**: deliver an external `support@` box to a redirect and everyone in it gets the mail.
- IMAP folder picker (**Load folders from server**) and optional cleanup of the remote Trash/Spam folders with a retention period.
- **Test connection** from the form and from every row of the list.
- getmail options in the form: read all/new, delete, delete after N days, limits per run, delete messages over a size, port, timeout. "Read all" only together with "Delete message" (otherwise everything left on the server would be fetched again at every run).
- **Spam check**: every fetched message goes through rspamd before delivery, like mail received over SMTP (headers, spam into Junk, Bayes); without it fetched mail is not scanned by Poste.io at all.
- Logs page with the getmail log, the **getmail errors** (with a red line on top of the getmail view when there are recent ones), the cron log and a status view; **log rotation** set from the same page.
- Hardened: getmail runs as `mail`, never root; configuration files `0600`; TLS certificates verified; passwords never sent back to the browser; explicit CSRF tokens; strict validation of every value written to getmail.

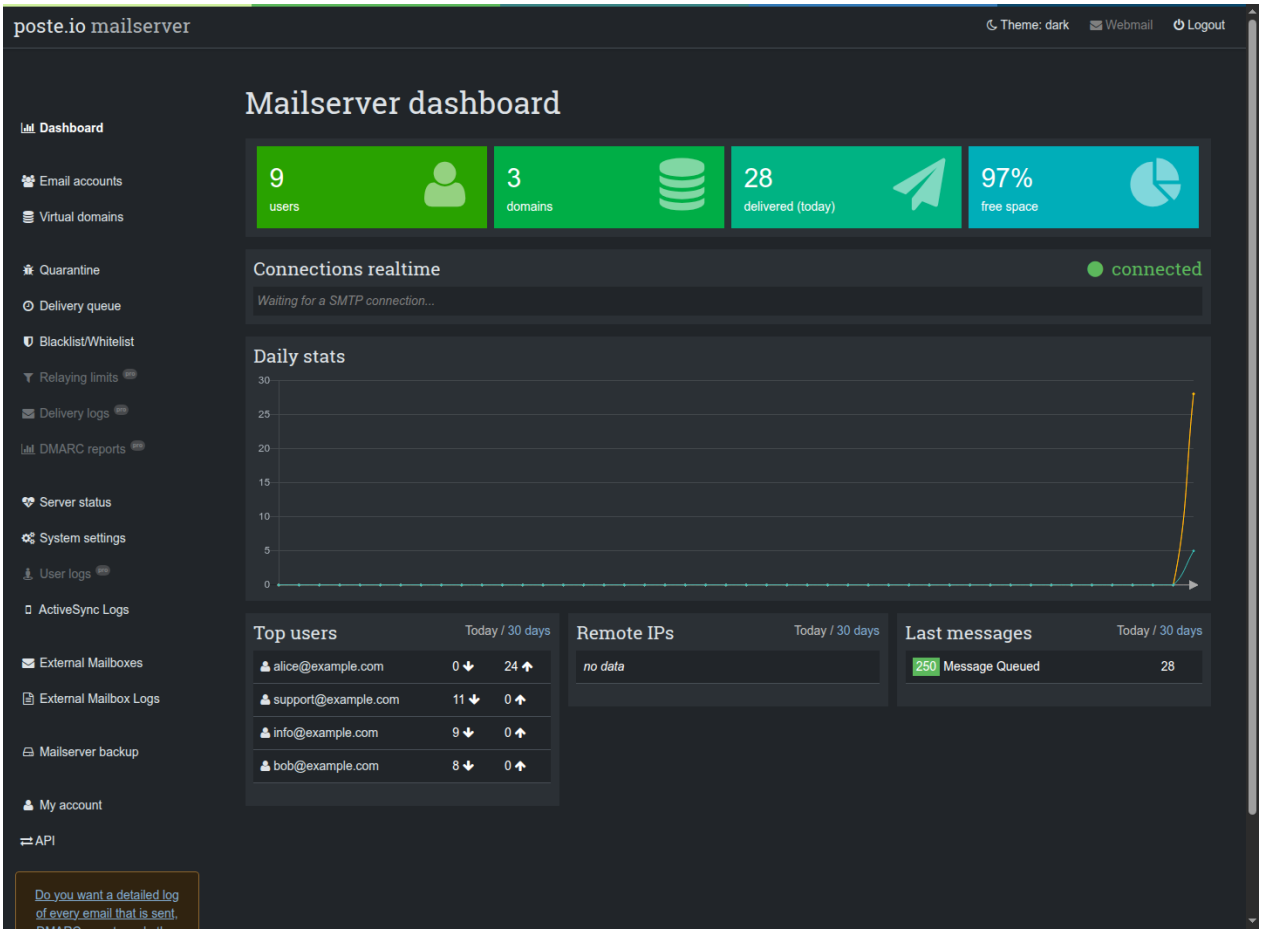
→ External Mailboxes

Copyright © 2026 Antonio Sanna



Copyright © 2026 Antonio Sanna

Dark theme



- Follows the system setting, or force light/dark from the top bar (remembered by the browser).
- No white flash on load; the login page follows the system too.
- Charts are recoloured and redrawn when the theme changes.
- The light theme is untouched: every rule applies to the dark theme only.

→ Dark theme

Calendars and contacts over ActiveSync

- One Exchange/ActiveSync account on the phone: mail, calendars, tasks and contacts, both ways, from Poste.io's built-in CalDAV/CardDAV server.
- Enabled per mailbox in the admin, as before (*Create calendar*, *Create address book*); a mailbox without them keeps mail only.
- A colleague's calendar shared **read-only** appears as an additional calendar: see whether someone is busy before booking them.
- If the DAV server is switched off or broken, mail keeps working.
- Fixes three Z-Push issues found while testing (an event refused by the server confirmed as saved, a crash on contacts without an address book, ActiveSync down with the DAV server).

→ Calendars and contacts over ActiveSync (existing installations: devices must resync once)

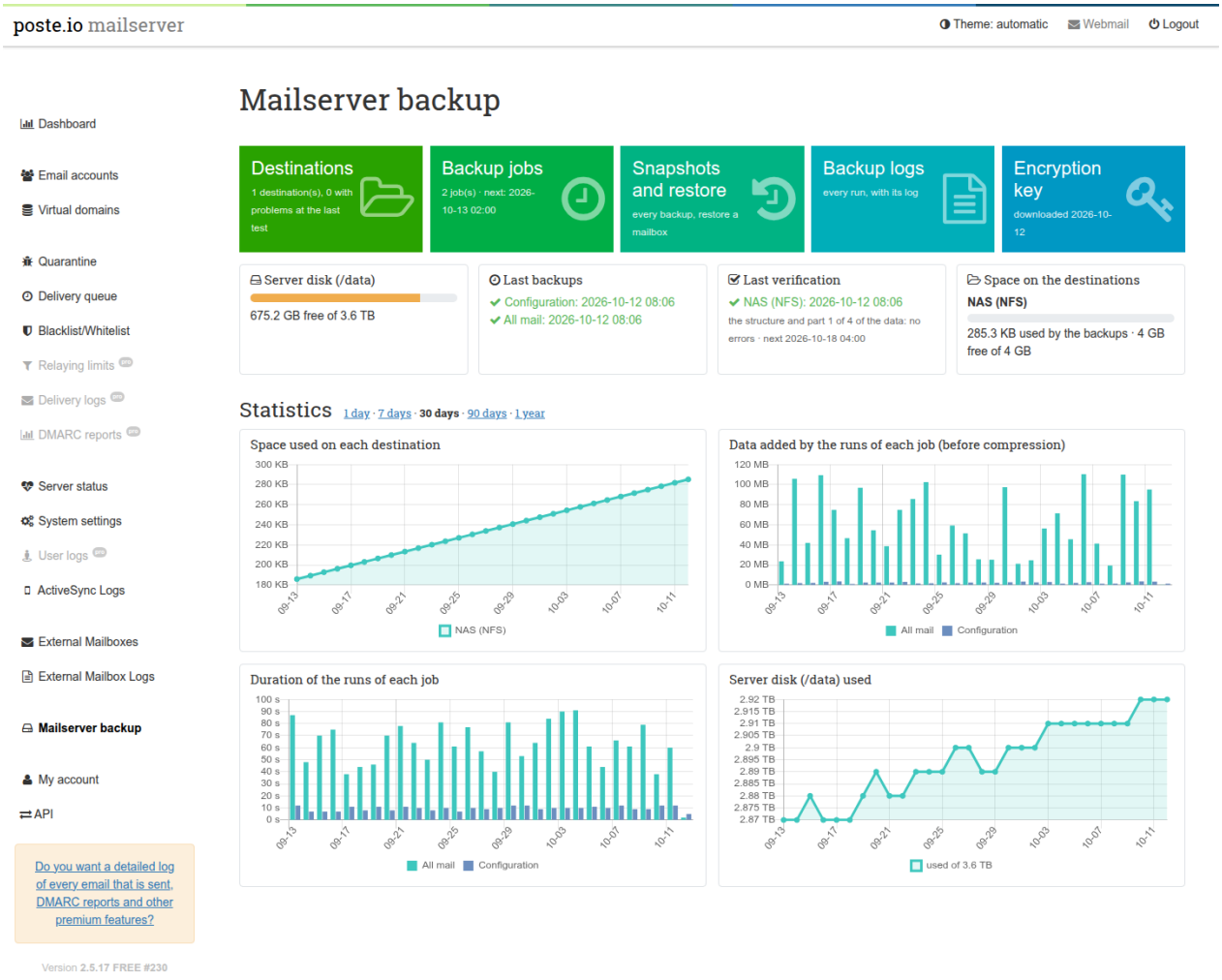
ActiveSync logs

- A page right after *User logs* with the Z-Push log and its errors log: filter by level, account and text, newest lines first.
- Log rotation set from the same page (Poste.io never rotates these logs): every day, week or month, old copies kept compressed.
- Read-only on purpose: the logs stay Z-Push's (`www-data`), as Poste.io wants them.

→ ActiveSync logs

Copyright © 2026 Antonio Sanna

Backup and restore



- The **configuration** (accounts, domains, settings, certificates, DKIM keys, calendars and address books, webmail settings, spam filter training) and **all the mail or chosen mailboxes**, every hour, day, week or month.
- To one or more destinations at once: **SMB/Samba, SFTP, S3**, a volume mounted by the host (NFS, a second disk), or restic's **REST server in append-only mode**, where a compromised mail server cannot delete the backups; a **Test** of each destination before saving it (reachable, empty folder, free space, a write).
- Made with restic *(https://restic.net)*: encrypted before leaving the server, deduplicated and compressed, so frequent backups store only what changed; keep rules per job (hourly, daily, weekly, monthly).
- Live progress, the log of every run, the space on each destination (yellow and red from limits you set), **statistics** over 30 days to a year, and a scheduled **verification** of each destination that reads the stored data again, a part at a time.
- **Restore a mailbox from the admin** into a new folder of the same or another mailbox: whole, some folders, the messages of a filter (dates, sender, subject), or **single messages** found in a searchable list; nothing already there is changed; remove the backups of a job; the backups read from **another machine with only the key file**; a **whole server rebuilt** on a new installation, each procedure run by the tests as documented.
- Hardened: the engine runs as `mail`, never root; settings validated again before use; SFTP only to a server whose SSH key was confirmed; passwords never sent back to the browser.

→ Backup and restore

Upstream fixes

Up to 2.5.19 moving a message to Junk did not train rspamd: the learning scripts were world-writable (Dovecot refuses to run them), they learned into a Bayes bucket that scanning never reads, and they left a copy of every message in `/tmp`. **Poste.io 2.5.20 ships all three fixes**, so the add-on is gone; on an older base build from release 1.2.0. The end-to-end tests still verify the learning behaviour on every build.

Copyright © 2026 Antonio Sanna

→ Upstream fixes

Quick start

```
git clone https://github.com/atsanna/poste.io-addons.git
cd poste.io-addons

# Pro (requires docker login to the Poste.io registry)
docker build -f Dockerfile-pro --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .
# or open source
docker build -f Dockerfile     --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .

# only some add-ons, e.g. the dark theme and calendars/contacts
sh select-addons.sh Dockerfile-pro dark-theme activesync-dav > Dockerfile.custom
docker build -f Dockerfile.custom --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .
```

Run the image as you run the official one (see `docker-compose.example.yml` *(https://github.com/atsanna/poste.io-addons/blob/main/docker-compose.example.yml)*). On an **existing** installation, read Installation first: the `/data` volume hides the folders the image prepares, and phones using ActiveSync must resync once.

License

MIT *(https://github.com/atsanna/poste.io-addons/blob/main/LICENSE)* © 2026 Antonio Sanna. The license covers the files of this repository only; Poste.io itself is licensed by its owners.

Copyright © 2026 Antonio Sanna

Installation

This repository does not ship a ready-made image: you build your own on top of the official Poste.io image, so you always know what is inside it.

Requirements

- Docker 20.10+ (BuildKit enabled by default).
- **Open source edition**: nothing else, the base image `analogic/poste.io` is public.
- **Pro edition**: a Poste.io Pro licence and access to the Poste.io registry (`docker login` with the credentials Poste.io gives you), because the base image is `poste.io/mailserver`.
- **Poste.io 2.5.24 or later**: its admin runs on Symfony 7, which the add-ons' pages are written for. For an older Poste.io use release 2.0.1.
- Tested with **Poste.io 2.5.24**: Pro in production, open source (`analogic/poste.io:2.5.24`) built and started in a test container. Other versions may work; the build stops on purpose if Poste.io changes the files the add-ons hook into (see Upgrading Poste.io).
- **Older versions, open source**: Docker Hub keeps only the latest `analogic/poste.io` tag, the previous ones disappear when a new version is out. To build an older release of this repository for the open source edition you need its base image still in your local Docker cache. The Pro registry keeps every version.

1. Build the image

```
git clone https://github.com/atsanna/poste.io-addons.git
cd poste.io-addons

# open source
docker build -f Dockerfile     --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .

# Pro (after docker login to the Poste.io registry)
docker build -f Dockerfile-pro --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .
```

To build only some of the add-ons, see Choosing the add-ons.

The build, one Dockerfile section per add-on:

1. **External Mailboxes**: installs `getmail6`, copies the admin pages into `/opt/admin` (new files only: no Poste.io file is replaced), rebuilds the Composer classmap (the Pro, and the open source 2.5.20, use a classmap-authoritative autoloader, new classes are invisible without it), appends the English and Czech translations, adds the two menu entries, prepares the getmail folders, `cron.sh` and the crontab of the `mail` user;
2. **Dark theme**: copies its files into `/opt/admin` and hooks them into `base.html.twig`;
3. **Calendars and contacts over ActiveSync**: installs `libawl-php`, configures Z-Push and adds its two backends;
4. **ActiveSync logs**: copies its page into `/opt/admin` (and rebuilds the classmap), adds its menu entry and translations, the rotation script, its nightly cron entry and a start script;
5. **Upstream fixes**: adopted by Poste.io in 2.5.20, nothing is applied any more (upstream fixes); on an older base use release 1.2.0.

2. Run it

Use the image exactly like the official one. A minimal example is in `docker-compose.example.yml` *(https://github.com/atsanna/poste.io-addons/blob/main/docker-compose.example.yml)*:

```
cp docker-compose.example.yml docker-compose.yml   # edit hostname, TZ, data path
docker compose up -d
```

Then open `https://<your host>/admin/`. The new entries **External Mailboxes**, **External Mailbox Logs**, **ActiveSync Logs** and **Mailserver backup** are in the left menu (super admins only); the theme switch is in the top bar.

Copyright © 2026 Antonio Sanna

3. Existing installations: the `/data` volume

Everything Poste.io keeps (mail, configuration, logs) lives in the `/data` volume, and **a volume hides whatever the image puts under `/data`**. On a brand new, empty named volume Docker copies the image content into it; with a bind mount (`./data:/data`) or an existing volume it does not. So, once, on an existing installation create the getmail folders and copy the runner:

```
docker exec mailserver sh -c '
  mkdir -p /data/var/getmail/5min /data/var/getmail/10min /data/var/getmail/15min \
           /data/var/getmail/30min /data/var/getmail/60min /data/var/getmail/disabled \
           /data/log/getmail &&
  chown -R mail:mail /data/var/getmail /data/log/getmail &&
  find /data/var/getmail /data/log/getmail -type d -exec chmod 0750 {} +'
docker cp external-mailboxes/cron.sh mailserver:/data/var/getmail/cron.sh
docker exec mailserver sh -c 'chown mail:mail /data/var/getmail/cron.sh && chmod 0750 /data/var/getmail/cron.sh'
```

`cron.sh` stays on the volume: when a new release of this repository changes it, copy it again the same way.

The getmail configuration files are on the volume too, written by the admin when a mailbox is saved. When a release changes what the admin writes in them (for example the rspamd check of fetched mail, which is a section of each file), existing mailboxes get it when they are **saved again** from the admin: open each external mailbox, leave the password empty (it keeps the saved one) and save. The CHANGELOG says when this is needed.

With **calendars and contacts over ActiveSync**, the first start of the new image changes Z-Push's folder identifiers: ask every device for a resync, otherwise phones end up with duplicated mail folders. The commands are in Calendars and contacts over ActiveSync. Later rebuilds and Poste.io upgrades with the add-on inside do not need it.

ActiveSync logs needs nothing by hand: its folder on the volume is created by a start script at every container start.

Backup needs nothing by hand either: its folder on the volume (`/data/var/backup`) and the encryption key are made by a start script (the key only once, at the first start). Then download the key from *Mailserver backup* > *Encryption key* and keep it outside the server (see Backup and restore).

4. Check that it works

```
# the mail user owns the schedule; root has none
docker exec mailserver crontab -u mail -l
docker exec mailserver crontab -u root -l      # "no crontab for root"

# after the first run (every 5 minutes for the 5min folder)
docker exec mailserver tail /data/log/getmail/getmail.log
```

In the admin, **External Mailbox Logs** shows the getmail log, the cron log and a status view (getmail version, folders, number of mailboxes).

```
# backup: the last results of the jobs (JSON), and the engine ticking
docker exec mailserver poste-backup status
```

In the admin, **Mailserver backup** says on its dashboard if the engine has not run for more than ten minutes or if the start script could not prepare its folder.

Upgrading Poste.io

Rebuild with the new version and recreate the container:

```
docker build -f Dockerfile-pro --build-arg POSTE_IO_VERSION=<new> -t my/poste.io:<new> .
```

The build is designed to **fail loudly** instead of producing a half-working image:

- `dark-theme-install.sh` stops if `base.html.twig` no longer has `</head>` on its own line or the right navbar list exactly once; (`grep -q` checks at the end of the block);
- `activesync-dav-install.sh` stops if any of the 19 Z-Push configuration lines it changes is not exactly as expected;
- `activesync-logs-install.sh` stops if the *User logs* menu entry is not found exactly once, followed by its `</li>`;
- `backup-install.sh` stops if a tool it needs (restic, rclone, the SSH tools, smbclient, sqlite3) or a PHP extension is missing, or if the end of the administrators' menu is not found exactly once.

Copyright © 2026 Antonio Sanna

Before each upgrade, check whether the upstream fixes are still needed: when Poste.io ships them, delete the two blocks from the Dockerfiles.

Uninstall

Rebuild/run the official image. The add-ons only add files; the data they leave in the volume is `/data/var/getmail` (mailbox configurations, with passwords), `/data/log/getmail`, `/data/var/activesync-logs`, the old copies of the Z-Push logs and `/data/var/backup` (the backup settings, with the passwords of the destinations, and the encryption key). Remove them if you no longer need them; keep the encryption key as long as you may need the backups. If you used calendars and contacts over ActiveSync, resync every device after the switch (the folder identifiers change back). To drop only some add-ons, see Choosing the add-ons.

Copyright © 2026 Antonio Sanna

Choosing the add-ons

The add-ons are independent: take all of them, or only the ones you want. Each one is **one section** of the Dockerfile, between two `ADD-ON` markers, and lives in its own folder of the repository.

Add-on	Name	Folder	What it changes in Poste.io
External Mailboxes	`external-mailboxes`	`external-mailboxes/`	adds admin pages and two menu entries, installs getmail6 and the `getmail-rspamd` filter (`/usr/local/bin`), adds a crontab for the `mail` user and the nightly rotation of its logs (`/etc/cron.d`, `/etc/logrotate.d`)
Dark theme for the admin	`dark-theme`	`dark-theme/`	adds a stylesheet and two templates, two `include` lines in `base.html.twig`
Calendars and contacts over ActiveSync	`activesync-dav`	`activesync-dav/`	19 lines of Z-Push configuration, two Z-Push backends, installs `libawl-php`
ActiveSync logs in the admin	`activesync-logs`	`activesync-logs/`	adds an admin page and one menu entry (after *User logs*), a start script (`/etc/cont-init.d`) and the nightly rotation of the Z-Push logs (`/etc/cron.d`, `/etc/logrotate.d`)
Backup and restore	`backup`	`backup/`	adds admin pages and one menu entry, the engine (`/usr/local/bin/poste-backup` and its library), a cron entry and a start script (`/etc/cron.d`, `/etc/cont-init.d`), installs restic, rclone, the OpenSSH client and smbclient

No add-on needs another one. The tests build the open source image with each add-on alone, as well as with all of them (see Tests).

Option 1: `select-addons.sh`

The script prints the Dockerfile with only the add-ons you name, everything else unchanged:

```
# Pro edition, only the dark theme and the backup
sh select-addons.sh Dockerfile-pro dark-theme backup > Dockerfile.custom
docker build -f Dockerfile.custom --build-arg POSTE_IO_VERSION=2.5.24 -t my/poste.io:2.5.24 .

# open source edition, only External Mailboxes
sh select-addons.sh Dockerfile external-mailboxes > Dockerfile.custom
```

It stops with an error on an unknown name, and lists the names it knows. With all six names it prints the Dockerfile exactly as it is.

Option 2: edit the Dockerfile

Each section looks like this:

```
#########################################################################
#  ADD-ON: Dark theme for the admin  [dark-theme]
#  docs/en/docs/dark-theme.md
#  To leave it out, delete this whole section (down to the next ADD-ON).
#########################################################################
...
```

Delete from that first `#####` line down to the line before the next `#####` / `ADD-ON` header (or to the end of the file for the last one). Keep the part above the first section (base image, labels).

Before you leave an add-on out of an existing installation

Removing an add-on from the image does not clean up what it left in the `/data` volume, and some add-ons need a step when they come or go:

- **External Mailboxes**: the mailbox configurations (with passwords) stay in `/data/var/getmail`, the log in `/data/log/getmail`. Delete them if you no longer need them.
- **Calendars and contacts over ActiveSync**: adding or removing it changes the identifiers of the ActiveSync folders, so **every device must resync** (Calendars and contacts over ActiveSync). Calendars and contacts themselves stay on the DAV server.

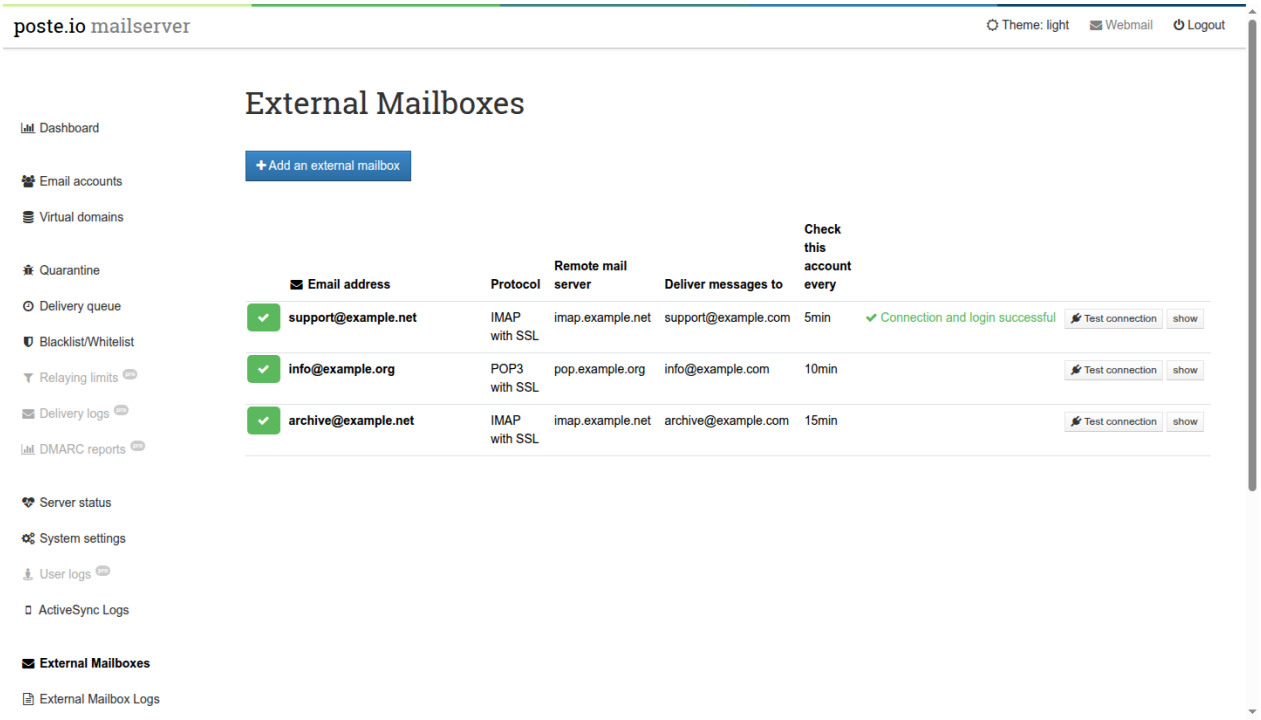
Copyright © 2026 Antonio Sanna

- **ActiveSync logs**: the rotation settings stay in `/data/var/activesync-logs`, the old copies of the Z-Push logs in `/data/log/z-push` (`*.log.1`, `*.log.N.gz`). Delete them if you no longer need them.
- **Backup**: the settings (with the passwords of the destinations) and the encryption key stay in `/data/var/backup`. The backups on the destinations are not touched: keep the key as long as you may need them (Backup and restore).
- **Dark theme**: nothing to do.

Copyright © 2026 Antonio Sanna

External Mailboxes

Fetches mail from mailboxes hosted elsewhere (POP3 or IMAP, with or without SSL) and delivers it into Poste.io, on a schedule, with getmail6 *(https://github.com/getmail6/getmail6)*. Everything is configured from the Poste.io admin (super admins only): **External Mailboxes** and **External Mailbox Logs** in the left menu.

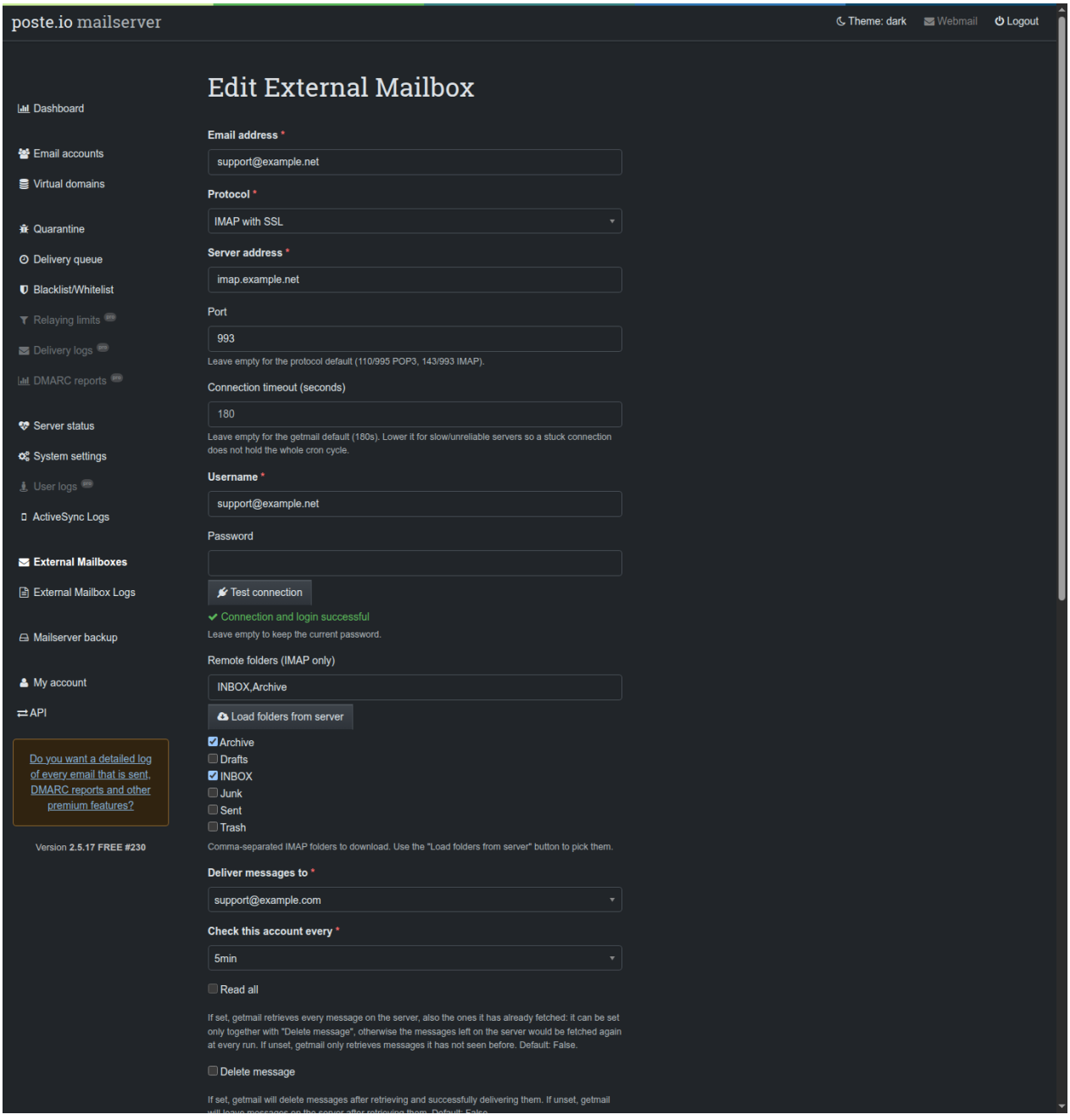


What it does

- **Protocols**: POP3, POP3 over SSL, IMAP, IMAP over SSL.
- **Schedule**: every 5, 10, 15, 30 or 60 minutes, or disabled.
- **Delivery**: to any local mailbox **or redirect** (alias). Delivery goes through Poste.io itself (`sendmail` → Haraka, inside the container), so filters and redirects apply: deliver an external `support@` mailbox to a local redirect and every member of the redirect gets a copy. In the destination menu redirects are shown as `support@example.com → alice@example.com (redirect)`.
- **IMAP folders**: choose which remote folders to download; **Load folders from server** lists the real folders of the account.
- **Remote cleanup (IMAP)**: optionally empties the remote Trash and/or Spam folders after a retention period (immediately, 1 day … 1 month). If a folder is both imported and cleaned up, "immediately" is raised to 1 day so a message can never be deleted before it is imported.
- **getmail options**: read all / only new messages, delete after download, delete after N days, add a `Received:` header, log to syslog, limits per run (messages and bytes, useful for the first import of a large mailbox), delete messages bigger than N bytes on the server, custom port and connection timeout.
- **"Read all" only with "Delete message"**: with "Read all" getmail fetches every message on the server, also the ones it already fetched; if they stay on the server, getmail warns that "all messages will be retrieved each time getmail is run". So "Read all" can be set only together with "Delete message": the form switches it off when "Delete message" is off, and the admin refuses the combination. For a new mailbox it is off (only the messages not fetched before, remembered by getmail in its `oldmail-*` files).
- **Test connection**: one login on the remote server, from the form (with the values being edited) and from every row of the list (with the saved configuration). Never automatic, and a single attempt per click: providers lock accounts after repeated failures.
- **Logs page**: the getmail log, the getmail errors and the cron log, with search, a line limit, auto-refresh and a status view (getmail version, folders, number of external mailboxes per schedule). **Clear** empties the getmail log or the getmail errors. Syslog and the mail log are offered only if the admin can read them; on a standard Poste.io it cannot (`root:adm`, `0640`), so they do not appear.
- **Getmail errors**: what getmail reports (configuration errors, failed logins, unreachable servers, crashes, warnings about the configuration) is written by `cron.sh` to its own log, each line with the time and the mailbox file it belongs to, and shown as **Getmail Errors** in the logs page. When there are lines from the last 24 hours, the default getmail view says so in a red line on top (how many, and the latest). A separate file because the admin cannot read `/data/log/mail.log` (`root:adm`, `0640`), where they also go, and because some of them (a configuration getmail cannot read, a crash) never reach getmail's own log.

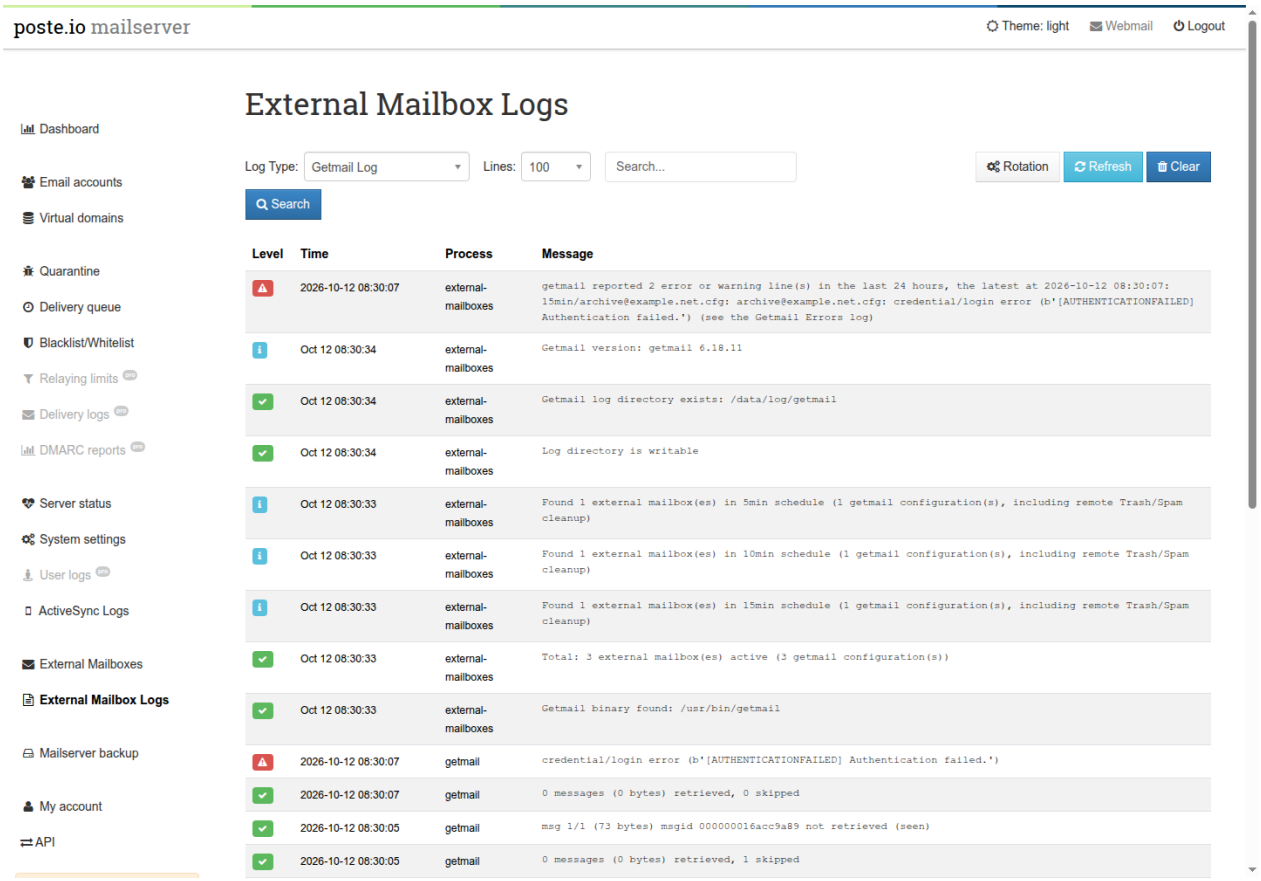
Copyright © 2026 Antonio Sanna

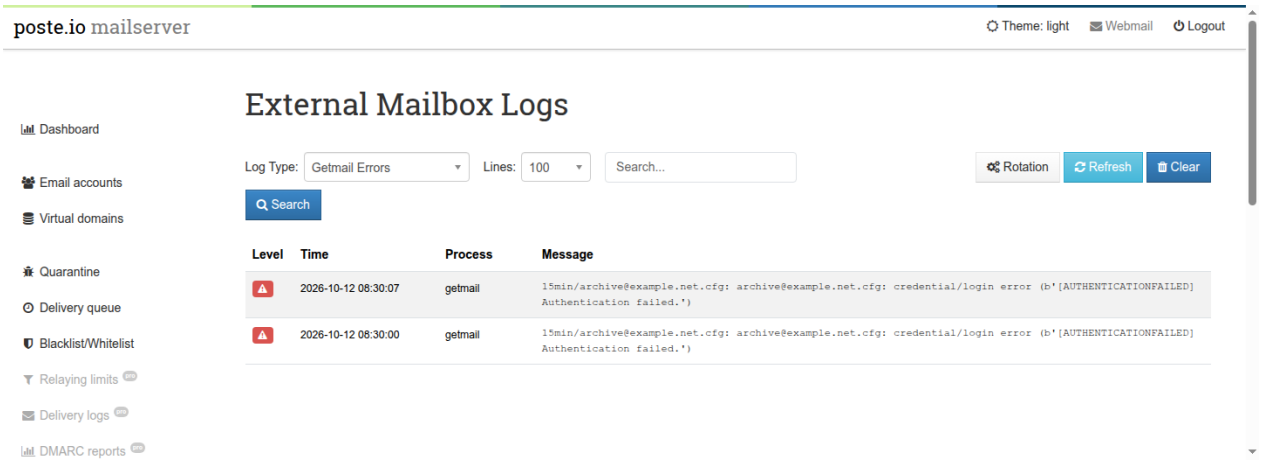
- **Log rotation**: `getmail.log` and `errors.log` are rotated by Poste.io's nightly logrotate, with settings chosen in the logs page (**Rotation** button): rotation on or off, how often the getmail log is rotated (day, week, month), the size above which the getmail errors are rotated (512 KB to 10 MB) and how many old copies are kept, compressed (1 to 12). Defaults: weekly, 1 MB, 4 copies. The errors are rotated by size and not by date because the red line of the getmail view reads only the current `errors.log`. See How it works below.
- **Spam check (rspamd)**: every fetched message is scanned by rspamd before delivery, by a getmail filter (`getmail-rspamd`, in the image). It gets the same `X-Rspamd-Score`, `X-Rspamd-Bar` and `X-Rspamd-Report` headers as mail received over SMTP (existing ones are replaced, so they cannot be forged), and from the score at which Poste.io marks spam (`munge_subject` in Haraka's `quarantine_decide.ini`, 6 by default) the `*****SPAM*****` subject prefix that the mailbox's sieve rule files into **Junk**. Above Poste.io's quarantine score the message goes to Junk too: nothing fetched is held back. Without the filter fetched mail is not scanned at all: getmail delivers through Haraka's local submission socket, where the rspamd plugin skips local connections (`local_ip=false`). The Bayes classifier learns into the bucket of the recipient's domain, like for SMTP mail. **Limits**: the message comes from a mailbox, not from an SMTP client, so the checks on the client's address and name do not apply (the one that would fire on every message, unknown hostname, is switched off), SPF cannot be checked, and DMARC relies on DKIM only (an unsigned message from a domain with a `reject` policy gets `DMARC_POLICY_REJECT`, +2). The remote provider checks SPF and DMARC with the real address when it receives the mail. **If rspamd does not answer** the message is delivered unchanged and the reason is written to the getmail errors (below): mail is never lost or delayed because of the filter.
- **Translations**: English and Czech, like the rest of the Poste.io admin (the browser chooses).



Copyright © 2026 Antonio Sanna

The logs page: the default getmail view, with the red line on top when there are recent errors, and the **Getmail Errors** view (example data).





How it works

The "database" is the file system, inside the `/data` volume:

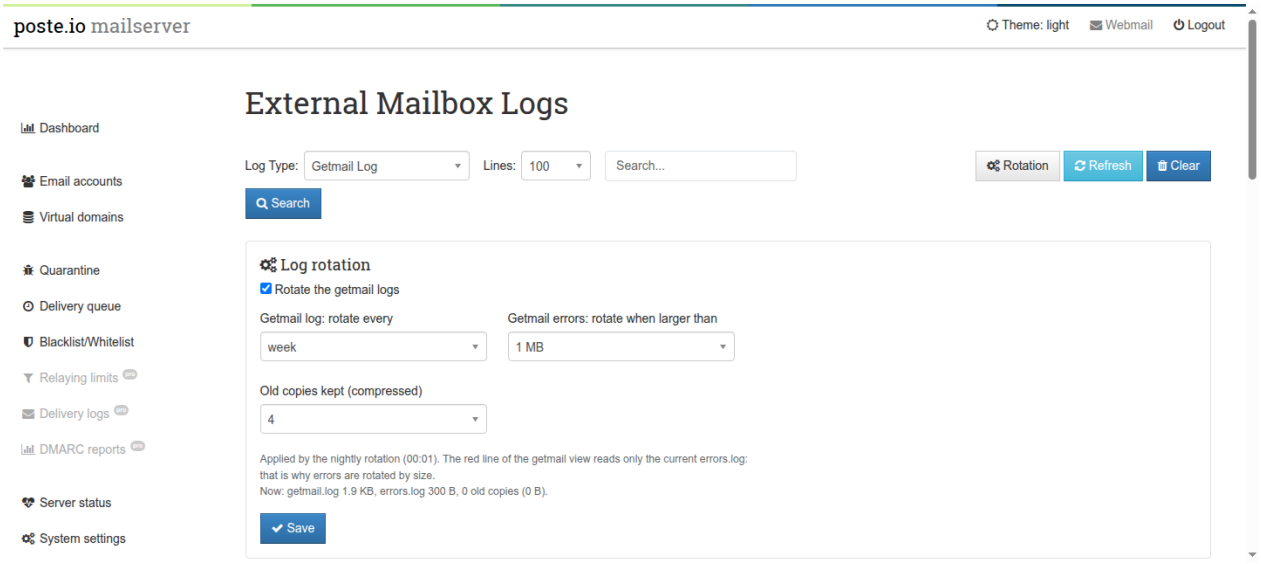
```
/data/var/getmail/
├── 5min/ 10min/ 15min/ 30min/ 60min/   one .cfg per mailbox; the folder is the schedule
├── disabled/                           mailboxes not scheduled
├── cron.sh                             the runner
└── <schedule>/oldmail-*                getmail's memory of already fetched messages
/data/log/getmail/getmail.log           operational log (written by getmail)
/data/log/getmail/errors.log            what getmail reports as errors (written by cron.sh)
```

A mailbox with remote cleanup has up to three files in its folder: `user@example.com.cfg` (download), `user@example.com.trash.cfg` and `user@example.com.spam.cfg` (fetch → discard to `/bin/true` → delete on the server). The cleanup files are not listed separately: they are created, updated and removed together with their mailbox.

Copyright © 2026 Antonio Sanna

The crontab of the `mail` user (`/etc/external-mailboxes.crontab`) runs `cron.sh <folder>` for each schedule; `cron.sh` runs getmail on each `.cfg` with `flock -n`, so a slow mailbox never overlaps with itself. getmail never runs as root. Each fetched message goes through the `getmail-rspamd` filter (the `[filter1]` section of the `.cfg`) and is then delivered with Poste.io's `sendmail`; what getmail prints as errors goes to `errors.log`.

Log rotation



The admin runs as `mail` and cannot write logrotate rules, which must belong to root. So the settings are saved in `/data/var/getmail/log-rotation.ini` (on the volume: they survive rebuilds and upgrades), and a root script, `/usr/local/bin/external-mailboxes-logrotate`, turns them into `/etc/logrotate.d/external-mailboxes` every night at 00:00 (`/etc/cron.d/external-mailboxes-logrotate`), one minute before Poste.io's logrotate. A change made in the admin applies from the next night. The image contains the rule with the default settings, written by the same script at build time.

The settings file is written by a less privileged user and read by root: the script executes nothing from it, and accepts only the values offered by the form (`enabled` 0/1, `getmail_every` daily/weekly/monthly, `errors_size` 512k/1M/5M/10M, `keep` 1/2/4/8/12); anything else means the default. The admin checks the same values before saving (and its own CSRF token).

```
/data/var/getmail/log-rotation.ini          settings (admin, mail, 0640)
/usr/local/bin/external-mailboxes-logrotate writes the rule (root)
/etc/cron.d/external-mailboxes-logrotate    runs it at 00:00
/etc/logrotate.d/external-mailboxes         the rule, used by logrotate at 00:01
```

Security

- **Files**: `/data/var/getmail` and `/data/log/getmail` belong to `mail:mail`, folders `0750`, `.cfg` and `oldmail-*` files `0600`. The admin (php-fpm runs as `mail`) writes them, cron reads them as `mail`.
- **Passwords**: getmail needs them in clear text in the `.cfg` files, so they are stored there (readable only by `mail`). They are **never sent back to the browser**: when editing, an empty password field means "keep the saved one", and the saved password is read and rewritten untouched.
- **Saved password and the test buttons**: with an empty password field, "Test connection" and "Load folders" use the saved password **only if protocol, server, port and username are exactly the saved ones**; otherwise they ask for it. Without this rule, changing the server in the form would be enough to send the saved password to any host.
- **TLS**: for the SSL protocols the generated configuration sets `ca_certs`, so getmail verifies the server certificate and its name (without it getmail uses `CERT_NONE`). "Test connection" and "Load folders" verify it too. The non-SSL protocols are plain text, as their name says.
- **CSRF**: Poste.io disables Symfony's form CSRF protection (`fos_rest.yaml`, `disable_csrf_role: PUBLIC_ACCESS`), so every form and AJAX call of the module carries its own explicit token, checked by the controller.
- **Validation**: protocol, schedule and retention must be values offered by the form; the email address becomes a file name and is checked against a strict pattern (no `/`, no `..`); the server name too; the destination must be an existing mailbox or redirect; values written into the `.cfg` file cannot contain line breaks (they would inject configuration lines).

Copyright © 2026 Antonio Sanna

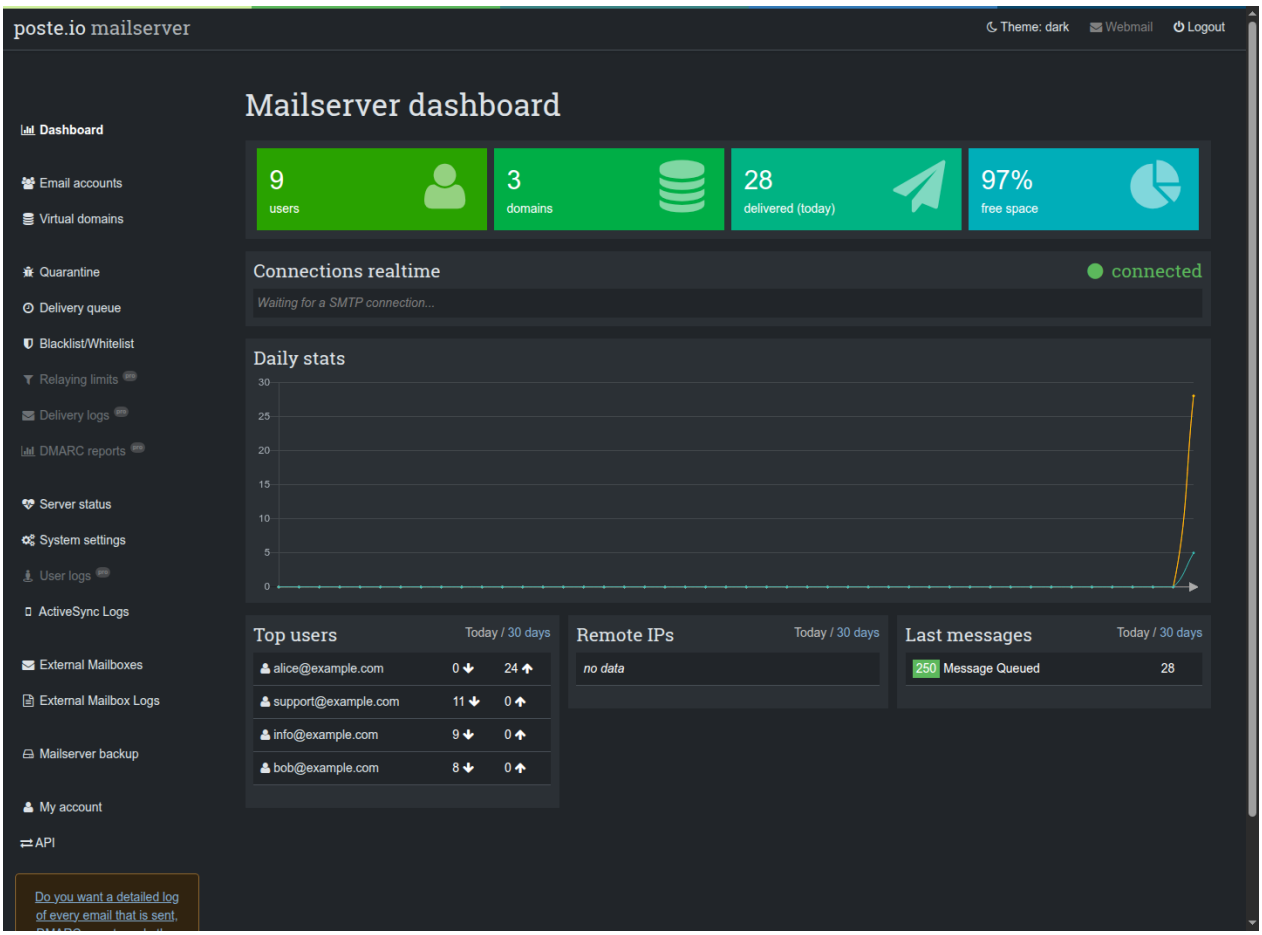
Notes and limitations

- **Gmail / Microsoft 365**: OAuth2 is not supported. Gmail accepts IMAP with an *app password* (2-step verification required) using the IMAP SSL protocol and `imap.gmail.com`; this was not tested by the author, use **Test connection** to check your account. Gmail folders are named like `[Gmail]/Spam`.
- **Multidrop** (one catch-all remote mailbox sorted by original recipient) is not offered: to send one external mailbox to several people, deliver it to a redirect.
- **POP3 test**: c-client accepts neither half-open nor read-only POP3 sessions, so the test opens `INBOX`; it closes without expunge, so no message is deleted.
- The status lines of the logs page are log output and stay in English.

Copyright © 2026 Antonio Sanna

Dark theme for the admin

A dark theme for the Poste.io administration pages (`/admin`), automatic and with a switch.



Behaviour

- **Automatic**: follows the operating system / browser setting (`prefers-color-scheme`), also when it changes while the page is open.
- **Switch**: the top bar gets a *Theme* item next to *Webmail* and *Logout*; each click goes **automatic → dark → light → automatic**. The choice is kept in the browser (`localStorage` , key `poste-admin-theme`); *automatic* removes it.
- **No white flash**: the theme is decided by a small script in `<head>` , before the page is painted.
- **Login page**: dark too when the system is dark (no switch there: the login page replaces the top bar).
- **Charts** (dashboard, server status): Chart.js draws with its own colours, so text, grid and the doughnut borders are adjusted, and the charts already on the page are redrawn when the theme changes.
- **Light theme untouched**: every rule is scoped to `[data-bs-theme=dark]` . This was checked by comparing the computed colours of a page with and without the stylesheet in light mode: no difference.

The webmail (Roundcube, Elastic skin) already has its own dark mode and is not touched.

How it works

The Poste.io admin is built on Bootstrap 5.3, whose dark colour mode only needs `data-bs-theme="dark"` on `<html>` : backgrounds, text, borders, tables, inputs and alerts follow. What remains are the colours Poste.io hard-codes for the light look, fixed in `dark-theme/admin/public/dark-theme.css` *(https://github.com/atsanna/poste.io-addons/blob/main/dark-theme/admin/public/dark-theme.css)*:

- the top bar and the left menu, the "pretty" button gradients, pagination;
- Chosen drop-down menus, help boxes, the dashboard section boxes and spinner;
- black text written inline in a few templates (`style="color: black"`).

Copyright © 2026 Antonio Sanna

Files:

File	Role
`dark-theme/admin/templates/Base/DarkTheme/_head.html.twig`	stylesheet + theme script, included before `</head>`
`dark-theme/admin/templates/Base/DarkTheme/_toggle.html.twig`	the *Theme* item in the top bar
`dark-theme/admin/public/dark-theme.css`	the dark-only corrections
`dark-theme/dark-theme.{en,cs}.yml`	translations of the switch
`dark-theme/dark-theme-install.sh`	adds the two `include`s to Poste.io's `base.html.twig` and the translations

`dark-theme-install.sh` is the only part that touches a Poste.io file, with two `include` lines. It refuses to run if it does not find its hook points exactly once, so a Poste.io update that moves them breaks the build instead of silently losing the theme.

Found a page that stays light?

Poste.io sometimes writes colours directly in a template. Open an issue with a screenshot and the page URL; the fix is usually one rule in `dark-theme.css`.

Copyright © 2026 Antonio Sanna

Calendars and contacts over ActiveSync

Poste.io ships two things that, up to 2.5.23, never met:

- **Z-Push**, the ActiveSync server phones use for the "Exchange" account type, configured for **mail only**: the phone's calendar and contacts switches are on, but nothing ever arrives;
- a **CalDAV/CardDAV server** (SabreDAV, at `https://<your host>/dav/`) where each mailbox can have calendars and address books, created from the admin.

With one ActiveSync account the phone gets the mail **and** the calendars and contacts of the DAV server: the same data a CalDAV/CardDAV client sees, both ways. **Since 2.5.24 Poste.io connects them itself**, with the stock CalDAV and CardDAV backends of Z-Push. This add-on (Poste.io 2.5.24 or later) keeps that and replaces only those two backends with corrected ones: an event refused by the server is not reported to the phone as saved, and the DAV server switched off does not take ActiveSync down (see the issues worked around, below).

What users get

The mailbox has (in the admin)	The phone shows
nothing	mail + the "Calendar" and "Contacts" Poste.io creates for it (see below)
a calendar	mail + that calendar (as the **default** calendar) + its tasks
an address book	mail + contacts
a calendar shared by a colleague	that calendar as an **additional** calendar

Everything goes both ways: an event or a contact created on the phone is stored on the server, and changes on the server reach the phone.

Enabling calendars and contacts for a mailbox

Calendars and address books are created per mailbox from the admin. Since 2.5.24, while DAV is enabled, Poste.io's DAV server also gives a mailbox that has none an empty "Calendar" and "Contacts" at its first DAV access (the webmail calendar, ActiveSync and invitations need somewhere to put things): the add-on cannot and does not prevent it.

1. **Settings → DAV (Experimental) → Enabled** must be ticked (it enables the DAV server itself).
2. **Mailboxes →** the mailbox **→ DAV** section: *Create calendar* and/or *Create address book*.
3. On the phone, in the ActiveSync account settings, calendar and contacts synchronisation must be on; then sync (or wait for the next sync).
4. After switching DAV on or off, **restart the container**: since 2.5.24 Poste.io chooses the ActiveSync backend (mail only, or mail + calendars and contacts) when the container starts.

A mailbox without a calendar or an address book keeps mail only on the phone. When a calendar or an address book is created or deleted in the admin, the phone adds or removes the folder at its next folder sync.

Seeing whether a colleague is busy: read-only sharing

In the DAV section of the colleague's mailbox, open the calendar's *share* page, pick the mailbox to share with and choose **read only**. On that user's phone the colleague's calendar appears as an additional calendar, with the events and their titles (the DAV server has no "free/busy only" level).

Writing in a read-only calendar from the phone is refused: a new event gets an error, a change gets a conflict (the server version wins), a deletion is not applied on the server and the event comes back at the next sync. Read-write sharing works too.

The default calendar

ActiveSync has one default calendar per account. Poste.io names calendars with random identifiers, so the add-on does not guess: it asks the server through `schedule-default-calendar-URL` (RFC 6638), the same property CalDAV clients use for invitations. SabreDAV answers with the **first calendar owned by the user**; calendars shared by colleagues are never the default.

Copyright © 2026 Antonio Sanna

Setting up the phone

Add an **Exchange / ActiveSync** account with the mailbox address and password and your mail server's host name, then make sure calendar and contacts synchronisation are enabled for that account. Tested with the Gmail app on Android; other ActiveSync clients should behave the same but were not tested.

When the DAV server is off or broken

Z-Push's combined backend refuses the whole login if any of its backends refuses it, so without care a DAV server that does not answer would take ActiveSync down for every user, **mail included**. The add-on prevents that:

- if, at login, the DAV server does not answer as it should (for example DAV disabled in the settings), the calendar and contacts backends accept the login **without folders**;
- the phone keeps syncing mail (new mail is delivered in the same reply), and removes calendars and contacts at its next folder sync;
- when the DAV server is back, calendars and contacts come back by themselves.

The Z-Push log then shows `the DAV server is not answering ... mail goes on`.

Installing on an existing installation: devices must resync

Build and run the image as described in Installation. The add-on changes Z-Push's configuration, and with it the identifiers of the mail folders; a phone that was already syncing would receive the new folders **without the old ones being removed** (duplicated mail folders). Right after the first start with the add-on, ask every device for a full resync:

```
# every user/device pair known to Z-Push
docker exec <container> sh -c 'cd /opt/z-push && runuser -u www-data -- php z-push-admin.php -a list'

# one pair (device ids are listed without dashes)
docker exec <container> runuser -u www-data -- php /opt/z-push/z-push-admin.php -a resync -u <user> -d <device>

# or all pairs at once
docker exec <container> sh -c 'cd /opt/z-push && runuser -u www-data -- php z-push-admin.php -a list | awk "NF == 2
&& \$1 != \"Device\" && \$1 !~ /^-/" | while read -r d us; do for u in $(echo "$us" | tr , " "); do runuser -u www-
data -- php z-push-admin.php -a resync -u "$u" -d "$d"; done; done'
```

Each phone then starts again from an empty folder list at its next sync (the mail app may need to be opened). Messages stay on the server; the phone only downloads them again.

Rebuilding the image or upgrading Poste.io with the add-on inside does not need a resync: the configuration stays the same and Z-Push's state lives on the `/data` volume (`/data/zpush`).

Removing the add-on

Remove the *ActiveSync: calendars and contacts* block from the Dockerfile, rebuild, recreate the container, then resync every device as above (the folder identifiers change back).

Upgrading Poste.io

`activesync-dav-install.sh` checks every line it changes in Z-Push's configuration and **stops the build** if one of them is not exactly as expected. A Poste.io release that changes Z-Push therefore breaks the build loudly instead of producing a half-configured image.

Limitations

- **Complex recurrences are simplified on the phone** (a Z-Push limit): for example "monthly on the 2nd and 3rd" arrives as "monthly on the 2nd". The event on the server is untouched.
- **No free/busy lookup when inviting**: Z-Push implements it for Kopano only, not for CalDAV. Use read-only sharing to see a colleague's calendar.
- **No shared or global address book**: Poste.io's DAV server shares calendars, not address books.
- **All of a user's address books appear as one contacts folder** (Z-Push's CardDAV backend); new contacts from the phone go to the first address book.
- **Tasks**: Z-Push shows every calendar twice, as events and as tasks.

Copyright © 2026 Antonio Sanna

How it works

Configuration changed by `activesync-dav/activesync-dav-install.sh` (3 lines in 2 files; each one, and the settings of Poste.io the backends rely on, is checked before anything is written):

File (in `/opt/z-push`)	Change
`config.php`	none: Poste.io sets `BACKEND_PROVIDER` at every start (`BackendCombined` while DAV is enabled)
`backend/combined/config.php`	the calendar backend `c` → `BackendPosteCalDAV`, the contacts backend `d` → `BackendPosteCardDAV` (the folder mapping of Poste.io is kept)
`backend/caldav/config.php`	none: server, port and path are the ones Poste.io writes
`backend/carddav/config.php`	default path `/dav/addressbooks/%u/` instead of `…/default/`: new contacts go to the user's first address book, no "default" one is created

Files installed in `/opt/z-push/backend/`:

File	Role
`postecaldav/postecaldav.php`	`BackendPosteCalDAV`, extends Z-Push's `BackendCalDAV`
`postecarddav/postecarddav.php`	`BackendPosteCardDAV`, extends Z-Push's `BackendCardDAV`
`postecaldav/postedav.php`	PROPFIND helper shared by both

Both backends talk to the DAV server inside the container, at the address Poste.io configures, with the credentials the phone logged in with (HTTP Basic, the mailbox password). Each login starts with one PROPFIND that is also the health check of the DAV server. The Debian package `libawl-php` (needed by Z-Push's CalDAV client) is part of Poste.io since 2.5.24; the Dockerfile still installs it, harmless, so the add-on does not depend on it; its directory is appended to PHP's include path by the backend only.

Z-Push issues worked around

Found while testing Z-Push 2.7.5 against Poste.io's DAV server, and fixed in the two backends without modifying Z-Push's own files:

1. **An event refused by the server was confirmed to the phone as saved.** `BackendCalDAV` never checks the result of the PUT, so an event written in a read-only shared calendar stayed on the phone while nobody else saw it. Now a refused write or deletion is an error for the device.
2. **A contact added by a user without an address book crashed Z-Push** (out of memory, HTTP 500 for the whole sync, mail included, retried by the phone forever). Now a user without an address book has no contacts folder, and a contact sent anyway gets an error.
3. **The DAV server switched off took ActiveSync down**, see above.

Troubleshooting

- Z-Push logs: `/var/log/z-push/z-push.log` and `z-push-error.log` in the container. The add-on's lines start with `BackendPosteCalDAV`, `BackendPosteCardDAV` or `PosteDAV`.
- What a device has: `z-push-admin.php -a list -u <user> -d <device>` (run as `www-data`, see above) shows the folders and the synchronised data (`Emails`, `Calendars`, `Tasks`, `Contacts`).
- Calendars missing for everybody: check that DAV is enabled in the settings; with DAV disabled `/dav/` answers *Enable DAV at your server settings*.
- Calendars missing for one user: that mailbox has no calendar in the admin, or the phone has calendar sync switched off for the account.

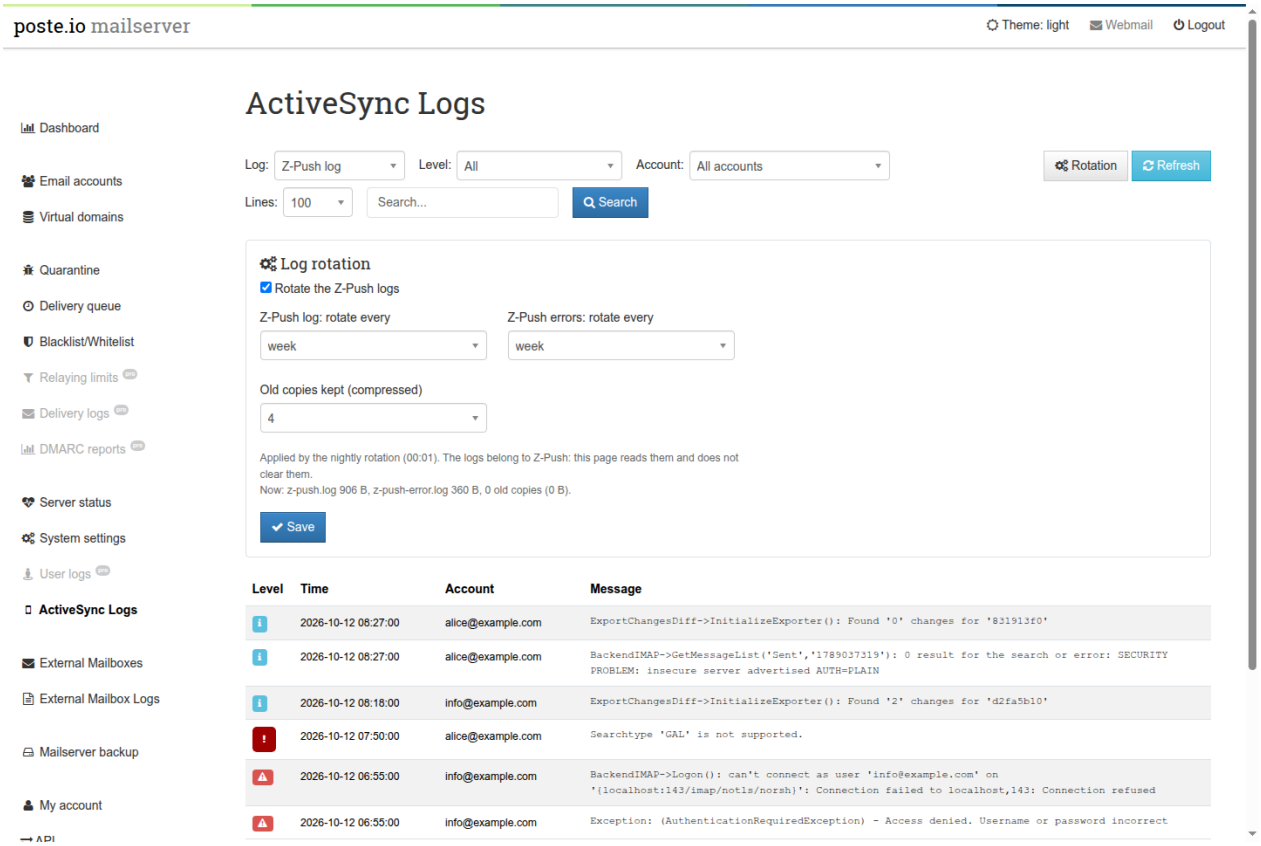
Tested

Poste.io 2.5.24 with Z-Push 2.7.5, Pro (in production) and open source (built and tested in a container). In a test container built from the Dockerfile: folder types and default calendar, server ↔ phone for events and contacts, read-only sharing, users with nothing enabled, an address book removed while the phone still had the folder, DAV switched off and on again, and the migration of an already-synchronised phone with and without the resync.

Copyright © 2026 Antonio Sanna

ActiveSync logs in the admin

A page of the Poste.io admin, **ActiveSync Logs**, to read the logs of Z-Push (the ActiveSync server phones talk to) and to set their rotation. Without it these logs are only inside the container, and Poste.io never rotates them: on a server with a couple of phones `z-push.log` grows by a few megabytes a month, forever.



The add-on is independent of the others: Z-Push and its logs are part of every Poste.io, with or without calendars and contacts over ActiveSync.

The page

The entry is in the left menu right after **User logs**, for super admins only.

- **Log**: `z-push.log` (everything Z-Push writes at its log level, INFO in Poste.io) or `z-push-error.log` (only its warnings and errors).
- **Level**: all lines, warnings and errors, or errors only (Z-Push's `ERROR` and `FATAL`).
- **Account**: the accounts found in the last 20,000 lines of the log; pick one to see only its lines.
- **Lines** and **Search** (plain text, case-insensitive).

The filters apply to the whole file, then the last 25–500 matching lines are shown, newest first. Each line shows Z-Push's level (warnings yellow, errors red), the time, the account and the message; a message that Z-Push writes over two lines is shown as one. The page refreshes itself every 30 seconds.

What you typically find there: a phone with a wrong password (`AuthenticationRequiredException`), a phone asking for something the server does not offer (`Searchtype 'GAL' is not supported`), IMAP refusing Z-Push's connection (mail server down), and at every sync the INFO line *"SECURITY PROBLEM: insecure server advertised AUTH=PLAIN"*: Z-Push talks to Poste.io's IMAP on `localhost` without TLS, which is noise, not a risk from outside.

There is **no Clear button**, on purpose. The logs belong to Z-Push (user `www-data`, mode `0644`), and Poste.io gives them back to `www-data` at every start; the admin runs as `mail` and can read them, not change them. Making them writable by the admin would mean changing permissions that Poste.io sets. The rotation keeps them small instead.

Copyright © 2026 Antonio Sanna

Log rotation

Rotation opens the settings: on/off, how often each log is rotated (every day, week or month), how many old copies are kept, compressed (1–12). The defaults are weekly for both and 4 copies. A line under the settings shows the current size of the two logs and of the old copies.

It works like the log rotation of External Mailboxes. The admin runs as `mail` and cannot write logrotate rules, which must belong to root, so:

1. the admin saves the settings in `/data/var/activesync-logs/log-rotation.ini` (on the volume: they survive rebuilds and upgrades), with its own CSRF token and only the values offered by the form;
2. a root script, `/usr/local/bin/activesync-logs-logrotate`, turns them into `/etc/logrotate.d/activesync-logs` every night at 00:00, one minute before Poste.io's logrotate, and at every container start. A change made in the admin applies from the next night.

The settings file is written by a less privileged user and read by root: the script executes nothing from it and accepts only the allowed values (`enabled` 0/1, `zpush_every` and `errors_every` daily/weekly/monthly, `keep` 1/2/4/8/12); anything else means the default.

The rule rotates the two logs as `www-data` (`su www-data www-data`) and recreates them `0644` owned by `www-data`, as Poste.io has them. No `copytruncate` is needed: Z-Push opens the file again for every line it writes, so it continues in the new file at once.

```
/data/var/activesync-logs/log-rotation.ini   settings (admin, mail, 0640)
/usr/local/bin/activesync-logs-logrotate     writes the rule (root)
/etc/cron.d/activesync-logs-logrotate        runs it at 00:00
/etc/cont-init.d/90-activesync-logs.sh       at every start: the settings folder, then the rule
/etc/logrotate.d/activesync-logs             the rule, used by logrotate at 00:01
```

Existing installations

Nothing to do by hand. `/data` is a volume, so a folder prepared in the image would be hidden on an installation that already has one; the settings folder is therefore created at every container start by `/etc/cont-init.d/90-activesync-logs.sh` (a start script run as root by the image's init, after Poste.io's own), which also writes the rule: from the first night the logs are rotated with the default settings, until they are changed in the page.

Files

File	Role
`activesync-logs/admin/src/Base/Controller/ActiveSyncLogsController.php`	the page and the saving of the settings
`activesync-logs/admin/templates/Base/ActiveSyncLogs/`	the page and its table (the table is reloaded every 30 seconds)
`activesync-logs/activesync-logs.{en,cs}.yml`	translations
`activesync-logs/activesync-logs-logrotate` + `.cron`	the root script and its nightly run
`activesync-logs/90-activesync-logs.sh`	start script: settings folder and rule
`activesync-logs/activesync-logs-install.sh`	installs everything above and adds the menu entry

`activesync-logs-install.sh` is the only part that touches a Poste.io file: it adds three lines to the menu template right after **User logs**. It refuses to run if it does not find that entry exactly once, followed by its `</li>`, so a Poste.io update that changes the menu breaks the build instead of leaving the page without its entry.

Removing it

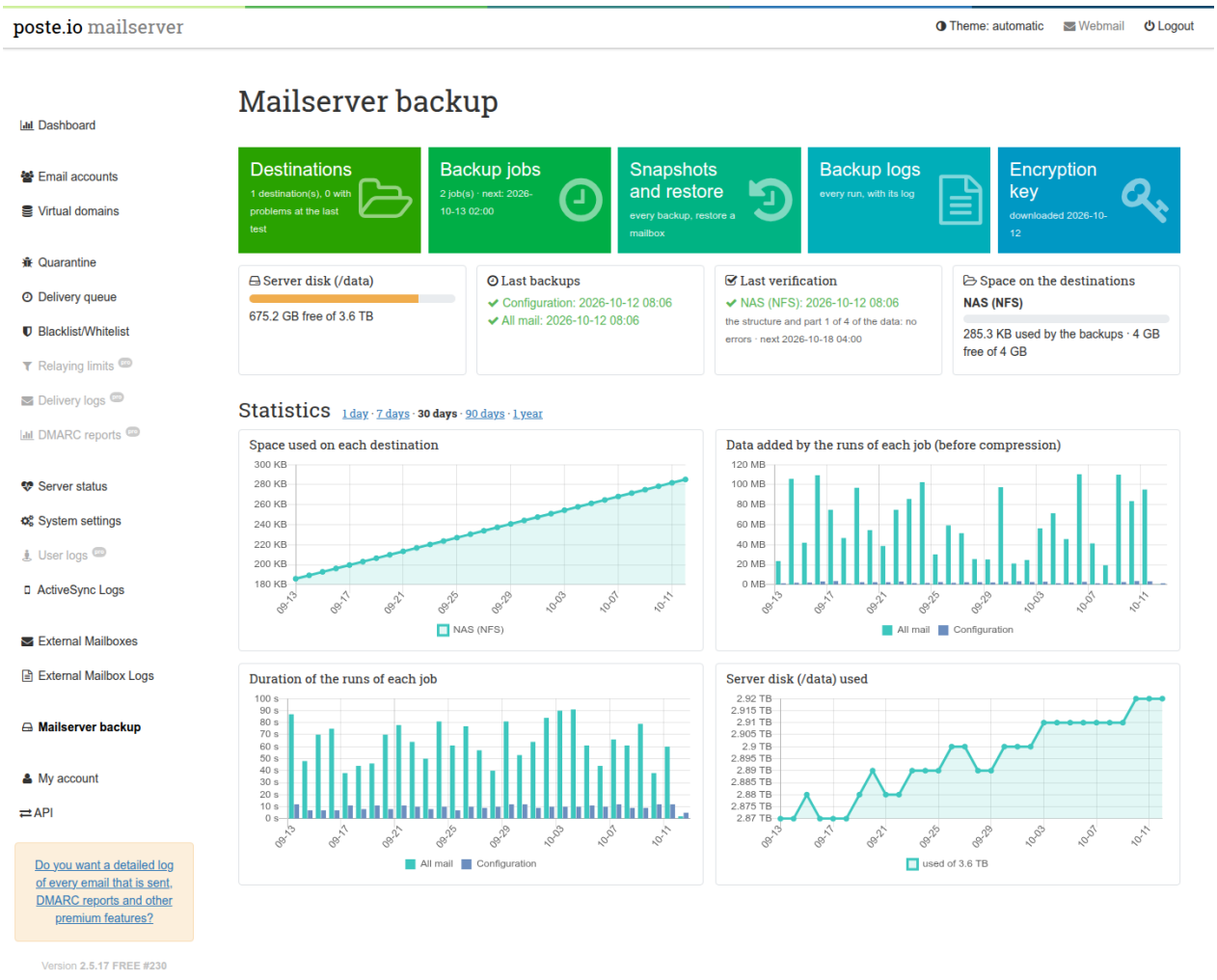
Build without the section. What stays in the volume is `/data/var/activesync-logs` (the settings) and, in `/data/log/z-push`, the old copies made by the rotation (`*.log.1`, `*.log.N.gz`); remove them if you no longer need them. The Z-Push logs themselves are Poste.io's and stay where they were.

Copyright © 2026 Antonio Sanna

Backup and restore

Encrypted, scheduled backups of a Poste.io server, made and checked from the admin: the server configuration, all the mail or only some mailboxes, sent to one or more destinations at the same time (SMB/Samba, SFTP, S3, or a volume mounted by the host such as NFS).

> ***Status of this add-on.*** *This page describes what is in the repository now. The add-on is being built in stages; what a stage does not do yet is said where it matters (for example notifications, which come next).*



The add-on is independent of the others and of anything outside the container: it does not need a backup system on the host. If the host already backs up the whole machine (a hypervisor snapshot, a VM backup), this add-on adds what those cannot give: backups of the mail server alone, off the machine, encrypted, frequent, with single mailboxes to restore.

- What is backed up
- How it works
- The encryption key
- Destinations
- Backup jobs
- Watching the backups
- Restoring
- Reading the backups without the server
- Rebuilding the whole server
- Good practice
- A backup of a server that is running

Copyright © 2026 Antonio Sanna

- Security
- If the mail server is compromised
- Files and folders
- Troubleshooting
- Limits
- Removing it

What is backed up

A job backs up the **server configuration**, the **mail**, or both.

Configuration (a few megabytes):

What	From
accounts, domains, aliases, passwords (hashed), limits	`/data/users.db`, copied with SQLite's own backup command while in use, then checked with an integrity check
calendars and address books (DAV)	`/data/dav.db`, same way
webmail settings, identities, address books	`/data/roundcube/roundcube.db` (it belongs to the webmail: the database and its write-ahead log are copied, again if they changed meanwhile, and the copy is checked)
server settings	`/data/server.ini`, `/data/instance-id`
TLS certificates (also Let's Encrypt)	`/data/ssl`
DKIM keys of each domain	`/data/domains/<domain>/private`, `public`, `selector`
spam filter training (rspamd's Bayes, in redis)	`/data/redis/dump.rdb`
ActiveSync device states	`/data/zpush`
settings of the add-ons and of Poste.io under `/data/var`	everything there except the virus database (downloaded again) and this add-on's own folder
the list of the messages of the backups (see Restoring)	`/data/var/backup/engine/index.db`, copied like the databases

Mail: either all of it (`/data/domains`, every mailbox with its folders and its filters, plus the messages held in quarantine) or only the mailboxes chosen in the job (their whole home folder: mail and filters).

Never backed up: the encryption key and the add-on's settings (they hold the passwords of the destinations: see Security), logs, the outgoing mail queue, the virus database, the cached rule maps of the spam filter (downloaded again), and the webmail's PGP keyrings (`roundcube/enigma`: readable only by the webmail; users who use PGP in Roundcube export their keys from there). More generally, a file of the configuration that the user `mail` cannot read is left out on purpose, and each run's log says how many.

How it works

Two parts:

- the **admin** saves settings and requests;
- the **engine** (`/usr/local/bin/poste-backup`, run by cron every minute) does the backups. It reads the settings and **validates them again** before using anything in them.

Both run as the user `mail`, the user Poste.io's admin already runs as and that owns the mail. The backup never runs as root: whatever the admin's settings say, the engine cannot do more than the admin itself could (see Security).

The backups are made by restic *(https://restic.net)*: encrypted, with deduplication (a file already stored is never stored again, so an hourly backup of a large server stores only what changed), checked by content. Remote destinations are reached through rclone *(https://rclone.org)*. Both come from the Debian packages of the image.

Each **destination** holds its own restic **repository**, and every job writes a full, independent copy to each of its destinations: losing one destination loses nothing. One destination can serve several jobs.

Each run of a job:

1. with mail: brings the list of the messages up to date (see The list of the messages); then copies the databases in a consistent state and checks the copies;

Copyright © 2026 Antonio Sanna

2. for each destination: checks it (a local folder must be a mounted volume, not on the disk of `/data`), creates the repository there **only if the folder is empty** (a folder with other files is never written to), backs up, then removes the old snapshots of that job according to its keep rules;
3. records the result for the admin (state and history).

A destination that fails does not stop the others. One run at a time: a long backup makes the next minutes wait (a lock), it is never started twice. After the jobs due, the engine runs the verifications due.

The result of a run on a destination is **ok**, **warning** or **error**:

- *ok*: the snapshot is complete. A message moved or deleted while the backup was reading the mailboxes (a message moved to another folder, a flag changed: Dovecot gives the file a new name) is not a problem: the run says how many with a note, and the next backup has them where they are now.
- *warning*: the snapshot was made, but a file could not be read (for example a permission problem), a mailbox chosen in the job no longer exists, or the old snapshots could not be removed. The run says what; look at its log.
- *error*: no snapshot on that destination.

The encryption key

Every backup is encrypted with a key made at the first start of the container (`/data/var/backup/settings/restic.key`, 48 random bytes). **Without the key no backup can be read**, not by you and not by anyone who gets hold of the destination.

The key is not part of any backup (a backup protected by a key stored inside it would be useless). **Download it** from *Mailserver backup* > *Encryption key* and keep it outside the server, where you keep your other passwords. The admin shows a warning until it has been downloaded once.

The key is never replaced: if the file is deleted, a new one is made at the next start, and that new key cannot read the backups made before. Put the old key back instead (same file, owner `mail`, mode `0600`).

Destinations

Mailserver backup > *Destinations* > *Add a destination*. The **Test** button next to *Save* tries the values of the form **before** they are saved: it checks that the destination can be reached, that the folder is empty or already a repository, the free space, and writes and removes a small file (a REST server reports no free space and is not written to: there the Test checks the certificate, the user and the repository). The page is not reloaded, so a password typed stays in its field. A saved destination can be tested again from its page. Use **one folder per mail server**.

Give the backups an account of their own on the destination, with access to that folder only (see Good practice).

SMB / Samba

Server, share, folder inside the share, user, password, and the domain if the server needs one. Works with Samba, Windows shares and NAS shares (SMB 2 or newer). The free and total space of the share are read with `smbclient`: the SMB client inside rclone 1.60 reports them divided by a factor that depends on the server (half, on Samba and on a QNAP NAS), so its numbers are not used.

SFTP

Server, port, user, folder, and a login **with a password** or **with a key**:

- *with a key*: nothing is pasted in the form. When the destination is saved, the admin makes a key pair (ed25519) for it, keeps the private key and shows the public key, with a ready command to run on the server as that user (it creates `~/.ssh` with mode 700 and adds the key to `~/.ssh/authorized_keys` with mode 600) and a button to copy it. On a NAS the user needs a home folder and the SFTP service on, or the key is not read. Then test the destination;
- **the server's own SSH key must be confirmed** before anything is sent. The first test shows the fingerprints the server offers; compare them with the server's (`ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub` on the server) and confirm. From then on a different key stops the backups instead of sending them to whoever answers. Changing the server or the port asks for a new confirmation.

The folder is required (the repository must not land among the user's own files).

S3

Endpoint (`https://...`), region (if the provider needs it), bucket, folder inside the bucket, access key and secret key. Works with Amazon S3 and with S3-compatible services (MinIO, Backblaze B2, Wasabi, Ceph, ...). If the bucket does not exist it is created, when the key is allowed to. S3 does not report free space.

Copyright © 2026 Antonio Sanna

A volume mounted by the host (NFS, a second disk, ...)

A container cannot mount network file systems itself (it would need privileges that make it far less safe), so the host mounts the share and passes it to the container as a volume, under `/backup`, `/mnt`, `/srv` or `/media`. For example with Docker Compose and an NFS export:

```
services:
  mailserver:
    # ...
    volumes:
      - ./data:/data
      - nfs-backup:/backup/nfs

volumes:
  nfs-backup:
    driver: local
    driver_opts:
      type: nfs
      o: "addr=nas.example.lan,rw,nfsvers=4"
      device: ":/export/mail-backup"
```

Then add a destination of type *Folder* with the path `/backup/nfs`. The engine refuses a folder that is **not a mount point** (the backup would stay inside the container and vanish with it), that is **on the same disk as `/data`** (it would not survive the loss of that disk), or that it **cannot write**: the backups are written by the user `mail`, **uid 8, gid 8**. Give the folder of the volume to that user (for an NFS export: a directory owned by 8:8 on the server, and an export that does not squash that user).

REST server, append-only (`rest-server`)

restic's own server, `rest-server` *(https://github.com/restic/rest-server)*, started with `--append-only`, **refuses to delete**: this mail server can only add backups there, and someone who takes it over cannot delete or overwrite the backups there either (see If the mail server is compromised). The old backups are then removed **on the server's side**, by the server's own schedule and with the key. HTTPS only.

On the NAS (or another machine), with Docker (QNAP Container Station, Synology Container Manager, ...): a folder for the data, a certificate whose name is the one the mail server will use (a host name, or an IP address), and a user:

```
mkdir -p /share/backup/rest && cd /share/backup/rest
# a certificate for the name used by the mail server (here nas.example.lan; for an
# address use subjectAltName=IP:192.0.2.10), valid ten years
openssl req -x509 -newkey ed25519 -nodes -keyout key.pem -out cert.pem -days 3650 \
    -subj "/CN=nas.example.lan" -addext "subjectAltName=DNS:nas.example.lan"
openssl x509 -in cert.pem -noout -fingerprint -sha256      # compare it in the admin
docker run -d --name rest-server --restart unless-stopped -p 8000:8000 \
    -v /share/backup/rest:/data -e OPTIONS="--append-only --tls --tls-cert /data/cert.pem --tls-key /data/key.pem" \
    restic/rest-server:0.12.1
docker exec -it rest-server create_user mailserver          # asks for the password
```

In the admin: *Add a destination*, type **REST server**, the server's name, port `8000`, a repository name (for example `mailserver`), the user and the password, **Accepts only new backups (append-only)** ticked. The first **Test** shows the certificate the server presents: compare its fingerprint with the one printed above, then confirm it. From then on only that certificate is accepted, also by the admin's own checks (another one is refused, as for the SSH keys of SFTP, even one signed by a public authority), and the user and password never appear on a command line (they go to restic in its environment). An expired certificate is refused.

A new certificate (renewed, or a new server under the same name): the backups to that destination stop with "the server's certificate changed". The **Test** on the destination's page then shows the new certificate with its fingerprint, to be compared and confirmed like the first one; nothing is sent to the server before. The address may be a name or an IP address (IPv6 too); the certificate must name it.

On an append-only destination this server never removes snapshots: the keep rules of the jobs are not applied there, and *Remove these backups* is not offered (and refused by the engine). Remove them on the server's side, for example once a week, with the same rules as the job and **the key** (the last line of the file downloaded from the admin), kept **outside** the folder the REST server serves:

```
tail -n 1 poste-backup-key.txt > /share/backup/restic.key && chmod 600 /share/backup/restic.key
docker run --rm -v /share/backup/rest:/data -v /share/backup/restic.key:/key:ro -e RESTIC_PASSWORD_FILE=/key \
    restic/restic:0.14.0 -r /data/mailserver forget --host poste --group-by host,tags \
    --keep-daily 7 --keep-weekly 4 --keep-monthly 6 --prune
```

Keep that key file readable only by the administrators of the NAS. Free space is not reported by a REST server (as with S3). Without `--append-only` on the server, untick the box: this server then removes the old snapshots there as on any other destination.

Copyright © 2026 Antonio Sanna

Verification

A backup that cannot be read is found only when it is needed, unless it is read before. Each destination is **verified** on a schedule of its own (on its page, *Verification*): restic checks the structure of the repository and **reads again a part of the stored data**, a different part each time, so that everything is read once in a number of verifications. By default: **every Sunday at 04:00, a quarter of the data** (all of it read in about four weeks). It can be every day, week or month, or never, and from all the data each time to 1/52 of it.

A damaged or missing file on the destination is found that way: the check of the structure alone does not read the data, the reading does (the tests change one byte of a data file on a destination and expect the verification to find it). The result shows on the dashboard (*Last verification*, and among the problems when it found errors), on the destinations page and on the destination's page with the last verifications; *Verify now* runs one at the next minute. The first scheduled verification of a new destination is at its next slot, not at once. While a verification runs no backup runs: one that is due starts right after (restic locks the repository for the check); a verification stops after 6 hours at most. A part where errors were found is read again at the next verification, until it is clean: the error stays on the dashboard until then, it does not disappear because another part was fine. A verification that could not run to its end (the destination not reachable, for example) is shown as failed and tried again an hour later.

Backup jobs

Mailserver backup > Backup jobs > Add a backup job:

- **What**: the server configuration, and no mail / all mail / only some mailboxes;
- **When**: every hour (at a minute), every day, every week (a day) or every month (day 1 to 28) at a time, in the **time zone of the schedules** (see below);
- **Destinations**: one or more, each gets a full copy;
- **Keep**: how many hourly, daily, weekly and monthly snapshots to keep. After each run the older snapshots of **that job** are removed: for each hour, day, week and month the last snapshot is kept, up to these numbers (restic's `forget --keep-*`, then `--prune`). At least one number must be above 0. Example: 24 hourly, 7 daily, 4 weekly, 12 monthly = every hour for a day, every day for a week, every week for a month, every month for a year.

Time zone of the schedules, chosen under the list of jobs: the hours of the jobs and the times shown in the backup pages are in that zone. Without a choice it is the container's time zone (its `TZ`, otherwise `/etc/timezone`), which in Poste.io's image is UTC unless `TZ` is set. The choice changes only the backups: the server and its logs keep their own time (usually UTC, which is a good thing for logs: no hour missing or repeated when the clocks change). The backup's own logs write each time with its offset (`+02:00`), so they can be compared with the others. After a change of zone, a job whose time has already passed today in the new zone may run once, at the next minute.

A job runs when its time comes; **Run now** (or *Save and run now*) starts it within a minute. A run missed because the server was off runs once, as soon as the server is back (not once per missed hour). A new job does not run for a time that was already past when it was created.

The page shows, for each job, the last run on each destination: how much new data it stored, how long it took, or the error, with a link to its log.

Deleting a job (or a destination) removes it from the settings only: the backups it made stay on the destinations, encrypted, and can still be restored. They are no longer thinned out by any keep rule: remove them from *Snapshots and restore* when they are not needed any more (see Removing the backups of a job).

Watching the backups

- **Dashboard**: the problems found (a failed run, a key never downloaded, the engine not running), the job running now with its **progress** (percentage, data and files done of the total, then removing old snapshots and measuring the space), the disk of the server, the last backups and the **space on each destination**: what the backups use there and, where the server reports it, the free space (S3 does not). The space is measured after each backup. The bars of the disk and of the destinations turn **yellow** from 80% used and **red** from 90%: *Space warnings*, on the destinations page, sets these two limits (they are also the limits of the warnings of the next stage).
- **Last verification** of each destination, and when the next one is (see Verification).
- **Statistics**, for the last day (one value per hour), or 7 days, 30 days, 90 days or a year (one value per day): the space used on each destination and on the server's disk (the last measure of the day), the data added by the runs of each job (a run writes the same data to each of its destinations: it counts once) and how long they took. They are drawn from the history of the runs (kept 400 days by default, see below); the space values are there from this version on.
- The dashboard, the jobs and the logs **update themselves**: every few seconds while a job runs, every 15 seconds otherwise; the page reloads when a job starts or ends.

Copyright © 2026 Antonio Sanna

- **Logs**: one line per run and destination, the most recent first, with a link to the log of the run (what the engine did, in English, with the output of restic). Under the list, how long they are kept: the logs of the runs for 90 days and the history (one line per run, for the statistics) for 400 days by default, from 1 to 3650 days. Older ones are removed after each run and once a day; the backups themselves are not touched.
- The pages of the admin are in **English and Czech** (Poste.io's two languages, chosen by the browser), including the messages of the tests, of the validation and of the runs. The logs themselves stay in English.

There are no notifications yet (e-mail, Telegram, a webhook, a status for monitoring): they are the next stages of this add-on. Until then a failed backup is seen only in the admin, so look at it from time to time, or watch `poste-backup status` (JSON) from your monitoring.

Restoring

Mailserver backup > *Snapshots and restore* lists the snapshots of a destination (read from it when the page opens), newest first: when, which job, what it holds (configuration, all mail, some mailboxes). From there:

- **Restore a mailbox…** (a snapshot with mail), in three steps on one page:
- **the mailbox**: the mailboxes of the snapshot that hold messages, with how many; one that is no longer on this server is marked. Empty folders (Poste.io makes one for every address, redirects included) are only counted, in a line under the list: there is nothing to restore there;
- **what**, one of two ways:
  - **whole folders, or the messages that match a filter**: the folders of that mailbox with their messages, all ticked (the whole mailbox), or only some of them; and, if you want only some messages, a filter: received from / to (the dates as the server counts them, both days included), the sender contains, the subject contains (case does not matter). Folders and filter go together, for example *only Sent, sender customer.com, from 1 to 7 October*;
  - **single messages**: the messages of the snapshot, newest first, 50 a page, with the date received, the folder, the sender and the subject; choose a folder or all of them, search the sender or the subject (case does not matter, accented letters too), tick the messages you want, across pages and searches (up to 1,000 at a time);
- **where**: the mailbox of this server to restore into (the same one is chosen when it exists; otherwise none is, and nothing starts until you choose), the name of a **new folder** (*Restored* unless you change it) and where to make it: at the top of the mailbox or inside one of its folders.

Before you confirm, the page says where the messages go. They go under the new folder with the snapshot's folders (… / *INBOX*, … / *Sent*), their flags (read, …), and subscribed, so that mail programs show them: **nothing already in the mailbox is changed**. If the name is taken, the folder gets *(2)*. Messages are never put back into their original folders: Dovecot's import does not recognise a message that is still there, so every one of them would appear twice. Move what you need from the restored folder with your mail program, then delete it.

Single messages go under the new folder in the same way, each in its folder (… / *INBOX*, … / *Sent*), and only those.

The restore runs in the background (the engine takes it within a minute: the page shows it, then its result and its log). Only the folders chosen are read from the destination, into a work folder on `/data`, then imported by Dovecot, so they are written twice: the engine adds up their size first and refuses the restore unless `/data` has room for twice that plus 256 MB (a full disk would stop the mail). The messages count towards the quota of the mailbox. Symbolic links in the restored folders are removed before the import, never followed. - **Restore the server…**: the whole server is restored on a **new installation**, not from a running server (its files would be replaced while in use). The page shows the commands of Rebuilding the whole server filled in for that destination and that snapshot (only for snapshots with the configuration); passwords are asked when they run and are never shown. For SFTP the server's keys are the ones confirmed in the admin, and a destination that logs in with a key needs a key of your own: the admin's key pair is not in the backups.

The list of the messages

A snapshot holds files: the sender, the subject and the date of each message come from a list the engine keeps on the server, `/data/var/backup/engine/index.db` (readable only by `mail`, like the rest of `/data/var/backup`). Before every run of a job with mail, the engine looks at the files of the mailboxes, reads the header of the files that are new since the last run (sender, subject; the date received and the size are the file's) and notes the messages that are no longer in their mailbox. The first run reads every message once; later runs read only the new ones. A problem with the list is written in the run's log and never makes the backup fail.

A message is known by its file name in the Maildir up to the flags: the same in every snapshot, and after a move to another folder. A message that is no longer in its mailbox stays in the list, because older snapshots still hold it; it is removed once no snapshot kept on any destination can hold it (it left its mailbox before the oldest snapshot with mail on every destination of the jobs with mail).

The list is part of the backup of the configuration, as a consistent copy (step 4 of Rebuilding the whole server puts it back). A message the list does not know, for example one deleted before the list was kept, is still shown and can be restored: with its date and size,

Copyright © 2026 Antonio Sanna

without sender and subject. Note that the sender and the subject of a deleted message stay in the list (and in the backups of the configuration) as long as a backup may hold the message itself.

If the list is ever damaged, the engine sets it aside (`index.db.damaged`) and makes it again from the mailboxes, and a backup of the configuration goes on without it: the list never makes a backup fail. The first run reads the header of every message; on a very large server it stops after 20 minutes and the next runs read the rest (the log says how many are left).

The page reads the files of a snapshot once and keeps them in a cache (`/var/cache/poste-backup/listing.db`, the last five mailboxes looked at), so that pages and searches answer at once.

Removing the backups of a job

Under the list, *Backups of each job* counts the snapshots of each job on that destination, also of jobs that no longer exist (*deleted job*). **Remove these backups** (after ticking the confirmation) removes every snapshot of that job from that destination and frees the data that only they hold (restic's `forget --prune`); the backups of the other jobs are not touched. It cannot be undone. A job that still exists makes new backups at its next run.

From the command line

`poste-backup restic <destination id> <restic command>` runs restic on a destination's repository with its settings and the key. The ids are in the admin's addresses (`/admin/backup/destinations/<id>/edit`) and in `/data/var/backup/settings/config.json`.

```
# the snapshots on a destination
docker exec mailserver poste-backup restic 0a1b2c3d4e5f snapshots

# what is in the latest snapshot of a job
docker exec mailserver poste-backup restic 0a1b2c3d4e5f ls latest --tag job:<job id>

# one mailbox back, into a separate folder (nothing is overwritten)
docker exec mailserver poste-backup restic 0a1b2c3d4e5f restore latest \
    --target /tmp/restore --include /data/domains/example.com/alice
```

The restored mailbox is then in `/tmp/restore/data/domains/example.com/alice`; copy the messages you need back into the mailbox's `Maildir` (owner `mail`), or into a new folder of it, for example:

```
docker exec mailserver sh -c 'H=/data/domains/example.com/alice; U=alice@example.com
  mkdir -p "$H/Maildir/.Restored/cur" "$H/Maildir/.Restored/new" "$H/Maildir/.Restored/tmp" &&
  cp -a "/tmp/restore$H/Maildir/cur/." "$H/Maildir/.Restored/cur/" &&
  chown -R mail:mail "$H/Maildir/.Restored" &&
  doveadm force-resync -u "$U" "*" && doveadm mailbox subscribe -u "$U" Restored'
```

This copies the messages of the inbox (`Maildir/cur`); the other folders are `Maildir/.<name>/cur` (Poste.io stores mail in the Maildir++ layout: `.Sent`, `.Junk`, …). The procedure is run by the add-on's tests, as written here.

The databases are in the snapshot as consistent copies under `/data/var/backup/engine/staging/` (`users.db`, `dav.db`, `roundcube.db`).

`poste-backup` always runs as the user `mail` (started as root, it switches), so the restored files belong to `mail`. That is right for the mail and for Poste.io's own files owned by `mail`; a file that belongs to another user on the server (for example the webmail's `roundcube.db`, owned by `www-data`) needs its owner set back by root after copying it into place (`chown www-data:www-data ...`).

When the server itself is lost, see the next two sections.

Reading the backups without the server

The backups are ordinary restic *(https://restic.net)* repositories. They can be read from any machine with **restic 0.14** or newer and **rclone 1.60** or newer (on Debian or Ubuntu: `apt install restic rclone openssh-client`; both are also in the image built from this repository): when the mail server is lost, or to look into the backups without touching it.

What you need:

- the **key file** downloaded from *Mailserver backup > Encryption key*;
- the **destination** as it was entered in the admin: its type, server, share or bucket, folder, user and password (or access key and secret key). The add-on's settings are not in the backups, because they hold these passwords: keep a note of them with the key.

The commands are for bash. First the key, which is the last line of the downloaded file (the file as a whole is not the key):

```

```

Copyright © 2026 Antonio Sanna

```
tail -n 1 poste-backup-key-mail.example.com.txt > restic.key
chmod 600 restic.key
export RESTIC_PASSWORD_FILE=$PWD/restic.key
```

Then the destination, one of the following. rclone reads it from variables (`RCLONE_CONFIG_BK_...` define a remote named `BK`), with the password "obscured" as rclone requires; `read -rsp` asks for it without showing it. `RESTIC_REPOSITORY` is the folder of the destination: for SMB the share and the folder in it, for SFTP the folder, for S3 the bucket and the folder in it.

SMB / Samba (add `RCLONE_CONFIG_BK_DOMAIN=...` if the server needs a domain):

```
export RCLONE_CONFIG_BK_TYPE=smb RCLONE_CONFIG_BK_HOST=nas.example.lan RCLONE_CONFIG_BK_USER=mailbackup
read -rsp 'Password: ' P; echo
export RCLONE_CONFIG_BK_PASS=$(printf '%s\n' "$P" | rclone obscure -) && unset P
export RESTIC_REPOSITORY=rclone:BK:share/folder
```

SFTP: the server's SSH key is checked first, as the admin does: read it, compare its fingerprints with the server's (`ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub` on the server), and give it to rclone. A server that shows another key is then refused.

```
ssh-keyscan -p 22 backup.example.com > known_hosts
ssh-keygen -lf known_hosts
export RCLONE_CONFIG_BK_TYPE=sftp RCLONE_CONFIG_BK_HOST=backup.example.com RCLONE_CONFIG_BK_PORT=22 \
       RCLONE_CONFIG_BK_USER=mailbackup RCLONE_CONFIG_BK_KNOWN_HOSTS_FILE=$PWD/known_hosts
read -rsp 'Password: ' P; echo
export RCLONE_CONFIG_BK_PASS=$(printf '%s\n' "$P" | rclone obscure -) && unset P
export RESTIC_REPOSITORY=rclone:BK:/srv/backup/mail
```

A destination that logged in with a key: the key pair made by the admin stayed on the lost server (it is never in the backups). Log in with the account's password, or add a key of yours to the account on the SFTP server and use it instead of the password: `export RCLONE_CONFIG_BK_KEY_FILE=~/.ssh/id_ed25519`.

S3 (add `RCLONE_CONFIG_BK_REGION=...` if the provider needs a region):

```
export RCLONE_CONFIG_BK_TYPE=s3 RCLONE_CONFIG_BK_PROVIDER=Other RCLONE_CONFIG_BK_ENDPOINT=https://s3.example.com \
       RCLONE_CONFIG_BK_ACCESS_KEY_ID=AKIAEXAMPLE
read -rsp 'Secret key: ' P; echo
export RCLONE_CONFIG_BK_SECRET_ACCESS_KEY=$P && unset P
export RESTIC_REPOSITORY=rclone:BK:bucket/folder
```

A folder (a volume mounted by the host): mount the same share or disk on this machine, then:

```
export RESTIC_REPOSITORY=/mnt/mail-backup
```

A REST server: the server's certificate in a file (the `cert.pem` made on the server, or the one the admin shows in *Restore the server*), and every restic command below with `--cacert server.pem` (a password with `@ : / %` or spaces goes into the address percent-encoded: `@` is `%40`):

```
read -rsp 'Password of the REST server: ' P; echo
export RESTIC_REPOSITORY="rest:https://mailserver:$P@nas.example.lan:8000/mailserver" && unset P
```

Then restic, as usual:

```
restic snapshots --group-by tags --latest 1    # the latest snapshot of each job
restic snapshots                               # all of them: time, paths, tags
restic ls <snapshot>                           # what one of them holds
restic restore <snapshot> --target ./restore --include /data/domains/example.com/alice
restic check                                   # the repository is intact
```

The tag `job:<id>` of a snapshot is its job, and its paths say what the job holds: `/data/var/backup/engine/staging` among them for the configuration, `/data/domains` for all the mail. Restored files belong to the user who restores them. A message is one file in the usual Internet message format: renamed to `.eml` it opens in most mail programs.

Reading does not change the repository. `restic check` takes it for itself while it runs (an exclusive lock): the server cannot write to it meanwhile, so run it when no backup is due. Never run `forget` or `prune` from another machine: the server removes its old snapshots itself.

Copyright © 2026 Antonio Sanna

The add-on's tests run these commands, as written here, for each kind of destination on a machine with only Debian 12, restic, rclone and ssh.

Rebuilding the whole server

When the server is lost, a new installation gets everything back from the backups: accounts and their passwords, domains, mail, server settings, certificates, DKIM keys, calendars and address books, webmail settings, spam filter training, the settings of the other add-ons. You need a job that backs up the **configuration** and one that backs up **all the mail** (or one job with both), the key file and the destination (see the previous section).

1. **Install the new server** as the old one was: the image built from this repository, the same version of Poste.io (upgrade afterwards), the same host name and ports, and a new, **empty** data folder. Start it and wait until the admin page answers (Poste.io prepares the data folder); do **not** set it up. Then stop it:

```
docker stop mailserver
```

1. **Put the key file in a folder** of the host, for example `/root/restore`, and open a shell, as root, in a container of the same image with the new server's data folder and that folder. No service runs in it:

```
docker run --rm -it -v /srv/mailserver/data:/data -v /root/restore:/restore --entrypoint bash <image>
```

Use the data folder (or volume) and the image of your installation, and add `--network <name>` if the destination can be reached only from the server's Docker network.

1. **Restore** the latest snapshot of the configuration and the latest of the mail (the same snapshot if one job holds both) straight into `/data`. In that shell:

```
cd /restore
# the key and the destination: as in "Reading the backups without the server"
restic snapshots --group-by tags --latest 1
restic restore <configuration snapshot> --target /
restic restore <mail snapshot> --target /
```

As root, restic puts every file back where it was, with its owner and permissions, over the files of the new installation.

1. **The databases**: the backup holds copies made while they were in use, consistent and checked. Put them in place (with the list of the messages of the backups, if the backup has it), and remove the journal files of the new installation's databases:

```
S=/data/var/backup/engine/staging
cp "$S/users.db" /data/users.db
cp "$S/dav.db" /data/dav.db
cp "$S/roundcube.db" /data/roundcube/roundcube.db
if [ -f "$S/index.db" ]; then install -o mail -g mail -m 600 "$S/index.db" /data/var/backup/engine/index.db; fi
rm -f /data/users.db-wal /data/users.db-shm /data/dav.db-wal /data/dav.db-shm \
      /data/roundcube/roundcube.db-wal /data/roundcube/roundcube.db-shm "$S"/*.db
```

1. **The key**, so that the server can read and continue the old backups; then leave the shell:

```
install -o mail -g mail -m 600 restic.key /data/var/backup/settings/restic.key
exit
```

1. **Start the server** (`docker start mailserver`) and log in to the admin as an administrator of the old server. Then, in *Mailserver backup*, add the destinations again (their **Test** says that the folder already holds a backup repository; for SFTP confirm the server's key again, and with a key login put the new public key on the server) and the jobs. The new jobs continue in the same repositories, and what is already there is not stored again. The snapshots of the old jobs stay, no longer thinned out: remove them by hand when they are not needed any more (see Backup jobs).

Not brought back: the logs, mail still waiting in the outgoing queue, the virus database (downloaded again), the add-on's own settings (step 6), and what is listed as never backed up under What is backed up. If the new server has another IP address, update the DNS records.

The add-on's tests run this procedure as written here: a backup restored on a new installation of the image, then every account, message, setting, database and owner compared with the old server.

Copyright © 2026 Antonio Sanna

Good practice

- **3-2-1**: keep at least one copy **off the machine** and, better, one off the premises (a NAS in another room is not off the premises; an S3 bucket or a server elsewhere is). A job can write to several destinations at once.
- **Two jobs** are usually enough: the **configuration** often (every hour: it is small) and **all the mail** every day, at night. Example keep rules: configuration 24 hourly, 7 daily; mail 7 daily, 4 weekly, 12 monthly.
- **Download the encryption key** at once and keep it in **two places** outside the server (a password manager and a printed copy, for example). Without it no backup can be read. Keep with it the details of each destination (server, folder, user, password): the add-on's settings are not in the backups, and after losing the server the key and these details are all you need (Reading the backups without the server).
- **An account of its own** on each destination, with access to the backup folder only. Not the NAS administrator: whoever controls the mail server would control the whole NAS.
- **Protect the backups from the server** itself: see If the mail server is compromised.
- **Test a restore** now and then (a mailbox into a test mailbox, from *Snapshots and restore*, see Restoring; the backups read from another machine with the key file, see Reading the backups without the server): a backup that was never restored is a hope, not a backup.
- **Look at the dashboard** after changes (a new destination, a new job, an upgrade), and from time to time: until notifications arrive, a failing backup does not tell anyone.
- **Space**: the first backup stores everything; later ones store only what changed (restic deduplicates: data already stored is never stored again). Plan the destination for the mail plus the history kept, and watch the space shown on the dashboard. As an order of magnitude, measured on a real server: a first backup of 9.7 GB of mail took 8 and a half minutes over SMB to a NAS on the same network, the next one 18 seconds (1.8 MB new), and the repository took 3.5 GB on the NAS (restic compresses the data).

A backup of a server that is running

The backup runs while the server works; nothing is stopped.

- **The databases** (accounts, calendars, webmail) are copied with SQLite's own backup command, which gives a consistent copy of a database in use, and each copy is checked before it is stored.
- **The mail** is in the Maildir format, made to be copied while in use: each message is a file that never changes. The only thing that can happen is a message moved or deleted while the backup reads its folder: it is missed by that backup (the run says how many, as a note) and taken by the next one. Dovecot's index files are rebuilt by Dovecot if a copy of them is not up to date.
- **A snapshot of the disk** (one instant for everything) can only be made by the host, with a file system that has snapshots (ZFS, LVM, btrfs) or by backing up the whole virtual machine; a container cannot do it without privileges this add-on does not ask for. If the host already makes them, they are a good complement: this add-on adds copies of the mail server alone, off the machine, encrypted, with single mailboxes to restore.

Security

- **No root.** The admin and the engine both run as `mail`. In Poste.io `/data` itself belongs to `mail`, so a program running as root there could be led by `mail` (with a symbolic link, a renamed folder, a changed database row) to read or write files it should not. Running the backups as `mail` removes that whole class of problems: nothing the admin writes can make the engine read or write more than `mail` already can. Root only makes `/data/var/backup` at start (after checking that `/data/var` is a real folder owned by root) and gives it to `mail`.
- **Mailbox folders** come from Poste.io's database: a folder is used only if it is exactly two levels under `/data/domains`, with no `.` / `..` part and no symbolic link anywhere in the path.
- **Everything is checked twice.** The admin refuses invalid values; the engine validates the settings file again before using it. Server names, folders, users and keys are checked against strict patterns (a folder can never look like an option, nothing can contain `..`), and commands are never run through a shell.
- **Secrets.** The passwords of the destinations and the encryption key are on the volume, readable by `mail` and root only, never sent back to the browser (a password field left empty keeps the saved one), passed to rclone through its environment ("obscured" through stdin, never on a command line). The settings with the passwords are not part of the backups: after rebuilding a server from a backup, enter the destinations again.
- **The backups** are encrypted with the key before leaving the server; the destination only ever sees encrypted data. They contain all the mail and the hashed passwords of the accounts: keep the destinations and the key safe.
- **SFTP** only talks to a server whose SSH key was confirmed.
- **Only super administrators** see the pages.

Copyright © 2026 Antonio Sanna

If the mail server is compromised

To make backups and remove the old ones, the server holds the credentials of the destinations with the right to **write and delete** there. Someone who takes control of the mail server (with the admin password, or through a flaw) can therefore delete or encrypt the backups too, unless the destination protects them. Protect them **on the destination side**, where the mail server has no say:

- a **NAS**: scheduled, read-only **snapshots** of the backup share (Btrfs or ZFS snapshots on Synology, QNAP, TrueNAS…), kept longer than the time you would need to notice a problem;
- **S3**: **versioning** of the bucket, or better **Object Lock** with a retention period, so that a deleted or overwritten object can be recovered;
- an **SFTP server**: snapshots of its file system, or a copy of the backup folder made by that server to a place the mail server cannot reach.
- a **REST server in append-only mode**: the mail server can only add there, whatever it is made to do; see REST server, append-only.

Files and folders

```
/usr/local/bin/poste-backup               the engine: tick, run, status, restic (as mail;
                                          started as root it switches to mail)
/usr/local/lib/poste-backup/lib/          its PHP library (shared with the admin's validation)
/etc/cron.d/poste-backup                  runs "poste-backup tick" every minute, as mail
/etc/cont-init.d/90-backup.sh             at every start: the folder (root), then folders
                                          inside and the key (once) as mail

/data/var/backup/                         mail:mail 0750
  settings/                               (the admin)
    config.json                           destinations and jobs (0600)
    restic.key                            the encryption key (0600)
    key-downloaded, tests.json            when the key was downloaded, last tests
    queue/                                requests: run now, restore a mailbox, remove
                                          the backups of a job
    ssh/<id>, <id>.pub                    key pairs of SFTP destinations
  engine/                                 (the engine)
    state.json                            last run of each job, last restores and removals
    history.jsonl                         one line per run and destination (with the space
                                          used there and on /data), and per verification
    logs/                                 log of each run (kept for the days set in the admin)
    index.db                              the list of the messages of the backups (0600)
    staging/                              consistent copies of the databases, during a run
    restore/                              a mailbox being restored, before it is imported
/var/cache/poste-backup                   restic's cache, and listing.db: the files of the
                                          snapshots last shown in the admin (rebuilt when needed)
/run/poste-backup.lock                    one run at a time
```

Troubleshooting

```
docker exec mailserver poste-backup status          # last results
docker exec mailserver ls -t /data/var/backup/engine/logs | head
docker exec mailserver poste-backup run <job id>    # run a job now and wait for it
```

The dashboard says when the engine has not run for more than ten minutes (while no backup is running): cron is then not running in the container. It also shows why the start script could not prepare the backup folder, if that happened (`/run/poste-backup.start-error`): for example `/data/var` replaced by something that is not a folder owned by root.

Limits

- **One repository per server**: do not point two mail servers to the same folder.
- No notifications yet: they come with the next stages.
- The statistics are per day, and per hour in the 1-day view (real hours: on the day the clock changes, an hour label shows twice or is skipped).
- A mailbox is restored into a new folder: whole, some of its folders, the messages of a filter (dates, sender, subject), or up to 1,000 single messages at a time chosen from a list. The whole server is restored from the command line, on a new installation.
- restic and rclone are the versions of the image's Debian release (restic 0.14, rclone 1.60 on Poste.io 2.5.24).
- The first backup of a large server takes a while (everything is new); later ones store only what changed.
- S3 does not report free space.

Copyright © 2026 Antonio Sanna

Removing it

Build without the section. What stays in the volume is `/data/var/backup` (settings, key, state). The backups on the destinations are not touched: keep the key if you may need them.

Copyright © 2026 Antonio Sanna

Upstream fixes

> ***Adopted upstream.*** *Poste.io **2.5.20** ships all three fixes (the learning scripts call `rspamc --rcpt`, remove their temporary file and are no longer world-writable), so the `upstream-fixes` add-on is gone from the Dockerfiles. On a base older than 2.5.20, build from release 1.2.0 (https://github.com/atsanna/poste.io-addons/releases/tag/v1.2.0). The end-to-end tests still verify the learning behaviour on every build. The rest of this page documents the three problems as they were.*

Three problems of the Poste.io image around spam learning: moving a message to **Junk** (or out of it) is supposed to teach rspamd, and up to 2.5.19 it did not (fixes 1 and 2); and every message learned stayed behind in `/tmp` (fix 3). The first two were reported to Poste.io; releases up to 1.2.0 carried a workaround, marked `UPSTREAM-FIX`, removed now that the base image fixes them.

1. The learning scripts are world-writable

In the base image `/etc/dovecot/sieve` and the `spam` / `ham` pipe scripts are `0777`. Dovecot refuses to execute a world-writable program (`executable ... is world-writable` in `/data/log/mail.log`), so the learning never happens.

Fix: `chmod 0755` on the folder and the two scripts, `0644` on the `.sieve` files.

2. Junk learning goes into a bucket nobody reads

Poste.io configures the rspamd Bayes classifier **per domain** (`per_user` returns the recipient domain), so scanning reads the key `RS<domain>`. The Junk scripts, however, call `rspamc learn_spam` / `learn_ham` **without a recipient**: rspamd then learns into `RS<address>`, which no scan ever reads.

Fix: imapsieve passes the IMAP user to the scripts (`pipe :copy "spam" ["${IMAPUSER}"]`) and the scripts call `rspamc --rcpt "$1" learn_spam`, so the learning lands in the domain bucket. The block ends with `grep -q` checks: if Poste.io changes the scripts, the build fails instead of shipping a fix that no longer applies.

3. Every learned message stays in /tmp

The `spam` / `ham` scripts copy the message into a temporary file (`input_tmpfile=$(mktemp)`) to give it to `rspamc` twice, and never remove it. Each message moved into or out of Junk leaves a copy of itself in the container's `/tmp`, readable by the `mail` user, until the container is recreated.

Fix: right after the `mktemp` line the scripts get

```
trap 'rm -f "$input_tmpfile"' EXIT
trap 'exit 1' HUP INT TERM
```

so the file is removed when the script ends, also when it is stopped. As in fix 2, a `grep` check fails the build if the scripts change.

Are they still needed?

No, from Poste.io 2.5.20 (checked on `poste.io/mailserver:2.5.20` and `analogic/poste.io:2.5.20`, scripts of 2026-10-21). On an older base, look at the base image:

```
docker run --rm --entrypoint sh poste.io/mailserver:<VERSION> -c \
  'ls -l /etc/dovecot/sieve; grep -n "pipe :copy" /etc/dovecot/sieve/*.sieve; grep -n "rspamc.*learn_\|tmpfile" /
etc/dovecot/sieve/spam /etc/dovecot/sieve/ham'
```

(`analogic/poste.io:<VERSION>` for the open source edition.)

- scripts `-rwxr-xr-x` → remove fix 1;
- `pipe` with an argument and `rspamc` with `--rcpt` (or an equivalent solution) → remove fix 2;
- the scripts remove their temporary file (`rm`, a `trap`, or no temporary file at all) → remove fix 3.

If the build fails on the `grep` checks instead, the scripts changed: read them before adapting the block.

Copyright © 2026 Antonio Sanna

Checking that learning works

Move a spam message to Junk, then in the container:

```
redis-cli -s /var/run/redis/redis.sock hget RS<your-domain> learns_spam
```

The counter of **your domain** must increase. (`rspamc stat` sums all keys and can be misleading.) Bayes starts classifying after 200 learned messages of each kind.

Copyright © 2026 Antonio Sanna

Tests



(https://github.com/atsanna/poste.io-addons/actions/workflows/tests.yml)

Every push and pull request builds the **open source** image and runs end-to-end tests against a real Poste.io container. The badge above (and the one in the README) is the result of the last run.

The **Pro** edition is not tested here: its base image comes from the Poste.io registry and needs a licence. The add-ons are the same files in both editions; the Pro image is tested by hand before each release.

What runs

`.github/workflows/tests.yml` *(https://github.com/atsanna/poste.io-addons/blob/main/.github/workflows/tests.yml)* runs seven jobs in parallel. Each builds the image from the `Dockerfile` with the Poste.io version pinned there (`ARG POSTE_IO_VERSION`), then runs `tests/run.sh` *(https://github.com/atsanna/poste.io-addons/blob/main/tests/run.sh)* on it:

Job	Image
`all`	the Dockerfile as it is, every add-on
`external-mailboxes`, `dark-theme`, `activesync-dav`, `activesync-logs`, `backup`	one add-on only, made with `select-addons.sh`

The single-add-on jobs prove that each add-on builds and works without the others; the tests of the add-ons that are not in the image are reported as `SKIP`.

How a run works

`tests/run.sh <image>`:

1. starts a throw-away container from the image: no published ports, its own volume, host name `mail.example.test`; waits until the admin, ActiveSync and IMAP answer. The tests are written in Python (standard library only): Poste.io's base image has no `python3` (External Mailboxes brings it with getmail6), so for an image without that add-on it is installed in the throw-away container;
2. creates the domain `example.test` and three mailboxes (`alice`, the super administrator, `bob` and `carol`) with Poste.io's own console commands; the passwords are random, generated inside the container and never printed;
3. exports the WBXML tables from the Z-Push in the image, so that the test client speaks exactly Z-Push's ActiveSync encoding;
4. runs `tests/test_addons.py` and `tests/test_activesync.py` inside the container; every check prints `PASS` or `FAIL` with what it found;
5. with the backup add-on: starts two more throw-away containers on the same private network as backup destinations, a Samba + SFTP server (`tests/backup-dest` *(https://github.com/atsanna/poste.io-addons/tree/main/tests/backup-dest)*) and an S3 server (rclone's own, pinned image); a `tmpfs` at `/backup/local` plays a volume mounted by the host. It runs `tests/test_backup.py`, then `tests/test_restore.sh` *(https://github.com/atsanna/poste.io-addons/blob/main/tests/test_restore.sh)*, which reads the backups from another container with only restic, rclone and ssh (`tests/backup-reader` *(https://github.com/atsanna/poste.io-addons/tree/main/tests/backup-reader)*) and rebuilds a whole server from them in a new container of the same image;
6. removes the containers, the volumes and the network. The exit code is the number of failed checks.

The admin is driven like a browser (`tests/panel.py` *(https://github.com/atsanna/poste.io-addons/blob/main/tests/panel.py)*): pages are loaded, forms are read from the page (including their CSRF token), filled in and submitted. The phone is simulated with real ActiveSync 14.0 requests (`tests/eas.py` *(https://github.com/atsanna/poste.io-addons/blob/main/tests/eas.py)*): Provision, FolderSync, Sync and Ping, as a device sends them. Nothing is configured behind Poste.io's back.

What is checked

External Mailboxes (`tests/test_addons.py`) - menu entries, the three pages, getmail installed; - the schedule is in the crontab of `mail` (five intervals), nothing for root; getmail folders `0750` owned by `mail`; - a form posted without its CSRF token is refused (HTTP 403, nothing saved); - a mailbox created from the admin is written as a getmail file `0600` owned by `mail`, with the IMAP folder list as a valid Python tuple; - the password is never sent back to the browser; - end to end: a message put in bob's mailbox is fetched by `cron.sh` (run as `mail`, like cron) over IMAP and delivered to alice; the logs page shows it; - rspamd on fetched mail: the getmail file has the `[filter1]` section; the fetched message arrives with `X-Rspamd-Score` (its report is printed); a fetched GTUBE test message arrives in Junk with the `*****SPAM*****` subject; - "Read all" only with "Delete message": off for a new mailbox; with "Delete message" off it is refused (HTTP 400, nothing saved), with it on it is saved; the comment above `read_all` in the generated file says what it does; - log rotation: the default rule (weekly, 1 MB, 4 copies) and its nightly cron entry; the settings in the logs page; saving

Copyright © 2026 Antonio Sanna

without the CSRF token is refused; saved settings (`0640`, owned by `mail`) are applied by the root script; a value not offered by the form is refused; a settings file changed behind the admin gives the defaults and runs nothing; rotation switched off leaves no getmail log in the rule; a forced rotation keeps `getmail.log.1`, recreates `getmail.log` and getmail writes to it; the page shows the old copies; - getmail errors: after a good run `errors.log` is empty (`0640`, owned by `mail`) and the getmail view has no error line; a mailbox with a wrong password is added from the admin and run: the login error is in `errors.log` with its mailbox, on top of the **Getmail Errors** view as an error, and summarised in a red line on top of the getmail view; after the mailbox is deleted from the admin, **Clear** without its CSRF token is refused and with it empties the file, and the red line is gone.

ActiveSync logs (`tests/test_addons.py`) - the menu entry right after *User logs*; the settings folder made by the start script on the fresh volume (`0750`, owned by `mail`); the default rule (both logs weekly, 4 copies, rotated as `www-data`, no `copytruncate`) and its nightly cron entry; - a real Z-Push error: an ActiveSync request with a wrong password is on top of the page as an error, with its account; the Z-Push logs stay `www-data` `0644`; - with known lines in Z-Push's format: newest first, a message written over two lines shown as one, the log text escaped (no HTML from the log), the level filters, the account menu and filter (an account not in the menu is ignored), the case-insensitive search, a search text that is not run by the shell, the errors log; - no Clear form; the rotation settings: refused without the CSRF token, saved `0640` owned by `mail`, applied by the root script, a value not offered by the form refused, a settings file changed behind the admin gives the defaults and runs nothing, rotation switched off leaves no Z-Push log in the rule; a forced rotation keeps `z-push.log.1`, recreates `z-push.log` `0644` owned by `www-data`, Z-Push writes to it at the next request, and the page shows the old copies; - `lint:yaml` and `lint:twig` pass.

Dark theme (`tests/test_addons.py`) - the two `include` lines are in `base.html.twig` once each; - the stylesheet is served as `text/css` and contains the dark rules; - the login page loads the stylesheet and the theme script in `<head>`; - the switch is in the top bar; translations (en, cs) present; - `lint:yaml` and `lint:twig` pass.

Upstream fixes (`tests/test_addons.py`) - the sieve folder and the learning scripts are not world-writable; - a message is moved to Junk over IMAP: the learning script runs, `rspamc` gets the recipient (`--rcpt`), rspamd learns it as spam, and the tokens land in the **domain's** Bayes bucket (the one used for scanning), not in a per-address one.

Calendars and contacts over ActiveSync (`tests/test_activesync.py`) - setup from the admin: DAV enabled in the settings, a calendar and an address book for alice, a calendar for bob shared **read-only** with alice, nothing for carol; - folders on the phone: mail unchanged; alice's calendar is the default one, bob's shared calendar an additional one; contacts only for who has an address book; carol gets mail only; - server → phone: events and a contact written over CalDAV/CardDAV arrive; a "2nd and 3rd of the month" recurrence arrives simplified (known Z-Push limit, checked so that a change is noticed); - phone → server: an event and a contact created on the phone are stored; writing in the read-only calendar is refused (new event 6, change 7, deletion 8) and the calendar is unchanged on the server; - an address book deleted in the admin while the phone still has the folder: a contact sent to it gets an error (not HTTP 500) and the folder is removed; - DAV switched off in the settings: mail keeps working (a new message is delivered, only the calendar gets "hierarchy changed"), Ping answers, the FolderSync removes only calendars and contacts, a new phone gets mail only; switched on again: calendars, contacts and their content come back.

Backup (`tests/test_backup.py`, `tests/backup_unit.php`) - installation: the menu entry, the pages, the folders and the encryption key (`mail`, `0750` / `0700` / `0600`; 48 random bytes), cron running the engine as `mail` every minute; unit checks of the schedules (also across daylight saving changes), of the validation, of the messages and their translations; - destinations: refused without the CSRF token; invalid values refused with a message (a folder on `/data`, `..`, a server name with `;`, a folder that looks like an option); the local volume (a real mount) passes the test, a folder that is not a mount point does not; SMB with the real size of the share (`df` on the Samba server: rclone 1.60 reports half); SMB with a wrong password saved but failing the test; S3; SFTP: the server's key fingerprints shown, nothing trusted until confirmed, then exactly the server's keys; SFTP with a key pair made by the admin; the passwords never sent back to the browser; - the **Test** button of the form: the values typed are tested and nothing is saved or remembered; wrong password, missing CSRF token, invalid server, the saved password reused only for the same server; SFTP keys confirmed in the form; - jobs and runs: a job to the five destinations run from the admin, a snapshot on each with the job's tag, one history line per destination; in the snapshot the database copies, redis, the DKIM keys and the mail, and **not** the key or the settings with the passwords; a deleted message restored identical; the mailbox restore of Backup and restore run as written; the accounts database restored intact from SFTP; keep rules per job; a selected mailbox only; one failing destination does not stop the others, a non-empty folder is never used; a file the backup cannot read gives a warning with how many; - hardening: settings changed behind the admin (an injected command, an invalid id) are reported and nothing of them runs; the engine's files are `mail`'s; a mailbox home changed in `users.db` (`..`, a symbolic link) never leads outside `/data/domains`; the start script refuses a `/data/var` that is not root's; a key that is a symbolic link is refused; - space on each destination, the logs page and the log of a run (names that are not logs, or links, are not found), live progress while a job runs, the pages in Czech, the time zone of the container and the one chosen in the admin, the days the logs are kept, one run at a time; - snapshots and restore: the snapshots of a destination with their job and content, the error of a destination that cannot be read; the restore page: step 1 lists the mailboxes of the snapshot that hold messages with how many and the time of the snapshot, marks one no longer on the server, and only counts the empty folders and the redirects; step 2 ticks the folders with messages and chooses the same mailbox, or none for a mailbox no longer on the server, and says when an address is not in the snapshot; a mailbox restored from the admin into another mailbox and twice into the same one: a new folder *Restored* (then *(2)*) with the same number of messages, subscribed, the message identical to the original, nothing of the mailbox's own mail changed, the engine's work folder removed; only the Sent folder into a new folder inside an existing one; a filter by subject, by the date received (a message of January 2020 sent with its date over IMAP), a date range with no message and a

Copyright © 2026 Antonio Sanna

text that starts with "-": nothing restored, no error; refused with their message: no mailbox chosen, a target that is not a mailbox, no folder, a folder not in the snapshot, a folder name with a dot, a parent folder that does not exist, dates the wrong way round, no CSRF token, a snapshot id that is not one; requests changed behind the admin (a path as address or as folder) refused by the engine, a snapshot that does not exist an error; a symbolic link in a folder restored removed, never followed; the list of the messages: kept only by `mail`, in the backup of the configuration as a copy, a subject and a sender encoded as headers are decoded, every message of a mailbox listed once, a message moved to another folder the same entry, a deleted one marked and kept, the log of the run says how it was updated; the messages of a snapshot in the admin: a search (case and accents do not matter) finds the message with its folder, sender, subject, date and size, nothing found is 0, one folder only, newest first, a message the list does not know shown without details, a folder that is a path or a page below 0 refused; two messages chosen of many restored into another mailbox and only those; refused with their message: none chosen, more than 1,000, a path outside the Maildir; a message that is not in the snapshot refused by the engine; cleaning: a message gone before the oldest backup kept is removed, one gone since then stays; the choice in Czech; the page of commands to restore the server (values quoted, no password, for SFTP with a key a key of one's own and the confirmed server keys), offered only for snapshots with the configuration; the backups of a deleted job removed after the confirmation, those of the other jobs untouched; - verification, statistics, space warnings: a new destination is verified every Sunday at 04:00, a quarter of the data; *Verify now* reads the first quarter, then the second; the settings saved from the form; all the data when set so; **one byte of a data file changed on the destination (through rclone) is found** by the verification, shown among the problems of the dashboard, on the destination's page and in the logs; a verification due by its schedule is run by the engine's tick; a history line of a backup has the space used on the destination and on `/data`; the statistics of 30 days, 90 days and a year (another value is 30), with the space, the data added and the duration of the job; the space warnings refused without the CSRF token and when yellow is not below red, then saved and taken by the bars; in Czech; - a REST server in append-only mode (restic's official `rest-server`, HTTPS with a certificate made by `run.sh`): the form's Test shows the server's certificate first (its fingerprint compared with `openssl`), then, confirmed, says reachable and empty; a wrong password, another certificate and an address that is not in the certificate are refused with their reason; three runs with "keep 1 per hour" leave the three snapshots there (nothing removed from this server) and the log says why; no password in the log; the snapshots page shows the note and no removal there, and the engine refuses one written behind the admin; a verification works; the commands to restore the server carry the certificate and `--cacert`, no password; another certificate saved for the server makes the backup fail before anything is sent; in Czech. Unit checks: the verification settings and their limits, when a verification is due, the request, the charts computed from history lines (one run counted once, older lines and verifications left out); a REST destination (accepted without a certificate yet, refused values, the address with the encoded password in the environment and never on the command line, `--cacert`, IPv6 in brackets).

Restore (`tests/test_restore.sh`, `tests/read_backups.sh`, `tests/rebuild_server.sh`, `tests/test_restore.py`): the two procedures of Backup and restore for a lost server, run as written there. - reading the backups without the server, on a Debian 12 container with only restic, rclone and ssh and the key file downloaded from the admin: from SMB, SFTP (with the password, and with a key of one's own), S3 and a folder, the snapshots are listed and a message restored identical; `restic check` finds no errors; another key, the whole downloaded file (instead of its last line) and an SFTP server with another key are refused; - rebuilding the whole server: a new installation of the image started once and stopped, the backup restored as root with no service running, the server started again; then the accounts and their (hashed) passwords, the domains, every message, the DKIM keys, the certificates, the calendars and address books, the webmail database, redis and the owners and permissions are those of the old server, the list of the messages of the backups is back; a user logs in over IMAP with the old password and finds the old message; the old administrator logs in; the old key is back and the new server reads the old backups from a destination entered again.

Running the tests yourself

With Docker, from the repository:

```
docker build -f Dockerfile --build-arg POSTE_IO_VERSION=2.5.24 -t poste-addons-test .
sh tests/run.sh poste-addons-test
```

With all the add-ons a run takes about fifteen minutes after the build (measured on a desktop machine), most of it for the backup and restore tests; without the backup add-on, about three. The same script works on a Pro image built with `Dockerfile-pro`.

Copyright © 2026 Antonio Sanna