grommunio Setup Guides

Official documentation

Version 2026.09.16 · docs.grommunio.com

Setup guides

The **Administration** manual describes grommunio as a system. The guides in this section are **task-oriented**: each one takes a single component or operational topic from prerequisites to a verified, working result, with a verification checklist and a troubleshooting table at the end.

All guides assume a working grommunio installation — the appliance set up as described in the Quickstart, with DNS, TLS and the first domain and user in place. Start with the post-installation checklist if you have just finished `grommunio-setup`.

Post-installation checklist
Verify services, repositories and ports; set up MX, SPF, DKIM and DMARC; prove end-to-end mail flow.

Antispam (Rspamd)
Understand scores and actions, test safely, train Bayes, manage allow/deny lists and operate grommunio-antispam.

SSO with Keycloak
Central login and MFA for grommunio Web and the collaboration apps with grommunio-auth and Keycloak.

Meet
Deploy browser-based video conferencing, optionally behind SSO, and connect it to the calendar.

Chat
Set up team messaging, activate it per domain and user, and verify multi-tenancy.

Files & Office
File sync and online document editing, integrated with grommunio Web.

Archive
Compliance e-mail archiving with full-text search and restore into the mailbox.

Mobile devices (EAS)
Exchange ActiveSync for phones and tablets: enable, connect, manage devices and policies, remote wipe.

Backup & restore
Consistent backups, a full disaster-recovery rehearsal and single-mailbox restores.

Related topics elsewhere

- High availability — the Pacemaker/DRBD reference cluster, including failover test scenarios.
- Kerberos SSO — transparent sign-in for domain-joined Windows clients (the alternative to Keycloak-based SSO).
- IMAP migration with imapsync — moving mailboxes from any IMAP server.
- Operations — TLS, updates, backup artefacts and size limits.

Offline reading

This section is also available to download for offline use:

- Download as PDF
- Download as EPUB

Post-installation checklist

When `grommunio-setup` finishes, the appliance is configured but empty: no mail domain, no user, no DNS records that other mail servers could trust. This page covers everything between that point and a system that delivers mail reliably in both directions. Work through the numbered steps in order; each one ends with a check you should pass before moving on.

The installation itself is documented elsewhere and is not repeated here: downloading the ISO and sizing the machine in the Quickstart, the console user interface and every dialog of the setup wizard in Guided Installation. If the wizard did not complete, fix that first (the log is `/var/log/grommunio-setup.log`).

How it fits together

A grommunio appliance is a set of cooperating services. After setup, all of them should be running; the checks in this guide follow the path a message takes through them.

Unit	Role	Listens on
`nginx`	Reverse proxy for grommunio Web, Admin UI, AutoDiscover, EAS, DAV; TLS termination	80, 443, 8080/8443 (Admin)
`postfix`	MTA: receives mail from the Internet and from clients, hands it to the antispam milter and to local delivery	25, 587 (465 where implicit-TLS submission is enabled)
`grommunio-antispam`	Rspamd-based filter: spam scoring, DKIM signing, optional anti-virus	11332 (milter), 11334 (controller), local only
`gromox-delivery-queue`, `gromox-delivery`	Local delivery: receive from Postfix, convert to MAPI objects, store in the mailbox	24 (LMTP/SMTP), local only
`gromox-http`	MAPI/HTTP, RPC over HTTP, EWS, AutoDiscover backend behind nginx	local only
`gromox-imap`, `gromox-pop3`	IMAP and POP3 access	143/993, 110/995
`gromox-midb`, `gromox-zcore`, `gromox-event`, `gromox-timer`	Message index, PHP-MAPI core, event bus, scheduled tasks	local only
`grommunio-admin-api`	Admin REST API behind nginx (Admin UI, `grommunio-admin` CLI)	via nginx
`mariadb`, `redis@grommunio`, `php-fpm`, `saslauthd`	Database, cache, PHP runtime for Web/Sync/DAV, SMTP authentication	local only

The public ports and the firewall defaults are listed in the Quickstart; the complete mail flow is described in Architecture.

Prerequisites

- `grommunio-setup` has completed; you know the FQDN (for example `mail.example.com`) and the primary mail domain (`example.com`) you entered, and the `admin` password.
- Root access to the appliance (console or SSH).
- Control over the public DNS zone of the mail domain and over the reverse DNS (PTR) of the public IP address, usually via the hosting provider.
- Inbound TCP 25 and 443 reachable from the Internet on the public IP; TCP 80 if you use Let's Encrypt.
- An external mailbox (another provider) to test delivery in both directions.

1. Check services, ports and repositories

A loaded web page does not prove that the mail path works. Verify the service set, the listening sockets and the package repositories before creating anything.

```
systemctl --failed
systemctl --type=service --state=running | grep -E 'gromox|grommunio|postfix|nginx|mariadb|redis|saslauthd|php-fpm'
ss -tulpn
hostname -f
timedatectl
getent hosts download.grommunio.com
zypper lr -u
zypper refresh
zypper list-patches
```

Expected results:

- `systemctl --failed` lists no units.
- The running services include `grommunio-admin-api`, `grommunio-antispam`, the `gromox-*` daemons from the table above, `mariadb`, `nginx`, `php-fpm`, `postfix`, `redis@grommunio` and `saslauthd`.
- `ss -tulpn` shows listeners on 25, 80, 443, 587, 110, 143, 993, 995 and on the Admin ports (8080; 8443 in addition after the TLS switch in step 2). Internal listeners on 24 (`delivery-queue`), 11332/11334 (`rspamd`), 3306 (`mysqld`) and 6379 (`redis-server`) are bound to localhost only.
- `hostname -f` returns the FQDN, not `localhost` or a short name; `timedatectl` reports a synchronised clock and the intended time zone.
- `getent hosts download.grommunio.com` resolves and `zypper refresh` succeeds. If it does not, fix the resolver first: a broken resolver is the most common reason for failing repository access and, later, for failing outbound mail. Hostname, resolver and time settings are changed through the CUI, see Guided Installation.

The appliance ships with `firewalld`, and `grommunio-setup` opens the service ports listed in the Quickstart in the `public` zone. Confirm the rule set and close what you do not offer:

```
systemctl is-active firewalld
firewall-cmd --list-all
```

Expected result: the services and ports from the Quickstart list, nothing more. Remove the entries for protocols you do not provide (for example POP3) and, on systems without a host firewall (`firewalld` not installed or inactive), restrict the Admin ports and unused services on the perimeter firewall or the hypervisor instead.

2. Open the Admin UI and grommunio Web

Open the Admin UI at `https://<FQDN>:8443/` and sign in as `admin`. The Admin UI is served on its own ports, not below the `/web/` path of grommunio Web. If the page does not load on 8443, the Admin API is still served unencrypted on port 8080 only, as it is during provisioning; switch it to TLS as described in Admin API TLS configuration before exposing it anywhere.

On the Dashboard, the services panel must show all grommunio and Gromox services as running, and the versions panel lists the installed components. Then open grommunio Web at `https://<FQDN>/web/`; the login page must load over HTTPS without an error other than the expected certificate warning if you chose a self-signed certificate.

Two housekeeping tasks belong here:

- The setup log `/var/log/grommunio-setup.log` contains generated credentials. Copy it to a safe place and remove it from the appliance, or at least restrict access to it.
- Change the `admin` password if you kept the generated one: `grommunio-admin passwd` (no user argument sets the password of `admin`). The `admin` account and the system `root` account are independent; see grommunio Admin User.

3. Create the first domain and user

The wizard records the primary mail domain but does not create it as a mail domain. Create the domain, then a first user. Both can be done in the Admin UI (Adding a domain, Adding a user) or on the command line. The CLI is reproducible and easy to script, so it is used here.

```
grommunio-admin domain create -u 25 --title "Example Inc." example.com
grommunio-admin user create --lang en_US --pop3-imap true --privWeb true --smtp true alice@example.com
grommunio-admin passwd alice@example.com
grommunio-admin user query username maildir pop3_imap smtp privWeb status
```

- `-u` sets the maximum number of users of the domain and is required. Add `--create-role` if you want a domain administrator role created together with the domain.

- The domain of the user is taken from the address; the domain must exist. `--privWeb`, `--smtp` and `--pop3-imap` grant login to grommunio Web, sending via SMTP and IMAP/POP3 access. Defaults for these flags can be preset in the Admin UI under *Defaults*.
- `grommunio-admin passwd` prompts for the password; `-a` generates one.

Expected result: the query lists the user with a `maildir` path below `/var/lib/gromox/user/` and status `normal` (`0`). Confirm the credentials without a browser:

```
grommunio-admin user login alice@example.com
```

Further recipes, including aliases, feature switches and bulk operations, are in Common administration tasks; the complete option lists are in grommunio-admin domain and grommunio-admin user.

4. Web login and a local test mail

Sign in to `https://<FQDN>/web/` as `alice@example.com`. An empty Inbox and a working calendar view confirm that the mailbox store, `gromox-zcore` and PHP are functional.

Next, inject a message on the server itself. This exercises Postfix, the antispam milter and the Gromox local delivery path without depending on DNS or the Internet:

```
printf 'From: alice@example.com\nTo: alice@example.com\nSubject: Local delivery test\n\nThis message was injected locally.\n' | se
postqueue -p
gromox-mailq
journalctl -u postfix -u gromox-delivery -u gromox-delivery-queue --since "10 minutes ago"
```

Expected result: the message appears in the Inbox in grommunio Web within seconds, both queues are empty and the journal shows Postfix handing the message to the local transport on port 24 and `gromox-delivery` storing it. A message that stays in the queue points to a problem in the local chain; see Troubleshooting below and Mail requeueing.

The same queues are visible in the Admin UI under Mail queue, together with flush, requeue and delete actions.

5. Publish the DNS records

A server can run perfectly and still be unable to exchange mail with the rest of the Internet. Receiving mail requires an MX record; being accepted by other servers requires a matching forward and reverse name, SPF, DKIM and DMARC; Outlook, mobile devices and other clients find the server through AutoDiscover. Publish the following records in the public zone of `example.com`, replacing the example IP address:

Record	Name	Value (example)	Purpose
A / AAAA	`mail.example.com`	public IPv4 / IPv6 of the appliance	Name of the server; must match the certificate and the HELO name
MX	`example.com`	`10 mail.example.com`	Inbound delivery
PTR	reverse of the public IP	`mail.example.com`	Reverse DNS; set at the hosting provider; must resolve back to the same IP
A / AAAA or CNAME	`autodiscover.example.com`	`mail.example.com`	AutoDiscover for Outlook, EAS devices and other clients
SRV (optional)	`_autodiscover._tcp.example.com`	`0 0 443 mail.example.com`	AutoDiscover for clients that use the SRV method
TXT	`example.com`	`v=spf1 a mx -all`	SPF: only hosts in the A and MX records may send for the domain
TXT	`dkim._domainkey.example.com`	`v=DKIM1; k=rsa; p=<public key>`	DKIM public key; generated in step 6
TXT	`_dmarc.example.com`	`v=DMARC1; p=none; rua=mailto:dmarc@example.com`	DMARC policy and aggregate reports

Notes on the values:

- The certificate must cover every name clients connect to, at least `mail.example.com` and `autodiscover.example.com`. If you chose Let's Encrypt in the wizard, both names must point to the appliance before the certificate can be issued; see TLS configuration and Certificate management.
- The SPF value above is the one the Admin UI proposes. If outbound mail leaves through a relayhost or a third-party service, add that sender with an `include:` or `ip4:` mechanism, otherwise your own mail fails SPF. Use `~all` (softfail) while you are still validating, then

move to `-all`.

- Start DMARC with `p=none`: receivers report but do not act, and the aggregate reports sent to the `rua` address tell you whether SPF and DKIM align for all legitimate mail. Tighten to `p=quarantine` and finally `p=reject` once the reports are clean for a while. The `rua` mailbox must exist; create it as a user or alias.
- The Admin UI additionally checks `autoconfig.example.com` (Thunderbird-style Mail Autoconfig) and SRV records for `_submission`, `_imaps`, `_pop3s`, `_caldavs` and `_carddavs`. They are optional; add them if you want clients to auto-configure without AutoDiscover. See autoconfig(7) and autodiscover(7).
- Split-horizon setups must publish the `autodiscover` record in the public view as well; see the AutoDiscover KB article.

Verify from outside the appliance's own network where possible, because your internal resolver may answer differently from the Internet:

```
dig +short MX example.com
dig +short A mail.example.com
dig +short -x 203.0.113.10
dig +short TXT example.com
dig +short TXT _dmarc.example.com
dig +short SRV _autodiscover._tcp.example.com
```

The Admin UI performs the same checks: open *Domains*, select the domain and look at the DNS health section. Each record is queried through the local resolver and through an external resolver, and the reachability check compares the server's detected public IP with the MX target. A new domain shows most items as missing; after publishing, every item you configured should turn green. DNS changes take up to the zone's TTL to propagate.

6. Create the DKIM key pair

DKIM signing is done by `grommunio-antispam`; the key pair is generated per domain from the Admin UI, which also hands you the DNS record to publish.

1. In the Admin UI, open *Domains*, select `example.com` and open the DKIM entry in the DNS health section. On a new domain, the dialog reports that the record `dkim._domainkey.example.com` is not resolvable and shows a placeholder for the public key.
2. Click *Generate DKIM keypair*. Keep *Type* `rsa` (`ed25519` is offered, but not every receiver verifies it) and *Output mode* `dns` for a standard TXT record. Leave *Selector* empty to use the default selector `dkim`; if you set another selector, the record name changes accordingly.
3. Click *Generate*. The dialog now shows the TXT record for DNS and a list of commands to run on the server.

On the server, the private key is written to `/var/lib/grommunio-admin-api/example.com.dkim.key` (owner `grommunio`, mode 0440; the public key is stored alongside with the suffix `.pub`, and a previous key pair is kept with the suffix `.old`). Publish only the TXT record; the private key never leaves the server.

The Admin API cannot write into the antispam data directory, so the key must be made available to `grommunio-antispam` manually. Run the commands shown in the dialog, which correspond to the following (replace `example.com`):

```
postconf -e 'non_smtpd_milters = $smtpd_milters'
mkdir -m 0700 /var/lib/grommunio-antispam/dkim
cp /var/lib/grommunio-admin-api/example.com.dkim.key /var/lib/grommunio-antispam/dkim/
chown -Rf groas:grommunio /var/lib/grommunio-antispam/dkim
chmod 600 /var/lib/grommunio-antispam/dkim/example.com.dkim.key
systemctl restart postfix
```

- `smtpd_milters` already points to `inet:localhost:11332`, but `non_smtpd_milters` is empty by default. Setting it passes mail that enters Postfix locally (the `sendmail` command, bounces, notifications) through the same milter as SMTP mail, so that it is signed as well.
- `grommunio-antispam` runs as user `groas` in group `grommunio`; the key must be readable by that user and by nobody else. If the package already created `/var/lib/grommunio-antispam/dkim`, `mkdir` reports that the directory exists; continue with the next command.
- The signing configuration in `/etc/grommunio-antispam/local.d/dkim_signing.conf` looks for the key at `/var/lib/grommunio-antispam/dkim/$domain.$selector.key` with the selector `dkim`, which is exactly the file name the Admin UI produces. If you chose a different selector in the dialog, rename the copied key to `example.com.<selector>.key` and change `selector` in that file accordingly. The default configuration signs mail from authenticated clients and from local senders.
- Repeat the copy for every additional domain you create; the directory only needs to be created once.

Check the result by sending a message from grommunio Web to an external mailbox and inspecting its headers there: a `DKIM-Signature:` header with `d=example.com; s=dkim` must be present and the receiver's `Authentication-Results:` header must report `dkim=pass`. A locally delivered message does not prove signing, because it never passes an external verifier.

> **⚠Caution**
>
> Keys under `/var/lib/grommunio-admin-api/` and `/var/lib/grommunio-antispam/dkim/` are part of the configuration and must be included in the backup. Generating a new key pair for a domain replaces the record you have to publish; rotate the DNS record together with the key.

7. End-to-end send and receive test

With DNS and DKIM in place, test the complete path in both directions against a real external mailbox.

Outbound:

1. In grommunio Web, send a message from `alice@example.com` to the external mailbox.
2. `postqueue -p` on the appliance must return an empty queue within a few seconds; `journalctl -u postfix --since "5 minutes ago"` must show `status=sent` for the recipient's MX.
3. At the receiver, open the headers. `Authentication-Results:` should show `spf=pass`, `dkim=pass` and `dmarc=pass`, and the message must not land in the spam folder.

Inbound:

1. Reply from the external mailbox to `alice@example.com`.
2. `journalctl -u postfix -u gromox-delivery --since "5 minutes ago"` must show the connection from the remote MX, the milter verdict and the local delivery.
3. The reply appears in the Inbox in grommunio Web; the antispam headers added by `grommunio-antispam` are visible in the message source.

Client access:

```
PASS='<strong-password>' gromox-dscli -e alice@example.com
openssl s_client -connect mail.example.com:993 -servername mail.example.com </dev/null
```

`gromox-dscli` performs the AutoDiscover lookup the way a client does and prints the resulting configuration (gromox-dscli(8)); the `openssl` call shows the certificate chain presented on IMAPS, which must be trusted by your clients and cover the name they connect to. For Outlook specifics, see the Outlook KB article.

Finally, confirm from a network you do not control (a mobile connection, a remote host) that the server is not an open relay: connecting to port 25 and issuing `RCPT TO` for a foreign domain without authentication must be rejected with `554 5.7.1 Relay access denied`. A test from inside your own network says nothing about this.

8. Updates, backup and mailbox maintenance

Updates

Treat the appliance like any production Linux mail server: refresh, review, patch in a maintenance window, then re-check the services.

```
zypper refresh
zypper list-patches
zypper patch
zypper ps -s
systemctl --failed
gromox-mailq
```

`zypper ps -s` lists services that still run old binaries after an update and should be restarted. Updates can also be triggered from the Admin UI (Updates tab). Repository types, subscription channels and the full procedure are in Updating grommunio; the reasoning behind the release cadence is in Update Cycle.

Backup

Define the backup before the first real mailbox arrives. The artefacts are the mailbox stores under `/var/lib/gromox/user/` and `/var/lib/gromox/domain/`, the MariaDB database `grommunio`, `/etc/grommunio*`, the certificates and, from step 6, the DKIM keys. Snapshot-

based and file-based approaches are described in Backup & Disaster Recovery; a complete restore procedure including single-mailbox recovery is in Backup and restore. Test the restore, not just the backup.

Mailbox maintenance with gromox-cleaner

Gromox ships `gromox-cleaner.timer` and the one-shot `gromox-cleaner.service`. The service runs `gromox-mbop` over every local mailbox, hard-deletes messages that have been soft-deleted for longer than the retention period, and removes attachment files that are no longer referenced by any message (gromox-cleaner.service(8)). Soft-deleted means the message is flagged as deleted and recoverable by the client until it is purged; the cleaner is what finally frees the space. It is not a backup, not an archive and does not empty the Deleted Items folder by itself.

The retention is taken from `softdelete_purgetime` in `/etc/gromox/gromox.cfg`; if the directive is not set, the unit falls back to `30d`. The timer runs daily, but it is not enabled after installation.

```
systemctl cat gromox-cleaner.service
systemctl status gromox-cleaner.timer
systemctl list-timers --all | grep gromox-cleaner
```

Once retention, backup and any archiving requirements are agreed, enable the timer and run one pass by hand:

```
systemctl enable --now gromox-cleaner.timer
systemctl start gromox-cleaner.service
journalctl -u gromox-cleaner.service --since "10 minutes ago"
```

Expected result: the service exits successfully and logs the mailboxes it processed; on a fresh system there is nothing to purge. The service requires `gromox-http` to be running. To also empty the Deleted Items folder of all users after a set time, extend the unit as shown in Mailbox maintenance.

Verification checklist

Area	Check	Expected result
Services	`systemctl --failed`; services panel on the Dashboard	No failed units; all grommunio and Gromox services running
Network	`ss -tulpn`; perimeter firewall rules	Listeners on 25, 80, 443, 587, 110/995, 143/993 and the Admin ports; only intended ports reachable from outside
Name resolution and time	`hostname -f`, `getent hosts download.grommunio.com`, `timedatectl`	FQDN returned; external names resolve; clock synchronised
Repositories	`zypper refresh`, `zypper list-patches`	Refresh succeeds; patch list is known and applied in a maintenance window
Admin UI	`https://<FQDN>:8443/`	Login as `admin` works over TLS; Dashboard shows versions and services
grommunio Web	`https://<FQDN>/web/`	Login as the test user works; Inbox and calendar open
Domain and user	`grommunio-admin domain list`, `grommunio-admin user query ...`, `grommunio-admin user login`	Domain active; user has a maildir, status `normal`, login succeeds
Local delivery	`sendmail` test, `postqueue -p`, `gromox-mailq`	Message in Inbox; both queues empty
DNS	`dig` checks; DNS health in the Admin UI	MX, A/AAAA, PTR, autodiscover, SPF, DKIM and DMARC resolve externally and match
TLS	Browser, `openssl s_client` on 443 and 993	Trusted chain; certificate covers `mail.` and `autodiscover.` names
DKIM	Headers of a message received externally	`DKIM-Signature` present, `dkim=pass`
Outbound delivery	Message to an external mailbox	Delivered to Inbox; `spf=pass`, `dkim=pass`, `dmarc=pass`
Inbound delivery	Reply from the external mailbox	Delivered to the user's Inbox; logged by Postfix and `gromox-delivery`
Relay	`RCPT TO` a foreign domain from outside, unauthenticated	Rejected with `Relay access denied`
AutoDiscover	`gromox-dscli -e alice@example.com`	Configuration returned for the mailbox
Reboot	`systemctl reboot`, then repeat the service and login checks	All services return; Web and Admin UI reachable

Area	Check	Expected result
Maintenance	`systemctl list-timers`	`gromox-cleaner.timer` and, with Let's Encrypt, `grommunio-certbot-renew.timer` scheduled
Backup	Documented procedure, first run completed	Mail stores, database, `/etc/grommunio*`, certificates and DKIM keys covered; restore tested

Troubleshooting

Work from the symptom to the cause: check the service state, read the relevant journal, verify the URL or DNS name, and only then change configuration. Log locations are listed in Troubleshooting; verbose logging options in Debugging messaging services.

Symptom	Likely cause	What to check / fix
`zypper refresh` fails, repositories unreachable	Resolver or gateway not configured on the appliance	`getent hosts download.grommunio.com`, `ip route`; correct DNS and gateway via the CUI network configuration
Admin UI does not answer on 8443	Admin API still served unencrypted on 8080	Follow Admin API TLS configuration; `nginx -t`, `systemctl restart nginx`
Browser or client rejects the certificate	Self-signed certificate, or `autodiscover.example.com` missing from the SAN list	Import a proper certificate or re-issue it with all names, see Certificate management
Let's Encrypt issuance failed during setup	Port 80 not reachable from the Internet, or DNS not yet pointing to the appliance	`/var/log/grommunio-setup.log`; fix DNS/firewall and request the certificate again as shown in TLS configuration
User cannot log in to grommunio Web	Wrong password, `privWeb` not set, or user status not `normal`	`grommunio-admin user login`, `grommunio-admin user query username privWeb status`, `grommunio-admin passwd`
Local test mail never reaches the Inbox	Message stuck in Postfix or in the Gromox queue; `gromox-delivery` or `gromox-http` not running	`postqueue -p`, `gromox-mailq`, `journalctl -u postfix -u gromox-delivery -u gromox-delivery-queue`; Mail requeueing
Outbound mail rejected or filed as spam	Missing or mismatching PTR, SPF failure, no DKIM signature, dynamic or listed IP address	DNS health in the Admin UI; `Authentication-Results` at the receiver; `dig -x` on the public IP; check the IP against common blocklists
No `DKIM-Signature` header on outgoing mail	Key not readable by `groas`, key file name does not match `<domain>.<selector>.key`, or locally submitted mail bypasses the milter	Repeat the commands from step 6; `ls -l /var/lib/grommunio-antispam/dkim/`; `postconf non_smtpd_milters`; `journalctl -u grommunio-antispam`
Mail from the Internet never arrives	MX record wrong, port 25 blocked by the provider or firewall, or domain not created in grommunio	`dig MX example.com`; `ss -tulpn` for port 25; `journalctl -u postfix`; `grommunio-admin domain list`
Outlook or mobile device cannot auto-configure	`autodiscover` record missing in the public zone, or certificate does not cover the name	`gromox-dscli -e user@example.com`; autodiscover(7); Outlook KB
Login or TLS errors after a reboot, timestamps wrong	Time not synchronised	`timedatectl`; configure NTP via the CUI timesync dialog
`gromox-cleaner.service` fails to start	`gromox-http` not running (the unit requires it)	`systemctl status gromox-http`; then `systemctl start gromox-cleaner.service`

Operating notes

- Logs: all grommunio and Gromox services log to the journal (`journalctl -u <unit>`); nginx and the PHP services for Web, Sync and DAV write to files under `/var/log/nginx/`, `/var/log/grommunio-sync/` and `/var/log/grommunio-dav/`. The Admin UI exposes the journals under Logs.
- Queues: watch `postqueue -p` and `gromox-mailq`, or the Mail queue view in the Admin UI. A growing queue is the earliest sign of a delivery problem.
- Certificates: with Let's Encrypt, `grommunio-certbot-renew.timer` renews weekly; port 80 must stay reachable. With imported certificates, track the expiry date yourself and restart the services after replacing the files.
- Monitoring: service state, queue length, disk usage of `/var/lib/gromox`, certificate expiry and the DMARC aggregate reports are the minimum set to watch.
- Firewall: `firewalld` on the appliance opens every service port after setup. Keep the Admin ports (8080/8443) restricted to administrative networks, on the host firewall (`firewall-cmd --remove-port=8080/tcp --zone=public --permanent`, then `firewall-cmd --reload`, once the TLS switch is done) and on the perimeter, and expose only 25, 80, 443, 587 and the IMAP/POP3 ports you need.

Related pages

- Quickstart
- Guided Installation (grommunio Appliance)

Antispam with grommunio-antispam (Rspamd)

grommunio-antispam is the mail filter of a grommunio installation. It is a packaged build of Rspamd with grommunio integration and a dedicated Redis instance. Postfix hands every message to it before the message reaches a mailbox or leaves the system, and its verdict decides whether a message is delivered, tagged as spam, greylisted or rejected.

This guide walks through the service as it is shipped with the grommunio Appliance: how mail flows through it, how to read its decisions, how to test it safely, how to train it without degrading it, and which parts of its state you must back up. At the end you have a filter whose behaviour you can explain from its history and logs instead of adjusting thresholds blindly.

How it fits together

Rspamd is a modular filter. Each check (SPF, DKIM, DMARC, DNS blocklists, Bayes, fuzzy hashes, regular-expression rules, URL reputation, anti-virus hooks, …) inserts a *symbol* with a weight into the result. The sum of the weights is the *score*; the score is compared against the configured thresholds and yields an *action*. There is no single yes/no decision, which is why diagnosing a misclassified message always means looking at its symbols.

Component	Listens on	Role
Postfix `smtpd`	25, 587	Receives mail and passes it to grommunio-antispam over the milter protocol (`smtpd_milters = inet:localhost:11332`)
`rspamd_proxy` worker	`127.0.0.1:11332`, `[::1]:11332`	Milter endpoint for Postfix; forwards to the scanning workers and applies the verdict
`normal` worker	`127.0.0.1:11333`, `[::1]:11333`	Scans messages and produces symbols, score and action
`controller` worker	`127.0.0.1:11334`, `[::1]:11334`	Web interface, HTTP API, learning, statistics; reached through the Admin UI path `/antispam/`
Redis instance `redis@grommunio`	`127.0.0.1:6379`	Bayes tokens, history, greylisting and rate-limit state, neural network data, DMARC reporting data
`gromox-delivery-queue` / `gromox-delivery`	port 24 (`virtual_transport = smtp:[::1]:24`)	Receive the accepted message from Postfix and store it in the mailbox

The processing sequence for an incoming message is:

1. Postfix receives the message and calls grommunio-antispam through the milter interface (`milter_protocol = 6`).
2. grommunio-antispam evaluates the message (and, if configured, hands it to an anti-virus scanner) and returns action, score and headers.
3. Postfix applies the action: `reject` becomes a permanent SMTP error, `soft reject` / `greylist` a temporary one, `add header` adds the header `X-Spam: Yes` to the message, and accepted messages are relayed to `gromox-delivery-queue`.
4. `gromox-delivery` converts the message to a MAPI object. If `lda_junk_rules` in `/etc/gromox/gromox.cfg` matches one of the message headers, the message is placed in *Junk E-mail* instead of the Inbox (see gromox.cfg(5)). The directive is empty by default.

Outgoing mail submitted on port 587 passes the same milter; there grommunio-antispam performs DKIM/ARC signing rather than filtering. Signing is set up in the post-installation guide. The overall mail flow is shown in the architecture chapter.

Paths that matter:

Path	Content
`/etc/grommunio-antispam/`	Configuration tree of the packaged Rspamd (generic Rspamd documentation uses `/etc/rspamd/` for the same layout)
`/etc/grommunio-antispam/local.d/`	Your settings; merged into the shipped defaults
`/etc/grommunio-antispam/override.d/`	Your settings; replace the shipped section entirely
`/etc/grommunio-antispam/local.d/worker-controller.inc`	Controller password for the web UI
`/etc/grommunio-antispam/local.d/redis.conf`	Connection to the `redis@grommunio` instance (`read_servers` / `write_servers`)
`/etc/grommunio-antispam/maps.d/`	Shipped maps (allowlists for SPF/DKIM/DMARC, redirectors, suspicious TLDs, …); do not edit
`/etc/grommunio-antispam/modules.d/`, `scores.d/`	Shipped module defaults and symbol scores; do not edit
`/var/lib/grommunio-antispam/`	Service data: DKIM keys (`dkim/`), statistics (`stats.ucl`, `rspamd.rrd`), cached maps, the maps you create
`/var/log/grommunio-antispam/rspamd.log`	Scan log (one line per message with action, score and symbols)

Path	Content
`/etc/redis/grommunio.conf`, `/var/lib/redis/default/`	Configuration and persistence (`dump.rdb`) of the Redis instance, i.e. the learned state

Files ending in `.conf` configure a module or the actions; files ending in `.inc` (for example `worker-controller.inc`, `options.inc`) are included into a section of the main configuration. Never edit the shipped files outside `local.d/` and `override.d/`; they are replaced on package updates. The service runs as user `groas`, group `grommunio`; everything it must read has to be readable by that account.

Prerequisites

- A working grommunio installation with mail flow, see Quickstart and the post-installation guide.
- Root shell access to the server (all commands below run there).
- A working recursive DNS resolver on the server. Almost every reputation check (SPF, DKIM, DMARC, DNS blocklists, URL lists) is a DNS query. Public resolvers rate-limit blocklist queries; use a local recursive resolver.
- The Rspamd controller password (set during setup). If you do not know it, reset it as described in Antispam: reset the password.
- Outbound UDP port 11335 if you want the public fuzzy-hash feeds to work.

1. Check services and configuration

Start every analysis with a state check. If a service is down or the configuration does not parse, no filter result is meaningful.

```
systemctl is-active grommunio-antispam postfix redis@grommunio
rspamadm configtest
rspamc stat
ss -ltnp | grep -E ':(24|25|587|11332|11333|11334)\b'
postconf -n | grep -Ei 'milter|virtual_transport'
journalctl -u grommunio-antispam --since "30 minutes ago" --no-pager
tail -n 20 /var/log/grommunio-antispam/rspamd.log
```

Expected result:

- all three services report `active`;
- `rspamadm configtest` ends with `syntax OK`;
- `rspamc stat` prints scan counters and the learned counts per statfile (both are low on a fresh system);
- the three Rspamd ports are bound to `127.0.0.1` and `::1` only;
- `postconf -n` shows `smtpd_milters = inet:localhost:11332`, `milter_protocol = 6` and `milter_default_action = accept`.

`milter_default_action = accept` means that Postfix accepts mail unscanned when grommunio-antispam is unreachable (fail-open). Change it to `tempfail` only if you prefer deferred mail over unfiltered mail during an outage.

Warnings in the journal or the log need judgement. DNS failures, unreadable maps or broken local includes are operational errors; missing statistics on a system without mail volume are not.

```
rspamadm configdump -m
rspamadm configdump actions
rspamadm configdump redis
```

`configdump -m` lists every module and whether it is enabled; `configdump

` prints the effective, merged configuration of one section, which is the fastest way to see whether a file in `local.d/` had the intended effect.

2. Open the web interface

The Rspamd controller is not exposed on its own port. The Admin UI's nginx vhost proxies it under the Admin port (`location /antispam/` in `/usr/share/grommunio-admin-common/nginx.d/antispam.conf`):

```
https://mail.example.com:8443/antispam/
```

The same URL is configured as the *Antispam* application link in the Admin UI app launcher (`rspamdWebAddress` in `/etc/grommunio-admin-common/config.json`, see Application links). Log in with the controller password from `/etc/grommunio-antispam/local.d/worker-controller.inc`. To change it, follow Antispam: reset the password and restart `grommunio-antispam`.

The interface has these tabs:

Tab	Use
Status	Uptime, scanned messages, actions distribution, learned counts
Throughput	Rate of scans and actions over time; useful after policy or DNS changes
Configuration	Effective action thresholds and the maps Rspamd knows about
Symbols	All registered symbols with their weights and descriptions
Scan/Learn	Paste a message to scan it or learn it as spam/ham
Test selectors	Check which values a selector extracts from a message before writing rules
History	The most recent scans with sender, recipient, subject, action, score, symbols and scan time

The controller distinguishes a read-only password (`password`) from a privileged one (`enable_password`); learning, editing maps and changing settings from the UI need the privileged level. The appliance sets only `password`, so the same password grants both levels. Connections from the server itself are trusted (`secure_ip = "127.0.0.1"` and `"::1"` in the shipped `worker-controller.inc`), which is why `rspamc` on the server works without a password. Keep the controller reachable from the management network only; do not publish port 11334.

The Admin UI dashboard additionally shows antispam statistics. The Admin API fetches them from the controller (`antispamUrl`, default `http://localhost:11334`, endpoints `stat`, `graph`, `errors`) and the Admin UI displays them when `loadAntispamData` is enabled.

3. Understand actions, scores and symbols

Check the thresholds that are in effect:

```
rspamadm configdump actions
rspamadm configdump -g
```

The shipped defaults are:

Action	Threshold	Effect
`no action`	below 4	Message is accepted
`greylist`	4	Message is deferred with a temporary SMTP error on the first attempt (soft reject, "Try again later"); a retry after 5 minutes is accepted
`add header`	6	Message is accepted and the header `X-Spam: Yes` is added
`reject`	15	Message is rejected with a permanent SMTP error

Further actions exist without thresholds: `soft reject` (used by greylisting and rate limits), `rewrite subject`, `discard` and `quarantine`. Do not change thresholds before you have observed real mail for a while; almost every misclassification is better fixed at the symbol that caused it.

To change a threshold, create a file in `local.d/`:

/etc/grommunio-antispam/local.d/actions.conf

```
reject = 15;
add_header = 6;
greylist = 4;
```

Symbol weights live in group files (`local.d/<module>_group.conf`); the `Symbols` tab shows the effective values. `configdump -g` prints the same on the command line. Always run `rspamadm configtest` after editing and reload the service (the unit defines a reload action that sends `SIGHUP`):

```
rspamadm configtest
systemctl reload grommunio-antispam
```

Headers and Junk folder placement

For the `add header` action the proxy worker sets `X-Spam: Yes` (header name configurable with `spam_header` in `local.d/worker-proxy.inc`). Additional diagnostic headers such as `X-Spamd-Result`, `X-Spam-Status` or `Authentication-Results` are produced by the `milter_headers` module, whose `use` list is empty by default; enable them in `local.d/milter_headers.conf` if you want the verdict visible in every message:

/etc/grommunio-antispam/local.d/milter_headers.conf

```
use = ["x-spamd-result", "x-spam-status", "authentication-results"];
```

By default a tagged message is delivered to the Inbox with the `X-Spam` header; it is then up to the client to filter it. To have `gromox-delivery` file tagged messages into *Junk E-mail* server-side, set `lda_junk_rules` in `/etc/gromox/gromox.cfg` to the header/value pair the filter adds and restart `gromox-delivery`:

/etc/gromox/gromox.cfg

```
lda_junk_rules=X-Spam=Yes
```

Messages filed into Junk E-mail this way are also what the automatic spam learning run (section 6) picks up, so a wrong `add header` verdict would be reinforced. Combine server-side Junk placement with a false-positive review.

4. Scan a first message

Create a harmless synthetic message and scan it with `rspamc symbols`. Read the symbols, not only the action: synthetic messages lack a real `Received` chain and use unresolvable domains, so they score higher than a real message would.

```
cat > /tmp/ham.eml <<'EOF'
From: Alice Example <alice@example.com>
To: Bob Example <bob@example.com>
Subject: Antispam validation test
Date: Mon, 7 Sep 2026 10:00:00 +0200
Message-ID: <ham-001@example.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8

Hello,

this is a harmless internal test message for antispam validation.
EOF

rspamc symbols < /tmp/ham.eml
```

Expected result: an action of `no action` or `add header` with symbols that explain the score, such as missing authentication results (`R_SPF_NA`, `R_DKIM_NA`, `DMARC_NA`) and symbols about the message structure. A test message scored as `add header` does not indicate a broken filter; it indicates that the data does not resemble production mail.

To emulate the SMTP context of a real delivery, pass the connecting IP, HELO, envelope sender and recipient:

```
rspamc --ip 203.0.113.10 --helo mx.partner.example --from alice@partner.example --rcpt bob@example.com symbols < /tmp/ham.eml
```

`rspamc` connects to the local controller; from other hosts, or if the local addresses are removed from `secure_ip`, pass the controller password with `-P <password>`.

5. Test spam detection with GTUBE

GTUBE is a harmless test string that every spam filter must treat as spam. It lets you verify the reject path without storing real spam.

```
cat > /tmp/gtube.eml <<'EOF'
From: Sender Example <sender@example.net>
To: Alice Example <alice@example.com>
Subject: GTUBE function test
Date: Mon, 7 Sep 2026 10:05:00 +0200
Message-ID: <spam-gtube-001@example.net>
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8

This is a safe spam-filter test message.
XJS*C4JDBQADN1.NSBN3*2IDNEN*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL*C.34X
EOF

rspamc symbols < /tmp/gtube.eml
```

Expected result: `Action: reject`, `Score: 15.00 / 15.00` and the symbol `GTUBE`. To test the whole chain including Postfix, send the same body from an external mailbox to a user on the server; the sender should receive a rejection and the message should appear in the History tab with the `GTUBE` symbol.

> **⚠Caution**
>
> Rspamd knows additional GTUBE-like patterns for the other actions. They are disabled by default (`gtube_patterns` in `local.d/options.inc`) and must stay disabled in production, because they allow a sender to force a verdict.

6. Train Bayes in a controlled way

The Bayes classifier (`BAYES_SPAM` / `BAYES_HAM`) is only as good as its training data. Learn unambiguous spam as spam and confirmed false positives as ham. Do not feed unreviewed user reports into it.

```
rspamc learn_spam /tmp/gtube.eml
rspamc learn_ham /tmp/ham.eml
rspamc stat
```

Expected result: the learned counts in `rspamc stat` and on the Status tab increase. The classifier stays silent until it has seen `min_learns` messages (200) in both classes, so a fresh system does not produce `BAYES_*` symbols for a while. This is normal: rules, reputation and authentication checks carry the filter until then.

The appliance ships the classifier with the Redis backend and with `autolearn = true` (`override.d/classifier-bayes.conf`): messages that end in `reject` are learned as spam and messages with a negative score as ham, automatically. Check the effective configuration with:

```
rspamadm configdump classifier
```

Autolearning fills the classifier quickly, but it also means that the classifier learns the mistakes of the other modules. If you see `BAYES_SPAM` on legitimate mail from a sender that was previously rejected for other reasons, learn a few of those messages as ham to correct it.

Automatic spam learning from the Junk folder

The package ships `grommunio-spam-run.service` and `grommunio-spam-run.timer` (daily, `Persistent=true`, randomised start). The service runs `/usr/sbin/grommunio-spam-run.sh`, which iterates over all users from the grommunio database, selects the messages in each user's *Junk E-mail* folder from the mailbox database and learns them as spam with `rspamc learn_spam` (exporting them with `gromox-exm2eml`, an alias of gromox-export(8), where necessary). The timer is installed but disabled by default. Inspect it before enabling it:

```
systemctl cat grommunio-spam-run.service
systemctl cat grommunio-spam-run.timer
systemctl is-enabled grommunio-spam-run.timer
```

Learning is separate from deleting. The unit invokes the script without arguments; only when the script is started with `-d` does it also delete the learned messages with gromox-mbop(8) `delmsg`, which is a hard delete. Do not enable deletion blindly: a message a user filed into Junk

by mistake would be learned as spam and then removed. If you want deletion, create a drop-in that changes `ExecStart` rather than editing the packaged unit.

Enable the timer once the Junk folder workflow is communicated to users and a false-positive process exists:

```
systemctl enable --now grommunio-spam-run.timer
systemctl start grommunio-spam-run.service
journalctl -t grommunio-spam-run --since "10 minutes ago" --no-pager
rspamc stat
```

The script logs through `systemd-cat` with the identifier `grommunio-spam-run`, so filter the journal with `-t`, not only with `-u`. Expected result: one "Learning spam for user ..." line per learned message (none if no user has filed anything into Junk yet) and a growing spam learn counter in `rspamc stat`. A "MySQL-Connection couldn't be established" error means the script could not read `/etc/gromox/mysql_adaptor.cfg` or reach the database.

There is no equivalent automatic ham run in the package. Ham training stays a manual, reviewed step: collect confirmed false positives, learn them with `rspamc learn_ham`, and keep the source messages until the next review.

7. Read history and symbols to diagnose a misclassification

The History tab is the shortest path from a user complaint to the cause. Open the message in question and read sender, recipient, subject, action, score, scan time and the list of symbols with their weights. Only then decide whether DNS, IP/domain reputation, content, an attachment, a URL, Bayes or a local rule was responsible.

On the command line, the scan log contains the same information, one line per message with queue ID, IP, sender, action, scores and symbols; correlate it with Postfix by queue ID:

```
grep '<queue-id>' /var/log/grommunio-antispam/rspamd.log
journalctl -u postfix --since "2 hours ago" --no-pager | grep '<queue-id>'
mailq
```

The `Symbols` tab shows every check that exists and its weight. Use it as the starting point for tuning: if one symbol regularly fires on legitimate mail, check DNS, local maps, authentication of the sender and several real examples before changing its weight. Raising a threshold to hide one symbol lowers the protection for every other message.

Work through misclassifications in this order:

```
Message classified wrongly?
  -> Are SPF, DKIM and DMARC results of the sender correct?
  -> Is the reputation of IP, domain and URLs plausible (RBL/URL symbols)?
  -> Did BAYES_* or NEURAL_* contribute, and was the training data right?
  -> Did a local rule, map or override contribute?
  -> Only then: add a narrow allow/deny entry or exception
```

Allowlisting first hides DNS, authentication, reputation or training problems instead of fixing them.

8. Allow and deny lists with multimap

Exceptions are useful and dangerous at the same time. Keep the rule definition in `local.d/` and the dynamic lists in a separate directory under `/var/lib/grommunio-antispam/`, so that package defaults stay untouched and the lists can be backed up, versioned or maintained by a tool. Rspamd watches map files and reloads them on change; no service reload is required for list edits.

Create the directory, owned by the service account, and the rules:

```
install -d -m 0750 -o groas -g grommunio /var/lib/grommunio-antispam/maps
```

/etc/grommunio-antispam/local.d/multimap.conf

```
ALLOWLIST_SENDER_DOMAIN {
  type = "from";
  filter = "email:domain";
  map = "/var/lib/grommunio-antispam/maps/allowlist_sender_domain.map";
  score = -5.0;
  description = "Allowlisted sender domains reviewed by mail operations";
}

ALLOWLIST_SENDER_ADDRESS {
  type = "from";
  filter = "email";
  map = "/var/lib/grommunio-antispam/maps/allowlist_sender_address.map";
  score = -5.0;
  description = "Allowlisted sender addresses reviewed by mail operations";
}

DENYLIST_SENDER_DOMAIN {
  type = "from";
  filter = "email:domain";
  map = "/var/lib/grommunio-antispam/maps/denylist_sender_domain.map";
  score = 8.0;
  description = "Denylisted sender domains reviewed by mail operations";
}

DENYLIST_SENDER_ADDRESS {
  type = "from";
  filter = "email";
  map = "/var/lib/grommunio-antispam/maps/denylist_sender_address.map";
  score = 8.0;
  description = "Denylisted sender addresses reviewed by mail operations";
}
```

Create the lists (one entry per line), set permissions and activate:

```
printf '%s\n' 'partner.example' > /var/lib/grommunio-antispam/maps/allowlist_sender_domain.map
: > /var/lib/grommunio-antispam/maps/allowlist_sender_address.map
: > /var/lib/grommunio-antispam/maps/denylist_sender_domain.map
: > /var/lib/grommunio-antispam/maps/denylist_sender_address.map
chown groas:grommunio /var/lib/grommunio-antispam/maps/*.map
chmod 0640 /var/lib/grommunio-antispam/maps/*.map
rspamadm configtest
systemctl reload grommunio-antispam
rspamc symbols < /tmp/ham.eml | grep -E 'ALLOWLIST|DENYLIST|Action|Score'
```

Expected result: a message from `partner.example` shows the `ALLOWLIST_SENDER_DOMAIN` symbol with `-5.0`.

Notes on the rule syntax:

- `type = "from"` matches sender addresses; `filter` selects the part (`email` for the full address, `email:domain` for the domain, `email:user` for the local part). Add `extract_from = "smtp";` to match the envelope sender only, otherwise header and envelope are considered.
- Other useful types are `ip` (connecting IP or network, for trusted relays), `rcpt`, `header` and `url`.
- A score adjusts the total; the other checks still run. For an unconditional decision use `prefilter = true;` together with `action = "accept";` or `action = "reject";`, which short-circuits the scan. Reserve this for technical relays whose mail you have verified, never for large freemail or cloud provider domains.
- Verify with the *Test selectors* tab what a rule actually sees (envelope sender, header sender, IP, URL parts) before adding entries.

Every exception needs an owner, a reason, an example message and a review date. After each change: `rspamadm configtest`, reload, scan or send a test message, and check History, score and symbols.

> **Mail fetched from external accounts**
>
> The appliance enables the `external_relay` module with a rule `FETCHMAIL { strategy = "local"; }` (`local.d/external_relay.conf`). For messages retrieved by fetchmail from external mailboxes, reputation checks use the first non-local hop from the `Received` headers instead of the local fetch, so IP-based allow and deny entries apply to the real sending relay.

9. Inbound authentication checks: SPF, DKIM, DMARC and ARC

These modules verify the identity of incoming mail. Outbound signing of your own domains is a separate task, covered in the post-installation guide.

Module	What it checks	Main symbols
`spf`	Whether the sending IP is authorised by the sender domain's SPF record	`R_SPF_ALLOW`, `R_SPF_FAIL`, `R_SPF_SOFTFAIL`, `R_SPF_NEUTRAL`, `R_SPF_NA`, `R_SPF_DNSFAIL`, `R_SPF_PERMFAIL`
`dkim`	Whether the message's DKIM signature validates	`R_DKIM_ALLOW`, `R_DKIM_REJECT`, `R_DKIM_TEMPFAIL`, `R_DKIM_PERMFAIL`, `R_DKIM_NA`
`dmarc`	Whether SPF or DKIM align with the `From:` domain and what the domain's policy requests	`DMARC_POLICY_ALLOW`, `DMARC_POLICY_REJECT`, `DMARC_POLICY_QUARANTINE`, `DMARC_POLICY_SOFTFAIL`, `DMARC_NA`, `DMARC_BAD_POLICY`
`arc`	Whether an ARC chain from forwarders or mailing lists is valid, preserving authentication across intermediaries	`ARC_ALLOW`, `ARC_REJECT`, `ARC_INVALID`, `ARC_NA`, `ARC_DNSFAIL`

```
rspamadm configdump spf
rspamadm configdump dkim
rspamadm configdump dmarc
rspamadm configdump arc
```

By default the DMARC symbols only add to the score. The `dmarc` module can be told to enforce the publisher's policy (`actions` section in `local.d/dmarc.conf`, with `reject` and `quarantine`), and it can generate aggregate reports (`reporting { enabled = true; ... }`, which needs Redis). Enable enforcement only after you have watched the `DMARC_POLICY_*` symbols on real mail for a while.

Many `*_DNSFAIL` and `*_TEMPFAIL` symbols in History point at the resolver, not at the senders. Fix DNS first.

For your own domains, verify the published records from the server and use the DNS health check in the domain view of the Admin UI:

```
dig TXT example.com +short
dig TXT _dmarc.example.com +short
dig TXT dkim._domainkey.example.com +short
```

Replace `dkim` with the selector you generated. Start DMARC for a domain in monitoring mode and tighten it only after the reports show that all legitimate senders, forwarders, newsletter systems and partner relays pass:

```
v=DMARC1; p=none; rua=mailto:dmarc@example.com
v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com
v=DMARC1; p=reject; rua=mailto:dmarc@example.com
```

10. Reputation and content checks: RBL, URL lists, fuzzy hashes, phishing

The `rbl` module queries DNS blocklists for the connecting IP, the `Received` chain, HELO, sender domains and the URLs found in the message (Spamhaus ZEN and DBL, SURBL, URIBL and others). Hits are strong signals but never look at them in isolation: a legitimate service can be compromised, misconfigured or listed temporarily.

```
rspamadm configdump rbl
rspamc symbols < /tmp/gtube.eml | grep -E 'RBL|URIBL|SURBL|DBL|PHISH|Action|Score'
```

The shipped list providers apply fair-use policies; high-volume commercial installations may need a subscription with the list operator. All of these checks depend on a fast local resolver.

The `fuzzy_check` module compares hashes of the message against shared fuzzy storages, by default the public feeds of rspamd.com (`FUZZY_DENIED`, `FUZZY_PROB`, `FUZZY_WHITE`). It needs outbound UDP port 11335:

```
rspamadm configdump fuzzy_check
rspamadm fuzzyping
```

The `phishing` module flags links whose visible text points to a different domain than the actual target (`PHISHED_URL`) and can additionally use the OpenPhish and PhishTank feeds (`openphish_enabled`, `phishtank_enabled`, disabled by default). Exceptions for known redirectors and legitimate domains are configured in `local.d/phishing.conf`; the shipped redirector list is `maps.d/redirectors.inc`.

```
rspamadm configdump phishing
grep -Ei 'fuzzy|phish|surbl|resolve|timeout' /var/log/grommunio-antispam/rspamd.log | tail -n 50
```

Timeouts in this output usually mean a firewall, proxy or resolver problem between the server and the external services.

11. Greylisting and rate limits

Both features create friction on purpose and therefore produce complaints if enabled without monitoring.

Greylisting is active by default. When a message's score reaches the `greylist` threshold (4), the proxy answers with a temporary SMTP error ("Try again later") and records the sender; a retry after the timeout (`timeout = 300` seconds) is accepted, and the record expires after one day (`expire = 86400`). Well-behaved MTAs retry; some notification systems and web applications do not, which is the main source of false positives. Exempt them by IP with `whitelisted_ip` in `local.d/greylist.conf`, or by domain in the file the shipped configuration already references, `/etc/grommunio-antispam/local.d/greylist-whitelist-domains.inc` (one domain per line). To disable greylisting entirely, set `enabled = false;` in `local.d/greylist.conf`.

The `ratelimit` module is loaded but has no rate buckets configured, so it does nothing until you define them in `local.d/ratelimit.conf`. Buckets use the syntax `"<messages> / <period>"`, for example `"10 / 1m"`, and can be keyed by sender IP, sender address or authenticated user. Exceeding a limit yields a soft reject. `postmaster` and `mailer-daemon` recipients are exempt by default (`whitelisted_rcpts`); add your own application senders with `whitelisted_ip` or `whitelisted_user`. A per-user limit also protects your outbound reputation when an account is compromised and used to send spam.

Both modules store their state in Redis and do nothing without a Redis connection.

```
rspamadm configdump greylist
rspamadm configdump ratelimit
rspamc stat | grep -Ei 'greylist|soft reject|reject|add header|no action'
```

Watch the `soft reject` and `greylist` counters and the History tab after tightening either feature.

12. Neural network and training data

The `neural` module trains a small neural network on the symbol vectors of messages that were confidently classified (`NEURAL_SPAM`, `NEURAL_HAM`). On the appliance it is enabled with the default training parameters (`max_trains = 1000` samples per class before the first training run). It needs time and a sufficient volume of correctly classified mail before it contributes anything; on a small installation it may never reach the training threshold, which is harmless.

```
rspamadm configdump neural
rspamc stat
```

Treat Bayes and neural data as operational data. If Redis, the learned statistics or your maps are lost, the filter still works from rules, reputation and authentication, but the site-specific experience is gone and the classifiers start again from zero.

13. Spam gets through: false negatives

When spam is delivered, do not raise global weights or lower thresholds immediately. Work through:

1. Verify that the message passed the filter at all: Postfix must have reached the milter, and the message must appear in History or in `rspamd.log`.
2. Read action, score, symbols and scan time of that entry.
3. Check reputation results: RBL, URL lists and the DNS symbols. `*_DNSFAIL` or missing RBL symbols point to resolver problems.
4. Check whether `BAYES_HAM` or `NEURAL_HAM` pulled the score down and whether similar messages were wrongly learned as ham (autolearn learns ham from negative scores).
5. Check whether a local map, allowlist, prefilter or override reduced the score.
6. Learn the message as spam and re-scan a comparable message.

7. Write a custom rule only when you have repeatable examples and a clear cause.

```
postconf -n | grep -Ei 'milter'
journalctl -u postfix --since "2 hours ago" --no-pager | grep -Ei 'milter|reject|warning'
rspamc symbols < /tmp/suspect.eml
rspamc learn_spam /tmp/suspect.eml
rspamadm configtest
```

Save the suspicious message as a raw `.eml` (including all headers) before scanning; a copy re-sent from a mail client loses the original SMTP context.

14. Anti-virus and external services

grommunio-antispam includes the Rspamd `antivirus` module (ClamAV, Sophos and others) and the `external_services` module (ICAP gateways, oletools for Office macro analysis, DCC, Pyzor, Razor, SpamAssassin, VirusTotal and more). Both are inactive until a service block is configured; the package ships `local.d/antivirus.conf.example` as a starting point. Whether to enable them depends on your operating model, data protection requirements, latency budget, licensing and existing security architecture.

/etc/grommunio-antispam/local.d/antivirus.conf

```
clamav {
  type = "clamav";
  servers = "127.0.0.1:3310";
  symbol = "CLAM_VIRUS";
  action = "reject";
  scan_mime_parts = true;
}
```

Every external check adds latency and a failure mode. Set timeouts consciously, monitor the symbols and the error log, and make sure that a hung scanner cannot block the mail flow:

```
rspamadm configdump antivirus
rspamadm configdump external_services
grep -Ei 'antivirus|clam|external|timeout|error' /var/log/grommunio-antispam/rspamd.log | tail -n 50
```

Verification checklist

Area	Check	Expected result
Packages	`rpm -q grommunio-antispam postfix redis`	All packages installed
Services	`systemctl is-active grommunio-antispam postfix redis@grommunio`	All `active`
Configuration	`rspamadm configtest`	`syntax OK`
Ports	`ss -ltnp | grep -E ':1133[234]\b'`	11332, 11333, 11334 bound to `127.0.0.1` and `::1` only
Postfix integration	`postconf -n | grep smtpd_milters`	`inet:localhost:11332`
Web UI	Open `https://mail.example.com:8443/antispam/`	Login works; Status tab shows counters
Admin UI	Dashboard	Antispam chart shows data
Ham scan	`rspamc symbols < /tmp/ham.eml`	Action and symbols explainable
Spam scan	`rspamc symbols < /tmp/gtube.eml`	`reject`, score 15, symbol `GTUBE`
End to end	Send a GTUBE message from an external mailbox	Rejected at SMTP level; entry in History and `rspamd.log`
Learning	`rspamc learn_spam` / `learn_ham`, then `rspamc stat`	Learned counters increase
Junk placement	Send a message that reaches `add header` to a test user	`X-Spam: Yes` header present; in Junk E-mail if `lda_junk_rules` is set
DNS	`dig TXT example.com`, `_dmarc.example.com`, selector record	SPF, DMARC and DKIM records resolve
Exceptions	Scan a message from an allowlisted domain	`ALLOWLIST_*` symbol present

Area	Check	Expected result
Restart	`systemctl restart grommunio-antispam redis@grommunio postfix`	Services return; learned counts persist
Logs	`grep -Ei 'error`	cannot

Troubleshooting

Symptom	Likely cause	What to check / fix
`rspamadm configtest` reports errors	Syntax error in a file under `local.d/` or `override.d/`	Fix the named file; remember that `.inc` and `.conf` files have different roles
Postfix logs `connect to Milter service inet:localhost:11332: Connection refused`	grommunio-antispam not running or proxy worker not bound	`systemctl status grommunio-antispam`, `ss -ltnp | grep 11332`, journal and `rspamd.log`
Mail is accepted unscanned while the filter is down	`milter_default_action = accept` (shipped default, fail-open)	Decide deliberately between `accept` and `tempfail`
Web UI rejects the password	Controller password unknown or changed	Reset it as in Antispam: reset the password, restart the service
Web UI unreachable under `/antispam/`	Admin nginx vhost not running, or `rspamdWebAddress` not set	`systemctl status nginx grommunio-admin-api`; check `/etc/grommunio-admin-common/config.json`
Admin dashboard shows no antispam data	Admin API cannot reach the controller, or `loadAntispamData` disabled	`antispamUrl` in the Admin API configuration; `journalctl -u grommunio-admin-api`
`rspamc` asks for a password or is denied	Local addresses removed from `secure_ip`	Pass `-P <password>` or restore `secure_ip` in `local.d/worker-controller.inc`
Many `R_SPF_DNSFAIL`, `R_DKIM_TEMPFAIL`, `*_DNSFAIL`, RBL timeouts	Resolver slow, rate-limited (public resolver) or unreachable	Use a local recursive resolver; `dig` from the server; `rspamd.log` for `resolve` / `timeout`
`FUZZY_*` symbols never appear	Outbound UDP 11335 blocked	Firewall; `rspamadm fuzzyping`
`BAYES_*` symbols never appear	Fewer than `min_learns` (200) messages learned per class, or Redis unreachable	`rspamc stat`; `systemctl status redis@grommunio`; `rspamadm configdump redis`
`BAYES_SPAM` on legitimate mail	Autolearn picked up rejects of a sender that is legitimate	Learn several of the messages as ham; check why they were rejected
Redis errors after a package update	`local.d/redis.conf` replaced, leaving `.rpmnew` / `.rpmsave`	Compare and restore the file, see post-update tasks
Legitimate mail is delayed or bounced by the sender's system	Greylisting and a sender that does not retry	History shows `soft reject` / `GREYLIST`; add the sender to `whitelisted_ip` or `greylist-whitelist-domains.inc`
Allowlisted sender still scored	Map not readable by `groas`, wrong `filter`, or header vs envelope mismatch	File owner/permissions; `rspamadm configdump multimap`; *Test selectors* tab; `rspamc symbols` output
Junk folder messages are not learned	`grommunio-spam-run.timer` not enabled, or run failed	`systemctl status grommunio-spam-run.timer`; `journalctl -t grommunio-spam-run`
Tagged spam lands in the Inbox	`lda_junk_rules` unset (default) or does not match `X-Spam=Yes`	Set `lda_junk_rules` in `/etc/gromox/gromox.cfg`, restart `gromox-delivery`; compare with the headers of a delivered message
Spam passes with a low score	Local allowlist, prefilter or wrongly learned ham	Symbols in History; `rspamadm configdump multimap`; re-learn as spam
`/var/log/grommunio-antispam/rspamd.log` grows without bound	No log rotation configured for the file	Add a logrotate rule for the file (rotate, compress, `copytruncate` or reload the service)

Operating notes

Monitoring. Watch more than the reject counter: scan time, Postfix queue length, DNS resolution, Redis state, mail volume, the error log of the controller, free disk space and sudden shifts in the action distribution after policy or DNS changes. The Status and Throughput tabs, the Admin UI dashboard and the scan log together cover this.

```
systemctl is-active grommunio-antispam postfix redis@grommunio
mailq
rspamc stat
df -h /var/log /var/lib
tail -n 200 /var/log/grommunio-antispam/rspamd.log
```

Logs. The scan log is `/var/log/grommunio-antispam/rspamd.log` (`type = "file"`, `level = "info"`, one line per message). The journal of `grommunio-antispam.service` only holds start/stop messages; Postfix logs to the journal. The History tab keeps the most recent scans in Redis (200 rows by default, adjustable in `local.d/history_redis.conf`); it is a diagnostic window, not an archive. Make sure `rspamd.log` is rotated; it grows by several hundred megabytes per month on a busy server. For Gromox-side delivery logging see Debugging messaging services.

Backup. Rspamd stores no mail. Back up:

- `/etc/grommunio-antispam/` (all local configuration; already part of the `/etc/grommunio*` file backup described in Operations);
- `/var/lib/grommunio-antispam/` (DKIM signing keys under `dkim/`, your map files, statistics);
- the Redis snapshot of the `redis@grommunio` instance: `dump.rdb` in the directory named by the `dir` directive of `/etc/redis/grommunio.conf` (`/var/lib/redis/default/` on the appliance). It holds Bayes tokens, history, greylisting and rate-limit state and neural data. Snapshots are written according to the `save` rules of that file, so a file copy is at most a few minutes behind.

After a restore run `rspamadm configtest`, start the services, open the web UI, repeat the GTUBE and ham scans and check that History and `rspamc stat` show the restored state. Mail data itself lives in Gromox, not in Redis; see Backup and restore.

Updates. Package updates may leave `.rpmnew` / `.rpmsave` files under `/etc/grommunio-antispam/local.d/` (notably `redis.conf`). Compare them after each update, then run `rspamadm configtest` and restart the service. Update procedure: Updating grommunio.

High availability. In a cluster, `/var/lib/grommunio-antispam` and the Redis persistence live on the shared volume and `grommunio-antispam` is part of the service resource group; see High availability.

Related pages

- Antispam: reset the web UI password
- Post-installation: DNS, DKIM signing and TLS
- Architecture: mail flow
- Administration: dashboard and application links
- Operations: updates and backup
- High availability
- Backup and restore
- gromox.cfg(5): `lda_junk_rules`
- gromox-mbop(8) and gromox-export(8)
- Debugging messaging services

Single sign-on with grommunio-auth and Keycloak

grommunio-auth adds a central OpenID Connect (OIDC) login to grommunio. It installs a packaged Keycloak instance (`grommunio-keycloak`) behind the appliance's nginx, creates a realm named `grommunio` whose users come straight from the grommunio user database, and configures grommunio Web to redirect browsers to that realm instead of showing its own password form. On top of the realm you can enforce multi-factor authentication (MFA) and broker an existing corporate identity provider (IdP), so that users keep one login for grommunio Web and for the other web applications (Files, Meet, Chat) that reuse the same realm.

This guide installs the packages, walks through the setup wizard, verifies realm, client, federation, tokens and introspection, adds MFA, optionally brokers an upstream IdP, and ends with an end-to-end login and reboot test. For transparent Kerberos logins of domain-joined Windows clients (Outlook) use the Kerberos single sign-on guide instead; both methods can coexist.

How it fits together

Component	Role	Listens on
nginx	Public entry point. Proxies `/auth/` to Keycloak; restricts `/auth/admin` , `/auth/metrics` and `/auth/health` to the networks listed in `/etc/grommunio-common/nginx/auth_allow.conf` .	443/tcp (public)
`grommunio-keycloak.service`	Keycloak, started as user `groauth` with `/etc/grommunio-keycloak/keycloak.conf` . Serves the realm `grommunio` under the relative path `/auth` .	9080/tcp, localhost only
grommunio user-storage provider	Keycloak plugin (`/opt/grommunio-keycloak/providers/grommunio-user-storage.jar`). Reads users from the grommunio database with the read-only credentials in `/etc/grommunio-keycloak/grommunio.properties` and verifies passwords through the PAM service `grommunioauth` (`pam_gromox.so`), so users that authenticate against LDAP in grommunio work as well.	—
MariaDB	Keycloak's own database `grommunio_keycloak` plus `SELECT` access to `grommunio.users` and `grommunio.user_properties` .	3306/tcp, localhost
grommunio Web (php-fpm pool user `groweb`)	Reads `/etc/gromox/keycloak.json` , runs the OIDC authorization-code flow, introspects and refreshes the access token, and logs on to gromox with the token.	unix socket
gromox-zcore	Accepts the token logon; `authmgr` verifies the token signature against `/etc/gromox/bearer_pubkey` , checks the expiry and maps the `email` claim to the mailbox.	`/run/gromox/zcore.sock`
gromox-http (exmdb)	Mailbox store access for zcore and the other daemons.	`[::1]:5000`

The login flow:

1. The browser opens `https://mail.example.com/web/` . Because `/etc/gromox/keycloak.json` exists, grommunio Web sends the browser to the authorization endpoint of the realm `grommunio` with scope `openid` .
2. nginx proxies `/auth/...` to Keycloak on port 9080. Keycloak shows the grommunio login theme, looks the user up through the grommunio user-storage provider and checks the password through PAM; MFA is asked for if it is configured.
3. Keycloak redirects back to `https://mail.example.com/web` with an authorization code. grommunio Web exchanges the code for tokens at the token endpoint (client `grommunio` plus client secret) and introspects the access token.
4. grommunio Web logs on to gromox-zcore with the access token. `authmgr` verifies the RS256 signature with the realm public key in `/etc/gromox/bearer_pubkey` , rejects expired tokens and resolves the `email` claim against the user database.
5. Every 280 seconds grommunio Web refreshes the token, falling back to introspection. If both fail, the browser is sent back to Keycloak.

Files created or used by the setup:

File	Content	Recommended owner / mode
`/etc/grommunio-keycloak/keycloak.conf`	Keycloak server options: database, port, relative path, hostname, proxy headers. Contains the Keycloak database password.	`groauth:gromox` , `0640`
`/etc/grommunio-keycloak/grommunio.properties`	JDBC connection of the user-storage provider to the grommunio database. Contains the read-only database password.	`groauth:gromox` , `0640`
`/etc/gromox/keycloak.json`	OIDC client configuration for grommunio Web: realm, `auth-server-url` , client id (`resource`) and client secret.	`root:groweb` , `0640`
`/etc/gromox/bearer_pubkey`	PEM public key of the realm `grommunio` , used by gromox to verify access tokens. Public data, may stay world-readable.	`root:root` , `0644`
`/etc/grommunio-common/nginx/auth_allow.conf`	`allow` rules for the restricted Keycloak paths.	`root:root` , `0644`
`/usr/share/grommunio-common/nginx/locations.d/grommunio-auth.conf`	Shipped nginx locations for `/auth` ; do not edit, place additions in `/etc/grommunio-common/nginx/locations.d/` .	package file

File	Content	Recommended owner / mode
`/usr/share/grommunio-common/nginx/upstreams.d/grommunio-auth.conf`	Shipped upstream `grommunio_keycloak` (`127.0.0.1:9080`).	package file
`/etc/pam.d/grommunioauth`	PAM stack used by the user-storage provider (`pam_gromox.so`).	package file
`/var/log/grommunio-setup-auth.log`	Log of the setup wizard. Contains the Keycloak admin password in clear text.	`root:root`
`/var/log/nginx/nginx-auth-access.log` , `nginx-auth-error.log`	nginx logs for the `/auth` locations.	nginx

Prerequisites

- A working grommunio installation reachable over HTTPS under its final fully qualified domain name (FQDN), for example `https://mail.example.com` , with valid DNS, a trusted TLS certificate and NTP. The FQDN is written into the Keycloak hostname, the OIDC client URLs and `keycloak.json` ; changing it afterwards means redoing the setup.
- Users exist in grommunio (created locally or imported from LDAP; see Users and LDAP in the administration manual). Keycloak does not store users or passwords of its own for this realm.
- Root shell access, a tested grommunio Admin login and a current backup (see Operations). Single sign-on changes the login path of every web user.
- A terminal for the setup wizard, which is a `dialog` -based text UI.

Record the starting point before you change anything:

```
hostname -f
rpm -q grommunio-web gromox grommunio-admin-api grommunio-common
systemctl is-active nginx php-fpm mariadb gromox-http gromox-zcore
curl -k -I https://mail.example.com/web/
```

Expected result: the FQDN users will type, active services and an HTTP response from grommunio Web. Fix anything that fails before continuing.

1. Install the packages

`grommunio-auth` depends on `grommunio-keycloak` , `grommunio-common` and `grommunio-setup` ; installing it pulls the Keycloak package in:

```
zypper refresh
zypper install grommunio-auth grommunio-keycloak
```

The Keycloak service is installed but not yet configured: `/etc/grommunio-keycloak/keycloak.conf` and `grommunio.properties` are empty until the wizard fills them from the templates in `/usr/share/grommunio-auth/` . Do not start `grommunio-keycloak` by hand at this point.

2. Run the setup wizard

```
/usr/share/grommunio-auth/setup-grommunio-auth.sh
```

> ⚠ **Caution**
>
> The wizard drops and recreates the Keycloak database. Running it on an already configured system removes the realm configuration (clients, MFA settings, identity providers). Use it for the initial setup only.

The wizard asks for:

Prompt	What to enter
FQDN	The public name users type in the browser, for example `mail.example.com` . It is lower-cased and becomes `hostname=` in `keycloak.conf` , the client URLs and the `auth-server-url` . Never enter an internal name or a port.

Prompt	What to enter
Keycloak database	*Create database locally* creates the database and user `grommunio_keycloak` with a random password. *Connect to existing database* expects a prepared database.
Keycloak grommunio database user	The read-only account the user-storage provider uses to read the grommunio user table. The default creates the user `groauth` with a random password and grants `SELECT` on `grommunio.users` and `grommunio.user_properties`.
Keycloak administrator password	Password of the `admin` user in the realm `master`. Accept the generated one or enter your own; it is shown again at the end and written to `/var/log/grommunio-setup-auth.log`.
Restricted grommunio auth paths	Space-separated IP addresses or networks that may reach `/auth/admin`, `/auth/metrics` and `/auth/health`, for example `10.0.0.0/24 127.0.0.1`. Leaving it empty writes `allow 0.0.0.0/0;`, which exposes the admin console to the Internet.

The wizard then, without further questions:

1. writes `/etc/grommunio-keycloak/keycloak.conf` and `/etc/grommunio-keycloak/grommunio.properties`;
2. writes `/etc/grommunio-common/nginx/auth_allow.conf`;
3. starts Keycloak once to create the `admin` user, waits for port 9080 and logs in with `kcadm.sh`;
4. creates the realm `grommunio` (enabled, *Remember me* on, login theme `grommunio`), the user-federation component `grommunio` and the confidential OIDC client `grommunio`;
5. exports the client configuration to `/etc/gromox/keycloak.json` and the realm public key to `/etc/gromox/bearer_pubkey`;
6. enables and starts `grommunio-keycloak.service` and restarts nginx.

Check the result:

```
systemctl is-active grommunio-keycloak nginx php-fpm mariadb
ss -ltnp | grep ':9080'
curl -k -s https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration | jq -r .issuer
```

Expected result: `active` for all services, a Java process listening on port 9080, and the issuer `https://mail.example.com/auth/realms/grommunio`. If port 9080 never opens, read `journalctl -u grommunio-keycloak`; the usual causes are database credentials and a wrong `hostname=`.

3. Work in the realm grommunio, not in master

Open the admin console at `https://mail.example.com/auth/admin/` from an address listed in `auth_allow.conf` and log in as `admin`. The realm `master` exists only to administer Keycloak itself. Everything that concerns grommunio (client, user federation, authentication flows, identity providers) lives in the realm `grommunio`: select it in the realm drop-down at the top left before you change anything. Under *Realm settings* → *General* the realm must be *Enabled*, and *Require SSL* must be *External requests*.

The same checks from the shell, with the bundled admin CLI. Log in once; the session token is stored in `/root/.keycloak/kcadm.config`:

```
/opt/grommunio-keycloak/bin/kcadm.sh config credentials --server http://localhost:9080/auth --realm master --user admin
/opt/grommunio-keycloak/bin/kcadm.sh get realms/grommunio --fields realm,enabled,sslRequired,loginTheme
```

Expected result: `"enabled" : true`, `"sslRequired" : "external"` and `"loginTheme" : "grommunio"`.

4. Check the client grommunio

In the realm `grommunio` open *Clients* → `grommunio`. This is the OIDC application grommunio Web uses. Compare with what the wizard created:

Setting	Expected value
Client ID	`grommunio`
Client authentication	On (confidential client, *Client ID and Secret*)
Root URL, Home URL	`https://mail.example.com:443/`
Valid redirect URIs	`https://mail.example.com/*`
Valid post logout redirect URIs	`https://mail.example.com:443/`
Web origins	`https://mail.example.com:443/`
Standard flow	Enabled (required for the browser login)

Setting	Expected value
Direct access grants	Enabled (password grant; used for the token test in step 9)
Default client scopes	`web-origins`, `acr`, `profile`, `roles`, `email`

`profile` and `email` are required: gromox resolves the mailbox from the `email` claim. On the *Credentials* tab, the client secret must be the value in `credentials.secret` of `/etc/gromox/keycloak.json`. If you regenerate the secret, update the file and restart php-fpm.

From the shell:

```
/opt/grommunio-keycloak/bin/kcadm.sh get clients -r grommunio -q clientId=grommunio --fields id,clientId,enabled,redirectUris,webO
/opt/grommunio-keycloak/bin/kcadm.sh get clients/<CLIENT_UUID>/protocol-mappers/models -r grommunio --fields name,protocolMapper
```

Replace `<CLIENT_UUID>` with the `id` from the first command. Expected result: an enabled, non-public client with the redirect URI above and the three session-note mappers *Client IP Address*, *Client Host* and *Client ID*. Whether an `oidc-audience-mapper` is present depends on the package version; step 10 adds one if token introspection needs it.

5. Check the user federation

Open *User federation* in the realm `grommunio`. The provider `grommunio` must be present and enabled. It makes grommunio users visible to Keycloak without copying them into Keycloak accounts; user administration stays in grommunio Admin (or LDAP). Under *Users*, search for `*` or for an address: your grommunio users appear with their primary e-mail address as username.

```
/opt/grommunio-keycloak/bin/kcadm.sh get components -r grommunio -q type=org.keycloak.storage.UserStorageProvider --fields name,pr
/opt/grommunio-keycloak/bin/kcadm.sh get users -r grommunio -q email=alice@example.com --fields username,email,enabled,federationL
```

Expected result: one component with `"providerId" : "grommunio"` and `"enabled" : [ "true" ]`, and the user record with a `federationLink` pointing to that component. If the user list is empty, check `/etc/grommunio-keycloak/grommunio.properties` and `journalctl -u grommunio-keycloak` for JDBC errors.

6. Enable multi-factor authentication

MFA is configured in the realm `grommunio`, under *Authentication*. Start with individual users, then decide whether to enforce it realm-wide.

Per user: require OTP enrolment

1. In the realm `grommunio` open *Users* and select the user, for example `alice@example.com`.
2. On the *Details* tab add *Configure OTP* to *Required user actions* and save.
3. Let the user log out completely and open `https://mail.example.com/web/` again.
4. After the password, Keycloak shows the TOTP enrolment with a QR code. The user scans it with an authenticator app and confirms the first one-time code. From then on every login asks for the code.

Realm-wide: make OTP mandatory

1. Under *Authentication* → *Flows* duplicate the built-in *browser* flow instead of editing it (for example `browser-otp`).
2. In the copy, set the OTP step (*OTP Form* inside the conditional sub-flow) to *Required*.
3. Bind the copy as the realm's *Browser flow* via *Action* → *Bind flow*.

> **⚠Caution**
>
> Bind a mandatory OTP flow only after you have tested it with one user and you have a documented way back: the `admin` account in the realm `master` is not affected by flows of the realm `grommunio`, and grommunio Web can fall back to its password form when `DISABLE_KEYCLOAK` is defined as `true` in `/etc/grommunio-web/config.php`.

7. Broker an existing identity provider (optional)

If your organisation already runs a central Keycloak or another OIDC provider, connect it as an identity provider of the realm `grommunio`. Users then authenticate upstream, and Keycloak issues the tokens grommunio needs.

1. In the realm `grommunio` open *Identity providers* and add a *Keycloak OpenID Connect* provider (or a generic *OpenID Connect v1.0* provider for other IdPs). Choose a short alias such as `corp-idp`.
2. Enter the upstream discovery URL or its endpoints (authorization, token, logout, user info), the client ID and the client secret of the client you create upstream.
3. Note the *Redirect URI* Keycloak displays, `https://mail.example.com/auth/realms/grommunio/broker/corp-idp/endpoint`, and register exactly this URI on the upstream confidential client.
4. Make sure the e-mail address the upstream IdP asserts equals the primary address of the grommunio user. gromox resolves the mailbox from the `email` claim, so a brokered identity without a matching grommunio user cannot log in.

Redirect the broker logout response

After a logout through the upstream provider, the browser returns to `/auth/realms/grommunio/broker/<alias>/endpoint/logout_response` and may end on a Keycloak page instead of grommunio Web. Add a redirect in the grommunio server block. nginx includes every `*.conf` file in `/etc/grommunio-common/nginx/locations.d/` there:

/etc/grommunio-common/nginx/locations.d/grommunio-auth-broker.conf

```
location ~ ^/auth/realms/grommunio/broker/[^/]+/endpoint/logout_response$ {
    return 302 $scheme://$host/web/;
}
```

```
nginx -t
systemctl reload nginx
curl -k -I https://mail.example.com/auth/realms/grommunio/broker/corp-idp/endpoint/logout_response
```

Expected result: status `302` with `location: https://mail.example.com/web/`. Do not put the block into `keycloak.conf` or into the shipped `grommunio-auth.conf` under `/usr/share`; package updates overwrite the latter.

8. Review the configuration files and permissions

After the wizard, the four files below must agree on the FQDN. The browser URL is `https://mail.example.com/auth/`; the Keycloak process itself listens on `http://localhost:9080/auth` and is only reachable through nginx.

/etc/grommunio-keycloak/keycloak.conf

```
db=mariadb
proxy=edge
http-port=9080
http-relative-path=/auth
http-host=localhost
http-enabled=true
db-username=grommunio_keycloak
db-password=<redacted>
db-url-database=grommunio_keycloak
db-url-host=localhost
hostname=mail.example.com
hostname-strict=false
hostname-strict-https=false
proxy-headers=xforwarded
```

/etc/grommunio-keycloak/grommunio.properties

```
grommunio.db-kind=org.mariadb.jdbc.Driver
grommunio.jdbc-url=jdbc:mariadb://localhost:3306/grommunio
grommunio.username=groauth
grommunio.password=<redacted>
```

/etc/gromox/keycloak.json

```
{
  "realm": "grommunio",
  "auth-server-url": "https://mail.example.com/auth/",
  "ssl-required": "external",
  "resource": "grommunio",
  "verify-token-audience": true,
  "credentials": {
    "secret": "<client-secret>"
  },
  "confidential-port": 0,
  "policy-enforcer": {
    "credentials": {}
  }
}
```

/etc/grommunio-common/nginx/auth_allow.conf

```
allow 10.0.0.0/24;
allow 127.0.0.1;
```

`keycloak.json` is generated by Keycloak's *keycloak-oidc-keycloak-json* installation export; keep the generated keys and only correct `auth-server-url` and `credentials.secret` if they diverge. Optionally add `"redirect-url": "https://mail.example.com/web"` to pin the callback URL grommunio Web sends, which helps when nginx serves several host names.

The wizard writes these files with the default umask, typically world-readable. The Keycloak files contain database passwords, `keycloak.json` contains the client secret; tighten them. grommunio Web runs as `groweb`, Keycloak as `groauth`:

```
chown groauth:gromox /etc/grommunio-keycloak/keycloak.conf /etc/grommunio-keycloak/grommunio.properties
chmod 0640 /etc/grommunio-keycloak/keycloak.conf /etc/grommunio-keycloak/grommunio.properties
chown root:groweb /etc/gromox/keycloak.json
chmod 0640 /etc/gromox/keycloak.json
stat -c '%U:%G %a %n' /etc/gromox/keycloak.json /etc/gromox/bearer_pubkey /etc/grommunio-keycloak/keycloak.conf /etc/grommunio-key
systemctl restart grommunio-keycloak php-fpm
systemctl is-active grommunio-keycloak nginx php-fpm gromox-http gromox-zcore
```

If another PHP component on the same host is configured to read `keycloak.json`, it must stay readable for that component's pool user as well. `bearer_pubkey` holds only the public key and is read by gromox on every token logon; leave it readable.

9. Verify discovery, token issuance and introspection

Check the OIDC foundation independently of the browser:

```
curl -k -s https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration | jq '.issuer,.authorization_endpoint,.
```

Expected result: four URLs below `https://mail.example.com/auth/realms/grommunio/`.

Obtain a token with the password grant (enabled on the client by the wizard) for a test user without MFA:

```
TOKEN_RESPONSE=$(curl -ks -d grant_type=password -d client_id=grommunio -d client_secret='<client-secret>' -d username='alice@exam
echo "$TOKEN_RESPONSE" | jq -r '.token_type,.expires_in'
ACCESS_TOKEN=$(echo "$TOKEN_RESPONSE" | jq -r .access_token)
printf '%s' "$ACCESS_TOKEN" | awk -F. '{print $2}' | base64 -d 2>/dev/null | jq '{azp,aud,email,exp}'
```

Expected result: `Bearer`, a lifetime in seconds, and a payload with `azp` = `grommunio` and the user's `email`. `invalid_client` means the client ID or secret is wrong; `invalid_grant` means a wrong password, a pending required action (for example *Configure OTP*) or an MFA requirement the password grant cannot satisfy.

Introspect the token the way grommunio Web does:

```
curl -k -sS -X POST -u 'grommunio:<client-secret>' -d "token=$ACCESS_TOKEN" https://mail.example.com/auth/realms/grommunio/protocc
```

Expected result: `"active": true`. If the answer is `"active": false` or an error, and `journalctl -u grommunio-keycloak` shows `INTROSPECT_TOKEN_ERROR` with `invalid_token`, continue with step 10.

10. Add an audience mapper for introspection

An access token whose `azp` is `grommunio` does not necessarily list `grommunio` in `aud`. Keycloak then refuses to introspect it for the client `grommunio` and logs `INTROSPECT_TOKEN_ERROR`, even though the browser login looks fine at first. The token must carry the client as audience in the access token and in the introspection response.

In the realm `grommunio` open *Clients* → `grommunio` → *Client scopes* → `grommunio-dedicated` → *Mappers* → *Add mapper* → *By configuration* → *Audience* and create:

```
Mapper type: Audience
Name: grommunio-self-audience
Included Client Audience: grommunio
Add to ID token: Off
Add to access token: On
Add to lightweight access token: Off
Add to token introspection: On
```

Existing tokens do not change. Request a new token as in step 9, confirm that `aud` now contains `grommunio`, and repeat the introspection; it must return `"active": true`. Verify that the mapper sits in the client's dedicated scope:

```
/opt/grommunio-keycloak/bin/kcadm.sh get clients/<CLIENT_UUID>/protocol-mappers/models -r grommunio --fields name,protocolMapper,c
```

11. Check the local loopback and localhost:5000

A common failure that looks like an SSO problem is local: gromox-http exposes the mailbox store (exmdb) on `[::1]:5000`, and `exmdb_hosts_allow` defaults to `::1` (see exmdb_provider(4gx) and gromox(7)). On the appliance, `localhost` resolves to `::1` first. If IPv6 is disabled on `lo`, internal components cannot reach `localhost:5000` and logins fail after Keycloak has already succeeded.

```
getent ahosts localhost
sysctl net.ipv6.conf.all.disable_ipv6 net.ipv6.conf.default.disable_ipv6 net.ipv6.conf.lo.disable_ipv6
ip -brief addr show lo
ss -ltnp | grep ':5000'
journalctl -u gromox-http -u gromox-zcore --since '30 minutes ago' --no-pager
```

Expected result: `::1` among the answers, all three sysctls `0`, `::1/128` on `lo`, and `http` listening on `[::1]:5000`. Disabling IPv6 on the loopback interface is not supported; this has nothing to do with IPv6 on the public interface, which may stay off.

12. End-to-end login test

The acceptance test is the complete user path, not a token:

1. Open `https://mail.example.com/web/` in a fresh private browser window. The grommunio Web login page must hand over to Keycloak within a second.
2. Log in with a grommunio user (and the OTP code if enrolled). Expected result: the browser returns to `https://mail.example.com/web/` and the mailbox opens.
3. Send a test mail to yourself to confirm the MAPI session works.
4. Log out from grommunio Web. Expected result: the Keycloak session ends and the next visit to `/web/` asks for credentials again.
5. Watch `journalctl -u gromox-zcore -f` during the test. A rejected token is logged as `zs_logon_token rejected` with the reason; a successful login produces no such line.

13. Reboot and persistence test

Single sign-on is finished only when it survives a restart without manual steps:

```
systemctl reboot
```

After reconnecting:

```
systemctl is-active grommunio-keycloak nginx php-fpm mariadb gromox-http gromox-zcore
curl -k -I https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration
curl -k -I https://mail.example.com/web/
```

Expected result: all services `active`, status `200` for the discovery document, and a working browser login as in step 12. Keycloak needs some time to start (`TimeoutStartSec=600` in the unit); nginx returns `502` for `/auth/` until it is up.

Verification checklist

Area	Check	Expected result
Installation	`rpm -q grommunio-auth grommunio-keycloak`	Both packages installed
Services	`systemctl is-active grommunio-keycloak nginx php-fpm mariadb gromox-http gromox-zcore`	`active` for all
Network	`ss -ltnp` shows `:9080` (Java, localhost) and `[::1]:5000` (gromox-http); `/auth/admin/` reachable only from `auth_allow.conf` networks	Ports bound; `403` from other networks
Realm	`kcadm.sh get realms/grommunio`	`enabled` true, `sslRequired` external
Client	`kcadm.sh get clients -r grommunio -q clientId=grommunio`	Confidential client, redirect URI `https://mail.example.com/*`, standard flow on
Federation	Users visible under *Users* in the realm `grommunio`	grommunio users listed with `federationLink`
Discovery	`.well-known/openid-configuration`	Issuer `https://mail.example.com/auth/realms/grommunio`
Token	Password grant for a test user	`Bearer` token, `email` claim present
Introspection	`token/introspect` with client credentials	`"active": true`, `aud` contains `grommunio`
Configuration	`stat` on the four configuration files	`keycloak.json` `root:groweb 0640`; Keycloak files `groauth:gromox 0640`
MFA	Login of a user with *Configure OTP*	TOTP enrolment, then OTP prompt on every login
Broker (if used)	Logout through the upstream IdP	`302` to `/web/`, no Keycloak page
End to end	Browser login, mailbox, logout	Complete round trip without errors
Restart	Reboot	Everything above still passes
Logs	`journalctl -u grommunio-keycloak -u gromox-zcore`, `/var/log/nginx/nginx-auth-error.log`	No `INTROSPECT_TOKEN_ERROR`, no `zs_logon_token rejected`, no upstream errors

Troubleshooting

Symptom	Likely cause	What to check / fix
Client, federation or IdP "exists" but grommunio Web does not redirect properly or Keycloak reports an unknown client	Objects were created in the realm `master`	Switch the realm selector to `grommunio`; recreate the objects there
Keycloak shows *Invalid parameter: redirect_uri* (event `invalid_redirect_uri`)	The URL the browser uses, `hostname=` in `keycloak.conf`, the client's *Valid redirect URIs* and `auth-server-url` in `keycloak.json` do not agree	Use one FQDN everywhere; set `redirect-url` in `keycloak.json` if nginx serves several host names
`invalid_client` at the token endpoint, or a login loop back to Keycloak	Wrong client secret in `/etc/gromox/keycloak.json`, or the file is not readable for `groweb` (php-fpm logs `Unable to load webapp Keycloak configuration`)	Copy the secret from the client's *Credentials* tab, set `root:groweb 0640`, restart php-fpm
Login succeeds at Keycloak, grommunio Web shows a login error; `gromox-zcore` logs `zs_logon_token rejected: Token did not validate`	`/etc/gromox/bearer_pubkey` missing, unreadable or stale after a realm key rotation	Regenerate the file from the realm's public key (see Operating notes)
Login succeeds at Keycloak, `zs_logon_token rejected` names the user	The `email` claim does not match a grommunio user (typical for brokered identities)	Align the upstream e-mail attribute with the grommunio primary address

Symptom	Likely cause	What to check / fix
Logout via the upstream IdP ends on a Keycloak page	No redirect for `/auth/realms/grommunio/broker/<alias>/endpoint/logout_response`	Add the location block from step 7 in `/etc/grommunio-common/nginx/locations.d/`, `nginx -t`, reload nginx
`journalctl -u grommunio-keycloak` shows `INTROSPECT_TOKEN_ERROR` / `invalid_token`; introspection returns `active: false`	Access token lacks `grommunio` in `aud`	Add the audience mapper in the `grommunio-dedicated` scope (step 10) and test with a new token
Internal errors around `localhost:5000`; `gromox-zcore` cannot reach exmdb	IPv6 loopback disabled or `localhost` no longer resolves to `::1`	Re-enable IPv6 on `lo`, check `getent ahosts localhost` and `ss -ltnp` for `[::1]:5000` (step 11)
`403 Forbidden` on `https://mail.example.com/auth/admin/`	Your address is not in `/etc/grommunio-common/nginx/auth_allow.conf`	Add an `allow` line, `nginx -t`, reload nginx
Port 9080 never opens; unit restarts	Database credentials or `hostname=` wrong in `keycloak.conf`; MariaDB not running	`journalctl -u grommunio-keycloak`; `systemctl is-active mariadb`; correct the file and restart the service
Passwords are suddenly rejected by Keycloak after a package update, while the grommunio Web password form still works	`pam_gromox.so` inside the running Keycloak process is stale after a Gromox update	`systemctl restart grommunio-keycloak`

Operating notes

- **Back up** `/etc/grommunio-keycloak/`, `/etc/gromox/keycloak.json`, `/etc/gromox/bearer_pubkey`, `/etc/grommunio-common/nginx/` and the database `grommunio_keycloak` together with the regular backup described in Operations. The realm configuration (client, flows, identity providers, OTP enrolments) lives only in that database.
- **Logs**: `journalctl -u grommunio-keycloak` for Keycloak (login events, `INTROSPECT_TOKEN_ERROR`), `/var/log/nginx/nginx-auth-access.log` and `nginx-auth-error.log` for the `/auth` locations, `journalctl -u gromox-zcore` for token logons (`zs_logon_token rejected`), the php-fpm log for grommunio Web's Keycloak client. Delete or protect `/var/log/grommunio-setup-auth.log`; it contains the admin password.
- **Updates**: restart `grommunio-keycloak` after every Gromox update, because the PAM module loaded into the Java process cannot be reloaded (see known issues). Package updates of `grommunio-auth` re-add `http-enabled`, `hostname-strict`, `hostname-strict-https` and `proxy-headers` to `keycloak.conf` if missing; they do not touch your other settings.
- **Key rotation**: if you rotate the RSA key of the realm `grommunio`, regenerate the public key file gromox uses, the same way the wizard does:

```
{ echo "-----BEGIN PUBLIC KEY-----"; curl -s http://localhost:9080/auth/realms/grommunio | jq -r .public_key; echo "-----END
```

- **Fallback**: to return to the classic password form temporarily, define `DISABLE_KEYCLOAK` as `true` in `/etc/grommunio-web/config.php`; removing `/etc/gromox/keycloak.json` has the same effect permanently.
- **Login page**: with Keycloak enabled the login page is rendered by Keycloak, so the grommunio Web disclaimer described in Disclaimer does not apply; use a Keycloak theme instead.
- **Multi-server setups**: OIDC sessions must stay on one home server; see the OIDC note in the architecture manual for the HAProxy cookie approach.

Related pages

- Kerberos single sign-on (SPNEGO) for domain-joined Windows clients
- Administration manual — Users, LDAP
- Architecture — authentication
- authmgr(4gx), exmdb_provider(4gx)
- grommunio Chat guide, grommunio Meet guide, grommunio Files and Office guide — reuse this realm for their own OIDC clients
- Operations, Troubleshooting

grommunio Meet: video conferencing

grommunio Meet is the video conferencing component of grommunio. It is built on the Jitsi stack (Prosody, Jicofo, Jitsi Videobridge and the Jitsi Meet web application) and is served under `https://mail.example.com/meet/` by the same nginx that serves grommunio Web. Users start meetings from grommunio Web or from the calendar, or simply share a room URL.

This guide installs Meet on the openSUSE-based grommunio appliance in three phases. Phase A makes Meet work without any single sign-on: packages, `grommunio-meet-setup`, generated configuration, nginx route and browser tests. Phase B optionally protects rooms with grommunio-auth/Keycloak through a third-party OIDC adapter. Phase C connects grommunio Web and the calendar workflow. Finish Phase A completely, including the reboot test, before you start Phase B; SSO problems are much easier to isolate when the plain conference path is known to work.

For the user-facing side (enabling the plugin, the Meet tab, adding a Meet link to an appointment) see Meet in grommunio Web. For camera, microphone and browser permission problems on client devices see Troubleshooting Meet, Audio/Video.

How it fits together

Component	Package / unit	Role	Listens on
nginx	`grommunio-common` include `grommunio-meet.conf` (shipped by `jitsi-meet`)	Serves the web app under `/meet/`, proxies signalling and the bridge channel	443/tcp
Prosody	`prosody`, `prosody.service`	XMPP server: rooms (MUC), lobby, breakout rooms, authentication	127.0.0.1:5280 (BOSH and WebSocket, plain HTTP behind nginx), 5222/tcp (XMPP, used locally by Jicofo and the bridge)
Jicofo	`jitsi-jicofo`, `jitsi-jicofo.service`	Conference focus: joins each room, picks a bridge, negotiates sessions with participants	XMPP client only
Jitsi Videobridge (JVB)	`jitsi-videobridge`, `jitsi-videobridge.service`	Selective forwarding unit: receives and routes all audio/video	10000/udp (media), 9090/tcp (bridge WebSocket, proxied as `/colibri-ws`), 127.0.0.1:8081 (private REST API, health)
Meet web app	`jitsi-meet`, `jitsi-meet-branding-grommunio`, `jitsi-meet-prosody-plugins`	Static frontend in `/srv/jitsi-meet`, client configuration `config.js`, Prosody modules in `/usr/share/jitsi/meet/prosody-plugins/`	via nginx
STUN/TURN	external: the managed relay `turn.grommun.io` (default) or your own coturn	Lets clients behind restrictive firewalls reach the bridge	outbound 3478/udp and 443/tcp

A browser loads `https://mail.example.com/meet/<room>`, which nginx serves from `/srv/jitsi-meet`. The web app then opens an XMPP connection over `/meet/xmpp-websocket` (fallback: BOSH over `/meet/http-bind`), which nginx forwards to Prosody on `127.0.0.1:5280`. Prosody hosts the room as a MUC on `conference.mail.example.com`; Jicofo (`focus@auth.mail.example.com`) joins the room, selects a videobridge from the `JvbBrewery@internal.auth.mail.example.com` MUC and sets up the media sessions. Media flows directly between each participant and the videobridge over UDP 10000; the bridge also opens a control WebSocket (`/colibri-ws/`) through nginx to port 9090. Clients that cannot send UDP fall back to the TURN relay on 443/tcp, which forwards to the bridge's public UDP 10000, so that port must be reachable in every case.

Network requirements

Direction	Port	Purpose
Inbound to the appliance	443/tcp	Web app, XMPP signalling, bridge channel (shared with grommunio Web)
Inbound to the appliance	10000/udp	Videobridge media for all participants; behind NAT forward `<public-ip>:10000/udp` to the appliance
Outbound from the appliance	3478/udp, 443/tcp	STUN/TURN relay (`turn.grommun.io` unless you configure your own)
Local only	5222, 5280, 5281, 8081, 9090	Prosody, JVB REST and WebSocket; do not expose these

The videobridge is a selective forwarding unit: all participants share the single UDP port, so a 50-person meeting needs the same ports as a two-person one. The packaged file `/usr/share/doc/packages/jitsi-meet/grommunio-meet-network.md` summarises these requirements and contains an external reachability check (see step 5).

Files you will work with

Path	Content
`/usr/sbin/grommunio-meet-setup`	Configures Prosody, Jicofo and the videobridge for the appliance; safe to re-run
`/etc/prosody/prosody.cfg.lua`, `/etc/prosody/conf.d/mail.example.com.cfg.lua`	Prosody global and host configuration (generated)
`/etc/prosody/certs/`	Prosody TLS certificates (Let's Encrypt copy or self-signed)
`/etc/jitsi/jicofo/jicofo.conf`, `/etc/jitsi/jicofo/config`	Jicofo HOCON configuration and Java system properties
`/etc/jitsi/videobridge/jvb.conf`, `/etc/jitsi/videobridge/sip-communicator.properties`	Videobridge configuration and NAT address mapping
`/etc/systemd/system/jitsi-videobridge.service.d/10-grommunio-config.conf`	Drop-in that makes the bridge load `jvb.conf`
`/etc/jitsi/.meet-secrets`	Generated XMPP passwords and bridge nickname (root only)
`/srv/jitsi-meet/config.js`	Client configuration served to browsers
`/usr/share/grommunio-common/nginx/locations.d/grommunio-meet.conf`	Packaged nginx locations for `/meet/` and `/colibri-ws/`
`/etc/grommunio-common/nginx/locations.d/`	Directory for your own additional nginx locations (used in Phase B)
`/etc/grommunio-web/config-meet.php`	grommunio Web Meet plugin defaults (Phase C)

Prerequisites

- A working grommunio appliance with grommunio Web reachable at `https://mail.example.com/web/` (see Installation and Quickstart).
- `hostname -f` returns the public name users will type (`mail.example.com`), and the TLS certificate covers that name. `grommunio-meet-setup` reuses a Let's Encrypt certificate from `/etc/letsencrypt/live/mail.example.com/` when present.
- UDP 10000 is reachable from the client networks (open it on perimeter and cloud firewalls, forward it if the appliance is behind NAT).
- Outbound 3478/udp and 443/tcp to the TURN relay, or your own coturn.
- For Phase B: a working grommunio-auth/Keycloak installation, see Single sign-on with Keycloak.
- Root shell access to the appliance.

1. Install the Meet packages

Meet is an optional role of `grommunio-setup`. If you did not select it during the initial setup, either re-run `grommunio-setup` and add the role `meet`, or install the packages directly:

```
zypper install jitsi-meet jitsi-jicofo jitsi-videobridge jitsi-meet-prosody-plugins jitsi-meet-branding-grommunio prosody
rpm -q jitsi-meet jitsi-jicofo jitsi-videobridge jitsi-meet-prosody-plugins jitsi-meet-branding-grommunio prosody
```

The `jitsi-meet` package installs the web application into `/srv/jitsi-meet`, the nginx include `/usr/share/grommunio-common/nginx/locations.d/grommunio-meet.conf`, the `grommunio-meet-setup` tool and the network requirements document. Its post-install script writes the host name into `/srv/jitsi-meet/config.js` and pins the `meet/` sub-directory used for the BOSH and WebSocket URLs. `jitsi-meet-branding-grommunio` adds the grommunio look (`index.html`, `custom.css`, `interface_config.js`, images).

2. Run grommunio-meet-setup

`grommunio-meet-setup` generates all service configuration for the FQDN you pass (default: `hostname -f`). It is idempotent: secrets are kept in `/etc/jitsi/.meet-secrets` and reused on every run, so you can re-run it after changing the host name, the public IP or the TURN settings.

```
grommunio-meet-setup mail.example.com
```

The tool performs these steps:

1. Writes the Prosody host configuration `/etc/prosody/conf.d/mail.example.com.cfg.lua` (virtual hosts, MUC components, lobby and breakout rooms, TURN `external_services`) and makes sure `/etc/prosody/prosody.cfg.lua` includes `conf.d/*.cfg.lua`.
2. Creates TLS certificates in `/etc/prosody/certs/` (copies Let's Encrypt files when present, otherwise self-signed) and adds the self-signed ones to the system trust store as `/etc/pki/trust/anchors/prosody-*.crt`.
3. Registers the XMPP accounts `focus@auth.mail.example.com` (Jicofo) and `jvb@auth.mail.example.com` (videobridge) with generated passwords.
4. Writes `/etc/jitsi/jicofo/jicofo.conf` and `/etc/jitsi/jicofo/config`.

5. Fills the bridge password and a unique bridge nickname into the packaged `/etc/jitsi/videobridge/jvb.conf` and installs the systemd drop-in that passes `-Dconfig.file=/etc/jitsi/videobridge/jvb.conf` to the bridge.
6. Detects the public IP via STUN and, if it differs from the local address, writes the NAT mapping into `/etc/jitsi/videobridge/sip-communicator.properties`.
7. Replaces the domain in `/srv/jitsi-meet/config.js`.
8. Opens `10000/udp` in firewalld when firewalld is active.
9. Enables and restarts `prosody`, `jitsi-videobridge`, `jitsi-jicofo` and reloads nginx.

Environment variables override the detection:

Variable	Effect
`JVB_PUBLIC_ADDRESS=203.0.113.10`	Advertise this public IP for media instead of the STUN-discovered one
`TURN_HOST=turn.example.com`	Use your own STUN/TURN server instead of the managed relay
`TURN_SECRET=<strong-password>`	Shared secret of your own coturn (`external_service_secret`)

Example for an appliance behind NAT with its own coturn:

```
JVB_PUBLIC_ADDRESS=203.0.113.10 TURN_HOST=turn.example.com TURN_SECRET=<strong-password> grommunio-meet-setup mail.example.com
```

Expected result: the last lines print `grommunio-meet-setup: done. Meet is at https://mail.example.com/meet` followed by the network summary. Then check the services:

```
systemctl is-active prosody jitsi-jicofo jitsi-videobridge nginx
```

All four must report `active` before you continue.

> **⚠Re-runs restart the Meet services**
>
> `grommunio-meet-setup` restarts Prosody, the videobridge and Jicofo, which ends running conferences. Run it in a maintenance window on a production system.

3. Review the generated configuration

You normally do not edit these files by hand; knowing their structure is what makes troubleshooting possible. Values shown as `<...>` are placeholders for the generated secrets, which must stay on the appliance.

3.1 Prosody

`/etc/prosody/conf.d/mail.example.com.cfg.lua` defines:

- `VirtualHost "mail.example.com"` with `authentication = "anonymous"`, the modules `bosh`, `websocket`, `external_services`, `ping`, `conference_duration`, `muc_lobby_rooms`, `muc_breakout_rooms` (plus `smacks` when the Prosody version provides it), and `main_muc = "conference.mail.example.com"`, `lobby_muc = "lobby.mail.example.com"`, `breakout_rooms_muc = "breakout.mail.example.com"`.
- `VirtualHost "auth.mail.example.com"` with `authentication = "internal_hashed"`, the login domain of `focus` and `jvb`.
- MUC components `conference.`, `breakout.`, `lobby.` and `internal.auth.mail.example.com` (the bridge brewery), plus the `focus.mail.example.com` `client_proxy` component and the helper components for speaker stats, A/V moderation, end conference and room metadata.
- `plugin_paths = { "/usr/share/jitsi/meet/prosody-plugins" }`.
- `external_services` with the STUN (3478/udp), TURN (3478/udp) and TURNS (443/tcp) entries of the relay, authenticated with `external_service_secret`.
- `cross_domain_websocket = { "https://mail.example.com" }`. Prosody serves plain HTTP on 5280 behind nginx; this entry allows the HTTPS origin of the browser. It must match the host name users open.

Anonymous authentication means anyone who knows a room URL can create and join that room. Rooms are protected by their names (and, optionally, by lobby or room password set by a moderator). Phase B replaces this with token authentication.

3.2 Jicofo

`/etc/jitsi/jicofo/jicofo.conf` (HOCON):

/etc/jitsi/jicofo/jicofo.conf

```
jicofo {
  xmpp {
    client {
      client-proxy = "focus.mail.example.com"
      xmpp-domain = "mail.example.com"
      domain = "auth.mail.example.com"
      username = "focus"
      password = "<focus-password>"
      disable-certificate-verification = true
    }
  }
  bridge { brewery-jid = "JvbBrewery@internal.auth.mail.example.com" }
}
```

The unit `jitsi-jicofo.service` reads `EnvironmentFile=-/etc/jitsi/jicofo/config` (Java system properties, log directory `/var/log/jitsi`) and starts `/usr/share/jitsi/jicofo/jicofo.sh`, which adds `-Dconfig.file=/etc/jitsi/jicofo/jicofo.conf` automatically when that file exists. If `jicofo.conf` is missing, Jicofo exits with "To run jicofo you need a configuration file". The legacy environment file `/etc/jitsi/jicofo/jitsi-jicofo.conf` from older setups is not read by the current unit.

3.3 Videobridge

`/etc/jitsi/videobridge/jvb.conf` (HOCON) is shipped by the package; the setup tool only fills in the password and nickname:

/etc/jitsi/videobridge/jvb.conf

```
videobridge {
  http-servers {
    public {
      host = 0.0.0.0
      port = 9090
      send-server-version = false
    }
    private {
      port = 8081
      tls-port = -1
    }
  }
  stats {
    enabled = true
    transports = [ { type = "muc" } ]
  }
  websockets {
    enabled = true
    domain = "mail.example.com:443"
    tls = true
  }
  apis.xmpp-client.configs {
    shard {
      hostname = "localhost"
      domain = "auth.mail.example.com"
      username = "jvb"
      password = "<jvb-password>"
      muc_jids = "JvbBrewery@internal.auth.mail.example.com"
      muc_nickname = "<unique-uuid>"
      disable-certificate-verification = true
    }
  }
}
ice4j {
  harvest {
    mapping {
      aws.enabled = false
    }
  }
}
```

Points worth knowing:

- The private REST port is 8081, not the Jitsi default 8080, because 8080 is already used by nginx for grommunio Web on the appliance. The health check is `http://127.0.0.1:8081/about/health`.
- `websockets.domain` must be the host name and port the browsers use; nginx forwards `/colibri-ws/` and `/meet/colibri-ws/` to port 9090.
- `muc_jids` must equal Jicofo's `bridge.brewery-jid`, and `muc_nickname` must be unique per bridge.
- `/usr/share/jitsi/videobridge/jvb.sh` does not add `-Dconfig.file` by itself, so the drop-in `/etc/systemd/system/jitsi-videobridge.service.d/10-grommunio-config.conf` with `Environment=JAVA_SYS_PROPS=-Dconfig.file=/etc/jitsi/videobridge/jvb.conf` is required. Without it the bridge falls back to defaults (port 8080, no brewery) and Jicofo never finds a bridge. Check with:

```
systemctl show -p Environment jitsi-videobridge
```

Expected result: the output contains `-Dconfig.file=/etc/jitsi/videobridge/jvb.conf`.

Public address behind NAT. When the STUN-discovered public IP differs from the local one, the setup tool writes `org.ice4j.ice.harvest.NAT_HARVESTER_LOCAL_ADDRESS` and `org.ice4j.ice.harvest.NAT_HARVESTER_PUBLIC_ADDRESS` to `/etc/jitsi/videobridge/sip-communicator.properties`. Re-run the tool with `JVB_PUBLIC_ADDRESS=<public-ip>` if the detection is wrong. If your environment translates the port as well (for example a lab that maps `<public-ip>:11001/udp` to the appliance's 10000/udp), use an explicit static mapping in `jvb.conf` instead:

/etc/jitsi/videobridge/jvb.conf (excerpt)

```
ice4j {
  harvest {
    mapping {
      aws.enabled = false
      stun.enabled = false
      static-mappings = [
        {
          local-address = "10.0.0.15"
          public-address = "203.0.113.10"
          local-port = 10000
          public-port = 11001
        }
      ]
    }
  }
}
```

In a lab whose appliance only has private (RFC 1918) addresses, the bridge health check fails by default; set `videobridge.health.require-valid-address = false` in `jvb.conf` for such test environments only.

3.4 Client configuration

`/srv/jitsi-meet/config.js` is served to every browser. The relevant keys:

/srv/jitsi-meet/config.js (excerpt)

```
var subdir = 'meet/';
var config = {
    hosts: {
        domain: 'mail.example.com',
        focus: 'focus.mail.example.com',
        muc: 'conference.mail.example.com'
    },
    bosh: '//mail.example.com/meet/http-bind',
    websocket: 'wss://mail.example.com/meet/xmpp-websocket',
    requireDisplayName: true,
    enableWelcomePage: true,
    prejoinConfig: {
        enabled: true,
    },
    p2p: {
        enabled: true,
        stunServers: [
        ]
    },
};
```

`hosts.domain`, `bosh` and `websocket` must use exactly the host name users open in the browser; otherwise the XMPP connection fails while the page itself still loads. `config.js` is an rpm configuration file, so package updates leave your version in place and write `config.js.rpmnew` next to it.

4. Verify the nginx route and the client files

The `/meet/` locations come from the packaged include, which nginx loads through `/usr/share/grommunio-common/nginx.conf`. After installing the packages nginx must have been reloaded (the setup tool does this).

```
nginx -t
nginx -T | grep -n "grommunio-meet.conf"
curl -kI https://mail.example.com/meet/
curl -kI https://mail.example.com/meet/config.js
curl -kI https://mail.example.com/meet/external_api.js
curl -sS -i http://127.0.0.1:8081/about/health
ss -lunp | grep ':10000 '
```

Expected results: `nginx -t` reports the configuration as valid; `nginx -T` lists the include; the three `curl -kI` calls return `HTTP/2 200` (or `HTTP/1.1 200 OK`); the health check returns `HTTP/1.1 200 OK` with an empty body; `ss` shows Java sockets bound to UDP 10000.

The include also proxies `/meet/http-bind` and `/meet/xmpp-websocket` to `127.0.0.1:5280` and `/colibri-ws/` to `127.0.0.1:9090`, and logs to `/var/log/nginx/nginx-meet-access.log` and `/var/log/nginx/nginx-meet-error.log`.

5. Test in the browser

Run the tests from a client network, not from the appliance itself.

1. **Single participant.** In a fresh browser profile open `https://mail.example.com/meet/standalone-test-room`. Expected result: the grommunio-branded pre-join screen appears, camera and microphone previews work, a display name is requested, and after joining you are the only participant in the room. In the browser's developer tools (Network tab) the WebSocket to `/meet/xmpp-websocket` is open.
2. **Two participants.** Join the same room from a second browser profile or device. Expected result: both participants see and hear each other; the connection indicator shows a bridge connection (not only P2P) once a third participant joins.
3. **Three participants.** Join from a third context. Expected result: all three tiles show live video; the participant list is stable.
4. **Stability.** Keep the session running for at least 120 seconds. Expected result: nobody is dropped and audio/video do not stall.
5. **Media path from outside.** From a machine outside your network, confirm that UDP 10000 answers STUN (`203.0.113.10` is the public IP of the appliance):

```
python3 - <<'PY'
import socket, struct, os
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.settimeout(4)
s.sendto(struct.pack(">HHI12s", 1, 0, 0x2112A442, os.urandom(12)), ("203.0.113.10", 10000))
print("reachable:", len(s.recvfrom(1024)[0]), "bytes")
PY
```

Expected result: `reachable: <n> bytes`. A timeout means UDP 10000 is blocked or not forwarded.

6. **Reboot.** Reboot the appliance and repeat the service check from step 2, the checks from step 4 and the single-participant test. Expected result: identical results without manual intervention.

Phase A is complete when all six tests pass.

6. Optional: single sign-on for protected rooms

By default anyone who knows a room URL can join. To require a grommunio login, Meet must be switched to token (JWT) authentication: Prosody and the web app accept a signed token, and something has to turn a Keycloak login into such a token. Jitsi does not ship that piece.

> **Third-party component**
>
> This section uses the **Jitsi Keycloak Adapter v2** by Nordeck (https://github.com/nordeck/jitsi-keycloak-adapter-v2), a small Deno service that is not part of grommunio and not covered by grommunio support. Its authors point to `jitsi-oidc-adapter` (jitsi-contrib) as the successor project; other adapters exist. Treat the adapter installation as your own integration, pin a release, and review its source before deploying it. Everything in 6.3 that touches Prosody and `config.js` is standard Jitsi token authentication and applies to any adapter.

The flow is: browser opens a room, Meet redirects to the adapter's `/oidc/auth`, the adapter redirects to Keycloak (realm `grommunio`), the user logs in (optionally with MFA), Keycloak redirects back to the adapter's `/oidc/tokenize`, the adapter mints a Jitsi JWT (`app_id` / `app_secret` shared with Prosody) and sends the browser back into the room with `?jwt=...`.

The adapter builds its own Keycloak redirect URI as `https://<Host header>/oidc/tokenize` and serves only `/oidc/auth`, `/oidc/tokenize` and `/oidc/health`. Publish these paths under `/oidc/` on the grommunio host exactly as shown; do not move them under `/meet/`.

6.1 Create the Keycloak client

In the Keycloak admin console of grommunio-auth, in the realm `grommunio` (not `master`), create an OpenID Connect client:

```
Client ID:              grommunio-meet
Client authentication:  On
Standard flow:          On
Valid redirect URIs:    https://mail.example.com/oidc/tokenize
Web origins:            https://mail.example.com
```

Copy the client secret from the client's *Credentials* tab; it becomes `KEYCLOAK_CLIENT_SECRET` below. (The adapter also supports a public client with client authentication off and an empty secret.)

Check that the realm's discovery document is reachable from the appliance:

```
curl -sS https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration | head -c 300
```

Expected result: JSON starting with `{"issuer":"https://mail.example.com/auth/realms/grommunio", ...`.

6.2 Install the adapter as a system service

The adapter needs the Deno runtime, which is not packaged for the appliance; install it from the upstream release archive into `/usr/local/bin` (see the adapter's `docs/setup-standalone.md`). Then install the adapter under its own unprivileged user with a root-owned environment file. Replace `<pinned-release-or-commit>` with the tag you reviewed.

```
deno --version
install -d -o root -g root -m 0755 /opt/jitsi-keycloak-adapter-v2
git clone https://github.com/nordeck/jitsi-keycloak-adapter-v2.git /opt/jitsi-keycloak-adapter-v2
git -C /opt/jitsi-keycloak-adapter-v2 checkout <pinned-release-or-commit>
groupadd --system jitsi-oidc-adapter
useradd --system --gid jitsi-oidc-adapter --home-dir /var/lib/jitsi-oidc-adapter --shell /usr/sbin/nologin jitsi-oidc-adapter
install -d -o jitsi-oidc-adapter -g jitsi-oidc-adapter -m 0750 /var/lib/jitsi-oidc-adapter
install -d -o root -g jitsi-oidc-adapter -m 0750 /etc/jitsi/meet/oidc-adapter
```

Generate a long random string for `JWT_APP_SECRET` (for example `openssl rand -hex 32`); it is shared between the adapter and Prosody only.

/etc/jitsi/meet/oidc-adapter/environment

```
KEYCLOAK_ORIGIN=https://mail.example.com/auth
KEYCLOAK_ORIGIN_INTERNAL=https://mail.example.com/auth
KEYCLOAK_REALM=grommunio
KEYCLOAK_CLIENT_ID=grommunio-meet
KEYCLOAK_CLIENT_SECRET=<keycloak-client-secret>
KEYCLOAK_MODE=query
JWT_ALG=HS256
JWT_HASH=SHA-256
JWT_APP_ID=grommunio-meet
JWT_APP_SECRET=<jitsi-jwt-app-secret>
JWT_EXP_SECOND=3600
HOSTNAME=127.0.0.1
PORT=9000
```

`KEYCLOAK_ORIGIN` is the base under which grommunio-auth publishes Keycloak; the adapter appends `/realms/<realm>/protocol/openid-connect/...`, so give it without a trailing slash. `KEYCLOAK_ORIGIN_INTERNAL` may point to an internal address if the public one is not reachable from the appliance.

```
chown root:jitsi-oidc-adapter /etc/jitsi/meet/oidc-adapter/environment
chmod 0640 /etc/jitsi/meet/oidc-adapter/environment
```

/etc/systemd/system/oidc-adapter.service

```
[Unit]
Description=grommunio Meet Keycloak OIDC adapter (Nordeck v2)
Documentation=https://github.com/nordeck/jitsi-keycloak-adapter-v2
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
User=jitsi-oidc-adapter
Group=jitsi-oidc-adapter
WorkingDirectory=/opt/jitsi-keycloak-adapter-v2
EnvironmentFile=/etc/jitsi/meet/oidc-adapter/environment
ExecStart=/usr/local/bin/deno run --allow-net --allow-env /opt/jitsi-keycloak-adapter-v2/src/adapter.ts
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
systemctl daemon-reload
systemctl enable --now oidc-adapter.service
systemctl is-active oidc-adapter.service
curl -sS -i http://127.0.0.1:9000/oidc/health
journalctl -u oidc-adapter -n 20 --no-pager
```

Expected result: the service is `active`, the health call returns `HTTP/1.1 200 OK` with body `healthy`, and the journal shows the adapter's start-up summary of its settings (secrets masked). The adapter only needs network access to Keycloak; if grommunio-auth uses a certificate the appliance does not trust, fix the trust store rather than disabling certificate checks.

6.3 Wire nginx, the web app and Prosody

nginx. Add a location for the adapter in the admin drop-in directory, which the grommunio nginx configuration includes alongside the packaged locations:

/etc/grommunio-common/nginx/locations.d/meet-oidc.conf

```
location ~ ^/oidc/ {
    proxy_pass http://127.0.0.1:9000;
    proxy_http_version 1.1;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header Host $http_host;
}
```

Web app. Point Meet at the adapter by appending to `/srv/jitsi-meet/config.js` (after the `var config = {...};` block):

/srv/jitsi-meet/config.js (append)

```
config.tokenAuthUrl = 'https://mail.example.com/oidc/auth?state={state}';
config.tokenAuthUrlAutoRedirect = true;
```

`{state}` is filled in by the web app with the room name, tenant and client type; the adapter uses it to send the user back to the right room. `tokenAuthUrlAutoRedirect` (from the adapter's documentation) makes the redirect happen without an extra login button; if your Meet version ignores it, users get a login button on the pre-join screen instead.

Prosody. In `/etc/prosody/conf.d/mail.example.com.cfg.lua` switch the main virtual host from anonymous to token authentication and enable token verification on the conference MUC. The `app_id` / `app_secret` pair must be identical to `JWT_APP_ID` / `JWT_APP_SECRET` of the adapter.

/etc/prosody/conf.d/mail.example.com.cfg.lua (changes)

```
VirtualHost "mail.example.com"
    authentication = "token"
    app_id = "grommunio-meet"
    app_secret = "<jitsi-jwt-app-secret>"
    allow_empty_token = false
    asap_accepted_issuers = { "grommunio-meet" }
    asap_accepted_audiences = { "grommunio-meet" }
    -- keep ssl, modules_enabled, main_muc, lobby_muc, breakout_rooms_muc as generated

Component "conference.mail.example.com" "muc"
    -- add to the existing modules_enabled list:
    modules_enabled = { "muc_hide_all"; "muc_meeting_id"; "muc_domain_mapper"; "muc_rate_limit"; "muc_password_whitelist"; "token_
```

The modules `mod_auth_token.lua` and `mod_token_verification.lua` are part of `jitsi-meet-prosody-plugins` and already on the plugin path.

Apply everything:

```
prosodyctl check config
nginx -t
systemctl restart prosody jitsi-jicofo jitsi-videobridge
systemctl reload nginx
systemctl is-active prosody jitsi-jicofo jitsi-videobridge oidc-adapter nginx
curl -kI https://mail.example.com/meet/
curl -ks https://mail.example.com/meet/config.js | grep -n tokenAuth
```

Expected result: all services active, `/meet/` still answers 200, and the delivered `config.js` contains both `tokenAuth` lines.

> ⚠ **Re-running grommunio-meet-setup reverts the Prosody changes**
>
> `grommunio-meet-setup` rewrites `/etc/prosody/conf.d/mail.example.com.cfg.lua` with `authentication = "anonymous"`. After any re-run, re-apply the token settings above. The nginx drop-in, the adapter service and the `config.js` additions are not touched by the tool, but a `jitsi-meet` package update may ship a new `config.js.rpmnew` to merge.

The adapter issues tokens with `iss` and `aud` set to `JWT_APP_ID`, `room` set to the room the user opened, and `sub` set to the Jitsi tenant (the path segment before the room name) or, without a tenant, to the host name.

> **Meet under /meet/ and the token tenant**
>
> Because Meet is served under `/meet/`, the web app reports the tenant `meet` and the adapter writes `sub = "meet"` into the token. Prosody's token module verifies that claim against the room's domain by default (`enable_domain_verification`, default `true`). If joining a protected room fails after a successful Keycloak login and `/var/log/prosody/prosody.err` reports a room or domain mismatch from the token module, add `enable_domain_verification = false` to the `VirtualHost "mail.example.com"` block as a diagnostic step; the token is then checked for signature, issuer, audience, expiry and room name only.

6.4 Optional: allow guests into rooms opened by a logged-in host

Token authentication as configured above requires every participant to log in. To let external guests join once a grommunio user has opened the room, follow the "Guest users" section of the adapter's standalone guide: it adds `allow_empty_token = true`, the modules `persistent_lobby` and `muc_wait_for_host` (both shipped in `jitsi-meet-prosody-plugins`), a `VirtualHost "guest.mail.example.com"` with `authentication = "jitsi-anonymous"` and `config.hosts.anonymousdomain = 'guest.mail.example.com'` in `config.js`. Test this variant separately from the basic SSO flow.

6.5 End-to-end SSO test

In a fresh browser profile open `https://mail.example.com/meet/sso-test-room`. Expected result, in order:

1. Meet redirects to `https://mail.example.com/oidc/auth?state=...`, which redirects to the Keycloak login page of the realm `grommunio`.
2. Log in as `alice@example.com` (complete MFA if configured).
3. Keycloak redirects to `https://mail.example.com/oidc/tokenize?...`, and the browser lands back in `/meet/sso-test-room` with a `jwt` parameter, showing the display name from the account.
4. A second logged-in user (`bob@example.com`) joins the same room; audio and video work as in Phase A.
5. An anonymous browser profile opening the room is sent to Keycloak instead of joining (unless you configured guest access in 6.4).

Only when all five points hold is SSO complete. If step 1 or 3 shows a Keycloak error page, read the *Events* view of the realm and the adapter journal; see the troubleshooting table.

7. Connect grommunio Web and the calendar

Users create and join meetings from grommunio Web; the steps for end users are described in Meet in grommunio Web. On the server side three settings matter.

Plugin defaults. `/etc/grommunio-web/config-meet.php` (linked from `/usr/share/grommunio-web/plugins/meet/config.php`) defines the array `MEET_DEFAULTS`. The `server` entry is the Meet base URL; by default it is derived from the host the user is logged in to:

/etc/grommunio-web/config-meet.php (excerpt)

```
define('MEET_DEFAULTS', [
  // Base URL of your Meet installation:
  'server' => 'https://' . $_SERVER['HTTP_HOST'] . '/meet/',

  // Uncomment to enable the plugin by default
  // 'enable' => true,

  // Uncomment to change default meeting opening behaviour, possible values: web browser popup
  'openin' => 'web',
  // ...
]);
```

Set `'server'` explicitly (`'https://mail.example.com/meet/'`) when Meet runs on a different host than grommunio Web, and uncomment `'enable' => true` if every user should get the Meet tab without enabling the plugin themselves. The other keys (`hidetabbarbutton`, `locationoverride`, `nolocationfix`, `noinvitation`, `invitationmessage`, `invitationhtml`) control the toolbar button, how the link is written into the appointment's location, and the invitation text template inserted into the appointment body. These are administrator defaults; they take effect for a user after the next login.

Per-user permission. In the Admin UI, each user has an *Allow Chat/Meet/Files/Archive* setting under the user's account properties (see Administration). grommunio Web asks the Admin API which plugins are disabled for a user, so the Meet plugin is unavailable to users without this permission.

App launcher. The Admin UI's application switcher reads `/etc/grommunio-admin-common/config.json`; make sure `videoWebAddress` points to your Meet URL (see Manual installation):

/etc/grommunio-admin-common/config.json (excerpt)

```
{
  "mailWebAddress": "https://mail.example.com/web",
  "videoWebAddress": "https://mail.example.com/meet"
}
```

Calendar workflow test. Log in to grommunio Web as `alice@example.com`, enable the Meet plugin in *Settings* > *Plugins* if it is not enabled by default, create a meeting request with `bob@example.com` as attendee, click *Add Meeting*, and save. Expected result: the Location field and the body contain `https://mail.example.com/meet/<room>`; both users can open the link (with Phase B, both are sent through Keycloak first) and meet in the same room. Room names generated by Web are random and not guessable; if users type their own keywords, advise them to choose names that are not easy to guess.

Verification checklist

Area	Check	Expected result
Packages	`rpm -q jitsi-meet jitsi-jicofo jitsi-videobridge jitsi-meet-prosody-plugins jitsi-meet-branding-grommunio prosody`	All six installed
Services	`systemctl is-active prosody jitsi-jicofo jitsi-videobridge nginx`	`active` for all
Bridge config loaded	`systemctl show -p Environment jitsi-videobridge`	Contains `-Dconfig.file=/etc/jitsi/videobridge/jvb.conf`
Bridge health	`curl -sS -i http://127.0.0.1:8081/about/health`	`HTTP/1.1 200 OK`
Bridge registered	`journalctl -u jitsi-jicofo -n 200 --no-pager`	No "no bridge available" errors after a room was joined
Web route	`curl -kI https://mail.example.com/meet/` and `/meet/config.js`	`200`
Media port	`ss -lunp | grep ':10000 '` and the external STUN check	Socket bound; STUN answer from outside
Firewall	perimeter and cloud firewall rules	443/tcp and 10000/udp inbound, 3478/udp and 443/tcp outbound
Single participant	Pre-join, camera, microphone, join	Joined; WebSocket to `/meet/xmpp-websocket` open
Multi-user	Two, then three browser contexts	All participants see and hear each other for 120 s
Reboot	Reboot the appliance, repeat service and URL checks	Identical results
Logs	`journalctl -u jitsi-jicofo -u jitsi-videobridge -p warning -n 50`, `/var/log/prosody/prosody.err`	No repeating errors
SSO (optional)	Fresh browser to a room	Keycloak login in realm `grommunio`, redirect back into the room, A/V works
grommunio Web	Meeting request with *Add Meeting*	Link in Location and body; attendees reach the room

Troubleshooting

Symptom	Likely cause	What to check / fix
`https://mail.example.com/meet/` returns 404 or the grommunio Web page	nginx include not loaded or nginx not reloaded after installation	`nginx -T | grep grommunio-meet.conf`; `nginx -t && systemctl reload nginx`
Page loads, but joining hangs at "connecting"	XMPP connection fails: wrong host in `config.js`, Prosody down, or WebSocket origin not allowed	Browser dev tools (WebSocket to `/meet/xmpp-websocket`); `systemctl status prosody`; `/var/log/prosody/prosody.err`; `hosts.domain` / `bosh` / `websocket` in `config.js` and `cross_domain_websocket` must use the host users type; re-run `grommunio-meet-setup <fqdn>`
Participants join but see "bridge unavailable" or never get video	Videobridge did not join the brewery: drop-in missing, password mismatch, `muc_jids` differs from `brewery-jid`	`systemctl show -p Environment jitsi-videobridge`; `journalctl -u jitsi-videobridge -u jitsi-jicofo -n 200`; compare `jvb.conf` and `jicofo.conf`; re-run `grommunio-meet-setup`
Two participants work, three or more do not, or media dies after a few seconds	UDP 10000 blocked, not forwarded, or wrong advertised address (P2P works, bridge does not)	Perimeter/NAT rules; external STUN check; `NAT_HARVESTER_*` in `sip-communicator.properties`; re-run with `JVB_PUBLIC_ADDRESS=<public-ip>`
Health check on 8081 returns 500 or "unhealthy"	Only private addresses available for ICE (lab) or mapping harvester failing	Fix public address mapping; in labs set `videobridge.health.require-valid-address = false`

Symptom	Likely cause	What to check / fix
`jitsi-jicofo` exits immediately	`/etc/jitsi/jicofo/jicofo.conf` missing	`journalctl -u jitsi-jicofo`; run `grommunio-meet-setup`
Bridge listens on 8080 instead of 8081, or fails to bind	`jvb.conf` not loaded (see above) or port 8080 already used by nginx	Restore the systemd drop-in; `ss -ltnp | grep -E ':(8080|8081) '`
Meet broke after a package update	`config.js`, `jvb.conf` or `prosody.cfg.lua` replaced or `.rpmnew` not merged	`ls /srv/jitsi-meet/*.rpm* /etc/jitsi/*/*.rpm* /etc/prosody/*.rpm*`; merge, re-run `grommunio-meet-setup`, re-apply Phase B changes
Clients cannot get camera or microphone	Browser or OS permissions on the client	See Troubleshooting Meet, Audio/Video
SSO: Keycloak shows "Invalid parameter: redirect_uri"	Redirect URI not registered; adapter uses `https://<host>/oidc/tokenize`	Add exactly that URI to the client's *Valid redirect URIs*; check realm *Events*
SSO: adapter logs "unauthorized" at `/oidc/tokenize`	Wrong `KEYCLOAK_CLIENT_SECRET`, realm or client ID	Copy the secret from the *Credentials* tab; compare `KEYCLOAK_REALM`, `KEYCLOAK_CLIENT_ID`; `systemctl restart oidc-adapter`
SSO: adapter cannot reach Keycloak	DNS, TLS trust or firewall between appliance and Keycloak	`curl https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration` from the appliance; `KEYCLOAK_ORIGIN_INTERNAL`
SSO: `oidc-adapter.service` fails to start	Deno missing, wrong path in `ExecStart`, unreadable environment file, port 9000 in use	`journalctl -u oidc-adapter`; `deno --version`; file mode `0640 root:jitsi-oidc-adapter`; `ss -ltnp | grep 9000` (change `PORT` and `proxy_pass` together)
SSO: `/oidc/auth` returns 404 from nginx	Drop-in location missing or shadowed	`nginx -T | grep -n oidc`; the file must be in `/etc/grommunio-common/nginx/locations.d/`
SSO: login loop between Meet and Keycloak	`tokenAuthUrl` wrong or `config.js` not delivering it; cookies/`SameSite` blocked	`curl -ks https://mail.example.com/meet/config.js | grep tokenAuth`; browser network log
SSO: login succeeds but Meet rejects the token	`app_id` / `app_secret` differ between adapter and Prosody; issuer or audience not accepted; virtual host still `anonymous`	Compare `JWT_APP_ID` / `JWT_APP_SECRET` with `app_id` / `app_secret`; `asap_accepted_issuers` / `asap_accepted_audiences`; `authentication = "token"`; `prosodyctl check config`; `/var/log/prosody/prosody.err`
SSO: Keycloak login succeeds, token is valid, but the room refuses the user with a room or domain mismatch	`sub` claim (`meet` tenant) does not match what Prosody's domain verification expects for the room	`/var/log/prosody/prosody.err`; set `enable_domain_verification = false` on the virtual host and retest
SSO: Meet loads but never asks for login	`tokenAuthUrl` missing from the delivered `config.js` or web app cached	Check `config.js`; hard-reload the browser

Operating notes

- **Logs.** Jicofo and the videobridge log to the journal: `journalctl -u jitsi-jicofo`, `journalctl -u jitsi-videobridge`. Prosody writes `/var/log/prosody/prosody.log` and `/var/log/prosody/prosody.err`; raise its log level in `/etc/prosody/prosody.cfg.lua` (`log = { info = ...}` to `debug`) only temporarily. nginx keeps Meet traffic in `/var/log/nginx/nginx-meet-access.log` and `nginx-meet-error.log`. The adapter (Phase B) logs to `journalctl -u oidc-adapter`.
- **Monitoring.** Poll `http://127.0.0.1:8081/about/health` (expects 200) and the four units. `/oidc/health` on the adapter returns `healthy`.
- **Ports.** Keep 5222, 5280, 5281, 8081, 9090 and 9000 unreachable from outside; only 443/tcp and 10000/udp are meant to be public.
- **Updates.** Update with `zypper ref && zypper up` as described in Operations. `config.js`, `jvb.conf` and `prosody.cfg.lua` are rpm configuration files: after an update look for `.rpmnew` / `.rpmsave` files, merge them, then re-run `grommunio-meet-setup <fqdn>` (which restarts the Meet services) and re-apply the Phase B Prosody changes. `zypper ps -s` lists services that still run old binaries.
- **Backup.** Include `/etc/jitsi/` (contains `.meet-secrets`), `/etc/prosody/`, `/var/lib/prosody/` (registered accounts), `/srv/jitsi-meet/config.js`, `/etc/grommunio-common/nginx/locations.d/`, `/etc/grommunio-web/config-meet.php` and, for Phase B, `/etc/jitsi/meet/oidc-adapter/` and the adapter checkout. Meetings themselves hold no persistent data.
- **Multiple servers.** In distributed setups the HAProxy/nginx front end must route `/meet`, `/colibri-ws` and the WebSocket upgrade to the Meet node; see the examples in Architecture.
- **Own TURN server.** Re-run `grommunio-meet-setup` with `TURN_HOST` and `TURN_SECRET` to switch from the managed relay to your coturn; the settings land in the `external_services` block of the Prosody host configuration.

Related pages

- Meet in grommunio Web - enabling the plugin, Meet tab, calendar links
- Troubleshooting Meet, Audio/Video - client-side device and browser permission problems
- Single sign-on with Keycloak - grommunio-auth/Keycloak setup used in Phase B
- Administration - per-user Meet permission
- Operations - updates, certificates, backup

- Architecture - HAProxy/nginx routing of `/meet` and `/colibri-ws` in multi-server setups
- Chat and Files and Office - the other collaboration components

grommunio Chat: teams, channels and users

grommunio Chat is the team messaging component of the grommunio stack. It is based on Mattermost, runs as a local service on the grommunio host and is published by nginx under `/chat/` on the same host name as grommunio Web and grommunio Admin. grommunio Admin holds the mapping between grommunio domains, grommunio users and the corresponding chat teams and chat accounts, so that chat is provisioned from the same place as mailboxes.

At the end of this guide you have a running chat server behind nginx, an Admin API connection that creates teams and accounts automatically, at least one domain team with two users exchanging messages in a channel, and a documented set of checks for the WebSocket connection, multi-tenancy boundaries and the optional single sign-on through grommunio-auth (Keycloak).

This page covers installation and administration. Day-to-day use of the Chat plugin inside grommunio Web (enabling the plugin, invitations, team roles) is described in Chat in grommunio Web.

How it fits together

Component	Role	Where
`grommunio-chat.service`	Mattermost-based chat server, runs as user `grochat`, listens on `127.0.0.1:8065` and on a local-mode Unix socket for CLI administration	`/etc/grommunio-chat/config.json`, `/var/lib/grommunio-chat`, `/var/log/grommunio-chat`
`grommunio-chat-ctl`	The `mmctl` administration command of the chat server	`/usr/bin/grommunio-chat-ctl`
MariaDB database `grochat`	Users, teams, channels and posts	local MariaDB, port 3306
nginx	Publishes `/chat/` on port 443 and proxies it, including WebSocket upgrades, to the upstream `chat_server`	`/usr/share/grommunio-common/nginx/locations.d/grommunio-chat.conf`, `/usr/share/grommunio-common/nginx/upstreams.d/grommunio-chat.conf`
PAM service `grommuniochat`	Lets chat accounts authenticate with grommunio credentials through `pam_gromox.so`	`/etc/pam.d/grommuniochat`
`grommunio-admin-api.service`	Talks to the chat REST API (`/chat/api/v4`) as a technical system administrator; creates a team per domain and an account per user	`/etc/grommunio-admin-api/conf.d/chat.yaml`
grommunio Web Chat plugin	Embeds `/chat/` in a tab of grommunio Web	`/etc/grommunio-web/config-chat.php`, see Chat in grommunio Web
grommunio-auth / Keycloak (optional)	OpenID Connect login for chat	realm `grommunio`, client `grommunio-chat`

Data flow: the browser opens `https://mail.example.com/chat/`, nginx forwards the request to the chat server on `127.0.0.1:8065`, the chat server reads and writes the `grochat` database. Real-time updates use a WebSocket connection on `/chat/api/v4/websocket`, for which the shipped nginx location sets the `Upgrade` headers; the chat server only accepts that connection when `ServiceSettings.SiteURL` matches the origin the browser uses.

How grommunio objects map to chat objects (as implemented by the Admin API):

grommunio object	Chat object
Domain with *Create grommunio-chat team* enabled (`chat=true`)	One team per domain. The team display name is the domain title (or the domain name), the team is invite-only. The team ID is stored as the domain's `chatID`.
User with *Create grommunio-chat user* enabled (`chat=true`)	One chat account. E-mail address = grommunio address, chat user name = address with `@` replaced by `_` (for example `alice_example.com`). The account is added to the domain team. Its ID is stored as the user's `chatID`.
User flag `chatAdmin`	Adds the `system_admin` role to the chat account.
User privilege `privChat`	Grants the *Chat* feature to the user in grommunio (feature switch, independent of the chat account).

Accounts created by the Admin API use the chat server's PAM authentication method, so users sign in with their grommunio e-mail address and grommunio password. Chat itself does not store a usable password for these accounts.

Prerequisites

- A working grommunio installation with DNS, TLS and reachable `/web/` and `/admin/` (see Quickstart and Post-installation).
- Root shell access to the grommunio host; `grommunio-admin` works.

- A strong, unique password for the `grochat` database user and one for the technical chat administrator. Keep them in a secret store or in root-only files; do not paste them into tickets.
- If you plan to use single sign-on: grommunio-auth with Keycloak is already set up as described in Single sign-on with Keycloak.

All commands below run as root on the grommunio host. Replace `mail.example.com` and `example.com` with your host name and mail domain.

1. Install the package and check the inventory

grommunio Admin ships the chat driver (the `mattermostdriver` Python package) even when the chat server itself is not installed. Check what is present before you install anything:

```
rpm -qa | grep -Ei 'grommunio-chat|mattermost' | sort
zypper search -s grommunio-chat
```

Install the chat server package. Depending on the repository it is called `grommunio-chat` or carries the Mattermost major version in its name (for example `grommunio-chat-v10`); use the name that `zypper search` shows:

```
zypper --non-interactive install grommunio-chat
```

The package installs these parts (check with `rpm -ql grommunio-chat`):

Path	Purpose
`/usr/bin/grommunio-chat`, `/usr/lib/systemd/system/grommunio-chat.service`	Server binary and unit (`User=grochat`, `WorkingDirectory=/usr/share/grommunio-chat/`)
`/usr/bin/grommunio-chat-ctl`	`mmctl` administration command
`/etc/grommunio-chat/config.json`	Server configuration, owned by `grochat`
`/etc/pam.d/grommuniochat`	PAM stack used for password logins (`pam_gromox.so`)
`/var/lib/grommunio-chat/`	Data directory (`files/` for attachments, `plugins/`, `client/`), mode 0700
`/var/log/grommunio-chat/`	Log directory (`/usr/share/grommunio-chat/logs` points here)
`/usr/share/grommunio-common/nginx/locations.d/grommunio-chat.conf`	nginx `location /chat` and WebSocket location
`/usr/share/grommunio-common/nginx/upstreams.d/grommunio-chat.conf`	nginx `upstream chat_server`

The directories and their ownership are created through `systemd-tmpfiles` (`/usr/lib/tmpfiles.d/grommunio-chat.conf`). Before moving on, `systemctl status grommunio-chat` should show the unit as loaded (it is not started yet) and `ls -ld /etc/grommunio-chat /var/lib/grommunio-chat /var/log/grommunio-chat` should show `grochat` as owner of the data and log directories.

2. Prepare the database and the configuration

The chat server needs its own database and a `config.json`. The package installs `/etc/grommunio-chat/config.json`; if your version ships only a `config.json.example`, copy it to `config.json` first and give it to `grochat` (`chown grochat:grochat`, `chmod 0640`).

Create the database and its user (replace `<strong-password>`):

```
CREATE DATABASE IF NOT EXISTS grochat CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER IF NOT EXISTS 'grochat'@'localhost' IDENTIFIED BY '<strong-password>';
GRANT ALL PRIVILEGES ON grochat.* TO 'grochat'@'localhost';
FLUSH PRIVILEGES;
```

Run the statements with `mariadb` as root.

Edit `/etc/grommunio-chat/config.json`. At minimum the following keys must be correct; keep all other keys. The excerpt shows only the relevant parts of each section.

/etc/grommunio-chat/config.json (excerpt)

```
{
  "ServiceSettings": {
    "SiteURL": "https://mail.example.com/chat",
    "ListenAddress": "127.0.0.1:8065",
    "EnableLocalMode": true,
    "LocalModeSocketLocation": "/var/tmp/grommunio-chat_local.socket"
  },
  "TeamSettings": {
    "EnableOpenServer": false,
    "RestrictCreationToDomains": "example.com"
  },
  "SqlSettings": {
    "DriverName": "mysql",
    "DataSource": "grochat:<strong-password>@tcp(localhost:3306)/grochat?charset=utf8mb4,utf8&readTimeout=30s&writeTimeout=30s"
  },
  "FileSettings": {
    "Directory": "/var/lib/grommunio-chat/files/"
  }
}
```

Key	Purpose
`ServiceSettings.SiteURL`	The exact URL users open in the browser, including the `/chat` path and without trailing slash. A mismatch in scheme, host, port or path makes the server reject WebSocket handshakes with HTTP 403.
`ServiceSettings.ListenAddress`	Bind to loopback only; nginx is the only client and the shipped upstream points to `127.0.0.1:8065`.
`ServiceSettings.EnableLocalMode`, `LocalModeSocketLocation`	Enables the Unix socket used by `grommunio-chat-ctl --local` for administration without a login.
`TeamSettings.EnableOpenServer`	`false`: nobody can self-register without an invitation. Accounts come from grommunio Admin.
`TeamSettings.RestrictCreationToDomains`	Comma-separated list of mail domains allowed to have accounts. Leave it empty to allow every domain, or list every grommunio domain that will use chat, including the domain of the technical administrator created in step 4.
`SqlSettings.DriverName`, `DataSource`	MariaDB connection for the `grochat` database.
`FileSettings.Directory`	Where attachments are stored; keep it inside `/var/lib/grommunio-chat`.

The file contains the database password; it must stay `grochat:grochat` with mode `0640` (the package sets this through tmpfiles). Before moving on, `python3 -m json.tool /etc/grommunio-chat/config.json >/dev/null` must succeed (valid JSON).

3. Start the service and check the nginx upstream

Start the chat server and check that the local-mode socket appears:

```
systemctl enable --now grommunio-chat
systemctl status grommunio-chat
ss -ltnp | grep 8065
test -S /var/tmp/grommunio-chat_local.socket && echo socket-ok
```

The first start creates the database schema; give it a few seconds and check `journalctl -u grommunio-chat` and `/var/log/grommunio-chat/mattermost.log` for SQL errors if the port or the socket does not appear.

nginx needs two snippets, both shipped by the package: the `location /chat` block (plus a separate location for `/chat/api/v*/websocket` that sets the `Upgrade` and `Connection` headers) and the upstream definition:

/usr/share/grommunio-common/nginx/upstreams.d/grommunio-chat.conf

```
upstream chat_server {
  server 127.0.0.1:8065;
}
```

Test the configuration and restart the web front end and the Admin API, so that both pick up the new service:

```
nginx -t
systemctl restart nginx grommunio-admin-api
```

If `nginx -t` fails with `host not found in upstream "chat_server"`, the upstream snippet is not included. Some appliance versions ship it as `/etc/grommunio-common/nginx/upstreams.d/grommunio-chat.conf.example`; in that case copy it to `grommunio-chat.conf` in the same directory and repeat `nginx -t`.

Check the published route:

```
curl -kI https://mail.example.com/chat/
curl -k https://mail.example.com/chat/api/v4/system/ping
```

Expected result: the first request returns `HTTP/2 200` (or `HTTP/1.1 200`), the second returns a JSON document containing `"status":"OK"`. nginx logs requests to `/chat` separately in `/var/log/nginx/nginx-chat-access.log` and `/var/log/nginx/nginx-chat-error.log`.

4. Create the technical chat administrator and connect the Admin API

grommunio Admin needs a chat account with the `system_admin` role to create teams and users. Create it through local mode, which needs no existing login. The domain restriction still applies: the address must belong to a domain listed in `RestrictCreationToDomains` (if that list is set); an address such as `admin@localhost` is rejected with `The email you provided does not belong to an accepted domain`.

```
export MMCTL_LOCAL_SOCKET_PATH=/var/tmp/grommunio-chat_local.socket
grommunio-chat-ctl --local user create --email chatadmin@example.com --username chatadmin --password '<strong-password>' --system-
```

`grommunio-chat-ctl` is the `mmctl` command of the Mattermost code base; all `mmctl` sub-commands and the global `--local` and `--json` flags work. `--local` connects to the Unix socket named by `MMCTL_LOCAL_SOCKET_PATH` (without the variable, `mmctl` looks for `/var/tmp/mattermost_local.socket`).

Tell the Admin API how to reach the chat server. The keys under `connection` are the options of the `mattermostdriver` Python package the API uses:

/etc/grommunio-admin-api/conf.d/chat.yaml

```
chat:
  connection:
    login_id: chatadmin
    password: '<strong-password>'
    url: mail.example.com
    basepath: /chat/api/v4
    port: 443
    scheme: https
    verify: True
```

Key	Meaning
`login_id`, `password`	The technical administrator created above
`url`, `port`, `scheme`, `basepath`	The API endpoint as published by nginx (`https://mail.example.com:443/chat/api/v4`)
`verify`	TLS verification: `True` (default), the path of a CA bundle, or `False`. `False` is only acceptable with a self-signed certificate on an isolated system; it also makes every `grommunio-admin` command that touches chat print an `InsecureRequestWarning`.

The Admin API runs as user `grommunio`; `/etc/grommunio-admin-api/conf.d/` is readable only by root and that user. Keep the file readable for the service and restart the API:

```
chown root:grommunio /etc/grommunio-admin-api/conf.d/chat.yaml
chmod 0640 /etc/grommunio-admin-api/conf.d/chat.yaml
systemctl restart grommunio-admin-api
```

Do not set mode `0600` with owner `root`; the API could not read the file and every chat operation would fail. Before moving on, `journalctl -u grommunio-admin-api -n 50` should show no connection or authentication errors for the `chat` service.

5. Activate chat for a domain and its users

Chat is enabled in two levels, and the order matters: first the domain (this creates the team), then the users (this creates the accounts and adds them to the team). Enabling a user before the domain is silently skipped; the Admin API logs `Could not enable chat for user '<user>': chat is not enabled for domain`.

Admin UI

1. Open **Domains**, select the domain and enable **Create grommunio-chat team**. Save.
2. Open **Users**, select a user and enable **Create grommunio-chat user**. Enable **Allow Chat** under the feature switches so the user sees the Chat entry in grommunio Web. Tick **grommunio-chat admin permission** only for users who should administer the chat server. Save.

The domain checkbox is greyed out when the Admin API cannot reach the chat server; the user checkbox is disabled while the domain has no team. To give new users the *Chat* feature automatically, set `privChat` in the system or domain **Defaults** (see Administration).

CLI

```
grommunio-admin domain modify example.com --chat true
grommunio-admin user modify alice@example.com --chat true --privChat true
grommunio-admin user modify bob@example.com --chat true --privChat true
```

Verify the result:

```
grommunio-admin domain query domainname chat chatID --format json-flat
grommunio-admin user query username chat chatAdmin privChat privWeb --format json-flat
```

Expected result: the domain shows `chat: true` and a 26-character `chatID`; each user shows `chat: true` and `privChat: true`. `--chat true` also works on `grommunio-admin domain create` and `grommunio-admin user create`, so new tenants and users can be provisioned with chat in one step.

If `user modify --chat true` does not set `chat: true`:

- `Cannot activate chat for deactivated domain` or `Cannot activate chat for locked user`: fix the status first.
- No error, `chat` stays `false`: check `journalctl -u grommunio-admin-api` for the warnings above (domain not enabled, chat service unreachable, user creation rejected by the chat server, for example because the domain is not in `RestrictCreationToDomains`, or `Unable to create the new team membership because the team has reached the limit of members`, which means `TeamSettings.MaxUsersPerTeam` in `config.json` is too low).

Do not repair the mapping by editing the `grochat` database. Fix the Admin API connection and repeat the `grommunio-admin` commands; the API reuses an existing team or account when the ID is already stored.

6. Log in and create channels

Open `https://mail.example.com/chat/` in a browser and sign in as `alice@example.com` with the grommunio password. After the first login the domain team is preselected and the default channels (Town Square, Off-Topic) are visible.

Create the channels your organisation needs. You can do that in the browser (see Chat in grommunio Web) or from the shell with local mode. The team name is the internal identifier the Admin API generated; look it up instead of guessing:

```
export MMCTL_LOCAL_SOCKET_PATH=/var/tmp/grommunio-chat_local.socket
grommunio-chat-ctl --local team list --json
```

Then create a channel in that team and add both users (users can be given as e-mail address or chat user name):

```
TEAM=<team-name-from-list>
grommunio-chat-ctl --local channel create --team "$TEAM" --name operations --display-name "Operations" --purpose "Monitoring, inci
grommunio-chat-ctl --local channel users add "$TEAM:operations" alice@example.com bob@example.com
```

Add `--private` to `channel create` for a channel that is only visible to its members. Before moving on, reload the browser: the channel *Operations* should be listed for `alice@example.com`.

7. Test with two users and check the WebSocket

A loading login page proves nothing about real-time delivery. Test with two separate browser profiles (or two devices):

1. Sign in as `alice@example.com` in the first browser and as `bob@example.com` in the second.
2. Both users should see the same team and the channel *Operations*.
3. Send a message as Alice. It must appear in Bob's window without a page reload, and Bob's reply must appear in Alice's window.

If messages only appear after a reload, the WebSocket connection is not established. In the browser developer tools (Network tab, filter *WS*), the request to `/chat/api/v4/websocket` must be answered with status `101`. A `403` on the WebSocket handshake means `SiteURL` differs from the URL in the address bar (scheme, host, port or path). A `502`, or a plain `200` without upgrade, means the request did not reach the WebSocket location of the shipped nginx snippet, for example because another proxy in front of nginx strips the `Upgrade` header.

Also confirm that the service survives a restart of the whole chain:

```
systemctl restart mariadb grommunio-chat nginx grommunio-admin-api
systemctl is-active mariadb grommunio-chat nginx grommunio-admin-api
curl -k https://mail.example.com/chat/api/v4/system/ping
```

8. Optional: single sign-on with grommunio-auth and Keycloak

Skip this step if you do not use grommunio-auth. Complete and test steps 1 to 7 with password login first, so that SSO problems are not confused with chat problems.

grommunio-auth manages one OpenID Connect client per application in the realm `grommunio` and exports the client data to `/var/cache/grommunio-auth/adaptor-config/`. Each application ships a script `setup-gk-app-g-<application>` in `/usr/share/grommunio-auth/adaptor-config-scripts/` that applies the exported data to the application's configuration. For chat, the script writes the Mattermost-compatible OpenID Connect settings (`Enable`, `Id`, `Secret`, `AuthEndpoint`, `TokenEndpoint`, `UserAPIEndpoint`, `DiscoveryEndpoint`, `ButtonText`) into `/etc/grommunio-chat/config.json`. You only have to provide the client in Keycloak.

Create the Keycloak client

In the Keycloak administration console (reachable through grommunio-auth under `/auth/`), switch to the realm `grommunio`. Do not work in the `master` realm. Check whether a client `grommunio-chat` already exists; if not, create it:

Setting	Value
Client type	OpenID Connect
Client ID	`grommunio-chat`
Client authentication	on (confidential client)
Standard flow	on
Direct access grants, Implicit flow, Service account roles	off
Valid redirect URIs	`https://mail.example.com/chat/login/gitlab/complete` and `https://mail.example.com/chat/signup/gitlab/complete`
Web origins	`https://mail.example.com`

Both callback paths are needed because the chat server returns to a different path for login and for first-time sign-up. The redirect URIs must match the published `/chat` path exactly.

Apply the adapter configuration

```
/usr/share/grommunio-auth/adaptor-config-scripts/setup-gk-app-g-chat
jq '.GitLabSettings | {Enable, Id, AuthEndpoint, TokenEndpoint, UserAPIEndpoint, DiscoveryEndpoint, ButtonText}' /etc/grommunio-ch
systemctl restart grommunio-chat
```

Expected result: `Enable` is `true`, `Id` is `grommunio-chat`, and the endpoints point to `https://mail.example.com/auth/realms/grommunio/...`. The client secret is written by the script and must not be printed or copied into documentation. If the `GitLabSettings` section is untouched after the script ran, check the `KeycloakSettings` section of `config.json` in the same way; it carries the same keys.

Test the SSO login

Open `https://mail.example.com/chat/` in a fresh browser profile. The login page should now show the Keycloak button. Click it, authenticate (including MFA if the realm requires it), and confirm that you land in the domain team. Repeat the two-user test of step 7 with both users signed in through SSO.

> **⚠Caution**
>
> Test SSO with a test user before announcing it. Accounts that were created with password login and accounts that arrive through SSO are matched by e-mail address; verify with one existing user that the SSO sign-in reaches the existing account and its channels rather than creating a second account.

9. Multi-tenancy: domain teams and where separation ends

Every grommunio domain with chat enabled gets its own team. A second domain therefore produces a second team on the same chat server:

```
grommunio-admin domain create tenant2.example.com -u 10 --title "Tenant 2" --chat true
grommunio-admin domain query domainname title chat chatID --format json-flat
```

Expected result: `example.com` and `tenant2.example.com` each show a different `chatID`. If `TeamSettings.RestrictCreationToDomains` is set, add the new domain to it and restart `grommunio-chat` before enabling its users.

Teams created by the Admin API are invite-only, so users of one domain do not see or join the team of another domain unless a team administrator invites them. This is organisational separation, not tenant isolation. Use it with the following limits in mind:

- **Separated per domain team:** team membership, public and private channels, team and channel roles, invitations, team-level settings.
- **Shared across all teams on the server:** the server configuration (`config.json`), plugins, integrations such as webhooks and bots, e-mail notification settings, file storage, the System Console (visible to every `system_admin`), search index, retention and compliance settings. Direct messages are possible between any two users of the server while `TeamSettings.RestrictDirectMessage` is `any`; set it to `team` to limit them to members of a common team.
- **Not separable on one server:** administrators (every `chatAdmin` is a system administrator of the whole server), SSO client, backup and restore (one database), security policy.

One domain team per department or subsidiary of the same organisation works well. For customers who require hard data-protection boundaries, separate administrators, separate SSO or separate backup and restore, run a separate grommunio installation (or at least a separate chat server) per customer.

10. Final check from grommunio Web

The integration is complete when a user reaches chat from grommunio Web without touching a second URL. The Web plugin is configured in `/etc/grommunio-web/config-chat.php`:

Constant	Meaning
`PLUGIN_CHAT_URL`	URL embedded in the Chat tab; defaults to `https://<host of the Web session>/chat/`, which matches the nginx route on a single host
`PLUGIN_CHAT_USER_DEFAULT_ENABLE`	`true` enables the plugin for every user without a visit to *Settings* > *Plugins*
`PLUGIN_CHAT_AUTOSTART`	`true` opens the Chat tab automatically at login

1. In grommunio Web, enable the **Chat** plugin under *Settings* > *Plugins* and reload (see Chat in grommunio Web). The **Chat** entry appears in the main navigation only for users with *Allow Chat* (`privChat`).
2. Open the Chat tab as `alice@example.com`. Depending on step 8, either the password login or the Keycloak login appears; complete it.
3. Post in *Operations* and confirm the message with `bob@example.com`, who uses the Chat tab in a second browser profile.

Optionally set `chatWebAddress` in `/etc/grommunio-admin-common/config.json` to `https://mail.example.com/chat` so that grommunio Admin shows Chat in its application launcher (see Administration).

Verification checklist

Area	Check	Expected result
Installation	`rpm -qa | grep grommunio-chat`	Chat server package is installed

Area	Check	Expected result
Services	`systemctl is-active grommunio-chat mariadb nginx grommunio-admin-api`	All four report `active`
Port	`ss -ltnp | grep 8065`	`grommunio-chat` listens on `127.0.0.1:8065` only
Local mode	`test -S /var/tmp/grommunio-chat_local.socket`	Socket exists; `grommunio-chat-ctl --local team list` returns the domain team(s)
nginx	`nginx -t`	`syntax is ok`, no `host not found in upstream "chat_server"`
Route	`curl -kI https://mail.example.com/chat/`	HTTP 200
API	`curl -k https://mail.example.com/chat/api/v4/system/ping`	`"status":"OK"`
Admin API coupling	`grommunio-admin domain query domainname chat chatID`	`chat: true` and a `chatID` for every chat-enabled domain, no `InsecureRequestWarning`
User mapping	`grommunio-admin user query username chat privChat`	`chat: true` for every provisioned user
Login	Browser login at `/chat/` with grommunio credentials	Domain team and default channels visible
WebSocket	Developer tools, request `/chat/api/v4/websocket`	Status 101, no 403
Two users	Message from Alice in *Operations*	Appears for Bob without reload; reply appears for Alice
Multi-tenancy	Second domain with `--chat true`	Own `chatID`, own team, users of domain A do not see team B
SSO (optional)	Keycloak button on `/chat/`, login round trip	User lands in the domain team; second user test passes
grommunio Web	Chat tab in grommunio Web	Login and messaging work inside the tab
Restart	`systemctl restart grommunio-chat nginx`	All checks above still pass
Logs	`journalctl -u grommunio-chat -u grommunio-admin-api -p warning`, `/var/log/grommunio-chat/mattermost.log`	No recurring errors during the tests

Troubleshooting

Symptom	Likely cause	What to check / fix
`nginx -t`: `host not found in upstream "chat_server"`	Upstream snippet not included	Make sure `/usr/share/grommunio-common/nginx/upstreams.d/grommunio-chat.conf` exists (reinstall the package) or copy the `.example` file in `/etc/grommunio-common/nginx/upstreams.d/`; run `nginx -t`, restart nginx
`/chat/` returns 502	Chat server not running or listening on another address	`systemctl status grommunio-chat`, `ss -ltnp | grep 8065`; `ServiceSettings.ListenAddress` must match the upstream (`127.0.0.1:8065`)
Messages appear only after reload; WebSocket handshake returns 403	`SiteURL` does not match the browser URL	Set `ServiceSettings.SiteURL` to exactly `https://mail.example.com/chat`, restart `grommunio-chat`; check reverse proxies and port forwards in front of nginx
Chat server fails to start, SQL errors in the journal	Database, user or password wrong	Compare `SqlSettings.DataSource` with the `CREATE USER` statement; test with `mariadb -u grochat -p grochat`
`user create` in local mode: `The email you provided does not belong to an accepted domain`	`RestrictCreationToDomains` excludes the address	Use an address in an allowed domain, or extend the list and restart `grommunio-chat`
`grommunio-chat-ctl --local` cannot connect	Local mode disabled or wrong socket path	`EnableLocalMode: true`; `LocalModeSocketLocation` and `MMCTL_LOCAL_SOCKET_PATH` must match
Admin UI: chat checkboxes disabled; API log shows connection or login errors	Admin API cannot log in to the chat server	`journalctl -u grommunio-admin-api`; verify `chat.yaml` (credentials, `url`, `port`, `basepath`, `verify`) and that the file is readable by user `grommunio`; restart `grommunio-admin-api`
`user modify --chat true` has no effect, log says `chat is not enabled for domain`	Domain team missing	`grommunio-admin domain modify <domain> --chat true` first, then repeat the user command
`Cannot activate chat for locked user` / `for deactivated domain`	Status prevents provisioning	Set the user or domain status to normal, then enable chat
API log: `Unable to create the new team membership because the team has reached the limit of members`	Team member limit reached	Raise `TeamSettings.MaxUsersPerTeam` in `config.json`, restart `grommunio-chat`, repeat `user modify --chat true`

Symptom	Likely cause	What to check / fix
User of a second domain cannot be created in chat	Domain missing in `RestrictCreationToDomains`	Add the domain to the list, restart `grommunio-chat`, repeat `user modify --chat true`
`InsecureRequestWarning` printed by `grommunio-admin` commands	`verify: False` in `chat.yaml`	Install a trusted certificate and set `verify: True` (or the path of your CA bundle)
Banner `Preview Mode: Email notifications have not been configured`	SMTP not configured in the chat server	Not needed for messaging; for production set the `EmailSettings` (`SMTPServer`, `SMTPPort`, `FeedbackEmail`, `SendEmailNotifications`) in `config.json` so that invitations and notifications are delivered
Keycloak button missing after step 8	Adapter did not write the settings or service not restarted	Re-run `setup-gk-app-g-chat`, check `Enable` in `GitLabSettings` (or `KeycloakSettings`) with `jq`, restart `grommunio-chat`
Keycloak: `Invalid parameter: redirect_uri`	Redirect URIs do not match the published path	Add both `/chat/login/gitlab/complete` and `/chat/signup/gitlab/complete` on `https://mail.example.com` to the client

Operating notes

- **Logs:** `journalctl -u grommunio-chat` for service start and stop, `/var/log/grommunio-chat/mattermost.log` (JSON lines) for the server itself and `/var/log/grommunio-chat/notifications.log` for notification delivery; `journalctl -u grommunio-admin-api` for provisioning problems (team and user creation, connection errors); `/var/log/nginx/nginx-chat-access.log` and `nginx-chat-error.log` for the HTTP layer. Log levels are set in `LogSettings` of `config.json`.
- **Backup:** back up `/etc/grommunio-chat/config.json` (contains the database password and, with SSO, the client secret), the data directory `/var/lib/grommunio-chat` (attachments under `files/`, installed plugins) and the MariaDB database `grochat`, together with the other grommunio databases. `/etc/grommunio*` is part of the standard appliance file backup; see Operations and Backup and restore. The team and account IDs stored in the grommunio database (`chatID`) must stay consistent with the `grochat` database; restore both from the same point in time.
- **Reset of the mapping:** if the chat database is rebuilt from scratch, `grommunio-admin chat remove-all` clears all stored `chatID` values of domains and users so that `--chat true` provisions fresh teams and accounts.
- **Updates:** update with `zypper` like the rest of the stack (see Operations). The package treats `config.json` as a configuration file: after an update, compare `/etc/grommunio-chat/config.json.rpmnew` with your file and merge new keys. The chat server applies database migrations on start; watch `journalctl -u grommunio-chat` after an update and re-run the ping and WebSocket checks.
- **Monitoring:** poll `https://mail.example.com/chat/api/v4/system/ping` and the `grommunio-chat` unit; alert on WebSocket errors reported by users, as they usually indicate a proxy or `SiteURL` change.
- **Removing users:** `grommunio-admin user delete` also removes the chat account; add `-c` (`--keep-chat`) to deactivate it instead. `grommunio-admin user modify <user> --delete-chat-user` removes the chat account of a user who stays in grommunio; see grommunio-admin user.

Related pages

- Chat in grommunio Web — enabling the plugin, invitations, team roles
- Administration — domain and user settings and Defaults in the Admin UI
- grommunio-admin domain and grommunio-admin user — CLI reference
- CLI cookbook — common provisioning commands
- Single sign-on with Keycloak — grommunio-auth and the `grommunio` realm
- Post-installation — DNS, TLS and base checks this guide builds on
- Operations and Backup and restore — backup artefacts and update procedure
- Architecture — routing `/chat` in multi-server and load-balanced setups

grommunio Files and Office: file sync and online editing

grommunio Files adds file storage, WebDAV access, sharing and a web interface to a grommunio installation. grommunio Office adds browser-based editing of Word, Excel and PowerPoint documents stored in Files, including collaborative editing. Both are optional components of the grommunio stack and are installed on the same host as the groupware in the default appliance layout.

This guide installs both components on an existing grommunio Appliance, connects Files to the grommunio user base, wires Files to Office, activates the Files integration in grommunio Web and validates the result with real file and document workflows. At the end, users log in to Files with their grommunio credentials, upload and share files, and open DOCX, XLSX and PPTX files in the browser from Files and from grommunio Web.

How it fits together

grommunio Files is a Nextcloud-based PHP application. It runs under PHP-FPM as the system user `grofiles` in its own pool (`grommunio-files-pool`, socket `/run/php-fpm/php-grommunio-files-fpm.sock`), is published by nginx under `/files/` on the groupware FQDN through the shipped location `/usr/share/grommunio-common/nginx/locations.d/grommunio-files.conf`, stores metadata in the MariaDB database `grofiles` and user data below `/var/lib/grommunio-files/data`. Redis (`redis@grommunio.service`, bound to `127.0.0.1:6379`) provides the distributed file locking and the local cache. The timer `grommunio-files-cron.timer` runs the Nextcloud background jobs (`cron.php`) every five minutes.

grommunio Office is an ONLYOFFICE Document Server. Its document service (`ds-docservice`, running as the user `groffice` and listening on port 8000) and converter (`ds-converter`) use RabbitMQ as message broker and the MariaDB database `groffice` for editing sessions and change tracking. nginx publishes the Document Server under `/office/` by proxying to `http://localhost:8000/` (`/usr/share/grommunio-common/nginx/locations.d/grommunio-office.conf`). The Document Server reads its configuration from `/etc/grommunio-office/` (`default.json`, `production-linux.json`). Files connects to Office through the bundled ONLYOFFICE app: the browser loads the editor from `/office/`, the Document Server fetches the document from Files through the configured storage URL and writes the saved document back through a callback.

grommunio Web integrates Files through its Files plugin. The plugin talks to Files over WebDAV (the default backend), lists folders inside the web client, attaches files from Files to e-mails, saves attachments to Files and, when Office is enabled, opens office file types in an editor tab.

Component	Package / units	Route or port	Data
grommunio Files	`grommunio-files`; `grommunio-files-cron.timer`	`https://<fqdn>/files/` (nginx location, PHP-FPM pool `grommunio-files-pool`)	`/var/lib/grommunio-files/data`, database `grofiles`
Redis	`redis@grommunio.service` (Valkey as `valkey@grommunio.service` on newer bases)	`127.0.0.1:6379`	cache, file locks (volatile)
grommunio Office	`grommunio-office` (pulls `grommunio-office-fonts` and system font packages); `ds-docservice`, `ds-converter`; one-shot `ds-themegen`, `ds-fontgen`; optional `ds-metrics`	`https://<fqdn>/office/` (nginx proxy to `localhost:8000`); `ds-docservice` on port 8000	database `groffice`
RabbitMQ	`rabbitmq-server`	local only	message queue for Office
grommunio Web Files plugin	part of `grommunio-web`	`https://<fqdn>/web/`	plugin settings in `/etc/grommunio-web/`

Paths that matter for Files:

Path	Purpose
`/usr/share/grommunio-files`	application root; contains `occ`
`/usr/share/grommunio-files/config/config.php`	Files configuration, including the database credentials and instance secret
`/usr/share/grommunio-files/apps`	bundled apps (read-only)
`/var/lib/grommunio-files/apps-external`	writable app directory
`/var/lib/grommunio-files/data`	user data, versions, trash
`/var/log/grommunio-files/files.log`, `upgrade.log`	Files application log and upgrade log
`/etc/php8/fpm/php-fpm.d/pool-grommunio-files.conf`	PHP-FPM pool for Files
`/usr/share/grommunio-common/nginx/locations.d/grommunio-files.conf`, `grommunio-office.conf`	nginx locations for `/files/` and `/office/` (shipped by the packages; do not edit in place)

Path	Purpose
`/etc/grommunio-office/default.json`	Office (Document Server) configuration, including the database credentials
`/usr/libexec/grommunio-office/server/schema/mysql/createdb.sql`	Office database schema

All `occ` commands in this guide are executed from `/usr/share/grommunio-files` as the `grofiles` user:

```
cd /usr/share/grommunio-files
sudo -u grofiles ./occ status
```

> **grommunio-setup**
>
> Current versions of grommunio-setup offer *files* and *office* as feature roles and perform the installation steps below automatically, including the Office wiring and the `fileWebAddress` link in `/etc/grommunio-admin-common/config.json`. The manual steps in this guide follow the same sequence, so that you can add the components to an installation without re-running the setup, and so that you know what to check afterwards. Older grommunio-setup versions re-initialise the whole installation when re-run; check the behaviour of your version before you use it on a production system.

Prerequisites

- A working grommunio installation with grommunio Web, Admin UI and mail, reachable under its production FQDN (`mail.example.com` in the examples). See Installation and Post-installation.
- A TLS certificate that browsers and the server itself trust. Files fetches documents and identity-provider metadata server-side, and Office fetches documents from Files, so a self-signed certificate breaks these paths unless you add lab-only workarounds.
- MariaDB, nginx, PHP-FPM and Redis running on the host.
- Disk space for user data, versions, trash, database and backups. Files keeps versions and deleted files in `/var/lib/grommunio-files/data`.
- Users who will use Files need the *Web*, *DAV* and *Files* privileges (`privWeb`, `privDav`, `privFiles`) in the Admin UI or via `grommunio-admin user modify`. The DAV privilege is required because Files authenticates users against the grommunio DAV endpoint.
- A snapshot or backup of the appliance before you start. See Backup and restore.

1. Record the baseline

Capture the state of the system before installing anything, so that later problems can be attributed to the new components rather than to pre-existing faults.

```
cat /etc/os-release
hostname -f
df -h
free -h
systemctl --failed
rpm -qa | grep -i grommunio | sort
ss -lntup
```

Expected result: `hostname -f` prints the FQDN that users will use for Files and Office, `systemctl --failed` lists no units, and nothing listens on port 8000 yet.

2. Install the packages

Files, Office and RabbitMQ come from the grommunio repositories. Install them together so that the PHP dependencies, fonts, Document Server and the nginx location snippets arrive in one consistent transaction.

```
zypper search -s grommunio-files grommunio-office
zypper --non-interactive install --auto-agree-with-licenses grommunio-files grommunio-office rabbitmq-server
rpm -q grommunio-files grommunio-office rabbitmq-server
```

Expected result: `rpm -q` prints a version for each of the three packages. `grommunio-office` pulls in `grommunio-office-fonts` and a set of system font packages, which the editor uses for rendering; RabbitMQ is not a package dependency and must be requested explicitly. Reload nginx and restart PHP-FPM so that the shipped `/files/` and `/office/` locations and the PHP pool are active:

```
nginx -t
systemctl reload nginx
systemctl restart php-fpm
```

Do not open Files in a browser yet; the instance is not initialised.

3. Create the Files database

Files needs its own database and database user. Keep the generated password in a root-only file; it goes into `config.php`, which must stay readable only by root and the `grofiles` user.

```
install -d -m 0750 /root/grommunio-secrets
openssl rand -base64 32 > /root/grommunio-secrets/files-db-password
chmod 0600 /root/grommunio-secrets/files-db-password
FILES_DB_PASSWORD="$(cat /root/grommunio-secrets/files-db-password)"
mariadb <<SQL
CREATE DATABASE IF NOT EXISTS grofiles CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
CREATE USER IF NOT EXISTS 'grofiles'@'localhost' IDENTIFIED BY '${FILES_DB_PASSWORD}';
GRANT ALL PRIVILEGES ON grofiles.* TO 'grofiles'@'localhost';
FLUSH PRIVILEGES;
SQL
```

If the central database runs on another host (see High availability), create the database and user there and use that host in the next step.

4. Initialise Files and set the trusted domains

The package ships an application tree without an initialised instance. Before running the installer, make sure `config.php` carries the appliance layout: the web root `/files`, the data directory, the log file, the two app paths, Redis for cache and locking, and web-based upgrades disabled (upgrades are done from the command line, see Operating notes). The following is the configuration that grommunio-setup writes; create the file if it does not exist yet.

/usr/share/grommunio-files/config/config.php

```
<?php
$CONFIG = array (
  'overwritewebroot' => '/files',
  'datadirectory' => '/var/lib/grommunio-files/data',
  'logfile' => '/var/log/grommunio-files/files.log',
  'theme' => 'theme-grommunio',
  'logtimezone' => 'UTC',
  'apps_paths' => array (
    0 => array (
      'path' => '/usr/share/grommunio-files/apps',
      'url' => '/apps',
      'writable' => false,
    ),
    1 => array (
      'path' => '/var/lib/grommunio-files/apps-external',
      'url' => '/apps-external',
      'writable' => true,
    ),
  ),
  'memcache.local' => '\\OC\\Memcache\\Redis',
  'filelocking.enabled' => true,
  'memcache.locking' => '\\OC\\Memcache\\Redis',
  'upgrade.disable-web' => true,
  'upgrade.automatic-app-update' => true,
  'updater.server.url' => '127.0.0.1',
);
```

Run the installer with the database credentials from step 3 and choose a password for the local Files administrator account. This account is a Files-internal account, independent of grommunio users; keep it for administration only.

```
cd /usr/share/grommunio-files
sudo -u grofiles ./occ maintenance:install --database mysql --database-name grofiles --database-user grofiles --database-pass "$FI
```

Set the trusted domains and the URLs Files uses when it generates links. `overwritewebroot` must stay `/files`, because nginx publishes Files under that path. Add every host name under which users reach Files (index `0` is `localhost`, set by the installer).

```
MAIL_FQDN="mail.example.com"
sudo -u grofiles ./occ config:system:set trusted_domains 1 --value="${MAIL_FQDN}"
sudo -u grofiles ./occ config:system:set overwrite.cli.url --value="https://${MAIL_FQDN}/files"
sudo -u grofiles ./occ config:system:set overwritewebroot --value="/files"
sudo -u grofiles ./occ config:system:set overwriteprotocol --value="https"
sudo -u grofiles ./occ config:system:set default_phone_region --value="<ISO-3166-1-country-code>"
```

Check before moving on:

```
sudo -u grofiles ./occ status
sudo -u grofiles ./occ config:system:get trusted_domains
```

Expected result: `installed: true`, `maintenance: false`, and the FQDN in the trusted domains list.

5. Redis, file locking and background jobs

File locking prevents two clients (or a client and the Office callback) from writing the same file concurrently; background jobs generate previews, expire versions and trash, and run app maintenance. Both must work before Office is connected.

The appliance runs Redis as `redis@grommunio.service`, bound to `127.0.0.1:6379`, which is also the address Files uses when no `redis` block is configured. Confirm the cache and locking settings from step 4, switch background jobs to cron and enable the timer:

```
systemctl is-active redis@grommunio.service
sudo -u grofiles ./occ config:system:set memcache.local --value="\\OC\\Memcache\\Redis"
sudo -u grofiles ./occ config:system:set memcache.locking --value="\\OC\\Memcache\\Redis"
sudo -u grofiles ./occ config:system:set filelocking.enabled --type=boolean --value=true
sudo -u grofiles ./occ background:cron
systemctl enable --now grommunio-files-cron.service grommunio-files-cron.timer
```

Only if your Redis instance listens elsewhere (for example a dedicated Redis host in a multi-node setup), point Files to it:

```
sudo -u grofiles ./occ config:system:set redis host --value="<redis-host>"
sudo -u grofiles ./occ config:system:set redis port --value="6379" --type=integer
```

Check before moving on:

```
systemctl list-timers grommunio-files-cron.timer
sudo -u grofiles ./occ status
```

Expected result: the timer has a next run time (it fires every five minutes and runs `cron.php` as `grofiles`), `occ config:app:get core backgroundjobs_mode` prints `cron`, and `occ status` still reports `maintenance: false`. If `occ` hangs or reports a Redis connection error, verify that the Redis instance is running and listening on the configured address (`ss -lntp | grep 6379`).

6. Connect Files to the grommunio user base

Files should not maintain a second, independent user directory. The `user_external` app with its `BasicAuth` backend authenticates every login against the grommunio DAV endpoint, so users log in to Files with the same address and password they use for grommunio Web. This also covers users that grommunio imports from LDAP, because the check goes through the grommunio authentication stack.

```
cd /usr/share/grommunio-files
sudo -u grofiles ./occ app:enable user_external
sudo -u grofiles ./occ config:system:set user_backends 0 class --value="\\OCA\\UserExternal\\BasicAuth"
sudo -u grofiles ./occ config:system:set user_backends 0 arguments 0 --value="https://${MAIL_FQDN}/dav"
```

The DAV endpoint must be reachable from the server itself under the FQDN and with a trusted certificate. Test the round trip with a real user before continuing:

```
curl -sS -o /dev/null -w '%{http_code}\n' -u 'alice@example.com:<password>' "https://${MAIL_FQDN}/dav/"
```

Expected result: an HTTP 2xx status. A `401` means the credentials or the user's DAV privilege are wrong; a certificate error means Files will fail the same way.

Confirm that the users who will use Files have the required privileges:

```
grommunio-admin user query username privWeb privDav privFiles --format json-flat
```

Set missing privileges in the Admin UI (*Users*, user detail, *Allow Chat/Meet/Files/Archive*, see Administration) or with `grommunio-admin user modify alice@example.com --privDav 1 --privFiles 1` (see grommunio-admin user).

For single sign-on through grommunio-auth and Keycloak instead of, or in addition to, the DAV login, see step 10.

7. Prepare Office: database and services

grommunio Office needs its own database and the schema from the package. The Document Server reads the credentials from `/etc/grommunio-office/default.json` under `services.CoAuthoring.sql`. The schema file contains `CREATE DATABASE` and `USE` statements for a differently named database; strip them and import the rest into `groffice`.

```
openssl rand -base64 32 > /root/grommunio-secrets/office-db-password
chmod 0600 /root/grommunio-secrets/office-db-password
OFFICE_DB_PASSWORD="$(cat /root/grommunio-secrets/office-db-password)"
mariadb <<SQL
CREATE DATABASE IF NOT EXISTS groffice CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
CREATE USER IF NOT EXISTS 'groffice'@'localhost' IDENTIFIED BY '${OFFICE_DB_PASSWORD}';
GRANT ALL PRIVILEGES ON groffice.* TO 'groffice'@'localhost';
FLUSH PRIVILEGES;
SQL
sed -e '/^CREATE DATABASE/d' -e '/^USE/d' /usr/libexec/grommunio-office/server/schema/mysql/createdb.sql | mariadb groffice
```

Write the credentials into `default.json` without changing the file's ownership and permissions (the file contains the password):

```
OFFICE_TMP="$(mktemp)"
jq --arg h localhost --arg n groffice --arg u groffice --arg p "${OFFICE_DB_PASSWORD}" '.services.CoAuthoring.sql.dbHost=$h | .ser
test -s "${OFFICE_TMP}" && cat "${OFFICE_TMP}" > /etc/grommunio-office/default.json
rm -f "${OFFICE_TMP}"
```

Start the services. `ds-themegen` and `ds-fontgen` are one-shot generators for editor themes and the font list from the fonts installed on the system; they run once and do not need to stay enabled. RabbitMQ must be up before the document service starts.

```
systemctl enable --now rabbitmq-server
systemctl start ds-themegen ds-fontgen
systemctl enable --now ds-converter ds-docservice
systemctl --no-pager is-active rabbitmq-server ds-converter ds-docservice
```

Check before moving on:

```
ss -lntp | grep ':8000'
curl -s "https://${MAIL_FQDN}/office/healthcheck"
```

Expected result: `ds-docservice` listens on port 8000 and the health check returns `true`. The health check covers the document service only; confirm separately that `rabbitmq-server` is `active` and not restarting in a loop, because collaborative editing and save callbacks depend on it. If the health check does not return `true`, check `journalctl -u ds-docservice -u ds-converter -u rabbitmq-server` before connecting Files.

> ⚠**Port 8000**
>
> By default the document service binds port 8000 on all interfaces, while nginx only needs to reach it on `localhost`. Port 8000 must not be reachable from outside the host: block it in the host firewall or on the network, and let all clients use `https://<fqdn>/office/` only.

8. Connect Files to Office

Files reaches Office through the ONLYOFFICE app. Set the public editor URL (loaded by the browser), the internal URL (used by Files server-side) and the storage URL (used by the Document Server to fetch and save documents). On a single-host appliance all three use the FQDN; the package default for `DocumentServerUrl` is the relative path `/office/`, which is equally valid when Files and Office share the host. `sameTab` opens documents in the Files tab instead of a new browser tab, which the grommunio Web integration relies on.

```
cd /usr/share/grommunio-files
sudo -u grofiles ./occ app:enable onlyoffice
sudo -u grofiles ./occ config:app:set onlyoffice DocumentServerUrl --value="https://${MAIL_FQDN}/office/"
sudo -u grofiles ./occ config:app:set onlyoffice DocumentServerInternalUrl --value="https://${MAIL_FQDN}/office/"
sudo -u grofiles ./occ config:app:set onlyoffice StorageUrl --value="https://${MAIL_FQDN}/files/"
sudo -u grofiles ./occ config:app:set onlyoffice sameTab --value=true
sudo -u grofiles ./occ config:app:set onlyoffice customizationForcesave --value=true
```

grommunio-setup additionally sets `customizationChat`, `customizationCompactHeader`, `customizationFeedback`, `customizationToolbarNoTabs` and `preview` for a compact editor inside grommunio Web; adjust these to taste with `occ config:app:set onlyoffice <key> --value=<true|false>`. It also sets the Files system option `csrf.disabled` to `true` when wiring Office. Keep that setting as grommunio-setup configured it; if you build the instance manually and the grommunio Web integration fails with CSRF errors, compare your `config.php` with a setup-generated one.

JWT secret. The Document Server can require a signed token on every request. As shipped, grommunio Office does not enforce tokens: in `/etc/grommunio-office/default.json`, `services.CoAuthoring.token.enable.browser`, `services.CoAuthoring.token.enable.request.inbox` and `services.CoAuthoring.token.enable.request.outbox` are `false`, and nginx restricts access to `/office/` and port 8000 to the host. If you enable token validation, set the secret in `services.CoAuthoring.secret.inbox.string` and `services.CoAuthoring.secret.outbox.string` (edit the file the same way as in step 7 to keep its permissions), restart `ds-docservice` and `ds-converter`, and configure the same secret in the Files app, otherwise the editor loads but every document open fails with a download or token error:

```
sudo -u grofiles ./occ config:app:set onlyoffice jwt_secret --value='<office-jwt-secret>'
```

Do not enable tokens on one side only. Package updates may leave a `default.json.rpmnew` next to the active file; merge new keys from it after updates.

> ⚠**Self-signed certificates**
>
> Files validates the Office certificate when it connects server-side, and Office validates the Files certificate when it fetches documents. With a self-signed certificate both directions fail. For an isolated lab only, you can disable certificate verification in the Files app:
>
> ```
> sudo -u grofiles ./occ config:app:set onlyoffice verify_peer_off --value=true
> ```
>
> Remove the setting again as soon as a trusted certificate is in place (`occ config:app:delete onlyoffice verify_peer_off`). Do not run a production system with peer verification disabled; use a certificate from a public or internal CA that the appliance trusts instead.

Open `https://mail.example.com/files/` as a regular user, create a new document from the *+* menu and confirm that the editor loads. The detailed edit-save-reopen test follows in step 13.

9. Integrate Files and Office with grommunio Web

The Files plugin of grommunio Web is configured server-side in its plugin configuration file, which on the appliance is `/etc/grommunio-web/config-files.php`. Three defines matter for this guide:

Define	Purpose	Value
`PLUGIN_FILES_USER_DEFAULT_ENABLE`	enables the plugin for all users; with `false`, each user enables it under *Settings*, *Plugins*	`true`
`PLUGIN_FILES_ONLYOFFICE_ENABLE`	opens office file types from Files in an Office editor tab instead of downloading them	`true`
`PLUGIN_FILES_ONLYOFFICE_FILETYPES`	file extensions opened in the editor; everything else is downloaded	`".doc,.docx,.docxf,.oform,.odp,.ods,.odt,.ppt,.pptx,.xls,.xlsx"`

Further defines in the same file: `PLUGIN_FILES_ASK_BEFORE_DELETE` (confirmation before deleting, default `true`), `PLUGIN_FILESBROWSER_LOGLEVEL` (`DEBUG`, `NORMAL`, `ERROR`, `NONE`), `PLUGIN_FILES_REDIS_HOST`, `PLUGIN_FILES_REDIS_PORT`, `PLUGIN_FILES_REDIS_AUTH` (Redis used for caching folder listings) and `PLUGIN_FILES_CACHE_TTL` (cache lifetime in seconds, default `900`).

```
grep -E 'PLUGIN_FILES_USER_DEFAULT_ENABLE|PLUGIN_FILES_ONLYOFFICE_ENABLE|PLUGIN_FILES_ONLYOFFICE_FILETYPES' /etc/grommunio-web/cor
```

Expected result:

/etc/grommunio-web/config-files.php

```
define('PLUGIN_FILES_USER_DEFAULT_ENABLE', true);
define('PLUGIN_FILES_ONLYOFFICE_ENABLE', true);
define('PLUGIN_FILES_ONLYOFFICE_FILETYPES', ".doc,.docx,.docxf,.oform,.odp,.ods,.odt,.ppt,.pptx,.xls,.xlsx");
```

If you change values, restart PHP-FPM and reload nginx afterwards:

```
systemctl restart php-fpm
systemctl reload nginx
```

Then, as a user, enable the Files plugin (if not enabled by default) and add a Files account in grommunio Web. The account uses the default WebDAV backend and points at your Files instance with the user's grommunio credentials. The user-facing steps are described in Files in grommunio Web.

Finally, publish the Files link in the Admin UI app launcher by setting `fileWebAddress` in `/etc/grommunio-admin-common/config.json` to `https://mail.example.com/files`, then restart `grommunio-admin-api` (see Application links). Office has no launcher entry; users reach it through Files.

10. Optional: single sign-on through grommunio-auth

With grommunio-auth and Keycloak in place (see Single sign-on with Keycloak), Files can use the same identity provider as grommunio Web through the `user_oidc` app. Office does not authenticate users itself; it works in the context of the Files session that opened the document.

Component	Authentication	User source	Browser session
grommunio Web	grommunio-auth / Keycloak	grommunio users	own session
grommunio Files	`user_oidc` via Keycloak, or DAV login (step 6)	Keycloak / grommunio DAV	own session; single sign-on when `user_oidc` is used
grommunio Office	callback from the Files app	document context of Files	editor session inside the Files page

The DAV backend from step 6 keeps working alongside `user_oidc`; the Files login page then offers both the password form and the Keycloak button.

1. In the Keycloak admin console, in the realm used by grommunio-auth (`grommunio` by default), create a confidential OpenID Connect client with client ID `grommunio-files`. Keep *Client authentication* and *Standard flow* enabled; leave *Direct access grants*, *Implicit flow* and *Service account roles* disabled. Register both callback variants of the `user_oidc` app as valid redirect URIs, because Files may return with or without `index.php` depending on the routing:

   - `https://mail.example.com/files/index.php/apps/user_oidc/code`
   - `https://mail.example.com/files/apps/user_oidc/code`

   Set the web origin to `https://mail.example.com`.

2. Enable the app and register the provider. The `user_oidc` app is installed but disabled by default, and its `occ` commands only exist while it is enabled. The `grommunio-files` package ships adaptor scripts under `/usr/share/grommunio-auth/adaptor-config-scripts/` (`setup-gk-app-g-files` and `delete-gk-app-g-files`, next to their grommunio Web counterparts). The Files adaptor reads the Keycloak client adapter file `/var/cache/grommunio-auth/adaptor-config/keycloak-grommunio-files.json` (fields `resource`, `credentials.secret`, `auth-server-url`, `realm`), which grommunio-auth writes when the client is created through it, and registers a provider named `grommunio Keycloak` with the user ID mapped to `preferred_username`, so that the Files user matches the account created by the DAV backend:

   ```
   cd /usr/share/grommunio-files
   sudo -u grofiles ./occ app:enable user_oidc
   /usr/share/grommunio-auth/adaptor-config-scripts/setup-gk-app-g-files
   sudo -u grofiles ./occ user_oidc:provider
   ```

   Without the adaptor file, register the provider manually with the client secret from Keycloak, using the same mappings the adaptor applies:

   ```
   sudo -u grofiles ./occ user_oidc:provider 'grommunio Keycloak' --clientid 'grommunio-files' --clientsecret '<client-secret>'
   ```

   `occ user_oidc:provider` without arguments lists the registered providers; `delete-gk-app-g-files` (or `occ user_oidc:provider:delete 'grommunio Keycloak'`) removes the provider again.

3. Files must be able to fetch the Keycloak discovery document server-side. In production this works with a trusted certificate. Files uses its own CA bundle at `/usr/share/grommunio-files/resources/config/ca-bundle.crt` in addition to the system trust store, so in an isolated lab with a self-signed certificate you would have to append the lab CA to that bundle as well; this is a lab workaround, not a production configuration.

   ```
   curl -fsS "https://mail.example.com/auth/realms/grommunio/.well-known/openid-configuration" > /dev/null
   curl -sI "https://mail.example.com/files/index.php/apps/user_oidc/login/1" | head -3
   ```

   Expected result: the discovery document downloads without a certificate error, and the login route answers with a `302` redirect to the Keycloak authorisation endpoint under `/auth/`.

4. Open `https://mail.example.com/files/` in a private browser window, choose the Keycloak login, authenticate (including the second factor if configured) and confirm that Files logs in the same user that grommunio Web shows.

11. Run the health checks

Check services and routes before testing with users. Files must redirect anonymous requests to its login page, Office must report healthy.

```
systemctl --no-pager --plain is-active nginx mariadb php-fpm rabbitmq-server ds-docservice ds-converter grommunio-files-cron.timer
systemctl --failed --no-pager --plain
ss -lntup | grep -E ':(443|3306|6379|8000)\s'
curl -sI "https://${MAIL_FQDN}/files/" | head -3
curl -s "https://${MAIL_FQDN}/office/healthcheck"
```

Expected result:

```
all listed units: active
0 failed units
nginx on 443, MariaDB on 3306, Redis on 127.0.0.1:6379, ds-docservice on port 8000
GET /files/ without session: HTTP 302 to the login page
GET /office/healthcheck: true
```

12. Log in and upload test files

Log in to `https://mail.example.com/files/` as a regular grommunio user (`alice@example.com` with her grommunio password when the DAV backend is used). Then upload one DOCX, one XLSX and one PPTX file, either through the web interface or over WebDAV. The WebDAV path uses the Files user name, which is the full e-mail address:

```
curl -sS -u 'alice@example.com:<password>' -T test.docx "https://${MAIL_FQDN}/files/remote.php/dav/files/alice@example.com/test.do
curl -sS -u 'alice@example.com:<password>' -T test.xlsx "https://${MAIL_FQDN}/files/remote.php/dav/files/alice@example.com/test.xl
curl -sS -u 'alice@example.com:<password>' -T test.pptx "https://${MAIL_FQDN}/files/remote.php/dav/files/alice@example.com/test.pp
```

Expected result: HTTP `201 Created` for each upload, and the three files appear in the user's root folder in the web interface with Office icons and a preview.

13. Edit, save and re-read a document

Opening the editor is not sufficient as an acceptance test; the saved content must reach the file in Files. Open `test.docx` from Files, type a unique marker text such as `Files Office validation`, wait for the editor to report that all changes are saved, close the editor, and read the file back over WebDAV:

```
curl -sS -u 'alice@example.com:<password>' -o /tmp/test.docx "https://${MAIL_FQDN}/files/remote.php/dav/files/alice@example.com/te
file /tmp/test.docx
unzip -p /tmp/test.docx word/document.xml | grep -c 'Files Office validation'
```

Expected result: `file` reports a Microsoft Word 2007+ document and `grep -c` prints `1` or more. Repeat the open test with `test.xlsx` and `test.pptx`; both must open in the spreadsheet and presentation editor respectively. Also open the same DOCX from two browser sessions (two users, or the same user in a second private window) and confirm that both see each other's edits, which proves that the Document Server, RabbitMQ and the Files callback work together.

14. Share with a second user

Sharing exercises the Files database, the user backend (the recipient must be resolvable) and the WebDAV view of the recipient. Share the DOCX from `alice@example.com` with `bob@example.com` through the OCS sharing API (`shareType=0` is a user share, `permissions=15` grants read, update, create and delete), then list Bob's root folder:

```
curl -sS -u 'alice@example.com:<password>' -H 'OCS-APIRequest: true' --data-urlencode 'path=/test.docx' -d 'shareType=0' --data-ur
curl -sS -u 'bob@example.com:<password>' -X PROPFIND -H 'Depth: 1' "https://${MAIL_FQDN}/files/remote.php/dav/files/bob@example.co
```

Expected result: the share call returns `"statuscode":200` in the `ocs.meta` object, and Bob's PROPFIND response lists `test.docx`. Log in as Bob in a separate browser session, open the shared file in the editor and save a change; Alice must see the change when she reopens the file.

> **Note**
>
> With the `user_external` backend, a user account is created in Files at the user's first login. If a share to `bob@example.com` fails with a user-not-found error, log in once as Bob (or check the sharing autocomplete restrictions in the Files administration settings), then repeat the share.

15. Test the grommunio Web workflows

Test Files and Office on their own first (steps 12 to 14), then the three workflows that the grommunio Web integration adds. Each item must succeed with a normal user account.

Workflow A: attach a file from Files to a new e-mail

1. In grommunio Web, open a new e-mail.
2. Open the attachment menu and choose the Files entry (visible only when the plugin is enabled and an account is configured).
3. Browse to a folder and select an existing file.
4. Send the message. Expected result: the file arrives as a regular attachment at the recipient.

Workflow B: save a received attachment to Files

1. Open an e-mail with an attachment.
2. Open the attachment's actions and choose the entry that saves it to Files.
3. Choose the target folder.
4. Expected result: the file is visible in that folder in Files, including over WebDAV.

Workflow C: edit an Office document from grommunio Web

1. In the Files tab of grommunio Web, double-click a `.docx`, `.xlsx` or `.pptx` file.
2. Expected result: the document opens in an Office editor tab inside grommunio Web (not as a download), because the extension is listed in `PLUGIN_FILES_ONLYOFFICE_FILETYPES`.
3. Change something, wait for the saved confirmation, close the tab and reopen the file. Expected result: the change is present.

Verification checklist

Area	Check	Expected result
Packages	`rpm -q grommunio-files grommunio-office rabbitmq-server`	all three installed
Services	`systemctl is-active nginx mariadb php-fpm rabbitmq-server ds-docservice ds-converter grommunio-files-cron.timer`	all `active`, `systemctl --failed` empty
Files instance	`occ status`	`installed: true`, `maintenance: false`
Network	`ss -lntup`	nginx on 443, `ds-docservice` on port 8000 (not reachable from outside), Redis on `127.0.0.1:6379`
Files route	`curl -sI https://<fqdn>/files/`	`302` to the login page
Office route	`curl -s https://<fqdn>/office/healthcheck`	`true`
Background jobs	`systemctl list-timers grommunio-files-cron.timer`; Files admin overview	timer scheduled; last cron run recent
Authentication	login with a grommunio user (DAV backend) and, if configured, with Keycloak	same user identity as in grommunio Web
Upload	WebDAV `PUT` of DOCX, XLSX, PPTX	`201 Created`, files visible in the web interface
Editing	edit DOCX, save, download, `unzip -p ... word/document.xml`	marker text present in the file
Collaboration	two sessions on one document	both see each other's changes
Sharing	OCS share to a second user, PROPFIND as that user	share created, file listed
grommunio Web	workflows A, B and C	attach from Files, save to Files, edit in Office tab
Restart	`systemctl restart nginx php-fpm rabbitmq-server ds-docservice ds-converter`; reboot	all functions unchanged afterwards
Logs	`/var/log/grommunio-files/files.log`, `journalctl -u ds-docservice -u ds-converter`	no repeated errors during the tests

Troubleshooting

Symptom	Likely cause	What to check / fix
`https://<fqdn>/files/` returns 404 or the nginx default page	nginx location for `/files/` not active, or PHP-FPM pool not loaded	`nginx -t`, `systemctl reload nginx`, `systemctl restart php-fpm`; verify the package installed correctly
"Access through untrusted domain" page	host name missing in `trusted_domains`	`occ config:system:get trusted_domains`, add the name with `occ config:system:set trusted_domains <n> --value=<host>`
Login with a grommunio user fails, Files administrator login works	`user_external` misconfigured, DAV URL not reachable from the server, missing `privDav` / `privFiles`, or certificate not trusted	`curl -u user:pass https://<fqdn>/dav/` from the server; `grommunio-admin user query username privDav privFiles`; check `files.log`
Files login page loads but `occ` or page loads are very slow	Redis unreachable; locking waits for timeouts	`ss -lntp`
Previews, trash expiry or version cleanup never happen	background jobs not running	`occ background:cron`, `systemctl enable --now grommunio-files-cron.timer`, check the last cron run in the Files administration overview
Editor stays blank or reports that the document service is unreachable	`ds-docservice` down, RabbitMQ down, wrong `DocumentServerUrl`, or health check fails	`curl -s https://<fqdn>/office/healthcheck`, `journalctl -u ds-docservice -u ds-converter -u rabbitmq-server`, `occ config:app:get onlyoffice DocumentServerUrl`
Health check returns `true`, but collaborative editing or saving is unreliable	`rabbitmq-server` failed or restarting in a loop (`activating (auto-restart)`)	`systemctl status rabbitmq-server`, `journalctl -u rabbitmq-server`; fix RabbitMQ, then restart `ds-docservice` and `ds-converter`
Editor loads, but opening a document fails with a download or callback error	Document Server cannot fetch from `StorageUrl`, certificate not trusted, or JWT secret mismatch	`curl -sI https://<fqdn>/files/` from the server; compare the token settings in `/etc/grommunio-office/default.json` with `occ config:app:get onlyoffice jwt_secret`; only in a lab: `verify_peer_off`
Document opens but saved changes are missing after reopening	callback from Office to Files blocked, or file locking broken	`files.log` for callback errors, Redis status, `filelocking.enabled`
Wrong or missing fonts in the editor	font list not generated after installing fonts	install the fonts, run `systemctl start ds-fontgen`, restart `ds-docservice` and `ds-converter`
Office file types download instead of opening in grommunio Web	`PLUGIN_FILES_ONLYOFFICE_ENABLE` false, or extension missing in `PLUGIN_FILES_ONLYOFFICE_FILETYPES`	check `/etc/grommunio-web/config-files.php`, restart `php-fpm`, reload grommunio Web
Files entry missing in grommunio Web	plugin not enabled for the user, or no account configured	*Settings*, *Plugins*, *Files*; then add an account, see Files in grommunio Web
Keycloak login button missing or redirect loop	`user_oidc` provider not registered, redirect URI mismatch, or discovery document not fetchable server-side	`occ user_oidc:provider`, compare redirect URIs in Keycloak with `/files/index.php/apps/user_oidc/code`, `curl` the discovery URL from the server
Files stuck in maintenance mode after an update	`occ upgrade` was interrupted	`occ upgrade`, then `occ maintenance:mode --off`; check `files.log`

Operating notes

Backup. In addition to the artefacts listed in Backup and disaster recovery, back up:

- Files: `/var/lib/grommunio-files` (data and external apps), the `grofiles` database and `/usr/share/grommunio-files/config/config.php`. `config.php` contains the instance ID, the instance secret and the database credentials; without it a restored database cannot be attached to the data directory.
- Office: the `groffice` database and `/etc/grommunio-office/default.json` (database credentials and token settings). Editing sessions are transient; a restored Office instance starts with no open documents.
- When a role is removed through grommunio-setup, it keeps a copy of these configuration files under `/etc/grommunio-common/setup-backup/<role>/` so that the role can be re-added against the preserved data. The directory only exists after such a removal.

Put Files into maintenance mode for a consistent file-level backup of a busy instance (`occ maintenance:mode --on`, back up, `occ maintenance:mode --off`), or use a filesystem snapshot. See Backup and restore.

Logs. Files writes to `/var/log/grommunio-files/files.log` (JSON lines; `occ log:manage` sets the level) and records command-line upgrades in `/var/log/grommunio-files/upgrade.log`. Office, converter and RabbitMQ log to the journal: `journalctl -u ds-docservice -u ds-converter -u rabbitmq-server`. HTTP-level errors for `/files/` and `/office/` are in the nginx logs. For the grommunio Web plugin, set `PLUGIN_FILESBROWSER_LOGLEVEL` to `DEBUG` temporarily and read the PHP-FPM and grommunio Web logs.

Updates. Update the packages with `zypper update` (see Updating grommunio). Web-based upgrades are disabled (`upgrade.disable-web`), so after a `grommunio-files` update run the database migration from the command line and make sure the instance leaves maintenance mode:

```
cd /usr/share/grommunio-files
sudo -u grofiles ./occ upgrade
sudo -u grofiles ./occ maintenance:mode --off
sudo -u grofiles ./occ status
```

After a `grommunio-office` update, compare `/etc/grommunio-office/default.json` with a new `default.json.rpmnew` if one appeared, restart `ds-docservice` and `ds-converter` and re-check `/office/healthcheck`. Open documents are closed by the restart; announce maintenance windows to users.

Routine occ tasks.

```
sudo -u grofiles ./occ app:update --all
sudo -u grofiles ./occ files:scan --all
sudo -u grofiles ./occ maintenance:repair
```

`files:scan` is needed when files are added or changed on disk outside Files (for example after a restore). Run `occ` only as `grofiles`; running it as root changes file ownership in the application tree.

Monitoring. Alert on `systemctl --failed`, on the `grommunio-files-cron.timer` not firing, on `/office/healthcheck` returning anything other than `true`, and on disk usage of `/var/lib/grommunio-files/data`. The Files administration overview lists background job status, missing indices and security warnings after each update.

Related pages

- Files in grommunio Web - enabling the plugin and adding an account as a user
- Administration - user privileges, application links
- Operations - updates, backup artefacts
- Single sign-on with Keycloak - grommunio-auth and Keycloak setup
- Post-installation - FQDN, TLS and baseline checks
- Backup and restore
- grommunio-admin user - setting privileges from the command line
- High availability - load balancer routes for `/files` and `/office`

grommunio Archive: e-mail archiving, search and restore

grommunio Archive is the e-mail archiving component of grommunio. It is based on the open source archiver Piler and stores an independent, deduplicated and encrypted copy of every message that passes through the mail system, indexes it for full-text search and lets users find, view, export and restore messages through a web interface under `/archive/`.

At the end of this guide you have an archive that receives a copy of each message from Postfix, a working search index, users who can log in with their grommunio credentials, and a proven restore path from the archive back into the mailbox. You also know which artefacts to back up and what to monitor.

> **⚠Compliance is your responsibility**
>
> grommunio Archive is the technical building block for retention and compliance archiving. Which messages must be retained, for how long, who may access them, how exports must look and how evidential value is preserved are legal and organisational questions. Define them with your organisation and legal counsel and map them onto the retention rules, legal hold and access roles described below.

How it fits together

Postfix delivers a message to the mailbox as usual and, in addition, creates a copy for an internal archive address. A transport map routes that address to the archive SMTP listener on `127.0.0.1:2693`. The archive daemon stores the message in `/var/lib/grommunio-archive/store`, writes metadata to the `groarchive` database in MariaDB and feeds the Sphinx full-text index. Users search and open messages in the web interface; the interface authenticates against the Gromox IMAP service and uses the same IMAP connection to restore messages into the user's mailbox.

Component	Unit / process	Listens on	Role
Archive SMTP listener	`grommunio-archive-smtp.service` (`piler-smtp`)	`listen_addr:2693`	Accepts the copies handed over by Postfix
Archive daemon	`grommunio-archive.service` (`piler`)	–	Deduplicates, encrypts and stores messages, writes metadata
Full-text index	`searchd.service` (Sphinx, runs as `groarchive`)	`127.0.0.1:9306` (MySQL protocol), `127.0.0.1:9312`	Serves search queries; indexes are refreshed by cron
Web interface	nginx location `/archive/` , PHP-FPM pool `grommunio-archive-pool`	HTTPS 443	Login, search, view, export, restore, administration
Database	MariaDB, database `groarchive`	local socket	Metadata, recipients, users, rules, audit log
IMAP service	Gromox imap(8gx)	993	Authentication backend and restore target

Files and directories you will work with:

Path	Purpose
`/etc/grommunio-archive/grommunio-archive.conf`	Daemon and SMTP listener configuration (template: `grommunio-archive.conf.dist`)

Path	Purpose
`/etc/grommunio-archive/config-site.php`	Web interface configuration (template: `config-site.dist.php`)
`/etc/grommunio-archive/sphinx.conf.dist`	Template that renders `/etc/sphinx/sphinx.conf`
`/etc/grommunio-archive/grommunio-archive.key`	Encryption key for the message store
`/etc/php8/fpm/php-fpm.d/pool-grommunio-archive.conf`	PHP-FPM pool, socket `/run/php-fpm/php-grommunio-archive-fpm.sock`
`/usr/share/grommunio-common/nginx/locations.d/grommunio-archive.conf`, `/usr/share/grommunio-common/nginx/upstreams.d/grommunio-archive.conf`	nginx location and upstream shipped with the package
`/usr/share/grommunio-archive/archive/`	Web interface document root
`/usr/share/grommunio-archive/db-mysql.sql`	Database schema including the built-in `admin@local` and `auditor@local` accounts
`/usr/libexec/grommunio-archive/`	Helper scripts: `indexer.delta.sh`, `indexer.main.sh`, `purge.sh`, `pilerpurge.py`, `import.sh`
`/var/lib/grommunio-archive/{store,tmp,sphinx,error,stat,imap}`	Message store, temporary files, Sphinx index files, failed messages, status beacons
`/var/spool/cron/tabs/groarchive`	Crontab of the `groarchive` user (indexer, purge, cleanup)

On the grommunio Appliance, `grommunio-setup` performs steps 2 to 6 when you select the Archive feature. Use those steps to verify or adjust the result, or to install Archive manually on a system that was not set up with the wizard. In the container deployment the same components are enabled with `ENABLE_ARCHIVE=true`, see Container deployment.

Prerequisites

- A working grommunio installation with DNS, a trusted TLS certificate for the public name (for example `mail.example.com`), time synchronisation, access to the Admin UI and functioning user mailboxes. See the post-installation guide.
- The Gromox IMAP service is running and reachable on port 993 under the same name that is on the TLS certificate; the archive web interface authenticates and restores over IMAP with TLS.
- Postfix is the MTA on the host, configured with the grommunio MySQL maps (see the Postfix table in High availability).
- MariaDB is local or reachable, and you can run SQL as the database administrator.
- Disk capacity for the message store. Plan `/var/lib/grommunio-archive` like mailbox storage: it grows with every message for the whole retention period.

Check the starting point:

```
hostname -f
rpm -qa | grep -Ei 'grommunio|gromox' | sort
systemctl is-active postfix nginx php-fpm mariadb
ss -ltn | grep -E ':993 |:25 '
```

1. Install the packages

The archive and the Sphinx search engine come from the grommunio repository.

```
zypper refresh
zypper install grommunio-archive sphinx
```

Expected result: `rpm -q grommunio-archive sphinx` prints both packages, the system user `groarchive` exists and `/var/lib/grommunio-archive` contains the `store`, `tmp`, `sphinx`, `stat` and `imap` directories.

```
id groarchive
ls -la /var/lib/grommunio-archive
ls /etc/grommunio-archive
```

2. Prepare the database

Archive uses its own database, `groarchive` by default. Create the database and a dedicated user, then import the schema.

```
mariadb -uroot <<'SQL'
CREATE DATABASE IF NOT EXISTS groarchive CHARACTER SET utf8mb4;
CREATE USER IF NOT EXISTS 'groarchive'@'localhost' IDENTIFIED BY '<strong-password>';
GRANT ALL PRIVILEGES ON groarchive.* TO 'groarchive'@'localhost';
FLUSH PRIVILEGES;
SQL
```

The shipped schema creates two built-in accounts, `admin@local` (administration) and `auditor@local` (read access to all archived mail). Their passwords are the placeholders `grommunioArchiveAdmin` and `grommunioArchiveAuditor` in the SQL file. Replace them while importing; never import the schema with the placeholders.

```
sed -e 's#grommunioArchiveAdmin#<archive-admin-password>#g' \
    -e 's#grommunioArchiveAuditor#<auditor-password>#g' \
    /usr/share/grommunio-archive/db-mysql.sql | mariadb groarchive
mariadb -N groarchive -e "SHOW TABLES" | wc -l
```

Expected result: around 40 tables, among them `metadata`, `rcpt`, `sph_index`, `retention_rule`, `archiving_rule`, `legal_hold`, `audit` and `user`.

3. Configure the daemon, the web interface and Sphinx

3.1 Daemon configuration

Copy the template and restrict it to root and the `groarchive` group, because it contains the database password.

```
cp -n /etc/grommunio-archive/grommunio-archive.conf.dist /etc/grommunio-archive/grommunio-archive.conf
chgrp groarchive /etc/grommunio-archive/grommunio-archive.conf
chmod 0640 /etc/grommunio-archive/grommunio-archive.conf
```

Set the values that apply to your environment:

/etc/grommunio-archive/grommunio-archive.conf

```
hostid=mail.example.com
listen_addr=127.0.0.1
listen_port=2693
mysqlsocket=/run/mysql/mysql.sock
mysqldb=groarchive
mysqluser=groarchive
mysqlpwd=<strong-password>
queuedir=/var/lib/grommunio-archive/store
workdir=/var/lib/grommunio-archive/tmp
sphxhost=127.0.0.1
sphxport=9306
encrypt_messages=1
default_retention_days=2557
extra_to_field=X-Envelope-To:
```

Key	Shipped default	Meaning
`hostid`	placeholder	SMTP HELO name of the listener. Use the domain part of the archive address (`mail.example.com` for `archive@mail.example.com`).
`listen_addr`	`0.0.0.0`	Address the SMTP listener binds to. Set `127.0.0.1` when Postfix runs on the same host; the shipped default listens on all interfaces.
`listen_port`	`2693`	Port of the SMTP listener. Must match the Postfix transport entry.
`mysqlhost`, `mysqlsocket`	empty, `/run/mysql/mysql.sock`	Database connection; the socket is used when `mysqlhost` is empty.

Key	Shipped default	Meaning
`mysqldb`, `mysqluser`, `mysqlpwd`	`groarchive`, `groarchive`, empty	Database credentials from step 2.
`queuedir`	`/var/lib/grommunio-archive/store`	Message store.
`workdir`	`/var/lib/grommunio-archive/tmp`	Temporary files while a message is processed.
`sphxhost`, `sphxport`	`127.0.0.1`, `9306`	MySQL-protocol endpoint of `searchd`.
`encrypt_messages`	`1`	Encrypt stored messages with `grommunio-archive.key`. Decide before the first message arrives and never change it afterwards.
`default_retention_days`	`2557`	Retention for messages without a matching retention rule (7 years + 2 days). Stored per message at archiving time.
`extra_to_field`	`X-Envelope-To:`	Header from which the archive reads the envelope recipient; Postfix adds it (step 6).
`username`	`groarchive`	Service account of the daemons.

Query the effective configuration at any time with `pilerconf -q <key>`, for example `pilerconf -q listen_addr`.

3.2 Encryption key

With `encrypt_messages=1` the store is unreadable without `/etc/grommunio-archive/grommunio-archive.key`. Create it once, only if it does not exist yet, and include it in every backup.

```
test -s /etc/grommunio-archive/grommunio-archive.key || head -c 56 /dev/urandom > /etc/grommunio-archive/grommunio-archive.key
chgrp groarchive /etc/grommunio-archive/grommunio-archive.key
chmod 0640 /etc/grommunio-archive/grommunio-archive.key
```

> **Danger**
>
> Losing or regenerating the key after messages have been archived makes the existing store permanently unreadable. Treat the key like a private key.

3.3 Web interface configuration

The web interface reads `/usr/share/grommunio-archive/archive/config.php` and then overrides it with `/etc/grommunio-archive/config-site.php`. Render the template by replacing its placeholders: `MYHOSTNAME` (public name of the appliance), `MYSMTP` (mail domain used for notification senders) and the `MYSQL_*` database values.

```
sed -e 's#MYHOSTNAME#mail.example.com#g' \
    -e 's#MYSMTP#example.com#g' \
    -e 's#MYSQL_HOSTNAME#localhost#' \
    -e 's#MYSQL_DATABASE#groarchive#' \
    -e 's#MYSQL_USERNAME#groarchive#' \
    -e 's#MYSQL_PASSWORD#<strong-password>#' \
    /etc/grommunio-archive/config-site.dist.php > /etc/grommunio-archive/config-site.php
chgrp groarchive /etc/grommunio-archive/config-site.php
chmod 0640 /etc/grommunio-archive/config-site.php
```

The template already contains the grommunio-specific settings; check them after rendering:

Setting	Value in the template	Meaning
`PATH_PREFIX`, `SITE_URL`	`/archive/`, `https://<MYHOSTNAME>/archive/`	Public URL of the web interface.
`ENABLE_IMAP_AUTH`	`1`	Users log in with their grommunio e-mail address and password, checked over IMAP.
`IMAP_HOST`, `IMAP_PORT`, `IMAP_SSL`	`<MYHOSTNAME>`, `993`, `true`	IMAP endpoint for login and restore. The host name must match the TLS certificate of the IMAP service.
`RESTORE_OVER_IMAP`	`1`	Restore writes the message back into the mailbox over IMAP.

Setting	Value in the template	Meaning
`IMAP_RESTORE_FOLDER_INBOX`, `IMAP_RESTORE_FOLDER_SENT`	`INBOX`, `Sent`	Target folders for restored received and sent messages.
`SMTP_DOMAIN`, `SMTP_FROMADDR`, `ADMIN_EMAIL`	derived from `MYSMTP`	Sender and contact addresses for notifications.
`DB_HOSTNAME`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`	from `MYSQL_*`	Database access of the web interface.

Defaults that remain in `config.php` unless you override them in `config-site.php`: `ENABLE_AUDIT = 1` (every view, download and restore is logged), `ENABLE_DELETE = 0` (nobody can delete from the archive through the interface), `BULK_DOWNLOAD_FOR_USERS = 1`, `ENABLE_PDF_DOWNLOAD = 0`, `SPHINX_HOSTNAME = 127.0.0.1:9306`, `SPHINX_MAIN_INDEX = main1,dailydelta1,delta1`, `SESSION_EXPIRY = 3600`.

3.4 Sphinx configuration and initial index

`sphinx.conf.dist` is a PHP template. Render it, insert the database credentials, hand the file to the `groarchive` and `sphinx` accounts and build the empty indexes once.

```
php /etc/grommunio-archive/sphinx.conf.dist > /etc/sphinx/sphinx.conf
sed -i -e 's#MYSQL_HOSTNAME#localhost#' \
       -e 's#MYSQL_DATABASE#groarchive#' \
       -e 's#MYSQL_USERNAME#groarchive#' \
       -e 's#MYSQL_PASSWORD#<strong-password>#' /etc/sphinx/sphinx.conf
chown groarchive:sphinx /etc/sphinx/sphinx.conf
chmod 0640 /etc/sphinx/sphinx.conf
chown -R groarchive:sphinx /var/lib/grommunio-archive/sphinx
runuser -u groarchive -- indexer --config /etc/sphinx/sphinx.conf --all
```

Expected result: `grep '^index ' /etc/sphinx/sphinx.conf` lists `main1` to `main4`, `dailydelta1`, `delta1`, `tag1` and `note1`, and `/var/lib/grommunio-archive/sphinx/` contains index files for each of them.

4. Enable the services and check the listeners

```
systemctl enable --now searchd.service grommunio-archive-smtp.service grommunio-archive.service
systemctl restart php-fpm.service
systemctl is-active searchd grommunio-archive-smtp grommunio-archive php-fpm nginx
ss -ltnp | grep -E ':2693|:9306|:9312'
```

Expected result: all five services are `active`; `searchd` listens on `127.0.0.1:9306` and `127.0.0.1:9312`; `piler-smtp` listens on port 2693 on the address you set in `listen_addr`. If the listener shows `0.0.0.0:2693`, set `listen_addr=127.0.0.1` and restart `grommunio-archive-smtp.service`, or restrict the port with the host firewall. Only HTTPS should be reachable from outside.

5. Publish the web interface under /archive/

The package ships the nginx configuration; nothing has to be written by hand. `/etc/nginx/conf.d/grommunio.conf` includes `/usr/share/grommunio-common/nginx.conf`, which loads `/usr/share/grommunio-common/nginx/locations.d/*.conf` and `/usr/share/grommunio-common/nginx/upstreams.d/*.conf`, plus local overrides from `/etc/grommunio-common/nginx/locations.d/` and `/etc/grommunio-common/nginx/upstreams.d/`. The archive snippet serves the document root, passes PHP to the `fpm_archive` upstream and rewrites the application routes (search, message view, bulk restore, retention, legal hold, audit). In essence:

/usr/share/grommunio-common/nginx/locations.d/grommunio-archive.conf (excerpt)

```
location /archive {
        alias /usr/share/grommunio-archive/archive;
        index index.php;
        try_files $uri $uri/ /archive/index.php;
        access_log /var/log/nginx/nginx-archive-access.log;
        error_log /var/log/nginx/nginx-archive-error.log;
}

location ~* ^/archive/(qr|js|sso|index).php {
        alias /usr/share/grommunio-archive;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_pass fpm_archive;
        include fastcgi_params;
}
```

/usr/share/grommunio-common/nginx/upstreams.d/grommunio-archive.conf

```
upstream fpm_archive {
        server unix:/run/php-fpm/php-grommunio-archive-fpm.sock;
}
```

Do not edit the files under `/usr/share`; place changes in `/etc/grommunio-common/nginx/locations.d/`. Test and reload:

```
nginx -t
systemctl reload nginx
ls -la /run/php-fpm/php-grommunio-archive-fpm.sock
curl -kI https://mail.example.com/archive/
```

Expected result: an HTTP `200` or a redirect to the login page, and the PHP-FPM socket owned by `groarchive`.

To link the archive from the Admin UI, set `archiveWebAddress` to `https://mail.example.com/archive` in `/etc/grommunio-admin-common/config.json` (see Administration).

6. Connect the mail flow in Postfix

Postfix creates the archive copy. Two mechanisms are available; the appliance uses the first one.

6.1 Archive everything (appliance default)

`always_bcc` adds the archive address as a recipient to every message Postfix accepts. A `transport_maps` entry routes that address to the archive listener, and a `check_recipient_access` map prepends an `X-Envelope-To` header per recipient so that the archive knows the real envelope recipient (this is what makes BCC recipients and distribution lists attributable).

```
postconf -e "always_bcc=archive@mail.example.com"
echo "archive@mail.example.com smtp:[127.0.0.1]:2693" > /etc/postfix/transport
postmap /etc/postfix/transport
echo '/(.*)/   prepend X-Envelope-To: $1' > /etc/postfix/grommunio-archiver-envelope.cf
```

Make sure `transport_maps` points at the map and that the recipient restrictions contain the envelope map right after the permit rules. The appliance uses these values:

/etc/postfix/main.cf (relevant settings)

```
always_bcc = archive@mail.example.com
transport_maps = lmdb:/etc/postfix/transport
smtpd_recipient_restrictions = permit_sasl_authenticated, permit_mynetworks,
    check_recipient_access pcre:/etc/postfix/grommunio-archiver-envelope.cf,
    reject_unknown_recipient_domain, reject_non_fqdn_hostname,
    reject_non_fqdn_sender, reject_non_fqdn_recipient,
    reject_unauth_destination, reject_unauth_pipelining
```

```
systemctl restart postfix
postconf always_bcc transport_maps smtpd_recipient_restrictions
postmap -q archive@mail.example.com lmdb:/etc/postfix/transport
```

Expected result of the last command: `smtp:[127.0.0.1]:2693`. The domain part of the archive address must equal `hostid` in the daemon configuration. Only mail that passes through Postfix is archived; in a standard grommunio installation this includes mail submitted by Outlook, grommunio Web and mobile clients, because Gromox hands outgoing messages to the local MTA (`outgoing_smtp_url`, see gromox.cfg(5)).

6.2 Archive selected users only

If only some mailboxes must be archived, leave `always_bcc` unset (`postconf -X always_bcc`) and use the per-user *E-Mail forward* in the Admin UI (user, tab *SMTP*, CC mode) with `archive@mail.example.com` as the destination. Postfix evaluates these entries through `recipient_bcc_maps = mysql:/etc/postfix/grommunio-bcc-forwards.cf`, which queries the `forwards` table with `forward_type = 0`; the transport and envelope maps from 6.1 are still required. See Administration for the forward settings and the CLI cookbook for scripted user changes.

```
postmap -q alice@example.com mysql:/etc/postfix/grommunio-bcc-forwards.cf
```

Expected result: the archive address for every user that should be archived, and no output for users that should not.

7. Grant users access

The web interface authenticates over IMAP, so an archive user needs the POP3/IMAP privilege in addition to the Archive privilege. This is the most common reason for a failing archive login, see KB: Archive. Set both in the Admin UI (user, *Account*: *Allow POP3/IMAP logins* and *Allow Archive*) or on the command line:

```
grommunio-admin user modify alice@example.com --pop3-imap true --privArchive true
grommunio-admin user query username pop3_imap privArchive -f username=alice@example.com
```

Expected result: `pop3_imap` and `privArchive` are `true` for the user. See grommunio-admin user for all fields.

Then open `https://mail.example.com/archive/` and log in with the full e-mail address and the mailbox password. Regular users see only messages they sent or received. The built-in `auditor@local` account can search across all mailboxes, `admin@local` manages domains, rules, legal hold and users. Keep these two accounts for auditors and administrators, use strong passwords and do not hand them to regular users.

8. Prove that messages are archived

Send a test message with a unique marker in the subject and follow it through Postfix, the SMTP listener, the daemon, the database and the index.

```
/usr/sbin/sendmail -t <<'MAIL'
From: Archive Test <bob@example.com>
To: alice@example.com
Subject: Archive validation MARKER-20260916-1
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8

This message verifies archiving, search and restore.
MAIL
```

Check each stage:

```
journalctl -u postfix --since "-10 min" --no-pager | grep 2693
journalctl -u grommunio-archive-smtp --since "-10 min" --no-pager
journalctl -u grommunio-archive --since "-10 min" --no-pager
mariadb groarchive -e "SELECT id, CAST(subject AS CHAR(255)) AS subject, FROM_UNIXTIME(arrived) AS arrived FROM metadata ORDER BY
```

Expected results:

- Postfix: `relay=127.0.0.1[127.0.0.1]:2693 ... status=sent (250 OK <ID>)` for the archive recipient, and the normal delivery to the mailbox.
- `grommunio-archive-smtp`: `received: <ID>, from=<...>, size=..., client=127.0.0.1`.
- `grommunio-archive`: a line ending in `status=stored` with the same ID, `retention=2557` (or the value of your rule). `status=duplicate` means the identical message was already archived; this is deduplication, not an error.
- The `metadata` query lists the test subject as the newest row. The `subject` column is a `blob`, hence the `CAST`.

The search index is refreshed by the `groarchive` crontab: `indexer.delta.sh` every 30 minutes (minute 5 and 35), `indexer.main.sh` nightly at 02:30, `tag1` / `note1` every 15 minutes. To make the message searchable immediately, run the delta indexer manually and query Sphinx directly over its MySQL protocol port:

```
crontab -l -u groarchive
runuser -u groarchive -- /usr/libexec/grommunio-archive/indexer.delta.sh
mariadb -h127.0.0.1 -P9306 -e "SELECT id FROM main1,dailydelta1,delta1 WHERE MATCH('MARKER-20260916-1');"
```

Expected result: the `id` from the `metadata` table. Indexer messages are logged to the journal with the prefix `INDEXER INFO` or `INDEXER ERROR`.

9. Search, open and restore

In the web interface, search for the marker. The hit must show date, sender, recipient, subject and size. Open the message and check that headers, body and the restore action are visible.

For a real restore test the message must first be gone from the mailbox:

1. Delete the test message from Alice's inbox in grommunio Web (and empty *Deleted Items*), or remove it with an IMAP client.
2. Confirm it is gone: search the mailbox in grommunio Web or list the folder over IMAP.
3. In Archive, open the message and use *Restore to mailbox*. The interface appends the message over IMAP into `INBOX` (sent messages go to `Sent`).
4. Confirm that the message is back in the mailbox with the original headers.

Every restore, view and download is recorded in the audit log (`ENABLE_AUDIT = 1`), which the auditor can search in the web interface. For automated acceptance tests, check the mailbox over IMAP rather than through the browser; it verifies the same folder without UI caching effects.

10. Export, attachments and user isolation

- Export: a user can download an opened message as EML and, with `BULK_DOWNLOAD_FOR_USERS = 1`, download all hits of a search. PDF export is off by default (`ENABLE_PDF_DOWNLOAD = 0`). For bulk exports on the server, `pilerexport` selects by date range (`-a`, `-b`), sender or recipient (`-f`, `-r`, `-F`, `-R`), a Sphinx match expression (`-w`) and writes EML files, optionally into zip archives (`-z`); run `pilerexport` without arguments for the option list.
- Attachments: open a message with an attachment; the attachment must be listed in the message view and be contained in the exported EML.
- Isolation: search for one of Alice's messages while logged in as Alice (hit) and as Bob (no hit). Only `auditor@local` and accounts with the auditor role see mail of other users.
- Deletion from the archive through the interface is disabled by default (`ENABLE_DELETE = 0`). Messages leave the archive only through retention and purge.

11. Retention, legal hold and purge

Retention works in two layers:

- `default_retention_days` in `grommunio-archive.conf` (2557 days) is written into each message's metadata when it is archived. Changing it affects new messages only.
- In the web interface, `admin@local` defines *retention rules* (retention period by domain, sender, recipient, subject, size or attachment type), *archiving rules* (exclusions that are not archived at all) and *legal hold* per address. Legal hold suspends purging for the affected mail.

Expired messages are removed by the daily purge job (`purge.sh` at 03:40 in the `groarchive` crontab), which runs `pilerpurge.py`. Preview what the next purge would remove without deleting anything:

```
runuser -u groarchive -- /usr/libexec/grommunio-archive/pilerpurge.py --dry-run --verbose
```

Record the retention decisions and the rule set together with the rest of your compliance documentation; the archive enforces them, but it does not define them.

Verification checklist

Check	How	Expected result
Packages	`rpm -q grommunio-archive sphinx`	Both installed
Services	`systemctl is-active searchd grommunio-archive-smtp grommunio-archive php-fpm nginx postfix`	All `active`
Listeners	`ss -ltnp | grep -E ':2693|:9306|:9312'`	2693 on `127.0.0.1` (or firewalled), 9306 and 9312 on `127.0.0.1`
Postfix routing	`postmap -q archive@mail.example.com lmdb:/etc/postfix/transport`	`smtp:[127.0.0.1]:2693`
Postfix copy	`postconf always_bcc` or `postmap -q <user> mysql:/etc/postfix/grommunio-bcc-forwards.cf`	Archive address returned
Web interface	`curl -kI https://mail.example.com/archive/`	HTTP 200 or redirect to login
User privileges	`grommunio-admin user query username pop3_imap privArchive -f username=<user>`	Both `true`
Login	Browser login with e-mail address and password	Search page opens
Archiving	Test mail, `journalctl -u grommunio-archive`	`status=stored` for the message
Database	`mariadb groarchive -e "SELECT COUNT(*) FROM metadata"`	Count increases with each archived message
Index	Sphinx `MATCH()` query after `indexer.delta.sh`	`id` of the test message
Search and view	Search marker in the web interface, open the hit	Headers, body, attachments and restore action visible
Restore	Delete from mailbox, restore from archive	Message back in `INBOX`
Isolation	Search as a second user	No hit for the first user's mail
Cron	`crontab -l -u groarchive`	Delta, main, purge and cleanup entries present
Restart	`systemctl restart searchd grommunio-archive-smtp grommunio-archive` and reboot test	Archiving and search work afterwards
Logs	`journalctl -u grommunio-archive -u grommunio-archive-smtp -u searchd --since -1h`	No errors

Troubleshooting

Symptom	Likely cause	What to check / fix
Login at `/archive/` fails for a grommunio user	POP3/IMAP or Archive privilege missing; IMAP service unreachable or certificate name mismatch	`grommunio-admin user query ... pop3_imap privArchive`; `ss -ltn | grep 993`; `IMAP_HOST` in `config-site.php` must match the certificate; see KB: Archive
`404` or blank page under `/archive/`	nginx snippet not loaded, PHP-FPM pool not running	`nginx -T | grep -A3 'location /archive'`; `ls -la /run/php-fpm/php-grommunio-archive-fpm.sock`; `/var/log/nginx/nginx-archive-error.log`; `systemctl restart php-fpm`
Postfix logs `connect to 127.0.0.1[127.0.0.1]:2693 ... Connection refused`, archive copies deferred	`grommunio-archive-smtp` not running or `listen_addr` / `listen_port` mismatch	`systemctl status grommunio-archive-smtp`; `ss -ltnp | grep 2693`; `postqueue -p` or the *Mail queue* page in the Admin UI
Message is in `metadata` but the search finds nothing	Delta indexer has not run yet (30-minute interval) or fails	Run `indexer.delta.sh` manually; `journalctl --since -1h | grep INDEXER`; `searchd` active; `SPHINX_MAIN_INDEX` unchanged
No `received:` line in `grommunio-archive-smtp` although Postfix reports `status=sent` to 2693	Another service answers on 2693 or the copy goes to a different host	`ss -ltnp | grep 2693`; `postconf transport_maps`; `postmap -q` on the transport map
Message not attributed to the recipient (found only by the auditor)	`X-Envelope-To` header missing	`postconf smtpd_recipient_restrictions` contains `check_recipient_access pcre:/etc/postfix/grommunio-`

Symptom	Likely cause	What to check / fix
		`archiver-envelope.cf`; `extra_to_field=X-Envelope-To:` in the daemon config
Restore does not arrive in the mailbox	`RESTORE_OVER_IMAP`, `IMAP_HOST` or TLS settings wrong; user lacks IMAP privilege	`config-site.php`; `/var/log/nginx/nginx-archive-error.log`; audit log entry for the restore; test an IMAP login for the user
Port 2693 reachable from other hosts	Shipped default `listen_addr=0.0.0.0`	Set `listen_addr=127.0.0.1`, restart `grommunio-archive-smtp`, or restrict the port in the host firewall
Files accumulate in `/var/lib/grommunio-archive/error`	Messages could not be stored (database down, permissions, disk full)	`journalctl -u grommunio-archive`; `df -h /var/lib/grommunio-archive`; `cat /var/lib/grommunio-archive/stat/error`
Archived messages cannot be opened after a restore of the platform	Wrong or missing `grommunio-archive.key`	Restore the key from backup; compare `encrypt_messages` with the value used when the messages were archived
Configuration ignored after a package update	New `.dist` template, old values kept in `.rpmsave`	Compare `/etc/grommunio-archive/*.dist` with your files, merge new keys

Operating notes

Backup and restore of the archive platform

Back up the whole set together, consistent with the general procedure in Operations and the backup and restore guide:

- Database dump of `groarchive`.
- `/var/lib/grommunio-archive` (message store, Sphinx index files, status beacons). The Sphinx indexes can be rebuilt from the database, the store cannot.
- `/etc/grommunio-archive` including `grommunio-archive.key`.
- `/etc/sphinx/sphinx.conf`.
- `/etc/postfix` (transport map, `grommunio-archiver-envelope.cf`, `main.cf`).
- `/etc/grommunio-common/nginx` and the PHP-FPM pool if you changed them.

```
install -d -m 0700 /var/backups/grommunio-archive
mariadb-dump --single-transaction groarchive | gzip -c > /var/backups/grommunio-archive/groarchive.sql.gz
tar -C / -czf /var/backups/grommunio-archive/archive-config-data.tar.gz \
    etc/grommunio-archive etc/sphinx/sphinx.conf etc/postfix var/lib/grommunio-archive
gzip -t /var/backups/grommunio-archive/groarchive.sql.gz
tar -tzf /var/backups/grommunio-archive/archive-config-data.tar.gz | head
```

The backup contains the encryption key and database passwords; store it with the same protection as the key itself.

Restore order: stop the three archive services, restore the configuration and the key, import the database dump, restore `/var/lib/grommunio-archive`, restore the ownership of `/var/lib/grommunio-archive` to `groarchive` as declared in `/usr/lib/tmpfiles.d/grommunio-archive.conf`, rebuild the indexes with `runuser -u groarchive -- indexer --config /etc/sphinx/sphinx.conf --all --rotate` and start the services. Finish with a search and a restore test.

Monitoring and logs

Monitor more than an HTTP 200 on `/archive/`:

What	How
Services	`systemctl is-active searchd grommunio-archive-smtp grommunio-archive`
Listeners	`ss -ltn` for 2693, 9306, 9312
Archive throughput	`mariadb groarchive -e "SELECT COUNT(*) FROM metadata"`; `status=stored` lines in `journalctl -u grommunio-archive`
Index freshness	Modification time of `/var/lib/grommunio-archive/stat/indexer` (touched by every indexer run); `INDEXER ERROR` lines in the journal
Purge	Modification time of `/var/lib/grommunio-archive/stat/purge`
Failed messages	`/var/lib/grommunio-archive/stat/error` (count, updated every 5 minutes)
Storage	`du -sh /var/lib/grommunio-archive`; free space on the file system
Web interface	`/var/log/nginx/nginx-archive-access.log`, `/var/log/nginx/nginx-archive-error.log`

Add a synthetic end-to-end check (send, search, restore) to your monitoring schedule; it is the only test that proves the whole chain.

Updates

Update with `zypper update` like the rest of the appliance. After an update, compare your configuration with the new `.dist` templates in `/etc/grommunio-archive/` and check `crontab -l -u groarchive`, then repeat the verification checklist.

Related pages

- KB: Archive login not possible
- Administration: users, privileges and e-mail forwards
- grommunio-admin user
- CLI cookbook
- Operations: backup and restore
- Backup and restore guide
- Post-installation guide
- High availability: Postfix integration
- Container deployment
- imap(8gx)

Mobile devices: Exchange ActiveSync and device management

grommunio serves phones and tablets over Exchange ActiveSync (EAS), implemented by grommunio-sync. Device management in grommunio covers the ActiveSync relationship between a mailbox and a device: listing devices, checking their sync status, forcing a full resync, removing stale server-side state, enforcing sync policies (PIN, lock timeout, encryption) and issuing a remote wipe.

It is not a unified endpoint management (UEM) product. grommunio does not install apps, distribute certificates or configure device settings outside the EAS policy set, and every policy is only as effective as the client's implementation of it.

This guide takes you from a working grommunio installation to a user whose device synchronises mail, calendar, contacts and tasks, and shows where devices are reviewed and managed: in grommunio Web, in the Admin UI and on the command line.

How it fits together

Component	Role for EAS
nginx	Terminates TLS on port 443 and passes `/Microsoft-Server-ActiveSync` to the grommunio-sync PHP-FPM pool (location shipped in `/usr/share/grommunio-common/nginx/locations.d/grommunio-sync.conf`)
grommunio-sync (PHP-FPM pool `grommunio-sync-pool`, socket `/run/php-fpm/php-grommunio-sync-fpm.sock`)	Implements EAS 2.5 to 16.1: provisioning, folder hierarchy sync, item sync, push (Ping), policies, wipe handshake
gromox-zcore	php-mapi transport; grommunio-sync reads and writes the mailbox through zcore
gromox-http	Information store (exmdb) behind zcore; also answers AutoDiscover at `/Autodiscover/Autodiscover.xml`
Redis	Transient state of running sync threads; feeds `grommunio-sync-top`, the Admin UI *Mobile devices* view and the "last connect" timestamps
grommunio-admin-api	Serves the effective sync policy and the wipe status of a device to grommunio-sync; keeps wipe requests in its database
User store	Persistent per-device sync state in the hidden `GS-SyncState` folder, outside `IPM_SUBTREE`

A device session runs as follows:

1. The device locates the server via AutoDiscover (or you enter the server name manually) and sends requests to `https://mail.example.com/Microsoft-Server-ActiveSync`.
2. nginx forwards the request to grommunio-sync, which authenticates the user through php-mapi against gromox-zcore. Users without the EAS privilege are rejected.
3. On the first contact the device is provisioned: grommunio-sync fetches the effective sync policy for the user from the Admin API and the device acknowledges it (PIN, lock timeout and so on).
4. The device runs FolderSync and Sync; grommunio-sync records the per-device state in the user's store. Between changes the device sits in a Ping request, which is how push mail works.

Because the device state lives inside the mailbox, it travels with the mailbox during backups, migrations and homeserver moves, and it is deleted with the mailbox. Redis holds only what is needed to display live connections.

Prerequisites

- A grommunio installation with working DNS and TLS, for example after the post-installation guide.
- DNS: `mail.example.com` resolves from every network your devices use, and `autodiscover.example.com` points to the same host (A/AAAA or CNAME). Alternatively publish an SRV record `_autodiscover._tcp.example.com`. See autodiscover(7).
- A publicly trusted TLS certificate covering both names. Mobile operating systems refuse untrusted certificates or require manual trust profiles; expired certificates stop synchronisation on every device at once.
- TCP port 443 reachable from the mobile networks. EAS needs no other port.
- Running services: `nginx`, `php-fpm`, `gromox-http`, `gromox-zcore`, Redis and `grommunio-admin-api`.
- A user with a mailbox and, for the first test, a device you can afford to reset. Policies apply during the first sync and a wipe test is destructive.

1. Check the server side

Confirm the host name, the installed components and the services before touching a device.

```
hostname -f
rpm -q grommunio-sync gromox grommunio-admin-api
systemctl is-active nginx php-fpm gromox-http gromox-zcore redis@grommunio grommunio-admin-api
```

Every service should report `active`. Then probe the two HTTPS endpoints from outside the server, without disabling certificate verification. If `curl` rejects the certificate, mobile devices will reject it too.

```
curl -sS -o /dev/null -w '%{http_code}\n' https://mail.example.com/Microsoft-Server-ActiveSync
curl -sS -o /dev/null -w '%{http_code}\n' https://autodiscover.example.com/Autodiscover/Autodiscover.xml
```

Expected result: `401` for both URLs. An unauthenticated request is answered with an authentication challenge, which proves that nginx, the PHP-FPM pool and gromox-http are wired up. A `404`, `502` or `503` points at the nginx configuration, a stopped `php-fpm` or a stopped `gromox-http`.

To test AutoDiscover the way an EAS client does, use gromox-dscli(8) with the `--eas` request schema:

```
PASS='<strong-password>' gromox-dscli --eas -e alice@example.com
```

Expected result: an XML response that names the server. With `-v`, the request and the raw response are printed for inspection.

2. Enable EAS for the user

ActiveSync access is a per-user privilege. Without it, the user cannot log in at the EAS endpoint even with a correct password.

Admin UI: open *Domains*, select the domain, open *Users*, click the user, and on the *Account* tab enable *Allow EAS*. Click *Save* at the bottom of the page. To pre-fill the privilege for newly created users, set it under *Defaults* (globally or per domain); see Administration.

CLI:

```
grommunio-admin user modify alice@example.com --privEas 1
grommunio-admin user query -f username=alice@example.com username status privEas
```

Expected result: the query lists the user with `privEas` set to `1`. Boolean fields accept `0` / `1`, `yes` / `no` and `true` / `false`. `grommunio-admin user show alice@example.com` prints the full record. To withdraw ActiveSync access later, set `--privEas 0`; this blocks new logins but leaves existing device states in the store (see step 6 for cleanup).

The privilege is stored in grommunio, not in LDAP. Users imported from LDAP get it through the defaults or by setting it explicitly as shown. See grommunio-admin user for all fields.

3. Connect a device

Use the account type *Microsoft Exchange* (sometimes labelled *Exchange ActiveSync* or *Exchange*), never *Microsoft 365* or *Outlook.com*, which are tied to Microsoft's cloud.

Setting	Value
E-mail address	`alice@example.com`
Username	`alice@example.com` (the full address; leave any *Domain* field empty)
Password	The user's grommunio password
Server	`mail.example.com` (manual setup only)
Encryption	SSL/TLS, port 443
Data to sync	Mail, Contacts, Calendars, Tasks/Reminders, Notes, as offered by the client

grommunio-sync expects the full e-mail address as login name (`USE_FULLEMAIL_FOR_LOGIN` in `/etc/grommunio-sync/grommunio-sync.conf.php`), which is also what AutoDiscover hands to the device.

With AutoDiscover

Enter the e-mail address and the password and let the device discover the server. The device queries `autodiscover.example.com` (or the SRV record) and receives the ActiveSync URL. Nothing else is required. If a client offers both a sign-in and a manual path, the sign-in path is the AutoDiscover path.

Manual server entry

When AutoDiscover is not published or the client insists on manual data, choose the manual or advanced option and enter server, username and password as in the table above. On iOS and iPadOS this is *Configure Manually* after entering the address; Android mail apps ask for *Server* and *Username* in their *Manual setup*; Outlook for iOS and Android exposes the server field under *Advanced settings* when the account type *Exchange* is chosen. Menu names vary between vendors and versions.

> **Outlook for iOS and Android**
>
> Outlook mobile treats third-party ActiveSync accounts differently from the built-in apps, and its support for them has changed between versions. The reference clients for grommunio are the built-in mail, calendar and contacts apps of iOS and Android.

Shared mailboxes on a phone

grommunio-sync supports impersonation over EAS. To put a shared mailbox on a device with full rights, use the combined login `<shared-mailbox>!<your-address>`, for example `sales@example.com!alice@example.com`, with Alice's own password. AutoDiscover understands this form and the server checks that Alice holds owner rights on the target store. A remote wipe issued against such a session is automatically reduced to an account-only wipe. See the release notes.

> **Caution**
>
> The device receives the sync policy during the first contact. If the policy requires a PIN or encryption, the device enforces it before the first sync completes. Use a device you control for the first tests.

4. Prove synchronisation

On the device, the folder list appears first (FolderSync), followed by the content of the folders (Sync). Send a message to the user and create an appointment on the device; the message should arrive on the device and the appointment should show up in grommunio Web.

On the server, list the user's devices:

```
grommunio-admin user devices alice@example.com list
```

Expected result: one line per device with the columns `ID`, `Device`, `Agent`, `Version` (EAS protocol version), `Last connect` and `Status` (wipe status). `grommunio-admin user devices alice@example.com show <DEVICE-ID>` prints the details recorded by grommunio-sync, including `devicetype`, `devicemodel`, `deviceos`, `useragent`, `firstsynctime`, `lastupdatetime`, `lastconnecttime`, `asversion` and `wipeStatus`.

Watch live connections with the console tool shipped with grommunio-sync:

```
grommunio-sync-top
```

It shows one line per running request with `PID`, `ADDRESS`, `USER`, `COMMAND` (for example `Provision`, `FolderSync`, `Sync`, `Ping`), `TIME`, `AGENT`, `DEVID`, additional information and the EAS version. Type `f:alice` to filter on a user, `l:` to grep the log, `t:` to tail the log, `e:` to tail the error log, `h` for help and `q` to quit. The Admin UI offers the same view under *Mobile devices*, and *Live Status* lists every EAS request together with the other HTTP protocols.

Check the grommunio-sync logs for the user:

```
grep -i 'alice@example.com' /var/log/grommunio-sync/grommunio-sync.log | tail -n 50
tail -n 100 /var/log/grommunio-sync/grommunio-sync-error.log
```

Expected result: provisioning, FolderSync and Sync entries for the device ID and no entries in the error log for this session.

5. Review devices in grommunio Web and the Admin UI

grommunio Web (user self-service). Under *Settings*, *Mobile Devices*, the user sees every device paired with the mailbox: friendly name, operating system, first and last sync, device ID, and on click the synchronised folders, the grommunio-sync version, the negotiated EAS version and the provisioning policy in force. The actions *Wipe Device* (requires the user's password), *Full Resync*, *Remove Device* and *Refresh* are described in Mobile Device Management. Point users to that page and to the clean-restart procedure in the user troubleshooting page before they contact you.

Admin UI, per user. Open the user and switch to the *Mobile devices* tab to see the devices of this mailbox. The tab offers *Remote wipe* and *Cancel remote wipe*. The *Sync policy* tab next to it holds the user-specific policy (see step 7).

Admin UI, server-wide. *Mobile devices* in the drawer is a live table of current connections, refreshed every two seconds, with a text filter and an activity filter. It is the graphical equivalent of `grommunio-sync-top`. Use it to see who is syncing right now, not as an inventory: devices that are idle between Ping requests appear and disappear as their requests come and go. See Administration.

6. Manage devices from the CLI

All device operations hang off `grommunio-admin user devices`. `USERSPEC` is the user name or ID; device IDs come from `list`.

```
grommunio-admin user devices alice@example.com list
grommunio-admin user devices alice@example.com show <DEVICE-ID>
grommunio-admin user devices alice@example.com resync <DEVICE-ID>
grommunio-admin user devices alice@example.com remove <DEVICE-ID>
grommunio-admin user devices alice@example.com wipe --mode account <DEVICE-ID>
grommunio-admin user devices alice@example.com wipe --mode normal <DEVICE-ID>
grommunio-admin user devices alice@example.com wipe --mode cancel <DEVICE-ID>
```

Subcommand	Effect	Takes effect
`list`	Devices of the user with EAS version, last connect time and wipe status	Immediately
`show DEVICE …`	Full device record from the sync state	Immediately
`resync [DEVICE …]`	Marks the device for a full resync: hierarchy first, then all content	On the device's next request
`remove [DEVICE …]`	Deletes the device state from the store and the device's wipe record	Immediately; a device that still has the account configured re-registers and performs a full sync
`wipe --mode MODE DEVICE`	Sets the wipe status: `normal` (whole device), `account` (this account's data only) or `cancel` (withdraw a pending request); default is `normal`	On the device's next authenticated request

`list`, `show`, `resync` and `remove` accept several device IDs; without an ID they act on all devices of the user. `wipe` requires exactly one device ID. Additional examples are in the CLI cookbook.

Use `resync` when items are missing on a device or a folder does not update. Use `remove` for devices that have been retired, or as part of the clean-restart procedure: remove the account from the device, remove the device state, wait a few minutes, recreate the account.

7. Sync policies

Sync policies are the EAS provisioning rules a device must accept before it may synchronise. grommunio applies them in three layers:

1. The server default, visible with `grommunio-admin config get sync.defaultPolicy`.
2. The domain policy: *Domains*, select the domain, *Sync policy* tab. It applies to all users of the domain.
3. The user policy: *Users*, select the user, *Sync policy* tab. It overrides the domain policy for this user.

In both tabs, blue controls mark values that deviate from the inherited policy; grey controls inherit. A policy change is pushed to the devices at their next contact, when grommunio-sync re-provisions them. The Admin UI exposes, among others:

Policy	Purpose
Password required	Device must have a lock PIN or password
Minimum password length	Minimum number of characters
Require alphanumeric password / Minimum password character sets	Complexity requirements
Allow simple passwords	Permit repeating or sequential PINs
Password expiration (days)	Force periodic PIN changes

Policy	Purpose
Number of failed login attempts allowed	Device wipes itself after that many wrong entries (client-dependent)
Inactivity (seconds) before device locks itself	Auto-lock timeout
Require encryption on storage card	Encryption of removable storage
Device encryption, camera, storage card, Wi-Fi, Bluetooth, browser, unsigned apps, attachments and maximum attachment size, mail and calendar age filters	Feature restrictions and sync limits from the EAS policy set

A common baseline for company devices is: password required, minimum length 6, lock after 300 to 900 seconds of inactivity, 10 failed attempts, device encryption required. Everything else is client-dependent: current iOS and Android releases honour PIN, complexity, lock timeout, encryption and camera rules; many other keys date from older device generations and are silently ignored. Validate a policy with a real device of each platform you support. Test clients that only speak the protocol do not enforce anything.

Two settings in the grommunio-sync configuration file `/etc/grommunio-sync/grommunio-sync.conf.php` (linked as `/usr/share/grommunio-sync/config.php`) govern provisioning as a whole: `PROVISIONING` (enabled by default) switches policy enforcement on, and `LOOSE_PROVISIONING` (disabled by default) admits devices that cannot provision while still enforcing policies on devices that can. Leave both at their defaults unless you must support legacy clients.

> **Note**
>
> Policies are not a substitute for revoking access. A lost device that never connects again never receives a new policy or a wipe. Change the password and remove the EAS privilege in that case.

8. Remote wipe

> **Danger**
>
> A remote wipe is destructive and cannot be undone. In `normal` mode, iOS devices and many Android devices perform a factory reset that removes personal photos, apps and settings. Verify the device ID and the mode before issuing it, and never test on a device that holds data you need.

Mode	Admin UI label	Result on the device
`normal`	Wipe all data	Full device wipe, vendor-dependent; equivalent to a factory reset on most platforms
`account`	Wipe only data related to this account	Removes the grommunio account and its mail, contacts and calendar items; other data stays
`cancel`	Cancel remote wipe	Withdraws a pending request that the device has not acknowledged yet

A wipe can be requested from three places: by the user in grommunio Web (*Wipe Device*, password required), by an administrator in the Admin UI (*Remote wipe* on the user's *Mobile devices* tab) and on the CLI with `wipe --mode`. In every case the request is stored as a pending status; the device receives the command in the response to its next authenticated request, acknowledges it and executes it. `list` shows the current status in the `Status` column.

Recommended procedure for a lost or stolen device:

1. Identify the device with `list` and `show`: match the model, the user agent and the last connect time against what the user reports.
2. Choose the mode. Prefer `account` on personally owned devices; use `normal` only where company policy and the user's consent cover a full reset.
3. Issue the wipe and leave the account credentials valid until the device has acknowledged the request. A device that can no longer authenticate is never told to wipe.
4. Watch `list` for the status change and the grommunio-sync log for the device's acknowledgement. Devices that are switched off, offline or have already had the account removed stay in pending state indefinitely; cancel the request if the device turns up.
5. After the acknowledgement, remove the device state with `remove`, change the user's password (`grommunio-admin passwd alice@example.com`) and, if appropriate, withdraw the EAS privilege.

Wipes issued against an impersonated shared-mailbox session are reduced to `account` mode by grommunio-sync, so a personal device is never reset because of a shared mailbox.

Verification checklist

Check	How	Expected result
DNS	`dig +short mail.example.com` and `dig +short autodiscover.example.com`	Both resolve to the grommunio host from the client networks
TLS	`curl -sS https://mail.example.com/` without `-k`	No certificate error; certificate covers both names
Services	`systemctl is-active nginx php-fpm gromox-http gromox-zcore redis@grommunio grommunio-admin-api`	All `active`
Endpoint	`curl -sS -o /dev/null -w '%{http_code}\n' https://mail.example.com/Microsoft-Server-ActiveSync`	`401`
AutoDiscover	`PASS='…' gromox-dscli --eas -e alice@example.com`	XML response naming the server
User privilege	`grommunio-admin user query -f username=alice@example.com privEas`	`1`
Provisioning	Add the account on a device	Device accepts the policy and, if required, asks for a PIN
Hierarchy and content	Device shows folders; a test mail and a test appointment sync both ways	Items visible on the device and in grommunio Web
Device registered	`grommunio-admin user devices alice@example.com list` and grommunio Web *Mobile Devices*	Device listed with a recent last connect time
Live view	`grommunio-sync-top` or Admin UI *Mobile devices* while the device syncs	Request lines for the user appear
Resync	`resync`, then wait for the next request	Device performs a full resync; items reappear
Remove	`remove`, with the account removed from the device first	Device disappears from the list and does not return
Policy change	Change the lock timeout in the domain or user policy	Device is re-provisioned at its next contact and applies the new value
Restart	`systemctl restart php-fpm nginx`	Devices reconnect and continue syncing
Logs	`/var/log/grommunio-sync/grommunio-sync-error.log`	No errors for the tested session

Troubleshooting

Symptom	Likely cause	What to check/fix
Device cannot find the server during setup	AutoDiscover DNS missing or not published in the client's DNS view; certificate does not cover `autodiscover.example.com`	Resolve `autodiscover.example.com` from the client network; test with `gromox-dscli --eas`; enable `oxdisco_request_logging` in `/etc/gromox/gromox.cfg` temporarily; fall back to manual server entry
Device reports a certificate error	Self-signed, incomplete chain or expired certificate	`curl` without `-k` from outside; install a publicly trusted certificate with the full chain
Account setup fails with wrong credentials although the password is correct	Username not the full address, `Domain` field filled, EAS privilege not set, user suspended	Use `alice@example.com` as username with an empty domain; `grommunio-admin user query -f username=alice@example.com status privEas`; `grommunio-admin user login alice@example.com` to test authentication
Endpoint always returns `401` for the device, `grommunio-sync.log` shows no login	Request never reaches PHP or authentication backend down	`systemctl status php-fpm gromox-zcore`; `/var/log/nginx/nginx-sync-error.log`; `/var/log/grommunio-sync/grommunio-sync-fpm.log`
Repeated `401` / `403` in `/var/log/nginx/nginx-sync-access.log` for one user	Stale password on a device after a password change, or privilege withdrawn	Identify the device by IP and user agent in `grommunio-sync-top`; update or remove the account on the device
Device not listed although it syncs	Looking at the wrong user or the Admin UI live view instead of the device list	`grommunio-admin user devices <user> list`; grommunio Web *Mobile Devices*, *Refresh*
Sync stuck, no new mail, last connect time old	Device offline, push (Ping) blocked by the network, client not provisioned, broken sync state	Check `Last connect` in `list`; watch `grommunio-sync-top` for Ping requests from the device; `resync`; as a last resort the clean-restart procedure (remove account, `remove`, wait, re-add)
Items missing or duplicated, flags not updating	Corrupted device state or change-number problems in the mailbox	`resync` first; if the mailbox itself shows ICS problems see `cgkreset` in gromox-mbop(8)
`resync` has no visible effect	The device has not connected since	Wait for the next request or trigger a manual sync on the device
Device reappears after `remove`	The account is still configured on the device	Remove the account on the device first, then `remove` the state
Policy not applied on the device	Device has not reconnected; client does not implement the policy key; `PROVISIONING`	Force a sync on the device; test with a native mail client; confirm `PROVISIONING` in `/etc/grommunio-sync/grommunio-`

Symptom	Likely cause	What to check/fix
	disabled	`sync.conf.php`
Wipe stays pending	Device offline, switched off, or the account was removed before the wipe was delivered; password changed before acknowledgement	Keep credentials valid until acknowledgement; `wipe --mode cancel` if the device turns up; otherwise rely on password change and privilege removal
Shared mailbox login rejected	Impersonation form wrong or missing owner rights	Login must be `shared@example.com!alice@example.com`; grant owner permission on the shared store
`grommunio-sync-top` aborts with `Permission denied` on `config.php`	The wrapper switches to the `grosync` user, which cannot read `/etc/grommunio-sync/grommunio-sync.conf.php` (mode 640, group `grosync`)	Check `id grosync` and the file's group; use the Admin UI *Mobile devices* view or `grommunio-admin user devices … list` in the meantime

Operating notes

Logs. grommunio-sync writes to files, not to the journal:

File	Content
`/var/log/grommunio-sync/grommunio-sync.log`	Request log: user, device ID, command, result
`/var/log/grommunio-sync/grommunio-sync-error.log`	Errors and warnings only
`/var/log/grommunio-sync/grommunio-sync-fpm.log`	PHP-FPM output of the grommunio-sync pool
`/var/log/nginx/nginx-sync-access.log`, `/var/log/nginx/nginx-sync-error.log`	HTTP status codes and proxy errors for `/Microsoft-Server-ActiveSync` (separate from the grommunio Web logs)

The default verbosity is `LOGLEVEL_INFO` (`LOGLEVEL` in `/etc/grommunio-sync/grommunio-sync.conf.php`). Raise it only for the duration of an investigation; the debug and WBXML levels record message content. `/etc/logrotate.d/grommunio-sync.lr` rotates the files under `/var/log/grommunio-sync/` once they exceed 4 MB, keeps four generations and deletes archives after a year. gromox-zcore and gromox-http log to the journal (`journalctl -u gromox-zcore -u gromox-http`); a general log overview is in Troubleshooting.

Monitoring. Monitor technical health rather than device inventories: HTTP reachability and the `401` challenge of the endpoint, certificate expiry, the share of `401` / `403` responses in `/var/log/nginx/nginx-sync-access.log`, growth of `grommunio-sync-error.log`, saturation of the PHP-FPM pool (each Ping request holds a worker) and stale `Last connect` times of devices that should be active. Device IDs, IMEIs and user agents are personal data; keep exports of device lists to what your policy requires.

State and backups. Device states are part of the user store and are therefore included in mailbox backups and in full exports with `gromox-exm2mt -ar /`; an export restricted to `IPM_SUBTREE` omits them. Redis contents are transient and need no backup. See Operations for the general backup procedure.

Housekeeping. Review device lists periodically and `remove` states of devices whose last connect time is months old. When a user leaves, withdraw the EAS privilege, wipe the account data if required, then delete the user; the states go with the mailbox.

Updates and high availability. Update grommunio-sync with `zypper` together with the rest of the stack and re-run the endpoint check and one device sync afterwards. Because grommunio-sync is stateless PHP behind nginx and the device state lives in the store, `/Microsoft-Server-ActiveSync` can be load-balanced across web nodes; the architecture and high availability pages contain the routing examples.

Related pages

- Mobile Device Management in grommunio Web
- Administration: users, mobile devices, sync policies
- grommunio-admin user and the CLI cookbook
- autodiscover(7), autodiscover(4gx) and gromox-dscli(8)
- Architecture: Exchange ActiveSync workflow
- Troubleshooting and the user troubleshooting page
- gromox-mbop(8)
- Release notes

Backup, disaster recovery and single-mailbox restore

This guide is the procedural complement to the Backup & Disaster Recovery section of the operations manual. That section lists the supported snapshot mechanisms and the backup artefacts of every grommunio component; this page shows how to turn those artefacts into a **logical backup set**, how to **rebuild a server from it** on a fresh appliance, and how to **restore a single mailbox** with the Gromox mailbox transfer tools.

At the end you will have a repeatable backup procedure, a tested full disaster recovery (DR) runbook, and two verified ways to get a single mailbox back: into a different mailbox, or into the original one.

How it fits together

grommunio stores its state in three places that must be captured **together**, from the same point in time:

Data	Location	Backup method
Directory data: domains, users, aliases, groups, roles, settings kept by grommunio-dbconf	MariaDB database `grommunio` (plus `grofiles`, `grochat`, `groarchive` if those components are installed)	`mysqldump` or a MariaDB backup agent
Mailbox stores: MAPI object databases (`exmdb/exchange.sqlite3`), IMAP index (`midb.sqlite3`), message content and attachments	`/var/lib/gromox`	File-level backup (`tar` / `rsync`) or filesystem snapshot
Configuration and secrets: Gromox daemon configuration and database credentials, Admin API and Admin Web configuration, TLS material, web server and MTA configuration	`/etc/gromox` , `/etc/grommunio*` , `/etc/nginx` , `/etc/postfix` , `/etc/letsencrypt` (if used)	File-level backup

A mailbox is only usable when its store directory under `/var/lib/gromox` and its row in the `grommunio` database match, and the daemons can only open the database when `/etc/gromox/mysql_adaptor.cfg` carries the credentials that the restored MariaDB instance accepts. Restore all three from the same backup run.

Two backup styles exist, and they complement each other:

- **Consistent snapshot.** A snapshot of the whole VM or of the filesystem holding `/var/lib/gromox` and the MariaDB data directory is consistent by construction and is the fastest way to roll a server or one mailbox back. The supported mechanisms are listed in Backup & Disaster Recovery; the appliance ships gromox-snapshot(8) for Btrfs/reflink snapshots of the mailbox storage.
- **Logical backup set.** A database dump, a tar archive of the data and configuration directories, and optionally per-mailbox exports in the Gromox Mailbox Transfer (GXMT) format. A logical set is portable, can be verified with checksums, and is the only form that lets you import one mailbox into a *different* mailbox.

For the export/import tools used below, the man pages are gromox-exm2mt(8) (export, an alias of gromox-export(8)) and gromox-mt2exm(8) (import, an alias of gromox-import(8)). The format itself is described on the Mailbox transfer format page.

> ⚠**Replication is not backup**
>
> DRBD, storage replication and cluster failover replicate deletions and corruption faithfully. A high-availability setup still needs the independent backup described here; see its Backup and restore section for the additional cluster artefacts (Corosync, Pacemaker CIB, DRBD configuration).

Prerequisites

- Root shell access on the grommunio server; all commands below run as `root`.
- A backup target outside the server (mounted share, object storage, backup server) with enough space for the database dump, the `/var/lib/gromox` tree and the configuration archive.
- For a full DR test: a second, freshly installed grommunio Appliance of the **same package versions** as the source (compare `rpm -qa` output for the `grommunio*` , `gromox*` , `mariadb*` , `nginx` , `postfix` and `redis*` packages). Restoring a database dump and SQLite stores across different major versions is not covered here.
- For single-mailbox restores: the restore target mailbox exists (see step 4).

Record the versions of the source system with the backup, so that a restore host can be provisioned to match:

```
cat /etc/os-release
grommunio-admin version
rpm -qa | grep -E "^(grommunio|gromox|mariadb|nginx|postfix|redis)" | sort
systemctl --failed
```

1. Create a logical backup set

Work in a dedicated directory on the backup target and give every run its own timestamped subdirectory, so that database dump, file archives, mailbox exports and checksums of one run stay together.

1.1 Dump the databases

Dump all databases in a single consistent transaction, including stored routines, triggers and events. `--add-drop-database` lets the dump replace the databases that a fresh appliance already created during setup:

```
mysqldump --all-databases --single-transaction --routines --triggers --events --add-drop-database > grommunio-mysql-all.sql
```

This is the same command the operations manual recommends under Database backup. Because the dump includes the MariaDB system database, it also carries the database accounts and grants that `/etc/gromox/mysql_adaptor.cfg` and the Admin API configuration refer to; that is what makes the restore in step 2 work without re-creating database users.

1.2 Archive the mailbox stores and the configuration

Archive `/var/lib/gromox` and the configuration directories. Keep extended attributes and ACLs, because the appliance relies on group-writable data directories shared between the `gromox` daemons and the Admin API:

```
tar --xattrs --acls -czf gromox-varlib.tar.gz -C /var/lib gromox
tar --xattrs --acls -czf grommunio-etc.tar.gz /etc/grommunio* /etc/gromox /etc/nginx /etc/postfix
```

Add `/etc/letsencrypt` to the second archive if the appliance obtains its certificates from Let's Encrypt, and `/etc/php8/fpm/php-fpm.d` if you changed PHP-FPM settings (see the artefact list in Backup & Disaster Recovery). If grommunio Files, Chat or Archive are installed, include their data directories from that list in the same run.

> **Consistency of the file archive**
>
> `tar` reads the store files while the daemons keep writing to them. For a backup that is guaranteed consistent, run the archive step from a filesystem snapshot (for example a gromox-snapshot subvolume, an LVM or a hypervisor snapshot) instead of the live directory, or stop the Gromox services for the duration of the archive as in step 2.1. Mailboxes are transactional SQLite databases, so a `tar` of a live system is usually recoverable, but a snapshot removes the doubt.

1.3 Export mailboxes in GXMT format (optional)

A GXMT export is a logical copy of one mailbox that can later be imported into *any* mailbox, which the file-level archive cannot do. Export the mailboxes you may need to restore individually, or all of them from a loop over `grommunio-admin user query`. The flags matter:

Flag	Meaning (from gromox-export(8))
`-u alice@example.com`	Source mailbox.
`-a`	Also export Folder Associated Information (FAI, hidden messages such as views and settings).
`-r`	Recurse into subfolders (GXMT only).
`-s`	Mark objects for **splicing**: on import, objects that came from a well-known folder (Inbox, Sent Items, Calendar, ...) are placed into the *target mailbox's* folder of the same kind instead of a newly created folder tree.
`/`	Folder specification; the MAPI root, which covers the whole store. `IPM_SUBTREE` exports only what Outlook and grommunio Web show.

Create both variants of a full export. The plain export is a faithful archive; the splice export is the one you import into another mailbox in step 4:

```
gromox-exm2mt -u alice@example.com -ar / > alice-full.mt
gromox-exm2mt -u alice@example.com -ars / > alice-full-splice.mt
```

The `-s` flag only changes the placement instructions written to the stream header; the message content is identical. If you want to keep just one file, keep the splice export.

1.4 Record checksums

Write SHA-256 checksums for the whole set and copy the checksum file together with the artefacts:

```
sha256sum * > SHA256SUMS
```

Verify the set on the restore side with `sha256sum -c SHA256SUMS` before you change anything on the target. Expected result: every line reports `OK`.

Before moving on, confirm that a GXMT export parses. Without `-u`, `gromox-mt2exm` only reads the stream and, with `-t`, prints the folder and message summary to stderr instead of importing anything:

```
gromox-mt2exm -t < alice-full-splice.mt
```

2. Full disaster recovery onto a fresh appliance

The scenario: the original server is lost. You install a new grommunio Appliance with the same package versions, do **not** create domains or users on it, and fill it entirely from the backup set. Copy the backup set to the new host and verify the checksums first.

2.1 Stop the services

Stop everything that reads the database or the mailbox stores, consumers first, then MariaDB:

```
systemctl stop postfix gromox-delivery gromox-delivery-queue gromox-http gromox-zcore gromox-midb gromox-imap gromox-pop3 gromox-e
systemctl stop mariadb
```

If additional components run on the host, stop their services as well before restoring their data, for example `grommunio-antispam`, `grommunio-chat`, `grommunio-archive` and `grommunio-archive-smtp`. Files and Office run inside `php-fpm` / `nginx` and stop with them.

2.2 Restore the files

Unpack the mailbox stores under `/var/lib` and the configuration archive relative to `/`, again preserving extended attributes and ACLs:

```
tar --xattrs --acls -xzf gromox-varlib.tar.gz -C /var/lib
tar --xattrs --acls -xzf grommunio-etc.tar.gz -C /
```

Restoring `/etc/gromox`, `/etc/grommunio*` and `/etc/nginx` also brings back the TLS certificate and key referenced by the web server configuration, the database credentials in `/etc/gromox/mysql_adaptor.cfg`, and the Admin API configuration. If the new host has a different hostname or IP address, adjust Postfix and nginx configuration afterwards.

2.3 Check ownership and permissions

On the appliance the system users and groups exist with consistent IDs, so a tar restore normally yields the correct owners. If backup and restore hosts were built differently, numeric UIDs/GIDs stored in the archive may map to other names, and the ACLs restored by `--acls` then refer to the wrong accounts. Check the result before starting any daemon:

```
getent passwd gromox
getent group gromox
getent group gromoxcf
namei -l /var/lib/gromox
find /var/lib/gromox -maxdepth 2 -printf '%u:%g %m %p\n' | head -50
ls -la /etc/gromox
```

Expected result: `getent` resolves the system user `gromox` (the Gromox daemons) and `grommunio` (the Admin API), and the groups `gromox` and `gromoxcf`; `grommunio` is a member of `gromox`, and `gromoxcf` contains `gromox` and `grommunio`. `/var/lib/gromox` and everything below it belongs to user `gromox` (or `grommunio`) and group `gromox`, mode `drwxrwx---`, so that daemons and Admin API can both write to it. `/etc/gromox` belongs to group `gromoxcf`; the configuration files inside must be readable by that group (`pam.cfg` may be world-readable). Compare with `ls -ld /var/lib/gromox` output recorded on the source system and fix deviations with `chown`/`chmod` on the affected trees before continuing.

2.4 Restore the database

Start MariaDB alone and replay the dump. Because the dump contains `DROP DATABASE`/`CREATE DATABASE` statements, the empty databases created by the appliance setup are replaced:

```
systemctl start mariadb
mysql < grommunio-mysql-all.sql
```

Restart MariaDB once afterwards so that the restored account and grant tables are reloaded, then confirm that the credentials from `/etc/gromox/mysql_adaptor.cfg` work:

```
systemctl restart mariadb
mysql --user=grommunio --password --database=grommunio -e "SELECT COUNT(*) FROM users;"
```

Take the username, password and database name from the `mysql_username`, `mysql_password` and `mysql_dbname` directives in `/etc/gromox/mysql_adaptor.cfg` (mysql_adaptor(4gx)). Expected result: the user count of the source system.

2.5 Start the services

Start the stack in dependency order: cache and PHP first, then the Admin API and web server, the Gromox core daemons, the protocol front ends, and finally delivery and the MTA:

```
systemctl start redis@grommunio php-fpm nginx grommunio-admin-api gromox-event gromox-timer gromox-midb gromox-zcore gromox-http g
systemctl --failed
```

Expected result: `systemctl --failed` lists no units. The same order is used by the Pacemaker service group in the high-availability reference architecture.

2.6 Verify the restored server

Check from the command line that domains, users and mailbox content are back:

```
grommunio-admin domain list
grommunio-admin user list
gromox-mbop -u alice@example.com ping
gromox-exm2mt -u alice@example.com -ar / > alice-after-dr.mt
gromox-mt2exm -t < alice-after-dr.mt
```

Expected result: the domain and user lists match the source system, `ping` returns without error (the store opened), and the parsed export lists the folders and messages you expect. Then sign in to the Admin UI at `https://mail.example.com:8443/` and confirm the users under *Domains* and *Global users*, and sign in to grommunio Web at `https://mail.example.com/` (which redirects to `/web/`) as a restored user to see the mailbox content. If Let's Encrypt or a public-CA certificate was restored, the browser should show no certificate warning.

3. Restore a mailbox from a filesystem snapshot

When the storage under `/var/lib/gromox` is snapshotted (gromox-snapshot, LVM, ZFS, hypervisor or storage-array snapshots as listed in Backup & Disaster Recovery), rolling a single mailbox back is a directory copy. Find the mailbox directory first:

```
grommunio-admin user query -f username=alice@example.com username maildir
```

Then make the information store close the mailbox, replace the directory contents from the snapshot, and reopen it:

```
gromox-mbop -u alice@example.com ( freeze ) ( unload )
```

Copy the mailbox directory from the snapshot over the live directory (for example with `rsync -HPavS`, as in File-based backup), preserving ownership, then:

```
gromox-mbop -u alice@example.com thaw
systemctl restart gromox-http gromox-midb
```

`freeze` halts new operations on the mailbox, `unload` closes its SQLite files, and `thaw` lifts the freeze (gromox-mbop(8)). The restart of `gromox-http` and `gromox-midb` invalidates runtime caches, as the operations manual advises after a snapshot restore. `unload` fails while a client still holds a notification channel on the mailbox; make sure the user is signed out of Outlook, grommunio Web and mobile devices before you start. Restoring a store directory into a *different* mailbox's directory is not supported by this method: store identity is embedded in the SQLite database. Use the GXMT import in step 4 for that.

4. Restore a single mailbox into a different mailbox

Use this when a user needs old content next to the current mailbox, when a departed user's mailbox has to be handed to a colleague, or when you want to inspect a backup without touching production data. The source is the splice export from step 1.3.

4.1 Create the target mailbox

The target can be an existing mailbox or a new one. Create a new one with the CLI (grommunio-admin user, grommunio-admin passwd) or in the Admin UI:

```
grommunio-admin user create restore-alice@example.com
grommunio-admin passwd restore-alice@example.com
```

4.2 Import the splice export

Import the stream into the target. `gromox-mt2exm` reads GXMT from standard input and needs only the target mailbox:

```
gromox-mt2exm -u restore-alice@example.com < alice-full-splice.mt
systemctl restart gromox-http gromox-midb gromox-zcore
```

Add `-t` to print each folder and message as it is processed, and `-c` to continue after an error on a single message instead of aborting; both are documented in gromox-import(8).

Restart `gromox-http`, `gromox-midb` and `gromox-zcore` after the import: the import tools write to the store through the information store engine, but the IMAP index (`midb`) and the grommunio Web/PHP-MAPI layer (`zcore`) keep their own runtime state that must be refreshed before the imported items appear everywhere. The source mailbox is untouched by the import.

4.3 Why the export must carry `-s`

The GXMT stream header contains a **folder map** that tells the importer where to put each exported folder:

- With `-s`, every well-known folder of the source store (root, IPM subtree, Inbox, Sent Items, Deleted Items, Calendar, Contacts, and so on) is mapped onto the target store's folder of the same kind. Nothing is created for them; their messages go straight into the target's

Inbox, Sent Items, etc. User-created subfolders are created below the mapped parent, and a subfolder whose name already exists at that place is reused. The result is a normal mailbox layout with the restored items merged in.

- Without `-s`, the map is empty and the exported root folder is "unanchored". The importer places unanchored objects in the anchor folder given with `-B`, which defaults to **Drafts** for private stores. Exporting `/` and importing without `-s` therefore recreates the *entire* source hierarchy (root, IPM subtree, Inbox, …) as a new folder tree underneath Drafts: the content arrives, but as a nested copy of the whole store rather than in the user's real folders. Without `-s` the importer also refuses to create a folder that already exists at the anchor unless `-x` is given.

Both behaviours are described in the option lists of gromox-export(8) (`-s`) and gromox-import(8) (`-B`, `-x`). For a restore that should look like the original mailbox, always export with `-ars /` and import without `-B`.

> **No duplicate detection**
>
> `gromox-mt2exm` does not check whether a message already exists in the target. Every message in the stream becomes a new message. Importing the same stream twice, or importing into a mailbox that already holds the same items, produces duplicates.

4.4 Verify

Sign in to grommunio Web as the target user and confirm that the restored messages sit in Inbox, Sent Items and the user-created folders, not under Drafts. From the shell, export the target and compare the folder and message summary with the source export:

```
gromox-exm2mt -u restore-alice@example.com -ar / > restore-alice-check.mt
gromox-mt2exm -t < restore-alice-check.mt
```

5. Restore a single mailbox into the original mailbox

Two options exist, depending on whether the mailbox should be rolled back as a whole or specific content re-imported.

Option A: roll the store back from a snapshot or file backup

Follow step 3 with the mailbox directory from the snapshot or from the `gromox-varlib.tar.gz` archive. This replaces the whole store, including folder structure, IMAP index and all settings stored in the mailbox, with the backed-up state. Content created after the backup is lost.

Option B: clear the mailbox and re-import the splice export

Importing a splice export into the original mailbox merges the backup into whatever is there and, because there is no duplicate detection, doubles every message that still exists. To avoid that, empty the mailbox first with gromox-mbop(8) `emptyfld`, then import:

```
gromox-mbop -u alice@example.com emptyfld -R -a IPM_SUBTREE
gromox-mbop -u alice@example.com purge-datafiles
gromox-mt2exm -u alice@example.com < alice-full-splice.mt
systemctl restart gromox-http gromox-midb gromox-zcore
```

`emptyfld -R` hard-deletes the messages of `IPM_SUBTREE` and, recursively, of every folder below it; `-a` includes the FAI messages that the `-a` export captured, so that view settings are not duplicated either. The folders themselves stay in place, which is what the splice import expects: well-known folders are reused, and user-created folders with matching names are reused as well. `purge-datafiles` removes the attachment and content files that the deleted messages referenced. Do not add `--nuke-folders`: it would delete the well-known folders below `IPM_SUBTREE`.

> **Danger**
>
> `emptyfld -R` without `--soft` is a hard deletion that cannot be undone from within the mailbox. Make sure the splice export you are about to import is complete (`gromox-mt2exm -t < file.mt`) and its checksum matches before you clear the mailbox. Have the user sign out of all clients first.

Option B keeps only what was in the export; items stored outside `IPM_SUBTREE` (search folders, shortcuts, grommunio-sync states) are neither deleted nor re-imported by this sequence. Clients that hold cached copies (Outlook Cached Mode, mobile devices) should be resynchronised afterwards; for ActiveSync devices use `grommunio-admin user devices alice@example.com resync`.

Verification checklist

Area	Check	Expected result
Backup set	`ls` of the run directory	Database dump, `gromox-varlib.tar.gz`, `grommunio-etc.tar.gz`, GXMT exports and `SHA256SUMS` are present and non-empty
Integrity	`sha256sum -c SHA256SUMS` on the restore host	Every artefact reports `OK`
Export validity	`gromox-mt2exm -t < file.mt`	Folder and message summary is printed, no error, exit status 0
Full DR: services	`systemctl --failed` after step 2.5	No failed units
Full DR: directory	`grommunio-admin domain list`, `grommunio-admin user list`	Same domains and users as the source
Full DR: mailboxes	`gromox-mbop -u <user> ping`, export and `-t` parse	Store opens; folders and messages match the source export
Full DR: web	Admin UI at `https://<FQDN>:8443/`, grommunio Web at `https://<FQDN>/`	Users visible in the Admin UI; a restored user sees mail in grommunio Web; no certificate warning
Ownership	`namei -l /var/lib/gromox`, `ls -la /etc/gromox`	Owners and groups as in step 2.3
Single-mailbox restore	Sign in as the target user	Restored items in Inbox, Sent Items and the original subfolders; nothing nested under Drafts
Source untouched	Export the source mailbox again	Unchanged folder and message summary
Secret handling	Review backup location and shell history	Backup set readable only by the backup operator; no credentials left in shell history or documentation

Troubleshooting

Symptom	Likely cause	What to check/fix
Imported mail appears as a folder tree underneath Drafts	Export was made without `-s`; unanchored root landed in the `-B` default folder	Re-export with `gromox-exm2mt -u <user> -ars /`, delete the nested tree in the target, import again
`gromox-mt2exm` aborts with a "folder already exists" error	Non-splice import into a mailbox that already has that folder	Use a splice export, or add `-x` to reuse existing folders
`Cannot satisfy splice request`	Splice export from a private store imported into a public store or vice versa	Choose a target of the same store type (`-u user@domain` for private, `-u @domain` for public)
Every message exists twice after an import	Stream imported twice or imported into a mailbox that still held the items	There is no duplicate detection; clear the mailbox first (step 5, option B) or import into an empty target
Gromox daemons fail to start after DR; logs show MariaDB access denied	Restored `/etc/gromox/mysql_adaptor.cfg` and the restored `mysql` system database disagree, or MariaDB was not restarted after the import	Restart `mariadb`, then test the credentials from `mysql_adaptor.cfg` with the `mysql` client as in step 2.4
Admin UI shows no domains after DR	Dump restored before the appliance's own empty database was dropped, or the wrong dump file used	Re-run `mysql < grommunio-mysql-all.sql` (the dump carries `--add-drop-database`), check `SELECT COUNT(*) FROM domains;`
Mailbox cannot be opened; "permission denied" in the `gromox-http` log	UID/GID mismatch after the tar restore	Compare `getent passwd gromox` on source and target, fix owners with `chown -R` per step 2.3
`gromox-mbop ... unload` fails	A client still holds a notification channel on the mailbox	Sign the user out of all clients (or stop `gromox-http`), retry
IMAP clients or grommunio Web do not show imported items	`midb` / `zcore` still hold pre-import state	`systemctl restart gromox-http gromox-midb gromox-zcore`
`mysqldump` fails with access denied	Run from an account without MariaDB root access	Run as `root` on the appliance, or pass `--user` / `--password` of a MariaDB account with full read access
Restored mailbox reports inconsistencies	Store files copied while the source was writing	Run `gromox-mbck` on the store (gromox-mbck(8)); take future file backups from a snapshot

Operating notes

- **Test restores, not backups.** Schedule a full DR rehearsal onto a throw-away appliance and a single-mailbox restore into a test mailbox at a fixed interval, and record the elapsed time against your recovery time objective.
- **Keep secrets out of the shell history.** Use `mysql --password` (prompt) rather than `--password=<value>`, or a client options file with restrictive permissions. The configuration archive contains database credentials and TLS private keys: encrypt the backup set at rest and

restrict access to it as tightly as to the server itself.

- **Store checksums separately.** Keep `SHA256SUMS` (or a signed copy) with the backup catalogue as well as next to the artefacts, so that a tampered or truncated artefact is noticed before a restore depends on it.
- **Version pinning.** Record the package list of every backup run. A restore host must match the MariaDB and Gromox versions of the dump and store files, or be upgraded from that version by the normal update path afterwards.
- **Snapshots plus logical backups.** Use snapshots for fast rollback of the server or of a whole mailbox; keep a logical set for portability and cross-mailbox restores. `gromox-snapshot.timer` runs hourly once enabled (`systemctl enable --now gromox-snapshot.timer`); retention is set in `/etc/gromox/snapshot.cfg`, which you create if it does not exist (gromox-snapshot(8)).
- **Other components.** grommunio Files, Chat, Archive and Antispam keep their own databases and data directories; include them in the same backup run as listed in Backup & Disaster Recovery.

Related pages

- Operations: Backup & Disaster Recovery
- High availability: Backup and restore
- Common administration tasks
- Gromox CLI utilities
- grommunio-admin user
- gromox-export(8) / gromox-exm2mt(8)
- gromox-import(8) / gromox-mt2exm(8)
- gromox-mbop(8)
- gromox-snapshot(8)
- Mailbox transfer format
- Database check (SQLite recovery)
- Migration overview