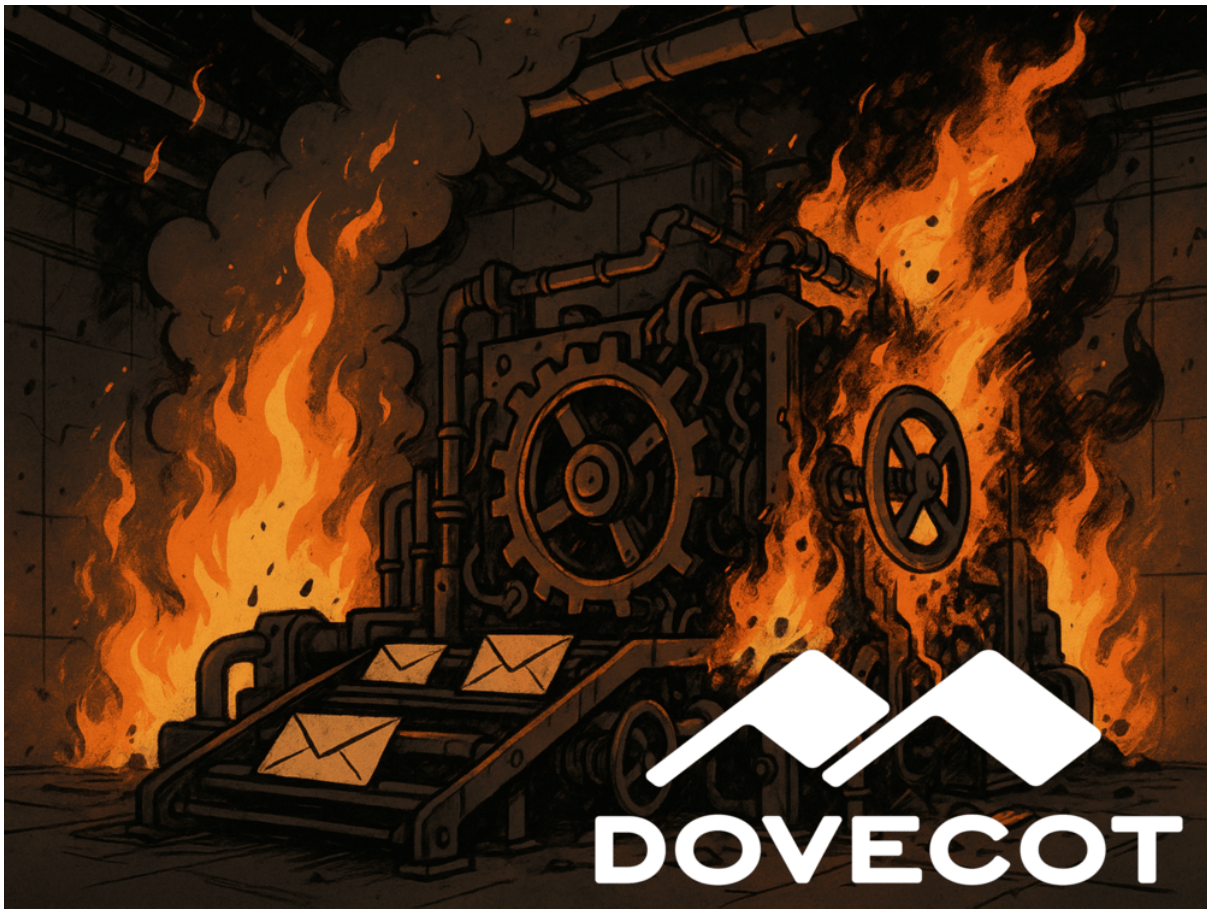
Breaking Changes

Upgrading Dovecot 2.3 to 2.4 in Debian Stable

Willem L. Middelkoop

June 4, 2025



Last week I ran into unexpected trouble during a routine maintenance procedure on my company's email infrastructure. Processing thousands of emails each day, we use Dovecot to give our clients access to their messages. That all came to a grinding halt when I updated to version 2.4 which features breaking changes... oh dear!

TL;DR: Upgrading Dovecot 2.3 → 2.4 on Debian

- **Dovecot 2.4 introduces breaking changes**: Configuration files from 2.3 are incompatible. The software will fail to start unless you rewrite them.
- **New config syntax and variable system**: Parameters like ‘ssl_cert‘, ‘mail_location‘, and ‘%d‘ have changed format and meaning.

- **No automated migration path**: Manual review and rewriting of configuration is required—especially if using SQL backends, Sieve, or custom auth.
- **Recommendation**: Read the official upgrade guide thoroughly before upgrading, and test configs in isolation.

Walk in the park

Pending some security updates and to resolve some compatibility issues, I took the decision to update some packages on a Debian GNU/Linux server that is responsible for my company's email services. This particular distribution of GNU/Linux is known for its stability and reliability. There was no reason to expect any trouble and I started the upgrade expecting it to finish in mere minutes. Just ”a walk in the park”, what could possibly go wrong..?

Requiring 'dovecot_config_version'

Most of the server came back online as expected almost immediately, disrupting services so little that even the uptime monitor alarm didn't pick up any outage - except for Dovecot. This particular type of software is responsible for IMAP and POP3 access to mailboxes. It fits as a finely tuned cog in a larger setup where other programs like Postfix rely upon its working to accept incoming email. Dovecot's failure to start caused a series of problems.

The first thing you do as a server administrator is to check the server daemons ('systemctl status dovecot') followed by inspecting the logs, e.g. by running 'journalctl -u dovecot'. That quickly revealed that Dovecot refused to start because a config parameter was missing: 'dovecot_config_version'.

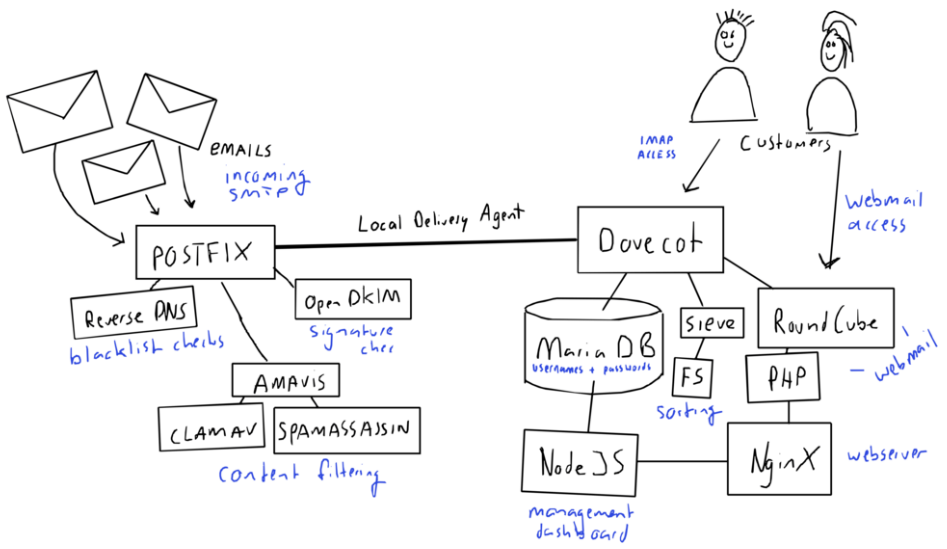
Simply add the parameter to the config at '/etc/dovecot/dovecot.conf' you might think? Well... NO. It turns out that Dovecot's creators put the error in for a good reason: the new 2.4 version of Dovecot is incompatible with previous 2.3 configuration files; meaning you'll need to rewrite many different parameters. It is not just the config variable names, it also introduces an entirely new syntax of settings variables expansion. An old variable value like '%d' now becomes '%{user | domain}'. The list of chances is long and there is no automated way of migrating your config. Imagine my face when I realised the sheer magnitude of my problem.

Integrated Complexity: cogs in a machine

The thing with email services is that it is *hard*. Not because of a single component being difficult, but because to run a complete email stack you need many different parts working together seamlessly. Here are some of the pieces that comprise my company's mail service:

- **Postfix:** The primary SMTP mail transfer agent (MTA), it communicates with other mail servers to receive and send email. It needs to know what email addresses are valid for receiving mails and who is authorised to send new messages.
- **Amavis filter:** All mails processed by the server go through the amavis filtering mechanism that checks individual messages using SpamAssissin and ClamAV.
- **SpamAssasin:** Using a complex set of rules and bayesian analyses, messages are automatically scanned for the likeliness of being spam.

- **ClamAV:** Messages are scanned for viruses. Attachments are extracted and analysed, file by file.
- **Reverse DNS blacklists:** In addition to scanning the messages, the mail server also does a background check on external servers it communicates with. Mail headers are parsed and individual hops are checked for their reputation.
- **OpenDKIM:** Some messages contain a 'domain key' signature, proofing that their sender is authorised. Messages with headers are checked, while others are signed using a miltering service tied into Postfix.
- **Sieve:** Incoming messages are automatically sorted using per user rules, e.g. having certain messages delivered in specific folders. It needs to know what rules apply to what addresses.
- **MariaDB / MySQL:** The email addresses, valid logins, usernames, password hashes and aliases are stored in a centralised database. Specific SQL queries are configured to check things like ”is this a valid password?”
- **Nginx:** The email server also has a web server that enables the management of accounts and to provide access to email via webmail.
- **NodeJS:** The management console utilises a NodeJS backend to enable server administrators and clients to setup their addresses and passwords.
- **Roundcube**: The server uses a widely used webmail program to enable clients to access their email via the browser.
- **PHP:** The webmail requires PHP to run, including some dependencies to have it talk to Dovecot (using IMAP) and MariaDB (using SQL).
- **Dovecot:** Then there is Dovecot, the actual piece of software that enables end users to access their email. It enables authenticated users to access the emails that are stored on the infrastructure's disks using protocols like IMAP or POP3.



A complete mail server stack comprises many smaller components finely tuned together

These pieces are all finely tuned to each other, like cogs in a complex machine they fit precisely. Their configuration is tailored to their specific role, often interfacing with multiple other pieces. You can imagine that an unexpected change in one of these cogs (like Dovecot) can cause quite the disruption!

There was no quick fix available, no automated configuration upgrading. Given this update being relatively new, most large language models (AI) did not have an answer either. ChatGPT simply generated suggestions based on the old 2.3 documentation. To resolve the issue I needed to apply *'old school Linux server skills'*: manually creating a new tailored configuration for Dovecot 2.4.

Stop to Prevent Data Loss

Once I grasped the nature of the problem, I immediately stopped most of the email services (e.g. bringing the engine to a complete stop). This prevents data loss as emails are stopped being processed. This causes other mail servers to queue emails, sometimes issuing 4xx SMTP errors (like 421, 451, 454 or 455). The design of the world wide email protocol accommodates temporary outages and other servers will simply try again later.

Rewriting the config by RTFM

The error message in the logs pointed me to the official Dovecot docs, and there it was: a yellowishly highlighted WARNING describing the configuration changes. It's a lengthy document: as PDF it counts 18 pages. On a normal day it would take some effort to get through this - imagine having to do this while the proverbial house is on fire!

Dovecot's configuration consists of various files that all setup parts of the software's behaviour. Like how authentication is done, or how it integrates with a Sieve filter, or where it gets the username and password hashes for login. The changes introduced in Dovecot 2.4 nearly affect all the parameters as the creators have introduced a new syntax for server variables.

Variable notation syntax

Previously the configuration used variable notation like '%u' as a placeholder for a username. The new syntax allows variables to be piped to a modifier, enabling normalisation and things like sub string selection. For example: to get an email domain from a username you no longer use '%d' but use the 'user'-variable as base to pipe it to the domain modifier by using this syntax: '%{user | domain}'.

Once I understood this new language, I really appreciated the work of art that it entails. It is an improvement over the older abbreviated variable names as you no longer need a dictionary to look things up: instead of something vague like '%r' you now read '%{remove_ip}'. It is inherently more readable, something I really value.

Encryption: 'ssl_cert' becomes 'ssl_server_cert_file'

Other changes include the way the config points to external files used for encryption. Previously you would have a setting 'ssl_cert' that could contain the actual SSL/TLS certificate as string. In version 2.4 you'll have the variable 'ssl_server_cert_file' taking a path pointing to the certificate on the filesystem instead. **It is not just renaming, the nature of the value is new, too!** To fix the configuration, you'll have to walk each of these updated parameters and determine their impact on your particular setup. It is time consuming and error prone.

Splitting the 'mail_location' setting

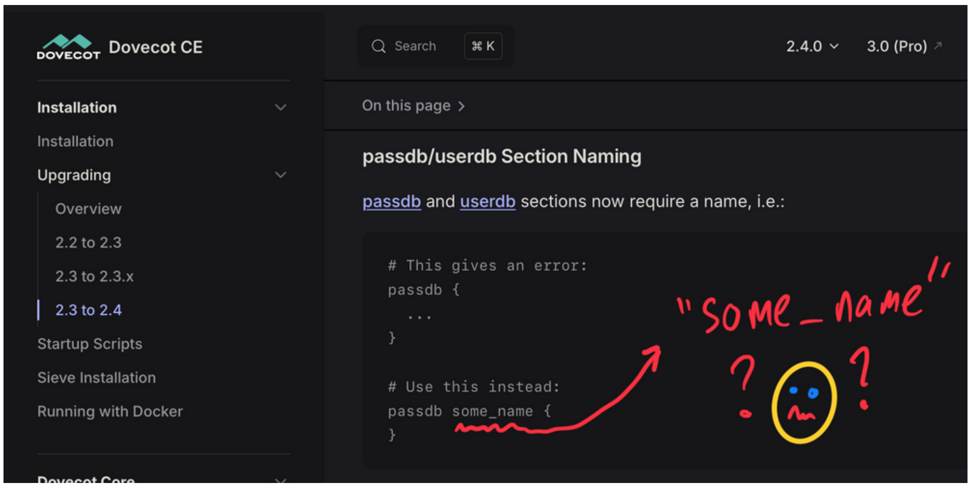
New in 2.4 is how the 'mail_location' is split up into multiple smaller variables. Previously a single variable would define where email was saved on disk. You'll now need to split this in two variables 'mail_driver' and 'mail_path', **AND** use the new server variable syntax. If you get any of these wrong, it will cause all email to disappear (trust me!). Depending on your specific storage configuration, you'll want to double check the username part (e.g. do you include the entire email address or just the prefix sans domain name?).

```
# Dovecot 2.3 config:
# mail_location = maildir:/var/vmail/%d/%u

# Corresponds to 2.4 syntax:
mail_driver = maildir
mail_path = /var/vmail/%{user | domain }/%{user | username}
```

Migrating mail_location into mail_driver and mail_path using the new server variable syntax

Dovecot with SQL backend: passdb and userdb

If your Dovecot installation works with an external database (or store) to authenticate usernames and passwords, you'll have to rewrite the 'auth-sql.conf.ext' in '/etc/dovecot/conf.d/' which is called/included by the 10-auth.conf file. In Dovecot 2.4 there is a new way of segmenting passdb and userdb settings. In the Dovecot 2.3 setup I used an external file to provide the database queries and connection string. During authentication the multiple queries are used to 1) authenticate users, and 2) load user-specific settings like the path of the email store.

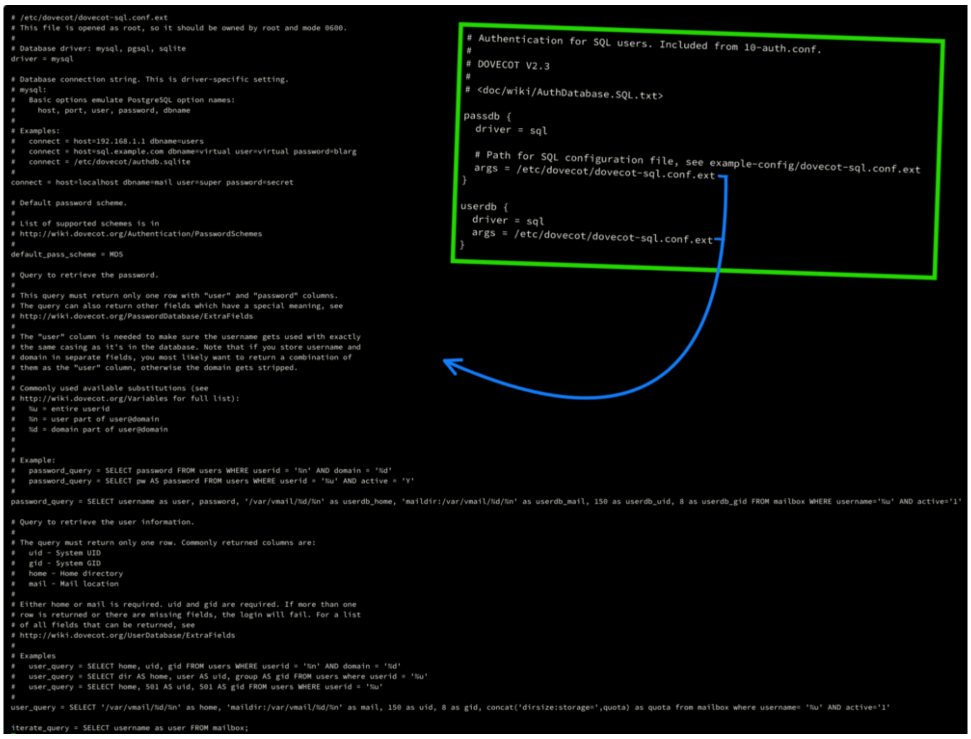


passdb/userdb Section Naming

passdb and userdb sections now require a name, i.e.:

```
# This gives an error:
passdb {
  ...
}

# Use this instead:
passdb some_name {
}
```

The userdb and passdb upgrade instructions are very helpful at first glance /s

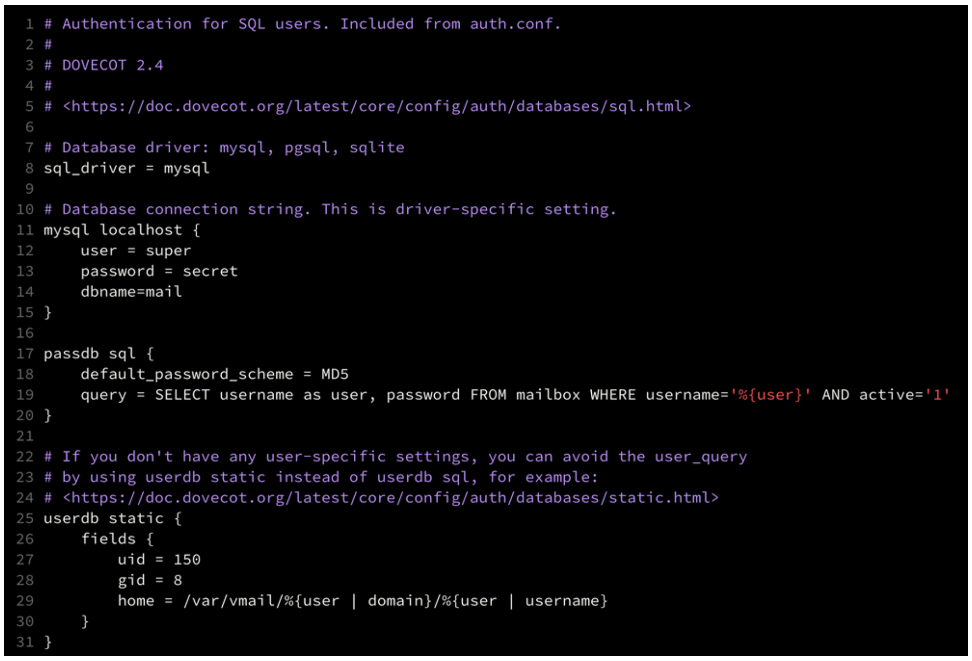


```
# /etc/dovecot/dovecot-sql.conf.ext
# This file is opened as root, so it should be owned by root and mode 0600.
#
# Database driver: mysql, pgsql, sqlite
driver = mysql

# Database connection string. This is driver-specific setting.
# mysql:
#   Basic options emulate PostgreSQL option names:
#     host, port, user, password, dbname
#
# Examples:
#   connect = host=192.168.1.1 dbname=users
#   connect = host=sql.example.com dbname=virtual user=virtual password=blarg
#   connect = /etc/dovecot/authdb.sqlite
#
connect = host=localhost dbname=mail user=super password=secret

# Default password scheme.
#
# List of supported schemes is in
# http://wiki.dovecot.org/Authentication/PasswordSchemes
#
default_pass_scheme = MD5

# Query to retrieve the password.
#
# This query must return only one row with "user" and "password" columns.
# The query can also return other fields which have a special meaning, see
# http://wiki.dovecot.org/PasswordDatabase/ExtraFields
#
# The "user" column is needed to make sure the username gets used with exactly
# the same casing as it's in the database. Note that if you store username and
# domain in separate fields, you most likely want to return a combination of
# them as the "user" column, otherwise the domain gets stripped.
#
# Commonly used available substitutions (see
# http://wiki.dovecot.org/Variables for full list):
#   %u = entire userid
#   %n = user part of user@domain
#   %d = domain part of user@domain
#
#
# Example:
#   password_query = SELECT password FROM users WHERE userid = '%n' AND domain = '%d'
#   password_query = SELECT pw AS password FROM users WHERE userid = '%u' AND active = 'Y'
#
password_query = SELECT username as user, password, '/var/vmail/%d/%n' as userdb_home, 'maildir:/var/vmail/%d/%n' as userdb_mail, 150 as userdb_uid, 8 as userdb_gid FROM mailbox WHERE username='%u' AND active='1'

# Query to retrieve the user information.
#
# The query must return only one row. Commonly returned columns are:
#   uid - System UID
#   gid - System GID
#   home - Home directory
#   mail - Mail location
#
# Either home or mail is required. uid and gid are required. If more than one
# row is returned or there are missing fields, the login will fail. For a list
# of all fields that can be returned, see
# http://wiki.dovecot.org/UserDatabase/ExtraFields
#
# Examples
#   user_query = SELECT home, uid, gid FROM users WHERE userid = '%n' AND domain = '%d'
#   user_query = SELECT dir AS home, user AS uid, group AS gid FROM users where userid = '%u'
#   user_query = SELECT home, 501 AS uid, 501 AS gid FROM users WHERE userid = '%u'
#
user_query = SELECT '/var/vmail/%d/%n' as home, 'maildir:/var/vmail/%d/%n' as mail, 150 as uid, 8 as gid, concat('dirsize:storage=',quota) as quota from mailbox where username= '%u' AND active='1'

iterate_query = SELECT username as user FROM mailbox;
```

```
# Authentication for SQL users. Included from 10-auth.conf.
#
# DOVECOT V2.3
#
# <doc/wiki/AuthDatabase.SQL.txt>

passdb {
  driver = sql

  # Path for SQL configuration file, see example-config/dovecot-sql.conf.ext
  args = /etc/dovecot/dovecot-sql.conf.ext
}

userdb {
  driver = sql
  args = /etc/dovecot/dovecot-sql.conf.ext
}
```

Dovecot 2.3 SQL backend setup (auth-sql.conf.ext and dovecot-sql.conf.ext)

In the new 2.4 version you have to restructure the parameters to be enclosed in sections that include a backend type indicator right behind the 'userdb' and 'passdb' keywords in the config. So, if you use a MySQL database as backend, you'll need 'userdb' becomes 'userdb sql'. You can include the connection string parameters right in the 'auth-sql.conf.ext' file so all relevant parameters are in one configuration file. To reduce complexity I rewrote the userdb section to use a the 'static' database driver: generating configuration variables like the user's mail path based on the user's username. See the dedicated documents.

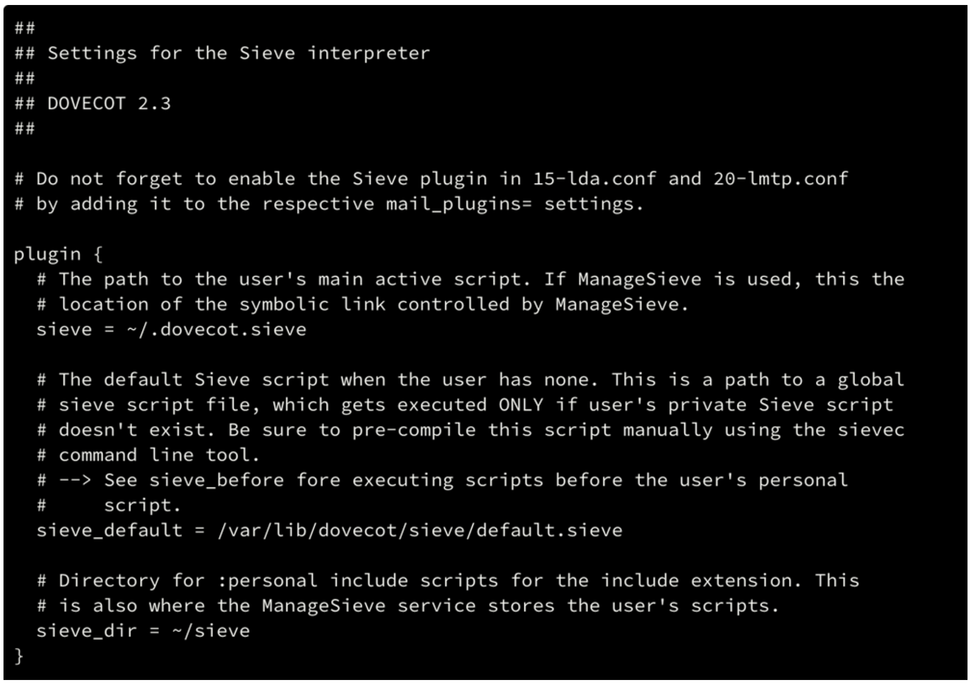


```
# Authentication for SQL users. Included from auth.conf.
#
# DOVECOT 2.4
#
# <https://doc.dovecot.org/latest/core/config/auth/databases/sql.html>

# Database driver: mysql, pgsql, sqlite
sql_driver = mysql

# Database connection string. This is driver-specific setting.
mysql localhost {
    user = super
    password = secret
    dbname=mail
}

passdb sql {
    default_password_scheme = MD5
    query = SELECT username as user, password FROM mailbox WHERE username='%{user}' AND active='1'
}

# If you don't have any user-specific settings, you can avoid the user_query
# by using userdb static instead of userdb sql, for example:
# <https://doc.dovecot.org/latest/core/config/auth/databases/static.html>
userdb static {
    fields {
        uid = 150
        gid = 8
        home = /var/vmail/%{user | domain}/%{user | username}
    }
}
```

Dovecot 2.4 SQL backend setup (auth-sql.conf.ext) - now much more compact than its predecessor. There is now only one SQL query as the userdb-ection now uses the static backend

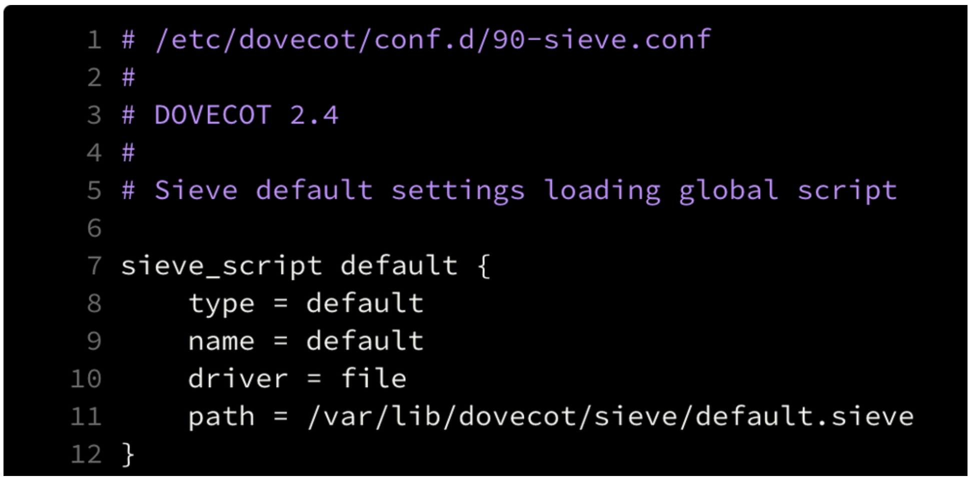
Sieve Plugin

If your setup uses Sieve to sort incoming emails, you'll need to adjust your setup with regard to the script's location, its defaults and behaviour. The new Dovecot 2.4 version introduces a concept called ”Script Storage” which enables dynamically loaded scripts to be executed for different users at various events. Our setup involves a simple default sieve script that sorts email flagged as spam automatically to a folder called Spam. The new syntax enables a more compact config:



```
##
## Settings for the Sieve interpreter
##
## DOVECOT 2.3
##

# Do not forget to enable the Sieve plugin in 15-lda.conf and 20-lmtp.conf
# by adding it to the respective mail_plugins= settings.

plugin {
  # The path to the user's main active script. If ManageSieve is used, this the
  # location of the symbolic link controlled by ManageSieve.
  sieve = ~/.dovecot.sieve

  # The default Sieve script when the user has none. This is a path to a global
  # sieve script file, which gets executed ONLY if user's private Sieve script
  # doesn't exist. Be sure to pre-compile this script manually using the sievec
  # command line tool.
  # --> See sieve_before fore executing scripts before the user's personal
  #     script.
  sieve_default = /var/lib/dovecot/sieve/default.sieve

  # Directory for :personal include scripts for the include extension. This
  # is also where the ManageSieve service stores the user's scripts.
  sieve_dir = ~/sieve
}
```

Sieve's config in Dovecot 2.3 (90-sieve.conf) with now obsolete variables sieve_default and sieve_dir

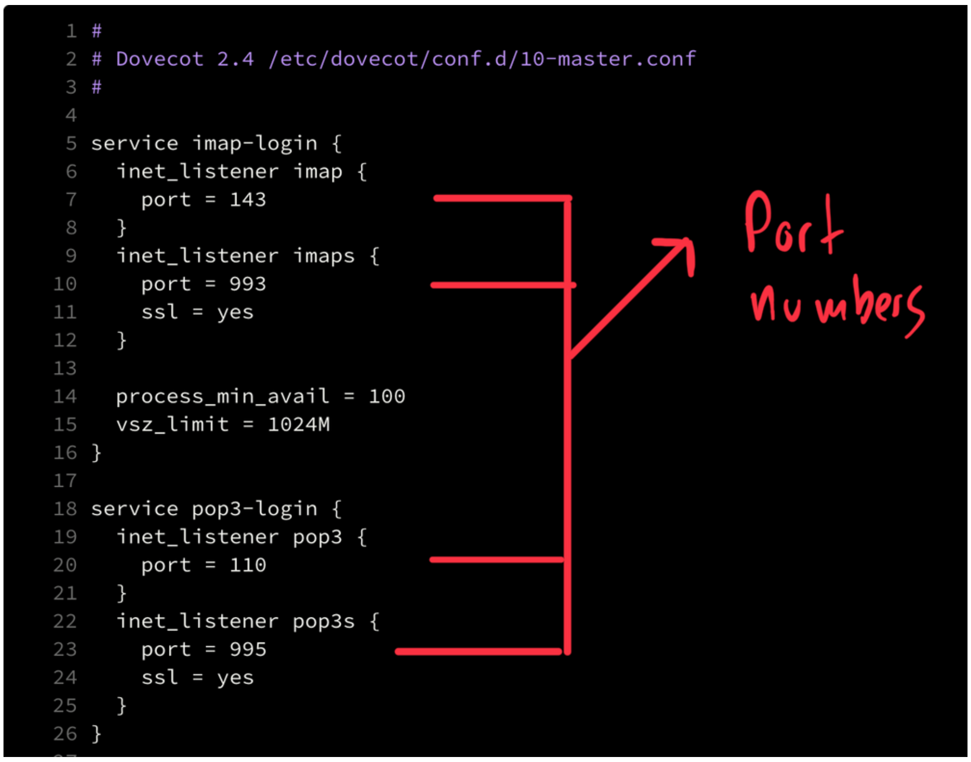


```
# /etc/dovecot/conf.d/90-sieve.conf
#
# DOVECOT 2.4
#
# Sieve default settings loading global script

sieve_script default {
    type = default
    name = default
    driver = file
    path = /var/lib/dovecot/sieve/default.sieve
}
```

Sieve's config in Dovecot 2.4 (90-sieve.conf)

Testing the New Config

You might be tempted to simply start the server again after redefining the 2.4 configuration. But, you should be very careful as a wrong configuration could cause clients to loose access to all emails and/or force them to re-download their entire mailboxes. Instead I opted for a gradual approach where I changed the IMAP (and POP3) port numbers to something arbitrary. This way I could test the new setup all by myself while keeping the actual customers from connecting to the server. This enabled me to fix some type errors, refine some of the new server variable syntax and deal with some odd log messages. You can control the server's port numbers from the '10-master.conf' file.



```
#
# Dovecot 2.4 /etc/dovecot/conf.d/10-master.conf
#

service imap-login {
  inet_listener imap {
    port = 143
  }
  inet_listener imaps {
    port = 993
    ssl = yes
  }

  process_min_avail = 100
  vsz_limit = 1024M
}

service pop3-login {
  inet_listener pop3 {
    port = 110
  }
  inet_listener pop3s {
    port = 995
    ssl = yes
  }
}
```

Setup the server's IMAP and POP numbers to enable private/gradual debugging of the new configuration

Conclusion

Outages like this are rare, thanks to the stability of Debian GNU/Linux, but not impossible. Daily backups and a tested disaster recovery plan are in place for my company's Lemmid Online email service. While a full failover wasn't required this time, prior drills proved crucial in swiftly adapting the configuration.

This incident was a reminder that even with automation and AI, deep system knowledge, preparation, and manual debugging remain essential tools when critical infrastructure components change unexpectedly and without automated migration paths.

Dovecot 2.4 Sample Config

If you find this post by looking for a solution for your own broken mail server, you might find these Dovecot 2.4 sample config files useful. I found them during my own repair work and they might save you some time figuring out the new syntax: https://source.willem.com/dovecot-2.4-sample-config/